SN8PC22

用户参考手册

Version 1.0

SONiX 8 位单片机

修改记录

版本	时间	修改说明
VER 1.0	2011.12	初版。

目录

1 产品简介

1.1 功能特性

- **存储器配置**
  OTP ROM：16K * 16 位。
  RAM：240 * 8 位。

- **8 层堆栈缓存器**

- **I/O 引脚配置**
  双向输入输出打开：P0，P1，P5。
  400mA IR 输出引脚：P5.4/IROUT。
  具有唤醒功能的引脚：P0，P1 电平变换触发。
  具有上拉电阻的打开：P0，P1。
  外部中断触发沿：P0.0（由 PEDGE 控制）。

- **1 级 LVD**

- **指令周期（Fcpu）**
  Fcpu = Fosc/2，Fpsc/4，Fosc/8，Fosc/16。

- **功能强大的指令集**
  单周期指令系统（1T）。
  绝大部分指令只需要一个周期。
  JMP 指令可寻址整个 ROM 区。
  CALL 指令可寻址整个 ROM 区。
  查表指令 MOVC 可寻址整个 ROM 区。

- **3 个中断源**
  2 个内部中断：T0，TC0。
  1 个外部中断：INT0。

- **1 个 8 位基本定时器 T0**
- **1 个 8 位定时/计数器 TC0**
- **1 跳动可编程控制占空比/周期的 IR 发射**
- **内置 IR 接收器**
- **内置看门狗定时器，时钟源为内部低速 RC 时钟（10KHz @3V）**

- **4 种系统时钟**
  外部高速时钟：RC，高达 8MHz。
  外部高速时钟：晶体，高达 8MHz。
  内部高速时钟：RC，8MHz。
  内部低速时钟：RC，10KHz（3V）。

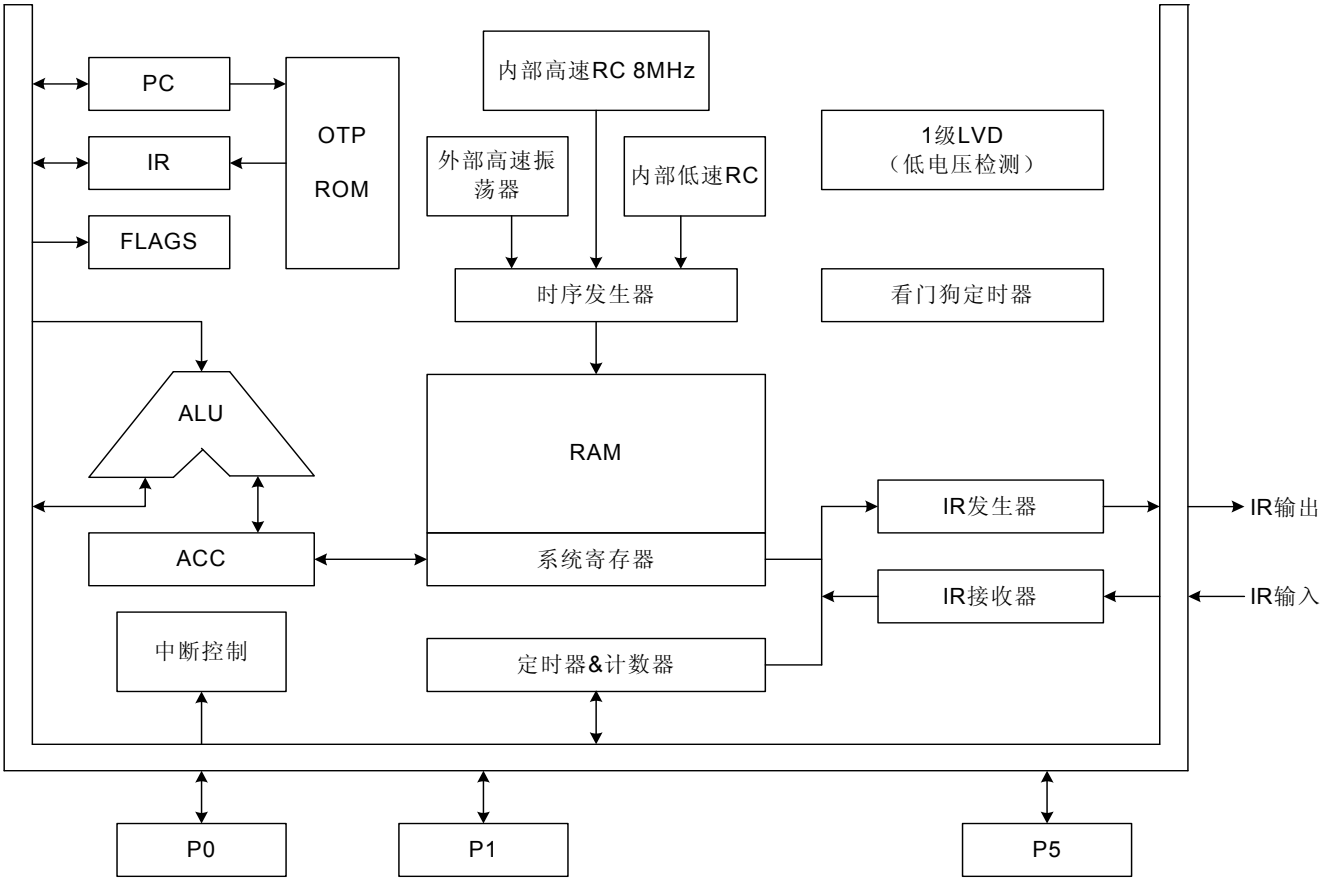
- **4 种工作模式**
  普通模式：高低速时钟都正常工作。
  低速模式：仅低速时钟正常工作。
  睡眠模式：高低速时钟都停止工作。
  绿色模式：由定时器周期性的唤醒。

- **封装形式**
  PDIP 20 pin
  SOP 20 pin
  SSOP 20 pin

☞ **产品特性比较表**

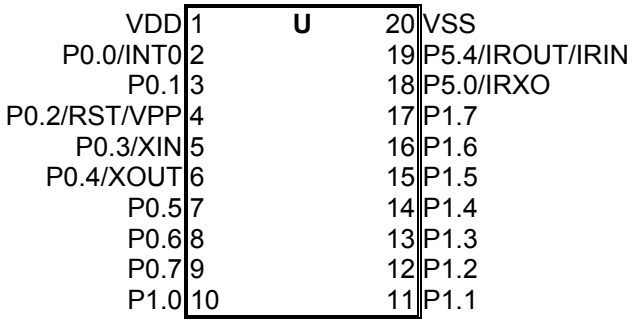
单片机型号	ROM	RAM	堆栈	I/O	振荡器			IR 接收	IR 输出	唤醒功能引脚数目	封装形式
					Ext. 455K	Ext. 4M	Int. 8M				
SN8PC20	2K*16	56	4	18	V	V	V	-	占空比/周期可编程控制，400mA 的灌电流	16	DIP20/SOP20/SSOP20
SN8PC21	1K*16	32	4	13	-	V	V	-	占空比/周期可编程控制，400mA 的灌电流	13	DIP16/SOP16
SN8PC22	16K*16	240	8	18	-	V	V	V	占空比/周期可编程控制，400mA 的灌电流	16	DIP20/SOP20/SSOP20

1.2 系统框图


PC
IR
FLAGS
OTP
ROM
内部高速RC 8MHz
外部高速振荡器
内部低速RC
1级LVD
（低电压检测）
时序发生器
看门狗定时器
ALU
RAM
ACC
系统寄存器
IR发生器
IR接收器
IR输出
IR输入
中断控制
定时器&计数器
P0
P1
P5

1.3 引脚配置

SN8PC22P (PDIP 20 pins)
SN8PC22S (SOP 20 pins)
SN8PC22X (SSOP 20 pins)

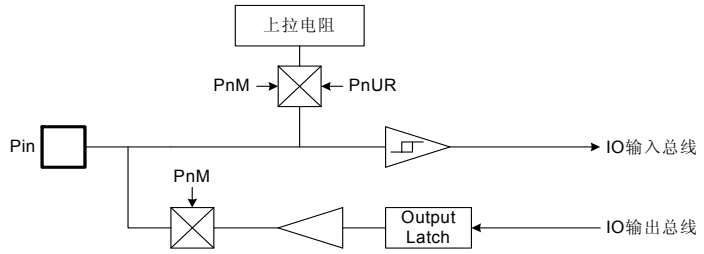


1.4 引脚说明

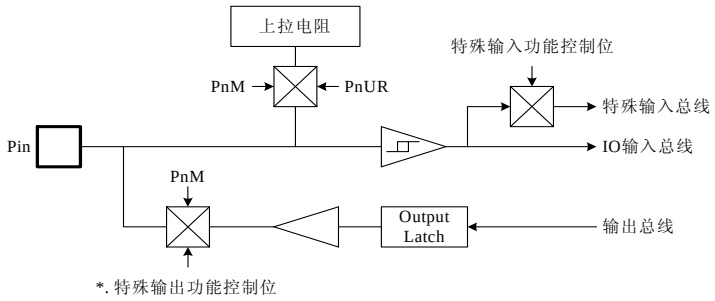
引脚名称	类型	功能说明
VDD, VSS	P	电源输入端。
P0.2/RST/VPP	I, P	RST：系统外部复位输入引脚，施密特触发，低电平有效，通常保持高电平。
		VPP：OTP 烧录电源输入引脚。
		P0.2：单向输入引脚，施密特触发，无上拉电阻。
XIN/P0.3	I/O	XIN：外部振荡器输入引脚。
		P0.3：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，电平变换时唤醒系统。
XOUT/P0.4	I/O	XOUT：外部振荡器输出引脚。
		P0.4：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，电平变换时唤醒系统。
P0.0/INT0	I/O	P0.0：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，电平变换时唤醒系统。
		INT0：外部中断输入引脚。
P0.1, P0[7:5]	I/O	P0：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，电平变换时唤醒系统。
P1[7:0]	I/O	P1：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，电平变换时唤醒系统。
P5.0/IRXO	I/O	P5.0：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻。
		IRXO：IR 接收器输出引脚。
P5.4/IROUT/IRIN	I/O	P5.4：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻。
		IROUT：占空比/周期可编程控制的 IR 信号输出引脚。
		IRIN：IR 信号输入引脚。

1.5 引脚电路结构图

- 双向 I/O 引脚

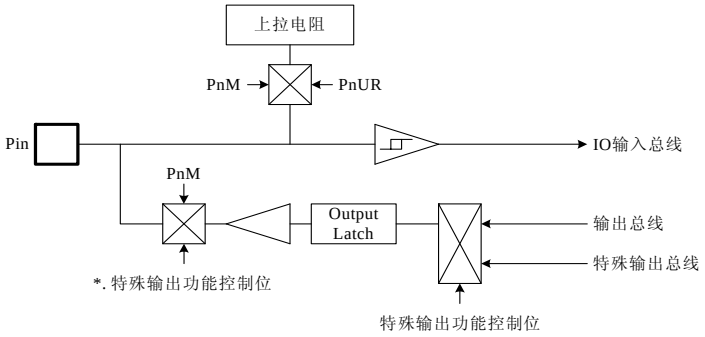


- 双向 I/O 引脚，与特殊数字输入功能引脚共用，如 INT0，事件计数器等



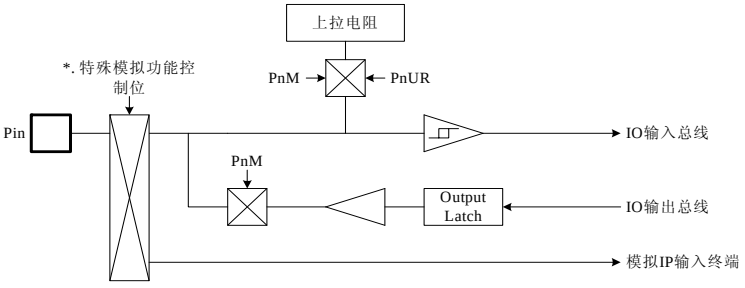
***.部分特殊功能直接切换I/O方向，无需通过PnM寄存器**

- 双向 I/O 引脚，与特殊数字输出功能引脚共用，如 IR 输出



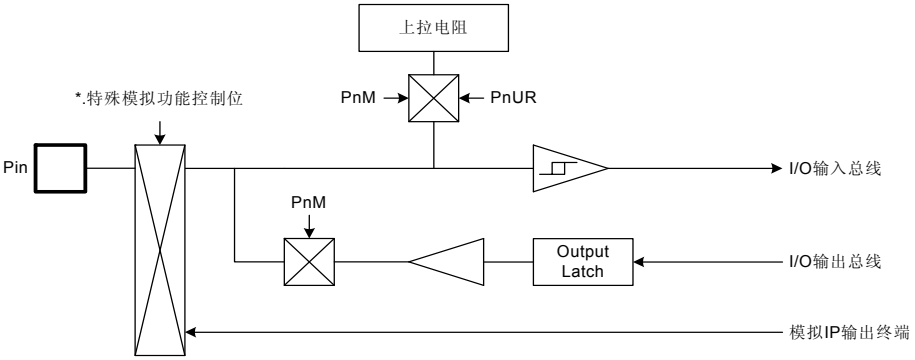
***.部分特殊功能直接切换I/O方向，无需通过PnM寄存器**

- 双向 I/O 引脚，与特殊模拟输入功能引脚共用，如 XIN



***.部分特殊功能直接切换I/O方向，无需通过PnM寄存器**

- 双向 I/O 引脚，与特殊模拟输入功能引脚共用，如 XOUT

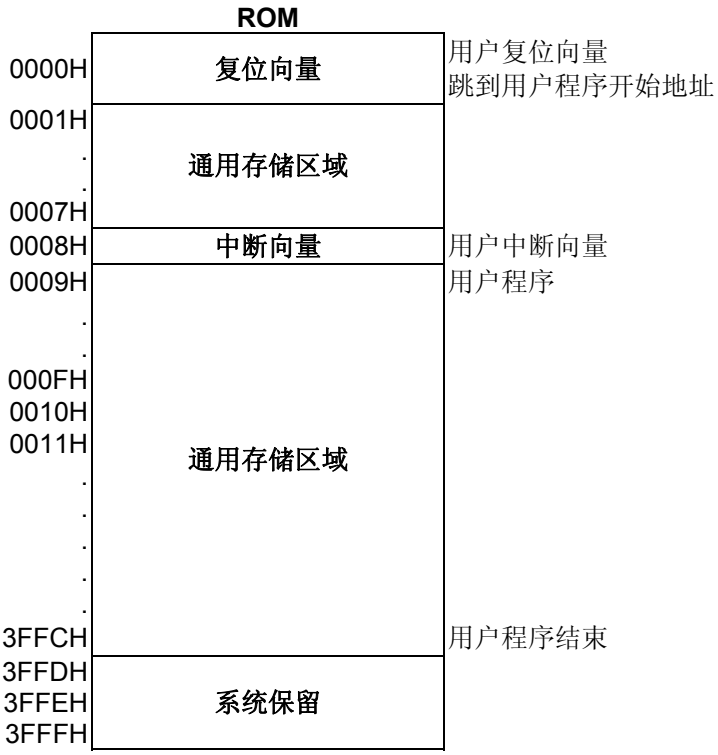


***.部分特殊功能直接切换I/O方向，无需通过PnM寄存器**

2 中央处理器（CPU）

2.1 程序存储器ROM

☞ **ROM：16K**



ROM 包括复位向量，中断向量，通用存储区域和系统保留区域：复位向量是程序的开始地址；中断向量是中断服务程序的开始地址；通用存储区域则是程序存储区，包括主循环，子程序和数据表。

2.1.1 复位向量（0000H）

具有一个字长的系统复位向量（0000H）。

- **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）。**

发生上述任一种复位后，程序将从 0000H 处重新开始执行，系统寄存器也都将恢复为默认值。根据 PFLAG 寄存器中的 NT0 和 NPD 标志位的内容可以判断系统复位方式。下面一段程序演示了如何定义 ROM 中的复位向量。

➢ **例：定义复位向量。**

```
        ORG     0               ;
        JMP     START           ; 跳至用户程序。
        …

        ORG     10H
START:                          ; 用户程序起始地址。
        …                       ; 用户程序。
        …
        ENDP                    ; 程序结束。
```

2.1.2 中断向量（0008H）

中断向量地址为 0008H。一旦有中断响应，程序计数器 PC 的当前值就会存入堆栈缓存器并跳转到 0008H 开始执行中断服务程序。用户必须定义中断向量，下面的示例程序说明了如何在程序存储器种定义中断向量。

✷ **注：通过“PUSH”、“POP”指令保存和恢复 ACC/PFLAG（不包括 NT0，NPD）寄存器，PUSH/POP 缓存器只有一层。.**

➢ **例：定义中断向量，中断服务程序紧随 ORG 8 之后。**

```
.CODE
                ORG     0
                JMP     START           ; 跳到用户程序。
                …
                ORG     8               ; 中断向量。
                PUSH                    ; 保存 ACC 和 PFLAG 寄存器。
                …
                …
                POP                     ; 恢复 ACC 和 PFLAG 寄存器。
                RETI                    ; 中断服务程序结束。
                …
START:                                  ; 用户程序开始。
                …                       ; 用户程序。
                …
                JMP     START           ; 用户程序结束。
                …

                ENDP                    ; 程序结束。
```

➢ **例：定义中断向量，中断服务程序在用户程序之后。**

```
.CODE
                ORG     0
                JMP     START           ; 跳到用户程序。
                …
                ORG     8               ; 中断向量。
                JMP     MY_IRQ          ; 跳到中断服务程序。

                ORG     10H
START:                                  ; 用户程序开始。
                …                       ; 用户程序。
                …
                …
                JMP     START           ; 用户程序结束。
                …
MY_IRQ:                                 ; 中断服务程序开始。
                PUSH                    ; 保存 ACC 和 PFLAG 寄存器。
                …
                …
                POP                     ; 恢复 ACC 和 PFLAG 寄存器。
                RETI                    ; 中断服务程序结束。
                …
                ENDP                    ; 程序结束。
```

✷ **注：从上面的程序中容易得出 SONiX 的编程规则，有以下几点：**
1、地址 0000H 的“JMP”指令使程序从头开始执行；
2、地址 0008H 是中断向量；
3、用户的程序应该是一个循环。

2.1.3 查表

在 SONiX 单片机中，对 ROM 区中的数据进行查找，寄存器 Y 指向所找数据地址的中间字节（bit8~bit15），寄存器 Z 指向所找数据地址的低字节（bit0~bit7）。执行完 MOVC 指令后，所查找数据低字节内容被存入 ACC 中，而数据高字节内容被存入 R 寄存器。

➢ **例：查找 ROM 地址为“TABLE1”的值。**

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

                                        ; 查找下一地址。
                INCMS   Z
                JMP     @F              ; Z 没有溢出。
                INCMS   Y               ; Z 溢出（FFH → 00），→ Y=Y+1
                NOP                     ;
                                        ;
@@:             MOVC                    ; 查表，R = 51H，ACC = 05H。
                …                       ;
TABLE1:         DW      0035H           ; 定义数据表（16 位）数据。
                DW      5105H
                DW      2012H
                …
```

✷ **注：当寄存器 Z 溢出（从 0FFH 变为 00H）时，寄存器 Y 并不会自动加 1。因此， Z 溢出时，Y 必须由程序加 1，下面的宏 INC_YZ 能够对 Y 和 Z 寄存器自动处理。**

➢ **例：宏 INC_YZ。**

```
INC_YZ          MACRO
                INCMS   Z
                JMP     @F              ; 没有溢出。

                INCMS   Y
                NOP                     ; 没有溢出。
@@:
                ENDM
```

➢ **例：通过“INC_YZ”对上例进行优化。**

```
                B0MOV   Y, #TABLE1$M    ; 设置 TABLE1 地址中间字节。
                B0MOV   Z, #TABLE1$L    ; 设置 TABLE1 地址低字节。
                MOVC                    ; 查表，R = 00H，ACC = 35H。

                INC_YZ                  ; 查找下一地址数据。
                                        ;
@@:             MOVC                    ; 查表，R = 51H，ACC = 05H。
                …                       ;
TABLE1:         DW      0035H           ; 定义数据表（16 位）数据。
                DW      5105H
                DW      2012H
                …
```

下面的程序通过累加器对 Y，Z 寄存器进行处理来实现查表功能，但需要特别注意进位时的处理。

➢ **例：由指令 B0ADD/ADD 对 Y 和 Z 寄存器加 1。**

```
                B0MOV   Y, #TABLE1$M    ; 设置 TABLE1 地址中间字节。
                B0MOV   Z, #TABLE1$L    ; 设置 TABLE1 地址低字节。

                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。
                NOP
GETDATA:                                ;
                MOVC                    ; 存储数据，如果 BUF = 0，数据为 0035H。
                                        ; 如果 BUF = 1，数据=5105H。
                                        ; 如果 BUF = 2，数据=2012H。
                …
TABLE1:         DW      0035H           ; 定义数据表（16 位）数据。
                DW      5105H
                DW      2012H
                …
```

2.1.4 跳转表

跳转表能够实现多地址跳转功能。由于 PCL 和 ACC 的值相加即可得到新的 PCL，因此，可以通过对 PCL 加上不同的 ACC 值来实现多地址跳转。ACC 值若为 n，PCL+ACC 即表示当前地址加 n，执行完当前指令后 PCL 值还会自加 1，可参考以下范例。如果 PCL＋ACC 后发生溢出，PCH 则自动加 1。由此得到的新的 PC 值再指向跳转指令列表中新的地址。这样，用户就可以通过修改 ACC 的值轻松实现多地址的跳转。

✷ 注：PCH 只支持 PC 增量运算，而不支持 PC 减量运算。当 PCL+ACC 后如有进位，PCH 的值会自动加 1。PCL-ACC 后若有借位，PCH 的值将保持不变，用户在设计应用时要加以注意。

➢ **例：跳转表。**

```
        ORG     0100H           ; 跳转表从 ROM 前端开始。

        B0ADD   PCL, A          ; PCL = PCL + ACC，PCL 溢出时 PCH 加 1。
        JMP     A0POINT         ; ACC = 0，跳至 A0POINT。
        JMP     A1POINT         ; ACC = 1，跳至 A1POINT。
        JMP     A2POINT         ; ACC = 2，跳至 A2POINT。
        JMP     A3POINT         ; ACC = 3，跳至 A3POINT。
```

SONiX 单片机提供一个宏以保证可靠执行跳转表功能，它会自动检测 ROM 边界并将跳转表移至适当的位置。但采用该宏程序会占用部分 ROM 空间。

➢ **例：如果跳转表跨越 ROM 边界，将引起程序错误。**

```
@JMP_A  MACRO   VAL
        IF      (($+1) !& 0XFF00) !!= (($+(VAL)) !& 0XFF00)
        JMP     ($ | 0XFF)
        ORG     ($ | 0XFF)
        ENDIF
        B0ADD   PCL, A
        ENDM
```

✷ 注：“VAL”为跳转表列表中列表个数。

➢ **例：宏“MACRO3.H”中，“@JMP_A”的应用。**

```
        B0MOV   A, BUF0         ; “BUF0”从 0 至 4。
        @JMP_A  5               ; 列表个数为 5。
        JMP     A0POINT         ; ACC = 0，跳至 A0POINT。
        JMP     A1POINT         ; ACC = 1，跳至 A1POINT。
        JMP     A2POINT         ; ACC = 2，跳至 A2POINT。
        JMP     A3POINT         ; ACC = 3，跳至 A3POINT。
        JMP     A4POINT         ; ACC = 4，跳至 A4POINT。
```

如果跳转表恰好位于 ROM BANK 边界处（00FFH~0100H），宏指令“@JMP_A”将调整跳转表到适当的位置（0100H）。

➢ **例：“@JMP_A”运用举例**

```
; 编译前
ROM 地址
                B0MOV       A, BUF0         ; “BUF0”从 0 到 4。
                @JMP_A      5               ; 列表个数为 5。
00FDH           JMP         A0POINT         ; ACC = 0，跳至 A0POINT。
00FEH           JMP         A1POINT         ; ACC = 1，跳至 A1POINT。
00FFH           JMP         A2POINT         ; ACC = 2，跳至 A2POINT。
0100H           JMP         A3POINT         ; ACC = 3，跳至 A3POINT。
0101H           JMP         A4POINT         ; ACC = 4，跳至 A4POINT。

; 编译后
ROM 地址
                B0MOV       A, BUF0         ; “BUF0”从 0 到 4。
                @JMP_A      5               ; 列表个数为 5。
0100H           JMP         A0POINT         ; ACC = 0，跳至 A0POINT。
0101H           JMP         A1POINT         ; ACC = 1，跳至 A1POINT。
0102H           JMP         A2POINT         ; ACC = 2，跳至 A2POINT。
0103H           JMP         A3POINT         ; ACC = 3，跳至 A3POINT。
0104H           JMP         A4POINT         ; ACC = 4，跳至 A4POINT。
```

2.1.5 CHECKSUM计算

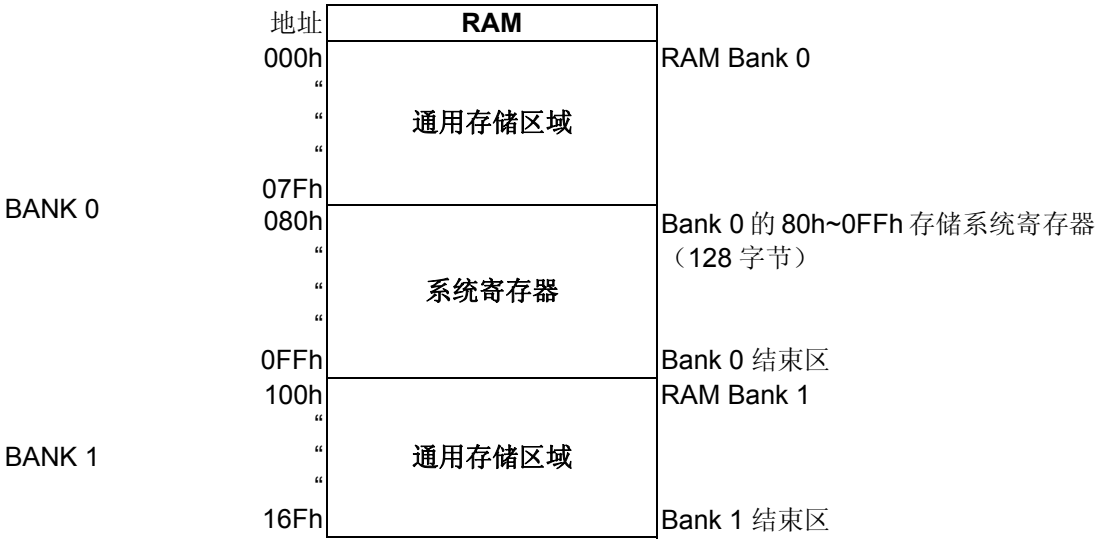
ROM 的末端位置的几个字限制使用，进行 Checksum 计算时，用户应避免对该单元的访问。

➢ **例：示例程序演示了如何对 00H 到用户程序结束进行 Checksum 计算。**

```
                MOV     A,#END_USER_CODE$L
                B0MOV   END_ADDR1, A        ; 用户程序终端地址低位字节存入 end_addr1。
                MOV     A,#END_USER_CODE$M
                B0MOV   END_ADDR2, A        ; 用户程序终端地址中间字节存入 end_addr2。
                CLR     Y                   ; 清 Y。
                CLR     Z                   ; 清 Z。
@@:
                MOVC
                B0BCLR  FC                  ; 清标志位 C。
                ADD     DATA1, A            ;
                MOV     A, R
                ADC     DATA2, A            ;
                JMP     END_CHECK           ; 检查 YZ 地址是否为代码的结束地址。
AAA:
                INCMS   Z                   ;
                JMP     @B                  ; 如果 Z != 00H，进行下一个计算。
                JMP     Y_ADD_1             ; 如果 Z = 00H，Y 加 1。
END_CHECK:
                MOV     A, END_ADDR1
                CMPRS   A, Z                ; 检查 Z 地址是否为用户程序结束地址低位地址。
                JMP     AAA                 ; 否，则进行 checksum 计算。
                MOV     A, END_ADDR2
                CMPRS   A, Y                ; 是则检查 Y 的地址是否为用户程序结束地址中间地址。
                JMP     AAA                 ; 否，则进行 Checksum 计算。
                JMP     CHECKSUM_END        ; 是则 Checksum 计算结束。

Y_ADD_1:
                INCMS   Y
                NOP
                JMP     @B                  ; 转至 checksum 计算。
CHECKSUM_END:
                …
                …
END_USER_CODE:
```

2.2 数据存储器（RAM）

☞ **RAM：240 X 8-bit**



240 字节的 RAM 分为 Bank0 和 Bank1，通过设置寄存器 RBANK 在 2 个 RAM 中转换。当 RBANK=0 时，程序直接在 BANK0 中运行；RBANK=1 时，直接在 BANK1 中运行。当在其中一个 BANK 中运行，而需要访问另一个 BANK 时，必须设置 RBANK。SONIX 提供 BANK0 型的指令（如 B0MOV、B0ADD、B0BTS1、B0BSET 等），可以在非 0 RAM BANK 区域直接控制 BANK0 的工作状态。

➢ **例：在 Bank1 中访问 Bank0，将 Bank0 RAM（WK00）的值一点到 Bank1 RAM（WK01）中。**

```
; Bank 1 (RBANK = 1)
        B0MOV   A, WK00     ; 使用 Bank0 型指令访问 Bank0 RAM。
        MOV     WK01,A
```

✷ **注：对于多个 BANK RAM 的操作，用户必须注意 RBANK 的设置以选择 RAM BANK，尤其在中断时的 RAM BANK 操作。RAM BANK 切换到 BANK0 时，系统不会自动保存 RBANK，必须通过程序保存，最好使用 BANK0 型指令来进行该操作。**

2.2.1 系统寄存器

2.2.1.1 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	RBANK	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	P0M	-	-	-	-	-	-	PEDGE
C	-	P1M	-	-	-	P5M	-	-	INTRQ	INTEN	OSCM	-	WDTR	TC0R	PCL	PCH
D	P0	P1	-	-	-	P5	-	-	T0M	T0C	TC0M	TC0C	-	-	-	STKP
E	P0UR	P1UR	-	-	-	P5UR	@HL	@YZ	TC0D	-	-	-	-	-	-	-
F	STK7L	STK7H	STK6L	STK6H	STK5L	STK5H	STK4L	STK4H	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

2.2.1.2 系统寄存器说明

H, L = 专用寄存器，@HL 寻址寄存器
R = 工作寄存器和 ROM 查表数据缓存器
PnM = Pn 模式控制寄存器
INTRQ = 中断请求寄存器
OSCM = 振荡器模式寄存器
TC0R = TC0 自动重装数据缓存器
Pn = Pn 数据缓存器
T0C = T0 计数寄存器
TC0C = TC0 计数寄存器
PnUR = Pn 上拉电阻控制寄存器
@YZ = 间接寻址寄存器
STK0~STK3 = 堆栈缓存器

Y, Z = 专用寄存器，@YZ 间接寻址寄存器，ROM 寻址寄存器
PFLAG = 特殊标志寄存器
PEDGE = P0.0 方向控制寄存器
INTEN = 中断使能寄存器
WDTR = 看门狗定时器清零寄存器
PCH, PCL = 程序计数器
T0M = T0 模式寄存器
TC0M = TC0 模式寄存器
STKP = 堆栈指针
@HL = 间接寻址寄存器
TC0D = TC0 占空比控制寄存器

2.2.1.3 系统寄存器的位定义

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
087H								RBANKS0	R/W	RBANK
0B8H	P07M	P06M	P05M	P04M	P03M		P01M	P00M	R/W	P0M
0BFH				P00G1	P00G0		IRXO	PEDGS	R/W	PEDGE
0C1H	P17M	P16M	P15M	P14M	P13M	P12M	P11M	P10M	R/W	P1M
0C5H				P54M				P50M	R/W	P5M
0C8H			TC0IRQ	T0IRQ				P00IRQ	R/W	INTRQ
0C9H			TC0IEN	T0IEN				P00IEN	R/W	INTEN
0CAH				CPUM1	CPUM0	CLKMD	STPHX		R/W	OSCM
0CCH	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0	W	WDTR
0CDH	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0	W	TC0R
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH			PC13	PC12	PC11	PC10	PC9	PC8	R/W	PCH
0D0H	P07	P06	P05	P04	P03	P02	P01	P00	R/W	P0
0D1H	P17	P16	P15	P14	P13	P12	P11	P10	R/W	P1
0D5H				P54				P50	R/W	P5
0D8H	T0ENB	T0rate2	T0rate1	T0rate0	T0TB	CPTS1	CPTS0	IRXEN	R/W	T0M
0D9H	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0	R/W	T0C
0DAH	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS	IRSTS	IREN	CREN	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DFH	GIE					STKPB2	STKPB1	STKPB0	R/W	STKP
0E0H	P07R	P06R	P05R	P04R	P03R		P01R	P00R	W	P0UR
0E1H	P17R	P16R	P15R	P14R	P13R	P12R	P11R	P10R	W	P1UR
0E5H				P54R				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
0E8H	TC0D7	TC0D6	TC0D5	TC0D4	TC0D3	TC0D2	TC0D1	TC0D0	R/W	TC0D
0F0H	S7PC7	S7PC6	S7PC5	S7PC4	S7PC3	S7PC2	S7PC1	S7PC0	R/W	STK7L
0F1H	1	1	S7PC13	S7PC12	S7PC11	S7PC10	S7PC9	S7PC8	R/W	STK7H
0F2H	S6PC7	S6PC6	S6PC5	S6PC4	S6PC3	S6PC2	S6PC1	S6PC0	R/W	STK6L
0F3H	1	1	S6PC13	S6PC12	S6PC11	S6PC10	S6PC9	S6PC8	R/W	STK6H
0F4H	S5PC7	S5PC6	S5PC5	S5PC4	S5PC3	S5PC2	S5PC1	S5PC0	R/W	STK5L
0F5H	1	1	S5PC13	S5PC12	S5PC11	S5PC10	S5PC9	S5PC8	R/W	STK5H
0F6H	S4PC7	S4PC6	S4PC5	S4PC4	S4PC3	S4PC2	S4PC1	S4PC0	R/W	STK4L
0F7H	1	1	S4PC13	S4PC12	S4PC11	S4PC10	S4PC9	S4PC8	R/W	STK4H
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H	1	1	S3PC13	S3PC12	S3PC11	S3PC10	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH	1	1	S2PC13	S2PC12	S2PC11	S2PC10	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH	1	1	S1PC13	S1PC12	S1PC11	S1PC10	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH	1	1	S0PC13	S0PC12	S0PC11	S0PC10	S0PC9	S0PC8	R/W	STK0H

✷ 注：

1、为了避免系统错误，在初始化时，请将上表所有寄存器的位都按照设计要求设置为确定的"1"或者"0"；

2、所有寄存器名都已在 SN8ASM 编译器中做了宣告；

3、用户使用 SN8ASM 编译器对寄存器的位进行操作时，须在该寄存器的位前加“F”；

4、指令“b0bset”，“b0bclr”，“bset”，“bclr”只能用于可读写的寄存器（“R/W”）。

2.2.2 累加器ACC

8 位数据寄存器 ACC 用来执行 ALU 与数据存储器之间数据的传送操作。如果操作结果为零（Z）或有进位产生（C 或 DC），程序状态寄存器 PFLAG 中相应位会发生变化。

ACC 并不在 RAM 中，因此在立即寻址模式中不能用“B0MOV”指令对其进行读写。

➢ **例：读/写 ACC。**

```
; 将立即数写入 ACC。
                MOV         A, #0FH

;把 ACC 中的数据存入 BUF 中。
                MOV         BUF, A
                B0MOV       BUF, A

; 把 BUF 中的数据送到 ACC 中。
                MOV         A, BUF
                B0MOV       A, BUF
```

系统执行中断时，不会自动保存 ACC 和 PFLAG，必须通过 PUSH、POP 指令来保存和恢复 ACC 和 PFLAG。

➢ **例：保存 ACC 和工作寄存器。**

```
INT_SERVICE:
                PUSH                    ; 保存 ACC 和 PFLAG。
                …
                …
                POP                     ; 恢复 ACC 和 PFLAG。

                RETI                    ; 退出中断。
```

2.2.3 程序状态寄存器PFLAG

寄存器 PFLAG 中包含 ALU 运算状态信息、系统复位状态信息和 LVD 检测信息，其中，位 NT0 和 NPD 显示系统复位状态信息，包括上电复位、LVD 复位、外部复位和看门狗复位；位 C、DC 和 Z 显示 ALU 的运算信息。

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

Bit [7:6] **NT0, NPD**：复位状态标志。

NT0	NPD	复位状态
0	0	看门狗复位
0	1	保留
1	0	LVD 复位
1	1	外部复位

Bit 2 **C**：进位标志。
1 = 加法运算后有进位、减法运算没有借位发生或移位后移出逻辑“1”或比较运算的结果 ≥ 0；
0 = 加法运算后没有进位、减法运算有借位发生或移位后移出逻辑“0”或比较运算的结果 < 0。

Bit 1 **DC**：辅助进位标志。
1 = 加法运算时低四位有进位，或减法运算后没有向高四位借位；
0 = 加法运算时低四位没有进位，或减法运算后有向高四位借位。

Bit 0 **Z**：零标志。
1 = 算术/逻辑/分支运算的结果为零；
0 = 算术/逻辑/分支运算的结果非零。

✷ **注：关于标志位 C、DC 和 Z 的更多信息请参阅指令集相关内容。**

2.2.4 程序计数器

程序计数器 PC 是一个 14 位二进制程序地址寄存器，分高 6 位和低 8 位。专门用来存放下一条需要执行指令的内存地址。通常，程序计数器会随程序中指令的执行自动增加。

若程序执行 CALL 和 JMP 指令时，PC 指向特定的地址。

	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PC	-	-	PC13	PC12	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	0	0
	PCH								PCL							

☞ **单地址跳转**

在 SONiX 单片机里面，有 9 条指令（CMPRS、INCS、INCMS、DECS、DECMS、BTS0、BTS1、B0BTS0 和 B0BTS1）可完成单地址跳转功能。如果这些指令执行结果为真，那么 PC 值加 2 以跳过下一条指令。

如果位测试为真，PC 加 2。

```
                B0BTS1      FC          ; 若 Carry_flag = 1 则跳过下一条指令。
                JMP         C0STEP      ; 否则执行 C0STEP。
                …
C0STEP:         NOP

                B0MOV       A, BUF0     ; BUF0 送入 ACC。
                B0BTS0      FZ          ; Zero flag = 0 则跳过下一条指令。
                JMP         C1STEP      ; 否则执行 C1STEP。
                …
                …
C1STEP:         NOP
```

如果 ACC 等于指定的立即数则 PC 值加 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      ; 否则跳至 C0STEP。
                …
                …
C0STEP:         NOP
```

执行加 1 指令后，结果为零时，PC 的值加 2，跳过下一条指令。

INCS：

```
                INCS        BUF0
                JMP         C0STEP
                …
C0STEP:         NOP
```

INCMS：

```
                INCMS       BUF0
                JMP         C0STEP
                …
C0STEP:         NOP
```

执行减 1 指令后，结果为零时，PC 的值加 2，跳过下一条指令。

DECS：

```
                DECS        BUF0
                JMP         C0STEP
                …
C0STEP:         NOP
```

DECMS：

```
                DECMS       BUF0
                JMP         C0STEP
                …
C0STEP:         NOP
```

☞ **多地址跳转**

执行 JMP 或 ADD M,A（M=PCL）指令可实现多地址跳转。执行 ADD M, A、ADC M, A 或 B0ADD M, A 后，若 PCL 溢出，PCH 会自动进位。对于跳转表及其它应用，用户可以通过上述 3 条指令计算 PC 的值而不需要担心 PCL 溢出的问题。

✷ 注：PCH 仅支持 PC 的递增运算而不支持递减运算。当 PCL+ACC 执行完 PCL 有进位时，PCH 会自动加 1；但执行 PCL-ACC 有借位发生，PCH 的值会保持不变。

➢ **例：PC = 0323H（PCH = 03H，PCL = 23H）。**

```
; PC = 0323H
                MOV         A, #28H
                B0MOV       PCL, A          ; 跳到地址 0328H。
                …

; PC = 0328H
                MOV         A, #00H
                B0MOV       PCL, A          ; 跳到地址 0300H。
                …
```

➢ **例：PC = 0323H（PCH = 03H，PCL = 23H）。**

```
; PC = 0323H
                B0ADD       PCL, A          ; PCL = PCL + ACC，PCH 的值不变。
                JMP         A0POINT         ; ACC = 0，跳到 A0POINT。
                JMP         A1POINT         ; ACC = 1，跳到 A1POINT。
                JMP         A2POINT         ; ACC = 2，跳到 A2POINT。
                JMP         A3POINT         ; ACC = 3，跳到 A3POINT。
                …
                …
```

2.2.5 H, L寄存器

寄存器 H 和 L 都是 8 位缓存器，主要有以下两个功能：

- **通用工作寄存器；**
- **RAM 数据寻址指针@HL。**

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	-	-	-	-	-	-	-	-

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	-	-	-	-	-	-	-	-

➢ **例：用 H、L 作为数据指针，访问 bank0 中 020H 处的内容。**

```
B0MOV   H, #00H
B0MOV   L, #20H
B0MOV   A, @HL
```

➢ **例：对 bank 0 中的数据进行清零处理。**

```
        CLR     H           ; H = 0，指向 bank 0。
        B0MOV   L, #7FH     ; L = 7FH。
CLR_HL_BUF:
        CLR     @HL         ; @HL 清零。
        DECMS   L           ; L – 1，如果 L = 0，程序结束。
        JMP     CLR_HL_BUF

        CLR     @HL
END_CLR:
        …
        …
```

2.2.6 Y, Z寄存器

寄存器 Y 和 Z 都是 8 位缓存器，主要用途如下：

- **普通工作寄存器；**
- **RAM 数据寻址指针@YZ；**
- **配合指令 MOVC 对 ROM 数据进行查表。**

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	-	-	-	-	-	-	-	-

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	-	-	-	-	-	-	-	-

➢ **例：用 Y、Z 作为数据指针，访问 bank0 中 025H 处的内容。**

```
B0MOV   Y, #00H     ; Y 指向 RAM bank 0。
B0MOV   Z, #25H     ; Z 指向 25H。
B0MOV   A, @YZ      ; 数据送入 ACC。
```

➢ **例：利用数据指针@YZ 对 RAM 数据清零。**

```
        B0MOV   Y, #0       ; Y = 0，指向 bank 0。
        B0MOV   Z, #7FH     ; Z = 7FH，RAM 区的最后单元。

CLR_YZ_BUF:
        CLR     @YZ         ; @YZ 清零。

        DECMS   Z           ;
        JMP     CLR_YZ_BUF  ; 不为零。

        CLR     @YZ
END_CLR:                    ;
        …
```

2.2.7 R寄存器

8 位缓存器 R 主要有以下两个功能：

- **作为工作寄存器使用；**
- **存储执行查表指令后的高字节数据。**（执行 MOVC 指令，指定 ROM 单元的高字节数据会被存入 R 寄存器而低字节数据则存入 ACC。）

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	-	-	-	-	-	-	-	-

✷ **注：关于 R 寄存器查表应用请参考查表章节。**

2.3 寻址模式

2.3.1 立即寻址模式

将立即数送入 ACC 或指定的 RAM 单元。

- **例：立即数 12H 送入 ACC。**

```
MOV     A, #12H
```

- **例：立即数 12H 送入寄存器 R。**

```
B0MOV   R, #12H
```

✷ **注：立即数寻址中，指定的 RAM 单元必须是 80H~87H 的工作寄存器。**

2.3.2 直接寻址模式

通过 ACC 对 RAM 单元数据进行操作。

- **例：地址 12H 处的内容送入 ACC。**

```
B0MOV   A, 12H
```

- **例：ACC 中数据写入 RAM 的 12H 单元。**

```
B0MOV   12H, A
```

2.3.3 间接寻址模式

通过指针寄存器（H/L，Y/Z）访问 RAM 数据。

- **例：用@HL 实现间接寻址。**

```
B0MOV   H, #0       ; H 清零以寻址 RAM bank 0。
B0MOV   L, #12H     ; 设定寄存器地址。
B0MOV   A, @HL
```

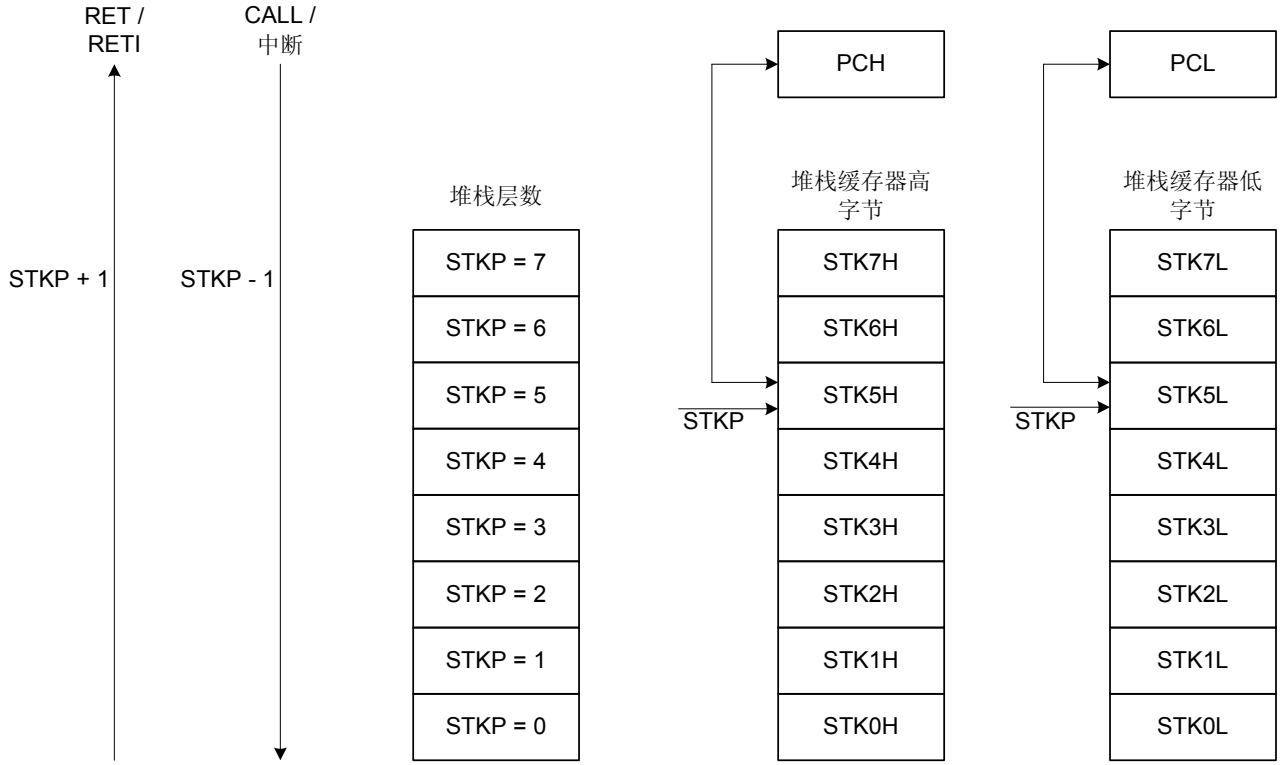
- **例：用@YZ 实现间接寻址。**

```
B0MOV   Y, #0       ; Y 清零以寻址 RAM bank 0。
B0MOV   Z, #12H     ; 设定寄存器地址。
B0MOV   A, @YZ
```

2.4 堆栈

2.4.1 概述

SN8PC22 的堆栈缓存器共有 8 层，程序进入中断或执行 CALL 指令时，用来存储程序计数器 PC 的值。寄存器 STKP 为堆栈指针，指向堆栈缓存器顶层，STKnH 和 STKnL 分别是各堆栈缓存器高、低字节。



2.4.2 堆栈寄存器

堆栈指针 STKP 是一个 3 位寄存器，存放被访问的堆栈单元地址，14 位数据存储器 STKnH 和 STKnL 用于暂存堆栈数据。以上寄存器都位于 bank 0。

通过入栈指令 PUSH 和出栈指令 POP 对堆栈缓存器进行操作。堆栈操作遵循后进先出（LIFO）的原则，入栈时堆栈指针 STKP 的值减 1，出栈时 STKP 的值加 1，这样，STKP 总是指向堆栈缓存器顶层单元。

系统进入中断或执行 CALL 指令之前，程序计数器 PC 的值被存入堆栈缓存器中进行入栈保护。

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

Bit[2:0] **STKPBn**：堆栈指针（n = 0 ~ 2）。

Bit 7 **GIE**：全局中断控制位。
0 = 禁止；
1 = 使能。

➢ **例：系统复位时，堆栈指针寄存器内容为默认值，但强烈建议在程序初始部分重新设定，如下面所示：**

```
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	-	-	SnPC13	SnPC12	SnPC11	SnPC10	SnPC9	SnPC8
读/写	-	-	R/W	R/W	R/W	R/W	R/W	R/W
复位后	-	-	0	0	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）。

2.4.3 堆栈操作举例

执行程序调用指令 CALL 和响应中断服务时，堆栈指针 STKP 的值减 1，指针指向下一个堆栈缓存器。同时，对程序计数器 PC 的内容进行入栈保存。

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

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	-

2.5 编译选项列表（CODE OPTION）

编译选项（CODE OPTION）是一种系统的硬件配置，包括振荡器类型，看门狗定时器的操作，复位引脚选项以及 OTP ROM 的安全控制。如下表所示：

编译选项	配置项目	功能说明
High_Clk	IHRC_8M	内部高速 8MHz RC，XIN/XOUT 引脚为 GPIO 引脚。
	IHRC_RTC	内部高速 8MHz RC，带有 0.5S RTC 功能，XIN/XOUT 引脚为 32768 晶振输入输出引脚。
	Ext_RC	外部高速振荡器采用廉价的 RC 振荡电路，XIN 引脚外接 RC 电路，XOUT 引脚为 GPIO 引脚。
	32K X'tal	外部高速振荡器采样低频、低功耗的晶体振荡器（如 32.768KHz）。
	12M X'tal	外部高速振荡器采用标准陶瓷/石英振荡器（如 12Hz）。
	4M X'tal	外部高速振荡器采用标准陶瓷/石英振荡器（如 4MHz）。
Fcpu	Fhosc/2	指令周期为 2 个振荡时钟。
	Fhosc/4	指令周期为 4 个振荡时钟。
	Fhosc/8	指令周期为 8 个振荡时钟。
	Fhosc/16	指令周期为 16 个振荡时钟。
Watch_Dog	Always_On	始终开启看门狗定时器，睡眠模式下也不例外。.
	Enable	开启看门狗定时器，但在睡眠模式下会处于关闭状态。
	Disable	关闭看门狗定时器。
Reset_Pin	Reset	使能外部复位引脚。
	P02	使能 P0.2 单向输入功能，无上拉电阻。
Security	Enable	ROM 代码加密。
	Disable	ROM 代码不加密。

2.5.1 Reset_Pin编译选项

复位引脚与单向输入引脚共用，由编译选项控制。

- **Reset**：使能外部复位引脚功能。当下降沿触发时，系统复位。
- **P02**：使能 P0.2 为单向输入引脚。此时禁止外部复位引脚功能。

2.5.2 Security编译选项

Security 编译选项是对 OTP ROM 的一种保护，当使能 Secuity 编译选项，ROM 代码加密，可以保护 ROM 的内容。

3 复位

3.1 概述

SN8PC22 有以下几种复位方式：

- **上电复位；**
- **看门狗复位；**
- **掉电复位；**
- **外部复位（仅在外部复位引脚处于使能状态）。**

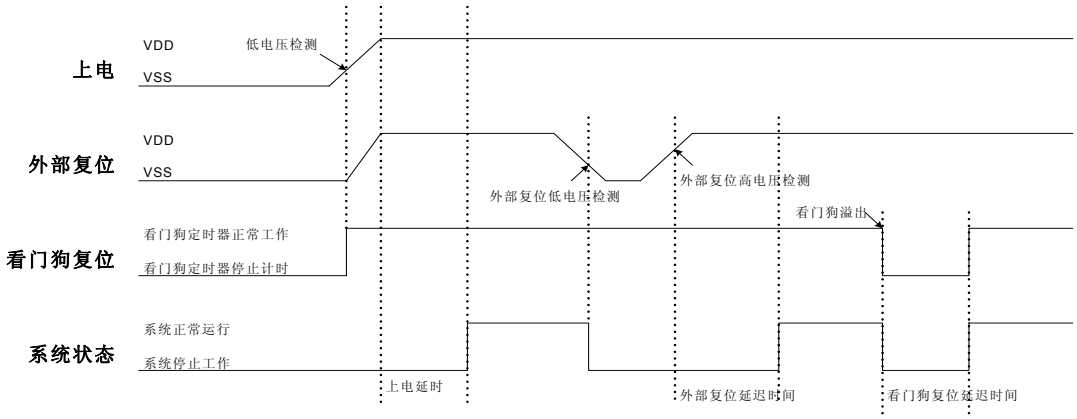
上述任一种复位发生时，所有的系统寄存器恢复默认状态，程序停止运行，同时程序计数器 PC 清零。复位结束后，系统从向量 0000H 处重新开始运行。PFLAG 寄存器的 NT0 和 NPD 两个标志位能够给出系统复位状态的信息。用户可以编程控制 NT0 和 NPD，从而控制系统的运行路径。

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	X	X	-	-	-	0	0	0

Bit [7:6] **NT0, NPD：**复位状态标志。

NT0	NPD	复位情况	说明
0	0	看门狗复位	看门狗溢出
0	1	保留	-
1	0	上电及 LVD 复位	电源电压低于 LVD 检测值
1	1	外部复位	外部复位引脚检测到低电平

任何一种复位情况都需要一定的响应时间，系统提供完善的复位流程以保证复位动作的顺利进行。对于不同类型的振荡器，完成复位所需要的时间也不同。因此，VDD 的上升速度和不同晶振的起振时间都不固定。RC 振荡器的起振时间最短，晶体振荡器的起振时间则较长。在用户终端使用的过程中，应注意考虑主机对上电复位时间的要求。



3.2 上电复位

上电复位与 LVD 操作密切相关。系统上电的过程呈逐渐上升的曲线形式，需要一定时间才能达到正常电平值。下面给出上电复位的正常时序：

- **上电**：系统检测到电源电压上升并等待其稳定；
- **外部复位（仅限于外部复位引脚使能状态）**：系统检测外部复位引脚状态。如果不为高电平，系统保持复位状态直到外部复位引脚释放；
- **系统初始化**：所有的系统寄存器被置为初始值；
- **振荡器开始工作**：振荡器开始提供系统时钟；
- **执行程序**：上电结束，程序开始运行。

3.3 看门狗复位

看门狗复位是系统的一种保护设置。在正常状态下，由程序将看门狗定时器清零。若出错，系统处于未知状态，看门狗定时器溢出，此时系统复位。看门狗复位后，系统重启进入正常状态。看门狗复位的时序如下：

- **看门狗定时器状态**：系统检测看门狗定时器是否溢出，若溢出，则系统复位；
- **系统初始化**：所有的系统寄存器被置为默认状态；
- **振荡器开始工作**：振荡器开始提供系统时钟；
- **执行程序**：上电结束，程序开始运行。

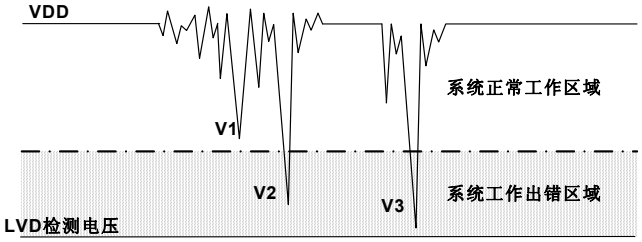
看门狗定时器应用注意事项：

- 对看门狗清零之前，检查 I/O 口的状态和 RAM 的内容可增强程序的可靠性；
- 不能在中断中对看门狗清零，否则无法侦测到主程序跑飞的状况；
- 程序中应该只在主程序中有一次清看门狗的动作，这种架构能够最大限度的发挥看门狗的保护功能。

✷ **注：关于看门狗定时器的详细内容，请参阅“看门狗定时器”有关章节。**

3.4 掉电复位

掉电复位针对外部因素引起的系统电压跌落情形（例如，干扰或外部负载的变化），掉电复位可能会引起系统工作状态不正常或程序执行错误。



掉电复位示意图

电压跌落可能会进入系统死区。系统死区意味着电源不能满足系统的最小工作电压要求。上图是一个典型的掉电复位示意图。图中，VDD 受到严重的干扰，电压值降的非常低。虚线以上区域系统正常工作，在虚线以下的区域内，系统进入未知的工作状态，这个区域称作死区。当 VDD 跌至 V1 时，系统仍处于正常状态；当 VDD 跌至 V2 和 V3 时，系统进入死区，则容易导致出错。以下情况系统可能进入死区：

DC 运用中：

DC 运用中一般都采用电池供电，当电池电压过低或单片机驱动负载时，系统电压可能跌落并进入死区。这时，电源不会进一步下降到 LVD 检测电压，因此系统维持在死区。

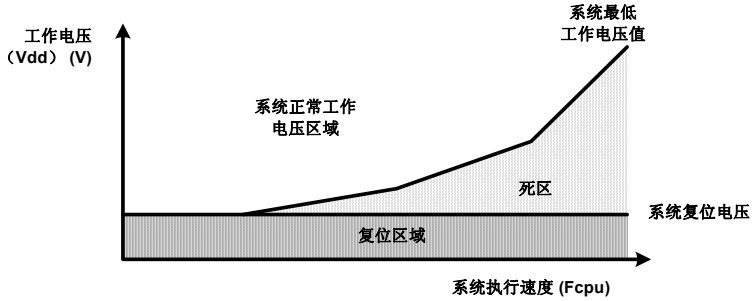
AC 运用中：

系统采用 AC 供电时，DC 电压值受 AC 电源中的噪声影响。当外部负载过高，如驱动马达时，负载动作产生的干扰也影响到 DC 电源。VDD 若由于受到干扰而跌落至最低工作电压以下时，则系统将有可能进入不稳定工作状态。

在 AC 运用中，系统上、下电时间都较长。其中，上电时序保护使得系统正常上电，但下电过程却和 DC 运用中情形类似，AC 电源关断后，VDD 电压在缓慢下降的过程中易进入死区。

3.4.1 系统工作电压

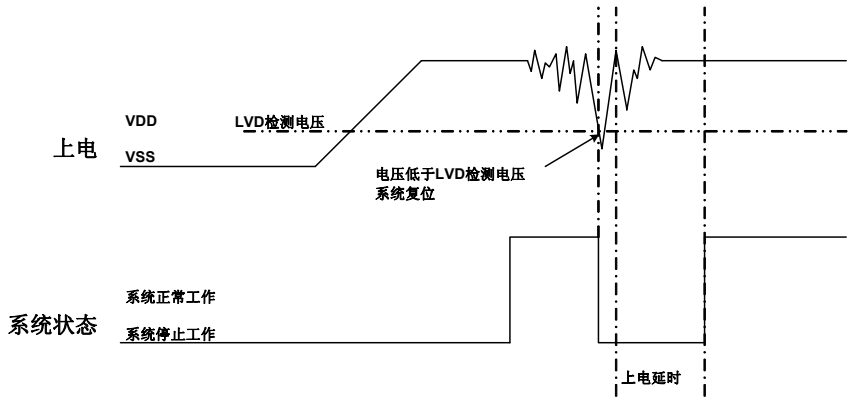
为了改善系统掉电复位的性能，首先必须明确系统具有的最低工作电压值。系统最低工作电压与系统执行速度有关，不同的执行速度下最低工作电压值也不同。



系统工作电压与执行速度关系图

如上图所示，系统正常工作电压区域一般高于系统复位电压，同时复位电压由低电压检测（LVD）电平决定。当系统执行速度提高时，系统最低工作电压也相应提高，但由于系统复位电压是固定的，因此在系统最低工作电压与系统复位电压之间就会出现一个电压区域，系统不能正常工作，也不会复位，这个区域即为死区。

3.4.2 低电压检测（LVD）



低电压检测（LVD）是 SONiX 8 位单片机内置的掉电复位保护装置，当 VDD 跌落并低于 LVD 检测电压值时，LVD 被触发，系统复位。不同的单片机有不同的 LVD 检测电平，LVD 检测电平值仅为一个电压点，并不能覆盖所有死区范围。因此采用 LVD 依赖于系统要求和环境状况。电源变化较大时，LVD 能够起到保护作用，如果电源变化触发 LVD，系统工作仍出错，则 LVD 就不能起到保护作用，就需要采用其它复位方法。

3.4.3 掉电复位性能改进

如何改善系统掉电复位性能，有以下几点建议：
LVD 复位；
看门狗复位；
降低系统工作速度；
采用外部复位电路（稳压二极管复位电路，电压偏移复位电路，外部 IC 复位电路）。

✷ **注：“稳压二极管复位电路”、“电压偏移复位电路”和“外部 IC 复位电路”能够完全避免掉电复位出错。**

看门狗复位：

看门狗定时器用于保证系统正常工作。通常，会在主程序中将看门狗定时器清零，但不要在多个分支程序中清看门狗。若程序正常运行，看门狗不会复位。当系统进入死区或程序运行出错的时候，看门狗定时器继续计数直至溢出，系统复位。

如果看门狗复位后电源仍处于死区，则系统复位失败，保持复位状态，直到系统工作状态恢复到正常值。

降低系统工作速度：

系统工作速度越快最低工作电压值越高，从而加大工作死区的范围，因此降低系统工作速度不失为降低系统进入死区几率的有效措施。所以，可选择合适的工作速度以避免系统进入死区，这个方法需要调整整个程序使其满足系统要求。

附加外部复位电路：

外部复位也能够完全改善掉电复位性能。有三种外部复位方式可改善掉电复位性能：稳压二极管复位电路，电压偏移复位电路和外部 IC 复位电路。它们都采用外部复位信号控制单片机可靠复位。

3.5 外部复位

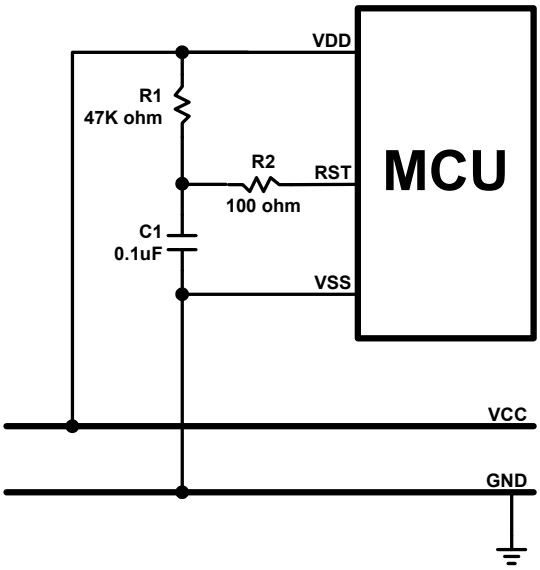
外部复位功能由编译选项“Reset_Pin”控制。将该编译选项置为“Reset”，可使能外部复位功能。外部复位引脚为施密特触发结构，低电平有效。复位引脚处于高电平时，系统正常运行。当复位引脚输入低电平信号时，系统复位。外部复位操作在上电和正常工作模式时有效。需要注意的是，在系统上电完成后，外部复位引脚必须输入高电平，否则系统将一直保持在复位状态。外部复位的时序如下：

- **外部复位（当且仅当外部复位引脚为使能状态）：**系统检测复位引脚的状态，如果复位引脚不为高电平，则系统会一直保持在复位状态，直到外部复位结束；
- **系统初始化：**所有的系统寄存器被置为初始状态；
- **振荡器开始工作：**振荡器开始提供系统时钟；
- **执行程序：**上电结束，程序开始运行。

外部复位可以在上电过程中使系统复位。良好的外部复位电路可以保护系统以免进入未知的工作状态，如 AC 应用中的掉电复位等。

3.6 外部复位电路

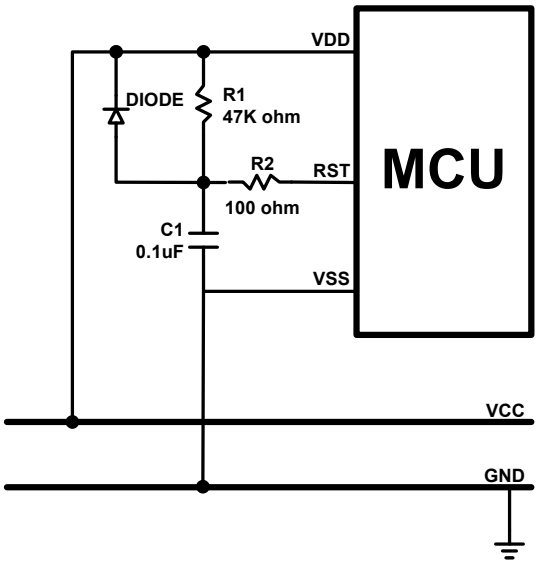
3.6.1 基本RC复位电路



上图为一个由电阻 R1 和电容 C1 组成的基本 RC 复位电路，它在系统上电的过程中能够为复位引脚提供一个缓慢上升的复位信号。这个复位信号的上升速度低于 VDD 的上电速度，为系统提供合理的复位时序，当复位引脚检测到高电平时，系统复位结束，进入正常工作状态。

✷ **注：此 RC 复位电路不能解决非正常上电和掉电复位问题。**

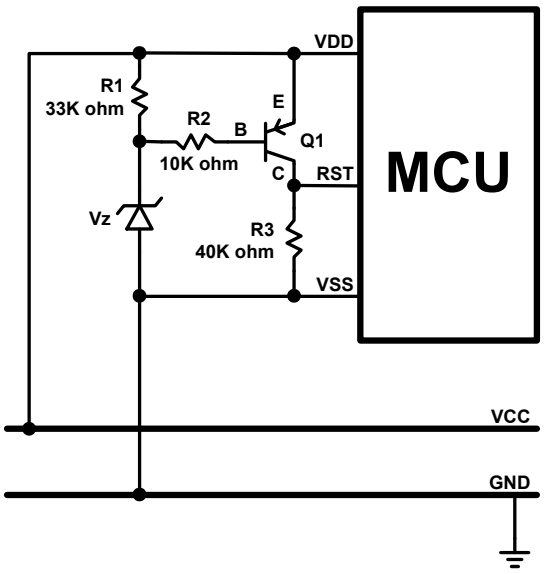
3.6.2 二极管& RC复位电路



上图中，R1 和 C1 同样是为复位引脚提供输入信号。对于电源异常情况，二极管正向导通使 C1 快速放电并与 VDD 保持一致，避免复位引脚持续高电平、系统无法正常复位。

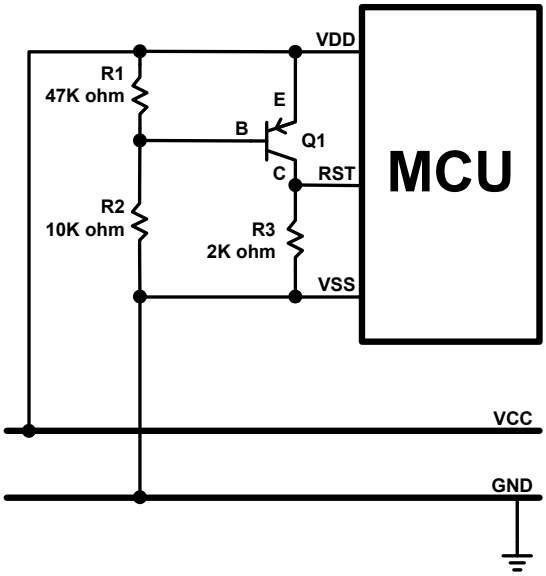
✷ **注：“基本 RC 复位电路”和“二极管及 RC 复位电路”中的电阻 R2 都是必不可少的限流电阻，以避免复位引脚 ESD（Electrostatic Discharge）或 EOS（Electrical Over-stress）击穿 。**

3.6.3 稳压二极管复位电路



稳压二极管复位电路是一种简单的 LVD 电路，基本上可以完全解决掉电复位问题。如上图电路中，利用稳压管的击穿电压作为电路复位检测值，当 VDD 高于“Vz + 0.7V”时，三极管集电极输出高电平，单片机正常工作；当 VDD 低于“Vz + 0.7V”时，三极管集电极输出低电平，单片机复位。稳压管规格不同则电路复位检测值不同，根据电路的要求选择合适的二极管。

3.6.4 电压偏置复位电路

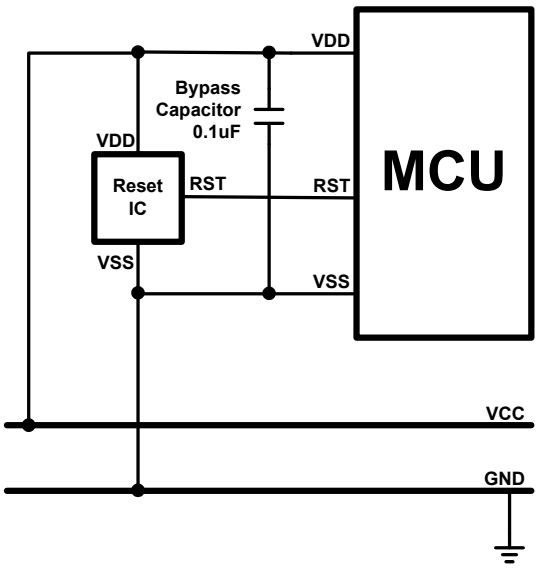


电压偏置复位电路是一种简单的 LVD 电路，基本上可以完全解决掉电复位问题。与稳压二极管复位电路相比，这种复位电路的检测电压值的精确度有所降低。电路中，R1 和 R2 构成分压电路，当 VDD 高于和等于分压值“0.7V x (R1 + R2) / R1”时，三极管集电极 C 输出高电平，单片机正常工作；VDD 低于“0.7V x (R1 + R2) / R1”时，集电极 C 输出低电平，单片机复位。

对于不同应用需求，选择适当的分压电阻。单片机复位引脚上电压的变化与 VDD 电压变化之间的差值为 0.7V。如果 VDD 跌落并低于复位引脚复位检测值，那么系统将被复位。如果希望提升电路复位电平，可将分压电阻设置为 R2>R1，并选择 VDD 与集电极之间的结电压高于 0.7V。分压电阻 R1 和 R2 在电路中要耗电，此处的功耗必须计入整个系统的功耗中。

✷ 注：在电源不稳定或掉电复位的情况下，“稳压二极管复位电路”和“偏压复位电路”能够保护电路在电压跌落时避免系统出错。当电压跌落至低于复位检测值时，系统将被复位。从而保证系统正常工作。

3.6.5 外部IC复位电路



外部复位也可以选用 IC 进行外部复位，但是这样一来系统成本将会增加。针对不用的应用要求选择适当的复位 IC，如上图所示外部 IC 复位电路，能够有效的降低电源变化对系统的影响。

4 系统时钟

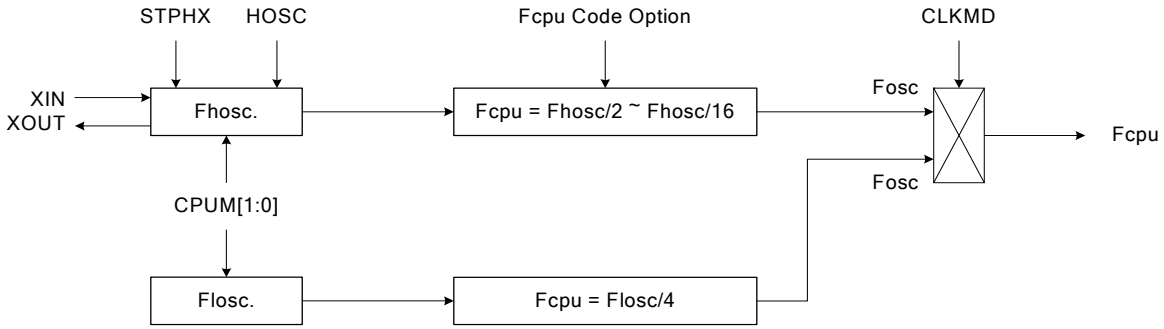
4.1 概述

SN8PC22 内置双时钟系统：高速时钟和低速时钟。高速时钟包括内部高速时钟和外部高速时钟，由编译选项 High_CLK 选择。低速时钟由内部低速振荡器提供，由 OSCM 寄存器的 CLKMD 位控制，高、低速时钟都可以作为系统时钟源。

- **高速振荡器**
  内部高速振荡器：高达 8MHz，称为 IHRC；
  外部高速振荡器：包括晶体（4MHz，12MHz，32KHz）振荡器和 RC 振荡器。

- **低速振荡器**
  内部低速振荡器：10KHz@3V，称为 ILRC。

- **系统时钟框图**



- **HOSC**：High_Clk 编译选项。
- **Fhosc**：外部高速/内部 RC 振荡器时钟频率。
- **Flosc**：内部低速 RC 时钟频率（10KHz@3V)。
- **Fosc**：系统时钟频率。
- **Fcpu**：指令执行频率。

SONIX 提供“Noise Filter”，由编译选项控制。高干扰环境下，Noise Fliter 可以滤除来自外部振荡器的高干扰信号以使系统正常工作。使能 Noise Filter 时，高速时钟下 Fcpu 被限制为 Fhosc/4。

4.2 指令周期FCPU

系统时钟速率，即指令周期（Fcpu），从系统时钟源分离出来，决定系统的工作速率。Fcpu 的速率由 Fcpu 编译选项决定，正常模式下，Fcpu=Fhosc/2~Fhosc/16。若高速时钟源为外部 4MHz 振荡器，则 Fcpu 编译选项选择 Fhosc/4，则 Fcpu 频率为 4MHz/4=1MHz。低速模式下，Fcpu=Flosc/4，即 10KHz/4=2.5KHz@3V。

高干扰环境下，强烈建议设置 Fcpu=Fhosc/4，以减少干扰的影响。

4.3 系统高速时钟

系统高速时钟包括外部高速时钟和内部高速时钟。外部高速时钟又包括4MHz、12MHz、32KHz晶体/陶瓷和RC振荡器，高速时钟振荡器由编译选项High_CLK选择。内部高速时钟支持实时时钟（RTC）功能，在IHRC_RTC模式下，内部高速时钟和外部32KHz振荡器有效，内部高速时钟作为系统时钟源，而外部32KHz振荡器为RTC时钟源，提供一个精确的实时时钟频率。

4.3.1 HIGH_CLK编译选项

对应不同的时钟功能，SONIX 提供多种高速时钟选项，由 High_CLK 选项控制。High_CLK 选项可以选择 IHRC_8M、RC、32K X'tal、12M X'tal 和 4M X'tal，以支持不同带宽的振荡器。

IHRC_8M：系统高速时钟源来自内部高速 8MHz RC 振荡器，XIN/XOUT 作为普通的 I/O 引脚，不连接任何外部振荡设备。

- **IHRC_RTC**：系统高速时钟源来自内部高速 8MHz RC 振荡器，RTC 时钟源为外部低速 32768Hz 振荡器。XIN/XOUT 连接外部 32768Hz 晶振，其 I/O 功能被禁止。
- **RC**：系统高速时钟源来自廉价的 RC 振荡电路，RC 振荡电路只需要和 XIN 引脚连接，XOUT 作为普通的 I/O 引脚。
- **32K X'tal**：系统高速时钟源来自外部低频 32768Hz 振荡器。该选项仅支持 32768Hz 晶体振荡器，RTC 正常工作。
- **12M X'tal**：系统高速时钟源来自外部高频晶体/陶瓷振荡器，其带宽为 10MHz~16MHz。
- **4M X'tal**：系统高速时钟源来自外部高频晶体/陶瓷振荡器，其带宽为 1MHz~10MHz。

关于功耗，选择 IHRC_RTC 选项时，绿色模式下内部高速振荡器和内部低速振荡器都停止工作，仅外部 32768Hz 晶振正常工作，此时，看门狗定时器不能选择 Always_On 选项，否则内部低速振荡器会正常工作。

4.3.2 内部高速RC振荡器（IHRC）

内部高速 8MHz RC 振荡器，普通环境下精确度为±2%，当选择 IHRC_8M 或 IHRC_RTC 时，使能内部高速振荡器。

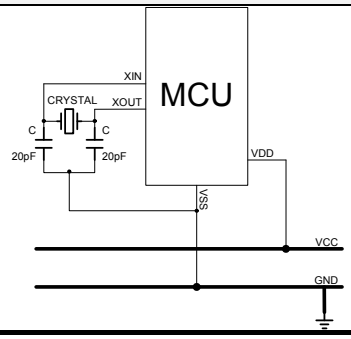
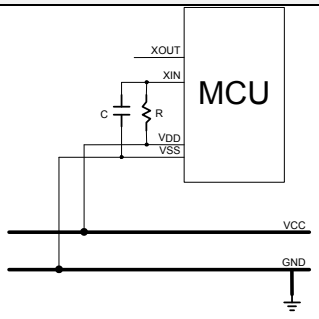
IHRC_8M：系统高速时钟为内部 8MHz RC 振荡器，XIN/XOUT 为普通 I/O 引脚。

- **IHRC_RTC**：系统高速时钟为内部 8MHz RC 振荡器，外部 32768Hz 晶振作为实时时钟的时钟源，XIN/XOUT 连接外部 32768Hz 晶振。

4.3.3 外部高速振荡器

外部高速振荡器包括 4MHz，12MHz，32KHz 和 RC。4M、12M 和 32K 可以使用晶体和陶瓷振荡器，XIN/XOUT 和 GND 之间需连接一个 20pF 的电容。廉价的 RC 振荡电路只需要和 XIN 引脚连接，电容的容值不能低于 100pF，电阻的阻值决定频率。

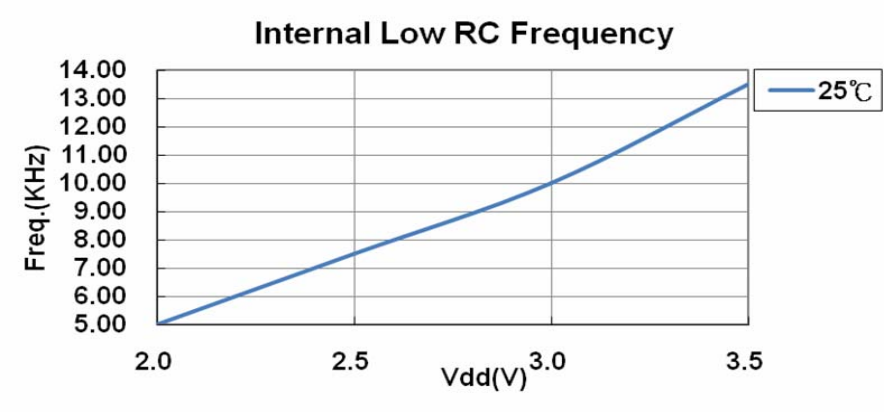
4.3.4 外部振荡应用电路

晶体/陶瓷	RC
	

✷ **注：晶体/陶瓷和电容 C 要尽可能的靠近单片机的 XIN/XOUT/VSS；电阻 R 和电容 C 要尽可能的靠近单片机的 VDD。**

4.4 系统低速时钟

系统低速时钟源即内置的低速振荡器，采用 RC 振荡电路。低速时钟的输出频率受系统电压和环境温度的影响，通常为 3V 时输出 10KHZ。输出频率与工作电压之间的关系如下图所示。



低速时钟可作为看门狗定时器以及系统低速模式的时钟源。由 OSCM 寄存器的 CLKMD 位来控制系统工作在低速模式。

- **Flosc = 内部低速 RC 振荡器（10KHz @3V）。**
- **低速模式 Fcpu = Flosc / 4。**

在睡眠模式下可以停掉内部低速 RC。

➢ **例：在睡眠模式下，停止内部低速振荡器。**

```
B0BSET      FCPUM0
```

✷ **注：不可以单独停止内部低速时钟；由寄存器 OSCM 的位 CPUM0 和 CPUM1（32K，禁止看门狗）的设置决定内部低速时钟的状态。**

4.5 OSCM寄存器

寄存器 OSCM 控制振荡器的状态和系统的工作模式。

095H	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	-

Bit 1 STPHX：高速振荡器控制位。
0 = 高速时钟正常运行；
1 = 高速振荡器停止，内部低速 RC 振荡器运行。

Bit 2 **CLKMD**：系统高/低速时钟模式控制位。
0 = 普通模式，系统采用高速时钟；
1 = 低速模式，系统采用内部低速时钟。

Bit[4:3] **CPUM[1:0]**：单片机工作模式控制位。
00 = 普通模式；
01 = 睡眠模式；
10 = 绿色模式；
11 = 系统保留。

STPHX 位为内部高速 RC 振荡器和外部高速振荡器的控制位。当 STPHX=0，内部高速 RC 振荡器和外部高速振荡器正常运行；当 STPHX=1，外部高速振荡器和内部高速 RC 振荡器停止运行。不同的高速时钟选项决定不同的 STPHX 功能。

- **IHRC_8M：STPHX=1，禁止内部高速 RC 振荡器；**
- **IHRC_RTC：STPHX=1，禁止内部高速 RC 振荡器和外部 32768Hz 振荡器；**
- **RC, 4M, 12M, 32K：STPHX=1，禁止外部振荡器。**

4.6 系统时钟测试

在设计过程中，用户可通过软件指令周期对系统时钟速度进行测试。

➢ **例：外部振荡器的 Fcpu 指令周期测试。**

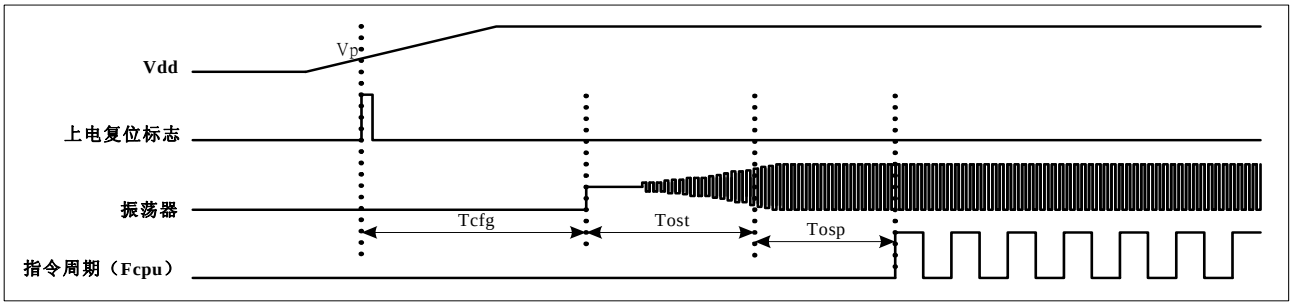
```
        B0BSET      P0M.0           ; P0.0 置为输出模式以输出 Fcpu 的触发信号。
@@:
        B0BSET      P0.0
        B0BCLR      P0.0
        JMP         @B
```

✷ **注：不能直接从 XIN 引脚测试 RC 振荡频率，因为探针的连接会影响测试的准确性。**

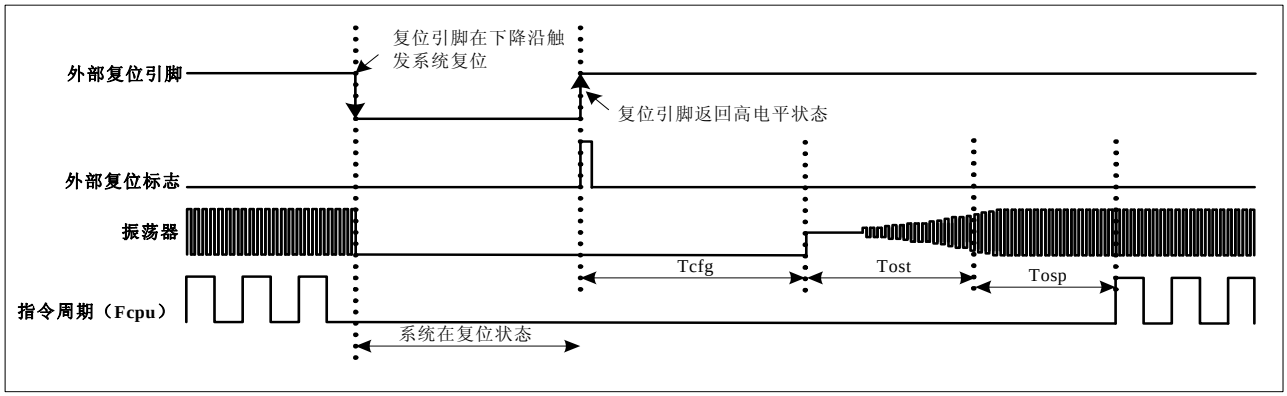
4.7 系统时钟时序

参数	符号	说明	典型值
硬件配置时间	Tcfg	2048*F_{ILRC}	64ms @ F_{ILRC} = 32KHz 128ms @ F_{ILRC} = 16KHz
振荡器启动时间	Tost	启动时间取决于振荡器的材料、工艺等。通常情况下，低速振荡器的启动时间要比高速振荡器的启动时间慢，RC 振荡器的启动时间要比晶体/陶瓷振荡器的启动时间快。	-
振荡器起振时间	Tosp	复位情况下的振荡器起振时间为 2048*Fhosc （使能上电复位，LVD 复位，看门狗复位，外部复位引脚）	64ms @ F_{hosc} = 32KHz 512us @ F_{hosc} = 4MHz 256us @ F_{hosc} = 8MHz
		睡眠模式唤醒情况的振荡器起振时间为： 2048*F_{hosc} ……晶体/陶瓷振荡器，如 32768Hz 晶振，4MHz 晶振，16MHz 晶振等； 32*F_{hosc}……RC 振荡器，如外部 RC 振荡电路，内部高速 RC 振荡器。	64ms @ F_{hosc} = 32KHz 512us @ F_{hosc} = 4MHz 256us @ F_{hosc} = 8MHz

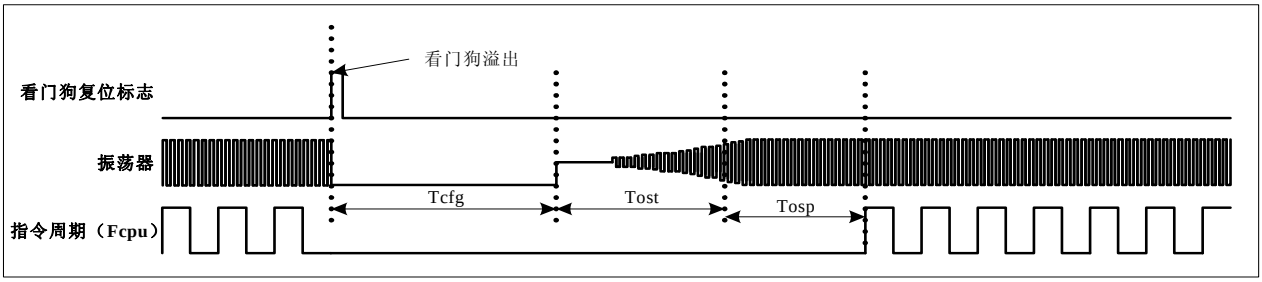
- 上电复位时序：



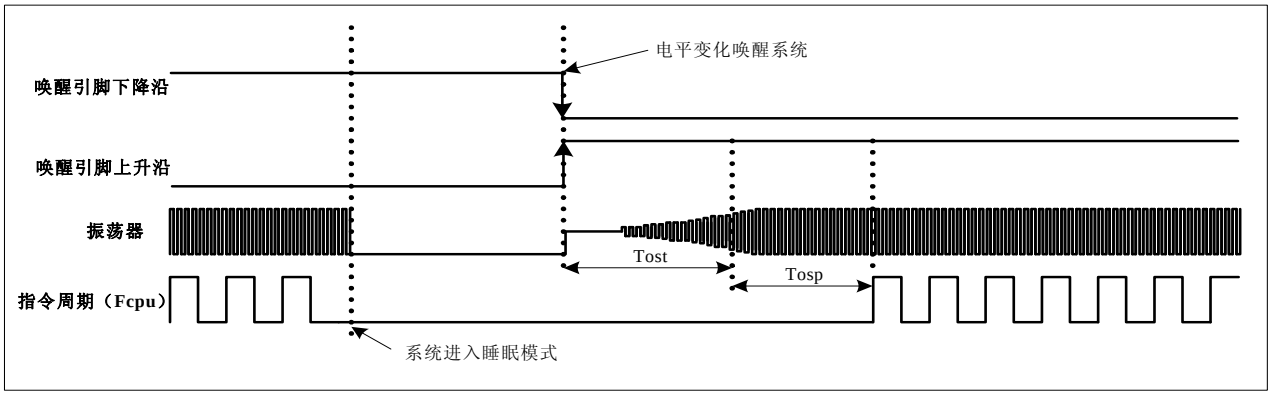
- 外部复位引脚复位时序：



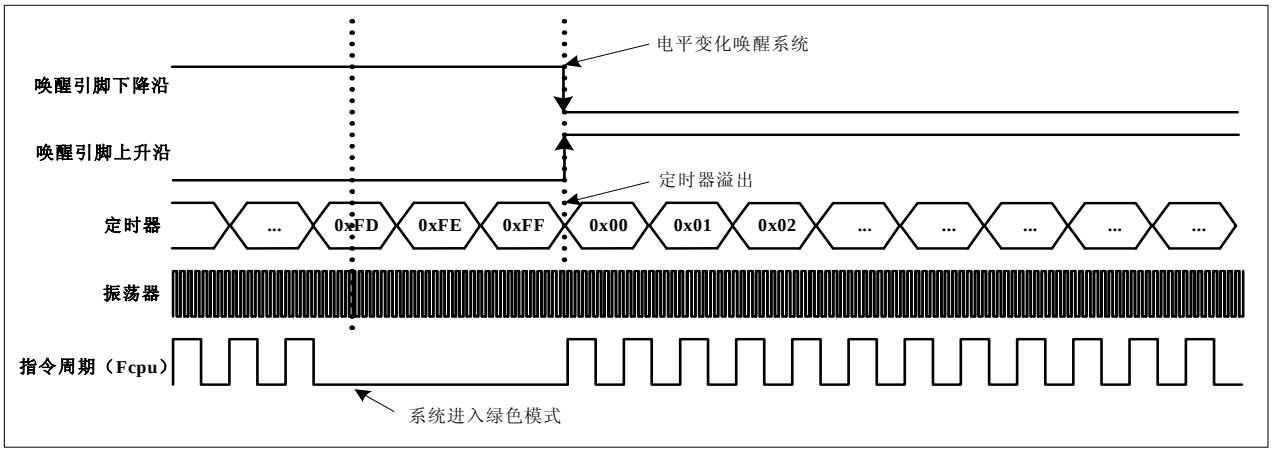
- 看门狗复位时序：



- 睡眠模式唤醒时序：

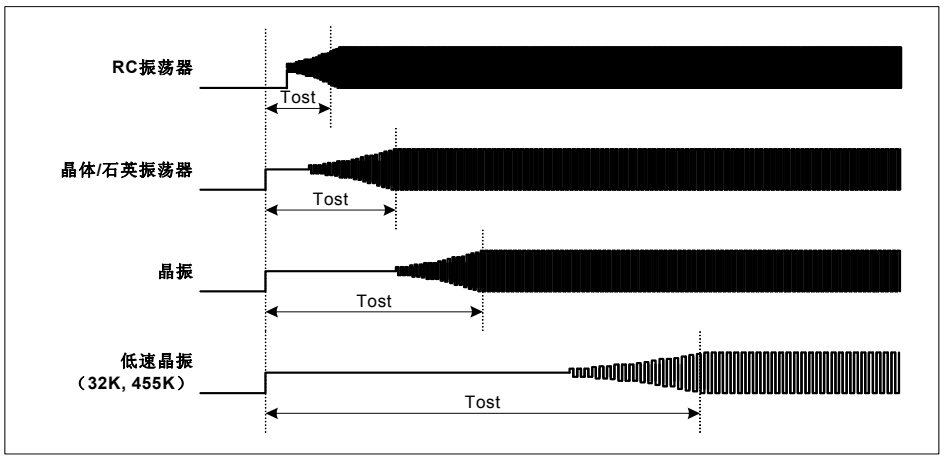


- 绿色模式唤醒时序：



- 振荡器启动时间

启动时间取决于振荡器的材料、工艺等。通常情况下，低速振荡器的启动时间要比高速振荡器的启动时间慢，RC 振荡器的启动时间要比晶体/陶瓷振荡器的启动时间快。



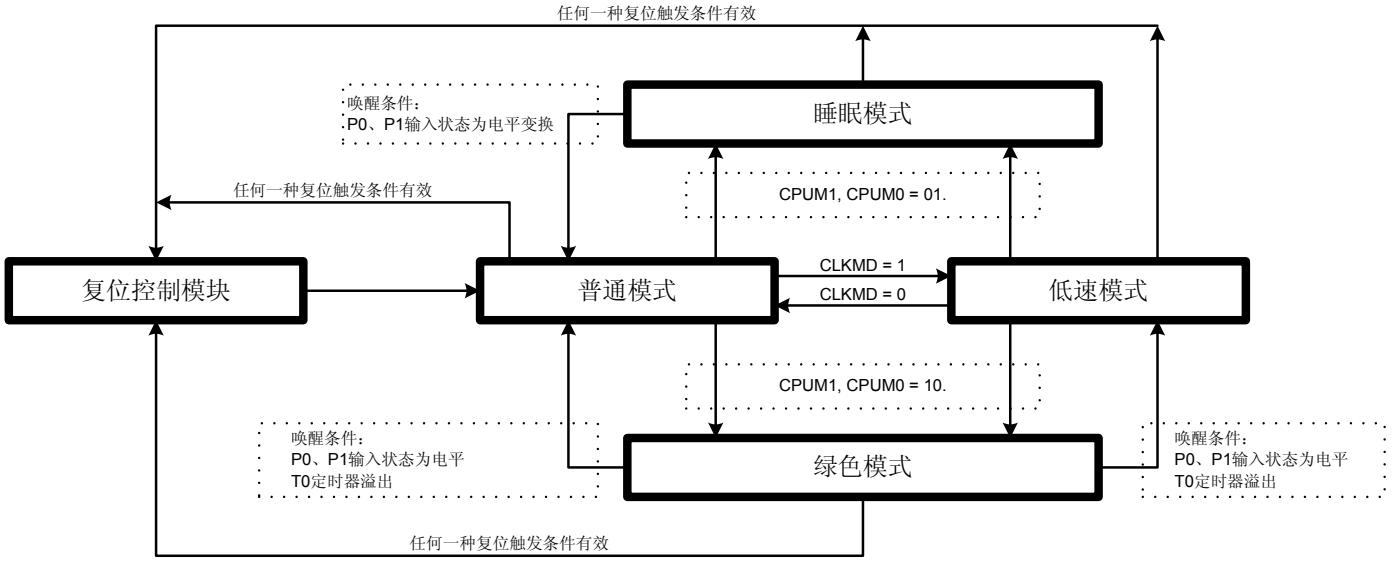
5 系统工作模式

5.1 概述

SN8PC22 可以在 4 种工作模式下以不同的时钟频率工作，这些模式可以控制振荡器的工作、程序的执行以及模拟电路的功能损耗。

- 普通模式：系统高速工作模式；
- 低速模式：系统低速工作模式；
- 省电模式：系统省电模式（睡眠模式）；
- 绿色模式：系统理想模式。

工作模式控制框图



工作模式时钟控制表

工作模式	普通模式	低速模式	绿色模式	睡眠模式
EHOSC	运行	STPHX	STPHX	停止
IHRC	运行	STPHX	STPHX	停止
ILRC	运行	运行	运行	停止
CPU 指令	执行	执行	停止	停止
T0 定时器	T0ENB	T0ENB	T0ENB	无效
看门狗定时器	Watch_Dog 编译选项	Watch_Dog 编译选项	Watch_Dog 编译选项	Watch_Dog 编译选项
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
唤醒功能	-	-	P0, P1, T0, 复位	P0, P1, 复位

- EHOSC：外部高速振荡器（XIN/XOUT）。
- IHRC：内部高速 RC 振荡器。
- ILRC：内部低速 RC 振荡器。

5.2 普通模式

普通模式是系统高速时钟正常工作模式，系统时钟源由高速振荡器提供。程序被执行。上电复位或任意一种复位触发后，系统进入普通模式执行程序。当系统从睡眠模式被唤醒后进入普通模式。普通模式下，高速振荡器正常工作，功耗最大。

- 程序被执行，所有的功能都可控制。
- 系统速率为高速。
- 高速振荡器和内部低速 RC 振荡器都正常工作。
- 通过 OSCM 寄存器，系统可以从普通模式切换到其它任何一种工作模式。
- 系统从睡眠模式唤醒后进入普通模式。
- 低速模式可以切换到普通模式。
- 从普通模式切换到绿色模式，唤醒后返回到普通模式。

5.3 低速模式

低速模式为系统低速时钟正常工作模式。系统时钟源由内部低速 RC 振荡器提供。低速模式由 OSCM 寄存器的 CLKMD 位控制。当 CLKMD=0 时，系统为普通模式；当 CLKMD=1 时，系统进入低速模式。切换进入低速模式后，不能自动禁止高速振荡器，必须通过 SPTHX 位来禁止以减少功耗。低速模式下，系统速率被固定为 Flosc/4（Flosc 为内部低速 RC 振荡器频率）。

- 程序被执行，所有的功能都可控制。
- 系统速率位低速（Flosc/4）。
- 内部低速 RC 振荡器正常工作，高速振荡器由 STPHX=1 控制。低速模式下，强烈建议停止高速振荡器。
- 通过 OSCM 寄存器，低速模式可以切换进入其它的工作模式。
- 从低速模式切换到睡眠模式，唤醒后返回到普通模式。
- 普通模式可以切换进入低速模式。
- 从低速模式切换到绿色模式，唤醒后返回到低速模式。

5.4 睡眠模式

睡眠模式是系统的理想状态，不执行程序，振荡器也停止工作。整个芯片的功耗低于 1uA。睡眠模式可以由 P0、P1 的电平变换触发唤醒。P1 的唤醒功能由 P1W 寄存器控制。从任何工作模式进入睡眠模式，被唤醒后都返回到普通模式。由 OSCM 寄存器的 CPUM0 位控制是否进入睡眠模式，当 CPUM0=1，系统进入睡眠模式。当系统从睡眠模式被唤醒后，CPUM0 被自动禁止（0 状态）。

- 程序停止执行，所有的功能被禁止。
- 所有的振荡器，包括外部高速振荡器、内部高速振荡器和内部低速振荡器都停止工作。
- 功耗低于 1uA。
- 系统从睡眠模式被唤醒后进入普通模式。
- 睡眠模式的唤醒源为 P0 和 P1 电平变换触发。

✷ 注：普通模式下，设置 STPHX=1 禁止高速时钟振荡器，这样，无系统时钟在执行，此时系统进入睡眠模式，可以由 P0、P1 电平变换触发唤醒。

5.5 绿色模式

绿色模式是另外的一种理想状态。在睡眠模式下，所有的功能和硬件设备都被禁止，但在绿色模式下，系统时钟保持工作，绿色模式下的功耗大于睡眠模式下的功耗。绿色模式下，不执行程序，但具有唤醒功能的定时器仍正常工作，定时器的时钟源为仍在工作的系统时钟。绿色模式下，有 2 种方式可以将系统唤醒：1、P0 和 P1 电平变换触发；2、具有唤醒功能的定时器溢出，这样，用户可以给定时器设定固定的周期，系统就在溢出时被唤醒。由 OSCM 寄存器 CPUM1 位决定是否进入绿色模式，当 CPUM1=1，系统进入绿色模式。当系统从绿色模式下被唤醒后，自动禁止 CPUM1（0 状态）。

- 程序停止执行，所有的功能被禁止。
- 具有唤醒功能的定时器正常工作。
- 作为系统时钟源的振荡器正常工作，其它的振荡器工作状态取决于系统工作模式的配置。
- 由普通模式切换到绿色模式，被唤醒后返回到普通模式。
- 由低速模式切换到绿色模式，被唤醒后返回到低速模式。
- 绿色模式下的唤醒方式为 P0、P1 电平变换触发唤醒和指定的定时器溢出。
- 绿色模式下 PWM 和 Buzzer 功能仍然有效，但是定时器溢出时不能唤醒系统。

✷ 注：sonix 提供宏“GreenMode”来控制绿色模式的工作状态，必要时使用宏“GreeMode”进绿色模式。该宏共有 3 条指令。但在使用 BRANCH 指令（如 BTS0、BTS1、B0BTS0、B0BTS1、INCS、INCMS、DECS、DECMS、CMPRS、JMP）时必须注意宏的长度，否则程序会出错。

5.6 工作模式控制宏

Sonix 提供工作模式控制宏以方便系统工作模式的切换。

宏名称	长度	说明
SleepMode	1-word	系统进入睡眠模式。
GreenMode	3-word	系统进入绿色模式。
SlowMode	2-word	系统进入低速模式并停止高速振荡器。
Slow2Normal	5-word	系统从低速模式返回到普通模式。该宏包括工作模式的切换，使能高速振荡器，高速振荡器唤醒延迟时间。

➢ **例：从普通模式/低速模式切换进入睡眠模式。**

```
SleepMode                       ; 直接宣告“SleepMode”宏。
```

➢ **例：从普通模式切换进入低速模式。**

```
SlowMode                        ; 直接宣告“SlowMode”宏。
```

➢ **例：从低速模式切换进入普通模式（外部高速振荡器停止工作）。**

```
Slow2Normal                     ; 直接宣告“Slow2Normal”宏。
```

➢ **例：从普通/低速模式切换进入绿色模式。**

```
GreenMode                       ; 直接宣告“GreenMode”宏。
```

➢ **例：从普通/低速模式切换进入绿色模式，并使能 T0 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 定时器。
        MOV         A,#20H      ;
        B0MOV       T0M,A       ; 设置 T0 时钟= Fcpu / 64。
        MOV         A,#74H
        B0MOV       T0C,A       ; 设置 T0C 的初始值= 74H（设置 T0 间隔值 = 10 ms）。
        B0BCLR      FT0IEN      ; 禁止 T0 中断。
        B0BCLR      FT0IRQ      ; 清 T0 中断请求。
        B0BSET      FT0ENB      ; 使能 T0 定时器。

; 进入绿色模式。
        GreenMode                ; 直接宣告“GreenMode”宏。
```

5.7 系统唤醒

5.7.1 概述

睡眠模式和绿色模式下，系统并不执行程序。唤醒触发信号可以将系统唤醒进入普通模式或低速模式。唤醒触发信号包括：外部触发信号（P0、P1 的电平变换）和内部触发（T0 定时器溢出）。

- 从睡眠模式唤醒后只能进入普通模式，且将其唤醒的触发只能是外部触发信号（P0、P1 电平变化）；
- 如果是将系统由绿色模式唤醒返回到上一个工作模式（普通模式或低速模式），唤醒触发信号可以是外部触发信号（P0、P1 电平变换）和内部触发信号（T0 溢出）。

5.7.2 唤醒时间

系统进入睡眠模式后，高速时钟停止运行。把系统从睡眠模式下唤醒时，单片机需要等待 2048 个外部高速振荡器时钟和 32 个内部高速时钟周期周期以使振荡电路进入稳定工作状态，等待的这一段时间就称为唤醒时间。唤醒时间结束后，系统才进入到普通模式。

✷ **注：从绿色模式下唤醒系统不需要唤醒时间，因为系统时钟在绿色模式下仍然正常工作。**

外部高速时钟振荡器的唤醒时间计算如下：

唤醒时间 = 1/Fosc * 2048（sec）+ 高速时钟启动时间

➢ **例：将系统从睡眠模式中唤醒，并设置系统进入普通模式。唤醒时间计算如下。**

唤醒时间= 1/Fosc * 2048 = 0.512 ms (Fosc = 4MHz)
总的唤醒时间 = 0.512 ms + 振荡器启动时间

内部高速 RC 振荡器的唤醒时间的计算如下：

唤醒时间 = 1/Fosc * 32 (sec) + 高速时钟启动时间

➢ **例：睡眠模式下，系统被唤醒进入普通模式。唤醒时间的计算如下：**

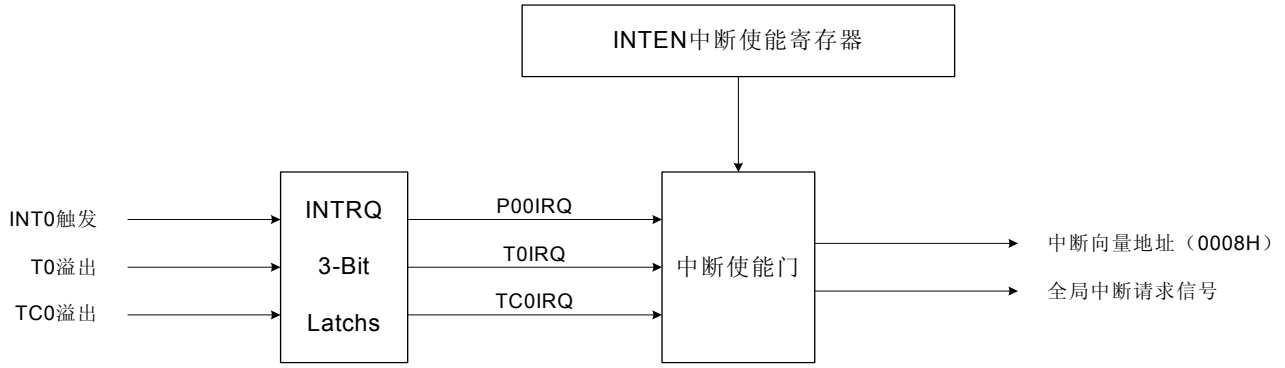
唤醒时间 = 1/Fosc * 32 = 2us（Fosc = 16MHz）

✷ **注：高速时钟的启动时间与 VDD 和振荡器类型有关。**

6 中断

6.1 概述

SN8PC22 提供 3 个中断源：2 个内部中断（T0/TC0）和 1 个外部中断（INT0）。系统从睡眠模式进入高速普通模式时，外部中断能够将单片机唤醒。一旦程序进入中断，寄存器 STKP 的位 GIE 将被硬件自动清零以避免再次响应其它中断。系统退出中断后，硬件自动将 GIE 置“1”，以响应下一个中断。中断请求存放在寄存器 INTRQ 中。



✷ **注：程序响应中断时，位 GIE 必须处于有效状态。**

6.2 中断使能寄存器INTEN

中断使能寄存器 INTEN 包括所有中断的使能控制位。INTEN 的有效位被置为“1”就使能了其相应的中断请求功能。一旦中断发生，程序进行压栈并跳转到中断向量（0008H）处执行中断服务程序。程序运行到指令 RETI 时，中断结束，系统退出中断服务。

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	-	-	TC0IEN	T0IEN	-	-	-	P00IEN
读/写	-	-	R/W	R/W	-	-	-	R/W
复位后	-	-	0	0	-	-	-	0

Bit 0 **P00IEN**：P0.0 外部中断（INT0）控制位。
0 = 禁止；
1 = 使能。

Bit 4 **T0IEN**：T0 中断控制位。
0 = 禁止；
1 = 使能。

Bit 5 **TC0IEN**：TC0 中断控制位。
0 = 禁止；
1 = 使能。

6.3 中断请求寄存器INTRQ

中断请求寄存器 INTRQ 中存放各中断请求标志。一旦有中断请求发生，INTRQ 中的相应位将被置“1”，该请求被响应后，程序应将该标志位清零。根据 INTRQ 的状态，程序判断是否有中断发生，并执行相应的中断服务。

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	-	-	TC0IRQ	T0IRQ	-	-	-	P00IRQ
读/写	-	-	R/W	R/W	-	-	-	R/W
复位后	-	-	0	0	-	-	-	0

Bit 0 **P00IRQ**：P0.0 中断（INT0）请求标志。
0 = INT0 无中断请求；
1 = INT0 有中断请求。

Bit 4 **T0IRQ**：T0 中断请求标志。
0 = T0 无中断请求；
1 = T0 有中断请求。

Bit 5 **TC0IRQ**：TC0 中断请求标志。
0 = TC0 无中断请求；
1 = TC0 有中断请求。

6.4 全局中断GIE

只有当全局中断控制位 GIE 置“1”的时候程序才能响应中断请求。一旦有中断发生，程序计数器（PC）指向中断向量地址（0008H），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

Bit 7 **GIE**：全局中断控制位。
0 = 禁止全局中断；
1 = 使能全局中断。

➢ **例：设置全局中断控制位（GIE）。**

```
B0BSET      FGIE        ; 使能 GIE。
```

✷ **注：在所有中断中，GIE 都必须处于使能状态。**

6.5 PUSH, POP

有中断请求发生并被响应后，程序转至 0008H 执行中断子程序。在响应中断之前，必须保存 ACC 和 PFLAG 的内容。系统提供 PUSH 和 POP 指令进行入栈保护和出栈恢复。

✷ **注： PUSH、POP 指令保存和恢复 ACC/PFLAG（不包括 NT0、NPD）的内容。PUSH/POP 缓存器只有一层。**

➢ **例：用 PUSH、POP 指令来保护和恢复 ACC 和 PFLAG。**

```
                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 PFALG。

                RETI                    ; 退出中断。
                …
                ENDP
```

6.6 INT0（P0.0）中断

INT0 被触发，则无论 P00IEN 处于何种状态，P00IRQ 都会被置“1”。如果 P00IRQ=1 且 P00IEN=1，系统响应该中断；如果 P00IRQ=1 而 P00IEN=0，系统并不会执行中断服务。在处理多中断时尤其需要注意。

如果中断的触发方向和唤醒功能的触发方向是一样的，则在系统由 P0.0 从睡眠模式和绿色模式唤醒时，INT0 的中断请求（INT0IRQ）就会被锁定。系统会在唤醒后马上进入中断向量地址执行中断服务程序。

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

Bit[4:3] **P00G[1:0]**：P0.0 中断触发控制位。

00 = 保留；
01 = 上升沿触发；
10 = 下降沿触发；
11 = 上升/下降沿触发（电平触发）。

➢ **例：INT0 中断请求设置，电平触发。**

```
        MOV     A, #18H
        B0MOV   PEDGE, A        ; INT0 置为电平触发。

        B0BCLR  FP00IRQ         ; INT0 中断请求标志清零。
        B0BSET  FP00IEN         ; 使能 INT0 中断。
        B0BSET  FGIE            ; 使能 GIE。
```

➢ **例：INT0 中断。**

```
        ORG     8H              ;
        JMP     INT_SERVICE
INT_SERVICE:

        …                       ; ACC 和 PFLAG 入栈保护。

        B0BTS1  FP00IRQ         ; 检测 P00IRQ。
        JMP     EXIT_INT        ; P00IRQ = 0，退出中断。

        B0BCLR  FP00IRQ         ; P00IRQ 清零。
        …                       ; INT1 中断服务程序。
        …
EXIT_INT:
        …                       ; ACC 和 PFLAG 出栈恢复。

        RETI                    ; 退出中断。
```

6.7 T0 中断

T0C 计数器溢出时，不管 T0IEN 是否使能，T0IRQ 会被置“1”，此时若 T0IEN＝1，则系统响应 T0 中断；若此时 T0IEN=0，则系统并不会响应 T0 中断。

➢ **例：T0 中断请求设置。Fcpu = 16MHz / 16。**

```
        B0BCLR      FT0IEN          ; 禁止 T0 中断。
        B0BCLR      FT0ENB          ;
        MOV         A, #20H         ;
        B0MOV       T0M, A          ; T0 时钟= Fcpu / 64。
        MOV         A, # 64H        ; T0C 初始值置为 64H。
        B0MOV       T0C, A          ; T0 间隔为 10 ms。

        B0BCLR      FT0IRQ          ; T0 中断请求标志清零。
        B0BSET      FT0IEN          ; 允许响应 T0 中断。
        B0BSET      FT0ENB          ;

        B0BSET      FGIE            ; 使能 GIE。
```

➢ **例：T0 中断程序。**

```
        ORG         8H              ;
        JMP         INT_SERVICE
INT_SERVICE:
        …                           ; ACC 和 PFLAG 入栈保存。
        B0BTS1      FT0IRQ          ; 检查是否有 T0 中断请求标志。
        JMP         EXIT_INT        ;

        B0BCLR      FT0IRQ          ; 清 T0IRQ。
        MOV         A, #64H
        B0MOV       T0C, A          ;
        …                           ; T0 中断程序。
EXIT_INT:
        …                           ; ACC 和 PFLAG 出栈恢复。

        RETI                        ; 退出中断。
```

6.8 TC0 中断

TC0C 溢出时，无论 TC0IEN 处于何种状态，TC0IRQ 都会置“1”。若 TC0IEN 和 TC0IRQ 都置“1”，系统就会响应 TC0 的中断；若 TC0IEN = 0，则无论 TC0IRQ 是否置“1”，系统都不会响应 TC0 中断。尤其需要注意多种中断下的情形。

➢ **例：TC0 中断请求设置。**

```
        B0BCLR      FTC0IEN         ; 禁止 TC0 中断。
        B0BCLR      FTC0ENB         ;
        MOV         A, #20H         ;
        B0MOV       TC0M, A         ; TC0 时钟＝Fcpu / 64。
        MOV         A, # 64H        ; TC0C 初始值＝64H。
        B0MOV       TC0C, A         ; TC0 间隔= 10 ms。

        B0BCLR      FTC0IRQ         ; 清 TC0 中断请求标志。
        B0BSET      FTC0IEN         ; 使能 TC0 中断。
        B0BSET      FTC0ENB         ;

        B0BSET      FGIE            ; 使能 GIE。
```

➢ **例：TC0 中断服务程序。**

```
        ORG         8H              ;
        JMP         INT_SERVICE
INT_SERVICE:

        …                           ; 保存 ACC 和 PFLAG。

        B0BTS1      FTC0IRQ         ; 检查是否有 TC0 中断请求标志。
        JMP         EXIT_INT        ; TC0IRQ = 0，退出中断。

        B0BCLR      FTC0IRQ         ; 清 TC0IRQ。
        MOV         A, #74H
        B0MOV       TC0C, A         ; 清 TC0C。
        …                           ; TC0 中断程序。
        …
EXIT_INT:
        …                           ; 恢复 ACC 和 PFLAG。

        RETI                        ; 退出中断。
```

6.9 多中断操作

在同一时刻，系统中可能出现多个中断请求。此时，用户必须根据系统的要求对各中断进行优先权的设置。中断请求标志 IRQ 由中断事件触发，当 IRQ 处于有效值“1”时，系统并不一定会响应该中断。各中断触发事件如下表所示：

中断	有效触发
P00IRQ	由 PEDGE 控制
T0IRQ	T0C 溢出
TC0IRQ	TC0C 溢出

多个中断同时发生时，需要注意的是：首先，必须预先设定好各中断的优先权；其次，利用 IEN 和 IRQ 控制系统是否响应该中断。在程序中，必须对中断控制位和中断请求标志进行检测。

➢ **例：多中断条件下检测中断请求。**

```
        ORG     8H                  ;
        JMP     INT_SERVICE
INT_SERVICE:
        …                           ; 保存 ACC 和 PFLAG。
INTP00CHK:                          ; 检查是否有 INT0 中断请求。
        B0BTS1  FP00IEN             ; 检查是否使能 INT0 中断。
        JMP     INTP01CHK           ; 跳到下一个中断。
        B0BTS0  FP00IRQ             ; 检查是否有 INT0 中断请求。
        JMP     INTP00              ; 进入 INT0 中断。
INTT0CHK:                           ; 检查是否有 T0 中断请求。
        B0BTS1  FT0IEN              ; 检查是否使能 T0 中断。
        JMP     INTTC0CHK           ; 跳到下一个中断。
        B0BTS0  FT0IRQ              ; 检查是否有 T0 中断请求。
        JMP     INTT0               ; 进入 T0 中断。
INTTC0CHK:                          ; 检查是否有 TC0 中断请求。
        B0BTS1  FTC0IEN             ; 检查是否使能 TC0 中断。
        JMP     INTTC1CHK           ; 跳到下一个中断。
        B0BTS0  FTC0IRQ             ; 检查是否有 TC0 中断请求。
        JMP     INTTC0              ; 进入 TC0 中断。
INT_EXIT:
        …                           ; 恢复 ACC 和 PFLAG。
        RETI                        ; 退出中断。
```

7 I/O口

7.1 概述

SN8PC22 内带 18 个 I/O 引脚，大多数 I/O 引脚与模拟引脚及特殊功能的引脚共用，详见下表：

I/O 引脚		共用引脚		共用引脚控制条件
引脚名称	引脚类型	引脚名称	引脚类型	
P0.0	I/O	INT0	DC	P00IEN=1
P0.2	I	RST	DC	Reset_Pin code option = Reset
		VPP	HV	OTP Programming
P0.4	I/O	XOUT	AC	High_CLK code option = 32K, 4M, 12M
P0.3	I/O	XIN	AC	High_CLK code option = RC, 32K, 4M, 12M
P5.0	I/O	IRXO	DC	IRXEN=1, IRXO=1.
P5.4	I/O	IROUT	DC	IREN = 1.
		IRIN	DC	IRXEN=1.

*** DC：数字特性；AC：模拟特性；HV：高压特性。**

7.2 I/O口模式

寄存器 PnM 控制 I/O 口的工作模式。

0B8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0M	P07M	P06M	P05M	P04M	P03M	-	P01M	P00M
读/写	R/W	R/W	R/W	R/W	R/W	-	R/W	R/W
复位后	0	0	0	0	0	-	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	P11M	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

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

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

✸ **注：用户可通过位操作指令（B0BSET、B0BCLR）对 I/O 口进行编程控制；**
✸ **注：P0.2 是单向输入引脚，P0M.2 未定义。**

➢ **例：I/O 模式选择。**

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

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

B0BCLR  P1M.0       ; P1.0 设为输入模式。

B0BSET  P1M.0       ; P1.0 设为输出模式。
```

7.3 I/O口上拉电阻

I/O 引脚内置上拉电阻，仅在输入模式时有效，可通过 PnUR 寄存器编程控制。当 PnUR 寄存器的相关位置 0 时，禁止上拉电阻，置 1 时，使能上拉电阻。

0E0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0UR	P07R	P06R	P05R	P04R	P03R	-	P01R	P00R
读/写	W	W	W	W	W	-	W	W
复位后	0	0	0	0	0	-	0	0

0E10H	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

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

✷ **注：P0.2 为单向输入引脚，无上拉电阻，故 P0UR.2 未定义。**

➢ **例：I/O 口的上拉电阻。**

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

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	P07	P06	P05	P04	P03	P02	P01	P00
读/写	R/W	R/W	R/W	R/W	R/W	R	R/W	R/W
复位后	0	0	0	0	0	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

0D5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5	-	-	-	P54	-	-	-	P50
读/写	-	-	-	R/W	-	-	-	R/W
复位后	-	-	-	0	-	-	-	0

✷ **注：当使能外部复位时， P02 的值保持为“1”。**

➢ **例：从输入口读取数据。**

```
B0MOV   A, P0     ; 从 P0 读数据。
B0MOV   A, P1     ; 从 P1 读数据。
B0MOV   A, P5     ; 从 P5 读数据。
```

➢ **例：写数据到输出端。**

```
MOV     A, #0FFH  ; 立即数 0FFH 写入所有输出口。
B0MOV   P0, A
B0MOV   P1, A
B0MOV   P5, A
```

➢ **例：写 1 位数据到输出口。**

```
B0BSET  P1.0      ; P1.0 和 P5.4 置“1”。
B0BSET  P5.4

B0BCLR  P1.0      ; P1.0 和 P5.4 置“0”。
B0BCLR  P5.4
```

8 定时器

8.1 看门狗定时器

看门狗定时器 WDT 是一个 4 位二进制计数器，用于监控程序的正常执行。如果由于干扰，程序进入了未知状态，看门狗定时器溢出，系统复位。看门狗的工作模式由编译选项控制，其时钟源由内部低速 RC 振荡器（10KHz @3V）提供。

看门狗溢出时间 = 8192 /内部低速振荡器周期（sec）

VDD	内部低速 RC Freq.	看门狗溢出时间
3V	10KHz	819.2ms

看门狗定时器的 3 种工作模式由编译选项“WatchDog”控制：

- **Disable**：禁止看门狗定时器功能。
- **Enable**：使能看门狗定时器功能，在普通模式和低速模式下有效，在睡眠模式和绿色模式下看门狗停止工作。
- **Always_On**：使能看门狗定时器功能，在睡眠模式和绿色模式下，看门狗仍会正常工作。

在高干扰环境下，强烈建议将看门狗设置为“Always_On”以确保系统在出错状态和重启时正常复位。

看门狗清零的方法是对看门狗计数器清零寄存器 WDTR 写入清零控制字 5AH。

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：如下是对看门狗定时器的操作，在主程序开头对看门狗清零。**

```
        MOV     A,#5AH          ; 看门狗定时器清零。
        B0MOV   WDTR,A
        …
        CALL    SUB1
        CALL    SUB2
        …
        JMP     MAIN
```

➢ **例：用宏指令@RST_WDT 清看门狗定时器。**

```
Main:
        @RST_WDT                ; 清看门狗定时器。
        …
        CALL    SUB1
        CALL    SUB2
        …
        JMP     Main
```

看门狗定时器应用注意事项如下：

- 对看门狗清零之前，检查 I/O 口的状态和 RAM 的内容可增强程序的可靠性；
- 不能在中断中对看门狗清零，否则无法侦测到主程序跑飞的状况；
- 程序中应该只在主程序中有一次清看门狗的动作，这种架构能够最大限度的发挥看门狗的保护功能。

➢ **例：如下是对看门狗定时器的操作，在主程序开头对看门狗清零。**

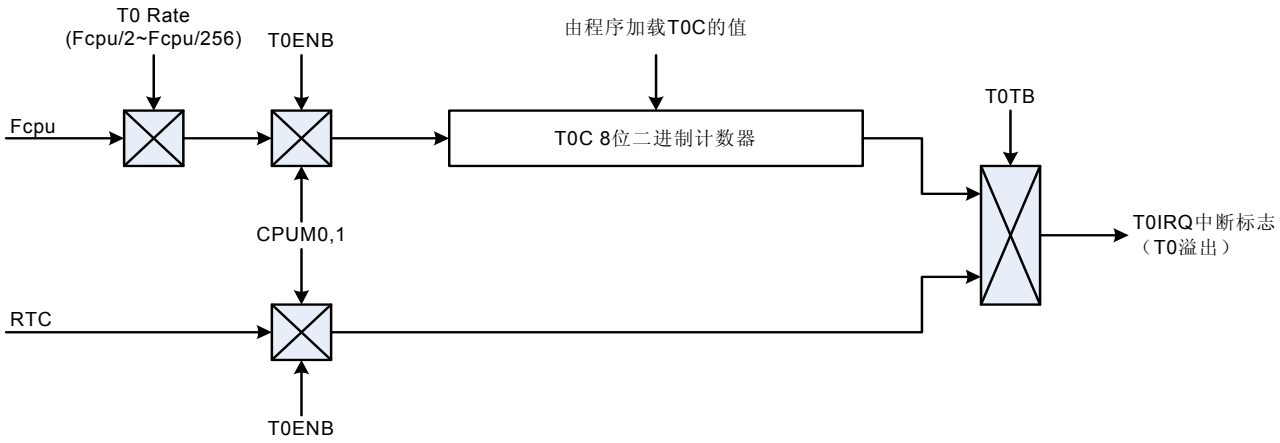
```
main:
        …                       ; 检测 I/O 口的状态。
        …                       ; 检测 RAM 的内容。
Err:    JMP $                   ; I/O 或 RAM 出错，不清看门狗等看门狗计时溢出。

Correct:                        ; I/O 和 RAM 正常，看门狗清零 。
                                ;
        MOV     A, #5AH         ; 在整个程序中只有一处地方清看门狗。
        B0MOV   WDTR, A
        …
        CALL    SUB1
        CALL    SUB2
        …
        JMP     MAIN
```

8.2 8 位基本定时器T0

8.2.1 概述

8 位二进制基本定时器 T0 具有定时器功能置：支持标志指示（T0IRQ）和中断操作（中断向量）。可以通过 T0M 和 T0C 寄存器控制间隔时间，支持 RTC 功能，具有在绿色模式下唤醒功能。在绿色模式下，T0 溢出，则将系统唤醒返回到上一个工作模式。

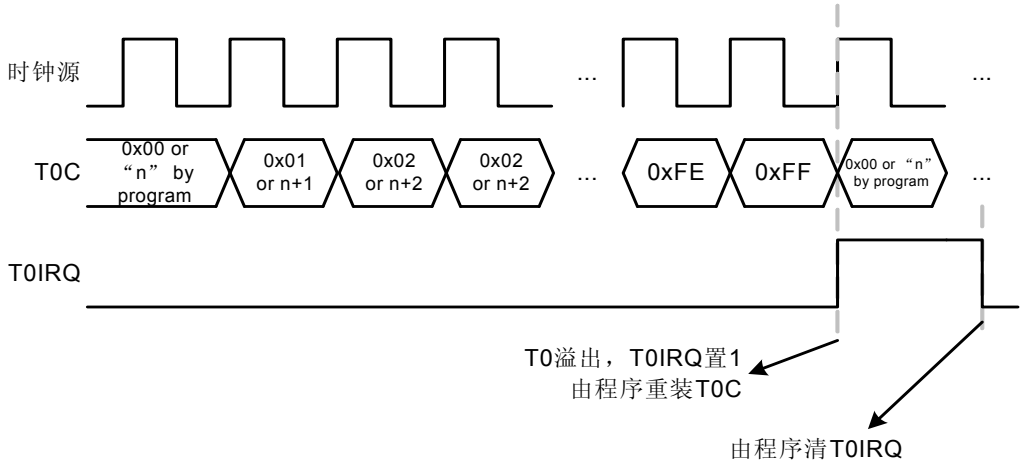
- **8 位可编程计数定时器：**根据选择的时钟频率周期性的产生中断请求。
- **中断功能：**T0 定时器支持中断功能，当 T0 溢出，T0IRQ 有效，程序计数器跳到中断向量地址执行中断。
- **RTC 功能：**T0 支持 RTC 功能。T0TB=1 时，RTC 时钟源由外部低速 32K 振荡器提供。RTC 功能仅在 High_Clk 选择 IHRC_RTC 时有效。
- **绿色模式唤醒功能：**T0 定时器在绿色模式下正常工作，溢出时将系统从绿色模式下唤醒。



✸ **注：RTC 模式下，T0 的间隔时间固定为 0.5S，T0C 的值为 256。**

8.2.2 T0 操作

T0 定时器由 T0ENB 控制。当 T0ENB=0 时，T0 停止工作；当 T0ENB=1 时，T0 开始计数。T0C 溢出（从 0FFH 到 00H）时，T0IRQ 置 1 显示溢出状态并由程序清零。T0 无内置双重缓存器，故 T0 溢出时由程序加载新值给 T0C，以选定合适的间隔时间。如果使能 T0 中断（T0IEN=1），T0 溢出后系统执行中断服务程序，在中断下必须由程序清 T0IRQ。T0 可以在普通模式、低速模式和绿色模式下工作，绿色模式下，T0 溢出时 T0IRQ 置 1，系统被唤醒。



T0 的时钟源为 Fcpu（指令周期），由 T0Rate[2:0]决定。详见下表：

T0rate[2:0]	T0 时钟	T0 间隔时间					
		Fhosc=16MHz, Fcpu=Fhosc/4		Fhosc=4MHz, Fcpu=Fhosc/4		IHRC_RTC 模式	
		max. (ms)	Unit (us)	max. (ms)	Unit (us)	max. (sec)	Unit (ms)
000b	Fcpu/256	16.384	64	65.536	256	-	-
001b	Fcpu/128	8.192	32	32.768	128	-	-
010b	Fcpu/64	4.096	16	16.384	64	-	-
011b	Fcpu/32	2.048	8	8.192	32	-	-
100b	Fcpu/16	1.024	4	4.096	16	-	-
101b	Fcpu/8	0.512	2	2.048	8	-	-
110b	Fcpu/4	0.256	1	1.024	4	-	-
111b	Fcpu/2	0.128	0.5	0.512	2	-	-
-	32768Hz/64	-	-	-	-	0.5	1.953

8.2.3 T0M模式寄存器

模式寄存器 T0M 设置 T0 的工作模式，包括 T0 前置分频器、时钟源等，这些设置必须在使能 T0 定时器之前完成。

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0rate2	T0rate1	T0rate0	T0TB	CPTS1	CPTS0	IRXEN
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

Bit3 **T0TB**：RTC 时钟源选择控制位。
0 = 禁止 RTC（T0 时钟源为 Fcpu）；
1 = 使能 RTC。

Bit [6:4] **T0RATE[2:0]**：T0 分频选择位。
000 = Fcpu/256；001 = Fcpu/128；010 = Fcpu/64；011 = Fcpu/32；100 = Fcpu/16；101 = Fcpu/8；110 = Fcpu/4；111 = Fcpu/2。

Bit 7 **T0ENB**：T0 启动控制位。
0 = 禁止；
1 = 使能。

✷ **注：RTC 模式下，T0RATE 处于无效状态。T0 的间隔时间固定为 0.5S。**

8.2.4 T0C计数寄存器

8 位计数器 T0C 溢出时，T0IRQ 置 1 并由程序清零，用来控制 T0 的中断间隔时间。必须保证写入正确的值到 T0C 寄存器，然后使能 T0 定时器以保证第一个周期准确无误。T0 溢出后，由程序加载一个正确的值到 T0C 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

T0C 初始值的计算公式如下：

T0C 初始值 = 256 -（T0 中断间隔时间 * T0 时钟 RATE）

➢ **例：T0 的中断间隔时间为 10ms，T0 时钟源为 Fcpu = 4MHz/4/=1MHz，T0RATE = 001（Fcpu/128）。**
T0 中断间隔时间为 10ms，T0 时钟 rate＝4MHz/4/128

T0C 初始值 = 256 -（T0 中断间隔时间 * T0 时钟 rate）
= 256 -（10ms * 4MHz / 4/ 128）
= 256 -（10-2 * 4* 106 / 4/128）
= B2H

✷ **注：RTC 模式下，T0C 为 256，T0 的间隔时间为 0.5S。不能在 RTC 模式下修改 T0C 的值。**

8.2.5 T0 操作举例

- **T0 定时器：**

```
; 复位 T0 定时器。
        MOV     A, #00H         ; 清 T0M。
        B0MOV   T0M, A

; 设置 T0 时钟源和 T0Rate。
        MOV     A, #0nnn0000b
        B0MOV   T0M, A

; 设置 T0C 寄存器获取 T0 间隔时间。
        MOV     A, #value
        B0MOV   T0C, A

; 清 T0IRQ。
        B0BCLR  FT0IRQ

; 使能 T0 定时器和中断功能。
        B0BSET  FT0IEN          ; 使能 T0 中断。
        B0BSET  FT0ENB          ; 使能 T0 定时器。
```

- **T0 在 RTC 模式下工作：**

```
; 复位 T0 定时器。
        MOV     A, #00H         ; 清 T0M。
        B0MOV   T0M, A

; 设置 T0 RTC 功能。
        B0BSET  FT0TB

; 清 T0C。
        CLR     T0C

; 清 T0IRQ。
        B0BCLR  FT0IRQ

; 使能 T0 定时器和中断功能。
        B0BSET  FT0IEN          ; 使能 T0 中断。
        B0BSET  FT0ENB          ; 使能 T0 定时器。
```

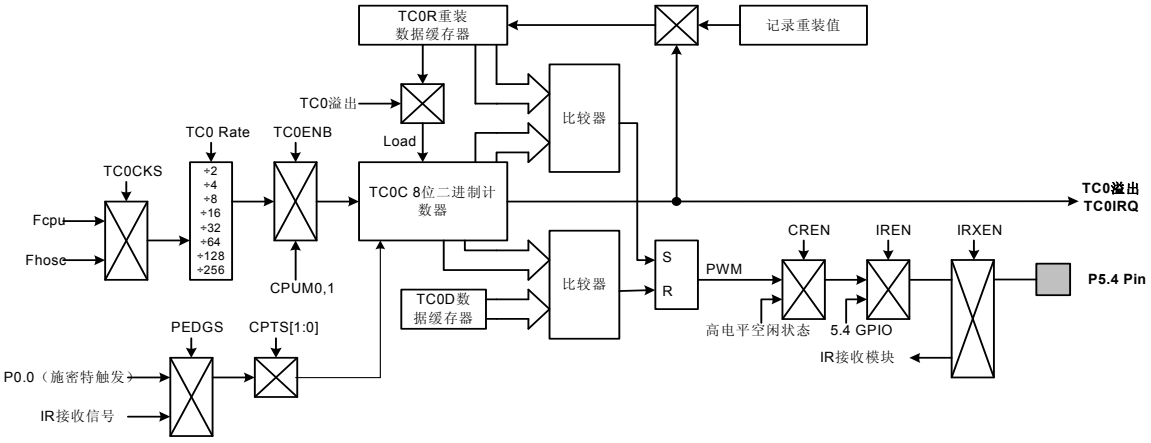
8.3 8 位定时器TC0/占空比可编程控制IR驱动

8.3.1 概述

8 位二进制定时器具有基本定时器、捕捉定时器和 PWM 功能。TC0 溢出（由 0FFH 到 00H）时，TC0 继续计数并给出一个溢出信号。TC0 内置捕捉定时器功能，用来测量输入信号的高电平脉冲、低电平脉冲和周期。TC0 还内置 PWM 功能，PWM 的周期和分辨率由 TC0R、TC0R 寄存器编程控制，故可以方便处理 Buzzer、PWM 和 IR 载波信号。TC0 支持自动重装功能。TC0 溢出时，TC0R 的值自动装入 TC0C。TC0 无绿色模式唤醒功能。

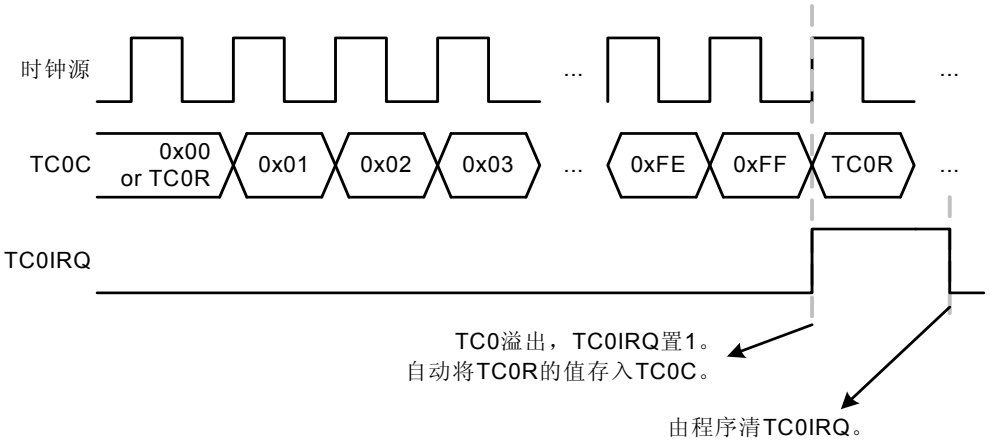
TC0 的主要用途如下：

- **8 位可编程定时器：**根据选择的时钟信号，产生周期中断；
- **中断功能：**TC0 定时器支持中断，当 TC0 溢出时，TC0IRQ 置 1，系统执行中断；
- **占空比/周期可编程控制 PWM 输出：**由 TC0rate 和 TC0R 寄存器控制占空比/周期；PWM 可作为 IR 载波信号发生器，IREN=1 时，TC0 生产 PWM，即使 CPTS[1:0]不为 0；
- **捕捉定时器：**捕捉定时器为自动开启停止定时器，由信号的边沿变换触发，捕捉定时器可以测量 IR 输入载波信号；
- **绿色模式功能：**绿色模式下，TC0 正常工作，但无唤醒功能。



8.3.2 TC0 操作

TC0 定时器由 TC0ENB 控制。当 TC0ENB=0 时，TC0 停止工作；当 TC0ENB=1 时，TC0 开始计数。使能 TC0 之前，先要设定好 TC0 的功能模式，如基本定时器、TC0 中断等。TC0C 溢出（从 0FFH 到 00H）时，TC0IRQ 置 1 以显示溢出状态并由程序清零。在不同的功能模式下，TC0C 不同的值对应不同的操作，若改变 TC0C 的值影响到操作，会导致功能出错。TC0 内置双重缓存器以避免此种状况的发生。在 TC0C 计数的过程中不断的刷新 TC0C，保证将最新的值存入 TC0R（重装缓存器）中，当 TC0 溢出后，TC0R 的值由自动存入 TC0C。进入下一个周期后，TC0 按新的配置工作。使能 TC0 时，自动使能 TC0 的自动重装功能。如果使能 TC0 中断功能（TC0IEN=1），在 TC0 溢出时系统执行中断服务程序，在中断时必须由程序清 TC0IRQ。TC0 可以在普通模式、低速模式和绿色模式下工作。绿色模式下，TC0 虽继续工作，设置 TC0IRQ 和 PWM 输出功能，但不能唤醒系统



TC0 根据不同的时钟源选择不同的应用模式，TC0 的时钟源由 Fcpu（指令周期）和 Fhosc（高速振荡时钟）提供，由 TC0CKS 控制。TC0CKS 选择时钟源来自 Fcpu 或者 Fhosc，当 TC0CKS=0 时，TC0 时钟源来自 Fcpu，可以由 TC0Rate[2:0]选择不同的分频。当 TC0CKS=1 时，TC0 时钟源来自 Fhosc，可以由 TC0Rate[2:0]选择不同的分频。

TC0CKS	TC0rate[2:0]	TC0 Clock	TC0 间隔时间			
			Fhosc=8MHz, Fcpu=Fhosc/2		Fhosc=4MHz, Fcpu=Fhosc/4	
			max. (ms)	Unit (us)	max. (ms)	Unit (us)
0	000b	Fcpu/256	16.384	64	65.536	256
0	001b	Fcpu/128	8.192	32	32.768	128
0	010b	Fcpu/64	4.096	16	16.384	64
0	011b	Fcpu/32	2.048	8	8.192	32
0	100b	Fcpu/16	1.024	4	4.096	16
0	101b	Fcpu/8	0.512	2	2.048	8
0	110b	Fcpu/4	0.256	1	1.024	4
0	111b	Fcpu/2	0.128	0.5	0.512	2
1	000b	Fhosc/1	0.032	0.125	0.064	0.25
1	001b	Fhosc/2	0.064	0.25	0.128	0.5
1	010b	Fhosc/4	0.128	0.5	0.256	1
1	011b	Fhosc/8	0.256	1	0.512	2
1	100b	Fhosc/16	0.512	2	1.024	4

8.3.3 TC0M模式寄存器

模式寄存器 TC0M 控制 TC0 的工作模式，包括 TC0 分频、时钟源和 PWM 功能等。 这些设置必须在使能 TC0 定时器之前完成。

0DAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0M	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS	IRSTS	IREN	CREN
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 0 **CREN**：IR 载波信号输出控制位（IREN 必须使能）。
0 = 禁止，P5.4 保持 IR 空闲状态；
1 = 使能，P5.4 输出 IR 载波信号。

Bit 1 **IREN**：IR 输出功能控制位。
0 = 禁止，P5.4 作为输入/输出口，TC0 为普通定时器或捕捉定时器；
1 = 使能，P5.4 为 IR 输出模式，输出高时为空闲状态，必须在 TC0ENB=1 时由 CREN 开始计数。

Bit 2 **IRSTS**：IR 信号状态指示位（IREN 必须使能）。
0 = 低电平；
1 = 高电平。

Bit 3 **TC0CKS**：TC0 时钟源控制位。
0 = Fcpu；
1 = Fhosc。

Bit [6:4] **TC0RATE[2:0]**：TC0 分频选择位。

TC0RATE [2:0]	TC0CKS = 0	TC0CKS = 1
000	Fcpu / 256	Fosc / 1
001	Fcpu / 128	Fosc / 2
010	Fcpu / 64	Fosc / 4
011	Fcpu / 32	Fosc / 8
100	Fcpu / 16	Fosc / 16
101	Fcpu / 8	
110	Fcpu / 4	
111	Fcpu / 2	

Bit 7 **TC0ENB**：TC0 启动控制位。
0 =关闭；
1 =开启。

8.3.4 TC0C计数寄存器

8 位计数器 TC0C 溢出时，TC0IRQ 置 1 并由程序清零，用来控制 TC0 的中断间隔时间。首先须写入正确的值到 TC0C 和 TC0R 寄存器，并使能 TC0 定时器以保证第一个周期正确。TC0 溢出后，TC0R 的值自动装入 TC0C。

0DBH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0C	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0
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

TC0C 初始值计算公式如下：

TC0C 初始值 = 256 -（TC0 中断间隔时间 * TC0 时钟 rate）

8.3.5 TC0R自动装载寄存器

TC0 内置自动重装功能，TC0R 寄存器存储重装值。TC0C 溢出时，TC0R 的值自动装入 TC0C 中。TC0 定时器工作在计时模式时，要通过修改 TC0R 寄存器来修改 TC0 的间隔时间，而不是通过修改 TC0C 寄存器。在 TC0 定时器溢出后，新的 TC0C 值会被更新，TC0R 会将新的值装载到 TC0C 寄存器中。但在初次设置 TC0M 时，必须要在开启 TC0 定时器前把 TC0C 以及 TC0R 设置成相同的值。

TC0 为双重缓存器结构。若程序对 TC0R 进行了修改，那么修改后的 TC0R 值首先被暂存在 TC0R 的第一个缓存器中，TC0 溢出后，TC0R 的新值就会被存入 TC0R 缓存器中，从而避免 TC0 中断时间出错以及 PWM 误动作。

0CDH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0R	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

TC0R 初始值计算公式如下：

TC0R 初始值 = 256 -（TC0 中断间隔时间 * TC0 时钟 rate）

➢ **例：计算 TC0C 和 TC0R 的值。T0 的中断间隔时间为 10ms，T0 时钟源为 Fcpu = 4MHz/4/=1MHz，T0RATE = 001（Fcpu/128）。**

T0 中断间隔时间为 10ms，T0 时钟 rate＝4MHz/4/128

T0C 初始值 = 256 -（T0 中断间隔时间 * T0 时钟 rate）
= 256 -（10ms * 4MHz / 4/ 128）
= 256 -（10-2 * 4* 106 / 4/128）
= B2H

8.3.6 TC0D PWM占空比寄存器

TC0D 寄存器用来控制 PWM 的占空比。PWM 模式下，TC0R 控制 PWM 的周期，TC0D 控制 PWM 的占空比。TC0C=TC0D 时，PWM 切换为高电平。这样在应用中易于设置 TC0D 以选择合适的 PWM 占空比。

0E8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0D	TC0D7	TC0D6	TC0D5	TC0D4	TC0D3	TC0D2	TC0D1	TC0D0
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

TC0D 初始值的计算方法如下：

TC0D 初始值 = TC0R + （PWM 脉冲高电平宽度周期 / TC0 时钟 rate）

➢ **例：计算 TC0D 的值。1/3 占空比 PWM，TC0 时钟源 Fcpu=8MHz/1=8MHz，TC0RATE=111（Fcpu/2）。TC0R = 97H，TC0 间隔时间=26.25us，PWM 周期频率为 38KHz，1/3 占空比条件下，PWM 低电平的宽度值约为 8.75us。**

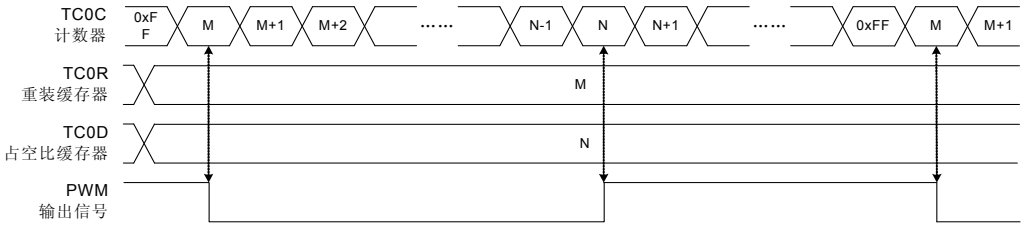
TC0D 初始值 = 97H +（PWM 脉冲高电平宽度值/TC0 时钟 Rate）
= B2H + (8.75us * 8MHz / 1 / 2)
= 97H + 23H
= BAH

普通 IR 信号表，TC0 时钟 rate=4MHz

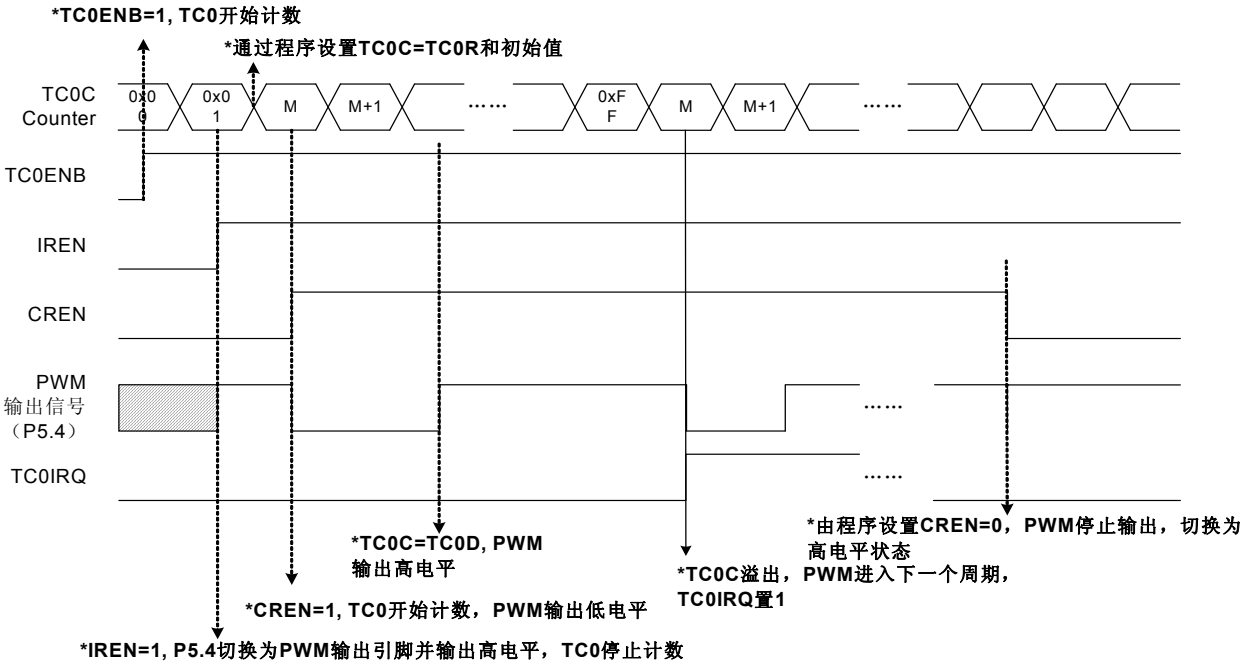
IR Freq. (KHz)	TC0C TC0R		TC0D						Freq.Error Rate
			1/2 duty		1/3 duty		1/4 duty		
	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	
32	131	83	193.50	C1	172.67	AC	162.25	A2	0.00%
36	145	91	200.50	C8	182.00	B6	172.75	AC	0.10%
38	151	97	203.50	CB	186.00	BA	177.25	B1	0.25%
39.2	154	9A	205.00	CD	188.00	BC	179.50	B3	0.04%
40	156	9C	206.00	CE	189.33	BD	181.00	B5	0.00%
56	185	B9	220.50	DC	208.67	D0	202.75	CA	0.60%

8.3.7 脉宽调制（PWM）

TC0 内置 PWM 功能，高电平时为 PWM 空闲状态。TC0 PWM 由 IREN 和 CREN 位控制，从 PWM 输出引脚 P5.4 输出 PWM 信号。IREN=1 时，使能 PWM 功能，P5.4 输出 PWM 信号（空闲状态=高电平）；IREN=0 时，P5.4 返回到上一个 GPIO 模式。PWM 信号由 TC0C、TC0R 和 TC0D 的比较结果产生。CREN=1，且 TC0C 溢出（从 0FFH 到 00H）时，PWM 输出低电平，进入 PWM 初始状态，同时，TC0R 的值加载到 TC0C 中，决定 PWM 的周期和分辨率。然后 TC0C 继续计数，系统开始比较 TC0C 和 TC0D 的值，TC0C=TC0D 时，PWM 输出高电平，TC0C 继续计数直至溢出，此时，整个 PWM 周期完成。TC0R 的值自动载入 TC0C，PWM 输出低电平并进入下一个周期。TC0D 决定低电平占空比的持续时间，TC0R 决定 PWM 的分辨率和周期。



PWM 的分辨率由 TC0R 决定，TC0R 的范围为 00H~0FFH。TC0R=00H 时，PWM 的分辨率诶 1/256；TC0R=80H，PWM 的分辨率为 1/128。TC0D 控制 PWM 低电平的脉宽，TC0C=TC0D 时，PWM 输出高电平，TC0D 的值必须大于 TC0R 的值，否则 PWM 信号一直保持高电平。PWM 输出过程中，TC0 溢出时，TC0IRQ 有效，TC0IEN=1 时，则使能 TC0 中断。但强烈建议小心同时使用 PWM 和 TC0 定时器功能，保证两种功能都能正常工作。PWM 输出引脚与 GPIO 引脚共用，IREN=1 时，该引脚自动切换为 PWM 输出引脚；禁止 PWM 时，IREN 位清零，该引脚自动返回到上一个 GPIO 模式。这样方便实现载波信号的 ON/OFF 操作，而无需控制 TC0ENB 位。



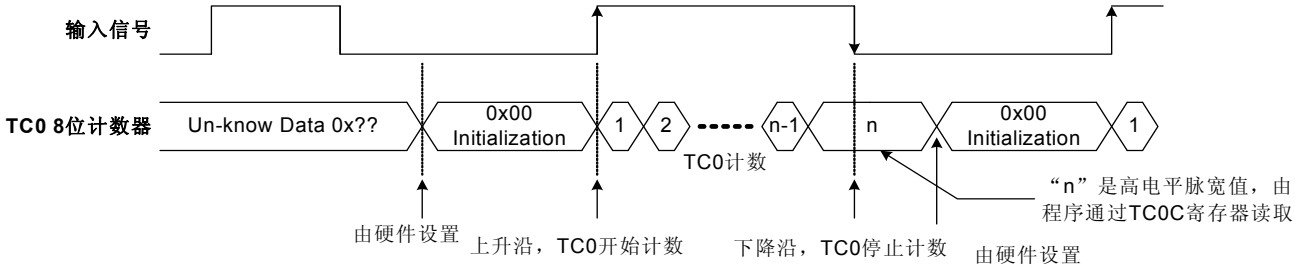
8.3.8 TC0 捕捉定时器

TC0 内置捕捉定时器功能，是一个简单的方法测量高电平脉宽、低电平脉宽和周期。TC0 捕捉定时器由 CPTS[1:0] 控制，CPTS[1:0]=00B 时，禁止 TC0 捕捉定时器；CPTS[1:0]不为 0 时，使能 TC0 捕捉定时器，TC0 计数器由输入信号的边沿变换控制。捕捉定时器的测量信号有两种，由 PEDGS 控制，PEDGS=0 时，捕捉定时器的信号为 P0.0；PEDGS=1 时，捕捉定时器的信号由 IR 接收器提供。

- **CPTS[1:0] = 01：测量高电平脉宽；**
- **CPTS[1:0] = 10：测量低电平脉宽；**
- **CPTS[1:0] = 11：测量输入信号的周期。**

8.3.8.1 测量高电平脉宽

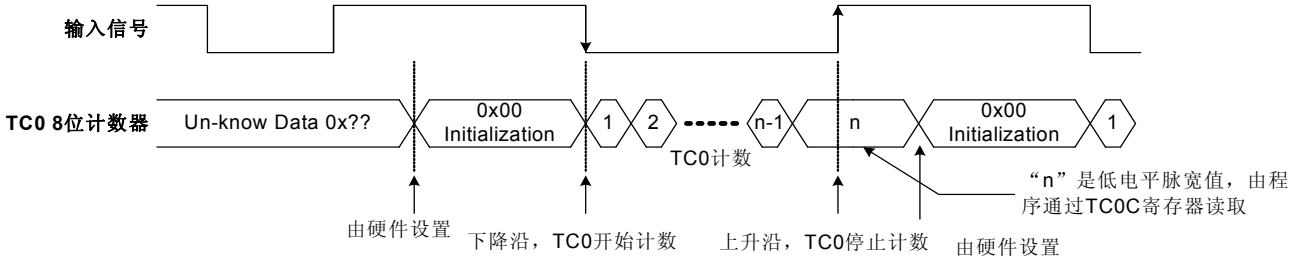
- **TC0ENB = 1. CPTS[1:0] = 01.**



测量高电平脉宽，TC0 在上升沿开始计数，下降沿停止计数。设置 CPTS[1:0]=01，上升沿时捕捉定时器开始测量高电平脉宽，高电平脉宽测量结束后，TC0 停止计数，TC0IRQ 置 1，TC0IEN=1 时执行 TC0 中断。

8.3.8.2 测量低电平脉宽

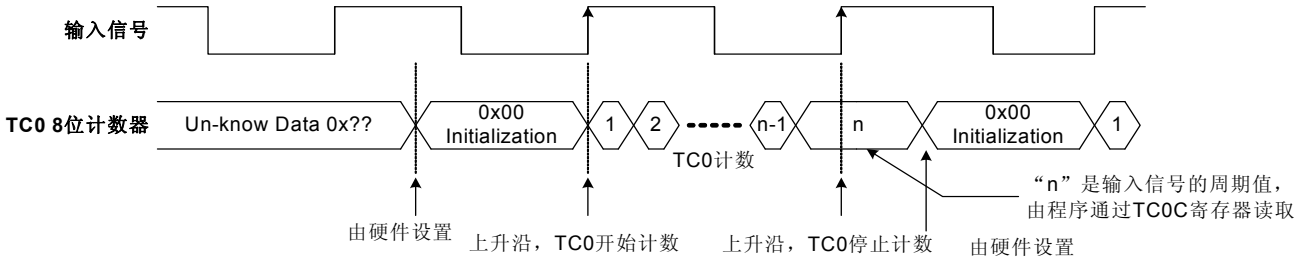
- **TC0ENB = 1. CPTS[1:0] = 10.**



测量低电平脉宽，TC0 在下降沿开始计数，上升沿停止计数。设置 CPTS[1:0]=10，下降沿时捕捉定时器开始测量低电平脉宽，低电平脉宽测量结束后，TC0 停止计数，TC0IRQ 置 1，TC0IEN=1 时执行 TC0 中断。

8.3.8.3 测量输入信号周期

- **TC0ENB = 1. CPTS[1:0] = 11.**



测量输入信号的周期，TC0 在上升沿开始计数，上升沿停止计数。设置 CPTS[1:0]=11，上升沿时捕捉定时器开始测量输入信号的周期，输入信号周期测量结束后，TC0 停止计数，TC0IRQ 置 1，TC0IEN=1 时执行 TC0 中断。

8.3.9 捕捉定时器控制寄存器

D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0rate2	T0rate1	T0rate0	T0TB	CPTS1	CPTS0	IRXEN
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 [2:1] **CPTS[1:0]**：TC0 捕捉定时器功能控制位。捕捉定时器源由 PEDGS 控制。
00 = 禁止；01 = 测量高电平脉宽；10 = 测量低电平脉宽；11 = 测量输入信号周期。

Bit 0 **IRXEN**：IR 接收器控制位。
0 = 禁止，P5.4 为 GPIO 引脚；
1 = 使能，P5.4 为输入引脚。

8.3.10 TC0 定时器操作举例

- **TC0 定时器**

```
; 复位 TC0。
        CLR     TC0M            ; 清 TC0M。

; 设置 TC0 时钟源和 TC0Rate。
        MOV     A, #0nnn0n00b
        B0MOV   TC0M, A

; 设置 TC0C 和 TC0R 获得 TC0 的间隔时间。
        MOV     A, #value       ; TC0C 必须和 TC0R 相等。
        B0MOV   TC0C, A
        B0MOV   TC0R, A

; 清 TC0IRQ。
        B0BCLR  FTC0IRQ

; 使能 TC0 定时器和中断功能。
        B0BSET  FTC0IEN         ; 使能 TC0 中断。
        B0BSET  FTC0ENB         ; 使能 TC0 定时器。
```

- **TC0 PWM**

```
; 复位 TC0。
        CLR     TC0M            ; 清 TC0M。

; 设置 TC0 时钟源和 TC0Rate。
        MOV     A, #0nnn0n00b   ; TC0rate[2:0]。
        B0MOV   TC0M, A

; 设置 TC0C 和 TC0R，获得 PWM 周期。
        MOV     A, #value1      ; TC0C 的值必须和 TC0R 的值相等。
        B0MOV   TC0C, A
        B0MOV   TC0R, A

; 设置 TC0D 寄存器，获取 PWM 占空比。
        MOV     A, #value2
        B0MOV   TC0D,A

; 设置 PWM 载波信号的输出。
        B0BSET  FIREN           ; 设置 IROUT 引脚，输出 PWM 载波信号。
        B0BSET  FCREN           ; 设置 PWM 载波信号的输出。

; 使能 PWM 和 TC0 定时器。
        B0BSET  FTC0ENB         ; 使能 TC0 定时器。
```

- **TC0 捕捉定时器：**

；复位 TC0。

```
        CLR     TC0M            ；清 TC0M。

；设置 TC0 时钟源和 TC0Rate。
        MOV     A, #0nnnm000b   ；“nnn”代表 TC0rate[2:0]，选择 TC0 时钟 rate。
        B0MOV   TC0M, A         ；“m”是 TC0 时钟源控制位。
        MOV     A, #00000mm0b   ；“mm”代表 CPTS[1:0]，选择 TC0 捕捉定时器功能。
        B0MOV   T0M, A          ；CPTS[1:0] = 01b/10b/11b，测量高电平脉宽、低电平脉宽和周期。

；选择捕捉定时器来源。
        B0BCLR  FPEDGS          ；P0.0。
；or
        B0BSET  FPEDGS          ；IR 输入信号。

；清 TC0IRQ。
        B0BCLR  FTC0IRQ

；使能 TC0 捕捉定时器功能和中断功能。
        B0BSET  FTC0IEN         ；使能 TC0 中断功能。
        B0BSET  FTC0ENB         ；使能 TC0 定时器。
```

9 IR发射/接收

9.1 概述

IR 信号由 TC0 定时器产生，IR 输出引脚为 400mA @VSS+0.5V sink 型。TC0M 寄存器的 IREN 位置 1 时，IROUT 引脚为 IR 输出模式。CREN=0 或者系统处于睡眠模式时，IROUT 引脚保持高电平状态。8 位二进制定时器 TC0 产生 IR 信号，占空比/周期由 TC0R 和 TC0D 控制：TC0R 决定 IR 的周期，TC0D 决定 IR 的占空比。使能 IR 输出功能时（CREN=1），IR 输出低电平，TC0C 的初始值为 TC0R，然后开始计数，当 TC0C=TC0D 时，IR 输出高电平。TC0C 溢出（从 0FFH 到 00H）时，IR 输出高电平，操作结束，系统自动装载 TC0R 的值到 TC0C，进入下一个周期。SN8PC22 还内置 IR 接收电路，用于接收 IR 载波信号，IR 二极管必须并联一个 33KΩ的电阻。IR 接收功能由 IRXEN 控制，从 IRIN 引脚接收信号。IRXEN=0 时，禁止 IR 接收功能，IRIN 引脚为 GPIO 模式或者 PWM 模式；IRXEN=1 时，使能 IR 接收功能，IRIN 引脚为 IR 接收引脚。

✸ 注：
1、睡眠模式下，系统自动强制 IROUT 输出高电平。
2、使能 IR 输出前设置 TC0C=TC0R 以保证第一个周期正确。
3、IR 接收模式下，IR 二极管必须并联一个 33KΩ的电阻， IR 发射模式下则不需要。

9.2 IR控制寄存器

0DAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0M	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS	IRSTS	IREN	CREN
读/写	R/W	R/W	R/W	R/W	R/W	R	R/W	R/W
复位后	0	0	0	0	0	0	0	0

Bit 2 **IRSTS**：IR 信号状态指示位（必须使能 IRXEN）。
0 = 低电平；
1 = 高电平。

Bit 1 **IREN**：IR 输出功能控制位。
0 = 禁止，P5.4 为 GPIO 模式，TC0 为普通定时器或捕捉定时器；
1 = 使能，P5.4 为 IR 输出模式，空闲状态为输出高电平，必须设置 TC0ENB=1，然后有 CREN 开始计数。

Bit 0 **CREN**：IR 载波信号输出控制位（必须使能 IREN）。
0 = 禁止，P5.4 保持 IR 空闲状态；
1 = 使能，P5.4 输出 IR 载波信号。

✷ **注：IR 载波信号输出的条件为：IREN=1，CREN=1。若 IREN=0，CREN=1，IROUT 引脚为 GPIO 模式。**

D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0rate2	T0rate1	T0rate0	T0TB	CPTS1	CPTS0	IRXEN
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 0 **IRXEN**：IR 接收控制位。
0 = 禁止，P5.4 为 GPIO 模式；
1 = 使能，P5.4 为输入模式。

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

Bit 1 **IRXO**：IR 接收器输出引脚控制位（必须使能 IRXEN）。
0 = 禁止，P5.0 为 GPIO 模式；
1 = 使能，P5.0 为 IR 接收器输出引脚。

Bit 0 **PEDGS**：触发源控制位。
0 = P0.0；
1 = IR 输入信号。

9.3 IR发射/接收操作流程

➢ 例：IR 发射操作。

```
; 设置 TC0C 和 TC0R，获得 IR 的周期。
        MOV     A, #IRCYCVAL    ;
        MOV     TC0C, A         ;
        MOV     TC0R, A
; 设置 TC0D，获得 IR 的占空比。
        MOV     A, #IRDUTYVAL   ;
        MOV     TC0D, A
; 使能 IR 输出。
        B0BSET  FIREN           ; 设置 IROUT 引脚为 IR 载波信号输出引脚。
        B0BSET  FCREN           ; 设置 IR 载波信号输出。
        B0BSET  FTC0ENB         ; 使能 TC0 定时器。
```

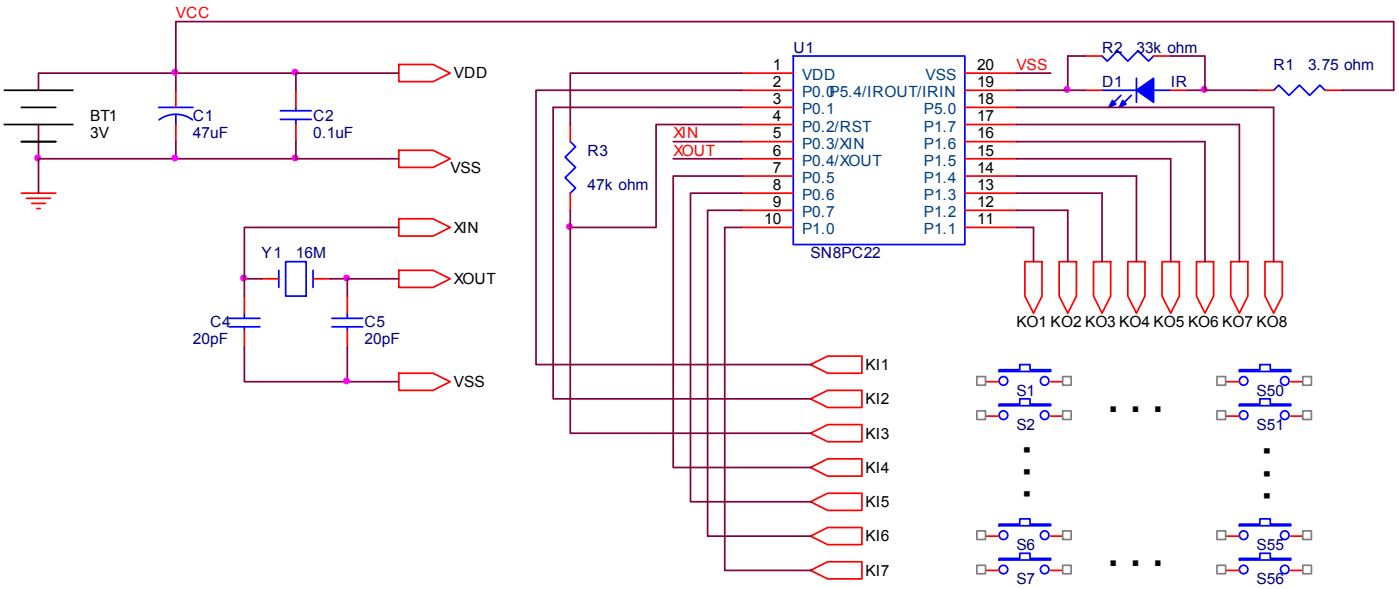
➢ 例：IR 接收操作。

```
; 设置 P5.0 为 GPIO 或 IR 接收器输出模式。
        B0BCLR  FIRXO           ; P5.0 为 GPIO 模式。
; or
        B0BSET  FIRXO           ; P5.0 为 IR 接收器输出模式。
; 使能 IR 输入。
        B0BSET  FIRXEN          ; 设置 IRIN 引脚为 IR 载波输入模式。

; 设置 TC0 捕捉定时器。
        MOV     A, #0nnnm000b   ; “nnn”为 TC0rate[2:0]，选择 TC0 时钟 rate。
        B0MOV   TC0M, A         ; “m”为 TC0 时钟源控制位。

        MOV     A, #00000mm0b   ; “mm”为 CPTS[1:0]，选择 TC0 捕捉定时器功能。
        OR      T0M, A          ; CPTS[1:0 =01b/10b/11b，使能高电平脉宽、低电平脉宽和周期测量。
; 设置 IR 输入信号。
        B0BSET  FPEDGS
; 清 TC0IRQ。
        B0BCLR  FTC0IRQ

; 使能 TC0 捕捉定时器功能和中断功能。
        B0BSET  FTC0IEN         ; 使能 TC0 中断功能。
        B0BSET  FTC0ENB         ; 使能 TC0 定时器。
```

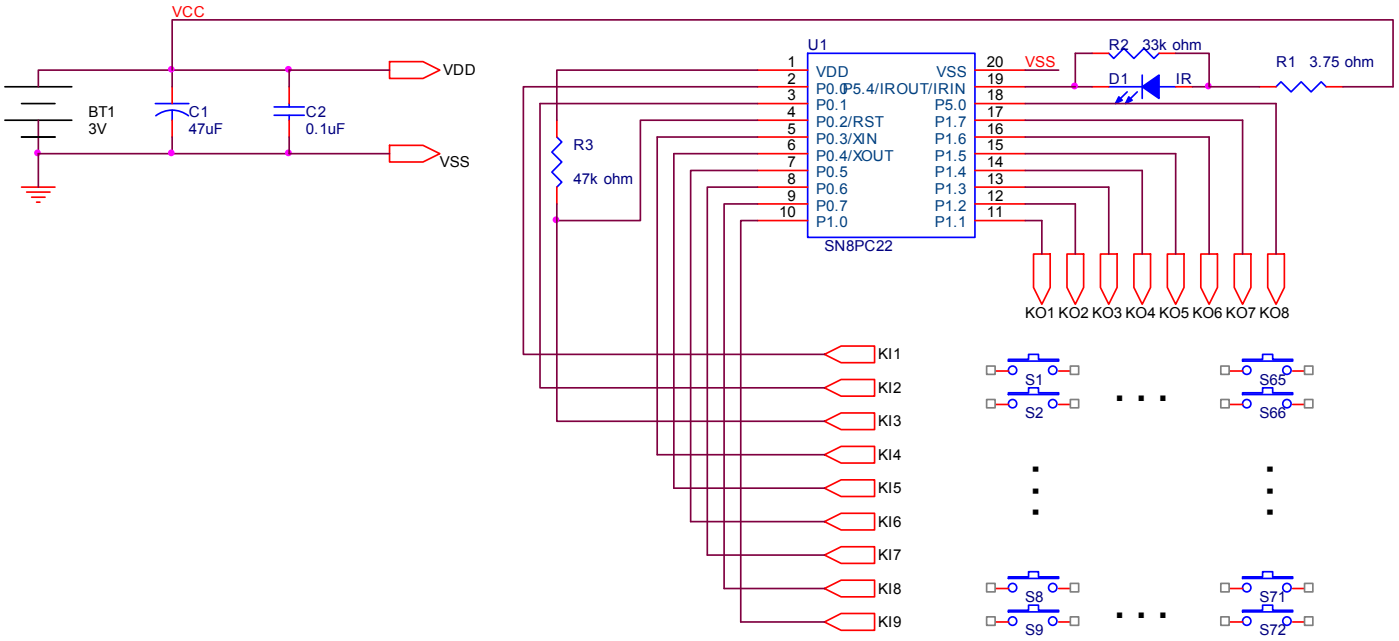
9.4 IR应用电路

IR 灌电流为 400mA @Vss+1.5V，IR 驱动电路的电阻为 3.75 ohm @Vdd=3V，不能低于 3.75 ohm，否则灌电流会超出预期。IR 接收模式下，R2 的值必须为 33KΩ，发射模式下则无此必要。

- **晶振模式，最多 56 个按键：**



- **IHRC_8M 模式，最多 72 个按键：**



10 指令集

Field	指令格式		描述	C	DC	Z	周期
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 仅适用地址是 0x80~0x87 的系统寄存器，如 R、Y、Z...，。）	-	-	-	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，如果产生进位则 C = 1，否则 C = 0。	√	√	√	1
	ADC	M,A	M ← A + M + C，如果产生进位则 C = 1，否则 C = 0。	√	√	√	1+N
	ADD	A,M	A ← A + M，如果产生进位则 C = 1，否则 C = 0。	√	√	√	1
	ADD	M,A	M ← A + M，如果产生进位则 C = 1，否则 C = 0。	√	√	√	1+N
	B0ADD	M,A	M (bank 0) ← M (bank 0) + A，如果产生进位则 C = 1，否则 C = 0。	√	√	√	1+N
	ADD	A,I	A ← A + I，如果产生进位则 C = 1，否则 C = 0。	√	√	√	1
	SBC	A,M	A ← A - M - /C，如果产生借位则 C=0，否则 C=1。	√	√	√	1
	SBC	M,A	M ← A - M - /C，如果产生借位则 C=0，否则 C=1。	√	√	√	1+N
	SUB	A,M	A ← A – M，如果产生借位则 C=0，否则 C=1。	√	√	√	1
	SUB	M,A	M ← A – M，如果产生借位则 C=0，否则 C=1。	√	√	√	1+N
	SUB	A,I	A ← A – I，如果产生借位则 C=0，否则 C=1。	√	√	√	1
LOGIC	AND	A,M	A ← A 与 M。	-	-	√	1
	AND	M,A	M ← A 与 M。	-	-	√	1+N
	AND	A,I	A ← A 与 I。	-	-	√	1
	OR	A,M	A ← A 或 M。	-	-	√	1
	OR	M,A	M ← A 或 M。	-	-	√	1+N
	OR	A,I	A ← A 或 I。	-	-	√	1
	XOR	A,M	A ← A 异或 M。	-	-	√	1
	XOR	M,A	M ← A 异或 M。	-	-	√	1+N
	XOR	A,I	A ← A 异或 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 ← M 带进位右移。	√	-	-	1
	RRCM	M	M ← M 带进位右移。	√	-	-	1+N
	RLC	M	A ← M 带进位左移。	√	-	-	1
	RLCM	M	M ← M 带进位左移。	√	-	-	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	比较，如果相等则跳过下一条指令，C 与 ZF 标志位可能受影响。	√	-	√	1 + S
	CMPRS	A,M	比较，如果相等则跳过下一条指令，C 与 ZF 标志位可能受影响。	√	-	√	1 + S
	INCS	M	A ← M + 1，如果 A = 0，则跳过下一条指令。	-	-	-	1+ S
	INCMS	M	M ← M + 1，如果 M = 0，则跳过下一条指令。	-	-	-	1+N+S
	DECS	M	A ← M - 1，如果 A = 0，则跳过下一条指令。	-	-	-	1+ S
	DECMS	M	M ← M - 1，如果 M = 0，则跳过下一条指令。	-	-	-	1+N+S
	BTS0	M.b	如果 M.b = 0，则跳过下一条指令。	-	-	-	1 + S
	BTS1	M.b	如果 M.b = 1，则跳过下一条指令。	-	-	-	1 + S
	B0BTS0	M.b	如果 M(bank 0).b = 0，则跳过下一条指令。	-	-	-	1 + S
	B0BTS1	M.b	如果 M(bank 0).b = 1，则跳过下一条指令。	-	-	-	1 + S
	JMP	d	跳转指令，PC15/14 ← RomPages1/0，PC13~PC0 ← d。	-	-	-	2
	CALL	d	子程序调用指令，Stack ← PC15~PC0，PC15/14 ← RomPages1/0，PC13~PC0 ← d。	-	-	-	2
MISC	RET		子程序跳出指令，PC ← Stack。	-	-	-	2
	RETI		中断处理程序跳出指令，PC ← Stack，并使能全局中断控制位。	-	-	-	2
	PUSH		保存 ACC 和 PFLAG（不包括 NT0 和 NPD）。	-	-	-	1
	POP		恢复 ACC 和 PFLAG（不包括 NT0 和 NPD）。	√	√	√	1
	NOP		空指令，无特别意义。	-	-	-	1

注： **1. “M”是系统寄存器或 RAM，M 为系统寄存器时 N = 0，否则 N = 1。**
2. 条件跳转指令的条件为真，则 S = 1，否则 S = 0。

11 电气特性

11.1 极限参数

Supply voltage (Vdd)..... - 0.3V ~ 6.0V
Input in voltage (Vin)..... Vss – 0.2V ~ Vdd + 0.2V
Operating ambient temperature (Topr)
SN8PC22P, SN8PC22S, SN8PC22X..... ..... 0°C ~ + 70°C
Storage ambient temperature (Tstor) ..... –40°C ~ + 125°C

11.2 电气特性

- **DC CHARACTERISTIC**

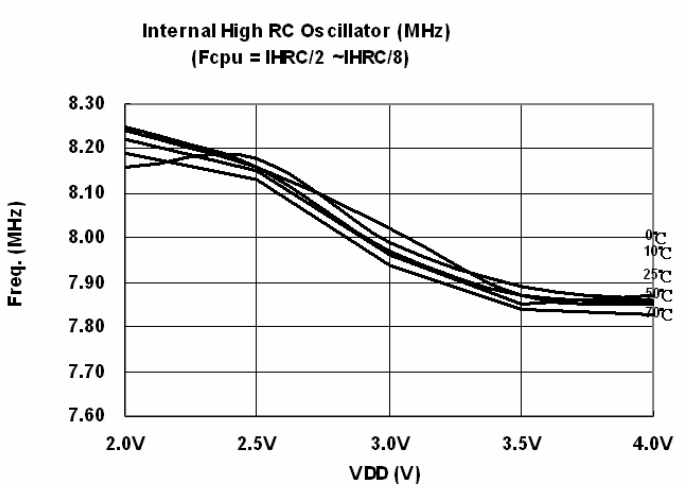
(All of voltages refer to Vss, Vdd = 3.0V, fosc = 4MHz,fcpu=1MHZ,ambient temperature is 25°C unless otherwise note.)

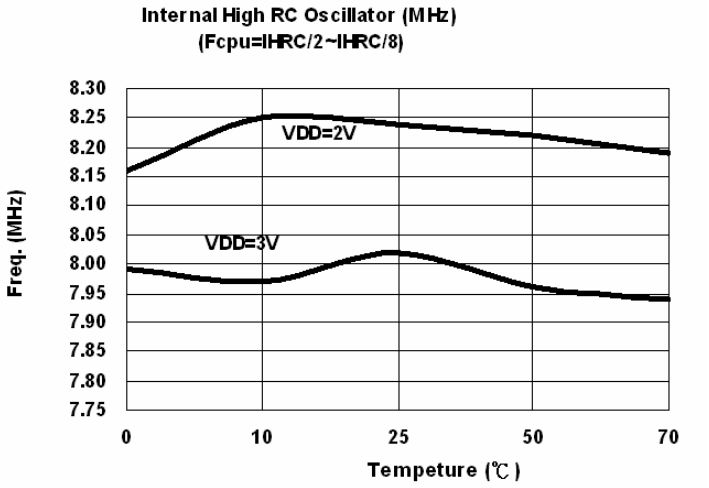
PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode, Vpp = Vdd, 25°C, Fcpu = 2mips.		1.8	3.0	3.6	V
RAM Data Retention voltage	Vdr			1.5	-	-	V
Vdd rise rate	Vpor	Vdd rise rate to ensure internal power-on reset		0.05	-	-	V/ms
Input Low Voltage	ViL1	All input ports		Vss	-	0.3Vdd	V
	ViL2	Xin/Xout		Vss	-	0.4Vdd	V
Input High Voltage	ViH1	All input ports		0.7Vdd	-	Vdd	V
	ViH2	Xin/Xout		0.6Vdd	-	Vdd	V
Reset pin leakage current	Ilekg	Vin = Vdd		-	-	2	uA
I/O port pull-up resistor	Rup	Vin = Vss , Vdd = 3V		100	200	300	KΩ
I/O port input leakage current	Ilekg	Pull-up resistor disable, Vin = Vdd		-	-	2	uA
I/O output source current sink current	IoH	Vop = Vdd – 0.5V		8	10	-	mA
	IoL1	Vop = Vss + 0.5V		8	12	-	
	IoL2	Vop = Vss + 1.5V, IR output pin		300	400	-	mA
INTn trigger pulse width	Tint0	INT0 interrupt request pulse width		2/fcpu	-	-	cycle
Supply Current	Idd1	Run Mode (No loading, Fcpu = Fosc/4)	Vdd= 3V, 4Mhz	-	1	2	mA
	Idd2	Slow Mode (Internal low RC, Stop high clock)	Vdd= 3V, 10Khz	-	5	10	uA
	Idd3	Sleep Mode	Vdd= 3V, 25°C	-	1	2	uA
	Idd4	Green Mode (No loading, Fcpu = Fosc/4 Watchdog Disable)	Vdd= 3V, 4Mhz	-	0.25	0.5	mA
			Vdd=3V, ILRC 10Khz ,	-	3	6	uA
Internal High Oscillator Freq.	Fihrc	Internal Hihg RC (IHRC)	25°C, Vdd= 3V, Fcpu = 1MHz	7.84	8	8.16	Mhz
LVD Voltage	Vdet0	Low voltage reset level.		-	-	1.8	V

“ *” These parameters are for design reference, not tested.

11.3 特性曲线

本章所列的各曲线图仅作设计参考，其中给出的部分数据可能超出了芯片指定的工作范围，为保证芯片的正常工作，请严格参照电气特性说明。





12 开发工具

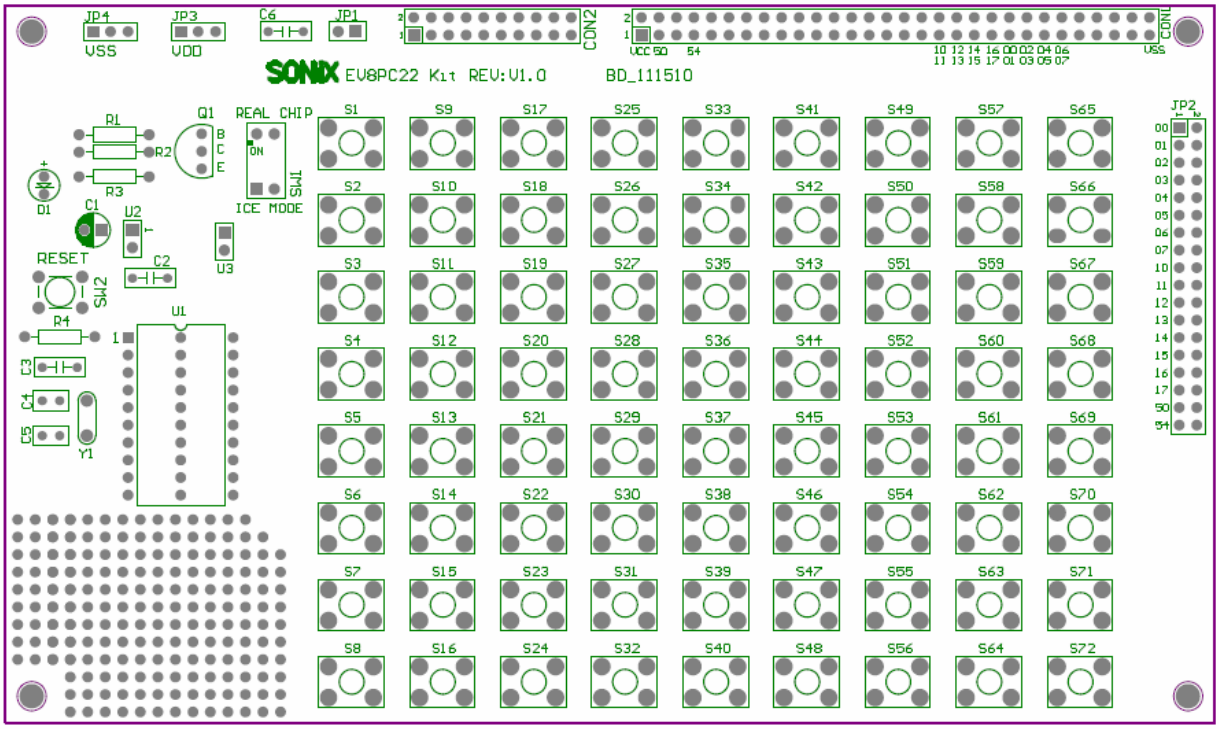
在进行 SN8PC22 的开发时，SONIX 提供 ICE（在线仿真器），IDE（集成开发环境）和 EV-Kit 开发工具。ICE 和 EV-Kit 为外部硬件装置，IDE 有一个友好的用户界面进行韧件开发与仿真。各工具的版本如下所示：

- **ICE：SN8ICE2K Plus 2。**
- **EV-kit：SN8PC22 _EV-kit V1.0。**
- **IDE：SONiX IDE M2IDE_V129 或更晚的版本。**
- **Writer：MP Pro。**

12.1 SN8PC22 EV-KIT

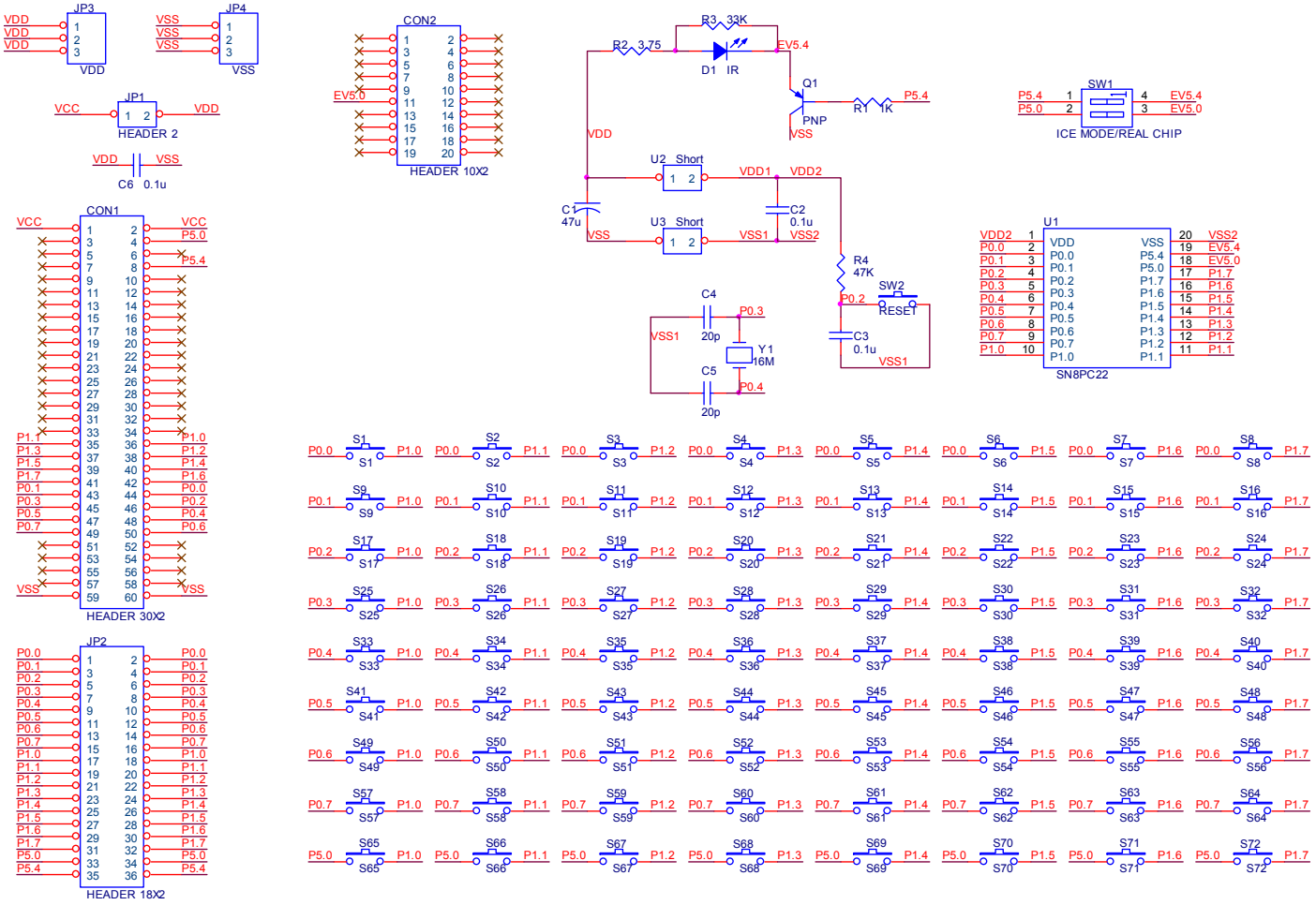
SN8PC22 EV-KIT 包括 ICE 接口，GPIO 接口和 IR 发射/接收驱动模块。

SN8PC22 EV-KIT PCB 外形图如下：



- CON1, CON2：ICE 接口，连接到 SN8ICE2K Plus 2。
- JP1：EV-KIT 接口，连接 VCC 或 VDD，VCC 的电源由 SN8ICE2K Plus 2 提供，VDD 则是 EV-KIT 的电源。
- JP2：GPIO 接口。
- U1：SN8PC22 EV-Chip，仿真 IR 接收，SN8PC22 DIP/SSOP 封装形式接口，用于连接用户目标板。
- U2/U3：内部短路接口，仿真时可以忽略。
- S1~S72：遥控按键。
- D1/R1/R2/R3/Q1：IR 发射/接收驱动短路，用于 ICE 仿真和实际芯片测试。
- SW1：ICE 模式和实际芯片模式控制开关。

SN8PC22 EV-KIT 短路图如下：



12.2 ICE和EV-KIT应用注意事项

SN8PC22 EV-KIT 包括矩阵按键和 IR 发射/接收驱动短路模块。

- IR 发射/接收驱动短路模块包括两种：一种用 ICE 仿真驱动 IR 载波信号，此时 SW1 必须切换为 ICE 模式；另一种是 SN8PC22 实际芯片测试，SW1 必须切换为实际芯片模式，并与仿真器断开连接。
- EV-KIT 是一个类似的遥控器，R2 的阻值必须大于 3.75Ω，灌电流为 400mA @Vdd=3V，R3 的阻值为 33KΩ。

13 OTP烧录

13.1 烧录引脚配置

SN8PC22 的烧录引脚信息			
单片机名称		SN8PC22P/S/X	
MP PRO Writer 接口		OTP IC 引脚配置	
JP3 编号	JP3 名称	IC 引脚编号	IC 引脚名称
1	VDD	1	VDD
2	GND	20	VSS
3	CLK	12	P1.2
4	CE	-	-
5	PGM	10	P1.0
6	OE	13	P1.3
7	D1	-	-
8	D0	-	-
9	D3	-	-
10	D2	-	-
11	D5	-	-
12	D4	-	-
13	D7	-	-
14	D6	-	-
15	VDD	-	-
16	VPP	4	RST
17	HLS	-	-
18	RST	-	-
19	-	-	-
20	ALSB/PDB	11	P1.1

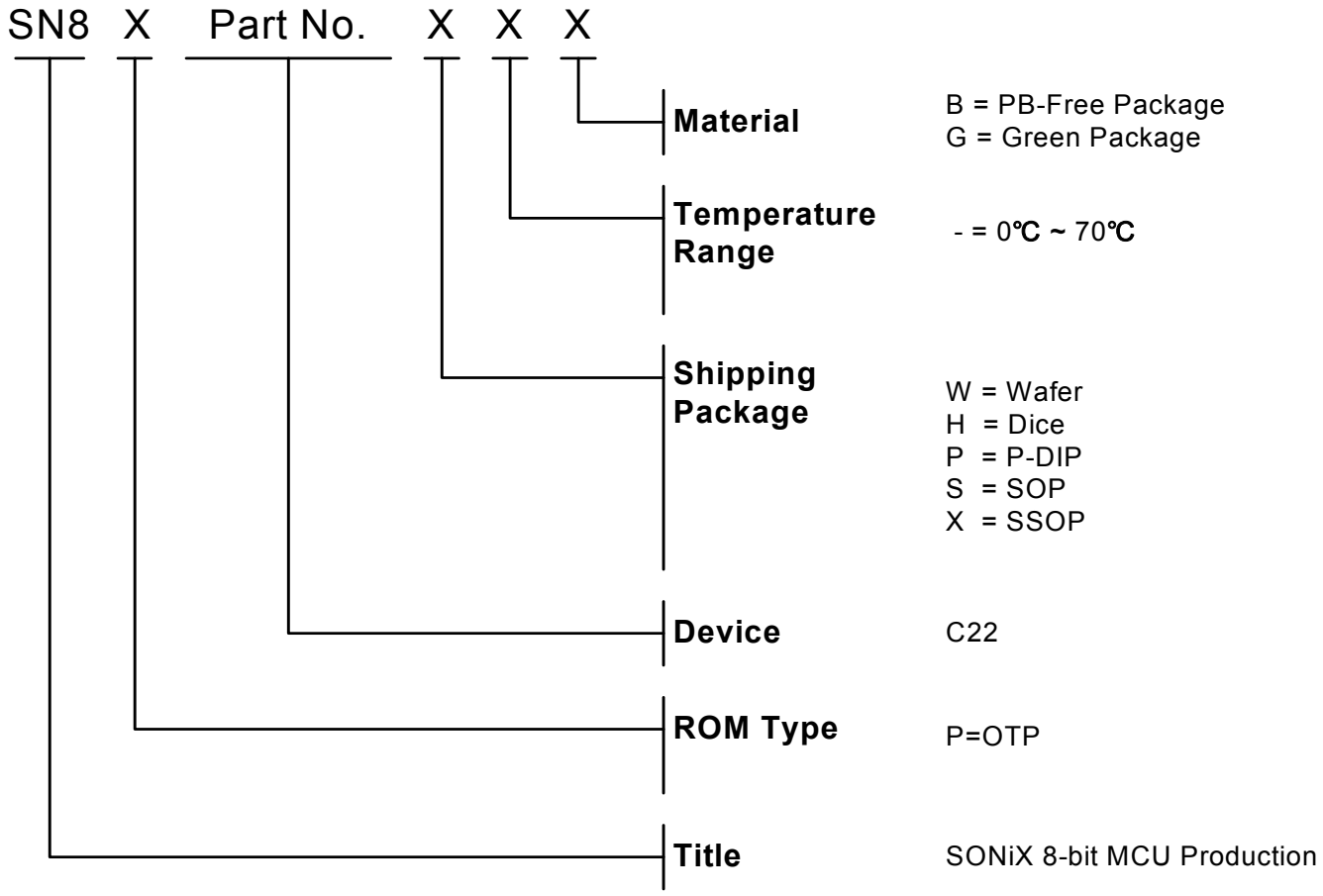
✷ 注：
1、SN8PC22 只支持 MP PRO Writer 的烧录，而不支持 MPIII Writer-LV 的烧录。
2、请参考 MP-PRO Writer 使用手册。

14 单片机正印命名规则

14.1 概述

SONiX 8 位单片机产品具有多种型号，本章将给出所有 8 位单片机分类命名规则，适用于空片 OTP 型单片机。

14.2 单片机正印说明



✷ 注：SN8PC22 没有-40ºC~85ºC 的温度范围。

14.3 命名举例

- **Wafer, Dice:**

单片机名称	ROM 类型	器件（Device）	封装形式	温度范围	封装材料
S8PC22W	OTP	C22	Wafer	0℃~70℃	-
SN8PC22H	OTP	C22	Dice	0℃~70℃	-

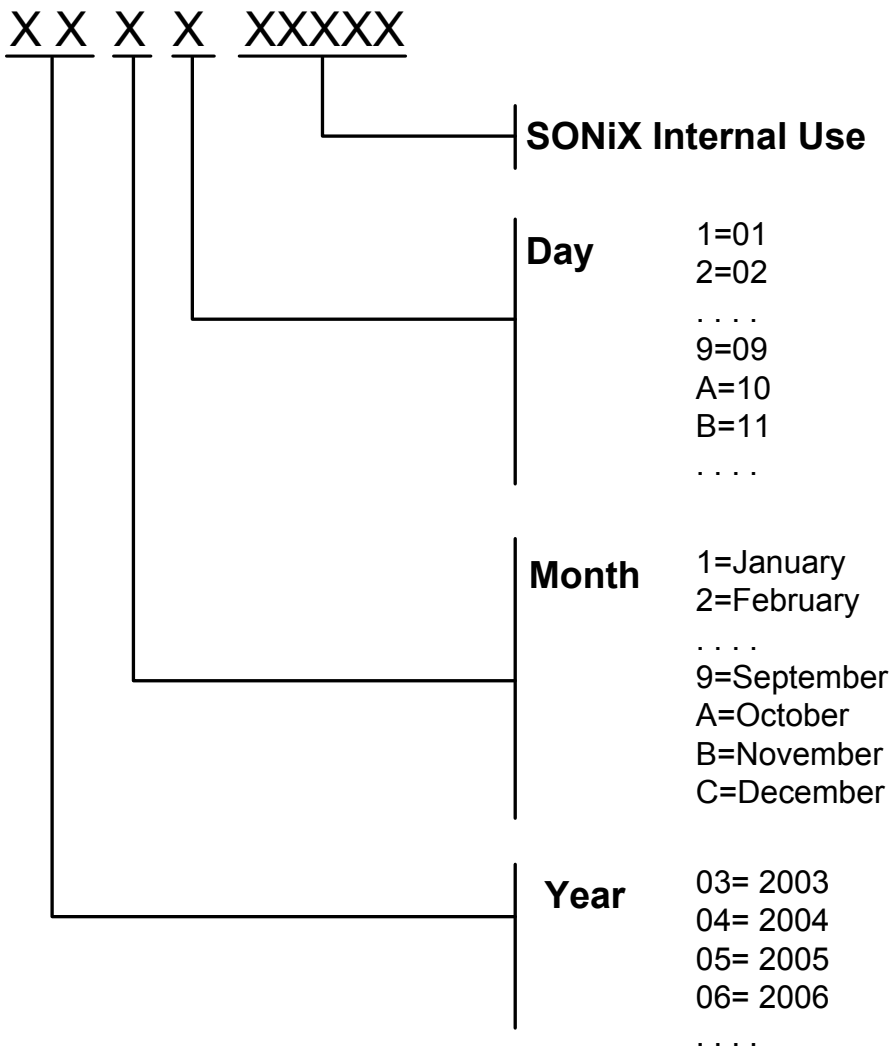
- 绿色封装：

单片机名称	ROM 类型	器件（Device）	封装形式	温度范围	封装材料
SN8PC22PG	OTP	C22	P-DIP	0℃~70℃	绿色封装
SN8PC22SG	OTP	C22	SOP	0℃~70℃	绿色封装
SN8PC22XG	OTP	C22	SSOP	0℃~70℃	绿色封装

- 无铅封装：

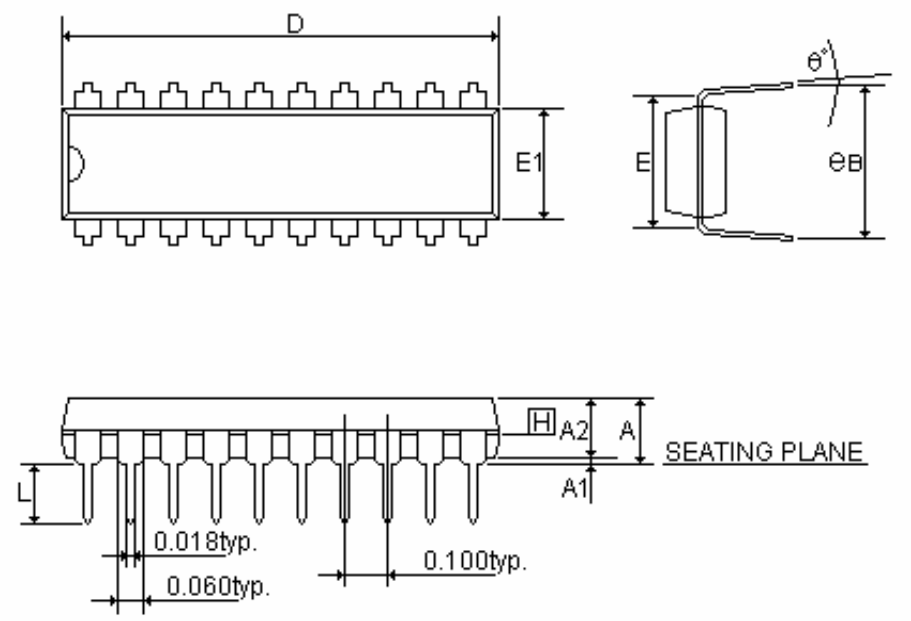
单片机名称	ROM 类型	器件（Device）	封装形式	温度范围	封装材料
SN8PC22PB	OTP	C22	P-DIP	0℃~70℃	无铅封装
SN8PC22SB	OTP	C22	SOP	0℃~70℃	无铅封装
SN8PC22XB	OTP	C22	SSOP	0℃~70℃	无铅封装

14.4 日期码规则



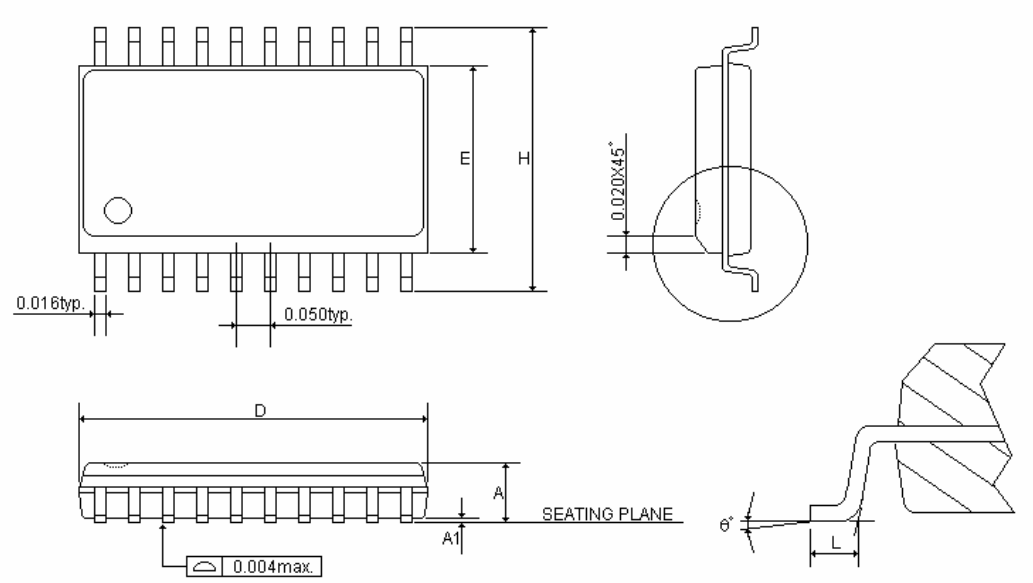
15 封装信息

15.1 P-DIP 20 PIN



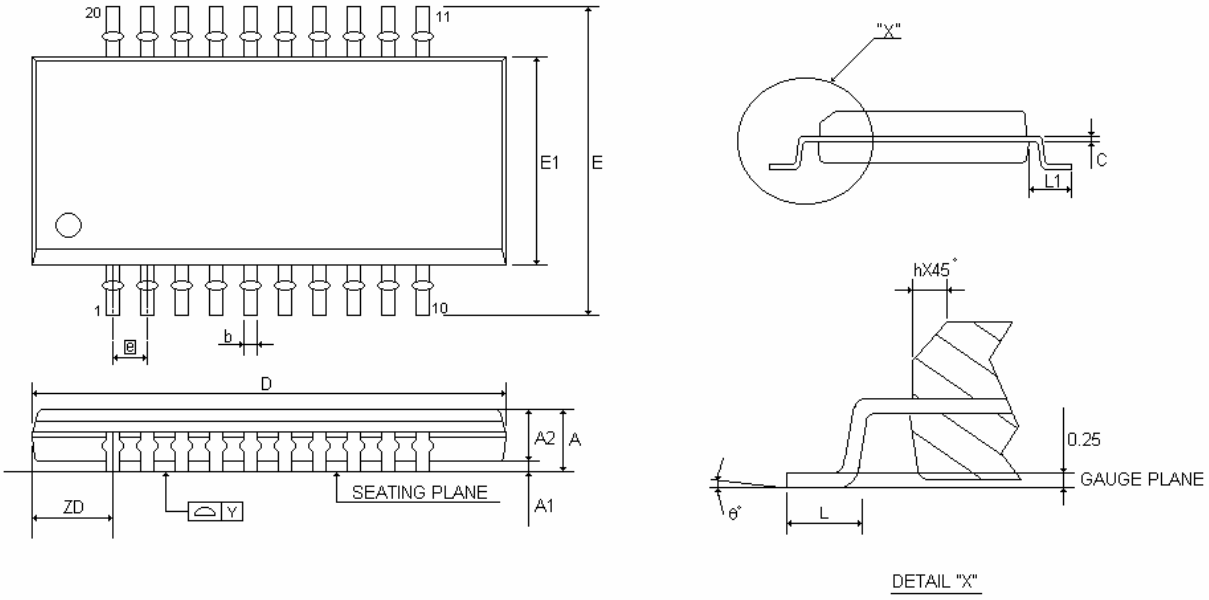
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.125	0.130	0.135	3.175	3.302	3.429
D	0.980	1.030	1.060	24.892	26.162	26.924
E		0.300			7.620	
E1	0.245	0.250	0.255	6.223	6.350	6.477
L	0.115	0.130	0.150	2.921	3.302	3.810
e B	0.335	0.355	0.375	8.509	9.017	9.525
θ°	0°	7°	15°	0°	7°	15°

15.2 SOP 20 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	0.496	0.502	0.508	12.598	12.751	12.903
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°

15.3 SSOP 20 PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.053	0.063	0.069	1.350	1.600	1.750
A1	0.004	0.006	0.010	0.100	0.150	0.250
A2	-	-	0.059	-	-	1.500
b	0.008	0.010	0.012	0.200	0.254	0.300
c	0.007	0.008	0.010	0.180	0.203	0.250
D	0.337	0.341	0.344	8.560	8.660	8.740
E	0.228	0.236	0.244	5.800	6.000	6.200
E1	0.150	0.154	0.157	3.800	3.900	4.000
[e]		0.025			0.635	
h	0.010	0.017	0.020	0.250	0.420	0.500
L	0.016	0.025	0.050	0.400	0.635	1.270
L1	0.039	0.041	0.043	1.000	1.050	1.100
ZD		0.059			1.500	
Y	-	-	0.004	-	-	0.100
θ°	0°	-	8°	0°	-	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作进一步说明的权利。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。

总公司：
地址：台湾新竹县竹北市台元街 36 号 10 楼之一
电话：886-3-5600-888
传真：886-3-5600-889
台北办事处：
地址：台北市松德路 171 号 15 楼之 2
电话：886-2-2759 1980
传真：886-2-2759 8180
香港办事处：
地址：香港九龙湾宏开道 8 号其士商业中心 15 楼 1519 室
电话：852-2723 8086
传真：852-2723 9179
松翰科技（深圳）有限公司
地址：深圳市南山区高新技术产业园南区 T2-B 栋 2 层
电话：86-755-2671 9666
传真：86-755-2671 9786
技术支持：
Sn8fae@SONiX.com.tw