SN8P1602B

用户手册

SONiX 8 位单片机

修正记录

版本号	日期	说明
VER1.0	2003 年 9 月	V1.0 第一版
VER1.1	2003 年 9 月	1. 删除校准表
		2. 删除 PCB 布线注意事项
		3. 调整编码选项说明
		4. 增加 PEDGE 寄存器的说明
		5. 修正 INTRQ 寄存器的说明
		6. 改变工作电压范围“2.2V~5.5V”，“2.4V~5.5V”(Fosc=3.579545MHz)，环境温度为 25°C
VER1.2	2003 年 12 月	1. 在 P1.4 的说明中增加“没有上拉电阻”。
		2. 更正 P00G[1:0](PEDGE 寄存器)的解说。 将“01＝上升沿”改为“01＝下降沿”； 将“10＝下降沿”改为“10＝上升沿”。
		3. 增加 MASK / OTP 关系表。

目　录

1 产品简介

1.1 概述

SN8P1602B 8 位微控制器采用 CMOS 技术，结构独特，具有低功耗、高性能的特点。

SN8P1602B 的 IC 结构设计一流，包括一个大容量的程序存储器（1K words），48-bytes 的数据存储器，一个 8 位定时/计数器(TC0)，一个看门狗定时器，两个中断源(TC0、INT0)，以及 4 层堆栈缓存区。此外，用户可自行选择芯片的振荡形式，有四种不同的外部振荡结构提供系统时钟：高/低速晶体振荡器/陶瓷谐振器和 RC 振荡器等。SN8P1602B 还可以通过程序设定内部 RC 振荡器作为低速模式时钟源。

1.1.1 SN8P1602/SN8P1603/SN8P1602A/SN8P1602B 比较表

项目	SN8P1602B	SN8P1602A	SN8P1602	SN8P1603
睡眠模式电流(3V)	<1uA	3～4 uA	3～4uA	70 uA
上拉电阻	是	是	—	—
上电复位/掉电复位	优秀	优秀	—	好
看门狗时钟源	高速时钟/内部 RC	高速时钟/内部 RC	高速时钟	高速时钟
内部 RC 时钟的看门狗时钟源 总是处于开放状态	是	是	—	—
绿色模式(Green Mode)	是	是	—	—
P0.0 中断边沿	下降/上升双边沿	下降/上升双边沿	下降沿	下降沿
Port1 唤醒功能	电平变化	电平变化	低电平	低电平
TC0 计数器	是	是	—	—
外部复位建议值	20K/0.1 uF	20K/0.33uF	20K/0.1 uF	20K/0.1 uF
上电延迟（4Mhz）	～200ms	～200ms	~70ms	~70ms
LVD	1.8V 总处于开放状态	1.8V 总处于开放状态	2.4V ON/OFF	2.4V ON/OFF

1.2 特性

- **存储器配置**
  OTP ROM：1K*16-bit
  RAM：48*8-bit

- **输入/输出引脚配置**
  单向输入：P0
  双向输入输出：P1, P2
  唤醒功能：P0, P1
  上拉电阻：P0,P1,P2
  外部中断：P0

- **一个内置看门狗定时器**
- **一个 8 位定时/计数器**

- **57 条功能强大的指令**
  每条指令周期为四个时钟周期
  所有指令为单字长
  绝大部分指令只需 1 个指令周期
  指令的最长周期位 2 个指令周期
  JMP 指令可在整个 ROM 区执行
  查表功能(MOVC)寻址整个 ROM 区

- **两个中断源(Interrupt)**
  一个内部中断：TC0
  一个外部中断：INT0

- **4 层堆栈缓存器(Stack)**

- **双重时钟系统提供 4 种操作模式**
  外部高速时钟：RC　最大 10 MHz
  外部高速时钟：晶体　最大 16MHz
  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/低速时钟停止
  绿色模式：由定时器周期性唤醒

- **封装**
  P-DIP18
  SOP18
  SSOP20

1.3 产品特性表

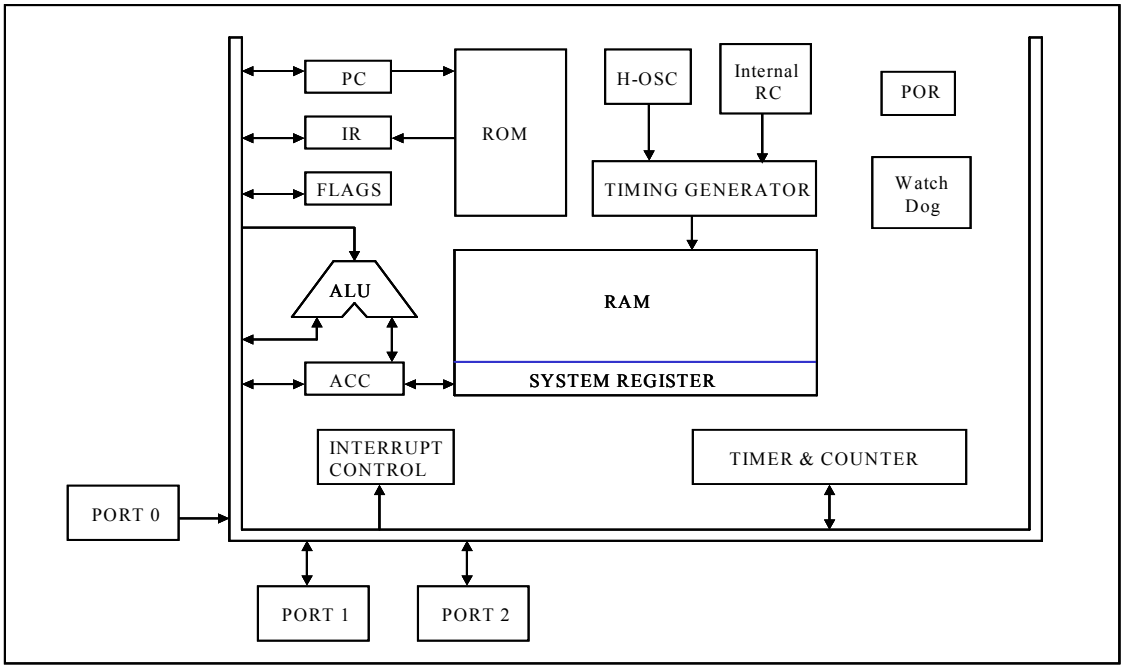
芯片	ROM	RAM	堆栈	定时器		I/O	绿色模式	PWM	唤醒功能引脚数目	封装
				TC0	TC1			Buzzer		
SN8P1602B	1K*16	48	4	V	-	14	V	-	6	DIP18/SOP18/SSOP20

1.4 MASK / OTP 关系表

MASK 版本	封装格式	OTP 版本	编译宣告
SN8A1602B	DIP18/SOP18/SSOP10	SN8P1602B	CHIP SN8P1602B

1.5 系统框图

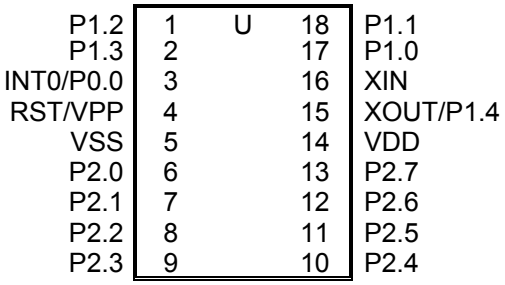
➢ **SN8P1602B**



1.6 引脚配置

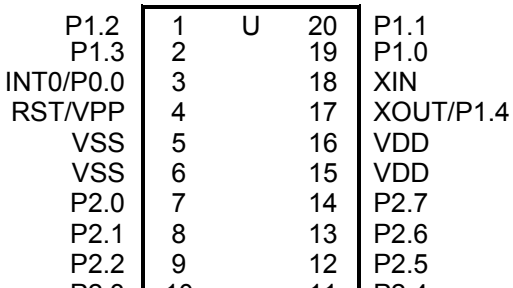
OTP

SN8P1602BP (P-DIP 18 pins)
SN8P1602BS (SOP 18 pins)



SN8P1602BP
SN8P1602BS

SN8P1602BX (SSOP 20 pins)



SN8P1602BX

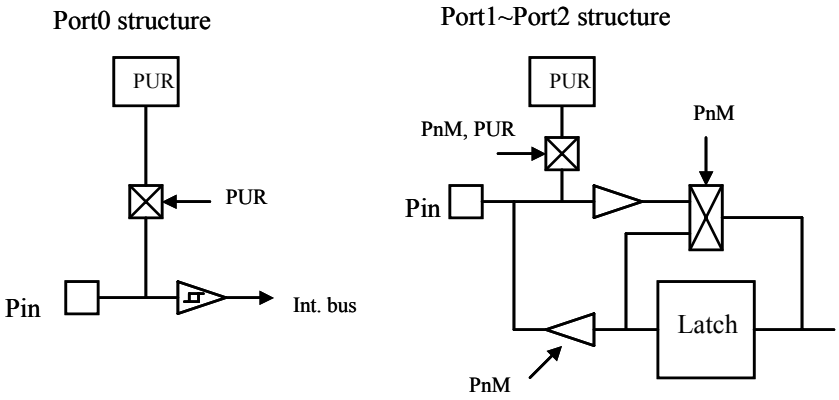
1.7 引脚说明

➢ **SN8P1602B**

引脚名	类型	说　　明
VDD, VSS	P	电源输入引脚， 建议在 VDD 与 VSS 之间接一个 0.1μF 的旁路电容
RST/VPP	I,P	RST：系统复位输入端，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	外部振荡器输入端，RC 模式的输入端
XOUT/P1.4	I/O	外部振荡器输出端，在 RC 模式中为基本输入输出功能引脚（P1.4），无上拉电阻。
P0.0/INT0	I	Port 0.0/INT0 触发引脚（施密特结构），内置上拉电阻。
P1.0~P1.4	I/O	输入/输出端，内置上拉电阻。
P2.0~P2.7	I/O	输入/输出端，内置上拉电阻。

1.8 引脚电路图

➢ **SN8P1602B**



➢ **注：所有锁存输出电路均为图腾柱结构**

2 编译选项表（Code Option）

SN8P1602B

编码选项	内容	功能说明
High_Clk	RC	外部振荡器采用 RC 振荡电路
	32K X'tal	外部振荡器采用低频振荡器(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 3.58MHz)
High_Clk / 2	Enable	外部高速时钟 2 分频, Fosc = 高速时钟 / 2
	Disable	Fosc = 高速时钟
OSG	Enable	使能振荡器保护功能，提高抗干扰性能。
	Disable	禁止振荡器保护功能。
Watch_Dog	Enable	使能看门狗定时器功能
	Disable	禁止看门狗定时器功能
Low Power	Enable	使能低功耗功能以节省工作电流。
	Disable	禁止低功耗功能。
Noise Filter	Enable	使能噪音滤除功能以提高抗干扰性能
	Disable	禁止噪音滤除功能
Security	Enable	允许 ROM 程式代码加密
	Disable	禁止 ROM 程式代码加密
INT_16K_RC	Always_ON	当强迫看门狗定时器的时钟源来自内部 16K RC 时，看门狗定时器一直处于是你状态（即使在省电模式和绿色模式下）。
	BY_CPUM	由寄存器 CPUM 控制内部 RC(16K，3V)时钟的使能与否。

表 2－1　SN8P1602B 编译选项表

该表只是一个设计向导，并不能用作测试或保证。只是提供在特定的范围内正确的操作信息。

编译选项（Code Option）		最低工作电压	
Enable	**Disable**	**4 MHz**	**16 MHz**
-	**Noise Filter/Low Power/OSG**	**2.2V**	**2.8V**
Noise Filter	**Low Power/OSG**	**2.2V**	**3.5V**
Low Power	**Noise Filter/OSG**	**2.2V**	**3.8V**
OSG	**Noise Filter/Low Power**	**2.2V**	**2.9V**

表 2－2　SN8P1602B 最小工作电压和编译选项及时钟频率的关系

注：

- **在高干扰环境下，建议使能"Noise Filter"，"OSG"选项及禁止"Low Power"选项。**
- **使能"Noise Filter"/"OSG"/"Low Power"选项会增加最低工作电压值。**
- **在普通模式下，使能"Low Power"选项会降低工作电流。**
- **如果在"High_Clk"选项中选择"32K X'tal"，则编译器会强制使能"OSG"。**
- **如果在"High_Clk"选项中选择"RC"，则编译器会强制使能"High_Clk /2"。**

3 存储器

3.1 程序存储器（ROM）

3.1.1 概述

SN8P1602B 提供 1024 * 16 位程序存储器， 通过 10 位的 PC (程序计数器)对程序存储器进行寻址，或由专用寄存器（R、Y、Z）对 ROM 进行查表访问。

- 1－word 复位向量入口地址
- 1－word 中断向量入口地址
- 1K words 通用存储区域
- 5－word 的保留区域

所有的程序存储器被分为 3 个代码区：0000H~0003H(复位向量区)，0004H~0007H(系统保留区)，0008H~0FFEH(中断向量区和通用存储区)。0008H 是中断向量的入口地址。

	ROM	
0000H	复位向量	程序开始
0001H	通用存储区	跳转到用户程序
0002H		跳转到用户程序
0003H		跳转到用户程序
0004H	系统保留	
0005H		
0006H		
0007H		
0008H	中断向量	中断入口地址
0009H	通用存储区	用户程序区
.		
.		
000FH		
0010H		
0011H		
.		
.		
03FEH		程序结束
03FFH	系统保留	

3.1.2 复位向量地址(0000H)

上电复位或看门狗溢出复位后，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：**

```
CHIP SN8P1602B
        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
```

3.1.3 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存器中并跳转至 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。下面的例子给出了如何在程序中定义中断向量。

➲ **例 1：**

```
CHIP SN8P1602B
.DATA       PFLAGBUF
.CODE
            ORG     0           ; 0000H
            JMP     START       ; 跳转到用户程序
            .                   ; 0004H ~ 0007H 保留
            ORG     8           ; 中断服务程序
            B0XCH   A, ACCBUF   ; B0XCH 指令不会改变 PFLAG 的值
            B0MOV   A, PFLAG
            B0MOV   PFLAGBUF, A ; 保存 PFLAG 的值
            .
            B0MOV   A, PFLAGBUF
            B0MOV   PFLAG, A    ; 恢复 PFLAG 的值
            B0XCH   A, ACCBUF   ; B0XCH 指令不会改变 PFLAG 的值
            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：**

```
CHIP SN8P1602B
.DATA       PFLAGBUF
.CODE
            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:                         ;中断服务程序起始地址
            B0XCH   A, ACCBUF   ; B0XCH 指令不会改变 PFLAG 的值
            B0MOV   A, PFLAG
            B0MOV   PFLAGBUF, A ; 保存 PFLAG 的值
            .
            B0MOV   A, PFLAGBUF
            B0MOV   PFLAG, A    ; 恢复 PFLAG 的值
            B0XCH   A, ACCBUF   ; B0XCH 指令不会改变 PFLAG 的值
            RETI                ; 中断服务程序结束

            ENDP                ; 程序结束
```

➢ **注意：从上面的程序中可以得出 SONIX 的主要编程规则：**

1. **地址 0000H 处的 “JMP”指令使程序跳转至通用 ROM 区，0004H~0007H 由系统保留，用户必须跳过 0004H~0007H。**
2. **0004H~0007H 是系统保留区，用户可以跳过该区域。我们建议用户在对 ROM 区作 CHECKSUM 时也要跳过该区域，详见下面的 Checksum 计算章节。**

3.1.4 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 0004 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                   ; 递增 YZ，地址加 1
                JMP     @B                  ;
                JMP     Y_ADD_1             ;
END_CHECK:
                MOV     A,END_ADDR1         ; 判断是否计算到代码结束位置
                CMPRS   A,Z                 ;
                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                         ; 不是 0004H 则返回继续计算
                INCMS   Z                   ; 是到 0004H 位置则 Z 加 4，跳过 0004H—0007H
                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:
```

3.1.5 通用程序存储区

位于 ROM 中 0009H~03FEH 的 1017-words 作为通用程序存储区，这一区域主要用来存储指令的操作代码和查表数据。SN8P1602B 中包括通过程序计数器实现的跳转表程序和通过寄存器（R、Y、Z）实现的查表程序。

在对程序计数器 PC 进行操作导致 PCL 溢出时，PCH 不会自动加 1，因此在跳转表程序和查表程序中，计数器不会自动跳过边界，当 PCL 发生溢位（从 0FFH 增至 000H）时，用户必须自行将 PCH 调整为 PCH+1。

3.1.6 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 “table_1”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), → 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。**

➢ **注: 因为程序计数器 PC 只有 12 位, X 寄存器在查表的时候实际是没有用处的，用户可以省略“B0MOV X, #TABLE1$H”. SONiX ICE 能够支持更大的程序寻址能力，所以必须保证 X 寄存器为 0，以避免查表中出现不可预知的错误。**

➲ **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
```

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。下面的例子给出了详细操作步骤：

➲ **例: 执行指令 B0ADD/ADD 增加 Y、Z**

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

                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，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           ; “
```

3.1.7 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。

执行“ADD PCL, A”如果有进位发生，结果并不会影响 PCH 寄存器。用户必须检查跳转表是否跨越了 ROM 的页边界或由 SONIX 编译软件产生的列表文件。我们建议用户把跳转表放在程序的开头（xx00H）以避免在编辑程序时出现错误。

➲ 例：

```
        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
```

在下面的例子中， 跳转表格从 0x00FD 开始，执行“B0ADD PCL A”时，如果 ACC = 0 或 1，跳转表指向正确的地址，但如果 ACC 大于 1，因为 PCH 不能自动加 1，程序就会出错：可以看到当 ACC=2 时，PCL=0，而 PCH 仍然保持为 0，则新的程序计数器 PC 将指向错误的地址 0x0000，程序失效。 因此，检查跳转表格是否跨越 ROM 的页边界(xxFFH to xx00H)非常重要。良好的编程风格是将跳转表格放在 ROM 的开始边界（如 0100H）。

➲ **例：如果跳转表格跨越 ROM 页边界，将引起程序错误：**

```
ROM Address
    .           .
    .           .
0X00FD      B0ADD   PCL, A      ; PCL = PCL + ACC，PCH 的值不能改变
0X00FE      JMP     A0POINT     ; ACC = 0
0X00FF      JMP     A1POINT     ; ACC = 1
0X0100      JMP     A2POINT     ; ACC = 2  ←跳转表跨越边界
0X0101      JMP     A3POINT     ; ACC = 3
    .           .
```

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 00FDH 到 0101H，那么宏指令“@JMP_A”将使跳转表格从 0100h 开始

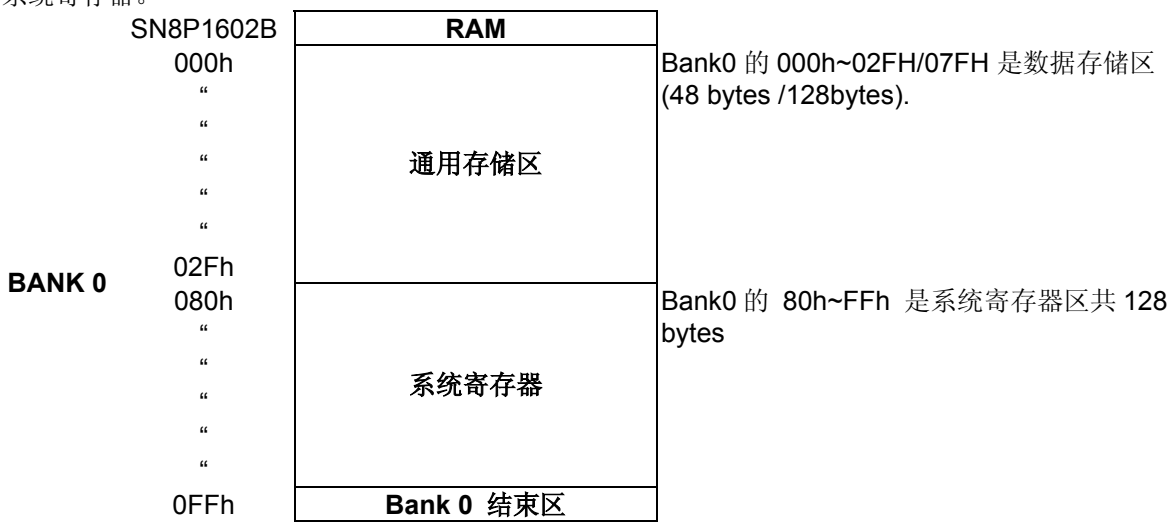
3.2 数据存储器（RAM）

3.2.1 概述

SN8P1602B 在内置数据存储区内有 48 bytes 的通用数据存储区。

- 48*8-bit RAM

存储器分为通用存储区、系统寄存器区两部分，把前面的 48 bytes 作为通用存储区域，把后面的 128 bytes 保留给系统寄存器。



3.2.2 工作寄存器

RAM Bank0 中的 82H~84H 是系统专用寄存器，可用作一般的工作缓存器或用来访问 ROM 和 RAM 中的数据。例如，所有的 ROM 列表可以通过 Y、Z 查找，RAM 可由 Y 和 Z 寄存器间接访问。

3.2.2.1 Y,Z 寄存器

Y 和 Z 寄存器是 8 位的缓存器，有三个主要的功能：

- 工作寄存器
- 寄存器@YZ，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	0	0	0	0	0	0	0	0

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	0	0	0	0	0	0	0	0

➲ **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
        .
```

3.2.2.2 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

（执行 MOVC 指令后，ROM 数据的高字节存入 R 寄存器中，低字节存入累加器 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	0	0	0	0	0	0	0	0

➢ **注：请参考使用 R 寄存器的查表功能“查表说明”。**

3.2.3 程序状态字（PFLAG）

程序状态字包括进位标志(C)，辅助进位标志(DC)和零标志(Z)，如果运算结果为零或者有进位、借位发生，PFLAG寄存器中相应位将受到影响。

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

3.2.3.1 复位/唤醒标志

NT0	NPD	说明
0	0	在睡眠模式期间，由看门狗唤醒。 只有使能“INT_16K_RC”后，这项功能才是有效的
0	1	在正常/低速/绿色模式下，看门狗溢出
1	0	在睡眠模式下，由复位引脚唤醒
1	1	外部复位或 LVD 复位有效

- **注：若禁止看门狗的复位功能，看门狗也可以用作固定周期的定时器。**

3.2.3.2 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”

3.2.3.3 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

3.2.3.4 零标志

Z=1: 指令执行后，ACC 的内容为零或运算结果为零
Z=0: 指令执行后，ACC 的内容非零或运算结果非零

3.3 累加器

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，所以执行“B0MOV”指令不能够访问 ACC，必须通过“MOV”指令对 ACC 进行读/写。

➲ **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，所以一旦中断发生，必须将 ACC 和 PFLAG 存储在由用户自定义的数据存储器中，如下所示：

➲ **例：保护 ACC 和工作寄存器**

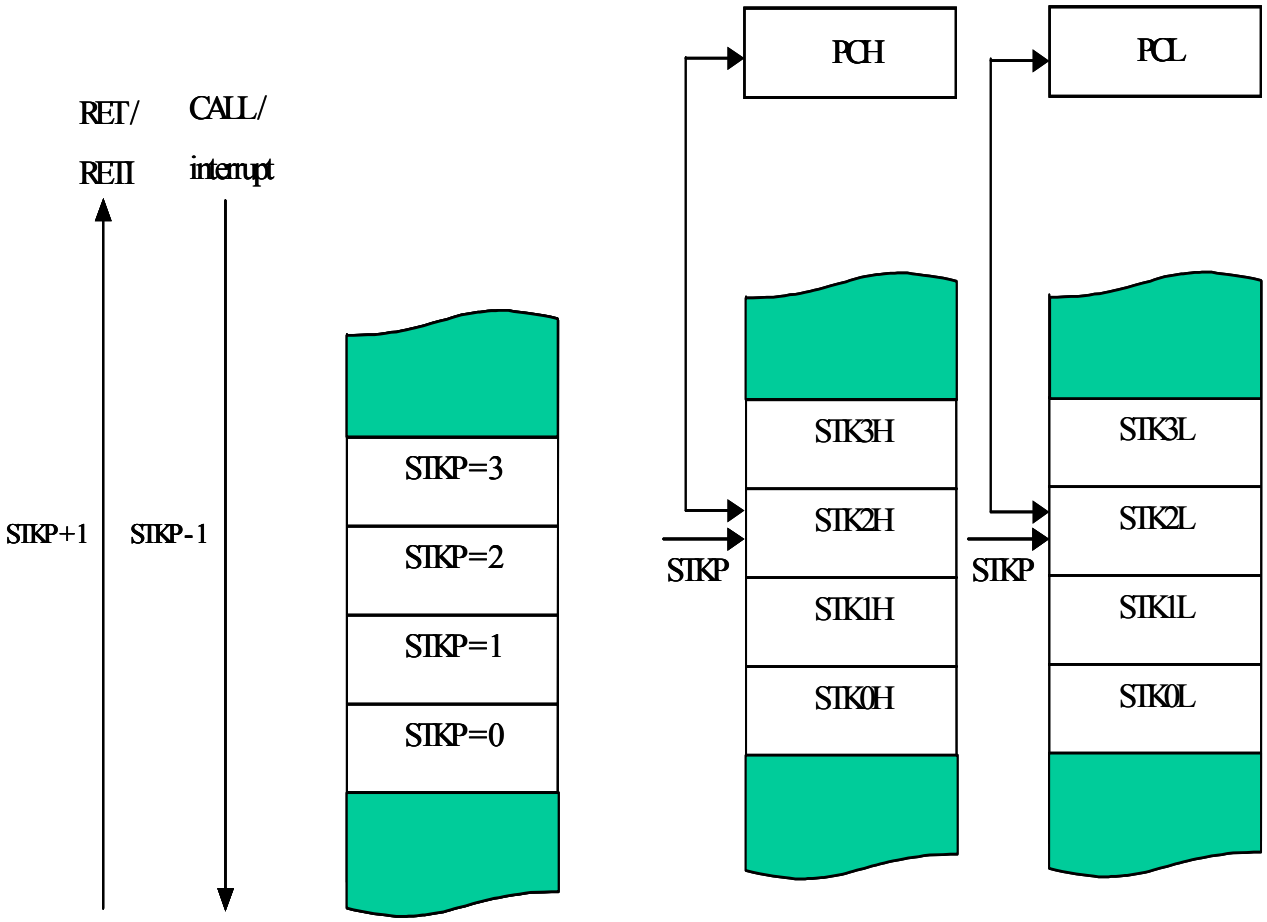
```
ACCBUF          EQU     00H             ; ACCBUF 用来保存 ACC 的数据.
PFLAGBUF        EQU     01H             ; PFLAGBUF 用来保存 PFLAG 的数据

INT_SERVICE:
                B0XCH   A, ACCBUF       ; 保存 ACC 的值
                B0MOV   A, PFLAG        ; 保存 PFLAG 的值
                B0MOV   PFLAGBUF,A
                .       .
                .
                .
                B0MOV   A, PFLAGBUF     ; 恢复 PFLAG 的值
                B0MOV   PFLAG,A
                B0XCH   A, ACCBUF       ; 用 B0XCH 指令恢复 ACC 的值不会改变 PFLAG 的值

                RETI                    ; 中断返回
```

➢ **注：为了保护并恢复 ACC，必须使用“B0XCH”指令，否则 ACC 会影响 PFLAG**

3.4 堆栈

3.4.1 概述

SN8P1602B 堆栈深度为 4 层。中断响应时，程序计数器 PC 的值会自动保存在堆栈缓存器中，堆栈指针 STKP 指示出当前栈顶位置以便保护和恢复数据，寄存器 STKnH 和 STKnL 存放程序计数器 PC 的数据。



3.4.2 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，10 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，堆栈操作遵循先进后出的原则。堆栈指针寄存器（STKP）和缓冲器 STKnH、STKnL 位于 BANK0。

➢ **SN8P1602B**

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**

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

➢ **SN8P1602B**

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

STKn= <STKnH , STKnL> (n = 3 ~ 0)

➢ **SN8P1602B**

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 = 3 ~ 0)

3.4.3 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，堆栈缓冲器中则保存了程序指针。入栈保护操作如下表所示：

➢ **SN8P1602B**

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	-
> 4	0	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冲器。堆栈恢复操作如下表所示：

➢ **SN8P1602B**

堆栈层数	*STKP*			*堆栈缓存器*		*说　明*
	STKPB2	**STKPB1**	**STKPB0**	高字节	低字节	
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	-

3.5 程序计数器

程序计数器 PC 是一个 10 位专用二进制计数器，分为 2 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。

执行程序调用（CALL）和跳转（JMP）指令时，下一条将要执行的目的地址会被重新装入 PC 的 0～9 位。

➢ **SN8P1602B**

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

3.5.1 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 PC 加 2，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      ; 否则跳转到 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      ; 否则跳转到 C0STEP.
            .
C0STEP:     NOP
```

➲ **如果增加/减小 1 后的结果是 0x00h 或 0xffh，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
```

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
```

3.5.2 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
                .       .               ;
```

4 寻址模式

4.1 概述

SN8P1602B 提供了三种 RAM 寻址模式：立即寻址、直接寻址和间接寻址。这 3 种不同寻址模式的应用将在下面一一描述。

4.2 立即寻址

将一个立即数送入累加器或指定的 RAM 单元：MOV A, # I； B0MOV M, # I

立即寻址模式

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

4.3 直接寻址

通过单元地址访问存储区：MOV A, 12H； MOV 12H, A

直接寻址模式

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

4.4 间接寻址

目的单元地址存放在指针寄存器（Y/Z）中，利用 MOV/B0MOV 指令在 ACC 和寄存器@YZ 之间读/写数据：

```
MOV     A, @YZ  ;  MOV  @YZ, A
```

➲ **例：对@YZ 间接寻址：**

```
CLR     Y               ; 清 Y，指向 bank 0.
B0MOV   Z, #12H         ;
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
```

5 系统寄存器

5.1 概述

RAM 中 bank0 的 80H~FFH 区域被留做系统专用寄存器，用来控制芯片的内部硬件资源，如输入/输出口状态、定时器和计数器等。下面的系统专用寄存器地址分配表为编写应用程序提供了方便快捷的参考基准源。通过 bank0 的读/写指令(B0MOV, B0BSET, B0BCLR…)或选定的 RAM 页(RBANK = 0)访问系统寄存器。

5.2 系统寄存器的配置（BANK0）

5.2.1 系统寄存器的字节

➢ **SN8P1602B**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-	-	R	Z	Y	-	PFLAG	RPAGE	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	PUR	PEDGE
C	P1W	P1M	P2M	-	-	-	-	-	INTRQ	INTEN	OSCM	-	-	-	PCL	PCH
D	P0	P1	P2	-	-	-	-	-	T0M	-	TC0M	TC0C	-	-	-	STKP
E	-	-	-	-	-	-	-	@YZ	-	-	-	-	-	-	-	-
F	-	-	-	-	-	-	-	-	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

说明

PFLAG = ROM 页和特殊标志寄存器
P1W = P1 口唤醒功能寄存器
PnM = Pn 口输入/输出模式寄存器
INTRQ = 中断请求寄存器
OSCM = 振荡模式寄存器
TCnM = 定时器模式寄存器
T0M.1 = TC0GN, TC0 唤醒功能寄存器
STKP = 堆栈指针
@YZ = RAM 间接寻址寄存器

R = 工作寄存器和 ROM 查表数据寄存器
Y, Z = 工作寄存器，@YZ 和 ROM 寻址寄存器
Pn = Pn 口数据缓存器
INTEN = 中断使能寄存器
PCH, PCL = 程序计数器
TCnC = 定时器计数寄存器

STK0~STK3 = 堆栈

5.2.2 系统寄存器位地址配置表

➢ **SN8P1602B**

地址	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	备注
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
0BEH	-	-	-	-	-	PUR2	PUR1	PUR0	W	PUR
0BFH	PEDGEN	-	-	P00G1	P00G0	-	-	-	W	PEDGE
0C0H	0	0	0	P14W	P13W	P12W	P11W	P10W	R/W	P1 口唤醒控制寄存器
0C1H	0	0	0	P14M	P13M	P12M	P11M	P10M	R/W	P1 口模式控制寄存器
0C2H	P27M	P26M	P25M	P24M	P23M	P22M	P21M	P20M	R/W	P2 口模式控制寄存器
0C8H	0	0	TC0IRQ	0	0	0	0	P00IRQ	R/W	INTRQ
0C9H	0	0	TC0IEN	0	0	0	0	P00IEN	R/W	INTEN
0CAH	WTCKS	WDRST	0	CPUM1	CPUM0	CLKMD	STPHX	0	R/W	OSCM
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	-	-	PC9	PC8	R/W	PCH
0D0H	-	-	-	-	-	-	-	P00	R	P0 数据缓存器
0D1H	-	-	-	P14	P13	P12	P11	P10	R/W	P1 数据缓存器
0D2H	P27	P26	P25	P24	P23	P22	P21	P20	R/W	P2 数据缓存器
0D8H	-	-	-	-	-	-	TC0GN	-	R/W	T0M
0DAH	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS	0	0	0	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DFH	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0	R/W	STKP 堆栈指针
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ 间接寻址指针
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H	-	-	-	-	-	-	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH	-	-	-	-	-	-	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH	-	-	-	-	-	-	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH	-	-	-	-	-	-	S0PC9	S0PC8	R/W	STK0H

➢ **注：**

a) **为了避免系统出错，要将上表中的未定义位都置“0”；**

b) **所有的寄存器名称已经在 SN8ASM 编译器是默认的；**

c) **寄存器中各位的名称已在 SN8ASM 汇编语言中以“F”为前缀定义过；**

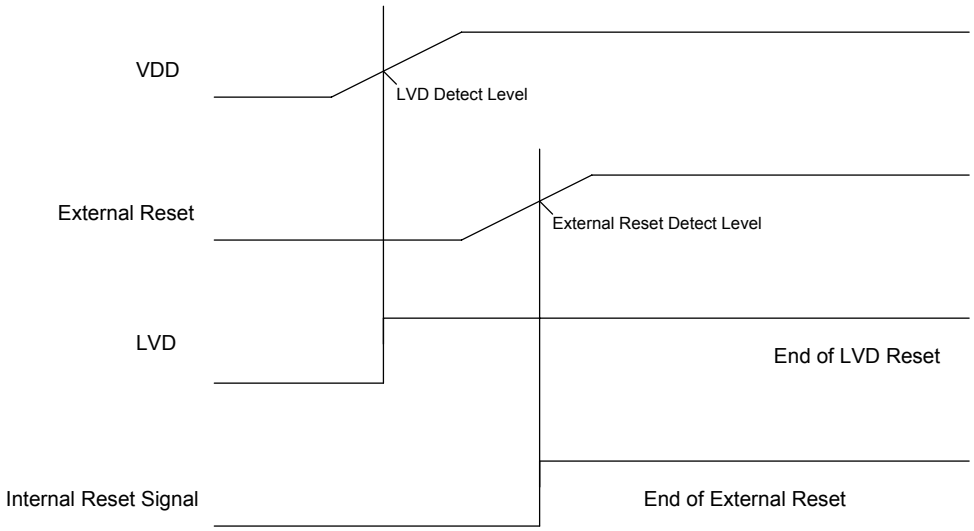
d) **指令“B0BSET”，“B0BCLR”，“BSET”，“BCLR”仅对“R/W”寄存器有效；**

e) **详细细节请查阅“系统寄存器参考表”。**

6 上电复位

6.1 概述

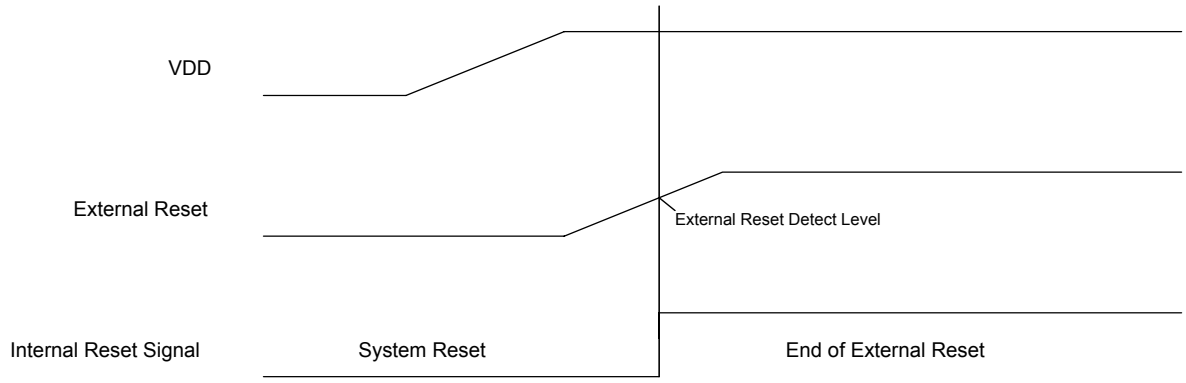
SN8P1602B 有两种系统复位方式：外部复位和内部低电压侦测（LVD）复位。外部复位电路是一个简单的 RC 电路，低电压侦测（LVD）是芯片内置电路。当其中任何一个复位信号产生时，系统复位并初始化系统寄存器，时序图如下：



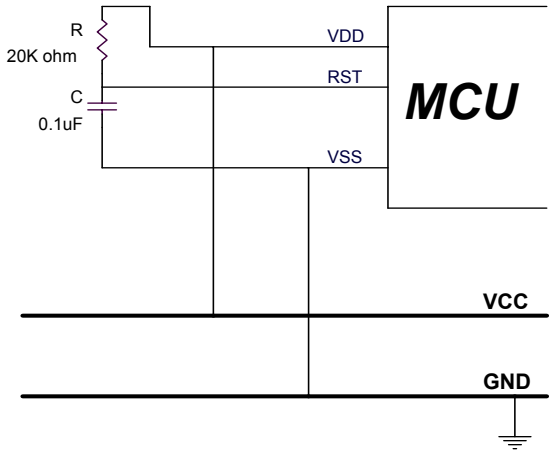
SN8P1602B 上电复位时序图

6.2 外部复位

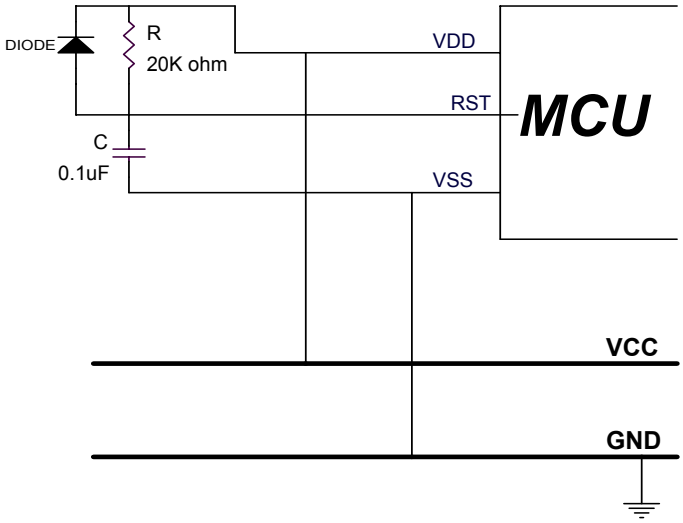
外部复位为低电平有效，当复位引脚侦测到低电压时，系统开始复位，直到侦测到的电压达到高电平。



用户必须确保 VDD 先于外部复位电压达到稳定状态，否则复位无效。外部复位电路是一个简单的 RC 电路，如下图所示：

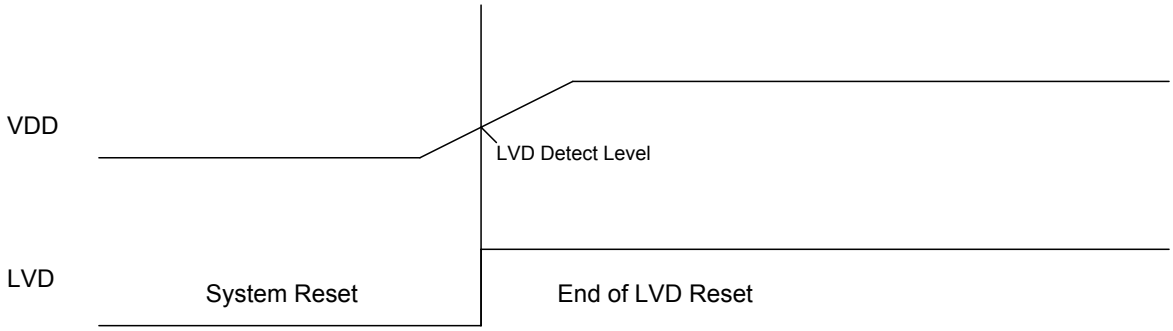


在某些情况下，通过在 VCC 和复位引脚之间放置一个二极管可以改善掉电复位。



6.3 低电压侦测（LVD）复位

LVD 为低电压侦测，当侦测到 VDD 低于设定值时，系统就会复位。低电压侦测电压为 1.8V，如果 VDD 低于 1.8V，系统就会复位。LVD 是由编译选项控制的，在电源较恶劣的情况下，用户应该开放这项功能。当 LVD 与外部复位电路中任何一个动作的时候，系统都会复位。



7 振荡器

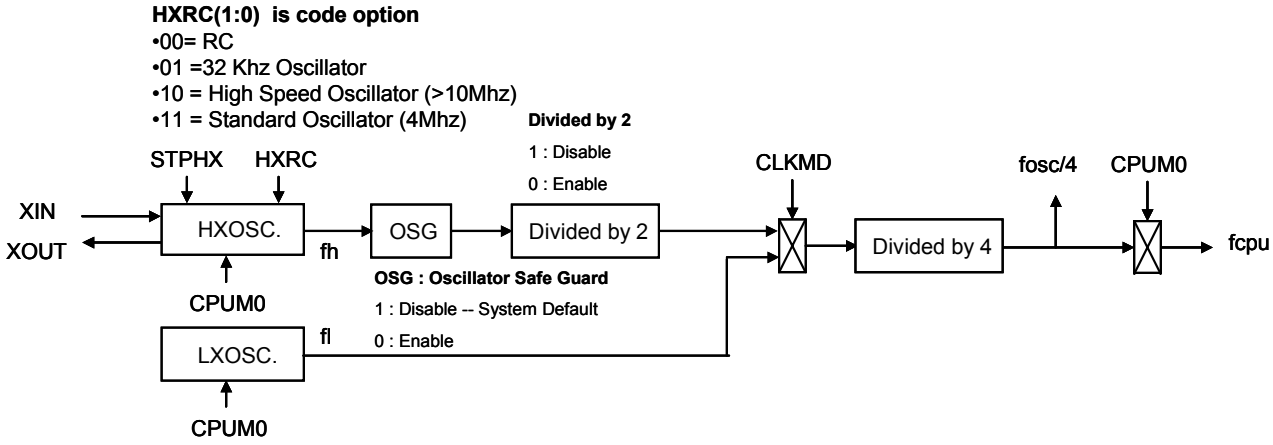
7.1 概述

SN8P1602B 是具有高速时钟和低速时钟的双时钟微控制器。高速时钟由外部振荡电路提供，低速时钟由芯片内部 RC 振荡电路提供。

外部高速时钟和内部低速时钟都可用作系统时钟（Fosc），系统时钟再经 4 分频后作为指令执行周期（Fcpu）。

Fcpu = Fosc / 4

7.1.1 时钟框图



- HXOSC：外部高速时钟
- LXOSC：内部低速时钟
- OSG：振荡器安全保护

7.1.2 OSCM 寄存器

振荡器控制寄存器 OSCM 决定了系统振荡源、系统模式以及看门狗定时器的时钟源等。

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

STPHX：外部高速振荡器控制位：0 =运行，1 = 停止。该位仅能控制外部高速振荡器，STPHX=1 时，内部低速振荡器仍然处于运行状态。

CLKMD：系统高/低速模式选择位：0 =普通模式（双重时钟），1 = 低速模式.

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。

WDRST：看门狗定时器复位位。0=看门狗定时器正常运行，1=看门狗定时计数器清零。

WTCKS：看门狗时钟源选择位。0=fcpu，1=内部 RC 低速时钟。

7.1.3 外部高速振荡器

SN8P1602B 可以工作于四种振荡器模式：RC 振荡器模式、外部高速振荡模式（12M 编译选项）、标准振荡模式（4M 编译选项）和低速振荡模式（32K 编译选项）。在不同的应用场合，用户可通过编译选项，为工作系统选择适当的高速时钟源。

➲ **例：停止外部高速振荡。**

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

➲ **例：在省电模式下，外部高速和内部低速振荡器都停止运行。**

```
B0BSET    FCPUM0        ; 停止外部高速和内部低速振荡器
                        ; 进入睡眠模式.
```

7.1.4 振荡器模式编译选项

SN8P1602B 在不同的应用下提供 4 种振荡模式，分别为：4M，12M，32K 和 RC，以支持不同的振荡器类型和频率。MCU 运行高速时钟系统比运行低速时钟系统需要的电流要多。对于晶振，有 3 个选择项，当振荡器的类型为 RC 型时，选择“RC”，系统将自动二分频。在编译前，用户可以从编译选项表中选择振荡器的类型。编译选项表如下所示：

Code Option	Oscillator Mode	Remark
00	RC 模式	从 XOUT 引脚输出频率为 Fcpu 的方波
01	32K	32768Hz
10	12M	12MHz ~ 16MHz
11	4M	3.58MHz

7.1.5 振荡器二分频编译选项

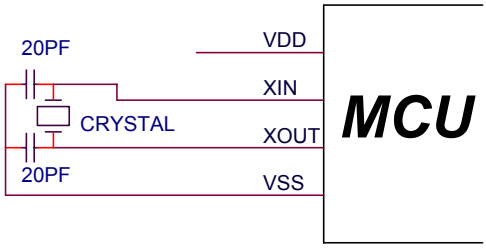
SN8P1602B 有一个将外部时钟 2 分频的编译选项：“HIGH_CLK /2”，如果“HIGH_CLK /2”是使能的，那么外部时钟频率被 8 分频后作为指令周期 Fcpu：Fcpu=Fosc/8；如果“HIGH_CLK /2”被禁止，则外部时钟频率被 4 分频后作为指令周期：Fcpu=Fosc/4。

- **注：在 RC 模式中，“HIGH_CLK /2”总是处于使能状态**

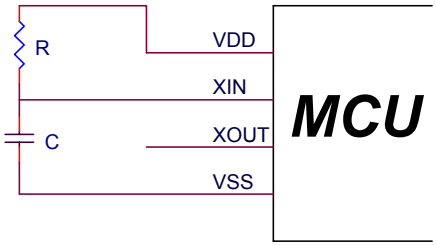
7.1.6 振荡器安全保护编译选项

SN8P1602B 自带了 OSG 以保证振荡器工作更为稳定，OSG 具体为一低通滤波电路，阻止外部振荡电路中的高频干扰进入系统中，这一功能使系统在有交流干扰的情况下保持稳定的工作状态。

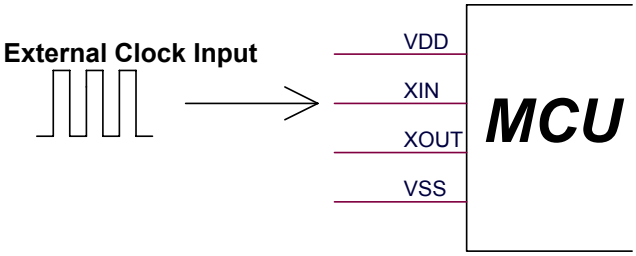
7.1.7 系统振荡器电路图



晶体振荡器



RC 振荡器



外部时钟输入

- **注 1：外部振荡器的 VDD 和 VSS 必须来自微控制器，而不是相邻的电源端。**
- **注 2：外部时钟输入可以选择 RC 振荡器或晶体振荡器，产生的时钟由 XIN 引脚输入。**
- **注 3：在 RC 模式下，外部时钟频率被 2 分频。**

7.1.8 外部 RC 振荡器频率测试

测试外部 RC 振荡器的频率 Fosc 有两种方法：一是测试 XOUT 的输出波形，在外部 RC 振荡器模式下，XOUT 的输出波形频率就是 Fcpu；二是通过程序测试指令周期 Fcpu 来测量，因为外部 RC 频率等于 4 倍的 Fcpu，通过 Fcpu 可以得到外部 RC 的频率 Fosc。测试外部振荡器频率 Fcpu 的程序如下：

➲ **例：外部振荡器的 Fcpu 周期**

```
        B0BSET      P1M.0           ; 设置 P1.0 为输出模式，输出频率信号.

@@:
        B0BSET      P1.0            ; 在低速模式下输出频率信号
        B0BCLR      P1.0            ;
        JMP         @B
```

- **注：不能直接由 XIN 测试 RC 频率，因为探针会影响 RC 的值。若想要测量 RC 频率最好利用输出脚波形来观察。**

7.1.9 内部低速振荡器

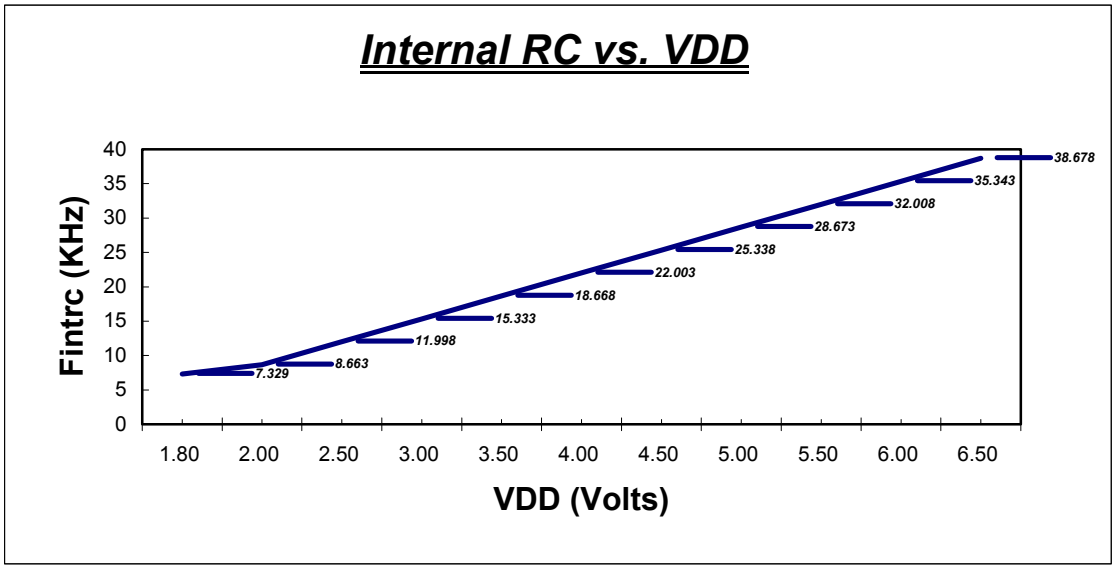
SN8P1602B 内置低速 RC 振荡器，作为它的低速时钟源。

➲ **例：停止内部低速振荡器**

```
B0BSET    FCPUM0        ; 停止外部高速和内部低速振荡器
                        ; 进入睡眠模式
```

➢ **注：内部低速时钟不能被单独停止，它由 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，输出频率信号

        B0BSET    FCLKMD        ; 切换到内部低速
@@:
        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
```

7.2 系统模式

7.2.1 概述

SN8P1602B 能够在如下四种不同的模式下相互转换：

- 普通模式
- 低速模式
- 省电模式（睡眠模式）
- 绿色模式

7.2.2 普通模式（Normal Mode）

普通模式下，系统时钟源为内部高速时钟。系统上电时，系统默认为普通模式，指令周期为 fosc /4。当外部高速振荡器是 3.58MHZ 时，指令周期为 3.58MHZ/4=895KHZ。所有的软件和硬件都能够在普通模式下运行和工作，系统可转入省电模式、低速模式和绿色模式。

7.2.3 低速模式(Slow Mode)

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

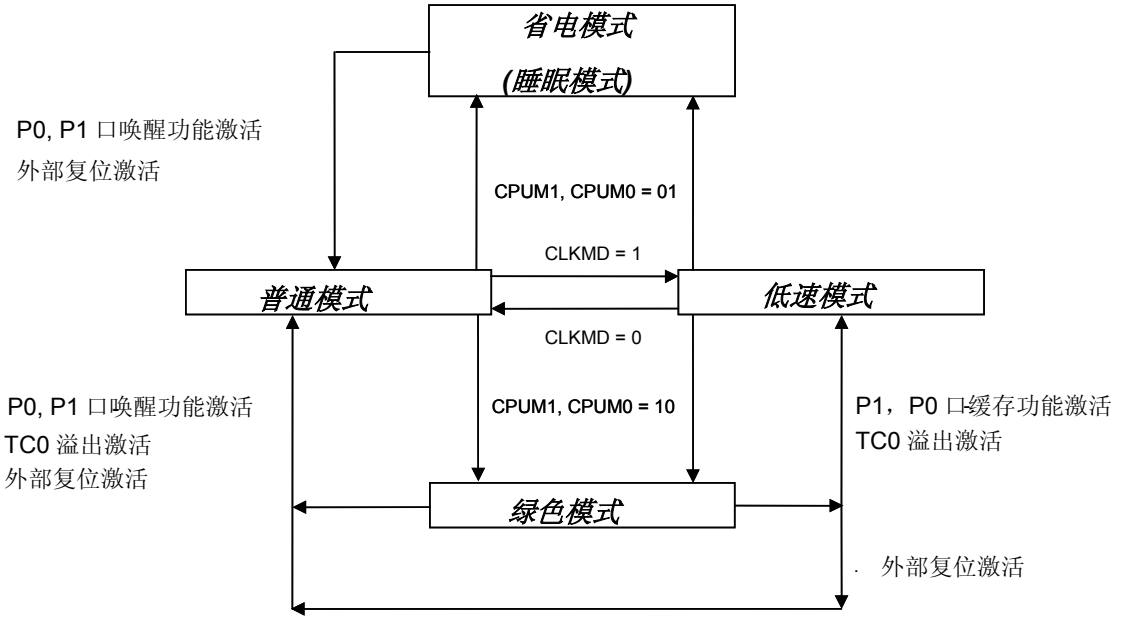
7.2.4 绿色模式(Green Mode)

绿色模式提供一个定时唤醒功能。用户可通过设置定时器 TC0 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，在普通模式下，定时器 TC0 的溢出时间较短，而在低速模式下，其溢出时间就较长。在实际应用中，用户可自由选择适当的模式。在绿色模式下，其电源功耗为 5u A(3V)。可以由 TC0 定时器或由 P0 的触发信号唤醒。

7.2.5 省电模式

省电模式也称睡眠模式，系统进入睡眠状态时，将停止工作，功耗低至近似于零。省电模式常用于电池供电等节电系统。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。

7.3 系统模式控制示意图



SN8P1602B

操作模式

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	STPHX	STPHX	停止	
LX osc.	运行	运行	运行	停止	
CPU	执行	执行	停止	停止	
TC0	*有效	*有效	*有效	无效	*由程序激活
WDT	有效	有效	INT_16K_RC	INT_16K_RC	
内部中断	全部有效	全部有效	TC0	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, TC0 复位	P0，P1 复位	

7.3.1 系统模式转换

普通/低速模式转换为省电模式
CPUM0 = 1

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

➢ **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。**

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
```

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
```

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 10mS 以等待外部振荡器稳定下来。

```
        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，由 TC0 唤醒**

```
; 设置定时器 TC0 的唤醒功能
        B0BCLR      FTC0IEN     ; 禁止 TC0 中断
        B0BCLR      FTC0ENB     ; 禁止 TC0 计数
        MOV         A,#20H      ;
        B0MOV       TC0M,A      ; 设置 TC0 时钟源 = Fcpu / 64
        MOV         A,#74H
        B0MOV       TC0C,A      ; 设置 TC0C 初始值 = 74H (设置 TC0 定时间隔 = 10 ms)
        B0BCLR      FTC0IEN     ; 禁止 TC0 中断
        B0BCLR      FTC0IRQ     ; 清 TC0 中断请求标志
        B0BSET      FTC0ENB     ; 使能 TC0 计数
        B0BSET      FTC0GN      ; 使能 TC0 唤醒功能
; 进入绿色模式
        B0BCLR      FCPUM0      ;设置 CPUMx = 10
        B0BSET      FCPUM1
```

➢ **注：如果 TC0ENB = 0 或者 TC0GN = 0, TC0 就没有将系统从绿色模式中唤醒进入普通/低速模式的功能。**

7.4 唤醒时间

7.4.1 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，外部高速振荡器可能需要经常的开关。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。

有两种情况需要唤醒时间：一是从省电模式转换到正常模式，二是从低速模式转换到正常模式。对前一种情况，SN8P1602B 提供了 2048 个振荡周期作为唤醒时间，后一种情况需要用户提供唤醒时间。

7.4.2 硬件唤醒

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，SN8P1602B 提供 2048 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×2048 (sec)＋X'tal 固定时间

X'tal 固定时间决定于 X'tal 的类型，一般的，大约为 2~4ms。

➲ **例：在省电模式，系统被 P0 或 P1 端的触发信号唤醒。唤醒时间结束后，系统进入普通模式，P0 和 P1 端唤醒时间如下：**

唤醒时间 = 1/Fosc× 2048 = 0.57 ms (Fosc = 3.58MHz)

总的唤醒时间＝0.57ms＋x'tal 固定时间

省电（睡眠）模式，只有具有唤醒功能的输入/输出端 P0，P1 能够将系统唤醒，进入通常模式。P0 的唤醒功能一直处于开放状态，而 P1 的唤醒功能受寄存器 P1W 控制。

➢ **SN8P1602B**

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

P10W~P14W：端口 1 唤醒功能控制位，0 = 无唤醒功能, 1 = 开放端口 1 的唤醒功能

7.4.3 外部唤醒触发控制

SN8P1602B 的唤醒触发方向由寄存器 PEDGE 控制。

PEDGE 初始值＝0xx0 0xxx

0BFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEDGE	PEDGEN	-	-	P00G1	P00G0	-	-	-
	R/W	-	-	R/W	R/W	-	-	-

Bit7 **PEDGEN:** 中断和唤醒触发边沿控制位。

0＝禁止边沿触发功能；

P0：低电平触发唤醒，下降沿触发中断；

P1：低电平触发唤醒。

1＝使能边沿触发功能。

P0.0：由 P00G1 和 P00G0 位控制唤醒和中断的触发；

P1：改变电平（下降沿或上升沿）触发唤醒。

Bit[4: 3] **P00G[1:0]:** port 0.0 边沿选择位。

00=保留

01=下降沿

10=上升沿

11=上升/下降双边沿

8 定时器

8.1 看门狗定时器（WDT）

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。程序中定期要将看门狗定时器清零(“B0BSET FWDRST”)，否则，看门狗定时器溢出会造成系统复位。设置 OSCM 寄存器的 WDRATE 位可以选择看门狗定时器的溢出时间。在省电模式下，看门狗定时器将被禁能。

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

WDRST：看门狗定时器复位位。0＝看门狗正常运行，1＝看门狗定时器清零。
WTCKS：看门狗定时器时钟源选择位。0＝Fcpu，1=内部 RC 低速时钟。

看门狗定时器溢出列表

WTCKS	CLKMD	Code Option	看门狗定时器溢出时间
0	0	4M_X'tal / 12M_X'tal / RC	1 / (fcpu÷2^{14}÷16)=293ms, Fosc=3.58MHz
0	0	32K_X'tal	1 / (fcpu÷2^{8}÷16)=500ms, Fosc=32768Hz
0	1	-	1 / (fcpu÷2^{14}÷16)=65.5ms, Fosc=16KHz@3V
1	-	-	1 / (16K÷512÷16)~0.5s@3V
-	-	开放 INT_16K_RC	1 / (16K÷512÷16)~0.5s@3V

注：由编译选项（Code Option）决定看门狗定时器的使能与否。

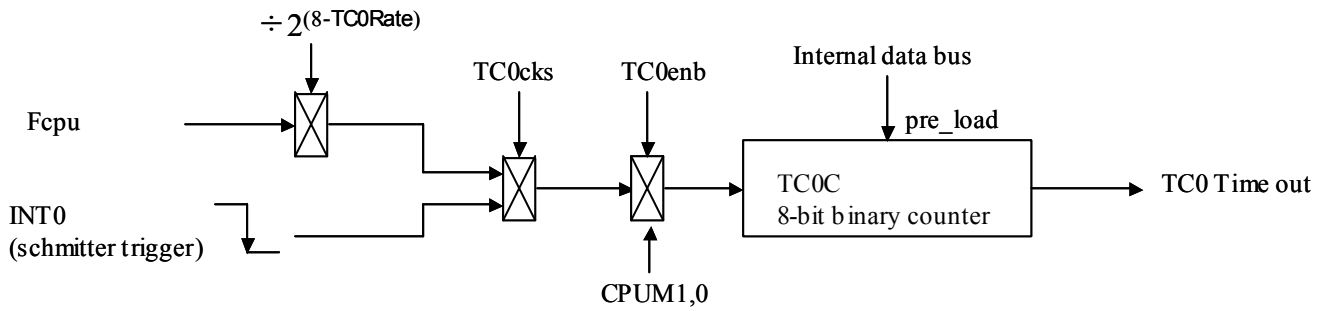
➲ **例：下面是看门狗定时器的操作，在程序的开始将看门狗定时计数器清零。**

```
Main:
        B0BSET      FWDRST      ;清看门狗定时器
        .
        CALL        SUB1
        CALL        SUB2
        .
        JMP         MAIN
```

8.2 定时/计数器 TC0

8.2.1 概述

定时/计数器 TC0 用来产生定时中断请求，利用 TC0M 寄存器从 GTMR 或者外部 INT0 引脚（下降沿触发）选择 TC0C 的时钟源，以进行精确的时间计数。如果 TC0 溢出（从 FFH 到 00H），TC0 将发出超时触发信号，请求 TC0 中断服务。



TC0 定时/计数器的主要功能如下：

- **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 INT0 输入引脚端，用外部时钟信号的下降沿记录系统“事件”。

8.2.2 TC0M 模式寄存器

TC0M 为 8 位可读/写模式控制寄存器。通过载入不同值，用户可以在执行程序过程中动态的调整定时器的时钟频率。

通过设置 TC0 的 TC0RATE0~TC0RATE2，定时器 TC0 提供了 8 种可选择的时钟源频率，从 fcpu/2 到 fcpu/256。TC0M 的初始值为 0，对应的时钟源频率为 fcpu/256。TC0M 的第 7 位 TC0ENB 位是 TC0 的启动控制位。它们共同决定了 TC0 定时器的时钟源频率和定时间隔。

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

TC0ENB：TC0 计数器控制位，“0” = 禁止，“1” = 使能
TC0RATE2~TC0RATE0：TC0 内部时钟选择位，000 = fcpu/256，001 = fcpu/128，…，110 = fcpu/4，111 = fcpu/2。
TC0CKS：TC0 时钟源控制位。0=Fcpu，1=来自 INT0/P0.0 的外部时钟。

- **注 1：S8KC ICE 不支持 PWM0OUT 和 TC0OUT 功能，PWM0OUT 和 TC0OUT 只用于 S8KD ICE（或以上版本）。**
- **注 2：若 TC0CKS=1，则 TC0 成为一个外部事件计数器。**

8.2.3 TC0C 计数寄存器

TC0C 是一个 8 位定时计数器，只要 TC0ENB 置“1”就开启定时器。TC0C 是个加 1 计数器，时钟源频率由 TC0RATE0~TC0RATE2 决定。当 TC0C 计到“0FFH”后，若再加 1 就会回到“00H”，产生溢出，此时，TC0 中断请求标志被置为“1”，如果 TC0 中断又同时被使能（TC0IEN =1），那么系统将执行 TC0 的中断服务程序。

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

TC0 间隔时间列表

TC0RATE	TC0CLOCK	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

TC0C 初始值的计算如下：

TC0 初始值 = 256－(TC0 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 TC0 定时器操作流程

定时/计数器 TC0 的工作流程如下：

- 置 TC0C 初始值，设置定时器中断间隔时间；
- TC0ENB 置为“1”，TC0 计数开始；
- 根据 TC0M 的选择的时钟源频率，每个时钟 TC0C 加 1；
- 如果 TC0 从“FFH”增至“00H”，TC0 溢出；
- 当 TC0 发生溢出，TC0IRQ 通过硬件设为“1”；
- 执行中断服务程序；
- 用户复位 TC0C，重新开始 TC0 定时器操作。

➲ **例：初始化 TC0M 和 TC0C.**

```
        B0BCLR      FTC0IEN         ; 禁止 TC0 中断
        B0BCLR      FTC0ENB         ; 停止 TC0 计数
        MOV         A,#20H          ;
        B0MOV       TC0M,A          ; 设置 TC0 的分频数为 Fcpu / 64
        MOV         A,#74H          ; 设置 TC0C 的初始值 74H
        B0MOV       TC0C,A          ; 定时时间 10 ms

        B0BSET      FTC0IEN         ; 使能 TC0 中断
        B0BCLR      FTC0IRQ         ; 清 TC0 中断请求标志
        B0BSET      FTC0ENB         ; 开始 TC0 计数
```

➲ **例：TC0 中断服务程序**

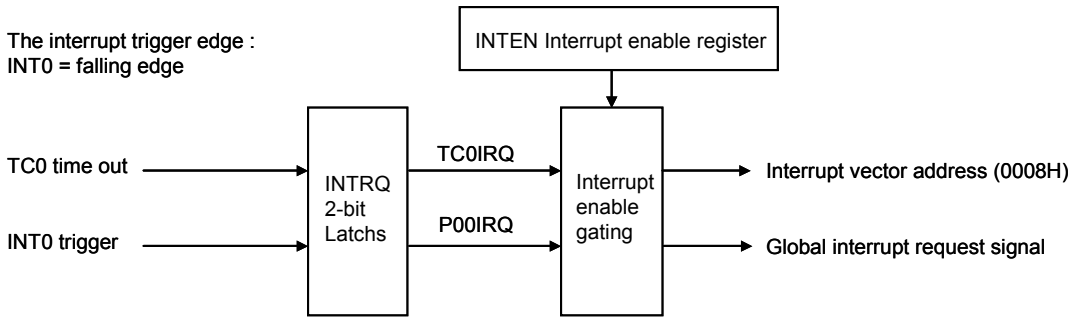
```
        ORG         8               ; 中断向量地址
        JMP         INT_SERVICE
INT_SERVICE:
        B0XCH       A, ACCBUF       ; 使用 B0XCH 不会影响到 C, Z 等标志位
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A

        B0BTS1      FTC0IRQ         ; 检查是否是 TC0IRQ 中断请求
        JMP         EXIT_INT        ; TC0IRQ = 0, 退出中断

        B0BCLR      FTC0IRQ         ; 清 TC0IRQ
        MOV         A,#74H          ; 重新设置 TC0C
        B0MOV       TC0C,A
        .           .               ; TC0 中断服务程序
        .           .
        JMP         EXIT_INT        ; TC0 中断服务程序结束
        .           .
EXIT_INT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A
        B0XCH       A, ACCBUF       ; 恢复 ACC

        RETI                        ; 中断放回
```

9 中断

9.1 概述

SN8P1602B 有 2 个中断源：一个内部中断源（TC0），一个外部中断源（INT0）。外部中断 INT0 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，用户可编程设置中断优先级。

➢ **SN8P1602B**



➢ **注：GIE 处于使能状态时，中断请求才能够被响应**

9.2 INTEN 中断使能寄存器

INTEN 为中断使能控制寄存器，包括 1 个内部中断和 1 个外部中断的控制位。INTEN 的某位被置为“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。

➢ **SN8P1602B**

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

P00IEN：外部 P0.0 中断控制位，0 = 禁止，1 =使能
TC0IEN：定时器 T0 中断控制位，0 = 禁止，1 =使能

9.3 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。

➢ **SN8P1602B**

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

P00IRQ：外部 P0.0 中断请求位， 0 = 无中断请求，1 = 请求中断服务
TC0IRQ：TC0 定时器中断请求位， 0 = 无中断请求，1 = 请求中断服务

中断发生时，无论 INTEN 是否使能，INTRQ 都会置“1”。若 INTEN =1，且 INTRQ =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 INTEN =0，不管 INTRQ 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。

9.4 中断操作

9.4.1 GIE 全局中断操作

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。

➢ **SN8P1602B**

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

GIE：全局中断控制位。0＝禁止，1＝使能

➲ **例： 设置总中断控制位(GIE)**

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

➢ **注：GIE 必须使能，所有中断才能被响应。**

9.4.2 INT0(P0.0)中断操作

外部中断触发的方式由 PEDGE 寄存器控制。

PEDGE 初始值＝0xx0 0xxx

0BFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEDGE	PEDGEN	-	-	P00G1	P00G0	-	-	-
	R/W	-	-	R/W	R/W	-	-	-

Bit7 **PEDGE**：中断和唤醒功能边沿控制位。
0＝禁止边沿触发功能。
P0：低电平触发唤醒，下降沿触发中断；
P1：低电平触发唤醒。
1＝使能边沿触发功能。
P0.0：由 P00G1 和 P00G0 位控制唤醒和中断的触发功能；
P1：改变电平（下降沿或上升沿）触发唤醒。

Bit[4:3] **P00G[1:0]**：P0.0 边沿选择位。00＝保留，01＝下降沿，10＝上升降沿，11＝上升/下降双向边沿。

➲ **例：INT0 中断请求**

```
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:
            B0XCH           A, ACCBUF       ; 保存 ACC 的值
            B0MOV           A, PFLAG
            B0MOV           PFLAGBUF, A

            B0BTS1          FP00IRQ         ; 判断是否有外部中断请求
            JMP             EXIT_INT        ;

            B0BCLR          FP00IRQ         ; 清中断标志
            .               .               ; 中断服务程序
EXIT_INT:
            B0MOV           A, PFLAGBUF
            B0MOV           PFLAG, A
            B0XCH           A, ACCBUF       ; 恢复 ACC 的值

            RETI                            ; 中断返回
```

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。

9.4.3 TC0 中断操作

➲ **例：初始化 TC0**

```
        B0BCLR      FTC0IEN         ; 禁止 TC0 中断
        B0BCLR      FTC0ENB         ; 停止 TC0 计数
        MOV         A, #20H         ;
        B0MOV       TC0M, A         ; 设置 TC0 的分频数为 fcpu / 64
        MOV         A, #74H         ; 设置 TC0C 初始值 = 74H
        B0MOV       TC0C, A         ; 定时时间 10 ms

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

        B0BSET      FGIE            ; 总中断使能
```

➲ **例: TC0 中断服务程序**

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

        B0XCH       A, ACCBUF       ; 保存 ACC 的值
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A

        B0BTS1      FTC0IRQ         ; 检查是否是 TC0IRQ 中断请求
        JMP         EXIT_INT        ; TC0IRQ = 0, 退出中断

        B0BCLR      FTC0IRQ         ; 清 TC0IRQ
        MOV         A, #74H
        B0MOV       TC0C, A         ; 重新设置 TC0C.
        .           .               ; TC0 中断服务程序
        .           .
EXIT_INT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A
        B0XCH       A, ACCBUF       ; 恢复 ACC 的值

        RETI                        ; 中断返回
```

当计数器 TC0C 溢出时，无论 TC0IEN 是否使能，TC0IRQ 都会置“1”。若 TC0IEN =1，且 TC0IRQ =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 TC0IEN =0，不管 TC0IRQ 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。

9.4.4 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	P0.0 触发。OTP 是下降沿触发。
TC0IRQ	TC0C 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**

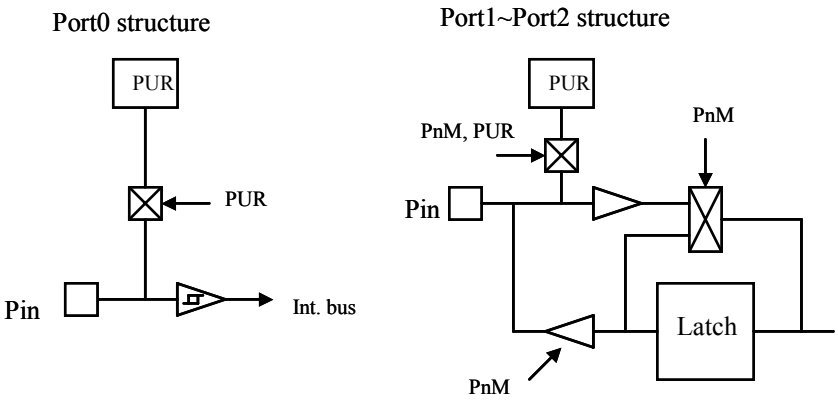
```
        ORG     8                   ; 中断向量地址

        B0XCH   A, ACCBUF           ; 保存 ACC 的值
        B0MOV   A, PFLAG            ; 保存 PFLAG 的值
        B0MOV   PFLAGBUF,A

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
INTTC0CHK:                          ; 检查是否有 TC0 中断
        B0BTS1  FTC0IEN             ; 检查是否允 TC0 中断
        JMP     INT_EXIT            ; 中断返回
        B0BTS0  FTC0IRQ             ; 检测是否有 TC0 中断请求
        JMP     INTTC0              ; 跳转到 TC0 中断服务程序
INT_EXIT:

        B0MOV   A, PFLAGBUF         ; 恢复 PFLAG 的值
        B0MOV   PFLAG,A
        B0XCH   A, ACCBUF           ; 恢复 ACC 的值

        RETI                        ; 中断返回
```

10 I/O 端口

10.1 概述

SN8P1602B 为用户提供 3 个 I/O 端口：一个单向输入端口 P0，2 个输入/输出端口 P1、P2。输入/输出的方向由 PnM 寄存器控制，用户可以用宏指令@SET_PUR 设置上拉电阻。系统复位后，所有端口都默认为无上拉电阻的输入模式。

- **SN8P1602B**



- **注：所有锁存输出电路均为图腾柱结构**

10.2 I/O 端口的功能表

- **SN8P1602B**

端口/引脚	I/O	功 能 说 明	备注
P0.0	I	基本输入功能	
		外部中断 (INT0)	<P00G1,P00G0>
		系统睡眠模式唤醒	<P00G1,P00G0>
P1.0~P1.4	I/O	基本输入/输出功能	
		系统睡眠模式唤醒	改变电平
P2.0~P2.7	I/O	基本输入/输出功能	

- **注：外部振荡器为 RC 模式时，P1.4 被开放**

10.3 I/O 模式

寄存器 PnM 控制端口的输入输出方向，P0 始终定义为输入模式。

➢ **SN8P1602B**

0C1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1M	0	0	0	P14M	P13M	P12M	P11M	P10M
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

➢ **SN8P1602B**

0C2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2M	P27M	P26M	P25M	P24M	P23M	P22M	P21M	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

PnM=0，Pn 处于输入模式；PnM=1，Pn 处于输出模式。

➢ **PnM 是可读/写寄存器，用户可通过 B0BSET, B0BCLR 等位操作指令对 PnM 进行操作.**

➲ **例：输入/输出模式选择**

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

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

10.4 I/O 上拉电阻寄存器

➢ **SN8P1602B**

0BEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PUR	-	-	-	-	-	PUR2	PUR1	PUR0
读/写	-	-	-	-	-	W	W	W
复位后	-	-	-	-	-	0	0	0

➲ **例：I/O 上拉电阻**

```
CLR     PUR         ; 禁止端口的上拉电阻

MOV     A, #07H     ; 使能端口的上拉电阻
B0MOV   PUR, A      ;
```

10.5 I/O 数据寄存器

➢ **SN8P1602B**

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

➢ **SN8P1602B**

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

➢ **SN8P1602B**

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

➲ **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
```

➲ **例：写数据到输出端**

```
MOV       A, #55H    ; P1，P2 端口写 55H
B0MOV     P1, A
B0MOV     P2, A
```

➲ **例：写 1-bit 的数据到输出端口**

```
B0BSET    P1.3       ; 设置 P1.3 和 P2.5 为“1”.
B0BSET    P2.5

B0BCLR    P1.3       ; 设置 P1.3 和 P2.5 为“0”.
B0BCLR    P2.5
```

➲ **例：检测端口**

```
B0BTS1    P0.0       ; 检测位 P0.0 是否为 1
.
B0BTS0    P1.2       ; 检测位 P1.2 是否为 0
.
```

11 编程

11.1 编程模版

```
;*****************************************************************************
; 文件名       : TEMPLATE.ASM
; 作   者      : SONiX
; 用   途      : Template Code for SN8X16XX
; 版   本      : 09/01/2002 V1.0 First issue
;*****************************************************************************
;*   (c) Copyright 2002, SONiX TECHNOLOGY CO., LTD.
;*****************************************************************************

CHIP     SN8P1602B                          ; 选择 IC 型号

;-----------------------------------------------------------------------------
;          包含文档
;-----------------------------------------------------------------------------
.nolist                                     ; 在列表文件中不列出

     INCLUDESTD          MACRO1.H
     INCLUDESTD          MACRO2.H
     INCLUDESTD          MACRO3.H

.list                                       ; 允许列表


;-----------------------------------------------------------------------------
;                                常量定义
;-----------------------------------------------------------------------------
;    ONE              EQU          1

;-----------------------------------------------------------------------------
;                                变量定义
;-----------------------------------------------------------------------------
.DATA
                org      0h                 ;数据放到 RAM 中，从地址 0 开始的地址
     Wk00       DS       1                  ;主循环用到的临时变量
     Iwk00      DS       1                  ;中断中用到的临时变量
     AccBuf     DS       1                  ;用来保存 ACC 数据的
     PflagBuf DS         1                  ;用来保存 PFLAG 数据的寄存器

;-----------------------------------------------------------------------------
;                                标志位定义
;-----------------------------------------------------------------------------

     Wk00B0     EQU      Wk00.0             ; Wk00 的第 0 位
     Iwk00B1         EQU      Iwk00.1       ; Iwk00 的第一位
;-----------------------------------------------------------------------------
;                                代码区
;-----------------------------------------------------------------------------

.CODE

     ORG             0                      ;代码开始地址
     jmp        Reset                       ;复位向量地址
                                            ;地址 4 到 7 系统保留
     ORG             8
```

```
    jmp         Isr                 ;中断向量地址

    ORG             10h
;----------------------------------------------------------------------
; 复位程序区
;----------------------------------------------------------------------
Reset:
    mov         A,#07Fh             ;初始化堆栈指针
    b0mov       STKP,A              ;禁止中断
    b0mov       PFLAG,#00h          ;pflag = x,x,x,x,x,c,dc,z
    mov         A,#40h              ;初始化系统模式，清看门狗
    b0mov       OSCM,A

    call        ClrRAM              ;清 RAM
    call        SysInit             ;系统初始化程序
    b0bset      FGIE                ;只能总中断

;----------------------------------------------------------------------
; Main routine
;----------------------------------------------------------------------
Main:
    b0bset      FWDRST              ;清看门狗定时器

     call    MnApp


    jmp         Main

;----------------------------------------------------------------------
; 主程序
;----------------------------------------------------------------------
MnApp:

    ; 在这里放置主程序

    ret

;----------------------------------
; Jump table routine
;----------------------------------
    ORG             0x0100          ;跳转表的位置最好放在页头

    b0mov       A,Wk00
    and         A,#3
    ADD             PCL,A
    jmp         JmpSub0
    jmp         JmpSub1
    jmp         JmpSub2
;----------------------------------
JmpSub0:
    ; Subroutine 1
    jmp         JmpExit

JmpSub1:
    ; Subroutine 2
    jmp         JmpExit

JmpSub2:
    ; Subroutine 3
    jmp         JmpExit

JmpExit:
```

```
    ret                                     ;返回主程序

;-----------------------------------------------------------------------
; Isr (中断服务程序)
; Arguments   :
; Returns :
; Reg Change:
;-----------------------------------------------------------------------
Isr:
;----------------------------------
; Save ACC
;----------------------------------

    b0xch       A,AccBuf                ;使用 B0xch 不会影响到 C，Z 标志

    b0mov           A,PFLAG
    b0mov           PflagBuf,A

;----------------------------------
; Interrupt service routine
;----------------------------------

    b0bts0      FP00IRQ
    jmp         INT0isr
    b0bts0      FTC0IRQ
    jmp         TC0isr

;----------------------------------
; 退出中断
;----------------------------------

IsrExit:

    b0mov           A, PflagBuf
     b0mov          PFLAG, A            ;恢复 PFlag 的值
    b0xch       A,AccBuf                ;恢复 ACC 寄存器
                                        ;使用 B0xch 不会影响到 C，Z 标志位
    reti                                ;中断返回


;-----------------------------------------------------------------------
; INT0 中断服务程序
;-----------------------------------------------------------------------
INT0isr:
    b0bclr      FP00IRQ

     ;在这里处理外部中断

    jmp         IsrExit

;-----------------------------------------------------------------------
; TC0 中断服务程序
;-----------------------------------------------------------------------
TC0isr:
    b0bclr      FTC0IRQ

     ;在这里处理 TC0 中断

    jmp         IsrExit

;-----------------------------------------------------------------------
; 系统初始化程序
```

```
; 初始化 I/O、定时器、中断等
;--------------------------------------------------------------------------
SysInit:

    ret

;--------------------------------------------------------------------------
;清 RAM
; 使用@YZ 寄存器清 RAM (00h~2Fh)
;--------------------------------------------------------------------------

ClrRAM:

    clr     Y                   ;
    b0mov   Z,#0x2f             ;设置@YZ 地址为 2fh

ClrRAM10:
    clr     @YZ                 ;清@YZ
    decms   Z                   ;z = z – 1，若 z=0 则跳过下一条指令
    jmp         ClrRAM10
    clr     @YZ                 ;清 0x00

    ret

;--------------------------------------------------------------------------
    ENDP
```

11.2 程序检查对照表

项　　目	说　　明
未定义	系统寄存器的所有的标记为“0”的（未定义）的位都必须置为“0”，以避免系统出错。
中断	RAM 初始化之前禁止开放中断。
未使用的 I/O 口	未使用的 I/O 端口应设为低电平输出模式或上拉的输入模式以减小电流损耗。
省电模式	开放端口 0 和端口 1 的内置上拉电阻，避免未知的系统唤醒。
堆栈缓存器	小心子程序调用及中断操作，避免堆栈溢出。
系统初始化	1. 堆栈初始化：STKP=0x7F。 2. RAM 清零。 3. 初始化所有的系统寄存器。
抗干扰性	1. 同时使能编译选项中的 OSG 和 High_Clk / 2。 2. 使能噪声滤波器。 3. 使能看门狗定时器，避免系统出错。 4. 未使用的 I/O 端口设为低电平输出模式。 5. 不断刷新 RAM 中重要的系统寄存器和变量以避免干扰。 6. 禁止低功耗功能。

12 指令集

类别	操作码		说明	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
	B0XCH	A,M	A ←(M (bank 0)	-	-	-	1
	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
	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
	B0ADD	M,A	M (bank 0) ← M (bank 0) + A, if occur carry, then C=1, else C=0	√	√	√	1
	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
	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
	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
	AND	A,I	A ← A and I	-	-	√	1
	OR	A,M	A ← A or M	-	-	√	1
	OR	M,A	M ← A or M	-	-	√	1
	OR	A,I	A ← A or I	-	-	√	1
	XOR	A,M	A ← A xor M	-	-	√	1
	XOR	M,A	M ← A xor M	-	-	√	1
	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
	RRC	M	A ← RRC M	√	-	-	1
	RRCM	M	M ← RRC M	√	-	-	1
	RLC	M	A ← RLC M	√	-	-	1
	RLCM	M	M ← RLC M	√	-	-	1
	CLR	M	M ← 0	-	-	-	1
	BCLR	M.b	M.b ← 0	-	-	-	1
	BSET	M.b	M.b ← 1	-	-	-	1
	B0BCLR	M.b	M(bank 0).b ← 0	-	-	-	1
	B0BSET	M.b	M(bank 0).b ← 1	-	-	-	1
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 + 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 + 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
	RETLW		PC ← Stack, and to load a value by PC+A	-	-	-	2
	NOP		No operation	-	-	-	1

13 电气特性

13.1 极限参数

(所有的电压以 Vss 为参考标准)

电源电压(Vdd) ..... - 0.3V ~ 6.0V
输入电压(Vin)..... Vss - 0.2V ~ Vdd + 0.2V
工作环境温度 (Topr)..... -20°C ~ + 70°C
存储环境温度(Tstor)..... -30°C ~ + 125°C
功率损耗(Pc)..... 500 mW

13.2 电气特性

SN8P1602B

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（OSG，禁止低功耗）		2.4	5.0	5.5	V
		普通模式（使能 OSG，禁止低功耗）		2.4	5.0		
		普通模式（禁止 OSG，使能低功耗）		2.9	5.0		
		普通模式（OSG，使能低功耗）		2.9	5.0		
		编程模式，Vpp = 12.5V		4.5	5.0	5.5	
RAM 数据保持电压	Vdr			-	1.5	-	V
输入低电压	ViL1	除特别申明的所有引脚		Vss	-	0.3Vdd	V
	ViL2	施密特触发输入端- Port0		Vss	-	0.2Vdd	V
	ViL3	复位引脚 ; Xin (RC 模式)		Vss	-	0.3Vdd	V
	ViL4	Xin (X'tal 模式)		Vss	-	0.3Vdd	V
输入高电压	ViH1	除特别申明的所有引脚		0.7Vdd	-	Vdd	V
	ViH2	施密特触发输入端—Port 0		0.8Vdd	-	Vdd	V
	ViH3	复位引脚 ; Xin (RC 模式)		0.9Vdd	-	Vdd	V
	ViH4	Xin (in X'tal 模式)		0.7Vdd	-	Vdd	V
复位引脚漏电流	Ilekg	Vin = Vdd		-	-	1	uA
输入输出端漏电流	Ilekg	禁止上拉电阻, Vin = Vdd		-	-	2	uA
Port1 输出电流	IoH	Vop = Vdd – 0.5V		-	12	-	mA
灌电流	IoL	Vop = Vss + 0.5V		-	15	-	
Port2 输出电流	IoH	Vop = Vdd – 0.5V		-	12	-	mA
灌电流	IoL	Vop = Vss + 0.5V		-	15	-	
INTn 触发脉冲宽度	Tint0	INT0 ~INT2 中断请求脉冲宽度		2/fcpu	-	-	cycle
振荡器频率	Fosc	晶体振荡器		32768	4M	16M	Hz
		VDD = 3V，外部 RC		-	6M	-	
		VDD = 5V，外部 RC		-	10M	-	
电源电流	Idd1	运行模式（禁止低功耗）	Vdd= 5V 4Mhz	-	3	8	mA
			Vdd= 3V 4Mhz	-	1	2	mA
			Vdd= 3V 32768Hz	-	50	100	uA
	Ldd2	运行模式（使能低功耗）	Vdd= 5V 4MHz	-	2	5	mA
			Vdd= 3V 4MHz	-	0.8	2	mA
	Idd3	睡眠模式 (高速时钟停止)	Vdd= 5V 32KHz Int.RC		25	50	uA
			Vdd= 3V 16KHz Int.RC		7	20	uA
	Idd4	睡眠模式	Vdd= 5V	-	1	2	uA
			Vdd= 3V	-	0.6	1	uA
	Idd5	绿色模式（高速时钟停止）	Vdd= 5V 32KHz Int.RC	-	15	30	uA
			Vdd= 3V 16KHz Int.RC	-	3	10	uA
LVD 侦测电压	Vdet	低电平侦测电压		-	1.8	-	V

13.3 特性曲线

在特定的范围内，这些曲线图的数据是一种设计向导。

SN8P1602B

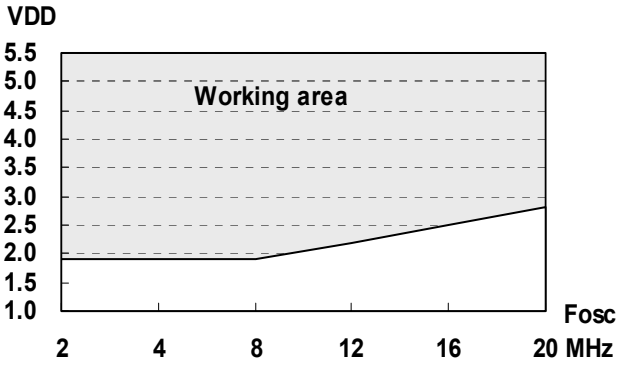


图 13－1 工作电压和频率的关系
（OSG，低功耗，禁止噪声滤波器）

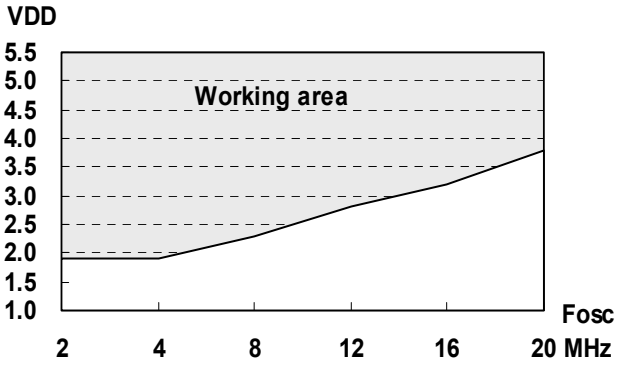


图 13－2 工作电压和频率的关系
（使能噪声滤波器，OSG，禁止低功耗）

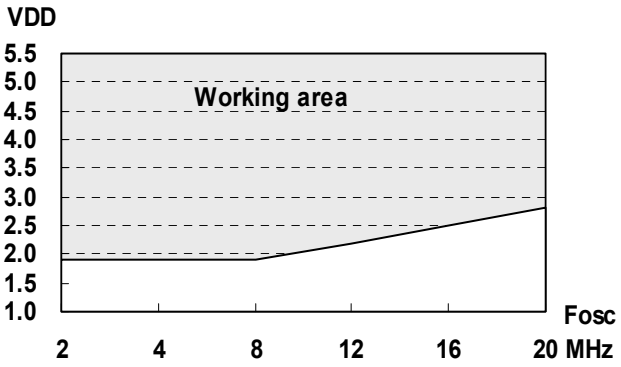


图 13－3 工作电压和频率的关系
（使能低功耗，OSG，禁止噪声滤波器）

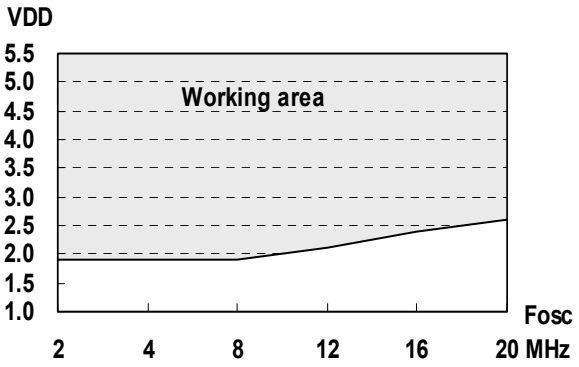


图 13－4 工作电压和频率的关系
（使能 OSG，噪声滤波器，禁止低功耗）

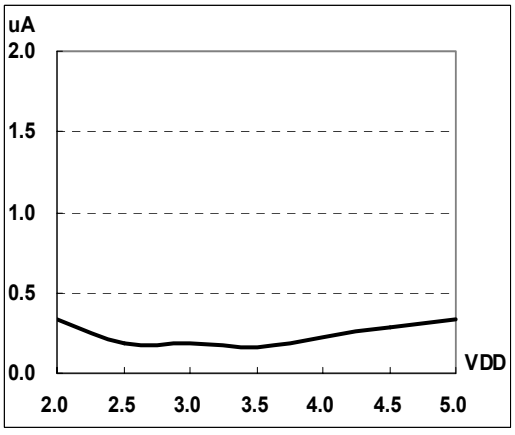


图 13－5 典型的睡眠电流 Idd4 vs. VDD

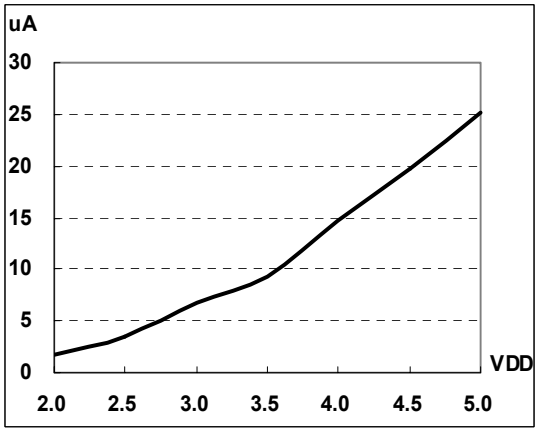


图 13－6 典型的低速电流 Idd3 vs. VDD (停止高速时钟)

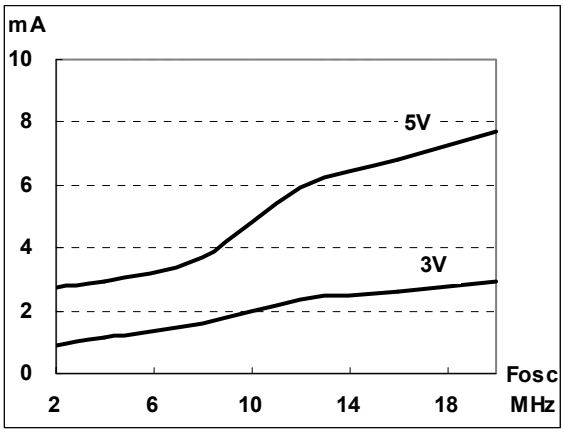


图 13－7 典型的常态电流 Idd1 vs. Fosc （屏蔽低功耗）

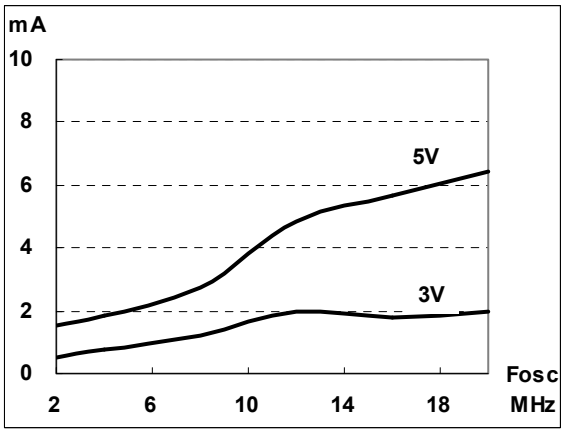


图 13－8 典型的常态电流 Idd2 vs. Fosc (开放低功耗)

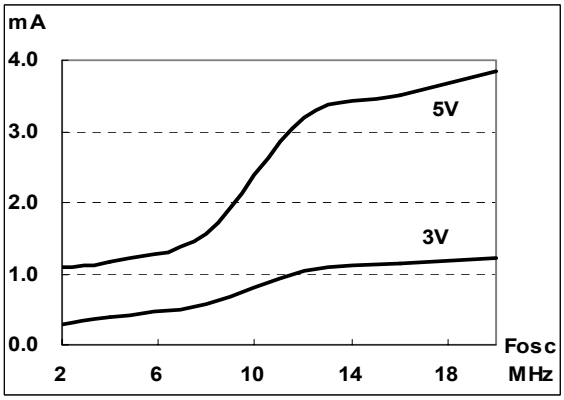


图 13-9 绿色模式电流 vs. Fosc (高速时钟仍然运行)

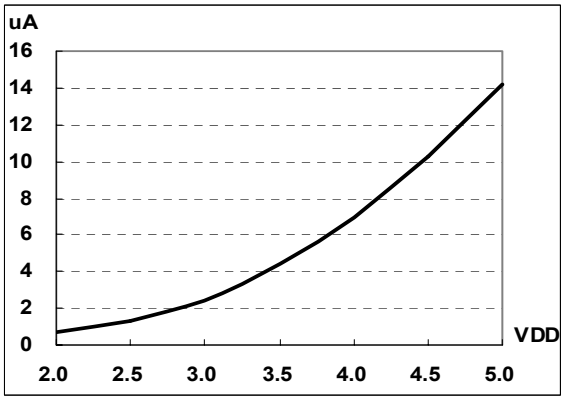


图 13-10 绿色模式电流 vs. VDD (高速时钟停止)

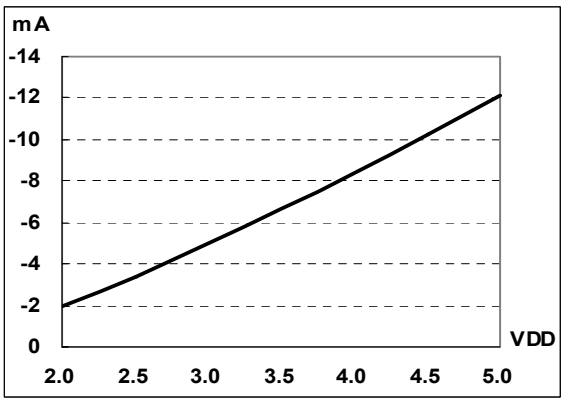


图 13-11 IOH vs. VDD

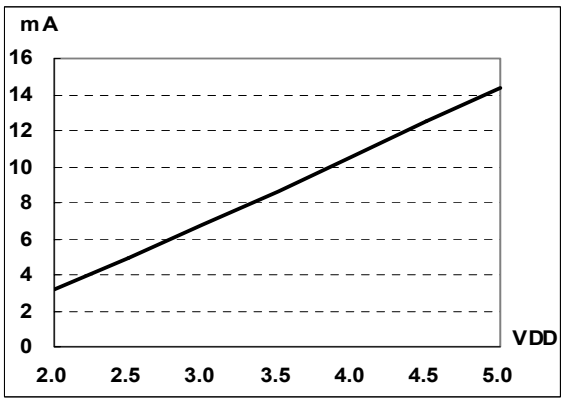


图 13-12 IOL vs. VDD

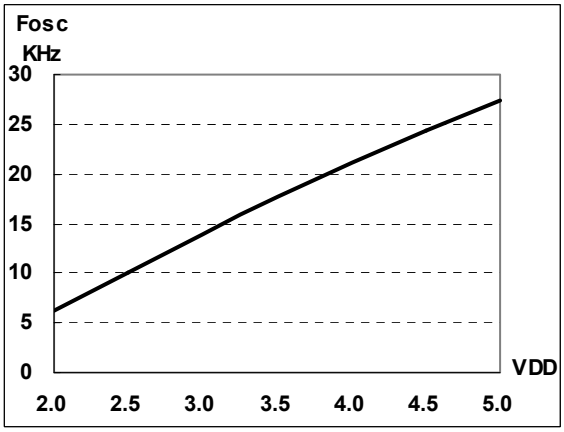
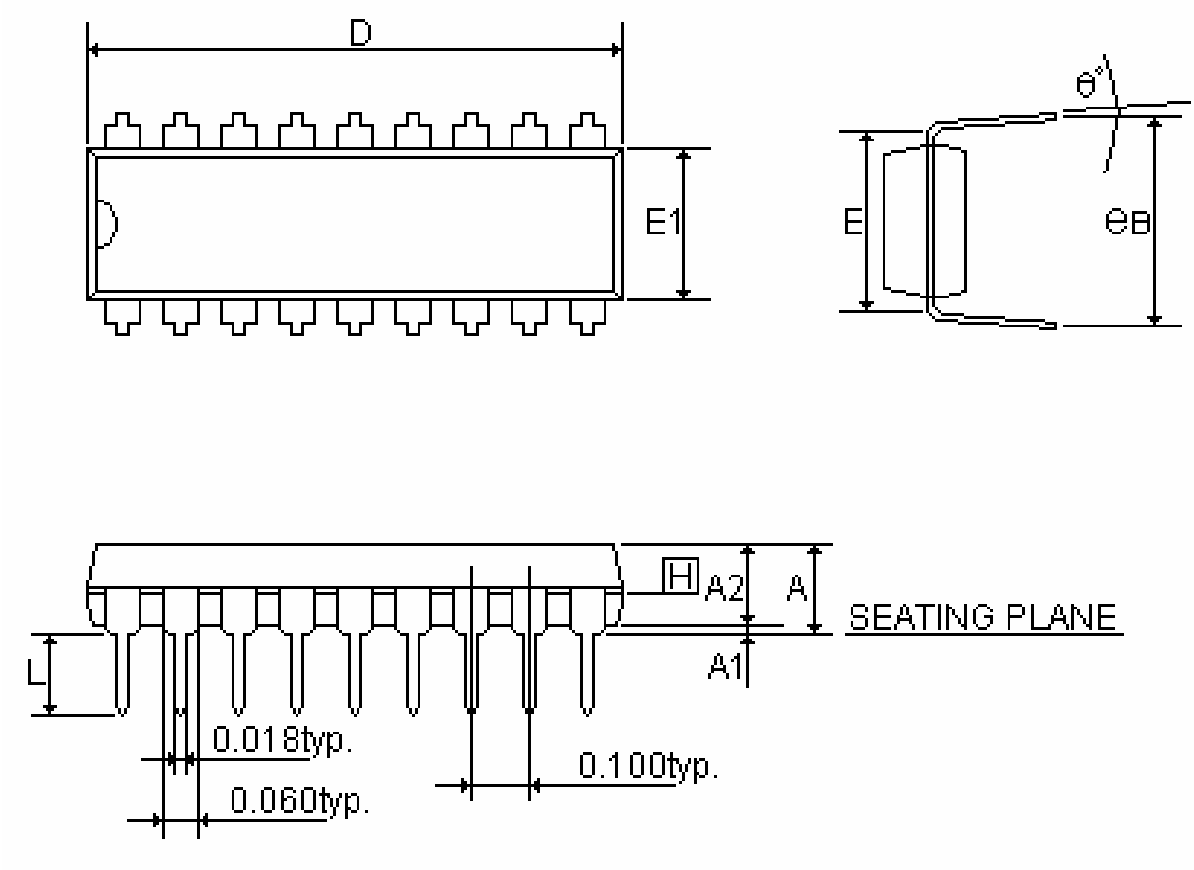


图 13-13 低速模式频率 vs. VDD

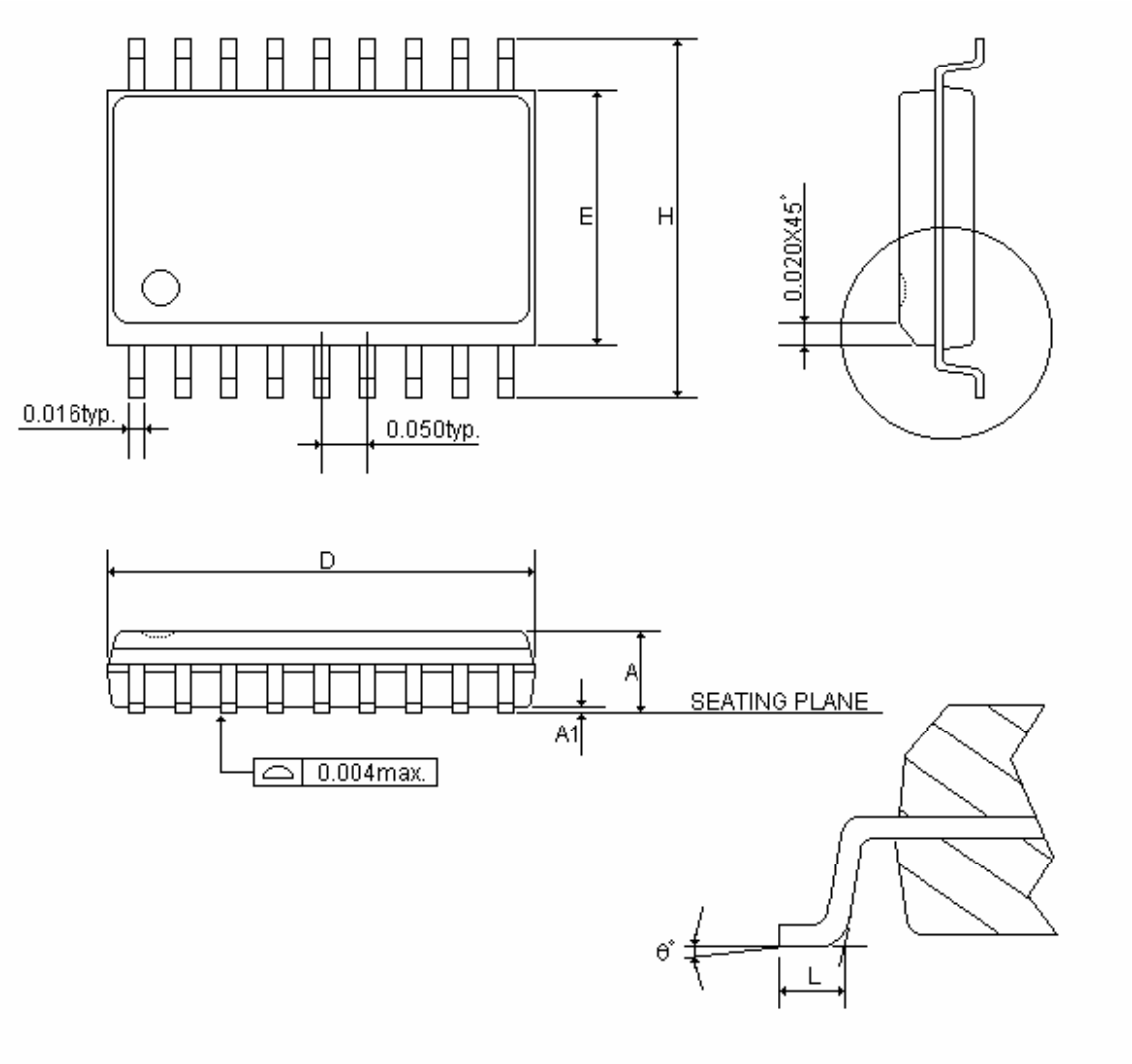
14 封装信息

14.1 P-DIP 18 PIN



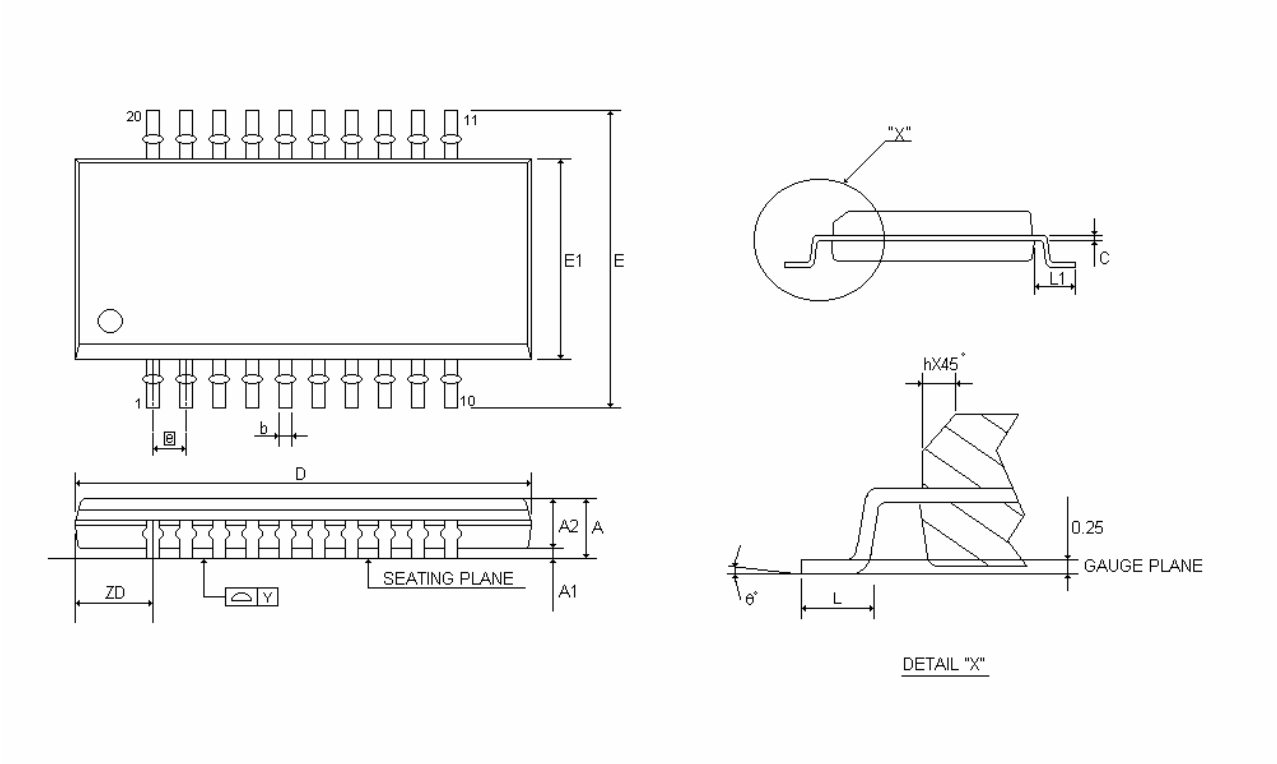
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.125	0.130	0.135	3.175	3.302	3.429
D	0.880	0.900	0.920	22.352	22.860	23.368
E		0.300			7.620	
E1	0.245	0.250	0.255	6.223	6.350	6.477
L	0.115	0.130	0.150	2.921	3.302	3.810
eB	0.335	0.355	0.375	8.509	9.017	9.525
θ°	0°	7°	15°	0°	7°	15°

14.2 SOP 18 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	0.447	0.455	0.463	11.354	11.557	11.760
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°

14.3 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 Kowllon.
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