SN8P1700A 系列

用户手册

SN8P1702A

SN8P1703A

SONiX 8 位单片机

修 正 记 录

版本号	日期	说明
VER0.1	2003 年 7 月	V1.0 第一版
VER0.2	2003 年 7 月	更改了看门狗的溢出列表
VER0.3	2003 年 7 月 2003 年 8 月	1. 修改了产品列表 2. 修改了 DC 电流特性 3. 更改特性 4. 将 SN8P1703 部分的数据改成 SN8P1706A 5. 在引脚说明的后面重新将编译选项表定位 6. 修正了 QTP 的校准表 7. 更改了寄存器的说明 8. 在电气特性里增加了 LVD 的标准值 1.8V 9. 在编译选项里增加了噪声滤波器
VER0.4	2003 年 9 月	1. 增加了 SN8P1702A SSOP20 到 Mask Mass 产品中 2. 增加定时器 TC1 到升级表中 3. 调整了第八章的表格/图的顺序 4. 调整了定时器 TC1/TC0 说明和列表 5. 调整了 PWM 的说明和列表 6. 调整了电气特性列表
VER0.5	2003 年 9 月	1. 调整了 ADC 的转换时间表 2. 调整了 PEDGE 寄存器的说明 3. 调整了 INTRQ 寄存器的说明 4. 删除了校准表 5. 将 SN8P1702A 和 SN8P1703A 的引脚分开说明 6. 删除了 PCB 布线 7. 增加了 P-DIP 20 和 SOP 20 的封装 8. 增加了 SN8A1702B 和 SN8A1703A 的说明
VER0.6	2003 年 12 月	1. 把最低工作电压从“2.2V”改为“2.4V” 2. 删除了“SN8P1702AOTP”芯片宣告 3. 删除了 SN8P1702A 和 SN8A1702A 所有的 SSOP20 的封装说明。 4. 在第一章节里修改了“OTP/MASK 的关系表”和“SN8P1702 的升级表”，将“SN8A1703A”重命名为“SN8A1703B” 5. 在第一章节里删除了 ADC 的等级表 6. 在“系统寄存器的字节”里将寄存器“TC0C”的名称改为“TC1C” 7. 删除了“OSG”的编译选项和所有的有关说明 8. 删除了“32K_X'tal”的编译选项和所有的有关说明 9. 更正了 P00G[1:0](PEDGE 寄存器)的定义 将“01=上升沿”改为“01=下降沿” 将“10=下降沿”改为“10=上升沿” 10. 把 SN8P1702（老版 OTP）的 P5.3 引脚由“P5.3/BZ1/PWM1”改为“P5.3” 11. 把 SN8P1702A/SN8P1703A 的 RST 由“RST”改为“RST/VPP” 12. 在编译选项章节中增加了“典型工作电压和系统时钟频率表的关系”
VER0.7	2003 年 12 月	1. 把 SN8A1702B(MASK)的“P5.3”改为“P5.3/BZ1/PWM1”
VER0.8	2004 年 7 月	1. 把 AVREFH 和 AVREFL 之间的最小电压由 1.2V 改为 2.0V。 2. 调整 ADC 的烧录注意事项。 3. 在 EOC 的说明中，把 “ADENB 位复位” 改为 “ADS 位复位”。 4. 调整了部分电气特性：AVref，Vani，增加 Tast。

目　　录

1 产品简介

1.1 概述

SN8P1702A/SN8P1703A 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。

SN8P1700A 系列内部包括一个大容量的程序存储器 OTP ROM，一个数据存储器 RAM，两个 8 位定时/计数器（TC0，TC1），一个看门狗定时器 WDT，三个中断源（TC0, TC1, INT0），4 通道的 8 位/12 位分辨率的 AD 转换器，两通道的高速 PWM 输出（PWM0, PWM1），两通道峰鸣器输出（BZ0, BZ1）和 8 层堆栈缓冲器。

此外，芯片有四种可选择的外部振荡源，用户可自行选择振荡方式：高/低速晶体振荡器、陶瓷谐振器和廉价的 RC 振荡器等。SN8P1700A 系列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 产品性能表

CHIP	ROM	RAM	堆栈	定时器			I/O	AVref	ADC	PWM	唤醒功能引脚数目	封装
				T0	TC0	TC1				Buzzer		
SN8P1702A	1K*16	128	8	-	V	V	12	-	4ch	2	3	DIP18/SOP18
SN8P1703A	1K*16	128		-	V	V	13	V	4ch	2	3	DIP20/SOP20/SSOP20

表 1-1 SN8P1702A / SN8P1703A 的产品列表

1.1.2 MASK / OTP 关系表

MASK	封装形式	OTP	汇编宣告
SN8A1702B	DIP18/SOP18	SN8P1702A	CHIP SN8P1702A
SN8A1703B	DIP20/SOP20 /SSOP20	SN8P1703A	CHIP SN8P1703A

1.1.3 SN8P1702 的升级（老版 OTP）

Chip	SN8P1702	SN8P1702A	SN8P1703A
汇编宣告	SN8P1702	SN8P1702A	SN8P1703A
标准电流(3V)	3uA	< 1uA	< 1uA
4MHz 工作电流(3V)	1.5mA	< 1mA	< 1mA
4MHz 工作电流(5V)	7mA	< 3mA	< 3mA
绿色模式	-	是	是
P0.0 中断边沿	下降沿	下降/上升/双边沿	下降/上升/双边沿
P1 唤醒	低电平	改变电平	改变电平
AVREFH	无	无	是
ADC 通道	4	4	4
P4CON 寄存器	-	是	是
RAM	64	128	128
GPIO	12	12	13
TC1	-	是	是
PWM	-	是	是
上拉电阻寄存器	通过端口	通过引脚	通过引脚
上拉电阻寄存器	@SET_PUR	PnUR	PnUR
SN8P1702 引脚兼容性	是	是	否
WDT 时钟源	高速时钟	高速时钟（内部 RC）	高速时钟（内部 RC）
内部 RC 总处于开发状态 WDT 的时钟源固定为内部 RC	-	是	是
上电延迟（4MHz/3V）	~70ms	~200ms	~200ms
MASK	SN8A1702A	SN8A1702B	SN8A1703B
封装	PDIP18/SOP18	PDIP18/SOP18	PDIP20/SOP20/SSOP20

➢ 注：在以后的新项目中，我们不建议使用 SN8P1702。

1.2 SN8P1702A / SN8P1703A 特性

- **存储器配置**
  OTP ROM：1K * 16 bits.
  RAM：128 * 8 bits.

- **I/O 配置**
  单向输入：P0
  输入/输出：P1，P4，P5
  具有唤醒功能的引脚：P0，P1
  上拉电阻寄存器：P0，P1，P4，P5
  外部中断：P0
  引脚 P4 和 ADC 输入共享.

- **两个 8 位定时/计数器. (TC0, TC1).**
- **内置看门狗定时器**
- **8 层堆栈缓冲器**

- **59 条功能强大的质量**
  一个指令周期为四个时钟
  所有指令长度均为 1 个字长
  绝大部分指令的周期为 1 个周期.
  查表指令(MOVC)寻址整个 ROM

- **三个中断源**
  两个内部中断：TC0, TC1
  一个外部中断：INT0.

- **一个 4 通道 12 位 ADC**

- **两个高速 PWM 输出.**
- **两个 Buzzer 输出(BZ0/BZ1)**

- **双重时钟系统提供 4 种操作模式**
  外部高速时钟：RC 最大 10MHz
  外部高速时钟：晶体 最大 16 MHz
  内部低速时钟：RC 16KHz(3V)，32KHz(5V)
  普通模式：高速时钟和内部低速时钟同时运行
  低速模式：仅内部低速时钟运行
  绿色模式：由定时器周期性唤醒
  睡眠模式：高速时钟和内部低速时钟都停止

- **封装 (支持的芯片格式)**
  SN8P1702A：PDIP 18 / SOP 18
  SN8P1703A：PDIP 20 / SOP 20 / SSOP20

1.3 系统框图

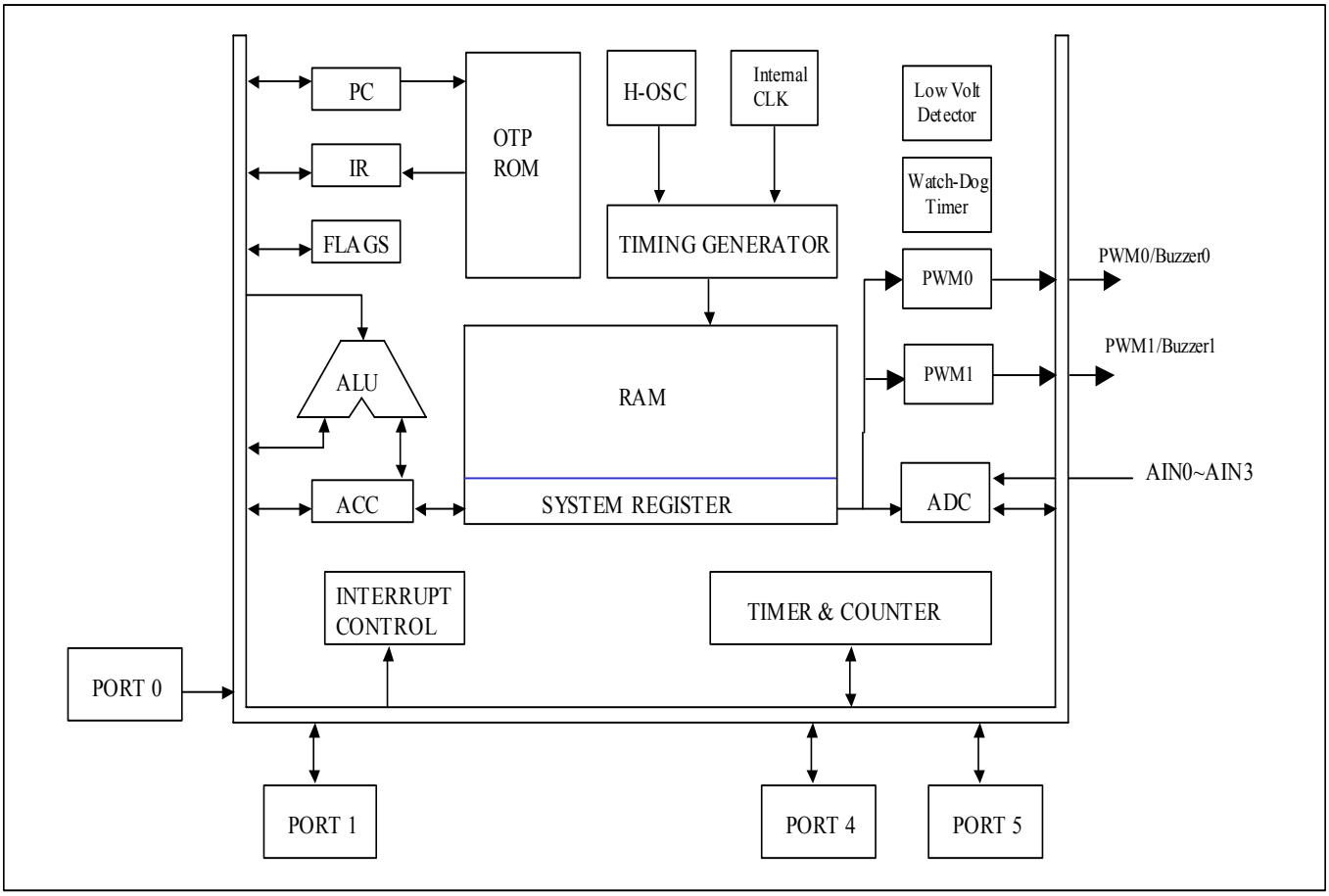


图 1-1　系统的简单框图

1.4 引脚配置

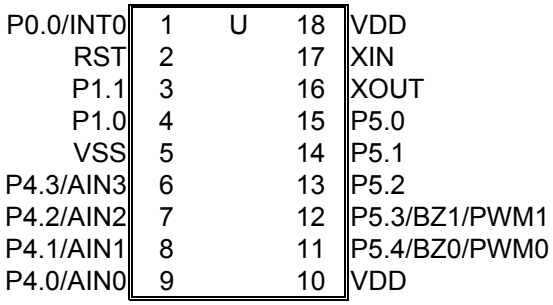
格式说明：SN8P170XAY

Y= P> PDIP， S > SOP，X> SSOP

1.4.1 SN8P1702A 引脚配置

OTP:

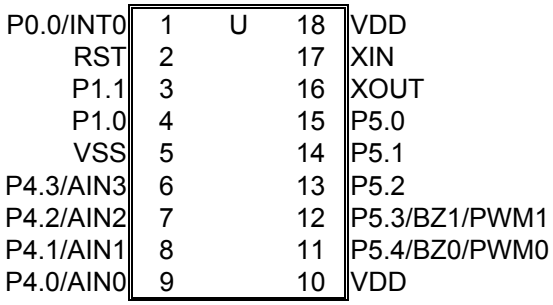
SN8P1702AS (SOP 18PIN) / SN8P1702AP (PDIP 18PIN)



SN8P1702AP
SN8P1702AS

MASK：

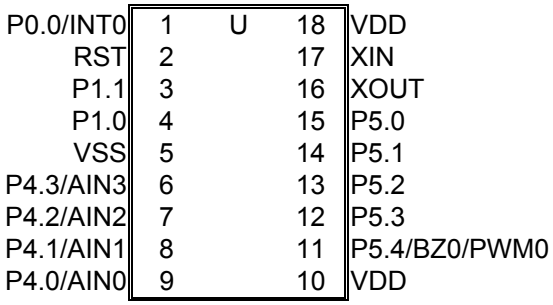
SN8A1702BS(SOP 18PIN) / SN8A1702BP(PDIP 18PIN)



SN8A1702BP
SN8A1702BS

老版 OTP：

SN8P1702S(SOP 18PIN) / SN8P1702P(PDIP 18PIN)

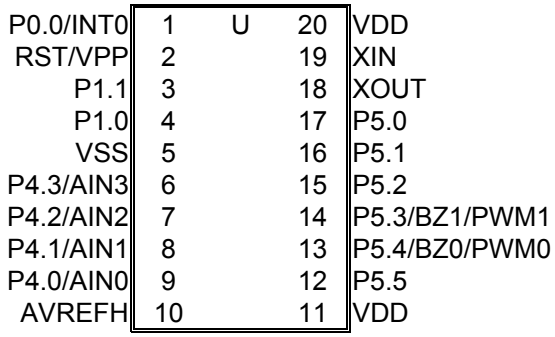


SN8P1702P
SN8P1702S

1.4.2 SN8P1703A 引脚配置

OTP:

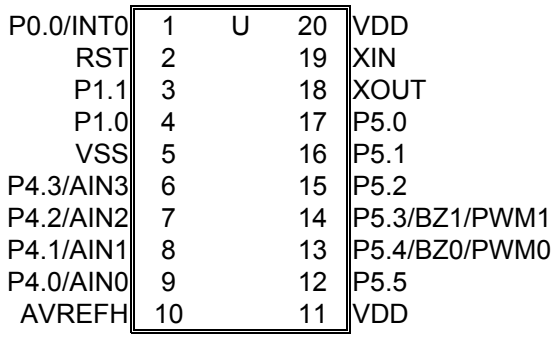
SN8P1703AS (SOP 20PIN) / SN8P1703AP (PDIP 20PIN) / SN8P1703AX (SSOP 20PIN)



SN8P1703AP
SN8P1703AS
SN8P1703AX

MASK:

SN8A1703BS (SOP 20PIN) / SN8A1703BP (PDIP 20PIN) / SN8A1703BX (SSOP 20PIN)



SN8A1703BP
SN8A1703BS
SN8A1703BX

1.5 引脚说明

PIN	类型	说　　明
VDD, VSS	P	数字电路电源输入端
RST/VPP	I	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, XOUT	I, O	外部振荡器引脚（RC 模式中为 XIN.）
P0.0 / INT0	I	P 0.0 / INT0 触发引脚 (施密特结构) ，内置上拉电阻
P1.0 ~ P1.1	I/O	P1.0~P1.1 输入/输出引脚，内置上拉电阻
P4.0 ~ P4.3	I/O	P4.0~P4.3 输入/输出引脚，内置上拉电阻
P5.0~P5.2, P5.5	I/O	P5.0~P5.2, P5.5 输入/输出引脚，内置上拉电阻
P5.3 / BZ1 / PWM1	I/O	P5.3 输入/输出引脚，Buzzer 或 PWM1 输出端，内置上拉电阻
P5.4 / BZ0 / PWM0	I/O	P5.4 输入/输出引脚，Buzzer 或 PWM0 输出端，内置上拉电阻
AVREFH	I	A/D 转换模拟参考电压高电平输入端
AIN0 ~ AIN3	I	A/D 转换输入通道

表 1-3　引脚说明

1.6 引脚电路图

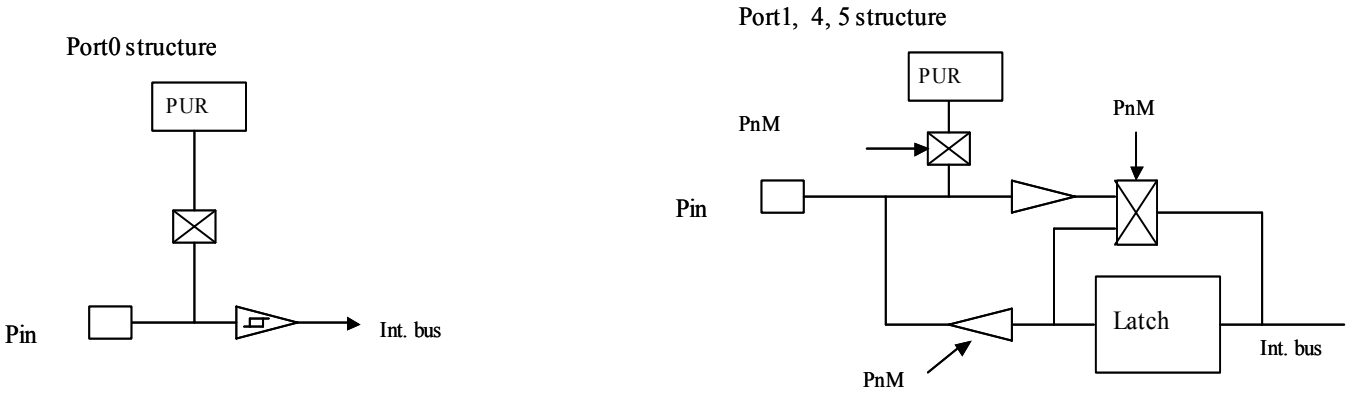


图 1-2　引脚电路图

➢ 注：所有锁存输出电路均为图腾柱结构

2 编译选项表（Code Option）

编译选项	内容	功 能 说 明
High_Clk	RC	外部高速时钟振荡器采用 RC 振荡电路
	12M X'tal	外部高速时钟振荡器采用高频晶体振荡器/陶瓷谐振器(如 12MHz)
	4M X'tal	外部高速时钟振荡器采用标准晶体振荡器/陶瓷谐振器(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
Watch_dog	Enable	使能看门狗定时器功能
	Disable	禁止看门狗定时器功能
Security	Enable	允许 ROM 程序代码加密
	Disable	禁止 ROM 程序代码加密
TC0_Counter	8-bit	TC0 作为 8 位计数器
	6-bit	TC0 作为 6 位计数器
	4-bi	TC0 作为 4 位计数器
TC1_Counter	8-bit	TC1 作为 8 位计数器
	6-bit	TC1 作为 6 位计数器
	5-bit	TC1 作为 5 位计数器
	4-bit	TC1 作为 4 位计数器
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
Low Power	Enable	使能低功耗功能以节电
	Disable	禁止低功耗功能
INT_16K_RC	Always ON	迫使看门狗定时器的时钟源来自 INT_16K_RC，则使看门狗定时器一直处于使能状态（即使在省电模式和绿色模式下）。
	By_CPUM	由 CPUM 寄存器控制内部 16K (3V) RC 时钟是否使能

表 2-1 SN8P1702A / SN8P1703A 的编译选项表

注：
- 在高干扰环境下，强烈建议使能"Noise Filter"选项；
- 如果使能"Noise Filter"选项，则编译器自动禁止"Low Power"选项；但若禁止"Noise Filter"选项，编译器则自动使能"Low Power"选项；
- 除了低速模式的其它模式下，使能"Low Power"选项会减小工作电流；
- 如果在"High_Clk"选项中选择"RC"，编译器将强迫使能"High_Clk/2"选项。

典型工作电压和系统时钟频率（Fosc）的关系表：

系统时钟（Fosc）	使能 Noise/禁止 Low Power	禁止 Noise/使能 Low Power
2 MHz	2.2	2.2
4 MHz	2.3	2.5
8 MHz	2.4	3.0
12 MHz	2.9	3.7
16 MHz	3.4	4.5

上表仅作为设计参考，而不作任何保证。

3 存储器

3.1 程序存储器（ROM）

3.1.1 概述

SN8P1702A / SN8P1703A 的程序存储器为 OTP ROM，其容量为 1K*16 位，可以由 12 位的程序计数器 PC 对程序存储器进行寻址，或由专用寄存器（R、X、Y、Z）对 ROM 进行查表访问。在标准配置中，1024*16 位的程序存储器分为 4 个区域：

- 1-word 的复位向量区
- 1-word 的中断向量区
- 5-word 的保留区
- 1K words (SN8P1702)

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~03FEH(中断向量区和通用存储区)。0008H 是中断向量的入口地址。

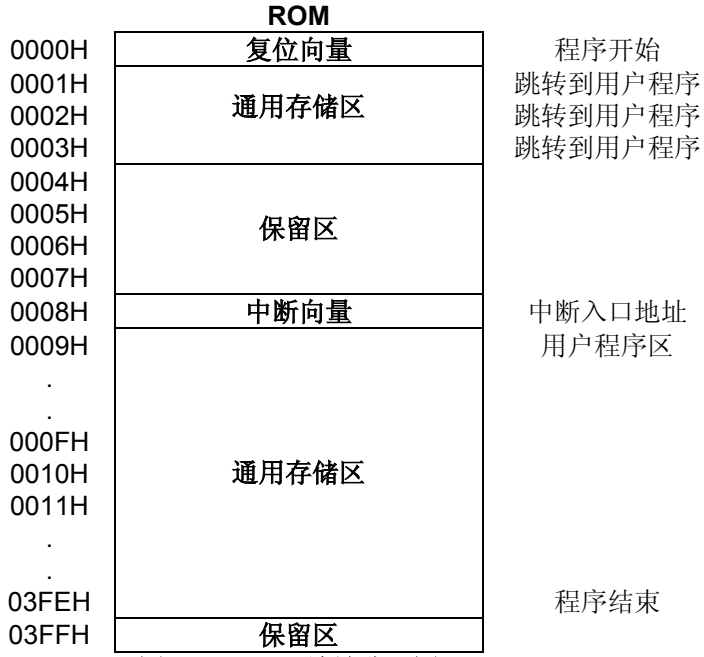


图 3-1 ROM 地址分配图

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： 上电复位，外部复位或看门狗定时器溢出复位：**

```
CHIP SN8P1702A
        ORG     0           ; 0000H
        JMP     START       ; 跳转到用户程序区
        .                   ; 0001H ~ 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：下面的程序包括中断程序，主程序位于中断服务程序后。**

```
CHIP SN8P1702A
                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 不会影响到 C, Z
                B0MOV   A, PFLAG
                B0MOV   PFLAGBUF, A     ; 保存 PFLAG 寄存器
                .
                B0MOV   A, PFLAGBUF
                B0MOV   PFLAG, A        ; 恢复 PFLAG 寄存器
                B0XCH   A, ACCBUF       ; B0XCH 不影响 C，Z
                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：下面的程序包括中断服务程序，中断服务程序的入口在通用程序存储区的指定地址**

```
CHIP SN8P1702A
                ORG     0               ; 0000H
                JMP     START           ; 跳转到用户程序.
                .                       ; 0004H ~ 0007H 保留

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

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

MY_IRQ:                                 ; 中断服务程序的起始地址
                B0XCH   A, ACCBUF       ; B0XCH 不会影响到 C, Z
                B0MOV   A, PFLAG
                B0MOV   PFLAGBUF, A     ; 保存 PFLAG 寄存器
                .
                B0MOV   A, PFLAGBUF
                B0MOV   PFLAG, A        ; 恢复 PFLAG 寄存器
                B0XCH   A, ACCBUF       ; B0XCH 不会影响到 C, Z
                RETI                    ; 中断返回

                ENDP                    ; 程序结束
```

➢ **注：从上面的程序中很容易的可以得出 SONIX 的主要编程规则，有以下几点：**

1. **地址 0000H 处的“JMP”指令使程序跳转至通用 ROM 区，0004H~0007H 由系统保留，用户必须跳过 0004H~0007H；**
2. **中断服务从 0008H 开始，用户可以将中断子程序的入口地址放在 0008H（例 1）或者在 0008H 写入一个跳转指令（例 2），而将整个中断子程序位于通用 ROM 区，从而形成模块化编程风格**

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$L
                B0MOV   END_ADDR1,A         ; 保存结束地址的低字节
                MOV     A,#END_USER_CODE$M
                B0MOV   END_ADDR2,A         ; 保存结束地址的高字节
                CLR     Y                   ; 清 Y 寄存器
                CLR     Z                   ; 清 Z 寄存器
@@:
                CALL    YZ_CHECK            ; 调用函数，判断 YZ 的值
                MOVC                        ;
                B0BSET  FC                  ; 清进位标志 C
                ADD     DATA1,A             ;
                MOV     A,R
                ADC     DATA2,A             ;
                JMP     END_CHECK           ; 检查 YZ 的值是否指向结束地址
AAA:
                INCMS   Z                   ; 递增 Z
                JMP     @B                  ; Z 不为 0，继续计算
                JMP     Y_ADD_1             ; Z=0，调整 Y
END_CHECK:
                MOV     A,END_ADDR1
                CMPRS   A,Z                 ; 检查 Y 是否等于结束地址的高字节
                JMP     AAA                 ; 不是则继续计算
                MOV     A,END_ADDR2         ; 是则结束计算
                CMPRS   A,Y                 ; 检查是否到 0004H 位置
                JMP     AAA                 ; 检查 Y 是否等于结束地址的高字节
                JMP     CHECKSUM_END        ; 不是则继续计算
YZ_CHECK:                                   ; 是则结束计算
                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                  ; 跳转到 CHECKSUM 计算
CHECKSUM_END:
                ……
                ……
END_USER_CODE:                              ; 程序结束
```

3.1.5 通用程序存储区

位于 ROM 中 0010H~0FEFH 的 992 字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。SN8P1702A/SN8P1703A 中包括用程序计数器 PC 实现的跳转表程序和通过寄存器（R、X、Y、Z）实现的查表程序。

在对程序计数器进行操作导致 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，X 寄存器指向数据 ROM 地址的高 8 位，Y 寄存器指向地址的中间 8 位，Z 寄存器指向低 8 位地址，执行 MOVC 指令后，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) 则, → Y=Y+1
                NOP                     ;
                                        ;
@@:             MOVC                    ; 查表, R = 51H, ACC = 05H.
                .       .               ;
TABLE1:         DW      0035H           ; 定义表格数据
                DW      5105H           ; “
                DW      2012H           ; “
```

- **注：当 Z 寄存器从 0XFFH 增至 0X00H 跨越页边界时，Y 寄存器不会自动增加。所以用户必须注意处理这种情况，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
```

另一种编译风格的查表是通过累加器增加间接寄存器 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$M    ; 取得表格地址高字节
                B0MOV   Z, #TABLE1$L    ; 取得表格地址低字节

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

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

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。如果跳转表跨越了 ROM 页边界（例如从 XXFFH 到 XX00H），将跳转表移动到下一页程序存储区的顶部（XX00H）。**注：一页包含 256words。**

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

```
        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，跳转表格指向正确的地址，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边界(xxFFH 到 xx00H)非常重要。良好的编译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 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
```

➢ **“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

SN8P1702A / SN8P1703A 有 256 字节内置数据存储器用来存储用户数据。

- 128 * 8 位数据存储器(bank 0)
- 128 * 8 位系统专用寄存器

该存储器分为 bank0 和 bank1 两页，用户可通过 RBANK 寄存器的页选择位访问两个 RAM 页中的所有数据单元。在 bank 0 中，前面的 128 字节作为用户通用区，剩下的 128 字节就作为系统寄存器。

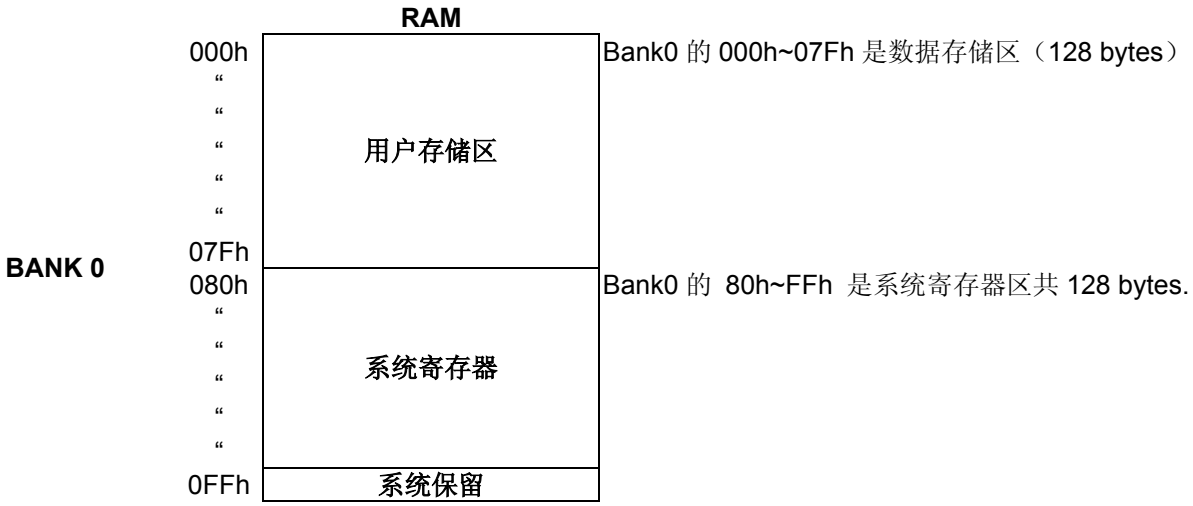


图 3－2 RAM 分配图

- **注：执行读指令“MOV A, M”后，系统寄存器中未定义的区域置为逻辑“high”。**

3.2.2 工作寄存器

RAM bank0 中 82H~84H 是系统专用寄存器，如 R、Y 和 Z 寄存器，如下表所示。这些寄存器可用作一般的工作缓冲器或用来访问 ROM 和 RAM 中的数据。例如，所有的 ROM 列表可以通过 R、Y、Z 查找，RAM 可由 Y 和 Z 寄存器间接寻址。

	82H	83H	84H
RAM	R	Z	Y
	R/W	R/W	R/W

3.2.2.1 Y,Z 寄存器

8 位寄存器 Y 和 Z 主要用作系统专用寄存器 Y 和 Z，@YZ 间接寻址和查表寻址寄存器。

Y 初始值=0000 0000

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
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Z 初始值=0000 0000

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
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

间接寻址寄存器@YZ 位于 RAM bank0 中 E7H 单元，通过 Y 和 Z 寄存器的内容访问 RAM 数据，并可通过 ACC 对该数据进行读/写。Y 寄存器的低 4 位决定了数据的 RAM 页数，Z 寄存器给出其具体地址。 Y 的高 4 位在 RAM 间接寻址模式中是无效的。

➲ **例：间接寻址访问 RAM bank 1 的 25H 单元**

```
B0MOV   Y, #01H     ; 把 BANK0 的 RAM 高 8Bit 地址送到 Y 寄存器
B0MOV   Z, #25H     ; 把 BANK0 的 RAM 底 8Bit 地址送到 Z 寄存器
B0MOV   A, @YZ      ; 读上面地址对应储存器的值到 ACC
```

➲ **例：用@YZ 对 RAM bank 1 清零**

```
        MOV     A, #1
        B0MOV   Y, A            ; Y = 1, bank 1
        MOV     A, #07FH
        B0MOV   Z, A            ; Y = 7FH，数据存储器的低地址
CLR_YZ_BUF:
        CLR     @YZ             ; 清零

        DECMS   Z               ; Y – 1，Y= 0 结束子程序
        JMP     CLR_YZ_BUF      ; 不为零继续计算

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

➢ **注：关于 Y, Z 寄存器的查表应用请参考“查表程序说明”**

3.2.2.2 R 寄存器

寄存器 R 是一个 8 位缓存器，可作为工作寄存器或在查找 ROM 数据时存放数据的高字节。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。

R 初始值=0000 0000

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
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

➢ **注：关于 R 寄存器的查表应用请参考“查表程序说明”**

3.2.3 程序状态字

程序状态字 PFLAG 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，将影响 PFLAG 寄存器。

PFLAG 初始值=xxxx x000

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PFLAG	-	-	-	-	-	C	DC	Z
	-	-	-	-	-	R/W	R/W	R/W

3.2.3.1 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”

3.2.3.2 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

3.2.3.3 零标志

Z = 1: 指令执行后，ACC 的结果为零
Z = 0: 指令执行后，ACC 的结果非零

3.3 累加器（ACC）

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

由于 ACC 不在数据存储器（RAM）中，所以执行“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 中数据**

```
; 读取 ACC 中的数据并存在 BUF 中

                MOV         BUF, A
                .           .
; 写入一个立即数到 ACC 中

                MOV         A, #0FH
                .           .
; 读取 BUF 中的数据存在 ACC 中

                MOV         A, BUF
                .           .
```

中断发生时，系统不会自动保存 ACC 和 PFLAG 的值。一旦中断发生，必须把 ACC 存储在用户自定义的存储器中，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

INT_SERVICE:
                B0XCH       A, ACCBUF       ; B0XCH 不影响标志位 C，Z

                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       ; 恢复 ACC 的值

                B0XCH       A, ACCBUF       ; 恢复 ACC 的值

                RETI                        ; 退出中断服务程序
```

➢ **注：为了保护和恢复 ACC，必须使用指令“B0XCH”，否则 ACC 会影响 PLAGE。**

3.4 堆栈

3.4.1 概述

SN8P1702A /SN8P1703A 共有 8 层堆栈，每层 12 位。堆栈缓冲器在中断现场保护和恢复时存放程序计数器 PC。堆栈指针 STKP 指示当前栈顶位置以便保护和恢复数据，12 位的寄存器 STKnH 和 STKnL 存放程序计数器 PC 的数据。

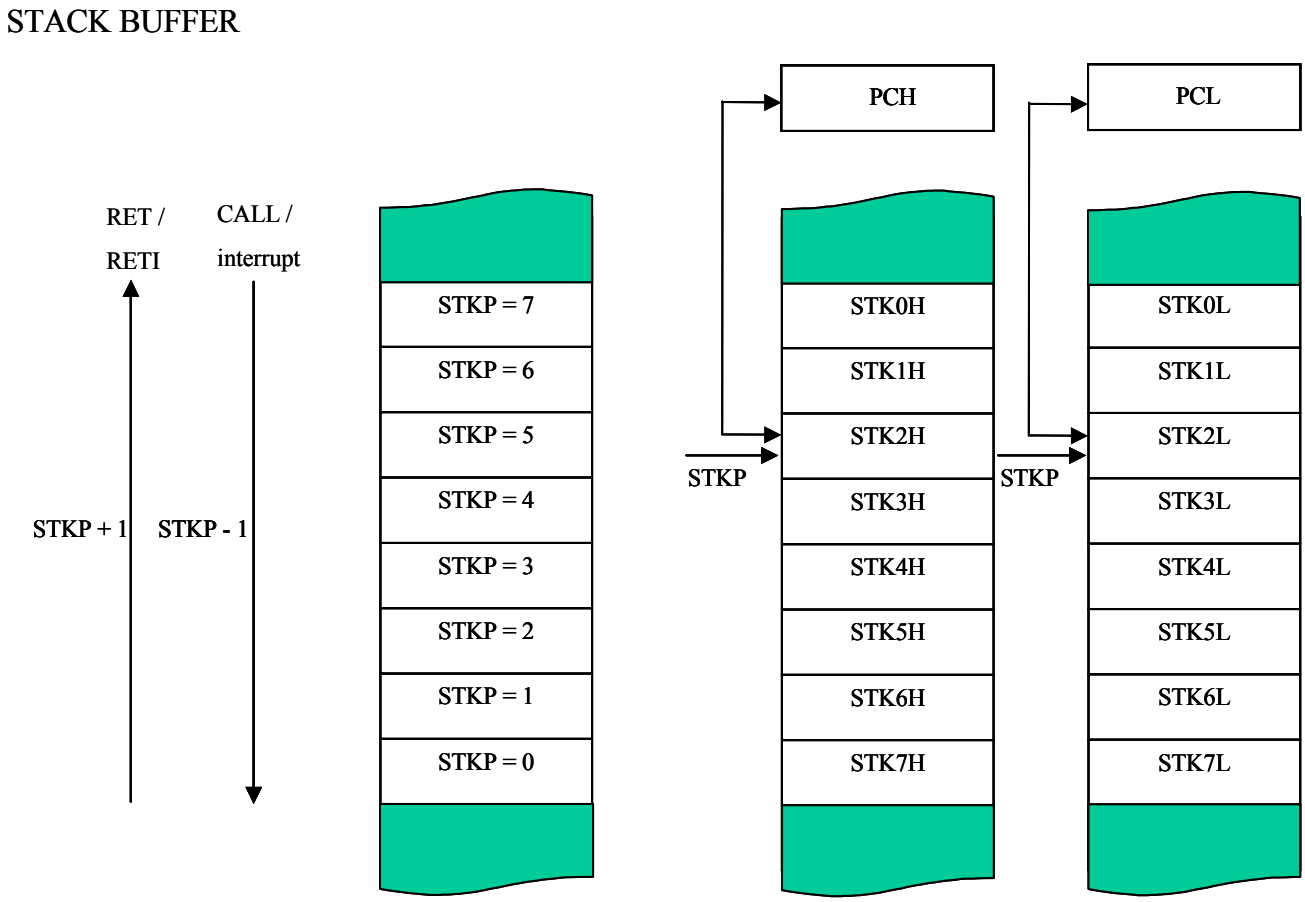


图 3—3　堆栈保护和堆栈恢复示意图

3.4.2 堆栈寄存器

堆栈指针(STKP)是一个 4 位的专用寄存器，用来指示堆栈栈顶位置，12 位的数据存储器（STKnH 和 STKnL）用来存储程序计数器（PC）的值。

堆栈操作有两种：入栈（Stack-Save）和出栈（Stack-Restore）。执行 Stack-Save 操作，STKP 自动减量；执行 Stack-Restore 操作，STKP 自动增量。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。

STKP(堆栈指针)初始值=0xxx 1111

0DFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKP	GIE	-	-	-	STKPB3	STKPB2	STKPB1	STKPB0
	R/W	-	-	-	R/W	R/W	R/W	R/W

STKBn：堆栈指针。（n = 0~3）
GIE：全局中断控制位。0 = 禁止，1 = 使能。详见中断章节。

➲ **例：堆栈指针（STKP）复位。**

```
MOV     A, #00001111B
B0MOV   STKP, A
```

STKn(堆栈缓冲器)初始值=xxxx xxxx xxxx xxxx, STKn = STKnH + STKnL (n = 7~0)

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

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
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

STKnH：在执行中断或 CALL 指令时存储 PCH 的数据。n：8 ~11.
STKnL：在执行中断或 CALL 指令时存储 PCH 的数据。n：0 ~7.

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，堆栈缓存器中则保存了程序指针。入栈保护操作如下表所示：

堆栈层数	STKP				堆栈缓冲器		说　明
	STKPB3	STKPB2	STKPB1	STKPB0	高字节	低字节	
0	1	1	1	1	STK0H	STK0L	-
1	1	1	1	0	STK1H	STK1L	-
2	1	1	0	1	STK2H	STK2L	-
3	1	1	0	0	STK3H	STK3L	-
4	1	0	1	1	STK4H	STK4L	-
5	1	0	1	0	STK5H	STK5L	-
6	1	0	0	1	STK6H	STK6L	-
7	1	0	0	0	STK7H	STK7L	-
>8	-	-	-	-	-	-	堆栈溢出

表 3－1　STKP, STKnH 和 STKnL 在堆栈保护中的关系

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				堆栈缓冲器		说明
	STKPB3	STKPB2	STKPB1	STKPB0	高字节	低字节	
7	1	0	0	0	STK7H	STK7L	-
6	1	0	0	1	STK6H	STK6L	-
5	1	0	1	0	STK5H	STK5L	-
4	1	0	1	1	STK4H	STK4L	-
3	1	1	0	0	STK3H	STK3L	-
2	1	1	0	1	STK2H	STK2L	-
1	1	1	1	0	STK1H	STK1L	-
0	1	1	1	1	STK0H	STK0L	-

表 3－2　STKP, STKnH 和 STKnL 在堆栈恢复中的关系

3.5 程序计数器（PC）

程序计数器 PC 是一个 12 位专用二进制计数器，由 4 位高字节和 8 位低字节组成，PC 总是指向下一条将要访问指令的地址，一般在程序执行过程中，PC 会随着指令的执行自动增量。

PC 初始值=xxxx 0000 0000 0000

	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	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

PCH 初始值=xxxx 0000

0CFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PCH	-	-	-	-	PC11	PC10	PC9	PC8
	-	-	-	-	R/W	R/W	R/W	R/W

PCL 初始值=0000 0000

0CEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PCL	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

3.5.1 单地址跳转

单地址跳转指令共有 9 条：CMPRS、INCS、INCMS、DECS、DECMS、BTS0、BTS1、B0BTS0、B0BTS1。如果指令的结果相匹配，PC 加 2，跳过当前指令的下一条指令。

➲ **如果位测试结果匹配，那么 PC 加 2，跳过当前指令的下一条指令；**

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

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

➲ **如果 ACC 与立即数或内存中的内容相等，那么 PC 加 2，跳过当前指令的下一条指令**

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

➲ **如果加 1（INCS、INCMS）/减 1（DECS、DECMS）后的结果为 0x00H/0xffH,那么 PC 加 2 跳过当前指令的下一条指令。**

```
INCS:
            INCS      BUF0        ; 如果 BUF0 = 0X00H，则跳过下一条指令
            JMP       C0STEP      ; 否则跳到 C0STEP.
            .
C0STEP:     NOP
INCMS:
            INCMS     BUF0        ; 如果 BUF0 = 0X00H，则跳过下一条指令
            JMP       C0STEP      ; 否则跳到 C0STEP.
            …
C0STEP:     NOP
DECS：
            DECS      BUF0        ; 如果 BUF0 = 0XFFH，则跳过下一条指令
            JMP       C0STEP      ; 否则跳到 C0STEP.
            .
C0STEP:     NOP
DECMS:
            DECMS     BUF0        ; 如果 BUF0 = 0XFFH，则跳过下一条指令
            JMP       C0STEP      ; 否则跳到 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: 设 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:设 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
                .       .               ;
```

4 寻址模式

4.1 概述

SN8P1702A / SN8P1703A 提供了三种 RAM 寻址方式：立即寻址、直接寻址和间接寻址。这 3 种不同寻址模式的用途将在下面一一描述。

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 RAM bank 0.
B0MOV   Z, #12H ; 送一个立即数 12H 到 Z
B0MOV   A, @YZ  ; 用数据指针 @YZ 取得相应存储器的地址 012H 的数据送给 ACC
```

4.5 寻址 RAM BANK0

RAM bank 0 的数据都可以通过这三种寻址方式进行读/写。

➲ **例 1: 立即数寻址（例如 B0xxx 指令）**

```
B0MOV   A, 12H      ; 赋一个立即数 12H（bank 0）到 ACC
```

➲ **例 2: @YZ 间接寻址**

```
CLR     Y       ; 清 Y，指向 RAM bank 0.
B0MOV   Z, #12H ; 送一个立即数 12H 到 Z
B0MOV   A, @YZ  ; 用数据指针 @YZ 取得相应存储器的地址 012H 的数据送给 ACC
```

5 系统寄存器

5.1 概述

RAM 中 bank0 的 80H~FFH 被留做系统专用寄存器，用来控制芯片的外部硬件资源，如输入/输出口状态、SIO、ADC、PWM、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 系统寄存器配置（BANK 0）

5.2.1 系统寄存器的字节

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-	-	R	Z	Y	-	PFLAG	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	P4CON	-
B	-	ADM	ADB	ADR	-	-	-	-	-	-	-	-	-	-	-	PEDGE
C	P1W	P1M	-	-	P4M	P5M	-	-	INTRQ	INTEN	OSCM	-	-	TC0R	PCL	PCH
D	P0	P1	-	-	P4	P5	-	-	T0M	-	TC0M	TC0C	TC1M	TC0C	TC1R	STKP
E	P0UR	P1UR	-	-	P4UR	P5UR	-	@YZ	-	-	-	-	-	-	-	-
F	STK7	STK7	STK6	STK6	STK5	STK5	STK4	STK4	STK3	STK3	STK2	STK2	STK1	STK1	STK0	STK0

表 5－1　系统专用寄存器地址分配

说明：

PFLAG = ROM 页和特殊标志寄存器
ADB = ADC 数据缓冲器
PnM = 输入输出模式寄存器
INTRQ = 中断请求寄存器
OSCM = 振荡器寄存器
TC0/1M = TC0/1 模式寄存器
TC0/1C = TC0/1 计数寄存器
TC0/1R = TC0/1 自动装载数据寄存器
STKP = 堆栈指针
@HL = 间接寻址寄存器
@YZ = 间接寻址寄存器

R = 工作寄存器和 ROM 查表数据缓存器
Y, Z = 工作寄存器，@YZ 和 ROM 寻址寄存器
RBANK = RAM Bank 选择寄存器
ADM = ADC 模式寄存器
ADR = ADC 精度寄存器
P1W = P1 口唤醒功能寄存器
Pn = Port n 数据缓冲器
PnUR= 上拉寄存器
INTEN = 中断使能寄存器
PCH, PCL = 程序计数器
STK0~STK7 = Stack 0 ~ stack 7 缓冲器

➢ **注：**

a) **所有的寄存器名称已经在 SN8ASM 汇编语言是默认的；**
b) **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；**
c) **在用指令检查空位时，都置为逻辑“H”；**
d) **ADR 寄存器中的低字节是只读寄存器；**
e) **指令“b0bset”, “b0bclr”, ”bset”, ”bclr” 仅对 “R/W” 寄存器有效。**

5.2.2 系统寄存器的位地址配置表

地址	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	读/写	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	-	-	-	-	-	C	DC	Z	R/W	PFLAG
0AEH	-	-	-	-	P4CON3	P4CON2	P4CON1	P4CON0	R/W	P4CON
0B1H	ADENB	ADS	EOC	GCHS	-	-	CHS1	CHS0	R/W	ADM 模式寄存器
0B2H	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4	R	ADB 数据缓冲器
0B3H	-	ADCKS1	ADLEN	-	ADB3	ADB2	ADB1	ADB0	R/W	ADR 寄存器
0BFH	PEDGEN	-	-	P00G1	P00G0	-	-	-	R/W	PEDGE
0C0H	0	0	0	0	0	0	P11W	P10W	W	P1W 唤醒寄存器
0C1H	0	0	0	0	0	0	P11M	P10M	R/W	P1M I/O 方向寄存器
0C4H	0	0	0	0	P43M	P42M	P41M	P40M	R/W	P4M I/O 方向寄存器
0C5H	0	0	P55M	P54M	P53M	P52M	P51M	P50M	R/W	P5M I/O 方向寄存器
0C8H	0	TC1IRQ	TC0IRQ	0	0	0	0	P00IRQ	R/W	INTRQ
0C9H	0	TC1IEN	TC0IEN	0	0	0	0	P00IEN	R/W	INTEN
0CAH	WTCKS	WDRST	WDRATE	CPUM1	CPUM0	CLKMD	STPHX	0	R/W	OSCM
0CDH	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0	W	TC0R
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	-	-	PC9	PC8	R/W	PCH
0D0H	-	-	-	-	-	-	-	P00	R	P0 数据缓冲器
0D1H	-	-	-	-	-	-	P11	P10	R/W	P1 数据缓冲器
0D4H	-	-	-	-	P43	P42	P41	P40	R/W	P4 数据缓冲器
0D5H	-	-	P55	P54	P53	P52	P51	P50	R/W	P5 数据缓冲器
0D8H	-	-	-	-	TC1X8	TC0X8	TC0GN	-	R/W	T0M
0DAH	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS	ALOAD0	TC0OUT	PWM0OUT	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DCH	TC1ENB	TC1rate2	TC1rate1	TC1rate0	0	ALOAD1	TC1OUT	PWM1OUT	R/W	TC1M
0DDH	TC1C7	TC1C6	TC1C5	TC1C4	TC1C3	TC1C2	TC1C1	TC1C0	R/W	TC1C
0DEH	TC1R7	TC1R6	TC1R5	TC1R4	TC1R3	TC1R2	TC1R1	TC1R0	W	TC1R
0DFH	GIE	-	-	-	STKPB3	STKPB2	STKPB1	STKPB0	R/W	STKP 堆栈指针
0E0H								P00R	R/W	P0UR
0E1H							P11R	P10R	R/W	P1UR
0E4H					P43R	P42R	P41R	P40R	R/W	P4UR
0E5H			P55R	P54R	P53R	P52R	P51R	P50R	R/W	P5UR
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ 间接寻址寄存器
0F0H	S7PC7	S7PC6	S7PC5	S7PC4	S7PC3	S7PC2	S7PC1	S7PC0	R/W	STK7L
0F1H	-	-	-	-	-	-	S7PC9	S7PC8	R/W	STK7H
0F2H	S6PC7	S6PC6	S6PC5	S6PC4	S6PC3	S6PC2	S6PC1	S6PC0	R/W	STK6L
0F3H	-	-	-	-	-	-	S6PC9	S6PC8	R/W	STK6H
“	“	“	“	“	“	“	“	“	“	“
“	“	“	“	“	“	“	“	“	“	“
“	“	“	“	“	“	“	“	“	“	“
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

表 5－2　系统寄存器的位地址配置表

➢　注：

a）为避免系统出错，程式中使用到上表中所有标“0”的位时要设定为“0”；

b）所有的寄存器名称在 SN8ASM 编译器中是默认的；

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。

6 上电复位

6.1 概述

SN8P1702A / SN8P1703A 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：

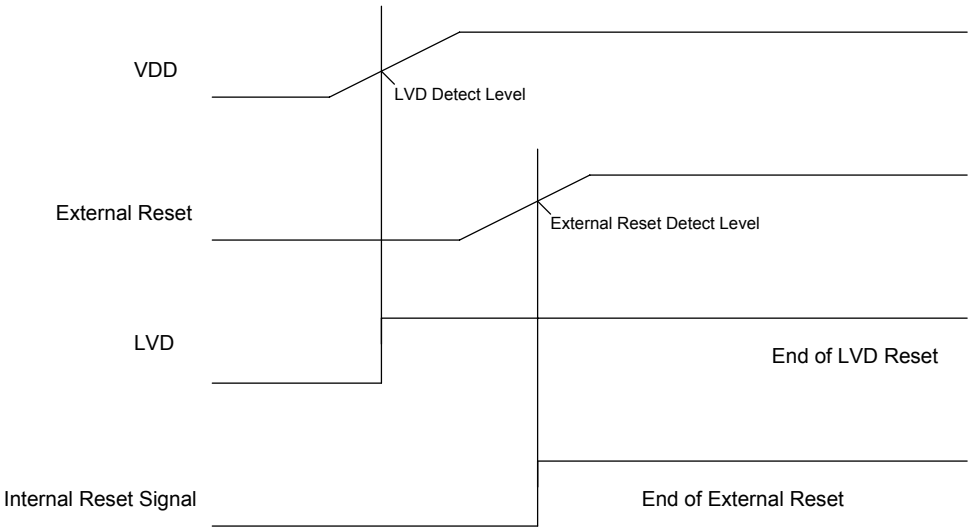


图 6－1　上电复位时序图

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。

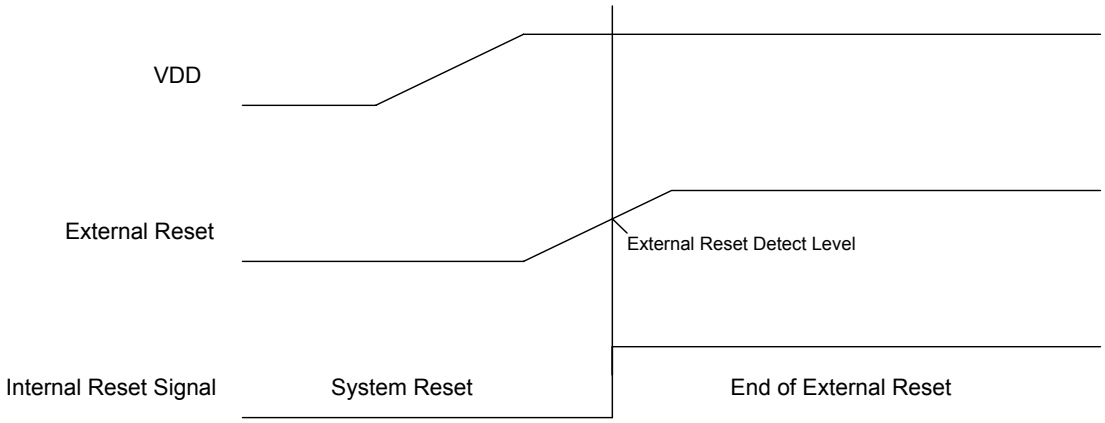


图 6－2　外部复位时序图

用户必须确保 VDD 先于外部复位电压达到稳定状态（图 6－2），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：

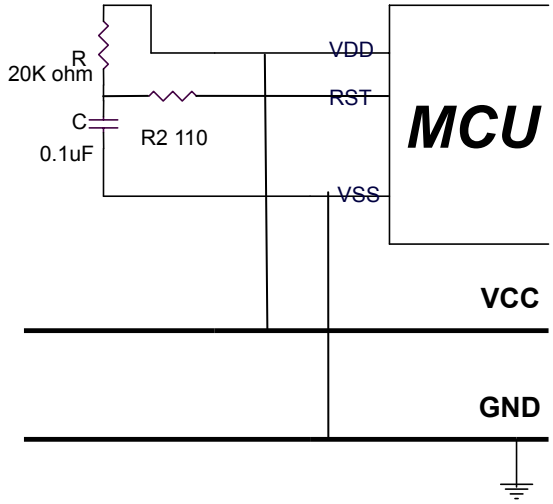


图 6－3　外部复位电路

➢ **注：使用 R2 可以避免外部干扰对 MCU 复位的影响，建议 R2 的值为 100～200Ω。**

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。

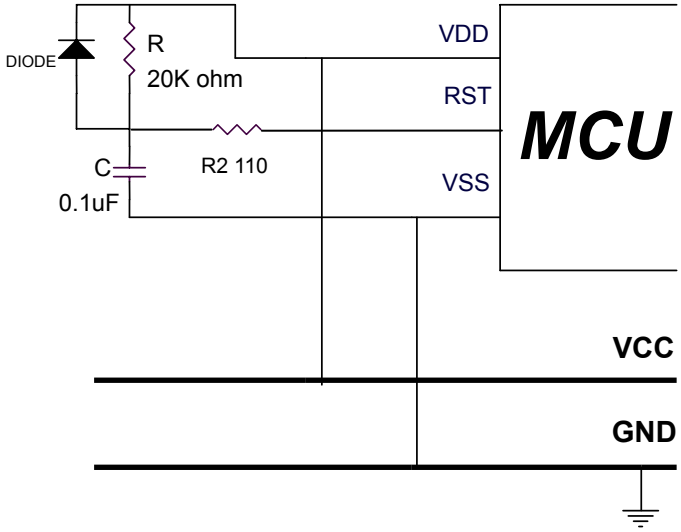


图 6－4　加二极管后的外部复位电路

➢ **注：使用 R2 可以避免外部干扰对 MCU 复位的影响，建议 R2 的值为 100～200Ω。**

7 振荡器

7.1 概述

SN8P1702A / SN8P1703A 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时钟周期(Fcpu)。

Fcpu = Fosc / 4

下面是需要用到系统时钟的外围设备：

- ✓ **定时计数器（TC0 /TC1）**
- ✓ **看门狗定时器**
- ✓ **AD 转换**
- ✓ **PWM 输出（PWM0, PWM1）**
- ✓ **蜂鸣器输出（TC0OUT, TC1OUT）**

7.1.1 时钟框图

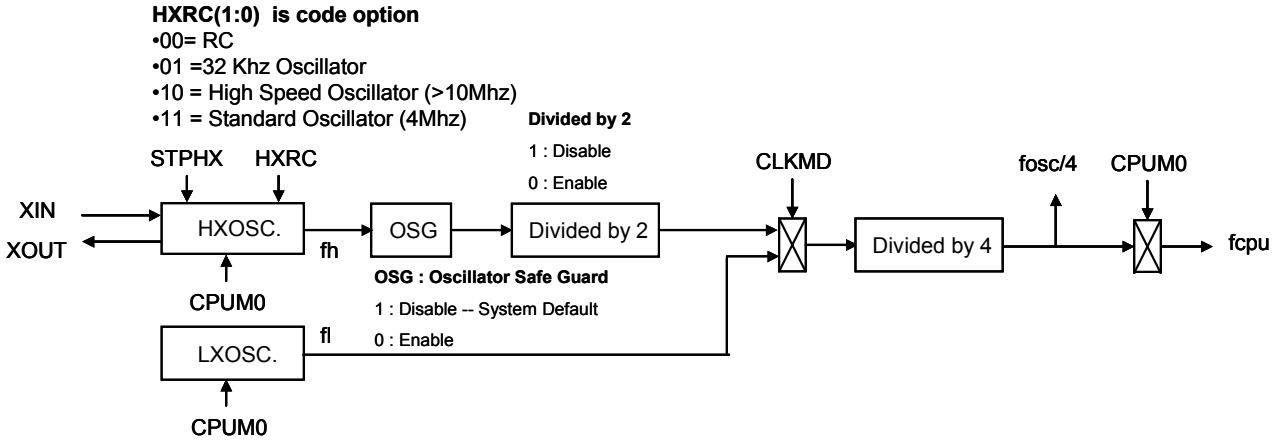


图 7－1　时钟框图

- ➢ HXOSC：外部高速时钟
- ➢ LXOSC：内部低速时钟

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。

OSCM 初始值 = 0000 000x

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	0	WDRST	Wdrate	CPUM1	CPUM0	CLKMD	STPHX	0
	-	R/W	R/W	R/W	R/W	R/W	R/W	-

Bit1 **STPHX:** 外部高速时钟控制位
0 = 自由运行
1 = 停止

➢ **注：该位仅能控制外部高速振荡器。如果 STPHX=1，内部低速时钟 RC 振荡器仍然运行。**

Bit2 **CLKMD:** 系统高/低速模式选择位
0 = 普通（双重）模式
1 = 低速模式

Bit[4:3] **CPUM[1:0]:** 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

Bit5 **Wdrate:** 看门狗定时器速率选择位
0 = Fcpu ÷ 2^{14}
1 = Fcpu ÷ 2^{8}

Bit6 **WDRAT:** 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

Bit7 **WTCKS：**看门狗定时器时钟源选择位
0 = Fcpu
1 = 内部 RC 低速时钟

7.1.3 外部高速振荡器

SN8P1702A / SN8P1703A 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: 外部高速振荡器停止.**

```
B0BSET    FSTPHX      ; 停止外部高速振荡器

B0BSET    FCPUM0      ; 停止外部高速振荡器和内部低速振荡器并将系统从睡眠(省电)模式中唤醒.
```

7.1.4 振荡器模式编译选项

SN8P1702A / SN8P1703A 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：

代码	振荡器模式	说　明
00	RC 模式	从 XOUT 引脚输出频率为 Fcpu 的方波
01	32K 晶振	32768Hz
10	12M 晶振	12MHz ~ 16MHz
11	4M 晶振	3.58MHz

7.1.5 振荡器二分频编译选项

SN8P1702A / SN8P1703A 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”通常处于使能状态。**

7.2 系统振荡器电路图

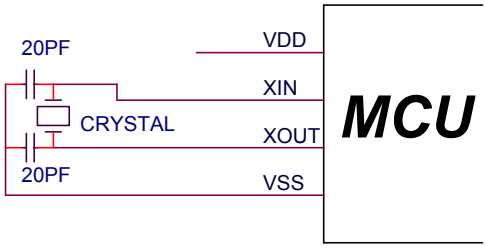


图 7－2　晶体振荡器

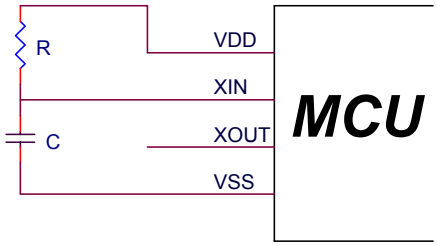


图 7－3　RC 振荡器

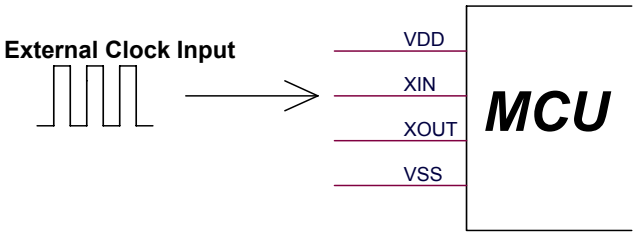


图 7－4　外部时钟输入

- **注 1：外部振荡电路的 VDD 和 VSS 必须来自微控制器，而不是相邻的其它电源端。**
- **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 振荡器模式下，外部时钟频率被 2 分频。**
- **注 4：外部振荡器的电源端和接地端必须和微控器的 VDD 和 VSS 相连，以提高整个系统的性能。**

7.3 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 频率就等于 4 倍的 Fcpu，通过 Fcpu 就可以得到外部 RC 的频率 Fosc。测试外部振荡器频率 Fcpu 的程序如下：

➲ **例:外部振荡器的指令周期测试**

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

@@:
        B0BSET    P1.0        ; 在 RC 模式下输出频率信号.
        B0BCLR    P1.0        ;
        JMP       @B
```

7.4 内部低速振荡器

SN8P1700A 内置低速 RC 振荡器可提供系统时钟、定时计数器、看门狗定时器、SIO 等的时钟源。

➲ **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：

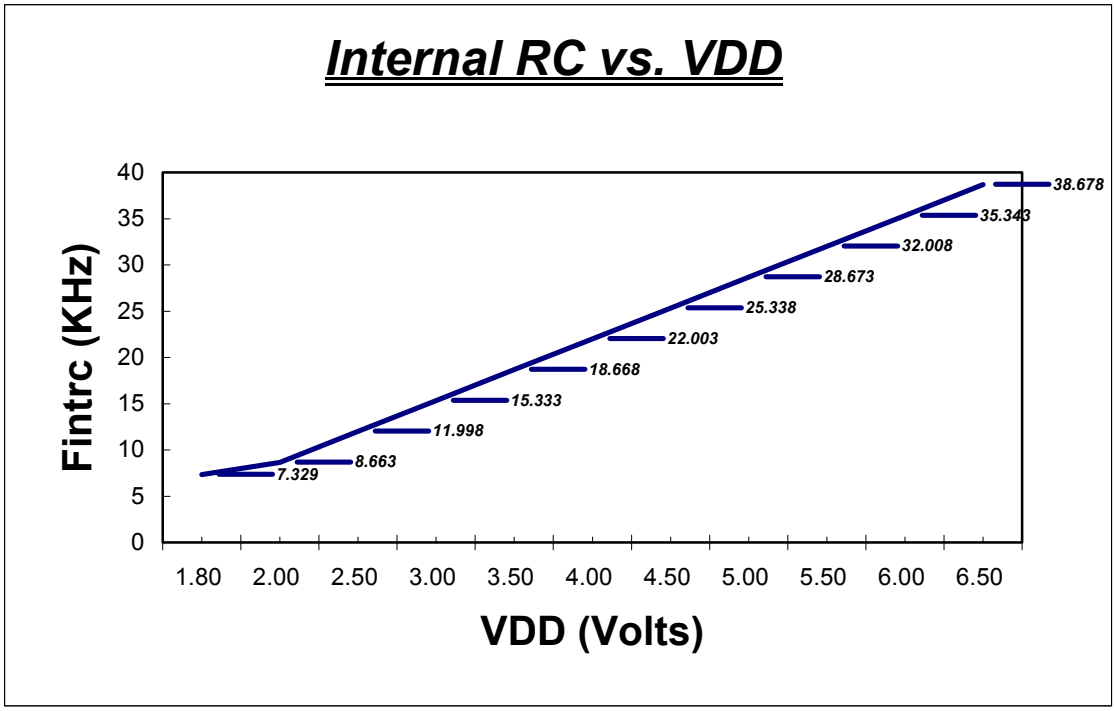


图 7－5　内部 RC 与 VDD 之间的函数关系

➲ **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          ;
        JMP       @B
```

7.5 系统模式

7.5.1 概述

SN8P1702A / SN8P1703A 能够在 4 种不同的模式之间转换，以减小功耗。

- 普通（高速）模式
- 低速模式
- 省电(睡眠)模式
- 绿色模式

实际应用中，用户可以利用 OSCM 寄存器调整系统的工作模式。 高速模式下，指令周期（Fcpu）为 Fosc / 4，低速模式下/3V 下，Fcpu 为 16KHz / 4。

7.5.2 普通模式

普通模式中，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。

7.5.3 低速模式

低速模式中，系统时钟源为内部低速时钟。设 CLKMD = 1，系统就进入内部低速模式。低速模式下的运行与普通模式的工作状态相似，仅时钟频率降低。系统在低速模式时可以转换到高速普通模式和省电模式。设 STPHX=1，可以停止外部高速振荡器，系统功耗也会降低。

7.5.4 绿色模式

绿色模式是一种低功耗模式，在绿色模式下，只有 TC0 仍然运行，而其他的硬件则停止工作。外部高速振荡器和内部低速振荡器仍然运行。设 CPUM1=1，CPUM0=0，系统就进入绿色模式。设 TC0GN=1（T0M 的第 1 位），使能 TC0 的绿色模式唤醒功能。系统可以由 TC0 和 P0 的电平变换来唤醒。

绿色模式提供定时唤醒功能。用户可通过设置 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，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，用户可根据情况选择合适的唤醒时间。在绿色模式下，其电源功耗为 5uA (3V)。

7.5.5 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，外部高速和低速振荡器均停止，P0、P1 的触发信号可将系统唤醒。

7.5.6 系统模式控制

系统模式框图

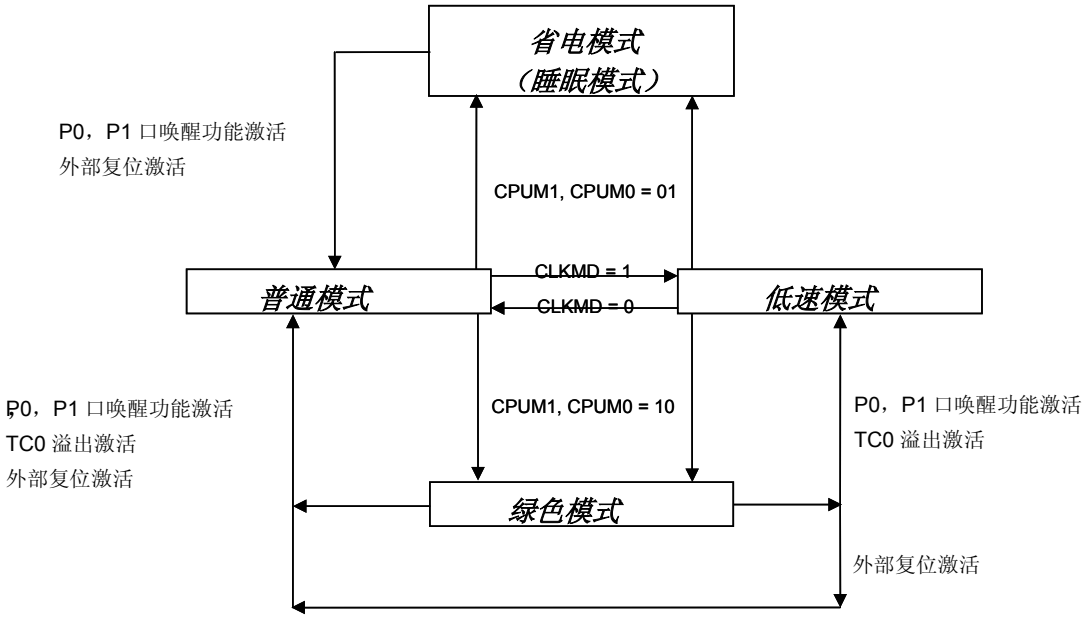


图 7－6　系统模式框图

工作模式	普通模式	低速模式	绿色模式	省电（睡眠）模式	说明
HX osc.	运行	STPHX	STPHX	停止	
LX osc.	运行	运行	运行	停止	
CPU 指令	执行	执行	停止	停止	
TC0	*有效	*有效	*有效	无效	*程序激活*
WDT	有效	有效	INT_16K_RC	无效	
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 复位信号	P0, P1, 复位信号	

表 7－1　操作模式说明

7.5.6.1 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，进入睡眠模式。
```

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
```

➢ **注：用户可以不用理会高速振荡器是否停止。**

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 以等待外部时钟稳定下来。

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

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

➲ **例：进入绿色模式，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 的初始值  = 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 定时器功能
        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 没有将系统从绿色模式中唤醒进入普通/低速模式的功能。**

7.6 唤醒时间

7.6.1 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停。外部高速振荡器重新启动需要的这一延迟时间称为唤醒时间。

有两种情况需要唤醒时间：一是从省电模式转换到普通模式，二是从低速模式转换到普通模式。对前一种情况，SN8P1702A / SN8P1703A 提供了 2048 个振荡时钟作为唤醒时间，后一种情况需要用户自行计算唤醒时间。

7.6.2 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，SN8P1702A / SN8P1703A 提供 2048 个外部高速振荡时钟作为唤醒时间，以使振荡电路达到稳定状态。唤醒时间结束后，系统进入通常模式，唤醒时间的计算如下：

唤醒时间 = 1/Fosc×2048 (sec)＋X'tal 稳定时间

X'tal 稳定时间决定于 X'tal 的类型，一般为 2~4ms。

➲ **例：在省电模式，系统由 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）模式下，具有唤醒功能的 P0 和 P1 都能将系统唤醒，P0 永远具有唤醒功能，而 P1 的唤醒功能受寄存器 P1W 控制。

P1W 初始值=xxxx xx00

0C0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1W	0	0	0	0	0	0	P11W	P10W
	-	-	-	-	-	-	W	W

Bit[1:0] **P11W, P10W:** Port1 唤醒功能控制位。
0 = 禁止 Port1 的唤醒功能
1 = 使能 Port1 的唤醒功能

7.6.3 外部唤醒触发控制

在 SN8P1702A / SN8P1703A 中由寄存器 PEDGE 控制唤醒触发的方向。

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]**：P0.0 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/计数器

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。在省电模式和绿色模式下，看门狗定时器将被禁止。

OSCM 初始值=0000 000x

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

Bit1 **STPHX**：外部高速振荡器控制位。
0 = 自由运行
1 = 停止

➢ **注：该位仅能控制外部高速振荡器，若 STPHX=1，内部低速 RC 振荡器仍然运行。**

Bit2 **CLKMD**：系统高/低速模式选择位。
0 = 普通（双重）模式
1 = 低速模式

Bit[4:3] **CPUM[1:0]**：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

Bit5 **Wdrate**：看门狗定时器速率选择位。
0 = Fcpu ÷ 2^{14}
1 = Fcpu ÷ 2^{8}

Bit6 **WDRST**：看门狗定时器复位位。
0 = 看门狗正常运行
1 = 将看门狗定时计数器清零

Bit7 **WTCKS**：看门狗定时器时钟源选择位
0 = Fcpu
1 =内部 RC 低速时钟

WTCKS	WTRATE	CLKMD	看门狗定时器溢出时间
0	0	0	1 / (Fcpu ÷ 2^{14} ÷ 16) = 293 ms, Fosc=3.58MHz
0	1	0	1 / (Fcpu ÷ 2^{8} ÷ 16) = 500 ms, Fosc=32768Hz
0	0	1	1 / (Fcpu ÷ 2^{14} ÷ 16) = 65.5s, Fosc=16KHz@3V
0	1	1	1 / (Fcpu ÷ 2^{8} ÷ 16) = 1s, Fosc=16KHz@3V
1	-	-	1 / (16K ÷ 512 ÷ 16) ~ 0.5s @3V

表 8－1 看门狗定时器溢出时间表

➢ **注：编译选项可控制是否运行看门狗定时器**

➲ **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 T0M 寄存器

T0M 初始值=xxxx 000x

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	-	-	-	-	TC1X8	TC0X8	TC0GN	-
	-	-	-	-	R/W	R/W	R/W	-

Bit3 **TC1X8**：TC1 定时器速率的 8 倍。
0 = 禁止
1 = 使能

Bit2 **TC0X8**：TC0 定时器速率的 8 倍。
0 = 禁止
1 = 使能

Bit0 **TC0GN**：开放 TC0 绿色模式的唤醒功能
0 = 禁止
1 = 使能

8.3 定时计数器 TC0

8.3.1 概述

定时/计数器 TC0 用来产生定时中断请求。它是具有自动重新装载功能的定时/计数器，主要由两部分组成：8 位自动装载寄存器 TC0R，用于保存计数参考值；8 位自动加 1 的计数器 TC0C。

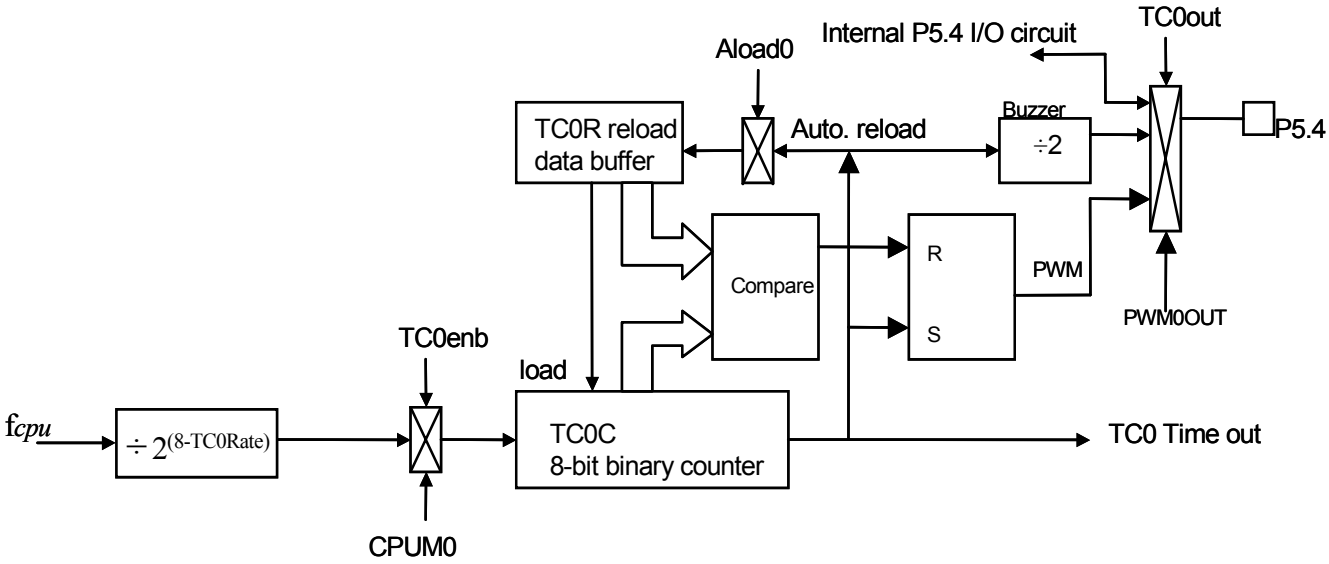


图 8－1　计数定时器 TC0 框图

TC0 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。
- **频率输出（BUZZER 输出）：**通过 BZ0 引脚 (P5.4)输出一个可选择的时钟频率。
- **PWM：**PWM 输出，由 PWM1OUT 位控制，通过 PWM0OUT 引脚 (P5.4)输出。

8.3.2 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 和 T0M 寄存器的 TC0X8 位，TC0 提供了 8 种可选择的速率。若 TC0X8=1，TC0 的速率是 TC0X8=0(初始值)的 8 倍，TC0M 的第 7 位 TC0ENB 是 TC0 定时器的开始控制位。

TC0M 初始值=0000 0000

0DAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0M	TC0ENB	TC0RATE2	TC0RATE1	TC0RATE0	TC0CKS	ALOAD0	TC0OUT	PWM0OUT
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit7 **TC0ENB**：TC0 计数器/BZ0/PWM0OUT 使能位
0 = 禁止
1 = 使能

Bit[6:4] TC0RATE[2:0]：TC0 时钟源选择位。TC0X8 是 T0M 寄存器的第 2 位。

TC0RATE [2:0]	TC0 时钟源	
	TC0X8 = 0	TC0X8 = 1
000	Fcpu/256 = Fosc/1024	Fosc/128
001	Fcpu/128 = Fosc/512	Fosc/64
…	…	…
110	Fcpu/4 = Fosc/16	Fosc/2
111	Fcpu/2 = Fosc/8	Fosc

注：Fcpu=Fosc/4

Bit3 **TC0CKS**：TC0 时钟源选择位
0 = Fcpu
1 =外部时钟来自 INT0/P0.0 引脚

Bit2 **ALOAD0**：TC0 自动装载功能控制位
0 = 无自动装载功能
1 = 自动装载

Bit1 **TC0OUT**：TC0 超时触发信号输出控制位
0 = 禁止 TC0 的信号输出，开放 P5.4 的基本输入/输出功能
1 = 开放 TC0 的信号输出，禁止 P5.4 的基本输入/输出功能 （自动禁止 PWM0OUT 功能）

Bit0 **PWM0OUT:** TC0 的 PWM 输出控制位
0 = 禁止 PWM 功能
1 = 开放 PWM 功能（TC0OUT 必须为 0）

- **注：当 TC0CKS=1 时，TC0 就是一个外部事件计数器。**

8.3.3 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。

TC0C 初始值=xxxx xxxx

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
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

8.3.4 TC0 溢出时间

TC0 的速率由 TC0Rate 和编译选项 TC0_Counter 共同确定，通过设置 TC0Rate 可以确定 TC0 的时钟频率，通过设置 TC0_Counter 可以使 TC0 为 8-bit、6-bit、5-bit 或 4-bit。

TC0C 的计算如下：

TC0C 初始值 = N－（TC0 中断间隔时间×输入时钟）

N 由编译选项 TC0_Counter 确定。

TC0_Counter	N	TC0C 的最大值
8-bit	256	255
6-bit	64	63
5-bit	32	31
4-bit	16	15

➢ **注：TC0C 的值必须小于等于 TC0 的最大值。**

➲ **例：设 Fosc = 3.58MHz，时间间隔为 10ms。**

TC0C(74H)=256－（10ms*fcpu/64）(TC0RATE=010，TC0_Counter=8-bit, TC0X8=0)

TC0C 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

➲ **例：设 Fosc=3.58MHz，时间间隔为 1.25ms。**

TC0C(74H)=256－（1.25ms* fcpu /64） (TC0RATE=010，TC0_Counter=8-bit, TC0X8=1)

TC0C 初始值 = 256 - (TC0 中断间隔时间×输入时钟)

= 256 - (1.25ms×3.58×10^6÷4÷32)

= 256 - (0.00125×3.58×10^6÷4÷32)

= 116

= 74H

➲ **例：设 Fosc=3.58MHz，时间间隔为 1ms。**

TC0C(74H)=256－（1ms* fcpu /64） (TC0RATE=010，TC0_Counter=6-bit, TC0X8=1)

TC0C 初始值 = 64 - (TC0 中断间隔时间 × 输入时钟)

= 64 - (1ms×3.58×10^6÷4÷64)

= 64 - (0.001×3.58×10^6÷4÷64)

= 64 - 14

= 32H

TC0_Counter=8-bit, TC0X8=0

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 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

TC0_Counter=6-bit, TC0X8=0

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/64	最大溢出间隔时间	单步间隔时间 = max/64
000	Fcpu/256	18.3 ms	286us	2000 ms	31.25 ms
001	Fcpu/128	9.15 ms	143us	1000 ms	15.63 ms
010	Fcpu/64	4.57 ms	71.5us	500 ms	7.8 ms
011	Fcpu/32	2.28 ms	35.8us	250 ms	3.9 ms
100	Fcpu/16	1.14 ms	17.9us	125 ms	1.95 ms
101	Fcpu/8	0.57 ms	8.94us	62.5 ms	0.98 ms
110	Fcpu/4	0.285 ms	4.47us	31.25 ms	0.49 ms
111	Fcpu/2	0.143 ms	2.23us	15.63 ms	0.24 ms

TC0_Counter=5-bit, TC0X8=0

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/32	最大溢出间隔时间	单步间隔时间 = max/32
000	Fcpu/256	9.15 ms	286us	1000 ms	31.25 ms
001	Fcpu/128	4.57 ms	143us	500 ms	15.63 ms
010	Fcpu/64	2.28 ms	71.5us	250 ms	7.8 ms
011	Fcpu/32	1.14 ms	35.8us	125 ms	3.9 ms
100	Fcpu/16	0.57 ms	17.9us	62.5 ms	1.95 ms
101	Fcpu/8	0.285 ms	8.94us	31.25 ms	0.98 ms
110	Fcpu/4	0.143 ms	4.47us	15.63 ms	0.49 ms
111	Fcpu/2	71.25 us	2.23us	7.81 ms	0.24 ms

TC0_Counter=4-bit, TC0X8=0

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/16	最大溢出间隔时间	单步间隔时间 = max/16
000	Fcpu/256	4.57 ms	286us	500 ms	31.25 ms
001	Fcpu/128	2.28 ms	143us	250 ms	15.63 ms
010	Fcpu/64	1.14 ms	71.5us	125 ms	7.8 ms
011	Fcpu/32	0.57 ms	35.8us	62.5 ms	3.9 ms
100	Fcpu/16	0.285 ms	17.9us	31.25 ms	1.95 ms
101	Fcpu/8	0.143 ms	8.94us	15.63 ms	0.98 ms
110	Fcpu/4	71.25 us	4.47us	7.81 ms	0.49 ms
111	Fcpu/2	35.63 us	2.23us	3.91 ms	0.24 ms

TC0_Counter=8-bit, TC0X8=1

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 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	Fosc/128	9.153 ms	35.754us	1000 ms	3.91 ms
001	Fosc/64	4.58 ms	17.877us	500 ms	1.95 ms
010	Fosc/32	2.29 ms	8.939us	250 ms	0.977 ms
011	Fosc/16	1.14 ms	4.470us	125 ms	0.488 ms
100	Fosc/8	0.57 ms	2.235us	62.5 ms	0.244 ms
101	Fosc/4	0.29 ms	1.117us	31.25 ms	0.122 ms
110	Fosc/2	0.14 ms	0.587us	15.63 ms	0.061 ms
111	Fosc	71.5 us	0.279us	7.81ms	0.03 ms

TC0_Counter=6-bit, TC0X8=1

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/64	最大溢出间隔时间	单步间隔时间 = max/64
000	Fosc/128	2.29 ms	35.754us	250 ms	3.91 ms
001	Fosc/64	1.14 ms	17.877us	125 ms	1.95 ms
010	Fosc/32	0.57 ms	8.939us	62.5 ms	0.977 ms
011	Fosc/16	0.29 ms	4.470us	31.25 ms	0.488 ms
100	Fosc/8	0.14 ms	2.235us	15.63 ms	0.244 ms
101	Fosc/4	71.5 us	1.117us	7.81ms	0.122 ms
110	Fosc/2	35.75 us	0.587us	3.905 ms	0.061 ms
111	Fosc	17.875 us	0.279us	1.953 ms	0.03 ms

TC0_Counter=5-bit, TC0X8=1

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/32	最大溢出间隔时间	单步间隔时间 = max/32
000	Fosc/128	1.14 ms	35.754us	125 ms	3.91 ms
001	Fosc/64	0.57 ms	17.877us	62.5 ms	1.95 ms
010	Fosc/32	0.29 ms	8.939us	31.25 ms	0.977 ms
011	Fosc/16	0.14 ms	4.470us	15.63 ms	0.488 ms
100	Fosc/8	71.5 us	2.235us	7.81ms	0.244 ms
101	Fosc/4	35.75 us	1.117us	3.905 ms	0.122 ms
110	Fosc/2	17.875 us	0.587us	1.953 ms	0.061 ms
111	Fosc	8.936 us	0.279us	0.976 ms	0.03 ms

TC0_Counter=4-bit, TC0X8=1

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/16	最大溢出间隔时间	单步间隔时间 = max/16
000	Fosc/128	0.57 ms	35.754us	62.5 ms	3.91 ms
001	Fosc/64	0.29 ms	17.877us	31.25 ms	1.95 ms
010	Fosc/32	0.14 ms	8.939us	15.63 ms	0.977 ms
011	Fosc/16	71.5 us	4.470us	7.81ms	0.488 ms
100	Fosc/8	35.75 us	2.235us	3.905 ms	0.244 ms
101	Fosc/4	17.875 us	1.117us	1.953 ms	0.122 ms
110	Fosc/2	8.936 us	0.587us	0.976 ms	0.061 ms
111	Fosc	4.468 us	0.279us	0.488 ms	0.03 ms

表 8-2 TC0 的时间列表

8.3.5 TC0R 自动装载寄存器

TC0R 是 8 位自动装载寄存器，用于 TC0OUT 和 PWM0OUT 的输出功能。应用 TC0OUT 时，必须使能 TC0R 寄存器并且要进行设置。TC0R 寄存器的主要功能如下：

- 存放自动装载值，当 TC0C 溢出时将其写入 TC0C 中；（ALOAD0=1）
- 存放 PWM0OUT 的占空比。

TC0R 初始值=xxxx xxxx

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
	W	W	W	W	W	W	W	W

TC0R 的计算如下：

TC0R 初始值=N－（TC0 间隔时间*输入时钟）

N 的值由编译选项 TC0_Counter 确定。

TC0_Counter	N	TC0R 的最大值
8-bit	256	255
6-bit	64	63
5-bit	32	31
4-bit	16	15

- **注：TC0R 的值必须小于等于 TC0R 的最大值。**
- **注：TC0R 是只写寄存器，不能执行 INCMS, DECMS 指令。**

8.3.6 TC0 操作流程

定时/计数器 TC0 的具体操作流程如下：

- 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 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，TC0C 无自动装载功能。（TC0_Counter=8-bit）**

```
B0BCLR      FTC0X8      ;
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：初始化 TC0M，TC0C 有自动装载功能。（TC0_Counter=8-bit）**

```
B0BCLR      FTC0X8      ; 设置 TC0=Fcpu/2 作为时钟源
B0BCLR      FTC0IEN     ; 禁止 TC0 中断
B0BCLR      FTC0ENB     ; 停止 TC0 计数
MOV         A,#20H      ;
B0MOV       TC0M,A      ; 设置 TC0 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
B0MOV       TC0R,A      ; 设置重新装载时间常数

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      ALOAD0      ; 使能 TC0R 自动重新装载功能
```

➲ 例：无自动装载功能的 **TC0** 中断服务程序。（**TC0_Counter=8-bit**）

```
        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     ; 保存 ACC

        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        ; 退出 TC0 中断服务程序
        .       .
EXIT_INT:
        B0MOV   A, PFLAGBUF
        B0MOV   PFLAG, A
        B0XCH   A, ACCBUF       ; 恢复 ACC

        RETI                    ; 中断返回
```

➲ 例：有自动装载功能的 **TC0** 中断服务程序。（**TC0_Counter=8-bit**）

```
        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
        B0MOV   A, PFLAG
        B0MOV   PFLAGBUF, A     ; 保护 ACC

        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
        .       .               ; 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                    ; 中断返回
```

8.3.7 TC0 时钟频率输出（BUZZER 输出）

TC0 定时/计数器有频率输出功能，通过设置 TC0 时钟频率，可以从 P5.4 输出时钟信号，同时 P5.4 的基本输入/输出功能会自动禁止。TC0 的输出信号是 2 分频的。由于 TC0 时钟源可以有多种选择，相应就可产生多种频率输出，此功能常用作蜂鸣器输出。

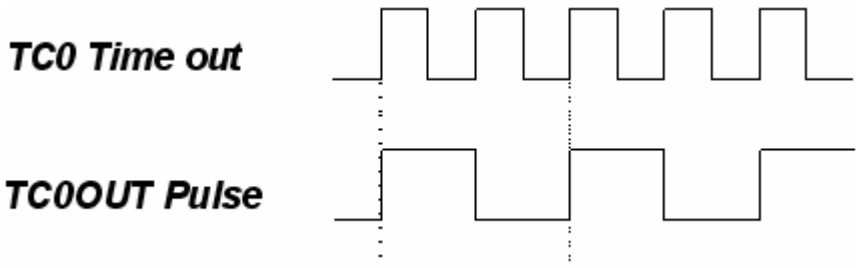


图 8－2　TC0OUT 脉冲频率

➲ **例：设置 TC0 的 TC0OUT(P5.4)输出，要求外部高速时钟 4MHz，TC0OUT 的频率 1KHz。因为 TC0OUT 是经过了 2 分频的，因此 TC0 的时钟设为 2KHz。TC0 时钟源来自外部振荡器，TC0 的速率为 Fcpu/4. TC0RATE2~TC0RATE1 = 110. TC0C = TC0R = 131. TC0X8 = 0, TC0_Counter=8-bit**

```
B0BCLR      FTC0X8          ; 设置 TC0X8 = 0
MOV         A,#01100000B
B0MOV       TC0M,A          ; 设置 TC0 的分频数为 Fcpu/4

MOV         A,#131          ; 设置自动重新装载的时间常数
B0MOV       TC0C,A
B0MOV       TC0R,A

B0BSET      FTC0OUT         ; 允许 TC0 输出到 P5.4 并禁止 P5.4 的 I/O 功能
B0BSET      FALOAD0         ; 使能 TC0 的自动装载功能
B0BSET      FTC0ENB         ; TC0 开始计数
```

8.3.8 TC0OUT 频率表

Fosc=4MHz, TC0 Rate=Fcpu/8, TC0_Counter=8-bit, TC0X8=0

TC0R	TC0OUT	TC0R	TC0OUT	TC0R	TC0OUT	TC0R	TC0OUT	TC0R	TC0OUT
0	0.2441	56	0.3125	112	0.4340	168	0.7102	224	1.9531
1	0.2451	57	0.3141	113	0.4371	169	0.7184	225	2.0161
2	0.2461	58	0.3157	114	0.4401	170	0.7267	226	2.0833
3	0.2470	59	0.3173	115	0.4433	171	0.7353	227	2.1552
4	0.2480	60	0.3189	116	0.4464	172	0.7440	228	2.2321
5	0.2490	61	0.3205	117	0.4496	173	0.7530	229	2.3148
6	0.2500	62	0.3222	118	0.4529	174	0.7622	230	2.4038
7	0.2510	63	0.3238	119	0.4562	175	0.7716	231	2.5000
8	0.2520	64	0.3255	120	0.4596	176	0.7813	232	2.6042
9	0.2530	65	0.3272	121	0.4630	177	0.7911	233	2.7174
10	0.2541	66	0.3289	122	0.4664	178	0.8013	234	2.8409
11	0.2551	67	0.3307	123	0.4699	179	0.8117	235	2.9762
12	0.2561	68	0.3324	124	0.4735	180	0.8224	236	3.1250
13	0.2572	69	0.3342	125	0.4771	181	0.8333	237	3.2895
14	0.2583	70	0.3360	126	0.4808	182	0.8446	238	3.4722
15	0.2593	71	0.3378	127	0.4845	183	0.8562	239	3.6765
16	0.2604	72	0.3397	128	0.4883	184	0.8681	240	3.9063
17	0.2615	73	0.3415	129	0.4921	185	0.8803	241	4.1667
18	0.2626	74	0.3434	130	0.4960	186	0.8929	242	4.4643
19	0.2637	75	0.3453	131	0.5000	187	0.9058	243	4.8077
20	0.2648	76	0.3472	132	0.5040	188	0.9191	244	5.2083
21	0.2660	77	0.3492	133	0.5081	189	0.9328	245	5.6818
22	0.2671	78	0.3511	134	0.5123	190	0.9470	246	6.2500
23	0.2682	79	0.3531	135	0.5165	191	0.9615	247	6.9444
24	0.2694	80	0.3551	136	0.5208	192	0.9766	248	7.8125
25	0.2706	81	0.3571	137	0.5252	193	0.9921	249	8.9286
26	0.2717	82	0.3592	138	0.5297	194	1.0081	250	10.4167
27	0.2729	83	0.3613	139	0.5342	195	1.0246	251	12.5000
28	0.2741	84	0.3634	140	0.5388	196	1.0417	252	15.6250
29	0.2753	85	0.3655	141	0.5435	197	1.0593	253	20.8333
30	0.2765	86	0.3676	142	0.5482	198	1.0776	254	31.2500
31	0.2778	87	0.3698	143	0.5531	199	1.0965	255	62.5000
32	0.2790	88	0.3720	144	0.5580	200	1.1161		
33	0.2803	89	0.3743	145	0.5631	201	1.1364		
34	0.2815	90	0.3765	146	0.5682	202	1.1574		
35	0.2828	91	0.3788	147	0.5734	203	1.1792		
36	0.2841	92	0.3811	148	0.5787	204	1.2019		
37	0.2854	93	0.3834	149	0.5841	205	1.2255		
38	0.2867	94	0.3858	150	0.5896	206	1.2500		
39	0.2880	95	0.3882	151	0.5952	207	1.2755		
40	0.2894	96	0.3906	152	0.6010	208	1.3021		
41	0.2907	97	0.3931	153	0.6068	209	1.3298		
42	0.2921	98	0.3956	154	0.6127	210	1.3587		
43	0.2934	99	0.3981	155	0.6188	211	1.3889		
44	0.2948	100	0.4006	156	0.6250	212	1.4205		
45	0.2962	101	0.4032	157	0.6313	213	1.4535		
46	0.2976	102	0.4058	158	0.6378	214	1.4881		
47	0.2990	103	0.4085	159	0.6443	215	1.5244		
48	0.3005	104	0.4112	160	0.6510	216	1.5625		
49	0.3019	105	0.4139	161	0.6579	217	1.6026		
50	0.3034	106	0.4167	162	0.6649	218	1.6447		
51	0.3049	107	0.4195	163	0.6720	219	1.6892		
52	0.3064	108	0.4223	164	0.6793	220	1.7361		
53	0.3079	109	0.4252	165	0.6868	221	1.7857		
54	0.3094	110	0.4281	166	0.6944	222	1.8382		
55	0.3109	111	0.4310	167	0.7022	223	1.8939		

表 8－3　TC0OUT 频率表（Fosc = 4MHz，TC0 Rate = Fcpu/8）

Fosc=16MHz, TC0 Rate=Fcpu/8, TC0_Counter=8-bit

TC0R	TC0OUT	TC0R	TC0OUT	TC0R	TC0OUT	TC0R	TC0OUT	TC0R	TC0OUT
0	0.9766	56	1.2500	112	1.7361	168	2.8409	224	7.8125
1	0.9804	57	1.2563	113	1.7483	169	2.8736	225	8.0645
2	0.9843	58	1.2626	114	1.7606	170	2.9070	226	8.3333
3	0.9881	59	1.2690	115	1.7730	171	2.9412	227	8.6207
4	0.9921	60	1.2755	116	1.7857	172	2.9762	228	8.9286
5	0.9960	61	1.2821	117	1.7986	173	3.0120	229	9.2593
6	1.0000	62	1.2887	118	1.8116	174	3.0488	230	9.6154
7	1.0040	63	1.2953	119	1.8248	175	3.0864	231	10.0000
8	1.0081	64	1.3021	120	1.8382	176	3.1250	232	10.4167
9	1.0121	65	1.3089	121	1.8519	177	3.1646	233	10.8696
10	1.0163	66	1.3158	122	1.8657	178	3.2051	234	11.3636
11	1.0204	67	1.3228	123	1.8797	179	3.2468	235	11.9048
12	1.0246	68	1.3298	124	1.8939	180	3.2895	236	12.5000
13	1.0288	69	1.3369	125	1.9084	181	3.3333	237	13.1579
14	1.0331	70	1.3441	126	1.9231	182	3.3784	238	13.8889
15	1.0373	71	1.3514	127	1.9380	183	3.4247	239	14.7059
16	1.0417	72	1.3587	128	1.9531	184	3.4722	240	15.6250
17	1.0460	73	1.3661	129	1.9685	185	3.5211	241	16.6667
18	1.0504	74	1.3736	130	1.9841	186	3.5714	242	17.8571
19	1.0549	75	1.3812	131	2.0000	187	3.6232	243	19.2308
20	1.0593	76	1.3889	132	2.0161	188	3.6765	244	20.8333
21	1.0638	77	1.3966	133	2.0325	189	3.7313	245	22.7273
22	1.0684	78	1.4045	134	2.0492	190	3.7879	246	25.0000
23	1.0730	79	1.4124	135	2.0661	191	3.8462	247	27.7778
24	1.0776	80	1.4205	136	2.0833	192	3.9063	248	31.2500
25	1.0823	81	1.4286	137	2.1008	193	3.9683	249	35.7143
26	1.0870	82	1.4368	138	2.1186	194	4.0323	250	41.6667
27	1.0917	83	1.4451	139	2.1368	195	4.0984	251	50.0000
28	1.0965	84	1.4535	140	2.1552	196	4.1667	252	62.5000
29	1.1013	85	1.4620	141	2.1739	197	4.2373	253	83.3333
30	1.1062	86	1.4706	142	2.1930	198	4.3103	254	125.0000
31	1.1111	87	1.4793	143	2.2124	199	4.3860	255	250.0000
32	1.1161	88	1.4881	144	2.2321	200	4.4643		
33	1.1211	89	1.4970	145	2.2523	201	4.5455		
34	1.1261	90	1.5060	146	2.2727	202	4.6296		
35	1.1312	91	1.5152	147	2.2936	203	4.7170		
36	1.1364	92	1.5244	148	2.3148	204	4.8077		
37	1.1416	93	1.5337	149	2.3364	205	4.9020		
38	1.1468	94	1.5432	150	2.3585	206	5.0000		
39	1.1521	95	1.5528	151	2.3810	207	5.1020		
40	1.1574	96	1.5625	152	2.4038	208	5.2083		
41	1.1628	97	1.5723	153	2.4272	209	5.3191		
42	1.1682	98	1.5823	154	2.4510	210	5.4348		
43	1.1737	99	1.5924	155	2.4752	211	5.5556		
44	1.1792	100	1.6026	156	2.5000	212	5.6818		
45	1.1848	101	1.6129	157	2.5253	213	5.8140		
46	1.1905	102	1.6234	158	2.5510	214	5.9524		
47	1.1962	103	1.6340	159	2.5773	215	6.0976		
48	1.2019	104	1.6447	160	2.6042	216	6.2500		
49	1.2077	105	1.6556	161	2.6316	217	6.4103		
50	1.2136	106	1.6667	162	2.6596	218	6.5789		
51	1.2195	107	1.6779	163	2.6882	219	6.7568		
52	1.2255	108	1.6892	164	2.7174	220	6.9444		
53	1.2315	109	1.7007	165	2.7473	221	7.1429		
54	1.2376	110	1.7123	166	2.7778	222	7.3529		
55	1.2438	111	1.7241	167	2.8090	223	7.5758		

表 8－4　TC0OUT 频率表（Fosc =16MHz，TC0 Rate = Fcpu/8）

8.4 定时/计数器 TC1

8.4.1 概述

定时/计数器 TC1 用来产生定时中断请求。TC1 具有自动重新装载功能，主要有两部分组成：8 位自动装载寄存器 TC1R，用于保存计数参考值；8 位自动加 1 的计数器 TC1C。

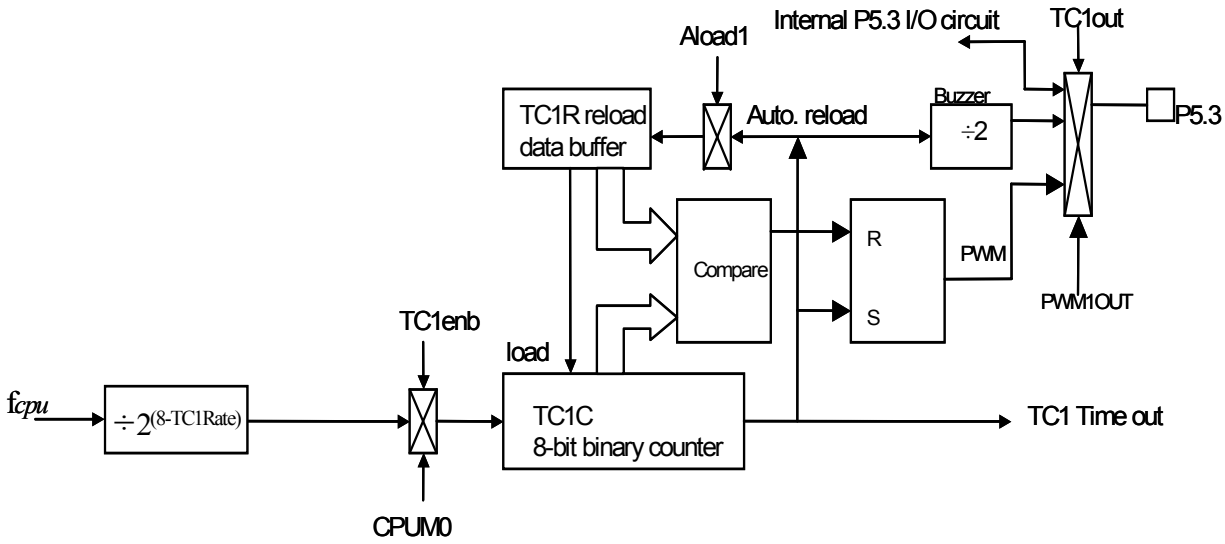


图 8－3　定时器 TC1 框图

TC1 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
- **频率输出（BUZZER 输出）：**通过 BZ1 引脚（P5.3）输出一个可选择的时钟频率。
- **PWM：**PWM 输出，由 PWM1OUT 位控制，通过 PWM1OUT 引脚 (P5.3)输出。

8.4.2 TC1M 模式寄存器

TC1M 为 8 位可读/写模式控制寄存器。通过载入不同的值，用户可以在执行程序的过程中动态的调整定时器的时钟频率。

通过设置 TC1 的 TC1RATE0~TC1RATE2 和 T0M 寄存器的 TC1X8 位，TC1 提供了 8 种可选择的速率。若 TC1X8=1，TC1 的速率是 TC1X8=0(初始值)的 8 倍，TC1M 的第 7 位 TC1ENB 是 TC1 定时器的开始控制位。

TC1M 初始值=0000 x000

0DCH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC1M	TC1ENB	TC1RATE2	TC1RATE1	TC1RATE0	0	ALOAD1	TC1OUT	PWM1OUT
	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W

Bit7 **TC1ENB:** TC1 计数器/BZ1/PWM1OUT 使能位
0 = 禁止
1 = 开放

Bit[6:4] **TC1RATE[2:0]:** TC1 时钟源选择位。TC1X8 是 TC1M 寄存器的第三位。

TC1RATE [2:0]	TC1 时钟源	
	TC1X8 = 0	TC1X8 = 1
000	Fcpu/256 = Fosc/1024	Fosc/128
001	Fcpu/128 = Fosc/512	Fosc/64
…	…	…
110	Fcpu/4 = Fosc/16	Fosc/2
111	Fcpu/2 = Fosc/8	Fosc

注：Fcpu = Fosc / 4

Bit2 **ALOAD1:** TC1 自动装载功能控制位
0 = 无自动装载功能
1 = 自动装载

Bit1 **TC1OUT:** TC1 超时触发信号输出控制位
0 = 禁止 TC1 的信号输出，开放 P5.3 的基本输入/输出功能。
1 = 开放 TC1 的信号输出，禁止 P5.3 的基本输入/输出功能。（自动禁止 PWM1OUT 功能）

Bit0 **PWM1OUT:** TC1 PWM 输出控制位
0 = 禁止 PWM 输出
1 = 开放 PWM 输出（TC1OUT 控制位必须为 0）

- **注：由于 SN8P1702A 和 SN8P1703A 没有 P0.1 作为事件计数器的时钟输入，TC1 不支持事件计数器模式。**
- **注：bit3 必须为“0”。**

8.4.3 TC1C 计数寄存器

TC1C 是一个 8 位定时计数器，只要 TC1ENB 被置“1”就开启定时器。TC1C 是个加 1 计数器，时钟源频率由 TC1RATE0~TC1RATE2 决定。当 TC1C 增至“0FFH”后，若再加 1 就会回到“00H”，发生溢出。此时，TC1 的中断请求标志被置为“1”，如果 TC1 中断又同时被使能（TC1IEN = 1），那么系统将执行 TC1 的中断服务程序。

TC1C 初始值=xxxx xxxx

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

8.4.4 TC1 溢出时间

TC1 的速率由 TC1Rate 和编译选项 TC1_Counter 共同确定，通过设置 TC1Rate 可以确定 TC1 的时钟频率，通过设置 TC1_Counter 可以使 TC1 为 8-bit、6-bit、5-bit 或 4-bit。

TC1C 的计算如下：

TC1C 初始值=N－（TC1 中断间隔时间*输入时钟）

N 的值由编译选项 TC1_Counter 确定。

TC1_Counter	N	TC1C 的最大值
8-bit	256	255
6-bit	64	63
5-bit	32	31
4-bit	16	15

➢ **注：TC1C 的值要小于或等于 TC1C 的最大值。**

➲ **例：设中断间隔时间为 10ms，Fosc = 3.58MHz.**

TC1C(74H) = 256 -（10ms*fcpu/64） (TC1RATE = 010, TC1_Counter = 8-bit, TC1X8 = 1)

TC1C 初始值 = 256 -（TC1 中断间隔时间 × 输入时钟）

= 256 - (10ms×3.58×10^6÷4÷64)

= 256 - (10^{-2}×3.58×10^6÷4÷64)

= 116

= 74H

➲ **例：设中断间隔时间为 1.25ms，Fosc = 3.58MHz.**

TC1C(74H) = 256 -（1.25ms*fcpu/64） (TC1RATE = 010, TC1_Counter = 8-bit, TC1X8 = 1)

TC1C 初始值 = 256 -（TC1 中断间隔时间×输入时钟）

= 256 - (1.25ms×3.58×10^6÷4÷32)

= 256 - (0.00125×3.58×10^6÷4÷32)

= 116

= 74H

➲ **例：设中断间隔时间为 1ms，Fosc = 3.58MHz.**

TC1C(32H) = 64 -（1ms*fcpu/64） (TC1RATE = 010, TC1_Counter = 8-bit, TC1X8 = 0)

TC1C 初始值 = 64 - (TC1 中断间隔时间×输入时钟)

= 64 - (1ms×3.58×10^6÷4÷64)

= 64 - (0.001×3.58×10^6÷4÷64)

= 64-16

= 32H

TC1_Counter=8-bit, TC1X8=0

TC1RATE	TC1CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 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

TC1_Counter=6-bit, TC1X8=0

TC1RATE	TC1CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间= max/64	最大溢出间隔时间	单步间隔时间= max/64
000	Fcpu/256	18.3 ms	286us	2000 ms	31.25 ms
001	Fcpu/128	9.15 ms	143us	1000 ms	15.63 ms
010	Fcpu/64	4.57 ms	71.5us	500 ms	7.8 ms
011	Fcpu/32	2.28 ms	35.8us	250 ms	3.9 ms
100	Fcpu/16	1.14 ms	17.9us	125 ms	1.95 ms
101	Fcpu/8	0.57 ms	8.94us	62.5 ms	0.98 ms
110	Fcpu/4	0.285 ms	4.47us	31.25 ms	0.49 ms
111	Fcpu/2	0.143 ms	2.23us	15.63 ms	0.24 ms

TC1_Counter=5-bit, TC1X8=0

TC1RATE	TC1CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间= max/32	最大溢出间隔时间	单步间隔时间= max/32
000	Fcpu/256	9.15 ms	286us	1000 ms	31.25 ms
001	Fcpu/128	4.57 ms	143us	500 ms	15.63 ms
010	Fcpu/64	2.28 ms	71.5us	250 ms	7.8 ms
011	Fcpu/32	1.14 ms	35.8us	125 ms	3.9 ms
100	Fcpu/16	0.57 ms	17.9us	62.5 ms	1.95 ms
101	Fcpu/8	0.285 ms	8.94us	31.25 ms	0.98 ms
110	Fcpu/4	0.143 ms	4.47us	15.63 ms	0.49 ms
111	Fcpu/2	71.25 us	2.23us	7.81 ms	0.24 ms

TC1_Counter=4-bit, TC1X8=0

TC1RATE	TC1CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间= max/16	最大溢出间隔时间	单步间隔时间= max/16
000	Fcpu/256	4.57 ms	286us	500 ms	31.25 ms
001	Fcpu/128	2.28 ms	143us	250 ms	15.63 ms
010	Fcpu/64	1.14 ms	71.5us	125 ms	7.8 ms
011	Fcpu/32	0.57 ms	35.8us	62.5 ms	3.9 ms
100	Fcpu/16	0.285 ms	17.9us	31.25 ms	1.95 ms
101	Fcpu/8	0.143 ms	8.94us	15.63 ms	0.98 ms
110	Fcpu/4	71.25 us	4.47us	7.81 ms	0.49 ms
111	Fcpu/2	35.63 us	2.23us	3.91 ms	0.24 ms

TC1_Counter=8-bit, TC1X8=1

TC1RATE	TC1CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 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	Fosc/128	9.153 ms	35.754us	1000 ms	3.91 ms
001	Fosc/64	4.58 ms	17.877us	500 ms	1.95 ms
010	Fosc/32	2.29 ms	8.939us	250 ms	0.977 ms
011	Fosc/16	1.14 ms	4.470us	125 ms	0.488 ms
100	Fosc/8	0.57 ms	2.235us	62.5 ms	0.244 ms
101	Fosc/4	0.29 ms	1.117us	31.25 ms	0.122 ms
110	Fosc/2	0.14 ms	0.587us	15.63 ms	0.061 ms
111	Fosc	71.5 us	0.279us	7.81ms	0.03 ms

TC1_Counter=6-bit, TC1X8=1

TC1RATE	TC1CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间= max/64	最大溢出间隔时间	单步间隔时间= max/64
000	Fosc/128	2.29 ms	35.754us	250 ms	3.91 ms
001	Fosc/64	1.14 ms	17.877us	125 ms	1.95 ms
010	Fosc/32	0.57 ms	8.939us	62.5 ms	0.977 ms
011	Fosc/16	0.29 ms	4.470us	31.25 ms	0.488 ms
100	Fosc/8	0.14 ms	2.235us	15.63 ms	0.244 ms
101	Fosc/4	71.5 us	1.117us	7.81ms	0.122 ms
110	Fosc/2	35.75 us	0.587us	3.905 ms	0.061 ms
111	Fosc	17.875 us	0.279us	1.953 ms	0.03 ms

TC1_Counter=5-bit, TC1X8=1

TC1RATE	TC1CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间= max/32	最大溢出间隔时间	单步间隔时间= max/32
000	Fosc/128	1.14 ms	35.754us	125 ms	3.91 ms
001	Fosc/64	0.57 ms	17.877us	62.5 ms	1.95 ms
010	Fosc/32	0.29 ms	8.939us	31.25 ms	0.977 ms
011	Fosc/16	0.14 ms	4.470us	15.63 ms	0.488 ms
100	Fosc/8	71.5 us	2.235us	7.81ms	0.244 ms
101	Fosc/4	35.75 us	1.117us	3.905 ms	0.122 ms
110	Fosc/2	17.875 us	0.587us	1.953 ms	0.061 ms
111	Fosc	8.936 us	0.279us	0.976 ms	0.03 ms

TC1_Counter=4-bit, TC1X8=1

TC1RATE	TC1CLOCK	高速模式 (Fcpu = 3.58MHz / 4)		低速模式 (Fosc = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间= max/16	最大溢出间隔时间	单步间隔时间= max/16
000	Fosc/128	0.57 ms	35.754us	62.5 ms	3.91 ms
001	Fosc/64	0.29 ms	17.877us	31.25 ms	1.95 ms
010	Fosc/32	0.14 ms	8.939us	15.63 ms	0.977 ms
011	Fosc/16	71.5 us	4.470us	7.81ms	0.488 ms
100	Fosc/8	35.75 us	2.235us	3.905 ms	0.244 ms
101	Fosc/4	17.875 us	1.117us	1.953 ms	0.122 ms
110	Fosc/2	8.936 us	0.587us	0.976 ms	0.061 ms
111	Fosc	4.468 us	0.279us	0.488 ms	0.03 ms

表 8-5 TC1 的时间列表

8.4.5 TC1R 自动装载寄存器

TC1R 是 8 位的自动装载寄存器，用于 TC1OUT 和 PWM1OUT 的输出功能。在 TC1OUT 功能下，用户必须使能 TC1R 并要进行设置。其主要功能如下：

- 存放自动装载值，当 TC1C 溢出时将其写入 TC1C 中（ALOAD1 = 1）；
- 存放 PWM1OUT 的占空比。

TC1R 初始值=xxxx xxxx

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

TC1R 的计算如下：

TC1R 初始值=N -（TC1 中断间隔时间*输入时钟）

N 的值由编译选项 TC1_Counter 确定。

TC1_Counter	N	TC1R 的最大值
8-bit	256	255
6-bit	64	63
5-bit	32	31
4-bit	16	15

- **注：TC1R 的值必须小于或等于 TC1R 的最大值**
- **注：TC1R 是只写寄存器，不能执行 INCMS, DECMS 指令。**

8.4.6 TC1 操作流程

定时/计数器 TC1 的工作流程如下：

- 置 TC1C 初始值，设置定时器中断间隔时间；
- TC1ENB 置为“1”，TC1 开始计数；
- 根据 TC1M 中选择的时钟源频率，每个时钟 TC1C 加 1；
- 如果 TC1 从“FFH”增至“00H”，TC1 溢出；
- 当 TC1 发生溢出，TC1IRQ 通过硬件设为“1”；
- 执行中断服务程序；
- 用户复位 TC1C，重新开始 TC1 定时器操作。

➲ **例：初始化 TC1M 和 TC1C，无自动装载功能 (TC1_Counter=8-bit, TC1X8=0)**

```
B0BCLR      FTC1X8          ;
B0BCLR      FTC1IEN         ; 禁止 TC1 中断
B0BCLR      FTC1ENB         ; 停止 TC1 计数
MOV         A,#20H          ;
B0MOV       TC1M,A          ; 设置 TC1 的分频数为 Fcpu / 64
MOV         A,#74H          ; 设置 TC1C 初始值 = 74H
B0MOV       TC1C,A          ; 定时时间 = 10 ms

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

➲ **例：初始化 TC1M 和 TC1C，有自动装载功能 (TC1_Counter=8-bit, TC1X8=0)**

```
B0BCLR      FTC1X8          ; 选择时钟源为 TC1=Fcpu/2
B0BCLR      FTC1IEN         ; 禁止 TC1 中断
B0BCLR      FTC1ENB         ; 停止 TC1 计数
MOV         A,#20H          ;
B0MOV       TC1M,A          ; 设置 TC1 的分频数为 Fcpu / 64
MOV         A,#74H          ; 设置 TC1C 初始值 = 74H
B0MOV       TC1C,A          ; 定时时间 = 10 ms
B0MOV       TC1R,A          ; 设置重新装载时间常数

B0BSET      FTC1IEN         ; 使能 TC1 中断
B0BCLR      FTC1IRQ         ; 清 TC1 中断请求
B0BSET      FTC1ENB         ; 开始 TC1 计数
B0BSET      ALOAD1          ; 使能 TC1R 自动重新装载功能
```

➲ **例: 无自动装载功能的 TC1 中断服务程序 (TC1_Counter=8-bit, TC1X8=0)**

```
        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     ; 保护 ACC

        B0BTS1  FTC1IRQ         ; 检查是否是 TC1IRQ 中断请求
        JMP     EXIT_INT        ; TC1IRQ = 0，退出中断

        B0BCLR  FTC1IRQ         ; 清 TC1IRQ
        MOV     A,#74H          ; 重新设置 TC1C
        B0MOV   TC1C,A
        .       .               ; TC1 中断服务程序
        .       .
        JMP     EXIT_INT        ; TC1 中断服务程序结束
        .       .
EXIT_INT:
        B0MOV   A, PFLAGBUF
        B0MOV   PFLAG, A
        B0XCH   A, ACCBUF       ; 恢复 ACC
        RETI                    ; 中断返回
```

➲ **例:有自动装载功能的 TC1 中断服务程序(TC1_Counter=8-bit, TC1X8=0)**

```
        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     ; 保护 ACC

        B0BTS1  FTC1IRQ         ; 检查是否是 TC1IRQ 中断请求
        JMP     EXIT_INT        ; TC1IRQ = 0，退出中断

        B0BCLR  FTC1IRQ         ; 清 TC1IRQ
        .       .               ; TC1 中断服务程序
        .       .
        JMP     EXIT_INT        ; TC1 中断服务程序结束
        .       .
EXIT_INT:
        B0MOV   A, PFLAGBUF
        B0MOV   PFLAG, A
        B0XCH   A, ACCBUF       ; 恢复 ACC
        RETI                    ; 中断返回
```

8.4.7 TC1 时钟频率输出（BUZZER 输出）

TC1 定时/计数器有频率输出功能，通过设置 TC1 时钟频率，可以从 P5.3 输出时钟信号，同时 P5.3 的基本输入/输出功能会自动禁止。TC1 的输出信号是 2 分频的。由于 TC1 时钟源可以有多种选择，相应就可产生多种频率输出，此功能常用作蜂鸣器输出。

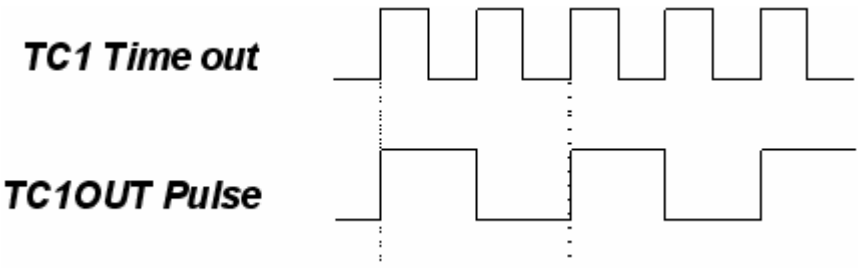


图 8－4　TC1OUT 脉冲频率

➲ **例：设置 TC1 的 TC1OUT(P5.3)输出，要求外部高速时钟 4MHz，TC1OUT 的频率 1KHz。因为 TC1OUT 是经过了 2 分频的，因此 TC1 的时钟设为 2KHz。TC1 时钟源来自外部振荡器，TC1 的速率为 Fcpu/4。所以 TC1RATE2~TC1RATE1 = 110. TC1C = TC1R = 131. TC1_Counter=8-bit, TC1X8=0**

```
B0BCLR      FTC1X8          ; 设置 TC1X8 = 0
MOV         A,#01100000B
B0MOV       TC1M,A          ; 设置 TC1 的分频数为 Fcpu/4

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

B0BSET      FTC1OUT         ; 使能 TC1 频率输出（P5.3），同时禁止 P5.3 的 I/O 功能
B0BSET      FALOAD1         ; 使能 TC1 的自动重新装载功能
B0BSET      FTC1ENB         ; TC1 开始计数
```

8.5 PWM

8.5.1 概述

PWM 功能使用的时基为 TC0 或 TC1，产生的 PWM 信号输出到 PWM0OUT 引脚（P5.4）或 PWM1OUT（P5.3）。当编译选项 TC0/TC1_Counter=8-bit，计数器共计数 256（0～255）次。8 位计数器的值和 TC0R/TC1R 的值作比较，当它们的值相等时，PWM 输出低电平；当计数器为 0 时，PWM 为高电平。下表中列出了 PWM 的占空比。

例如，TC0_Counter=8-bit 时，在开始的 256 个输入时钟内，PWM 输出无效。当 TC0C/TC1C 的计数值增至 FFH 并清零时，PWM 开始输出高电平。其脉宽比（占空比）由 TC0R/TC1R 的值决定，8 位 PWM 数据寄存器 TC0R/TC1R 是只写寄存器。不同的 TC0_Counter / TC0_Counter 导致不同的 PWM 占空比，因此用户要通过选择不同的 TC0_Counter / TC0_Counter 来产生不同的 PWM 输出。

向参考寄存器 TC0R/TC1R 中写入 00H 可以使 PWM 输出保持在低电平；在 PWM 运行时，可以通过调整 TC0R/TC1R 可改变 PWM 输出的占空比。

TC0X8/TC1X8	PWM0 频率	PWM1 频率
0	$Fosc/(2^{10-TC0RATE})/N$	$Fosc/(2^{10-TC1RATE})/N$
1	$Fosc/(2^{7-TC0RATE})/N$	$Fosc/(2^{7-TC1RATE})/N$

N 的值由 TC0_Counter / TC1_Counter 决定

TC0_Counter/TC1_Counter	N	PWM 占空比
8-bit	256	0/256 ~ 255/256
6-bit	64	0/64 ~ 63/64
5-bit	32	0/32 ~ 31/32
4-bit	16	0/16 ~ 15/16

表 8-6 PWM 频率的计算公式

TC0X8 TC1X8	TC0_Counter TC1_Counter	TC0/TC1 溢出边界	PWM 占空比	最大 PWM 频率 (Fosc = 4MHz)	注
0	8-bit	FFh to 00h	0/256 ~ 255/256	1.953125K	每计数 256 次溢出
0	6-bit	3Fh to 40h	0/64 ~ 63/64	7.8125K	每计数 64 次溢出
0	5-bit	1Fh to 20h	0/32 ~ 31/32	15.625K	每计数 32 次溢出
0	4-bit	0Fh to 10h	0/16 ~ 15/16	31.25K	每计数 16 次溢出
1	8-bit	FFh to 00h	0/256 ~ 255/256	15.625	每计数 256 次溢出
1	6-bit	3Fh to 40h	0/64 ~ 63/64	62.5K	每计数 64 次溢出
1	5-bit	1Fh to 20h	0/32 ~ 31/32	125K	每计数 32 次溢出
1	4-bit	0Fh to 10h	0/16 ~ 15/16	250K	每计数 16 次溢出

表 8-7 PWM 频率的最大值（TC0RATE / RC1RATE =111）

参考寄存器的值(TC0R/TC1R)	TC0/1_Counter=8-bit 占空比	TC0/1_Counter=6-bit 占空比	TC0/1_Counter=5-bit 占空比	TC0/1_Counter=4-bit 占空比
0000 0000	0/256	0/64	0/32	0/16
0000 0001	1/256	1/64	1/32	1/16
0000 0010	2/256	2/64	2/32	2/16
...	...	...	...	...
0000 1110	14/256	14/64	14/32	14/16
0000 1111	15/256	15/64	15/32	15/16
0001 0000	16/256.	16/64	16/32	N/A
...	...	...	...	N/A
0001 1110	30/256	30/64	30/32	N/A
0001 1111	31/256	31/64	31/32	N/A
0010 0000	32/256.	32/64	N/A	N/A
...	...	...	N/A	N/A
0011 1110	62/256	62/64	N/A	N/A
0011 1111	63/256	63/64	N/A	N/A
0100 0000	64/256.	N/A	N/A	N/A
...	...	N/A	N/A	N/A
1111 1110	254/256	N/A	N/A	N/A
1111 1111	255/256	N/A	N/A	N/A

表 8-8 PWM 占空比列表

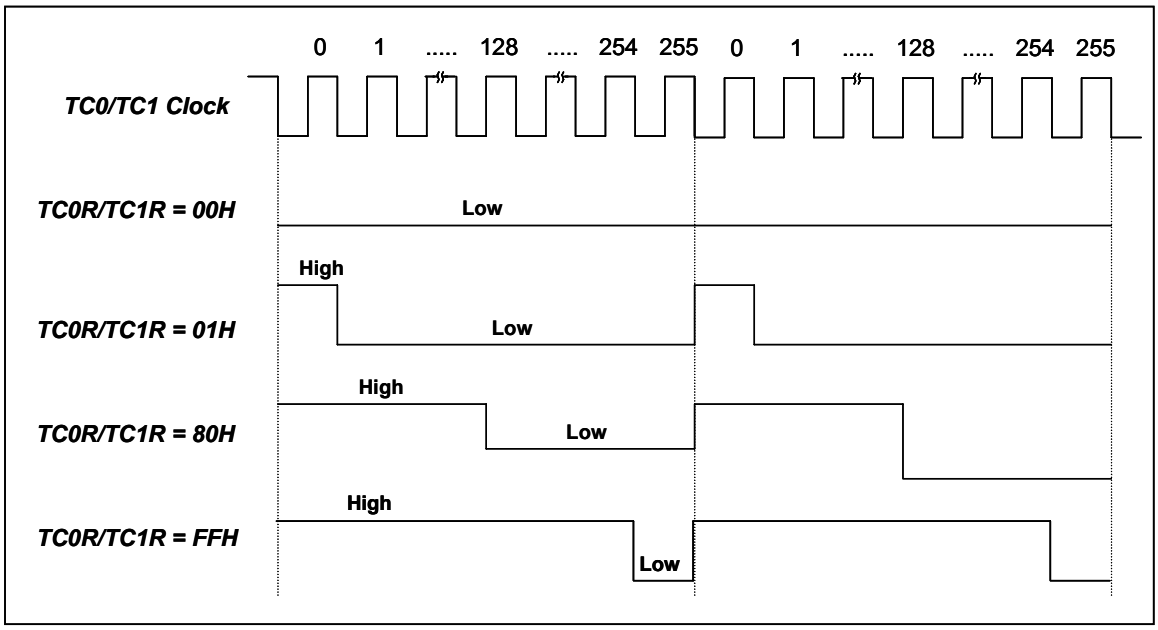


图 8-5　不同的 TC0R / TC1R 的 PWM 输出 (TC0 / TC1_Counter=8-bit)

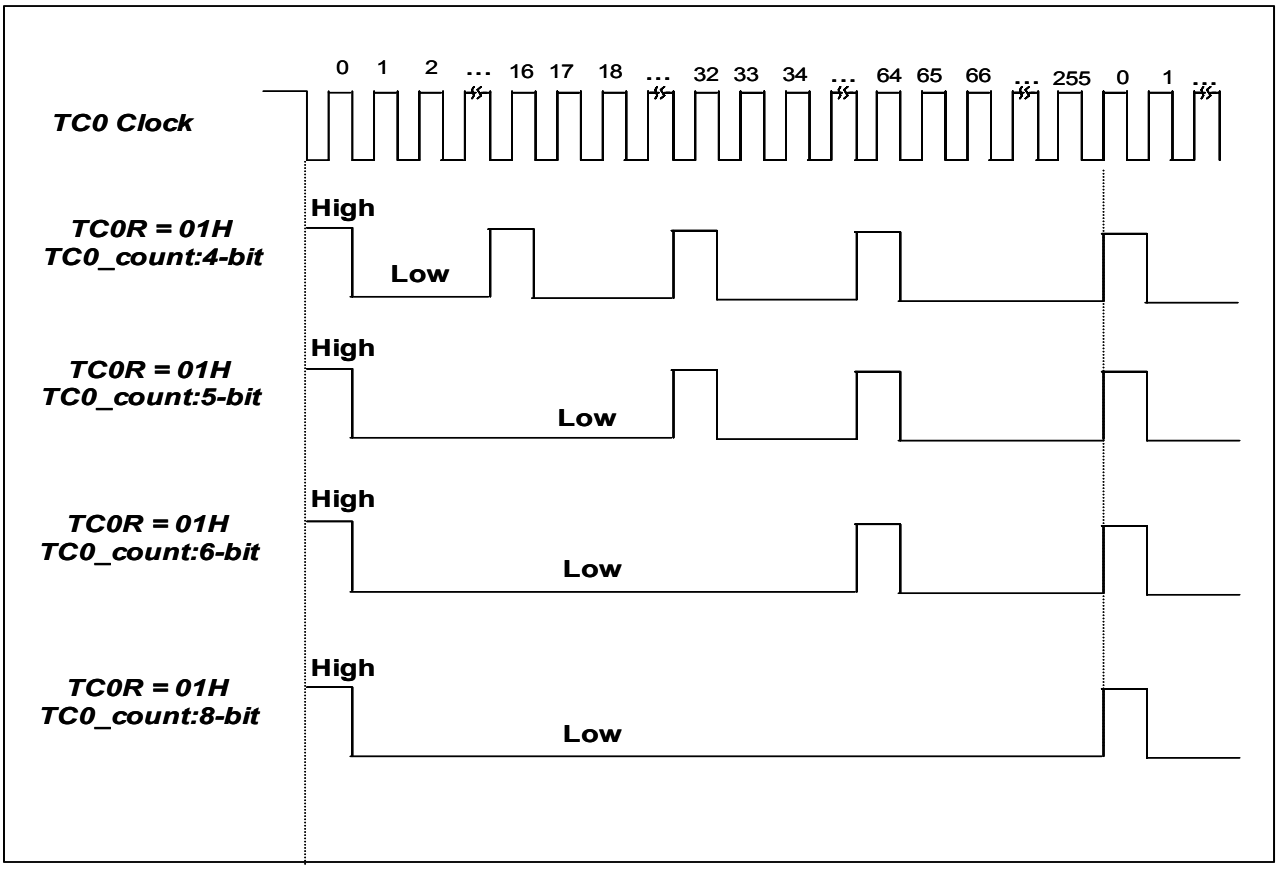


图 8-6　不同的 TC0_Counter 的 PWM 输出

8.5.2 PWM 操作举例

➲ **例：设置 PWM0 的输出 PWM0OUT(P5.4)，其中，外部高速振荡器时钟 4MHz，PWM 输出占空比为 30/256。PWM 的输出频率是 1KHz。PWM 的时钟源来自外部振荡器，TC0 的速率是 Fcpu/4。TC0RATE2~TC0RATE1 = 110. TC0C = TC0R = 30，TC0X8 =0, TC0_Counter=8-bit。**

```
B0BCLR      FTC0X8
MOV         A,#01100000B
B0MOV       TC0M,A          ; 设置 TC0 的分频数为 Fcpu/4
MOV         A,#0x00         ; TC0 的初始时间
B0MOV       TC0C,A
MOV         A,#30           ; 设置 PWM 的占空比为 30/256

B0MOV       TC0R,A

B0BCLR      FTC0OUT         ; 禁止 TC0OUT 功能
B0BSET      FPWM0OUT        ; 使能 PWM0 输出到 P5.4 并禁止 P5.4 的 I/O 功能
B0BSET      FTC0ENB         ; TC0 开始计数
```

➢ **注 1：TC0R 和 TC1R 是只写寄存器，不能执行 INCMS, DECMS 指令。**
➢ **注 2：初始化 TC0C 时，要保证第一个占空比正确。开放 TC0 后，不能调整 TC0R 的值，否则 PWM 输出的占空比会出错。**

➲ **例：调整 TC0R/TC1R 的值。**

```
MOV         A, #30H         ; 赋予一个立即数
B0MOV       TC0R, A

INCMS       BUF0            ; 从 BUF0 得到一个新的 TC0R 值
B0MOV       A, BUF0
B0MOV       TC0R, A
```

➢ **注 3：PWM 的占空比变化的时候，最好同时改变 TC0C/TC1C 和 TC0R/TC1R 的值，避免 PWM 信号受到干扰;**
➢ **注 4：当开放 PWM0/ PWM1 输出功能的时候，TC0OUT/TC1OUT 必须置“0”;**
➢ **注 5：PWM 功能可同时在中断请求下运行。**

9 中断

9.1 概述

SN8P1702A / SN8P1703A 提供 3 个中断源：2 个内部中断（TC0, TC1）和 1 个外部中断（INT0）。外部中断可以将系统从睡眠模式唤醒进入高速模式，其输入引脚 INT0 与 P0.0 共享。当系统进入到中断服务程序时，总中断控制位 GIE 清零；系统退出中断服务程序后，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寄存器 INTRQ 中，用户可编程设置中断优先级。

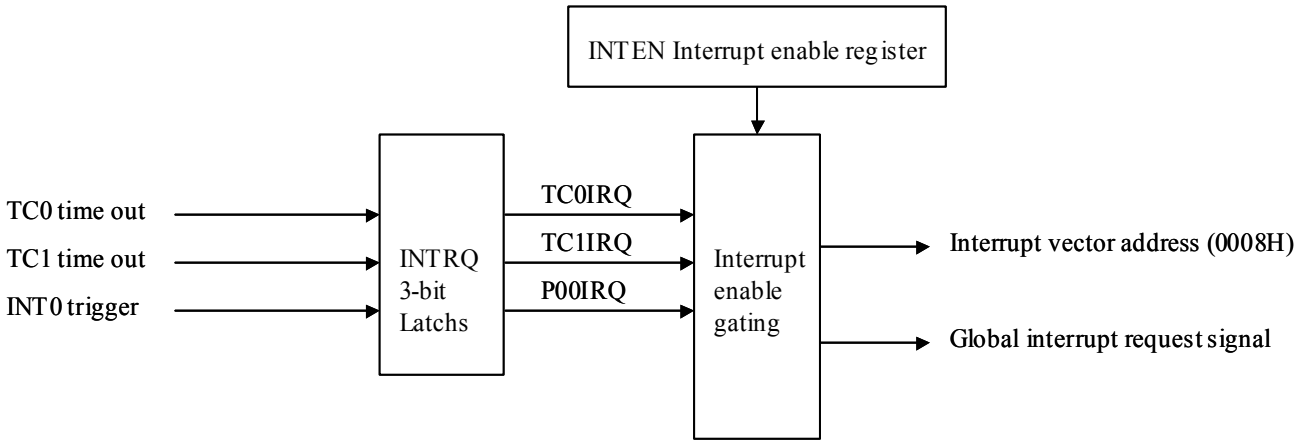


图 9-1　SN8P1702A / SN8P1703A 的中断

➢ **注：GIE 必须开放，才能响应各中断。**

9.2 INTEN 中断使能寄存器

INTEN 是中断请求控制寄存器，包括 2 个内部中断和 1 个外部中断的控制位。若 INTEN 的某位被置“1”，则该位对应的中断请求能够被响应。一旦有中断发生，程序跳至 ORG8 执行中断服务程序。当执行中断服务返回指令（RETI）时，退出中断服务程序。

INTEN 初始值=x00x xxx0

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	0	TC1IEN	TC0IEN	0	0	0	0	P00IEN
	-	R/W	R/W	-	-	-	-	R/W

Bit0　**P00IEN:** 外部中断 P0.0 控制位
0 = 禁止
1 = 使能

Bit5　**TC0IEN:** 内部中断 TC0 控制位
0 = 禁止
1 = 使能

Bit6　**TC1IEN:** 内部中断 TC1 控制位
0 = 禁止
1 = 使能

9.3 INTRQ 中断请求寄存器

INTRQ 是中断请求标志寄存器，包括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。

INTRQ 初始值=x00x xxx0

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	0	TC1IRQ	TC0IRQ	0	0	0	0	P00IRQ
	-	R/W	R/W	-	-	-	-	R/W

Bit0 **P00IRQ:** 外部中断 P0.0 请求位
0 = 无中断请求
1 = 请求中断

Bit5 **TC0IRQ:** 内部中断 TC0 请求位
0 = 无中断请求
1 = 请求中断

Bit6 **TC1IRQ:** 内部中断 TC1 请求位
0 = 无中断请求
1 = 请求中断

当中断发生时，不管 INTEN 是否使能，INTRQ 都置“1”。若 INTEN=1，且 INTRQ=1，则系统就进入中断向量执行中断；若 INTEN=0，不管 INTRQ 是否等于 1，系统都不执行中断。用户要注意多中断的操作。

9.4 中断操作举例

SN8P1702A / SN8P1703A 提供 3 个中断，详细操作如下：

9.4.1 GIE 总中断操作

GIE 是总中断控制位。所有的中断都要在 GIE=1 的前提下才能够被响应。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。

STKP 初始值=0xxx 1111

0DFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKP	GIE	-	-	-	STKPB3	STKPB2	STKPB1	STKPB0
	R/W	-	-	-	R/W	R/W	R/W	R/W

Bit7 **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 GIE 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

P0.0 的的中断触发方向决定于 PEDGE 寄存器。

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：低电平唤醒，下降沿中断。
P1：低电平唤醒
1 = 使能边沿触发功能
P0.0：由 P00G0 和 P00G1 控制其唤醒和中断触发边沿。
P1：改变电平（下降或上升沿）唤醒。

Bit[4:3] **P00[1:0]:** P0.0 边沿选择位
00 = 保留
01 = 上升沿
10 = 下降沿
11 = 上升/下降沿

➲ **例：初始化 INT0**

```
        B0BSET      FP00IEN         ; 使能 INT0 中断
        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       ; 使用 B0XCH 不影响标志位 C、Z
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A     ; 保存 ACC

        B0BTS1      FP00IRQ         ; 检查是否是 P00IRQ 中断请求
        JMP         EXIT_INT        ; P00IRQ = 0，退出中断服务程序

        B0BCLR      FP00IRQ         ; 清 P00IRQ
        .           .               ; INT0 中断服务程序
        .           .
EXIT_INT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A
        B0XCH       A, ACCBUF       ; 恢复 ACC

        RETI                        ; 中断返回
```

当 INT0 中断发生时，无论 P00IEN 是否使能，P00IRQ 都被置为“1”。若 P00IEN=1，且 P00IRQ=1，那么系统就进入中断向量地址（ORG8）执行中断服务程序；但若 P00IEN=0，不管 P00IRQ 是否为 1，系统都不进入中断。用户应注意多中断下的操作。

9.4.3 TC0 中断操作

当计数器 TC0C 溢出时，无论 TC0IEN 是否使能，TC0IRQ 都会置为“1”。若 TC0IEN=1，且 TC0IRQ=1，那么系统就进入中断向量地址（ORG8）执行中断服务程序；若 TC0IEN=0，不管 TC0IRQ 是否等于 1，系统都不进入中断向量。用户应注意多中断下的操作。

➲ **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 分频数为 Fcpu / 64
        MOV         A, #74H         ; 设置 TC0C 的初始值 = 74H
        B0MOV       TC0C, A         ; 定时时间间隔 = 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            ; 使能总中断
```

➲ **例：TC0 中断服务**

```
        ORG         8               ; 中断向量地址
        JMP         INT_SERVICE
INT_SERVICE:
        B0XCH       A, ACCBUF       ; 使用 B0XCH 不影响标志位 C、Z
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A     ; 保存 ACC


        B0BTS1      FTC0IRQ         ; 检查是否是 TC0 中断请求
        JMP         EXIT_INT        ; TC0IRQ = 0，退出中断

        B0BCLR      FTC0IRQ         ; 清 TC0IRQ
        MOV         A, #74H
        B0MOV       TC0C, A         ; 重新装载时间常数
        .                           ; TC0 中断服务程序
        .               .
EXIT_INT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A
        B0XCH       A, ACCBUF       ; 恢复 ACC

        RETI                        ; 中断返回
```

9.4.4 TC1 中断操作

当计数器 TC1C 溢出时，无论 TC1IEN 是否使能，TC1IRQ 都会置为“1”。若 TC1IEN=1，且 TC1IRQ=1，那么系统就进入中断向量地址（ORG8）执行中断服务程序；若 TC1IEN=0，不管 TC1IRQ 是否等于 1，系统都不进入中断向量。用户应注意多中断下的操作。

➲ **例：初始化 TC1**

```
        B0BCLR      FTC1IEN         ; 禁止 TC1 中断
        B0BCLR      FTC1ENB         ; 停止 TC1 计数
        MOV         A, #20H         ;
        B0MOV       TC1M, A         ; 设置 TC1 分频数为 Fcpu / 64
        MOV         A, #74H         ; 设置 TC1C 的初始值 = 74H
        B0MOV       TC1C, A         ; 定时时间 = 10 ms

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

        B0BSET      FGIE            ; 使能总中断
```

➲ **例：TC1 中断服务**

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


        B0BTS1      FTC1IRQ         ; 检查是否是 TC1 中断请求
        JMP         EXIT_INT        ; TC1IRQ = 0，退出中断

        B0BCLR      FTC1IRQ         ; 清 TC1IRQ
        MOV         A, #74H
        B0MOV       TC1C, A         ; 重新装载时间常数
        .               .           ; TC1 中断服务程序
        .               .
EXIT_INT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A
        B0XCH       A, ACCBUF       ; 恢复 ACC

        RETI                        ; 中断返回
```

9.4.5 多中断操作

大多数情况下，用户需要同时处理多个中断，这时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　事　件
P00IRQ	P0.0 触发，下降/上升沿
TC0IRQ	TC0C 溢出
TC1IRQ	TC1C 溢出

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

➲ **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
        B0MOV       A, PFLAG
        B0MOV       PFLAGBUF, A
                                        ;
INTP00CHK:                              ; 检查是否有 INT0 中断
        B0BTS1      FP00IEN             ; 检查是否允许外部中断 P00
        JMP         INTTC0CHK           ; 跳到下一个中断
        B0BTS0      FP00IRQ             ; 检查是否有 INT0 中断请求
        JMP         INTP00              ; 跳到 INT0 中断服务程序
INTTC0CHK:                              ; 检查是否有 TC0 中断
        B0BTS1      FTC0IEN             ; 检查是否允许 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              ; 跳到 TC0 中断服务程序
INTTC1HK:                               ; 检查是否有 TC1 中断
        B0BTS1      FTC1IEN             ; 检查是否允许 TC1 中断
        JMP         INT_EXIT            ; 跳到下一个中断
        B0BTS0      FTC1IRQ             ; 检查是否有 TC1 中断请求
        JMP         INTTC1              ; 跳到 TC1 中断服务程序
INT_EXIT:
        B0MOV       A, PFLAGBUF
        B0MOV       PFLAG, A
        B0XCH       A, ACCBUF           ; 恢复 ACC

        RETI                            ; 中断返回
```

10 输入/输出

10.1 概述

SN8P1702A / SN8P1703A 为用户提供 4 个 I/O：一个单向输入端口 P0，3 个输入输出端口(P1, P4, P5)。输入/输出端的方向由 PnM 寄存器控制（N=0, 1, 4, 5），用 PnUR 寄存器控制上拉寄存器。系统复位后，所有端口都默认为无上拉电阻的输入模式。

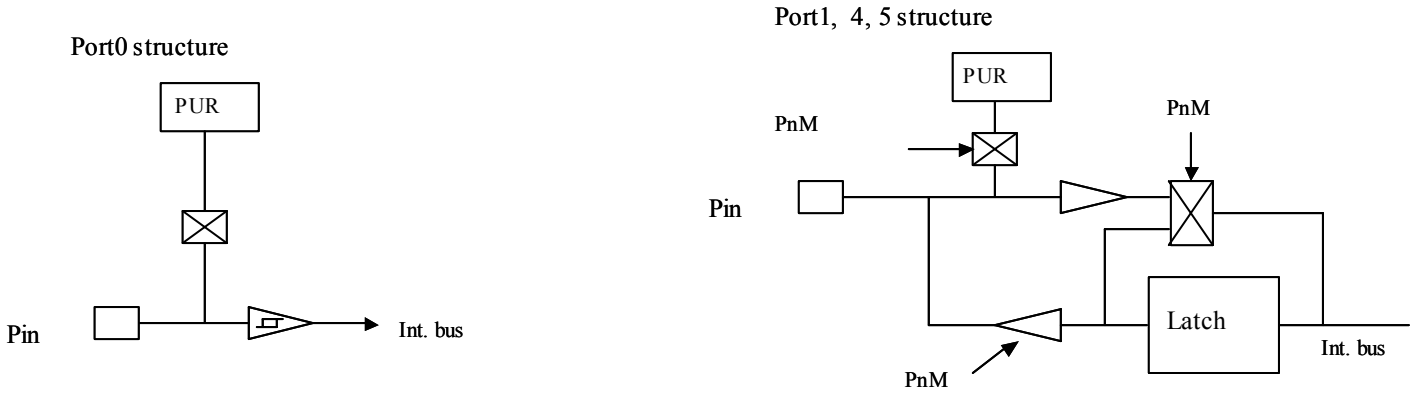


图 10－1　输入输出结构图

➢ 注：所有的输出锁存电路均为图腾柱结构

10.2 输入/输出功能表

Port/Pin	I/O	功能说明	注
P0.0	I	基本输入功能	
		外部中断(INT0)	
		系统睡眠模式唤醒	
P1.0~P1.1	I/O	基本输入/输出功能	
		系统睡眠模式唤醒	
P4.0~P4.3	I/O	基本输入/输出功能	
		ADC 模拟信号输入	
P5.0~P5.5	I/O	基本输入/输出功能	

表 10-1　I/O 功能表

10.3 上拉电阻

SN8P1702A / SN8P1703A 系列的 P0，P1，P4，P5 都有内置上拉电阻。用户可以通过引脚设置上拉电阻。

寄存器	P0UR							
地址	E0H							
Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
位名称	-	-	-	-	-	-	-	P00R
读/写	-	-	-	-	-	-	-	R/W
复位后	0	0	0	0	0	0	0	0

寄存器	P1UR							
地址	E1H							
Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
位名称	-	-	-	-	-	-	P11R	P10R
读/写	-	-	-	-	-	-	R/W	R/W
复位后	0	0	0	0	0	0	0	0

寄存器	P4UR							
地址	E4H							
Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
位名称	-	-	-	-	P43R	P42R	P41R	P40R
读/写	-	-	-	-	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

寄存器	P5UR							
地址	0E0H							
Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
位名称	-	-	P55R	P54R	P53R	P52R	P51R	P50R
读/写	-	-	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

CHIP SN8P1703A

```
        ORG 0x10
Main:
        MOV     A, #01H
        B0MOV   P0UR,A          ; 使能 P0.0 的上拉电阻
```

➲ **例 2：开放上拉电阻。**

CHIP SN8P1703A

```
        ORG 0x10
Main:
        MOV     A, #01H
        B0MOV   P0UR,A          ; 使能 P0 的上拉电阻
        MOV     A, #03H
        B0MOV   P1UR,A          ; 使能 P1 的上拉电阻
        MOV     A, #0FH
        B0MOV   P4UR,A          ; 使能 P4 的上拉电阻
        MOV     A, #01FH
        B0MOV   P5UR,A          ; 使能 P5 的上拉电阻
```

➢ **注：使能 P0 和 P1 的内置上拉电阻，避免未知因素将系统从睡眠模式中唤醒。**

10.4 I/O 端口模式

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。

P1M 初始值=xxxx xx00

0C1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1M	0	0	0	0	0	0	P11M	P10M
	-	-	-	-	-	-	R/W	R/W

Bit[1:0] **P1[1:0]M:** P1.0~P1.1 输入/输出方向控制位。
0 = 输入模式
1 = 输出模式

P4M 初始值=xxxx 0000

0C4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P4M	0	0	0	0	P43M	P42M	P41M	P40M
	-	-	-	-	R/W	R/W	R/W	R/W

Bit[3:0] **P1[3:0]M:** P4.0~P4.3 输入/输出方向控制位。
0 = 输入模式
1 = 输出模式

P5M 初始值=xx00 0000

0C5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5M	0	0	P55M	P54M	P53M	P52M	P51M	P50M
	-	-	R/W	R/W	R/W	R/W	R/W	R/W

Bit[5:0] **P1[5:0]M:** P5.0~P5.5 输入/输出方向控制位。
0 = 输入模式
1 = 输出模式

PnM 的每位都设为 0，则 I/O 口就为输入模式，都设为 1，则为输出模式。输入模式下，通过宏指令@SET_PUR 控制上拉电阻，输出模式下，上拉电阻被禁止。

- **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: 端口模式设置**

```
CLR     P1M         ; 设置为输入模式
CLR     P4M
CLR     P5M

MOV     A, #0FFH    ; 设置为输出模式
B0MOV   P1M, A
B0MOV   P4M, A
B0MOV   P5M, A

B0BCLR  P1M.0       ; 设置 P1.0 为输入模式

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

10.5 I/O 数据寄存器

P0 初始值=xxxx xxx0

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0	-	-	-	-	-	-	-	P00
	-	-	-	-	-	-	-	R

P1 初始值=xxxx xx00

0D1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1	-	-	-	-	-	-	P11	P10
	-	-	-	-	-	-	R/W	R/W

P4 初始值=xxxx 0000

0D4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P4	-	-	-	-	P43	P42	P41	P40
	-	-	-	-	R/W	R/W	R/W	R/W

P5 初始值=xx00 0000

0D5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5	-	-	P55	P54	P53	P52	P51	P50
	-	-	R/W	R/W	R/W	R/W	R/W	R/W

➲ **例：从输入端口读取数据**

```
B0MOV   A, P0     ; 读 P0 口的数据
B0MOV   A, P1     ; 读 P1 口的数据
B0MOV   A, P4     ; 读 P4 口的数据
B0MOV   A, P5     ; 读 P5 口的数据
```

➲ **例：写入数据到输出端口**

```
MOV     A, #55H   ; 所有的端口写 55H
B0MOV   P1, A
B0MOV   P4, A
B0MOV   P5, A
```

➲ **例：将 1-bit 数据送入输出端**

```
B0BSET  P1.1      ; 设置 P1.1 和 P4.0 为“1”
B0BSET  P4.0

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

➲ **例：端口位检测试**

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

11 4 通道 A/D 转换

11.1 概述

SN8P1702A / SN8P1703A 的 A/D 转换器共有 4 个通道，4096 阶的分辨率，可以将模拟信号转换成 12 位的数字信号，进行 AD 转换时，首先要选择输入通道（AIN0 ~ AIN3），然后将 GCHS 和 ADS 置“1”，启动 AD 转换。转换结束后，系统自动将 EOC 位置为“1”，并将转换结果存入寄存器 ADB 中。通过 ADR 寄存器的 ADLEN 位可以选择分辨率（8 位或 12 位）。

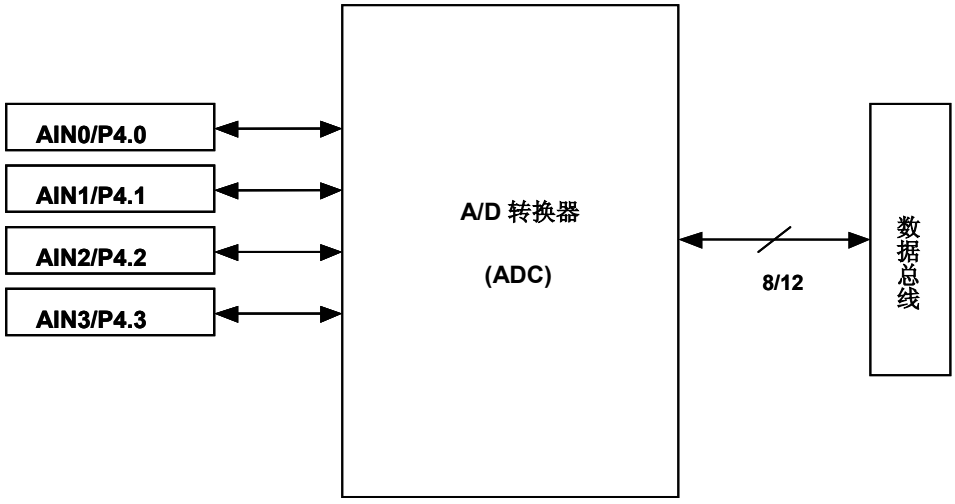


图 11－1　AD 转换功能结构图

- 注：分辨率为 8 位时，转换时间为 12 个 ADC 输入时钟；
  分辨率为 12 位时，转换时间为 16 个 ADC 输入时钟。
- 注：模拟输入电压必须在 AVREFH 和 AVSS 之间。
- 注：AVREFH 和 SN8P1702A 内部的 VDD 连接。
- 注：AVREFH 的值必须在 AVDD 和 AVSS＋2.0V 之间。
- 注：ADC 设计时的注意事项：
  - 设 ADC 的 I/O 口为输入模式。
  - 禁止 ADC 输入引脚的上拉电阻。
  - 省电模式下设置 P4CON 的有关位以避免额外功耗。
  - 使能 ADC (ADENB = 1)后延迟 100us 等待 ADC 电路准备好转换。
  - 在进入睡眠模式前禁止 ADC(设置 ADENB = 0)以省电。

11.2 ADM 寄存器

ADM 初始值=0000 xx00

0B1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADM	ADENB	ADS	EOC	GCHS	-	-	CHS1	CHS0
	R/W	R/W	R/W	R/W	-	-	R/W	R/W

Bit[1:0] **CHS[1:0]:** ADC 输入通道选择位。
00 = AIN0
01 = AIN1
10 = AIN2
11 =AIN3

Bit4 **GCHS:** 全局通道选择位
0 = 禁止 AIN 通道
1 = 使能 AIN 通道

Bit5 **EOC:** ADC 状态位
0 = 转换过程中
1 = 转换结束，ADS 复位

Bit6 **ADS:** ADC 启动位
0 = 停止转换
1 = 启动转换

Bit7 **ADENB:** ADC 控制位
0 = 禁止
1 = 使能

11.3 ADR 寄存器

ADR 初始值=x00x 0000

0B3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADR	-	ADCKS1	ADLEN	ADCKS0	ADB3	ADB2	ADB1	ADB0
	-	R/W	R/W	R/W	R	R	R	R

Bit[3:0] **ADBn:** ADC 数据缓存器
8 位 ADC：ADB11~ADB4;
12 位 ADC：ADB11~ADB0.

Bit5 **ADLEN:** ADC 分辨率选择位
0 = 8 位
1 = 12 位

Bit6， Bit4 **ADCKS1, ADCKS0:** ADC 的时钟源选择位

ADCKS1	ADCKS0	ADC 时钟源	注
0	0	Fcpu/4	在普通模式和低速模式下都有效
0	1	Fcpu/2	在普通模式和低速模式下都有效
1	0	Fhosc	只在普通模式下有效
1	1	Fhosc/2	只在普通模式下有效

11.4 ADB 寄存器

ADB 初始值=0000 0000

0B2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADB	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4
	R	R	R	R	R	R	R	R

ADB 为 8 位数据缓存器，用来保存 ADC 转换结果。ADB 只包含了 AD 转换结果中的高 8 位，把 ADB 寄存器和 ADR 的低半字节结合在一起，可得到一个 12 位的转换结果。ADB 为只读寄存器，在 8 位 ADC 模式下，AD 转换结果保存在寄存器 ADB 中；在 12 位模式下，则分别保存在寄存器 ADB 和 ADR 中。

- **注：在上电时，ADB[0:11]是未知的。**

11.5 P4CON 寄存器

ADB 初始值=xxxx 0000

0AEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P4CON	0	0	0	0	P4CON3	P4CON2	P4CON1	P4CON0
	-	-	-	-	R/W	R/W	R/W	R/W

P4CON 是 P4 的配置寄存器，该寄存器可在未选择的、连接到模拟输入源上的 ADC 通道消除漏电流。P4CON[3:0] 置于高电平时，可以将 P4 的数字输入通道隔离在外。例如：AIN0(P4.0)和 AIN1(P4.1)都设为 ADC 通道，AIN0 作为转换通道（CHS[1:0]=00），表示 P4.1 可以作为数字输入模式（若 P41M=0），这种情况下，就可能有电流从模拟输入源漏出。设 P4CON=1，就可以消除 AIN1 的漏电流。同理，当 AIN1 作为转换通道时，P4CON0 也必须设为“1”。因此 P4 的任何引脚作为 ADC 通道时，都要将 P4CON 相关的位置于高电平以消除漏电流，尤其在进入睡眠模式前。

Bit[3:0] **P4C0N:** Port4 配置寄存器

P4CON3	0	让 P4.3 的数字信号通过
	1	将 P4.3 的数字信号隔离在外
P4CON2	0	让 P4.2 的数字信号通过
	1	将 P4.2 的数字信号隔离在外
P4CON1	0	让 P4.1 的数字信号通过
	1	将 P4.1 的数字信号隔离在外
P4CON0	0	让 P4.0 的数字信号通过
	1	将 P4.0 的数字信号隔离在外

- **注 1：当 P4 为基本输入/输出端口时，设 P4CON[3:0]=0**
- **注 2：当 P4 为 ADC 输入通道时，设 P4CON[3:0]=1**

AIN 的输入电压与 ADB 的输出数据的关系

AIN n	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4	ADB3	ADB2	ADB1	ADB0
0/4096*AVREFH	0	0	0	0	0	0	0	0	0	0	0	0
1/4096*AVREFH	0	0	0	0	0	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
4094/4096*AVREFH	1	1	1	1	1	1	1	1	1	1	1	0
4095/4096*AVREFH	1	1	1	1	1	1	1	1	1	1	1	1

针对不同的应用，用户可能需要一个 8 位到 12 位之间的分辨率。对于这种情况，可以通过对保存在 ADR 和 ADB 中的转换结果进行处理得到。首先，用户必须选择 12 位分辨率的模式，进行 AD 转换，然后在转换结果中去掉最低的几位得到需要的结果。如下表所示：

ADC 分辨率	ADB								ADR			
	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4	ADB3	ADB2	ADB1	ADB0
8-bit	O	O	O	O	O	O	O	O	x	x	x	x
9-bit	O	O	O	O	O	O	O	O	O	x	x	x
10-bit	O	O	O	O	O	O	O	O	O	O	x	x
11-bit	O	O	O	O	O	O	O	O	O	O	O	x
12-bit	O	O	O	O	O	O	O	O	O	O	O	O
O = 可选位，x = 未使用的位												

11.6 ADC 转换时间

12 位 ADC 转换时间=1/(ADC clock/4)*16 sec

8 位 ADC 转换时间=1/(ADC clock/4)*12 sec

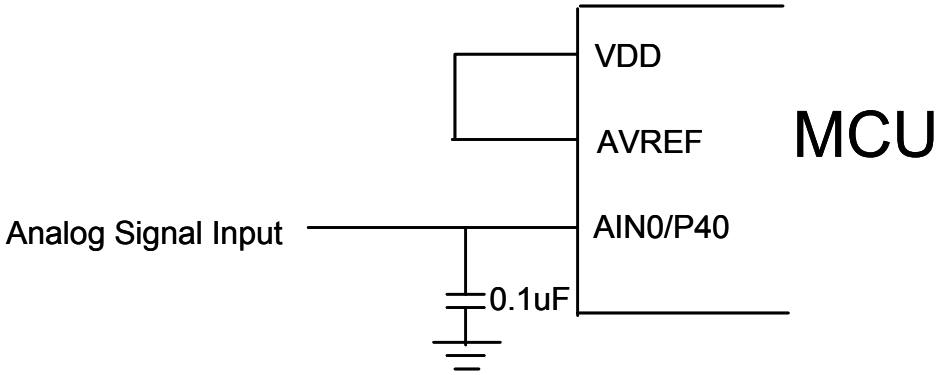
高速时钟(fosc)为 3.58MHz

ADLEN	ADCKS1	ADCKS0	ADC	ADC 转换时间
0 (8 位)	0	0	Fcpu/4	1/((3.58MHz/4)/4/4)*12 = 214.5 us
	0	1	Fcpu/2	1/((3.58MHz/4)/2/4)*12 = 107.3 us
	1	0	Fhosc	1/(3.58MHz/4)*12 = 13.4 us
	1	1	Fhosc/2	1/(3.58MHz/2/4)*12 = 26.8 us
1 (12 位)	0	0	Fcpu/4	1/((3.58MHz/4)/4/4)*16 = 286 us
	0	1	Fcpu/2	1/((3.58MHz/4)/2/4)*16 = 143 us
	1	0	Fhosc	1/(3.58MHz/4)*16 = 17.9 us
	1	1	Fhosc/2	1/(3.58MHz/2/4)*16 = 35.8 us

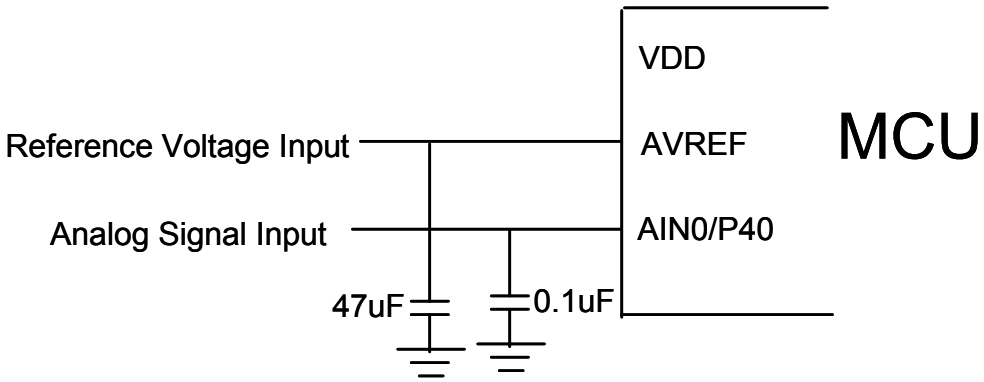
➲ **例：选取 AIN0 ~ AIN1 作为 ADC 的输入通道，分辨率为 12 位**

```
ADC0:
        MOV     A, #60H
        B0MOV   ADR, A          ; 设在 ADC 为 12 位，时钟源为 Fosc.
        B0SET   FP4CON1         ; 隔离通道 AIN1 避免漏电流
        B0CLR   FP4CON0         ; 选择通道 AIN0 作为 ADC 通道
        MOV     A,#90H
        B0MOV   ADM,A           ; 使能 ADC 并选择通道 AIN0
        B0SET   P4CON1
        B0BSET  FADS            ; 开始转换
WADC0:
        B0BTS1  FEOC            ; 判断转换是否结束
        JMP     WADC0           ; 未结束则跳到 WADC0
        B0MOV   A,ADB           ; 得到转换数据
ADC1:
        B0SET   FP4CON0         ; 隔离 AIN0 通道避免漏电流
        B0CLR   FP4CON1         ; 选择通道 AIN1 作为 ADC 通道
        MOV     A,#91H          ;
        B0MOV   ADM,A           ; 使能 ADC 并选择通道 AIN1
        B0BSET  FADS            ; 开始转换
        .       .               .
QEXADC:
        B0BCLR  FGCHS           ; 释放 ADC 通道
```

11.7 ADC 电路



AVREFH 连接到 VDD



AVREFH 连接到外部 AD 参考电压上

图 11－2　AD 转换的 AINx 和 AVREFH 电路

- **注：AIN 和 GND 之间的旁路电容有助于模拟信号的稳定。**

12 编程

12.1 编程模板

```
;*************************************************************************************
; 文件名 : TEMPLATE.ASM
; 作　者 : SONiX
; 用　途 : Template Code for SN8X17XX
; 版　本 : 09/01/2002 V1.0  First issue
;*************************************************************************************
;*    (c) Copyright 2002, SONiX TECHNOLOGY CO., LTD.
;*************************************************************************************
CHIP      SN8P1703A                          ; 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           ;数据放在 Bank0 中从地址 0x00 开始的地址
    Wk00B0          DS          1            ;主循环用到的临时变量
    Iwk00B0         DS          1            ;中断中用到的临时变量
    AccBuf          DS          1            ;用来保存 ACC 数据的寄存器
    PflagBuf        DS          1            ;用来保存 PFLAG 数据的寄存器

                org     100h                 ;Bank1 数据区
    BufB1       DS      20                   ;Bank1 中的临时变量

;-------------------------------------------------------------------------------
;                                标志位定义
;-------------------------------------------------------------------------------
    Wk00B0_0        EQU         Wk00B0.0     ;Wk00B0 的第 0 位
     Iwk00B0_1      EQU          Iwk00B0.1   ;Iwk00 的第 1 位

;-------------------------------------------------------------------------------
;                                代码区
;-------------------------------------------------------------------------------

.CODE

    ORG             0                        ;代码开始位置
    jmp             Reset                    ;复位向量地址
                                             ;地址 4～7 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
    b0mov           RBANK,#00h          ;设置 RAM 位于 bank0
    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
    b0bset          FGIE                ;使能总中断

;-------------------------------------------------------------------------
;   主程序循环区
;-------------------------------------------------------------------------
Main:
    b0bclr          FWDRST              ;清看门狗定时器

    call            MnApp

    jmp             Main

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

; 在这里仿真主程序

    ret

;-----------------------------------
跳转表程序
;-----------------------------------
    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:
    ; 子程序 1
    jmp         JmpExit

JmpSub1:
    ; 子程序 2
    jmp         JmpExit

JmpSub2:
    ; 子程序 3
    jmp         JmpExit

JmpExit:
```

```
    ret                                         ;返回主程序

;-------------------------------------------------------------------------------
; Isr (中断服务程序)
; Arguments   :
; Returns :
; Reg Change:
;-------------------------------------------------------------------------------
Isr:
;-----------------------------------
;     保存 ACC 和工作寄存器的值
;-----------------------------------
    b0xch       A,AccBuf            ;使用 B0XCH 不会影响到 C、Z 标志
    ;b0mov      A,PFLAG
    ;b0mov      PflagBuf,A

;-----------------------------------
;    检查是否有中断发生
;-----------------------------------

IntP00Chk:
    b0bts1      FP00IEN
    jmp         IntTc0Chk
    b0bts0      FP00IRQ
    jmp         P00isr

    ;如果需要，可以在这里插入其它的中断

    IntTc0Chk:
    b0bts1      FTC0IEN
    jmp         IsrExit
    b0bts0      FTC0IRQ
    jmp         TC0isr

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

IsrExit:

; 对于 SN8X1702 应采用以下方法
        ;b0mov          A,PFLAG
        ;b0mov          PflagBuf,A
        b0xch           A,AccBuf        ;使用 B0XCH 不会影响到 C、Z 标志位
        reti                            ;退出中断

;-------------------------------------------------------------------------------
;     INT0 中断服务程序
;-------------------------------------------------------------------------------
P00isr:
    b0bclr      FP00IRQ

;在这里处理外部中断 P0.0

    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~7Fh)
;-------------------------------------------------------------------------
ClrRAM:

; RAM Bank 0
    clr         Y                   ;选择 bank0
    b0mov       Z,#0x7f             ;设置@YZ 地址为 7fh

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

; RAM Bank 1
    mov         A,#1
    b0mov       Y,A                 ;选择 bank1
    b0mov       Z,#0x7f             ;设置@YZ 地址为 17fh

    ClrRAM20:
    clr         @YZ                 ;清@YZ
    decms       Z                   ;Z = Z – 1，若 Z = 0 则跳过下一条指令
    jmp         ClrRAM20
    clr         @YZ                 ;清 0x100
    ret
    ;-------------------------------------------------------------------------
    ENDP
```

12.2 程序检查对照表

项　　目	说　　明
上拉电阻	宏指令@SET_PUR 控制上拉电阻，详见 I/O 章节
未定义位	系统寄存器中所有标记为“0”（未定义）的位都必须置“0”，以避免系统出错。
ADC	ADC 的模拟输入端应设置为无上拉电阻的输入模式
PWM0	PWM0 (P5.4) 为输出模式
PWM1	PWM1 (P5.3) 为输出模式
中断	RAM 初始化之前禁止中断.
未使用的 I/O 口	未使用的 I/O 口应设为上拉/下拉的输入模式或低电平输出模式以减小电流损耗
睡眠模式	使能 P0 和 P1 的内置上拉电阻，避免未知的系统唤醒
堆栈缓冲器	避免堆栈溢出
系统初始化	1. 堆栈初始化时：STKP=0x7F，GIE=0 2. RAM 清零. 3. 初始化所有的系统寄存器
抗干扰性	1. 使能编译选项中的 OSG 和 High_Clk / 2 2. 使能看门狗定时器 3. 未使用的 I/O 口应设为低电平输出模式 4. 不断刷新 RAM 中重要的系统寄存器和变量以避免干扰

13 指令表

类	操作码		说　　明	C	DC	Z	Cycle
	MOV	A,M	A ← M	-	-	√	1
M	MOV	M,A	M ← A	-	-	-	1
O	B0MOV	A,M	A ← M (bnak 0)	-	-	√	1
V	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
E	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
	ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
A	ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
R	ADD	A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
I	ADD	M,A	M ← M + A, if occur carry, then C=1, else C=0	√	√	√	1
T	B0ADD	M,A	M (bank 0) ← M (bank 0) + A, if occur carry, then C=1, else C=0	√	√	√	1
H	ADD	A,I	A ← A + I, if occur carry, then C=1, else C=0	√	√	√	1
M	SBC	A,M	A ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
E	SBC	M,A	M ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
T	SUB	A,M	A ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
I	SUB	M,A	M ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
C	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
	AND	A,M	A ← A and M	-	-	√	1
L	AND	M,A	M ← A and M	-	-	√	1
O	AND	A,I	A ← A and I	-	-	√	1
G	OR	A,M	A ← A or M	-	-	√	1
I	OR	M,A	M ← A or M	-	-	√	1
C	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
	SWAP	M	A (b3~b0, b7~b4) ←M(b7~b4, b3~b0)	-	-	-	1
P	SWAPM	M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-	1
R	RRC	M	A ← RRC M	√	-	-	1
O	RRCM	M	M ← RRC M	√	-	-	1
C	RLC	M	A ← RLC M	√	-	-	1
E	RLCM	M	M ← RLC M	√	-	-	1
S	CLR	M	M ← 0	-	-	-	1
S	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
	CMPRS	A,I	ZF,C ← A - I,　If A = I, then skip next instruction	√	-	√	1 + S
B	CMPRS	A,M	ZF,C ← A – M,　If A = M, then skip next instruction	√	-	√	1 + S
R	INCS	M	A ← M + 1, If A = 0, then skip next instruction	-	-	-	1 + S
A	INCMS	M	M ← M + 1, If M = 0, then skip next instruction	-	-	-	1 + S
N	DECS	M	A ← M - 1, If A = 0, then skip next instruction	-	-	-	1 + S
C	DECMS	M	M ← M - 1, If M = 0, then skip next instruction	-	-	-	1 + S
H	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
M	RET		PC ← Stack	-	-	-	2
I	RETI		PC ←Stack, and to enable global interrupt	-	-	-	2
s	NOP		No operation	-	-	-	1
c	@SET_PUR	VAL	Enable or disable pull-up resisters. Bit N of VAL: "0" disable port N pull-up, "1" enable port N pull-up	-	-	√	-

14 电气特性

14.1 极限参数

（所有电压以 Vss 为参考基准）

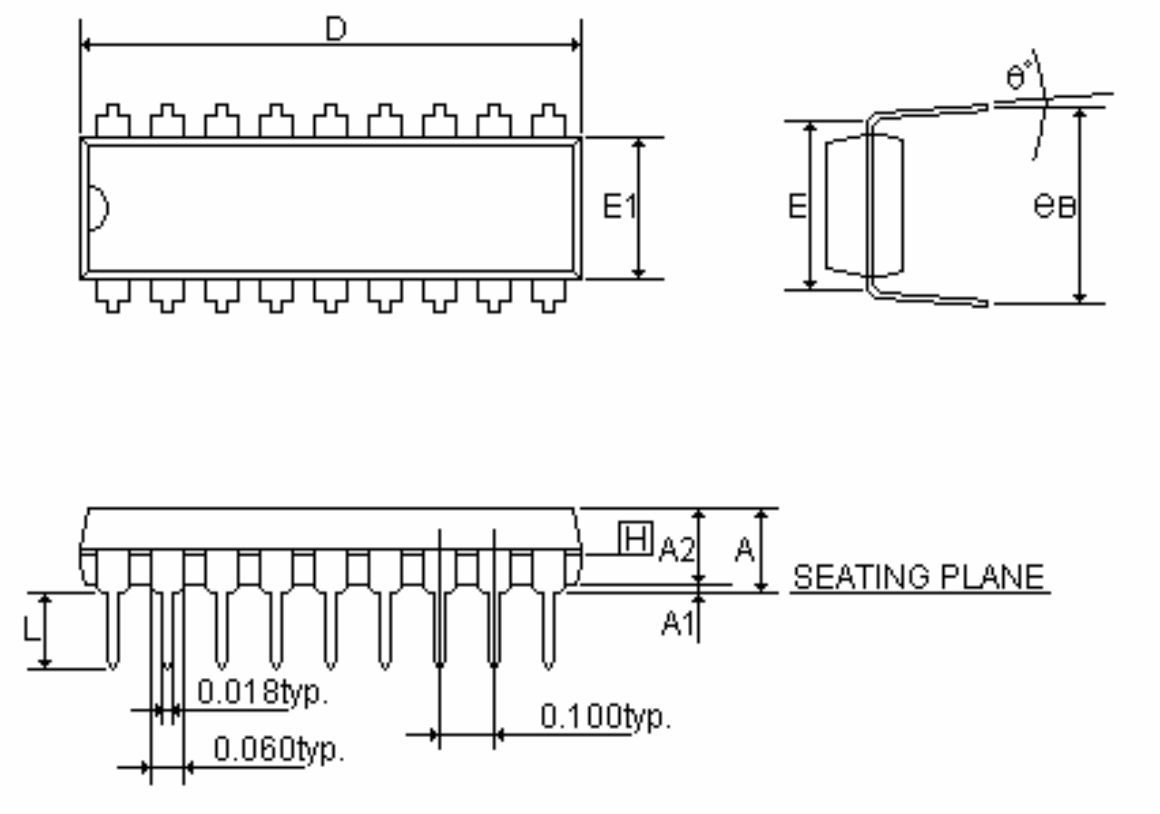
电源电压(Vdd)..... - 0.3V ~ 6.0V
输入电压(Vin).....Vss - 0.2V ~ Vdd + 0.2V
工作环境温度(Topr).....0°C ~ + 70°C
存储温度(Tstor).....-30°C ~ + 125°C
功耗(Pc).....500 mW

14.2 标准电气特性

参数	符号	说明		最小值	标准值	最大值	单位
工作电压	Vdd	普通模式, Vpp = Vdd		2.4	5.0	5.5	V
		编程模式, Vpp = 12.5V		4.5	5.0	5.5	
RAM 数据保持电压	Vdr			-	1.5	-	V
内部 POR	Vpor	Vdd 的上升速率以确保内部的上电复位		-	0.05	-	V/ms
输入低电压	ViL1	除特别申明的所有引脚		Vss	-	0.3Vdd	V
	ViL2	施密特触发输入端—Port 0		Vss	-	0.2Vdd	V
	ViL3	复位引脚；Xin (RC 模式)		Vss	-	0.2Vdd	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 (X'tal 模式)		0.7Vdd	-	Vdd	V
复位引脚漏电流	Ilekg	Vin = Vdd		-	-	2	uA
上拉电阻	Rup	Vin = Vss , Vdd = 5V		-	100	-	KΩ
端口输入漏电流	Ilekg	禁止上拉电阻, Vin = Vdd		-	-	2	uA
P1 输出电流	IoH	Vop = Vdd - 0.5V		-	15	-	mA
灌电流	IoL	Vop = Vss + 0.5V		-	15	-	
P4 输出电流	IoH	Vop = Vdd - 0.5V		-	15	-	mA
灌电流	IoL	Vop = Vss + 0.5V		-	15	-	
P5 输出电流	IoH	Vop = Vdd - 0.5V		-	15	-	mA
灌电流	IoL	Vop = Vss + 0.5V		-	15	-	
INT0 触发脉冲宽度	Tint0	INT0 中断请求脉冲宽度		2/fcpu	-	-	cycle
AVREFH 输入电压（SN8P1702A 中 AVREFH=Vdd）	Varef	Vdd = 5.0V		2.0V	-	Vdd	V
AIN0 ~ AIN3 输入电压	Vani			Vss	-	Avref	V
ADC 使能时间	Tast	设置 ADENB = 1 后准备开始转换。		-	100	-	uS
振荡器频率	Fhosc	晶体模式		32768	4M	16M	Hz
		VDD=3V，外部 RC 模式		-	6M	-	
		VDD=5V，外部 RC 模式		-	10M	-	
电源电流 (禁止 ADC)	Idd1	运行模式（禁止低功耗）	Vdd= 5V 4Mhz	-	2.5	6	mA
			Vdd= 3V 4Mhz	-	1	2	mA
	Idd2	运行模式（使能低功耗）	Vdd= 5V 4Mhz	-	1.6	3	mA
			Vdd= 3V 4Mhz	-	0.7	1.5	mA
	Idd3	低速模式（停止高速时钟）	Vdd=5V, 32KHz Int.RC	-	30	60	uA
			Vdd=5V, 32KHz Int.RC	-	7	20	uA
	Idd4	睡眠模式	Vdd= 5V	-	1	2	uA
			Vdd= 3V	-	0.6		uA
	Idd5	绿色模式（停止高速时钟）	Vdd=5V, 32KHz Int.RC	-	16	40	uA
			Vdd=3V, 32KHz Int.RC		3	10	uA
ADC 电流消耗	IADC	Vdd= 5V		-	0.6	1	mA
		Vdd= 3V		-	0.4	0.8	mA
LVD 侦测电压	vdet	低电压侦测电平		-	1.8	-	V

15 封装

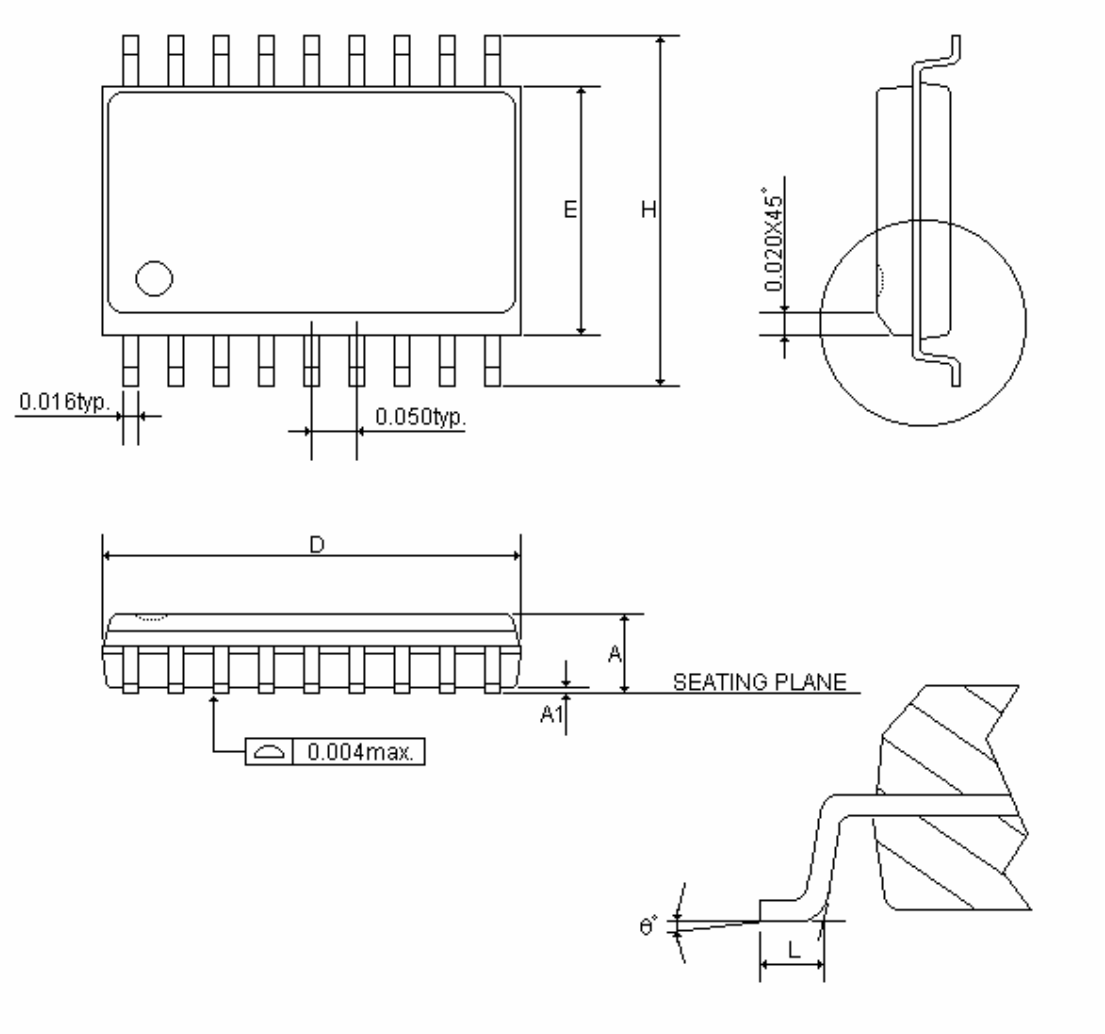
15.1 P-DIP18 PIN



Symbols	MIN.	NOR.	MAX.
A	-	-	0.210
A1	0.015	-	-
A2	0.125	0.130	0.135
D	0.880	0.900	0.920
E		0.300BSC.	
E1	0.245	0.250	0.255
L	0.115	0.130	0.150
e B	0.335	0.355	0.375
θ°	0	7	15

单位：INCH

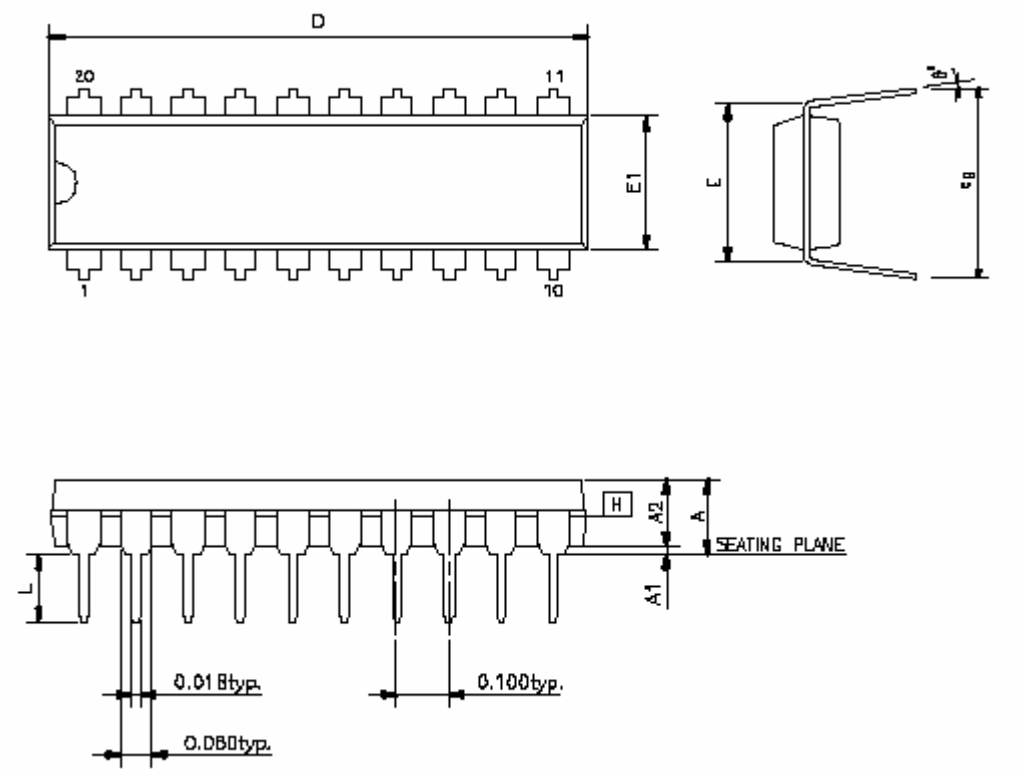
15.2 SOP18 PIN



Symbols	MIN.	MAX.
A	0.093	0.104
A1	0.004	0.012
D	0.447	0.463
E	0.291	0.299
H	0.394	0.419
L	0.016	0.050
θ°	0	8

单位：INCH

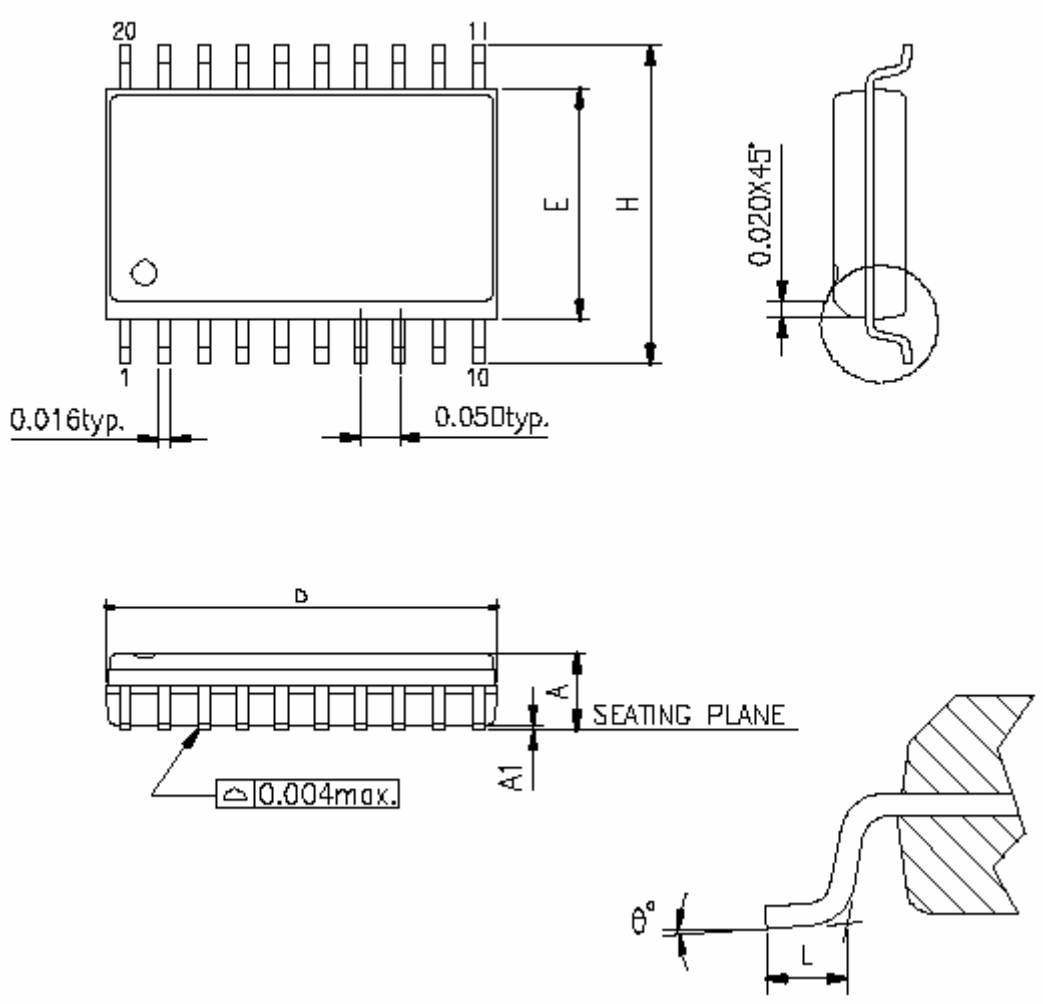
15.3 P-DIP 20 PIN



Symbols	MIN.	NOR.	MAX.
A	-	-	0.210
A1	0.015	-	-
A2	0.125	0.130	0.135
D	0.9B	1.D30	1.060
E	0.300BSC.		
E1	0.245	0.250	0.255
L	0.115	0.130	0.150
e B	0.335	0.355	0.375
θ°	0	7	15

单位： INCH

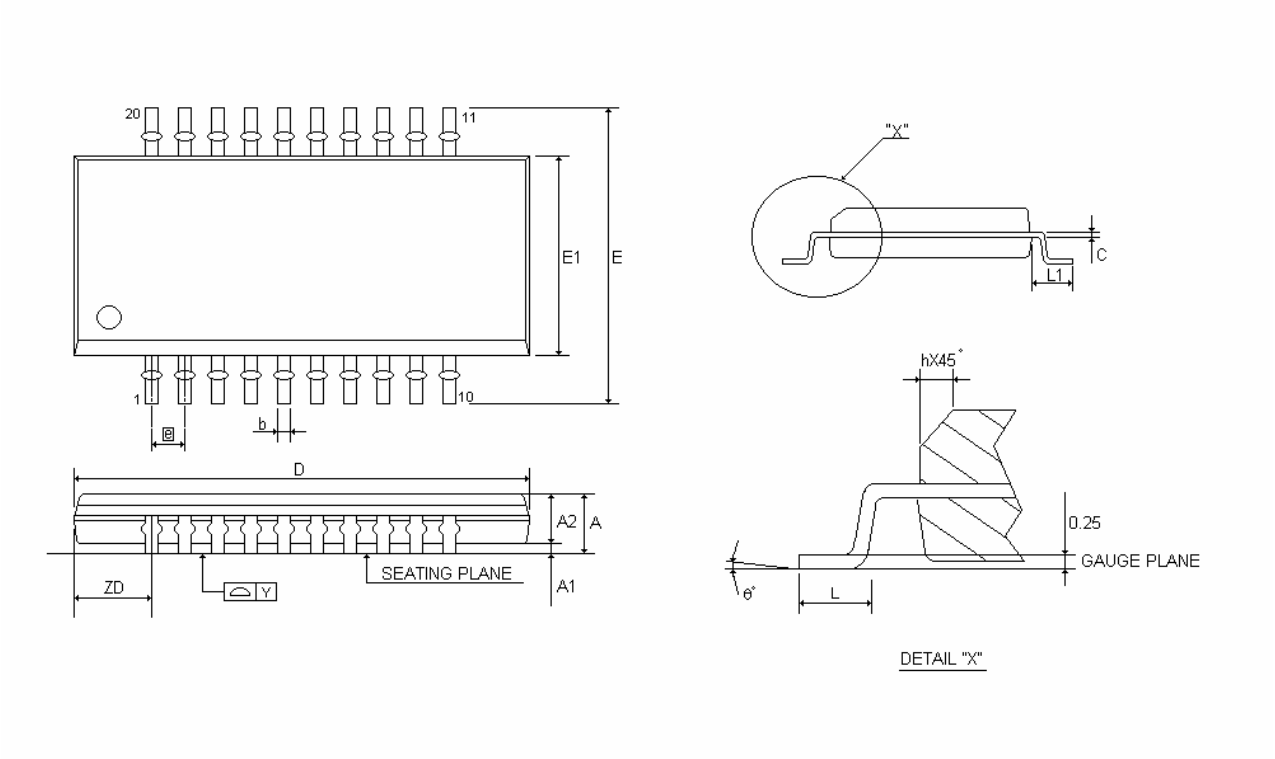
15.4 SOP 20 PIN



Symbols	MIN.	MAX.
A	0.093	0.104
A1	0.004	0.012
D	0.496	0.508
E	0.291	0.299
H	0.394	0.419
L	0.016	0.050
θ°	0	8

单位：INCH

15.5 SSOP 20 PIN



Symbols	DIMENSION (MM)			DIMENSION (MIL)		
	MIN,	NOM.	MAX.	MIN.	NOM.	MAX.
A	1.35	1.60	1.75	53	63	69
A1	0.10	0.15	0.25	4	6	10
A2	-	-	1.50	-	-	59
b	0.20	0.254	0.30	8	10	12
b1	0.20	0.254	0.28	8	11	11
C	0.18	0.203	0.25	7	8	10
C1	0.18	0.203	0.23	7	8	9
D	8.56	8.66	8.74	337	341	344
E	5.80	6.00	6.20	228	236	244
E1	3.80	3.90	4.00	150	154	157
e	0.635 BSC			25 BSC		
h	0.25	0.42	0.50	10	17	20
L	0.40	0.635	1.27	16	25	50
L1	1.00	1.05	1.10	39	41	43
ZD	1.50 REF			58 REF		
Y	-	-	0.10	-	-	4
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
传真：0755－26719786