

# SN8P2711A

## 用户参考手册

Version 1.0

## SONiX 8 位单片机

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

## 修正记录

版本号	日期	内容
VER 1.0	2008 年 3 月	初版。
	2008 年 5 月	1. 修改烧录信息内容。

## 目 录

修正记录.....	2
目 录.....	3
1 产品简介.....	6
1.1 功能特性.....	6
1.2 系统结构框图.....	7
1.3 引脚配置.....	8
1.4 引脚说明.....	8
1.5 引脚电路结构图.....	9
2 中央处理器（CPU）.....	10
2.1 存储器.....	10
2.1.1 程序存储器（ROM）.....	10
2.1.1.1 复位向量（0000H）.....	10
2.1.1.2 中断向量（0008H）.....	11
2.1.1.3 查表.....	12
2.1.1.4 跳转表.....	14
2.1.1.5 CHECKSUM计算.....	15
2.1.2 编译选项表（CODE OPTION）.....	16
2.1.3 数据存储器（RAM）.....	16
2.1.4 系统寄存器.....	17
2.1.4.1 系统寄存器列表.....	17
2.1.4.2 系统寄存器说明.....	17
2.1.4.3 系统寄存器的位定义.....	18
2.1.4.4 累加器.....	19
2.1.4.5 程序状态寄存器PFLAG.....	20
2.1.4.6 程序计数器.....	21
2.1.4.7 Y, Z寄存器.....	23
2.1.4.8 R寄存器.....	23
2.2 寻址模式.....	24
2.2.1 立即寻址.....	24
2.2.2 直接寻址.....	24
2.2.3 间接寻址.....	24
2.3 堆栈.....	25
2.3.1 概述.....	25
2.3.2 堆栈寄存器.....	25
2.3.3 堆栈操作举例.....	26
3 复位.....	27
3.1 概述.....	27
3.2 上电复位.....	28
3.3 看门狗复位.....	28
3.4 掉电复位.....	29
3.4.1 概述.....	29
3.4.2 系统工作电压.....	29
3.4.3 掉电复位性能改进.....	30
3.5 外部复位.....	32
3.6 外部复位电路.....	33
3.6.1 RC复位电路.....	33
3.6.2 二极管及RC复位电路.....	33
3.6.3 稳压二极管复位电路.....	34
3.6.4 电压偏置复位电路.....	34
3.6.5 外部IC复位.....	35
4 系统时钟.....	36
4.1 概述.....	36
4.2 时钟框图.....	36
4.3 OSCM寄存器.....	36
4.4 系统高速时钟.....	37
4.4.1 内部高速RC振荡器.....	37
4.4.2 外部高速时钟.....	38
4.4.2.1 石英/陶瓷振荡器.....	38
4.4.2.2 RC振荡器.....	39
4.4.2.3 外部时钟源.....	39

4.5	系统低速时钟.....	40
4.5.1	系统时钟测试.....	40
5	系统工作模式.....	41
5.1	概述.....	41
5.2	系统模式切换.....	42
5.3	唤醒时间.....	43
5.3.1	概述.....	43
5.3.2	唤醒时间.....	43
6	中断.....	44
6.1	概述.....	44
6.2	中断请求使能寄存器INTEN.....	44
6.3	中断请求寄存器INTRQ.....	45
6.4	GIE全局中断.....	45
6.5	PUSH, POP处理.....	46
6.6	INT0 (P0.0) 中断.....	47
6.7	INT1 (P0.1) 中断.....	48
6.8	TC0 中断.....	49
6.9	TC1 中断.....	50
6.10	ADC中断.....	51
6.11	多中断操作举例.....	52
7	I/O口.....	53
7.1	I/O口模式.....	53
7.2	I/O上拉电阻寄存器.....	54
7.3	I/O口数据寄存器.....	54
7.4	P4 口ADC共用引脚.....	55
8	定时器.....	57
8.1	看门狗定时器.....	57
8.2	定时/计数器TC0.....	58
8.2.1	概述.....	58
8.2.2	TC0M模式寄存器.....	59
8.2.3	TC1X8, TC0X8, TC0GN标志.....	59
8.2.4	TC0C计数寄存器.....	60
8.2.5	TC0R自动装载寄存器.....	61
8.2.6	TC0 时钟频率输出 (蜂鸣器输出).....	62
8.2.7	TC0 操作流程.....	63
8.3	定时/计数器TC1.....	64
8.3.1	概述.....	64
8.3.2	TC1M模式寄存器.....	65
8.3.3	TC1X8 标志.....	65
8.3.4	TC1C计数寄存器.....	66
8.3.5	TC1R自动装载寄存器.....	67
8.3.6	TC1 时钟频率输出 (蜂鸣器输出).....	68
8.3.7	TC1 操作流程.....	69
8.4	PWM功能说明.....	70
8.4.1	概述.....	70
8.4.2	TCnIRQ和PWM输出占空比.....	71
8.4.3	PWM输出占空比和TCnR的变化.....	72
8.4.4	PWM编程举例.....	73
9	5+1 通道ADC.....	74
9.1	概述.....	74
9.2	ADM寄存器.....	74
9.3	ADR寄存器.....	75
9.4	ADB寄存器.....	75
9.5	P4CON寄存器.....	76
9.6	VREFH寄存器.....	76
9.7	AD转换时间.....	77
9.8	ADC操作实例.....	78
9.9	ADC电路.....	80
10	指令集.....	81
11	电气特性.....	82
11.1	极限参数.....	82
11.2	电气特性.....	82
12	开发工具.....	84
12.1	在线仿真器 (ICE).....	84

12.2	OTP烧录器 .....	84
12.3	SN8IDE .....	84
12.4	SN8P2711 EV KIT .....	85
12.4.1	PCB说明 .....	85
12.4.2	SN8P2711 EV KIT与SN8ICE 2K的连接 .....	87
12.5	OTP烧录信息 .....	88
12.5.1	烧录转接板信息 .....	88
12.5.2	烧录引脚信息 .....	90
13	封装信息 .....	91
13.1	P-DIP 14 PIN .....	91
13.2	SOP 14 PIN .....	92
13.3	SSOP 16 PIN .....	93
14	芯片正印命名规则 .....	94
14.1	概述 .....	94
14.2	芯片型号说明 .....	94
14.3	命名举例 .....	95
14.4	日期码规则 .....	95

# 1 产品简介

**SN8P2711A** 是 SN8P2711 的升级版，IHRC 提高到  $16\text{MHz}\pm 2\%$ 。产品优秀的高抗干扰性能为家电产品提供最佳的解决方案。

- SN8P2711A 兼容 SN8P2711;
- IHRC 性能由  $16\text{MHz}\pm 5\%$  提高到  $16\text{MHz}\pm 2\%$ ;
- 掉电复位性能更加准确;
- SN8P2711 的代码可直接用于 SN8P2711A。可以将 SN8P2711 的原始 SN8 档直接编程为 SN8P2711A，无需在 SN8P2711A 源代码中宣告和重新编译。

## 1.1 功能特性

- ◆ **存储器配置**  
OTP ROM 空间: 1K \* 16 位。  
RAM 空间: 64 字节。
- ◆ **4 层堆栈缓存器**
- ◆ **I/O 引脚配置**  
输入输出双向端口: P0、P4、P5。  
单向输入引脚: P0.4, 与复位引脚共用。  
具有唤醒功能的端口: P0 电平触发。  
内置上拉电阻端口: P0、P4、P5。  
外部中断引脚:  
P0.0: 由寄存器 PEDGE 控制;  
P0.1: 下降沿触发。
- ◆ **3 级低电压检测系统 (LVD)**  
系统复位, 监控系统电源。
- ◆ **5 个中断源**  
3 个内部中断: TC0、TC1、ADC。  
2 个外部中断: INT0、INT1。
- ◆ **强大的指令系统**  
**单时钟系统 (1T)。**  
大部分指令只需要一个时钟周期。  
跳转指令 JMP 可在整个 ROM 区执行。  
调用指令 CALL 可在整个 ROM 区执行。  
查表指令 MOVc 可寻址整个 ROM 区。
- ◆ **5+1 通道 12 位 ADC.**  
5 个外部 ADC 输入。  
一个内部电池检测。  
内部 AD 参考电压 (VDD、4V、3V、2V)。
- ◆ **两个 8 位定时/计数器**  
TC0: 自动装载定时器/计数器/PWM0/ Buzzer 输出。  
TC1: 自动装载定时器/计数器/PWM1/ Buzzer 输出。
- ◆ **内置看门狗定时器, 其时钟源由内部低速 RC 振荡器提供 (16KHz @3V, 32KHz @5V)**
- ◆ **双时钟系统**  
外部高速时钟: RC 模式, 高达 10 MHz。  
外部高速时钟: 晶体模式, 高达 16 MHz。  
内部高速时钟: RC 模式, 高达 16MHz。  
内部低速时钟: RC 模式, 16KHz(3V), 32KHz(5V)。
- ◆ **工作模式**  
普通模式: 高、低速时钟同时工作。  
低速模式: 只有低速时钟工作。  
睡眠模式: 高、低速时钟都停止工作。  
绿色模式: 由 TC0 周期性的唤醒。
- ◆ **封装形式**  
P-DIP 14 pins。  
SOP 14 pins。  
SSOP 16 pins。

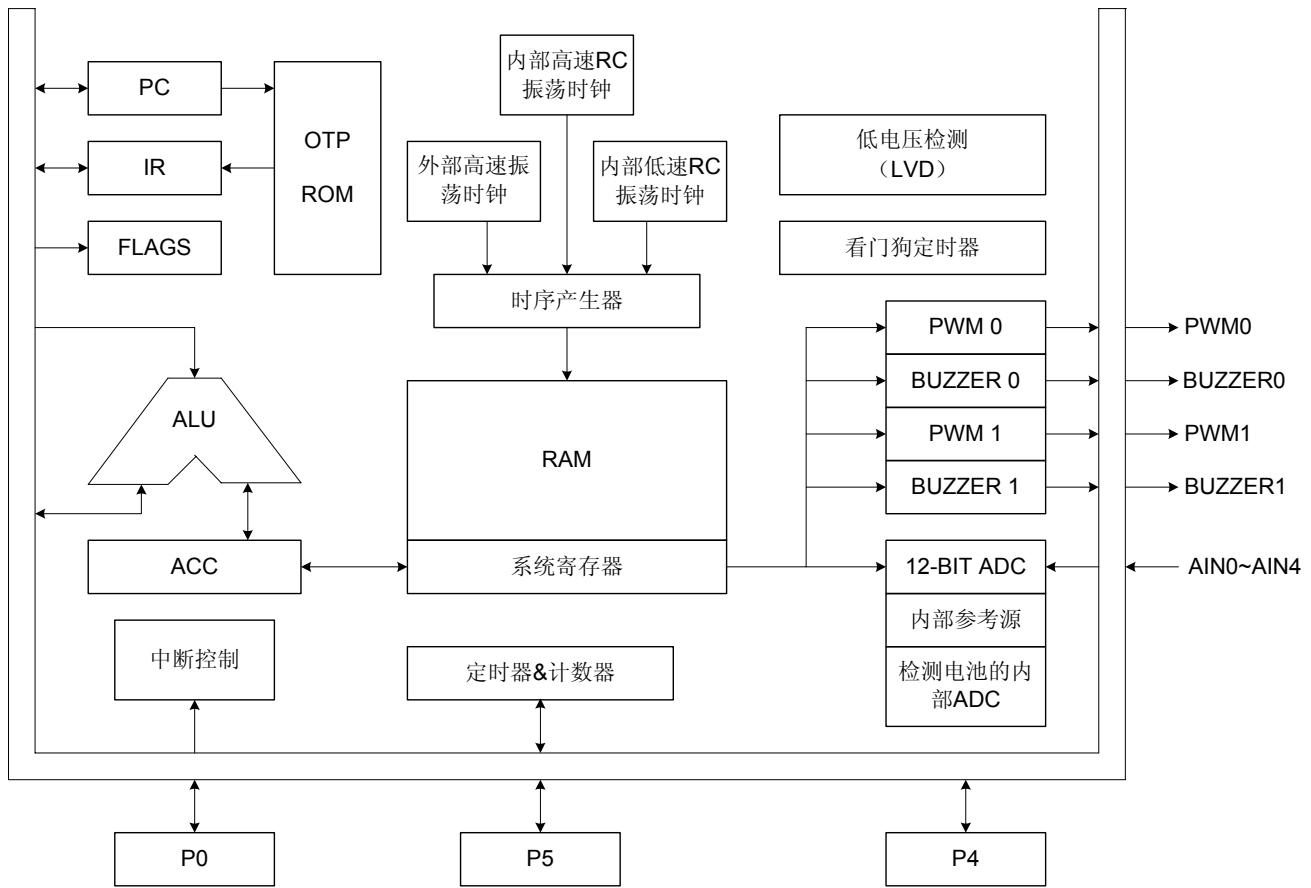
### 特性列表

单片机型号	ROM	RAM	堆栈	定时器		I/O	ADC	绿色模式	PWM	唤醒功能 引脚数目	封装形式
				TC0	TC1				Buzzer		
SN8P2711	1K*16	64	4	V	V	12	5+1 ch	V	2	5	P-DIP 14/SOP 14/SSOP 16
SN8P2711A	1K*16	64	4	V	V	12	5+1 ch	V	2	5	P-DIP 14/SOP 14/SSOP 16

### SN8P2711 升级为 SN8P2711A 注意事项

项目	SN8P2711	SN8P2711A
PCL	PCL 不能保存在地址 0xxFEH 和 0xxFFH 处	没有限制
32KHz 振荡器模式	不支持	支持
固件差别	SN8P2711 的 SN8 档可直接通过 MPIII Writer 烧录 SN8P2711A	SN8P2711A 的 SN8 档(如重新编译的 SN8P2711 的源代码而没有宣告称 SN8P2711A, SN8P2711 的新代码……) 不能直接通过 MPIII Writer 烧录 SN8P2711 的芯片。

## 1.2 系统结构框图



### 1.3 引脚配置

SN8P2711AP (P-DIP 14 pins)

SN8P2711AS (SOP 14 pins)

VDD	1	U	14	VSS
P0.3/XIN	2		13	P4.4/AIN4
P0.2/XOUT	3		12	P4.3/AIN3
P0.4/RST/VPP	4		11	P4.2/AIN2
P5.3/BZ1/PWM1	5		10	P4.1/AIN1
P5.4/BZ0/PWM0	6		9	P4.0/AIN0/VREFH
P0.1/INT1	7		8	P0.0/INT0

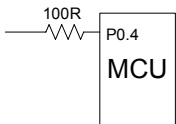
**SN8P2711AP**  
**SN8P2711AS**

SN8P2711AX (SSOP 16 pins)

VDD	1	U	16	VSS
P0.3/XIN	2		15	P4.4/AIN4
P0.2/XOUT	3		14	P4.3/AIN3
P0.4/RST/VPP	4		13	P4.2/AIN2
P5.3/BZ1/PWM1	5		12	P4.1/AIN1
P5.4/BZ0/PWM0	6		11	P4.0/AIN0/VREFH
P0.1/INT1	7		10	P0.0/INT0
NC	8		9	NC

**SN8P2711AX**

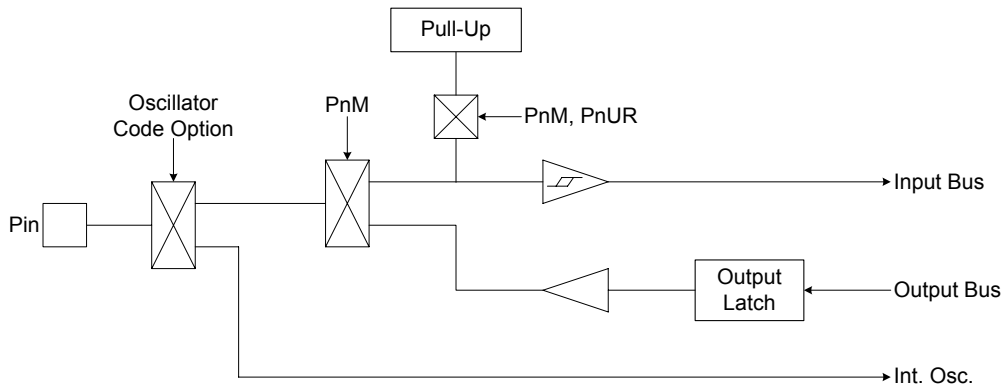
### 1.4 引脚说明

引脚名称	类型	功能说明
VDD, VSS	P	电源输入端。
P0.4/RST/VPP	I, P	<p>P0.4: 禁止外部复位时为单向输入引脚，施密特触发，<b>无内置上拉电阻</b>，作普通输入引脚使用时，用户需在单片机的 <b>P0.4</b> 外面串接一个 <b>100 欧姆</b> 的电阻（如右图所示，电阻要尽可能的靠近单片机），具有唤醒功能。</p> <p>RST: 系统复位输入引脚，施密特结构，低电平触发，通常保持高电平。</p> <p>VPP: OTP 烧录引脚。</p> 
P0.3/XIN	I/O	<p>P0.3: 双向输入/输出引脚，输入模式时为施密特触发，内置上拉电阻，具有唤醒功能。</p> <p>XIN: 使能外部振荡电路（晶体/RC 振荡电路）时为振荡信号输入引脚。</p>
P0.2/XOUT	I/O	<p>P0.2: 双向输入/输出引脚，输入模式时为施密特触发，内置上拉电阻，具有唤醒功能。</p> <p>XOUT: 使能外部晶体振荡器时为振荡器输出引脚。</p>
P0[1:0]/INT[1:0]	I/O	<p>双向输入/输出引脚，输入模式时为施密特触发，内置上拉电阻，具有唤醒功能。</p> <p>外部中断触发引脚（施密特触发）。</p> <p>TC1/TC0 事件计数器的信号输入引脚。</p>
P4.0/AIN0/VREFH	I/O	<p>P4.0: 双向输入/输出引脚，<b>非施密特触发</b>，内置上拉电阻。</p> <p>AIN0: ADC 输入通道。</p> <p>VERFH: ADC 参考电压的高电平输入引脚。</p>
P4.[4:1]/AIN[4:1]	I/O	<p>P4 [4:1]: 双向输入/输出引脚，<b>非施密特触发</b>，内置上拉电阻。</p> <p>AIN[4:1]: ADC 输入通道。</p>
P5[4:3]/BZ[1:0]/PWM[1:0]	I/O	<p>双向输入/输出引脚，输入模式时为施密特触发，内置上拉电阻。</p> <p>Buzzer 输出引脚/PWM 输出引脚。</p>

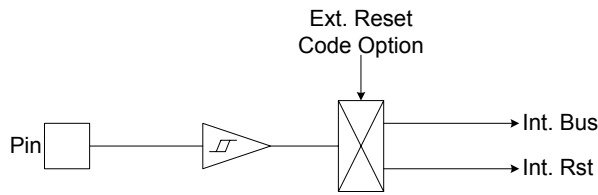


## 1.5 引脚电路结构图

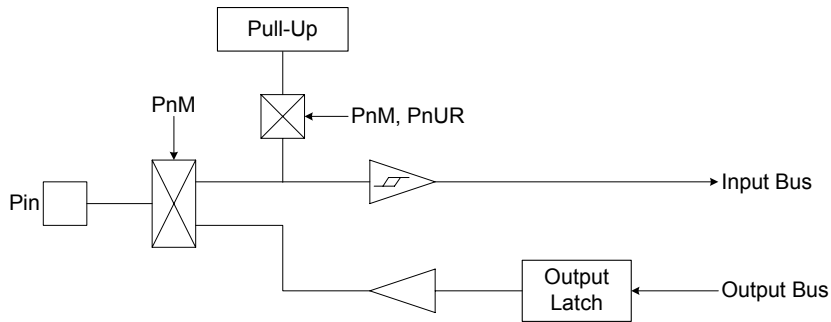
**P0.2、P0.3:**



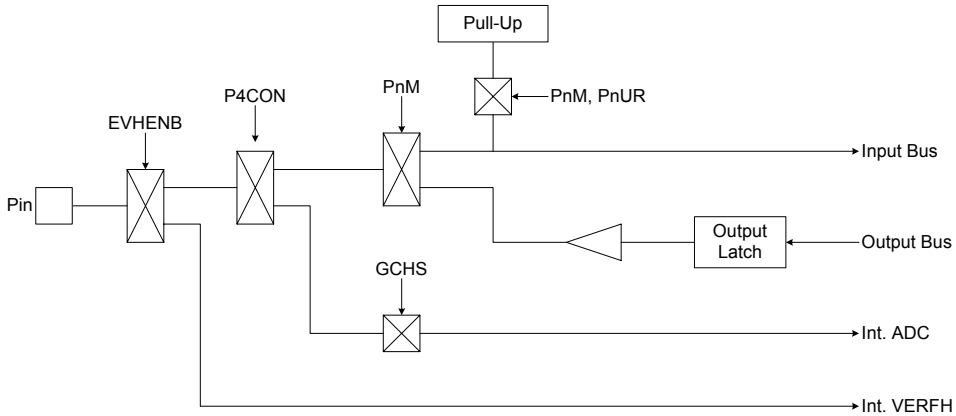
**P0.4:**



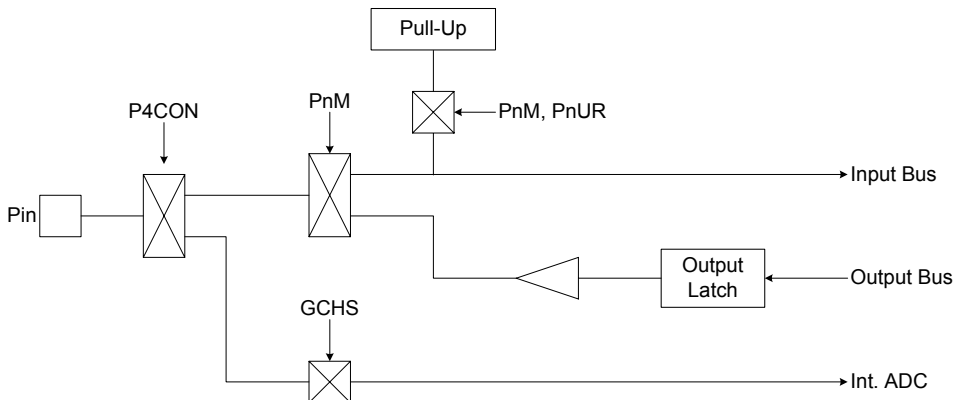
**P0、5:**



**P4.0:**



**P4:**

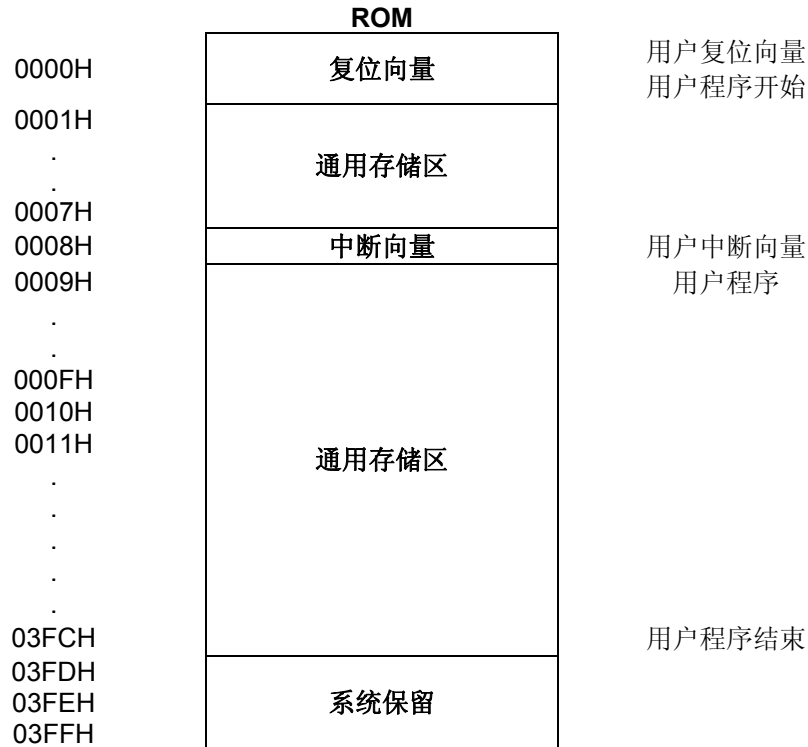


# 2 中央处理器（CPU）

## 2.1 存储器

### 2.1.1 程序存储器（ROM）

☞ ROM: 1K



#### 2.1.1.1 复位向量（0000H）

具有一个字长的系统复位向量（0000H）。

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

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

➤ 例：定义复位向量。

```

ORG      0          ;
JMP      START     ; 跳至用户程序。
...

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

```

## 2.1.1.2 中断向量 (0008H)

中断向量地址为 0008H。一旦有中断响应，程序计数器 PC 的当前值就会存入堆栈缓存器并跳转到 0008H 开始执行中断服务程序。0008H 处的第一条指令必须是“JMP”或“NOP”。下面的示例程序说明了如何编写中断服务程序。

\* 注：“PUSH”，“POP”指令用于存储和恢复 ACC/PFLAG，NT0、NTD 不受影响。PUSH/POP 缓存器是唯一的，且仅有一层。

➤ 例：定义中断向量，中断服务程序紧随 **ORG 8H** 之后。

```
.CODE
    ORG      0
    JMP      START      ; 跳至用户程序。
    ...
    ORG      8H          ; 中断向量。
    PUSH     ; 保存 ACC 和 PFLAG。
    ...
    POP      ; 恢复 ACC 和 PFLAG。
    RETI     ; 中断结束。
START:
    ...
    ; 用户程序开始。
    ...
    JMP      START      ; 用户程序结束。
    ...
    ENDP           ; 程序结束。
```

➤ 例：定义中断向量，中断程序在用户程序之后。

```
.CODE
    ORG      0
    JMP      START      ; 跳至用户程序。
    ...
    ORG      8H          ; 中断向量。
    JMP      MY_IRQ     ; 跳至中断程序。
START:
    ORG      10H         ; 用户程序开始。
    ...
    JMP      START      ; 用户程序结束。
MY_IRQ:
    ...
    ; 中断程序开始。
    PUSH     ; 保存 ACC 和 PFLAG。
    ...
    POP      ; 恢复 ACC 和 PFLAG。
    RETI     ; 中断程序结束。
    ...
    ENDP           ; 程序结束。
```

\* 注：从上面的程序中容易得出 SONiX 的编程规则，有以下几点：

1. 地址 0000H 的“JMP”指令使程序从头开始执行；
2. 地址 0008H 是中断向量；
3. 用户的程序应该是一个循环。

## 2.1.1.3 查表

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

➤ 例：查找 ROM 地址为“TABLE1”的值。

```

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

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

```

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

➤ 例：宏 INC\_YZ。

```

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

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

```

➤ 例：通过“INC\_YZ”对上例进行优化。

```

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

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

```

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

➤ 例: 由指令 **B0ADD/ADD** 对 Y 和 Z 寄存器加 1。

```

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

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

        B0BTS1  FC          ; 检查进位标志。
        JMP     GETDATA     ; FC = 0。
        INCMS   Y           ; FC = 1。
        NOP

GETDATA:
        MOV    ;
        MOV    ; 存储数据, 如果 BUF = 0, 数据为 0035H。
        MOV    ; 如果 BUF = 1, 数据=5105H。
        MOV    ; 如果 BUF = 2, 数据=2012H。

TABLE1:
        DW     0035H        ; 定义数据表 (16 位) 数据。
        DW     5105H
        DW     2012H
        ...

```

## 2.1.1.4 跳转表

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

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

## ➤ 例：跳转表。

```

ORG      0100H      ; 跳转表从 ROM 前端开始。

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

```

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

## ➤ 例：宏“MACRO3.H”中，“@JMP\_A”的应用。

```

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

```

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

## ➤ 例：如果跳转表跨越 ROM 边界，将引起程序错误。

```

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

```

\* 注：“VAL”为跳转表列表中列表个数。

## ➤ 例：“@JMP\_A”运用举例

; 编译前  
ROM 地址

```

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

```

; 编译后  
ROM 地址

```

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

```

## 2.1.1.5 CHECKSUM 计算

ROM 的最后一个地址是系统保留区，用户应该在计算 Checksum 时跳过该区域。

➤ 例：下面的程序说明如何从 00H 至用户代码结束的区域内进行 Checksum 计算。

```

MOV      A,#END_USER_CODE$L
B0MOV   END_ADDR1, A      ; 用户程序结束地址低位地址存入end_addr1。
MOV      A,#END_USER_CODE$M
B0MOV   END_ADDR2, A      ; 用户程序结束地址中间地址存入end_addr2。
CLR      Y                ; 清 Y。
CLR      Z                ; 清 Z。

@@:
MOV      FC
B0BCLR  FC                ; 清标志位 C。
ADD      DATA1, A
MOV      A, R
ADC      DATA2, A
JMP      END_CHECK        ; 检查 YZ 地址是否为代码的结束地址。

AAA:
INCMS   Z
JMP     @B                ; 若 Z != 00H, 进行下一个计算。
JMP     Y_ADD_1           ; 若 Z = 00H, Y+1。

END_CHECK:
MOV     A, END_ADDR1
CMPRS  A, Z                ; 检查 Z 地址是否为用户程序结束地址低位地址。
JMP     AAA                ; 否, 则进行 Checksum 计算。
MOV     A, END_ADDR2
CMPRS  A, Y                ; 是则检查 Y 的地址是否为用户程序结束地址中间地址。
JMP     AAA                ; 否, 则进行 Checksum 计算。
JMP     CHECKSUM_END      ; 是则 Checksum 计算结束。

Y_ADD_1:
INCMS   Y
NOP
JMP     @B                ; 跳转到 Checksum 计算。

CHECKSUM_END:
...
...
END_USER_CODE:           ; 程序结束。

```

## 2.1.2 编译选项表 (CODE OPTION)

编译选项	内容	功能说明
High_Clk	IHRC_16M	高速时钟采用内部 16MHz RC 振荡电路, XIN/XOUT (P0.3/P0.2) 为普通的 I/O 引脚。
	RC	外部高速时钟振荡器采用廉价的 RC 振荡电路, XOUT (P0.2) 为普通的 I/O 引脚。
	12M X'tal	外部高速时钟振荡器采用高频晶体/陶瓷振荡器 (如 12MHz)。
	4M X'tal	外部高速时钟振荡器采用标准晶体/陶瓷振荡器 (如 4MHz)。
Watch_Dog	Always_On	始终开启看门狗定时器, 即使在睡眠模式和绿色模式下也处于开启状态。
	Enable	开启看门狗定时器, 但在睡眠模式和绿色模式下关闭。
	Disable	关闭看门狗定时器。
Fcpu	Fhosc/1	指令周期 = 1 个时钟周期, 必须关闭杂讯滤波功能。
	Fhosc/2	指令周期 = 2 个时钟周期, 必须关闭杂讯滤波功能。
	Fhosc/4	指令周期 = 4 个时钟周期。
	Fhosc/8	指令周期 = 8 个时钟周期。
	Fhosc/16	指令周期 = 16 个时钟周期。
Reset_Pin	Reset	使能外部复位引脚。
	P04	P0.4 为单向输入引脚, 无上拉电阻。
Security	Enable	ROM 程序加密。
	Disable	ROM 程序不加密。
Noise_Filter	Enable	使能杂讯滤除功能, Fcpu = Fosc/4~Fosc/16。
	Disable	禁止杂讯滤除功能, Fcpu = Fosc/1~Fosc/16。
LVD	LVD_L	VDD 低于 2.0V 时, 系统复位。
	LVD_M	VDD 低于 2.0V 时, 系统复位; PFLAG 寄存器的 LVD24 位作为 2.4V 低电压监测器。
	LVD_H	VDD 低于 2.4V 时, 系统复位; PFLAG 寄存器的 LVD36 位作为 3.6V 低电压监测器。

## \* 注:

1. 在干扰较大的情况下, 建议开启杂讯滤波功能, 此时 Fcpu = Fosc/4 ~ Fosc/128, 并将 Watch\_Dog 设置为 "Always\_On";
2. 如果用户定义看门狗为 "Always\_On", 编译器会自动开启看门狗定时器;
3. 编译选项 Fcpu 仅针对外部高速时钟, 在低速模式下 Fcpu = Fosc/4。

## 2.1.3 数据存储器 (RAM)

RAM: 64 字节

地址	RAM	
000H	通用存储区	
"		
"		
"		
"		
03FH	系统寄存器	
080H		
"		
"		
"		
0FFH	Bank 0 的结束区	

Bank0 的 080H~0FFH 是系统寄存器区 (128 字节)。



## 2.1.4 系统寄存器

### 2.1.4.1 系统寄存器列表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-	-	R	Z	Y	-	PFLAG	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	P4CON	VREFH
B	-	ADM	ADB	ADR	-	-	-	-	P0M	-	-	-	-	-	-	PEDGE
C	-	-	-	-	P4M	P5M	-	-	INTRQ	INTEN	OSCM	-	WDTR	TC0R	PCL	PCH
D	P0	-	-	-	P4	P5	-	-	T0M	-	TC0M	TC0C	TC1M	TC1C	TC1R	STKP
E	P0UR	-	-	-	P4UR	P5UR	-	@YZ	-	-	-	-	-	-	-	-
F	-	-	-	-	-	-	-	-	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

### 2.1.4.2 系统寄存器说明

R = 工作寄存器和 ROM 查表数据缓存器  
PFLAG = ROM 页和特殊标志寄存器  
VREFH = ADC 参考电压寄存器  
ADB = ADC 数据缓存器  
PnM = Pn 模式控制寄存器  
INTRQ = 中断请求寄存器  
OSCM = 振荡模式寄存器  
TC0R = TC0 自动装载数据缓存器  
Pn = Pn 数据缓存器  
TC0M = TC0 模式寄存器  
TC1M = TC1 模式寄存器  
TC1R = TC1 自动装载实际缓存器  
PnUR = Pn 上拉电阻控制寄存器  
STK0~STK3 = 堆栈寄存器

Y, Z = 专用寄存器, @YZ 间接寻址寄存器, ROM 寻址寄存器  
P4CON = P4 配置控制寄存器  
ADM = ADC 模式寄存器  
ADR = ADC 精度选择寄存器  
PEDGE = P0.0 模式控制寄存器  
INTEN = 中断使能寄存器  
WDTR = 看门狗清零寄存器  
PCH, PCL = 程序计数器  
T0M = TC0/TC1 加速和 TC0 唤醒功能寄存器  
TC0C = TC0 计数寄存器  
TC1C = TC1 计数寄存器  
STKP = 堆栈指针  
@YZ = 间接寻址寄存器

## 2.1.4.3 系统寄存器的位定义

地址	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	注释
082H	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0	R/W	R
083H	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0	R/W	Z
084H	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0	R/W	Y
086H	NT0	NPD	LVD36	LVD24		C	DC	Z	R/W	PFLAG
0AEH				P4CON4	P4CON3	P4CON2	P4CON1	P4CON0	R/W	P4CON
0AFH	EVHENB						VHS1	VHS2	R/W	VREFH
0B1H	ADENB	ADS	EOC	GCHS		CHS2	CHS1	CHS0	R/W	ADM
0B2H	ADB11	ADB10	ADB9	ADB8	ADB7	ADB6	ADB5	ADB4	R	ADB
0B3H		ADCKS1		ADCKS0	ADB3	ADB2	ADB1	ADB0	R/W	ADR
0B8H					P03M	P02M	P01M	P00M	R/W	P0M
0BFH				P00G1	P00G0				R/W	PEDGE
0C4H				P44M	P43M	P42M	P41M	P40M	R/W	P4M
0C5H				P54M	P53M				R/W	P5M
0C8H	ADCIRQ	TC1IRQ	TC0IRQ				P01IRQ	P00IRQ	R/W	INTRQ
0C9H	ADCIEN	TC1IEN	TC0IEN				P01IEN	P00IEN	R/W	INTEN
0CAH				CPUM1	CPUM0	CLKMD	STPHX		R/W	OSCM
0CCH	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0	W	WDTR
0CDH	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0	W	TC0R
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH							PC9	PC8	R/W	PCH
0D0H				P04	P03	P02	P01	P00	R/W	P0
0D4H				P44	P43	P42	P41	P40	R/W	P4
0D5H				P54	P53				R/W	P5
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	TC1CKS	ALOAD1	TC1OUT	PWM1OUT	R/W	TC1M
0DDH	TC1C7	TC1C6	TC1C5	TC1C4	TC1C3	TC1C2	TC1C1	TC1C0	R/W	TC1C
0DEH	TC1R7	TC1R6	TC1R5	TC1R4	TC1R3	TC1R2	TC1R1	TC1R0	W	TC1R
0DFH	GIE					STKPB2	STKPB1	STKPB0	R/W	STKP
0E0H					P03R	P02R	P01R	P00R	W	P0UR
0E4H				P44R	P43R	P42R	P41R	P40R	W	P4UR
0E5H				P54R	P53R				W	P5UR
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H							S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH							S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH							S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH							S0PC9	S0PC8	R/W	STK0H

\* 注:

1. 所有寄存器的名称在 SN8ASM 编译器中做了宣告;
2. 寄存器中各位的名称已在 SN8ASM 编译器中以“F”为前缀定义过;
3. 指令“B0BSET”、“B0BCLR”、“BSET”、“BCLR”只能用于“R/W”寄存器。

#### 2.1.4.4 累加器

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

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

➤ **例：读/写 ACC。**

; 数据写入 ACC。

```
MOV      A, #0FH
```

; 读取 ACC 中的数据并存入 BUF。

```
MOV      BUF, A
B0MOV    BUF, A
```

; BUF 中的数据写入 ACC。

```
MOV      A, BUF
B0MOV    A, BUF
```

系统执行中断操作时，ACC 和 PFLAG 中的数据不会自动存储，用户需通过程序将中断入口处的 ACC 和 PFLAG 中的数据送入存储器进行保存。可通过“PUSH”和“POP”指令对 ACC 和 PFLAG 等系统寄存器进行存储及恢复。

➤ **例：ACC 和工作寄存器中断保护操作。**

INT\_SERVICE:

```
PUSH                                ; 保存 PFLAG 和 ACC。
```

```
...
```

```
POP                                  ; 恢复 ACC 和 PFLAG。
```

```
RETI                                 ; 退出中断。
```

### 2.1.4.5 程序状态寄存器 PFLAG

寄存器 PFLAG 中包含 ALU 运算状态信息、系统复位状态信息和 LVD 检测信息，其中，位 NT0 和 NPD 显示系统复位状态信息，包括上电复位、LVD 复位、外部复位和看门狗复位；位 C、DC 和 Z 显示 ALU 的运算信息。位 LVD24 和 LVD36 显示了单片机供电电压状况。

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>PFLAG</b>	NT0	NPD	LVD36	LVD24	-	C	DC	Z
读/写	R/W	R/W	R	R	-	R/W	R/W	R/W
复位后	X	X	0	0	-	0	0	0

Bit [7:6] **NT0, NPD**: 复位状态标志。

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

Bit 5 **LVD36**: 3.6V LVD 工作电压标志，LVD 编译选项为 LVD\_H 时有效。  
0 = 系统工作电压 VDD 超过 3.6V，低电压检测器没有工作；  
1 = 系统工作电压 VDD 低于 3.6V，说明此时低电压检测器已处于监控状态。

Bit 4 **LVD24**: 2.4V LVD 工作电压标志，LVD 编译选项为 LVD\_M 时有效。  
0 = 系统工作电压 VDD 超过 2.4V，低电压检测器没有工作；  
1 = 系统工作电压 VDD 低于 2.4V，说明此时低电压检测器已处于监控状态。

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

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

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

\* 注：关于标志位 C、DC 和 Z 的更多信息请参阅指令集相关内容。

### 2.1.4.6 程序计数器

程序计数器 PC 是一个 10 位二进制程序地址寄存器，分高 2 位和低 8 位。专门用来存放下一条需要执行指令的内存地址。通常，程序计数器会随程序中指令的执行自动增加。

若程序执行 CALL 和 JMP 指令时，PC 指向特定的地址。

	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>PC</b>	-	-	-	-	-	-	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
复位后	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

#### ☞ 单地址跳转

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

如果位测试为真，PC 加 2。

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

```
...
C0STEP:   NOP
```

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

```
...
...
C1STEP:   NOP
```

如果 ACC 等于指定的立即数则 PC 值加 2，跳过下一条指令。

```
CMPRS    A, #12H     ; 如果 ACC = 12H，则跳过下一条指令。
JMP      C0STEP     ; 否则跳至 C0STEP。
```

```
...
...
C0STEP:   NOP
```

执行加 1 指令后，结果为零时，PC 的值加 2，跳过下一条指令。

**INCS:**

```
INCS     BUF0
JMP      C0STEP
```

```
...
C0STEP:   NOP
```

**INCMS:**

```
INCMS    BUF0
JMP      C0STEP
```

```
...
C0STEP:   NOP
```

执行减 1 指令后，结果为零时，PC 的值加 2，跳过下一条指令。

**DECS:**

```
DECS     BUF0
JMP      C0STEP
```

```
...
C0STEP:   NOP
```

**DECMS:**

```
DECMS    BUF0
JMP      C0STEP
```

```
...
C0STEP:   NOP
```

## ☞ 多地址跳转

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

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

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

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

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

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

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

### 2.1.4.7 Y, Z 寄存器

寄存器 Y 和 Z 都是 8 位缓存器，主要用途如下：

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

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

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

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

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

➤ 例：利用数据指针@YZ 对 RAM 数据清零。

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

CLR\_YZ\_BUF:

```
CLR      @YZ          ; @YZ 清零。
```

```
DECMS   Z            ;
JMP     CLR_YZ_BUF   ; 不为零。
```

```
CLR      @YZ
```

END\_CLR:

```
...
```

### 2.1.4.8 R 寄存器

8 位缓存器 R 主要有以下两个功能：

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

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

## 2.2 寻址模式

### 2.2.1 立即寻址

将立即数送入 ACC 或指定的 RAM 单元。

- 例：立即数 12H 送入 ACC。  
MOV           A, #12H
- 例：立即数 12H 送入寄存器 R。  
B0MOV        R, #12H

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

### 2.2.2 直接寻址

通过 ACC 对 RAM 单元数据进行操作。

- 例：地址 12H 处的内容送入 ACC。  
B0MOV        A, 12H
- 例：ACC 中数据写入 RAM 的 12H 单元。  
B0MOV        12H, A

### 2.2.3 间接寻址

通过指针寄存器 (Y/Z) 访问 RAM 数据。

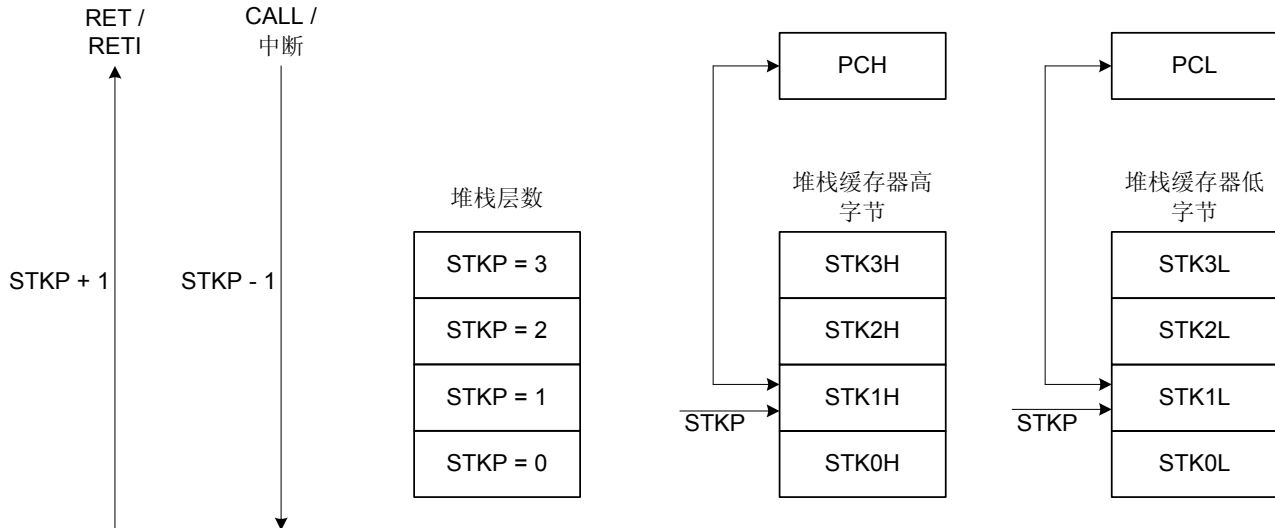
- 例：用 @YZ 实现间接寻址。  
B0MOV        Y, #0                   ; Y 清零以寻址 RAM bank 0。  
B0MOV        Z, #12H               ; 设定寄存器地址。  
B0MOV        A, @YZ



## 2.3 堆栈

### 2.3.1 概述

SN8P2711A 的堆栈缓存器共 4 层，程序进入中断或执行 CALL 指令时，用来存储程序计数器 PC 的值。寄存器 STKP 为堆栈指针，STK<sub>n</sub>H 和 STK<sub>n</sub>L 分别是各堆栈缓存器的高、低字节。



### 2.3.2 堆栈寄存器

堆栈指针 STKP 是一个 3 位寄存器，存放被访问的堆栈单元地址，10 位数据存储器 STK<sub>n</sub>H 和 STK<sub>n</sub>L 用于暂存堆栈数据。以上寄存器都位于 bank 0。

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

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

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

Bit[2:0] **STKPB<sub>n</sub>**: 堆栈指针 (n = 0 ~ 2)。

Bit 7 **GIE**: 全局中断控制位。

0 = 禁止;

1 = 使能。

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

```
MOV     A, #0000111B
B0MOV  STKP, A
```

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>STK<sub>n</sub>H</b>	-	-	-	-	-	-	SnPC9	SnPC8
读/写	-	-	-	-	-	-	R/W	R/W
复位后	-	-	-	-	-	-	0	0

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>STK<sub>n</sub>L</b>	SnPC7	SnPC6	SnPC5	SnPC4	SnPC3	SnPC2	SnPC1	SnPC0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

**STK<sub>n</sub> = STK<sub>n</sub>H, STK<sub>n</sub>L (n = 3 ~ 0)。**

### 2.3.3 堆栈操作举例

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

堆栈层数	STKP			缓存器		备注
	STKPB2	STKPB1	STKPB0	高字节	低字节	
0	1	1	1	保留	保留	-
1	1	1	0	STK0H	STK0L	-
2	1	0	1	STK1H	STK1L	-
3	1	0	0	STK2H	STK2L	-
4	0	1	1	STK3H	STK3L	-
> 4	0	1	0	-	-	缓存器已满，错误

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			缓存器		备注
	STKPB2	STKPB1	STKPB0	高字节	低字节	
4	0	1	1	STK3H	STK3L	-
3	1	0	0	STK2H	STK2L	-
2	1	0	1	STK1H	STK1L	-
1	1	1	0	STK0H	STK0L	-
0	1	1	1	保留	保留	-

# 3 复位

## 3.1 概述

SN8P2711A 有以下几种复位方式：

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

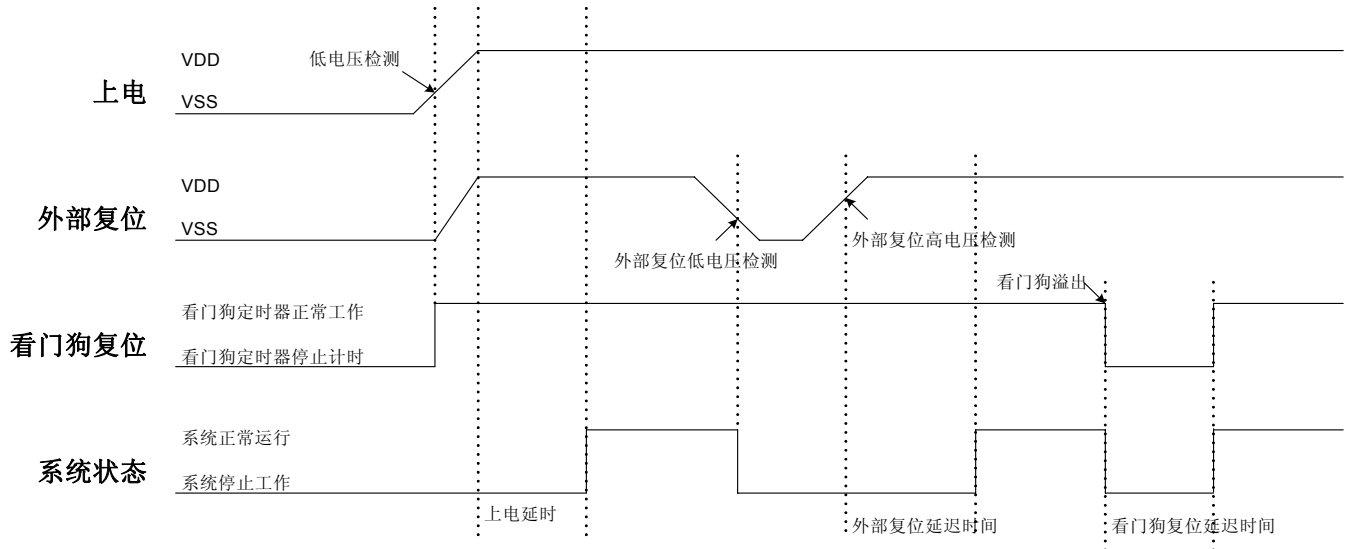
上述任一种复位发生时，所有的系统寄存器恢复默认状态，程序停止运行，同时程序计数器 PC 清零。复位结束后，系统从向量 0000H 处重新开始运行。PFLAG 寄存器的 NT0 和 NPD 两个标志位能够给出系统复位状态的信息。用户可以编程控制 NT0 和 NPD，从而控制系统的运行路径。

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>PFLAG</b>	NT0	NPD	LVD36	LVD24	-	C	DC	Z
读/写	R/W	R/W	R	R	-	R/W	R/W	R/W
复位后	X	X	0	0	-	0	0	0

Bit [7:6] **NT0, NPD**: 复位状态标志。

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

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



## 3.2 上电复位

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

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

## 3.3 看门狗复位

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

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

### 看门狗定时器应用注意事项：

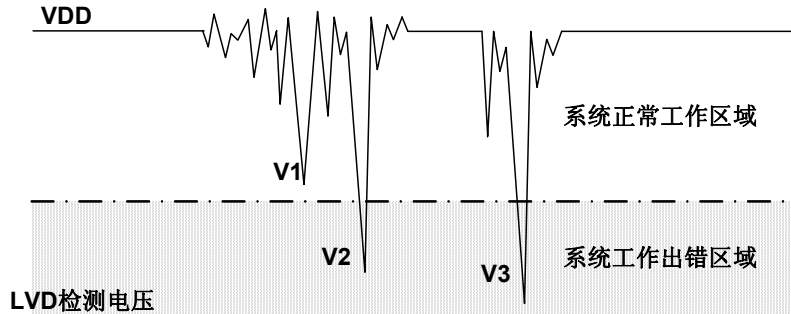
- 对看门狗清零之前，检查 I/O 口的状态和 RAM 的内容可增强程序的可靠性；
- 不能在中断中对看门狗清零，否则无法侦测到主程序跑飞的状况；
- 程序中应该只在主程序中有一次清看门狗的动作，这种架构能够最大限度的发挥看门狗的保护功能。

\* 注：关于看门狗定时器的详细内容，请参阅“看门狗定时器”有关章节。

## 3.4 掉电复位

### 3.4.1 概述

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



掉电复位示意图

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

#### DC 运用中：

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

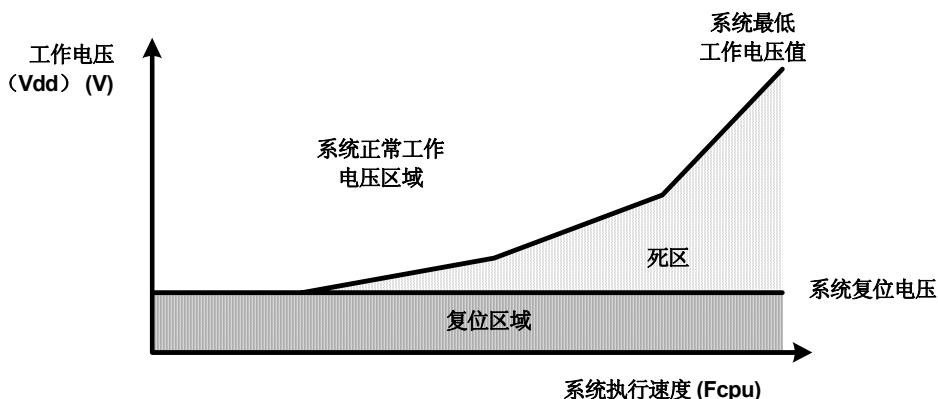
#### AC 运用中：

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

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

### 3.4.2 系统工作电压

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



系统工作电压与执行速度关系图

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

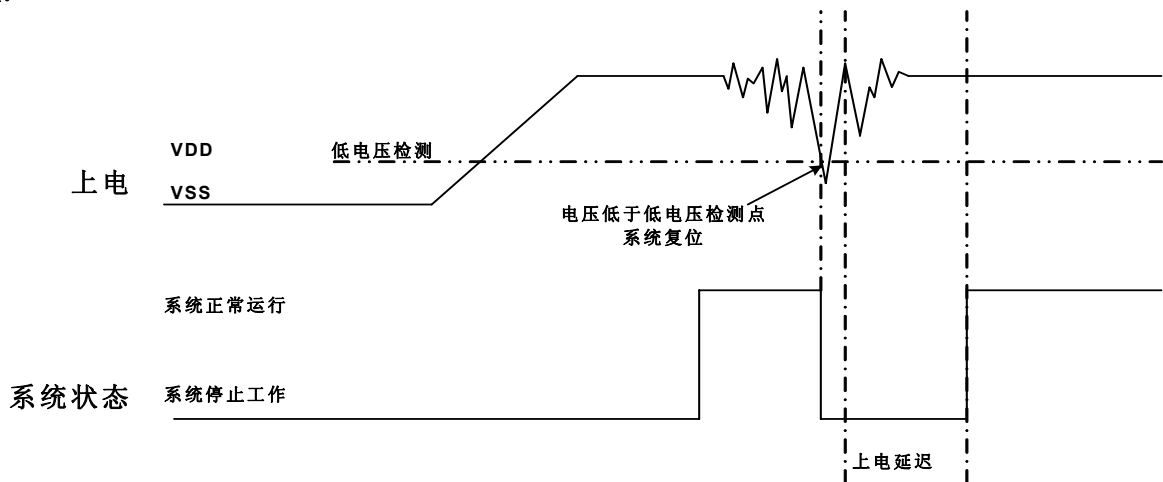
### 3.4.3 掉电复位性能改进

如何改善系统掉电复位性能，有以下几点建议：

- LVD 复位；
- 看门狗复位；
- 降低系统工作速度；
- 采用外部复位电路（稳压二极管复位电路，电压偏移复位电路，外部 IC 复位）。

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

LVD 复位：



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

LVD 设计为三层结构（2.0V/2.4V/3.6V），由 LVD 编译选项控制。对于上电复位和掉电复位，2.0V LVD 始终处于使能状态；2.4V LVD 具有 LVD 复位功能，并能通过标志位显示 VDD 状态；3.6V LVD 具有标记功能，可显示 VDD 的工作状态。LVD 标志功能只是一个低电压检测装置，标志位 LVD24 和 LVD36 给出 VDD 的电压情况。对于低电压检测应用，只需查看 LVD24 和 LVD36 的状态即可检测电池状况。

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>PFLAG</b>	NT0	NPD	LVD36	LVD24	-	C	DC	Z
读/写	R/W	R/W	R	R	-	R/W	R/W	R/W
复位后	X	X	0	0	-	0	0	0

Bit 5 **LVD36**: 3.6V LVD 工作电压标志，LVD 编译选项为 LVD\_H 时有效。  
0 = 系统工作电压 VDD 超过 3.6V，低电压检测器没有工作；  
1 = 系统工作电压 VDD 低于 3.6V，说明此时低电压检测器已处于监控状态。

Bit 4 **LVD24**: 2.4V LVD 工作电压标志，LVD 编译选项为 LVD\_M 时有效。  
0 = 系统工作电压 VDD 超过 2.4V，低电压检测器没有工作；  
1 = 系统工作电压 VDD 低于 2.4V，说明此时低电压检测器已处于监控状态。

LVD	LVD 编译选项		
	LVD_L	LVD_M	LVD_H
2.0V 复位	有效	有效	有效
2.4V 标志	-	有效	-
2.4V 复位	-	-	有效
3.6V 标志	-	-	有效

**LVD\_L**

如果  $VDD < 2.0V$ ，系统复位；  
LVD24 和 LVD36 标志位无意义。

**LVD\_M**

如果  $VDD < 2.0V$ ，系统复位；  
LVD24: 如果  $VDD > 2.4V$ ，LVD24 = 0；如果  $VDD \leq 2.4V$ ，LVD24 = 1；  
LVD36 标志位无意义。

**LVD\_H**

如果  $VDD < 2.4V$ ，系统复位；  
LVD36: 如果  $VDD > 3.6V$ ，LVD36 = 0；如果  $VDD \leq 3.6V$ ，LVD36 = 1；  
LVD24 标志位无意义。

**\* 注:**

- a) LVD 复位结束后，LVD24 和 LVD36 都将被清零；
- b) LVD 2.4V 和 LVD3.6V 检测电平值仅作为设计参考，不能用作芯片工作电压值的精确检测。

**看门狗复位:**

看门狗定时器用于保证系统正常工作。通常，会在主程序中将看门狗定时器清零，但不要在多个分支程序中清看门狗。若程序正常运行，看门狗不会复位。当系统进入死区或程序运行出错的时候，看门狗定时器继续计数直至溢出，系统复位。如果看门狗复位后电源仍处于死区，则系统复位失败，保持复位状态，直到系统工作状态恢复到正常值。

**降低系统工作速度:**

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

**附加外部复位电路:**

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

### 3.5 外部复位

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

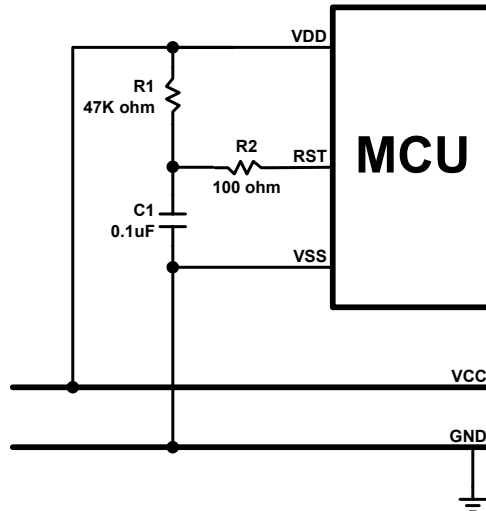
- **外部复位（当且仅当外部复位引脚为使能状态）：**系统检测复位引脚的状态，如果复位引脚不为高电平，则系统会一直保持在复位状态，直到外部复位结束；
- **系统初始化：**初始化所有的系统寄存器；
- **振荡器开始工作：**振荡器开始提供系统时钟；
- **执行程序：**上电结束，程序开始运行。

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



## 3.6 外部复位电路

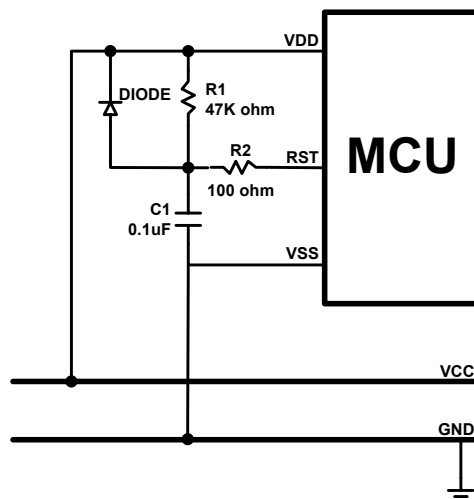
### 3.6.1 RC 复位电路



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

\* 注：此 RC 复位电路不能解决非正常上电和掉电复位问题。

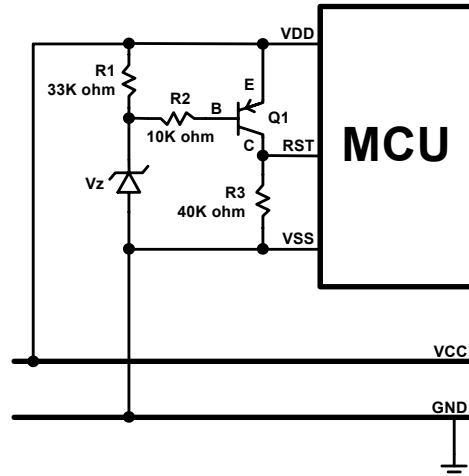
### 3.6.2 二极管及 RC 复位电路



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

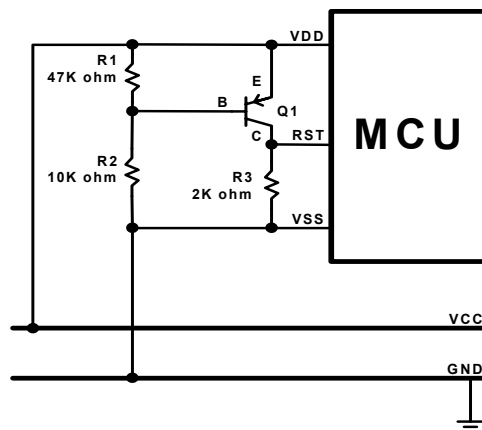
\* 注：“基本 RC 复位电路”和“二极管及 RC 复位电路”中的电阻 R2 都是必不可少的限流电阻，以避免复位引脚 ESD (Electrostatic Discharge) 或 EOS (Electrical Over-stress) 击穿。

### 3.6.3 稳压二极管复位电路



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

### 3.6.4 电压偏置复位电路

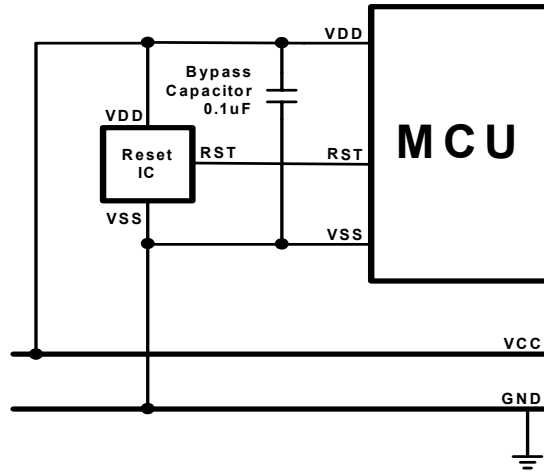


电压偏置复位电路是一种简单的 LVD 电路，基本上可以完全解决掉电复位问题。与稳压二极管复位电路相比，这种复位电路的检测电压值的精确度有所降低。电路中，R1 和 R2 构成分压电路，当 VDD 高于和等于分压值“ $0.7V \times (R1 + R2) / R1$ ”时，三极管集电极 C 输出高电平，单片机正常工作；VDD 低于“ $0.7V \times (R1 + R2) / R1$ ”时，集电极 C 输出低电平，单片机复位。

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

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

### 3.6.5 外部 IC 复位



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

# 4 系统时钟

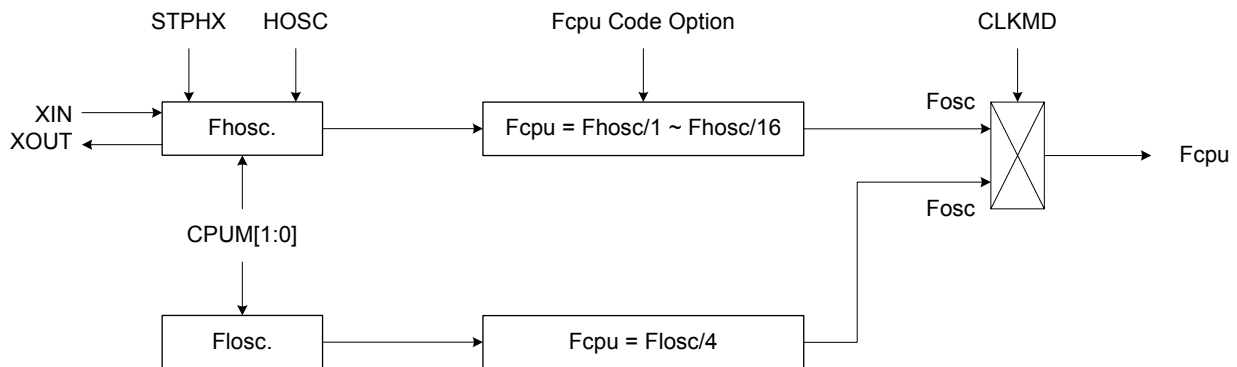
## 4.1 概述

SN8P2711A内带双时钟系统：高速时钟和低速时钟。高速时钟由外部晶振和内置的16MHz RC振荡电路（IHRC 16KHz）提供，低速时钟由内置的低速RC振荡电路（ILRC 16KHz @3V, 32KHz @5V）提供。两种时钟都可作为系统时钟源Fosc，系统工作在低速模式时，Fosc 4分频后作为一个指令周期。

- ☞ 普通模式 (高速时钟):  $F_{cpu} = F_{osc} / N$ ,  $N = 1 \sim 16$ ,  $F_{cpu}$  的编译选项决定 N 的值。
- ☞ 低速模式 (低速时钟):  $F_{cpu} = F_{osc}/4$ 。

在干扰较严重的运用条件下，SONiX 提供的杂讯滤波器能够对外部干扰进行隔离以保护系统的正常工作。

## 4.2 时钟框图



- HOSC: High\_Clk 编译选项。
- Fhosc: 外部高速/内部 RC 振荡器时钟频率。
- Fosc: 内部低速 RC 时钟频率（16KHz@3V, 32KHz@5V）。
- Fosc: 系统时钟频率。
- Fcpu: 指令执行频率。

## 4.3 OSCM 寄存器

寄存器 OSCM 控制振荡器的状态和系统的工作模式。

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

- Bit 1     **STPHX**: 外部高速振荡器控制位。  
0 = 运行;  
1 = 停止, 内部低速 RC 振荡器仍然运行。
- Bit 2     **CLKMD**: 系统时钟模式控制位。  
0 = 普通 (双时钟) 模式, 高速时钟作为系统时钟;  
1 = 低速模式, 低速时钟作为系统时钟。
- Bit[4:3]   **CPUM[1:0]**: CPU 工作模式控制位。  
00 = 普通模式; 01 = 睡眠模式; 10 = 绿色模式; 11 = 系统保留。

- 例: 停止高速振荡器。  
B0BSET     FSTPHX                   ; 停止外部高速振荡器。

## 4.4 系统高速时钟

内部 16MHz RC 振荡器和外部振荡器都可作为系统高速时钟源，由编译选项“High\_Clk”控制。

High_Clk	说明
IHRC	内部 16MHz RC 振荡器作为系统时钟源，XIN 和 XOUT 引脚为通用 I/O 口。
RC	外部 RC 振荡器为系统高速时钟，XOUT 引脚为通用 I/O 口。
32K	外部 32768Hz 低速振荡器作为系统高速时钟。
12M	外部高速振荡器作为系统高速时钟，典型频率为 12MHz。
4M	外部振荡器作为系统高速时钟，典型频率为 4MHz。

### 4.4.1 内部高速 RC 振荡器

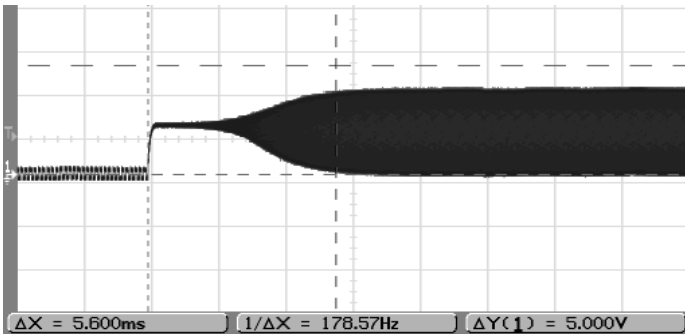
编译选项“IHRC\_16M”和“IHRC\_RTC”控制单片机的内置 RC 高速时钟（16MHz）。若选择“IHRC\_16M”，则内置 16MHz RC 振荡器作为系统时钟源，XIN 和 XOUT 引脚作为通用 I/O 口。

- **IHRC:** 系统高速时钟来自内置 16MHz RC 振荡器，XIN/XOUT 引脚作为普通的 I/O 引脚。

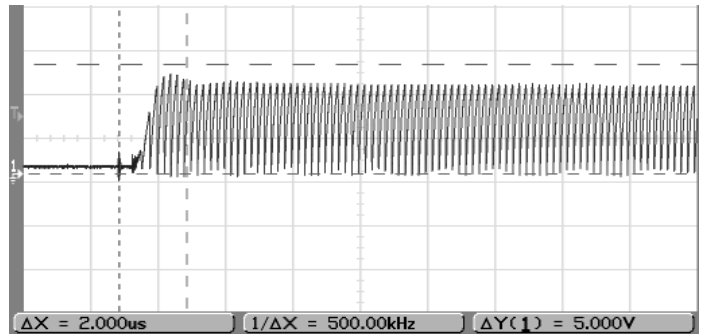
## 4.4.2 外部高速时钟

外部高速时钟共三种模式：石英/陶瓷振荡器，RC 及外部时钟源，由编译选项 High\_Clk 控制具体模式的选择。石英/陶瓷振荡器和 RC 振荡器的上升时间各不相同。RC 振荡器的上升时间相对较短。振荡器上升时间与复位时间的长短密切相关。

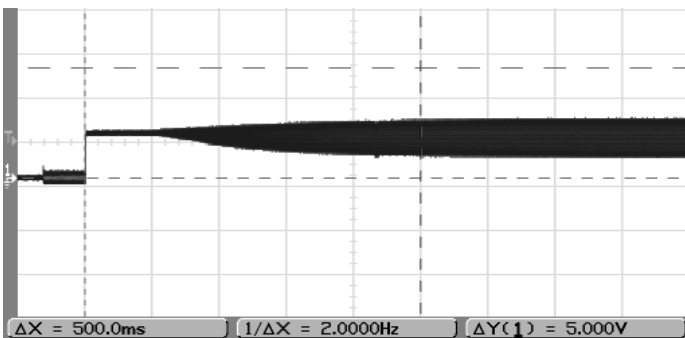
4MHz Crystal



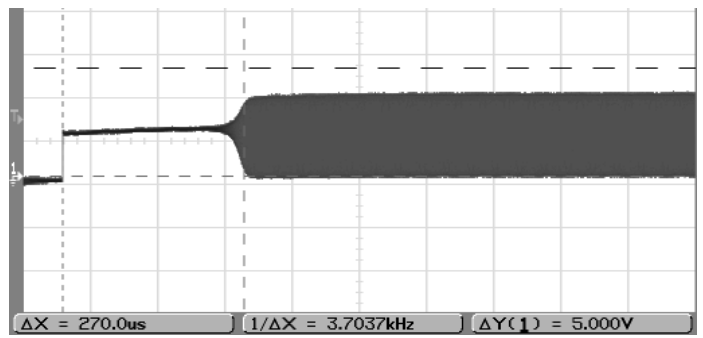
RC



32768Hz Crystal

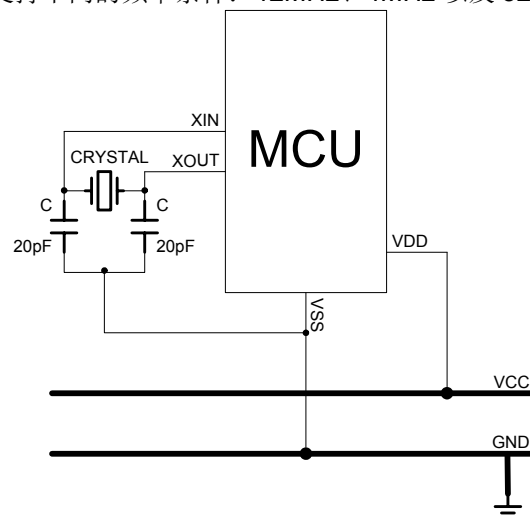


4MHz Ceramic



### 4.4.2.1 石英/陶瓷振荡器

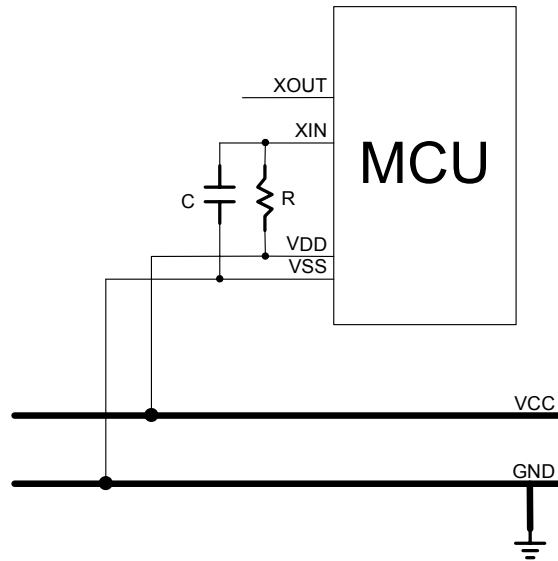
石英/陶瓷振荡器由 XIN/XOUT 口驱动，对于高速、普通和低速三种不同工作模式，振荡器的驱动电流也不同。不同的工作模式下，编译选项 High\_Clk 支持不同的频率条件：12MHz、4MHz 以及 32KHz 工作频率。



\* 注：上图中，XIN/XOUT/VSS 引脚与石英/陶瓷振荡器以及电容 C 之间的距离越近越好。

#### 4.4.2.2 RC 振荡器

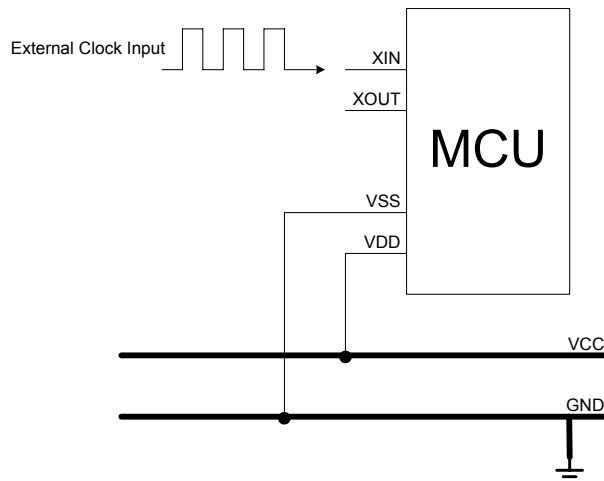
通过编译选项 High\_Clk 的设置可控制 RC 振荡器的选择，RC 振荡器输出频率最高可达 10MHZ。改变 R 可改变输出频率的大小，电容 C 的最佳容量为 50P~100P，引脚 XOUT 为通用 I/O 口，如下图所示：



\* 注：电容 C 和电阻 R 应尽可能的接近单片机的 VDD。

#### 4.4.2.3 外部时钟源

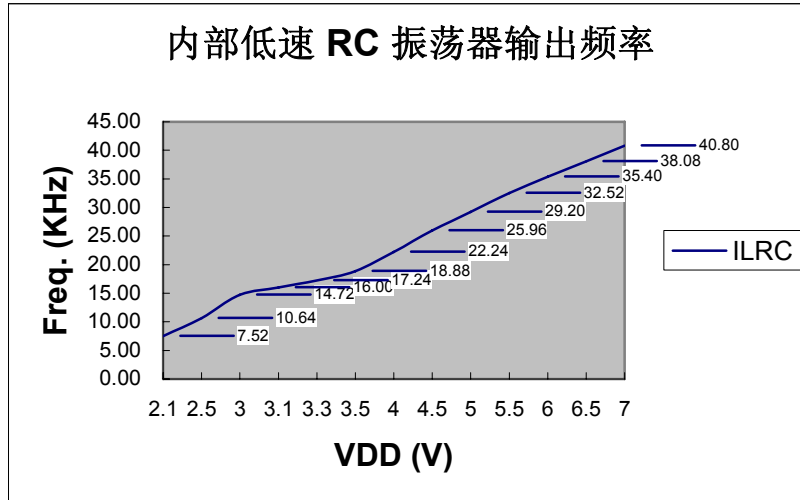
单片机可选择外部时钟信号作为系统时钟，由编译选项 High\_Clk 控制，从 XIN 脚送入。



\* 注：外部振荡电路中的 GND 必须尽可能的接近单片机的 VSS 端口。

## 4.5 系统低速时钟

系统低速时钟源即内置的低速振荡器，采用 RC 振荡电路。低速时钟的输出频率受系统电压和环境温度的影响，通常为 5V 时输出 32KHZ，3V 时输出 16KHZ。输出频率与工作电压之间的关系如下图所示。



低速时钟可作为看门狗定时器的时钟源。由 CLKMD 控制系统低速工作模式。

- ☞ **Fosc = 内部低速 RC 振荡器 (16KHz @3V、32KHz @5V)。**
- ☞ **低速模式 Fcpu = Fosc / 4。**

系统工作在睡眠模式下，可以停止低速 RC 振荡器。

- **例：停止内部低速振荡器。**  
B0BSET      FCPUM0

\* 注：不可以单独停止内部低速时钟；由寄存器 OSCM 的位 CPUM0 和 CPUM1 的设置决定内部低速时钟的状态。

### 4.5.1 系统时钟测试

在设计过程中，用户可通过软件指令周期 Fcpu 对系统时钟速度进行测试。

- **例：外部振荡器的 Fcpu 指令周期测试。**  
B0BSET      P0M.0                      ; P0.0 置为输出模式以输出 Fcpu 的触发信号。
- @@:  
B0BSET      P0.0  
B0BCLR      P0.0  
JMP            @B

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

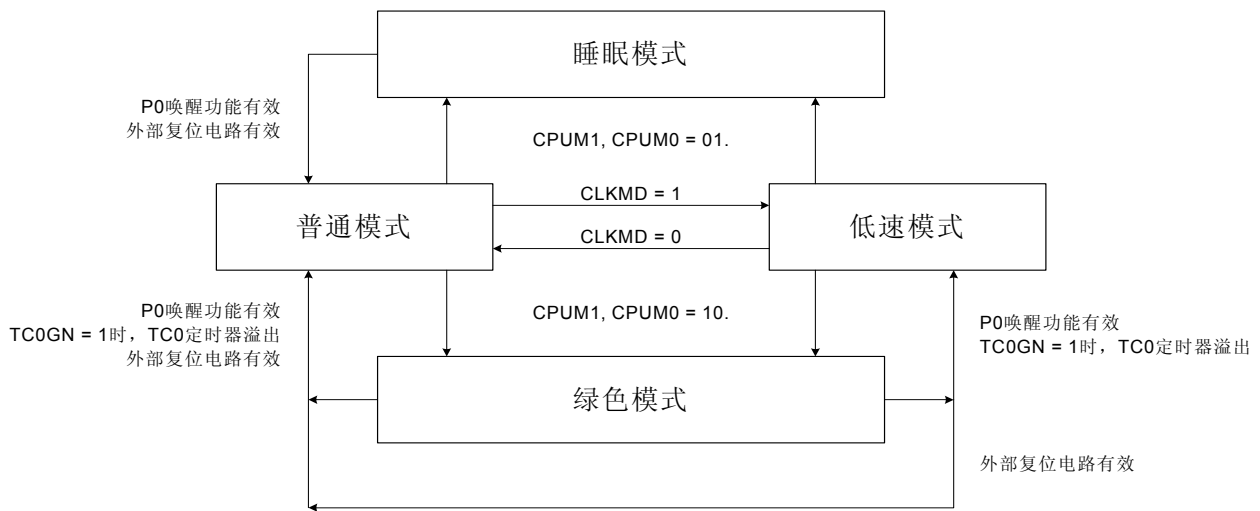


# 5 系统工作模式

## 5.1 概述

SN8P2711A 可在如下四种工作模式之间进行切换：

- 普通模式 (高速模式)；
- 低速模式；
- 睡眠模式；
- 绿色模式。



系统模式转换结构图

### 工作模式说明

工作模式	普通模式	低速模式	绿色模式	睡眠模式	备注
EHOSC	运行	STPHX 控制	STPHX 控制	停止	
IHRC	运行	STPHX 控制	STPHX 控制	停止	
ILRC	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	*TC0ENB = 1 时有效
TC1	*有效	*有效	*有效	无效	*TC1ENB = 1 时有效
看门狗定时器	Watch_Dog 编译选项控制	Watch_Dog 编译选项控制	Watch_Dog 编译选项控制	Watch_Dog 编译选项控制	参考 CODE OPTION 说明
内部中断	全部有效	全部有效	TC1	全部无效	
外部中断	全部有效	全部有效	全部有效	全部有效	
唤醒功能	-	-	P0, TC0, 复位	P0, 复位	

**EHOSC:** 外部高速时钟。

**IHRC:** 内部高速时钟 (16M RC 振荡器)。

**ILRC:** 内部低速时钟 (3V 时 16K RC 振荡器, 5V 时 32K 振荡器)。

## 5.2 系统模式切换

➤ 例：系统由普通/低速模式切换进入睡眠模式。  
 B0BSET FCPUM0 ;

\* 注：睡眠模式下，仅具有唤醒功能的引脚和复位引脚能够将系统唤醒回到普通模式。

➤ 例：系统由普通模式切换进入低速模式。  
 B0BSET FCLKMD ; CLKMD = 1, 系统进入低速模式。  
 B0BSET FSTPHX ; 停止外部高速振荡器。

➤ 例：系统由低速模式切换进入普通模式（外部高速振荡器仍然运行）。  
 B0BCLR FCLKMD ;

➤ 例：系统由低速模式切换进入普通模式（外部高速振荡器停止运行）。  
 在外部高速振荡器停止工作的情况下，系统至少需要延迟 20ms 才能回到普通运行模式。  
 B0BCLR FSTPHX ; 启动外部高速振荡器。

```

MOV A, #54 ; 若 VDD = 5V, 内部 RC=32KHz 将延迟 0.125ms×162= 20.25ms
B0MOV Z, A ; 等待外部时钟稳定。
@@: DECMS Z ;
    JMP @B ;
    
```

B0BCLR FCLKMD ; 进入普通模式。

➤ 例：系统由普通/低速模式切换进入绿色模式。  
 B0BSET FCPUM1 ;

\* 注：绿色模式下如果禁止 TC0 的唤醒功能，则只有具有唤醒功能的引脚和复位引脚可以将系统唤醒（具有唤醒功能的引脚将系统返回到上一个工作模式，复位引脚将系统返回到普通模式）。

➤ 例：系统由普通/低速模式切换进入绿色模式，开启 TC0 的唤醒功能。  
 ; 使能 TC0 的唤醒功能。

```

B0BCLR FTC0IEN ; 禁止 TC0 中断。
B0BCLR FTC0ENB ; 禁止 TC0 定时器。
MOV A, #20H ;
B0MOV TC0M, A ; 设置 TC0 时钟 = Fcpu / 64。
MOV A, #64H ;
B0MOV TC0C, A ; 设置 TC0C = 64H (TC0 间隔时间 = 10 ms)。
B0BCLR FTC0IEN ; 禁止 TC0 中断。
B0BCLR FTC0IRQ ; 清 TC0 中断请求标志。
B0BSET FTC0GN ; 开启 TC0 的唤醒功能。
B0BSET FTC0ENB ; 开启 TC0 定时器。
    
```

; 进入绿色模式。

```

B0BCLR FCPUM0 ;
B0BSET FCPUM1 ;
    
```

\* 注：TC0 和具有唤醒功能的引脚都可以将系统从绿色模式唤醒返回到上一个工作模式，TC0 的唤醒功能由程序控制，TC0GN 必须置 1。

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>TOM</b>	-	-	-	-	TC1X8	TC0X8	TC0GN	-
读/写	-	-	-	-	R/W	R/W	R/W	-
复位后	-	-	-	-	0	0	0	-

Bit 1 **TC0GN**: TC0 绿色模式下唤醒功能控制位。  
 0 = 禁止;  
 1 = 使能。

## 5.3 唤醒时间

### 5.3.1 概述

在绿色模式和睡眠模式下，系统并不执行程序指令，唤醒触发信号能够将处于睡眠状态的系统唤醒到普通模式或低速模式。这里的唤醒触发信号包括外部触发信号（P0 引脚的电平变化）和内部触发信号（TC0 溢出信号）。

- 从睡眠模式唤醒后只能进入普通模式，且将其唤醒的触发只能是外部触发信号；
- 由绿色模式唤醒回到系统前一工作模式（普通模式或低速模式）可以用外部触发或者内部触发。

### 5.3.2 唤醒时间

系统进入睡眠模式后，高速时钟是处于停止状态的。把系统从睡眠模式下唤醒时，单片机需要等待 2048 个外部高速振荡器时钟周期以使振荡电路进入稳定工作状态，等待的这一段就称为唤醒时间。唤醒时间结束后，系统才进入到普通模式。

\* 注：将系统从绿色模式中唤醒是不需要唤醒时间的，因为在绿色模式下高速时钟仍然正常工作。

唤醒时间的计算如下：

$$\text{唤醒时间} = 1/F_{osc} * 2048 \text{ (sec)} + \text{高速时钟启动时间}$$

\* 注：高速时钟的启动时间与 VDD 和振荡器类型有关。

➤ 例：将系统从睡眠模式中唤醒，并设置系统进入普通模式。唤醒时间计算如下。

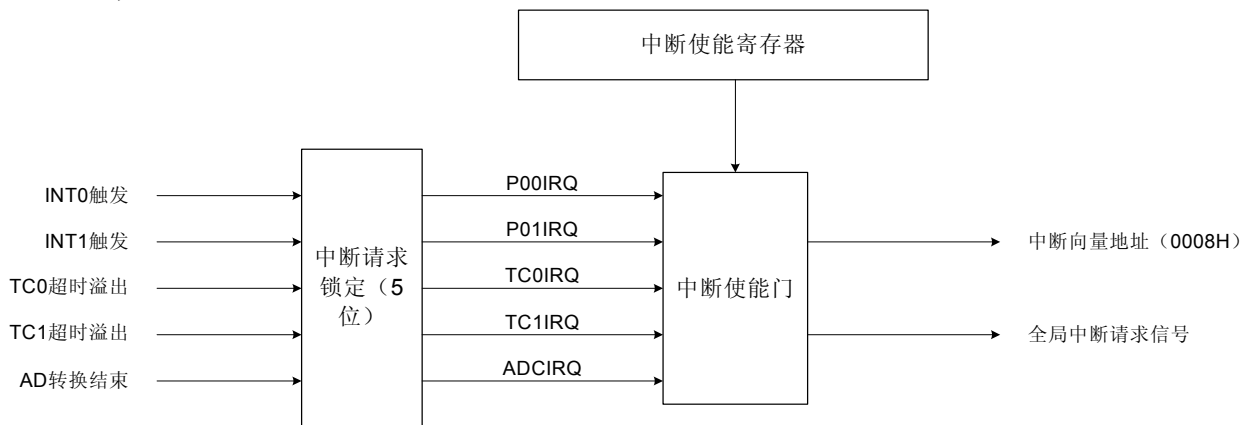
$$\text{唤醒时间} = 1/F_{osc} * 2048 = 0.512 \text{ ms (} F_{osc} = 4\text{MHz)}$$

$$\text{总的唤醒时间} = 0.512 \text{ ms} + \text{振荡器启动时间}$$

# 6 中断

## 6.1 概述

SN8P2711A 提供 5 个中断源：3 个内部中断（TC0/TC1/ADC）和 2 个外部中断（INT0/INT1）。外部中断可以将系统从睡眠模式中唤醒进入高速模式，在返回到高速模式前，中断请求被锁定。一旦程序进入中断，寄存器 STKP 的位 GIE 被硬件自动清零以避免响应其它中断。系统退出中断后，硬件自动将 GIE 置“1”，以响应下一个中断。中断请求存放在寄存器 INTRQ 中。



\* 注：程序响应中断时，必须开启全局中断控制位 GIE。

## 6.2 中断请求使能寄存器 INTEN

中断请求控制寄存器 INTEN 包括所有中断的使能控制位。INTEN 的有效位被置为“1”则系统进入该中断服务程序，程序计数器入栈，程序转至 0008H 即中断程序。程序运行到指令 RETI 时，中断结束，系统退出中断服务。

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>INTEN</b>	ADCIEN	TC1IEN	TC0IEN	-	-	-	P01IEN	P00IEN
读/写	R/W	R/W	R/W	-	-	-	R/W	R/W
复位后	0	0	0	-	-	-	0	0

Bit 0 **P00IEN**: P0.0 外部中断 (INT0) 控制位。

0 = 无效;  
1 = 有效。

Bit 1 **P01IEN**: P0.1 外部中断 (INT1) 控制位。

0 = 无效;  
1 = 有效。

Bit 5 **TC0IEN**: TC0 中断控制位。

0 = 无效;  
1 = 有效。

Bit 6 **TC1IEN**: TC1 中断控制位。

0 = 无效;  
1 = 有效。

Bit 7 **ADCIEN**: ADC 中断控制位。

0 = 无效;  
1 = 有效。

## 6.3 中断请求寄存器 INTRQ

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

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>INTRQ</b>	ADCIrq	TC1IRQ	TC0IRQ	-	-	-	P01IRQ	P00IRQ
读/写	R/W	R/W	R/W	-	-	-	R/W	R/W
复位后	0	0	0	-	-	-	0	0

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

Bit 1 **P01IRQ**: P0.1 中断 (INT1) 请求标志。  
0 = INT1 无中断请求;  
1 = INT1 有中断请求。

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

Bit 6 **TC1IRQ**: TC1 中断请求标志。  
0 = TC1 无中断请求;  
1 = TC1 有中断请求。

Bit 7 **TC0IRQ**: ADC 中断请求标志。  
0 = ADC 无中断请求;  
1 = ADC 有中断请求。

## 6.4 GIE 全局中断

只有当全局中断控制位 GIE 置“1”的时候程序才能响应中断请求。一旦有中断发生，程序计数器 (PC) 指向中断向量地址 (ORG8)，堆栈层数加 1。

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

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

➤ 例: 设置全局中断控制位 (GIE)。  
BOBSET FGIE ; 使能 GIE。

\* 注: 在所有中断中, GIE 都必须处于使能状态。

## 6.5 PUSH, POP 处理

有中断请求发生并被响应后，程序转至 0008H 执行中断子程序。响应中断之前，必须保存 ACC、PFLAG 的内容。芯片提供 PUSH 和 POP 指令进行入栈保存和出栈恢复，从而避免中断结束后可能的程序运行错误。

\* 注：“PUSH”、“POP”指令仅对 ACC 和 PFLAG 作中断保护，而不包括 NT0 和 NPD。PUSH/POP 缓存器是唯一的且仅有一层。

➤ 例：对 ACC 和 PAFLG 进行入栈保护。

```
                ORG      0
                JMP      START

                ORG      8H
                JMP      INT_SERVICE

START:          ORG      10H
                ...

INT_SERVICE:   PUSH                    ; 保存 ACC 和 PFLAG。
                ...
                POP                    ; 恢复 ACC 和 PFLAG。

                RETI                   ; 退出中断。
                ...
                ENDP
```

## 6.6 INTO (P0.0) 中断

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

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

\* 注：INT0 的中断请求被 P0.0 的唤醒触发功能锁定。

\* 注：P0.0 的中断触发边沿由 PEDGE 控制。

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

Bit[4:3] **P00G[1:0]**: P0.0 中断触发控制位。  
 00 = 保留;  
 01 = 上升沿触发;  
 10 = 下降沿触发;  
 11 = 上升/下降沿触发 (电平触发)。

➤ 例：INT0 中断请求设置，电平触发。

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

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

➤ 例：INT0 中断。

```
ORG      8H
JMP      INT_SERVICE ;

INT_SERVICE:

...      ; 保存 ACC 和 PFLAG。

B0BTS1   FP00IRQ      ; 检查是否有 P00 中断请求标志。
JMP      EXIT_INT     ; P00IRQ = 0, 退出中断。

B0BCLR   FP00IRQ      ; 清 P00IRQ。
...      ; INT0 中断服务程序。
...

EXIT_INT:

...      ; 恢复 ACC 和 PFLAG。

RETI     ; 退出中断。
```

## 6.7 INT1 (P0.1) 中断

INT1 被触发，则无论 P01IEN 处于何种状态，P01IRQ 都会被置“1”。如果 P01IRQ = 1 且 P01IEN = 1，系统响应该中断；如果 P01IRQ = 1 而 P01IEN = 0，系统并不会执行中断服务。在处理多中断时尤其需要注意。

如果中断的触发方向和唤醒功能的触发方向是一样的，则在系统由 P0.1 从睡眠模式和绿色模式唤醒时，INT1 的中断请求 (INT1IRQ) 就会被锁定。系统会在唤醒后马上进入中断向量地址执行中断服务程序。

\* 注：INT1 的中断请求被 P0.1 的唤醒触发功能锁定。

\* 注：P0.1 中断由下降沿触发

### 例：INT1 中断请求设置。

```
B0BCLR      FP01IRQ      ; 清 INT1 中断请求标志。
B0BSET      FP01IEN     ; 使能 INT1 中断。
B0BSET      FGIE        ; 使能 GIE。
```

### 例：INT1 中断。

```
ORG      8H      ;
INT_SERVICE:
    JMP      INT_SERVICE

    ...          ; 保存 ACC 和 PFLAG。

B0BTS1    FP01IRQ      ; 检查是否有 P01 中断请求标志。
JMP      EXIT_INT     ; P01IRQ = 0，退出中断。

B0BCLR    FP01IRQ      ; 清 P01IRQ。
    ...          ; INT1 中断服务程序。
EXIT_INT:
    ...          ; 恢复 ACC 和 PFLAG。

    RETI         ; 退出中断。
```



## 6.8 TC0 中断

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

### ➤ 例：TC0 中断请求设置。

```

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

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

B0BSET    FGIE           ; 使能 GIE。

```

### ➤ 例：TC0 中断服务程序。

```

ORG       8H             ;
JMP       INT_SERVICE

INT_SERVICE:
...           ; 保存 ACC 和 PFLAG。

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

B0BCLR    FTC0IRQ        ; 清 TC0IRQ。
MOV       A, #64H       ; 清 TC0C。
B0MOV     TC0C, A        ; TC0 中断程序。
...
...

EXIT_INT:
...           ; 恢复 ACC 和 PFLAG。

RETI      ; 退出中断。

```

## 6.9 TC1 中断

TC1C 溢出时，无论 TC1IEN 处于何种状态，TC1IRQ 都会置“1”。若 TC1IEN 和 TC1IRQ 都置“1”，系统就会响应 TC1 的中断；若 TC1IEN = 0，则无论 TC1IRQ 是否置“1”，系统都不会响应 TC1 中断。尤其需要注意多种中断下的情形。

### ➤ 例：设置 TC1 中断请求。

```

B0BCLR    FTC1IEN    ; 禁止 TC1 中断。
B0BCLR    FTC1ENB    ; 关闭 TC1 定时器。
MOV       A, # 20H    ;
B0MOV     TC1M, A     ; 设置 TC1 时钟=Fcpu / 64。
MOV       A, # 64H    ; 设置 TC1C 初始值=64H。
B0MOV     TC1C, A     ; 设置 TC1 间隔时间=10 ms。

B0BCLR    FTC1IRQ    ; 清 TC1 中断请求标志。
B0BSET    FTC1IEN    ; 使能 TC1 中断。
B0BSET    FTC1ENB    ; 开启 TC1 定时器。

B0BSET    FGIE       ; 使能 GIE。

```

### ➤ 例：TC1 中断服务程序。

```

ORG       8H          ;
INT_SERVICE:
JMP       INT_SERVICE

...           ; 保存 ACC 和 PFLAG。

B0BTS1    FTC1IRQ    ; 检查是否有 TC1 中断请求标志。
JMP       EXIT_INT   ; TC1IRQ = 0，退出中断。

B0BCLR    FTC1IRQ    ; 清 TC1IRQ。
MOV       A, #64H    ;
B0MOV     TC1C, A     ; 清 TC1C。
...        ; TC1 中断服务程序。
...

EXIT_INT:
...           ; 恢复 ACC 和 PFLAG。

RETI      ; 退出中断。

```

## 6.10 ADC 中断

当 ADC 转换完成后，无论 ADCIEN 是否使能，ADCIQR 都会置“1”。若 ADCIEN 和 ADCIQR 都置“1”，那么系统就会响应 ADC 中断。若 ADCIEN = 0，不管 ADCIRQ 是否置“1”，系统都不会进入 ADC 中断。用户应注意多种中断下的处理。

### ➤ 例：ADC 中断设置。

```

B0BCLR          FADCIEN          ; 禁止 ADC 中断。

MOV             A, #10110000B    ;
B0MOV          ADM, A           ; 允许 P4.0 ADC 输入，使能 ADC 功能。
MOV            A, #00000000B    ; 设置 AD 转换速率 = Fcpu/16。
B0MOV

B0BCLR          FADCIRQ         ; 清除 ADC 中断请求标志。
B0BSET          FADCIEN         ; 使能 ADC 中断。
B0BSET          FGIE            ; 使能 GIE。

B0BSET          FADS            ; 开始 AD 转换。

```

### ➤ 例：ADC 中断服务程序。

```

ORG             8H              ; 中断向量地址。
JMP             INT_SERVICE

INT_SERVICE:

...              ; 保存 ACC 和 PFLAG。

B0BTS1          FADCIRQ         ; 检查是否有 ADC 中断。
JMP             EXIT_INT        ; ADCIRQ = 0，退出中断。

B0BCLR          FADCIRQ         ; 清 ADCIRQ。
...              ; ADC 中断服务程序。
...

EXIT_INT:

...              ; 恢复 ACC 和 PFLAG。

RETI            ; 退出中断。

```

## 6.11 多中断操作举例

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

中断	有效触发
P00IRQ	由 PEDGE 控制
P01IRQ	下降沿触发
TC0IRQ	TC0C 溢出
TC1IRQ	TC1C 溢出
ADCIRQ	AD 转换完成

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

➤ 例：多中断条件下检测中断请求。

```

        ORG          8H          ;
        JMP          INT_SERVICE
INT_SERVICE:
        ...                    ; 保存 ACC 和 PFLAG。

INTP00CHK:                    ; 检查是否有 P00 中断请求。
        B0BTS1      FP00IEN    ; 检查是否使能 P00 中断。
        JMP         INTP01CHK  ; 跳到下一个中断。
        B0BTS0      FP00IRQ    ; 检查是否有 P00 中断请求。
        JMP         INTP00     ; 进入 INT0 中断。

INTP01CHK:                    ; 检查是否有 P01 中断请求。
        B0BTS1      FP01IEN    ; 检查是否使能 P01 中断。
        JMP         INTTC0CHK  ; 跳到下一个中断。
        B0BTS0      FP01IRQ    ; 检查是否有 P01 中断请求。
        JMP         INTP01     ; 进入 INT1 中断。

INTTC0CHK:                    ; 检查是否有 TC0 中断请求。
        B0BTS1      FTC0IEN    ; 检查是否使能 TC0 中断。
        JMP         INTTC1CHK  ; 跳到下一个中断。
        B0BTS0      FTC0IRQ    ; 检查是否有 TC0 中断请求。
        JMP         INTTC0     ; 进入 TC0 中断。

INTTC1CHK:                    ; 检查是否有 TC1 中断请求。
        B0BTS1      FTC1IEN    ; 检查是否使能 TC1 中断。
        JMP         INTADCHK   ; 跳到下一个中断。
        B0BTS0      FTC1IRQ    ; 检查是否有 TC1 中断请求。
        JMP         INTTC1     ; 进入 TC1 中断。

INTADCHK:                    ; 检查是否有 ADC 中断请求。
        B0BTS1      FADCIEN    ; 检查是否使能 ADC 中断。
        JMP         INT_EXIT   ;
        B0BTS0      FADCIRQ    ; 检查是否有 ADC 中断请求。
        JMP         INTADC     ; 进入 ADC 中断。

INT_EXIT:
        ...                    ; 恢复 ACC 和 PFLAG。

        RETI                    ; 退出中断。

```

# 7 I/O 口

## 7.1 I/O 口模式

寄存器 PnM 控制 I/O 口的工作模式。

0B8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P0M</b>	-	-	-	-	P03M	P02M	P01M	P00M
读/写	-	-	-	-	R/W	R/W	R/W	R/W
复位后	-	-	-	-	0	0	0	0

0C4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P4M</b>	-	-	-	P44M	P43M	P42M	P42M	P40M
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

0C5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P5M</b>	-	-	-	P54M	P53M	-	-	-
读/写	-	-	-	R/W	R/W	-	-	-
复位后	-	-	-	0	0	-	-	-

Bit[7:0] **PnM[7:0]**: Pn 模式控制位 (n = 0~5)。

0 = 输入模式;

1 = 输出模式。

\* 注:

1. 用户可以用位操作指令 (B0BSET, B0BCLR) 对 I/O 口进行操作;
2. P0.4 是单向输入引脚, P0.4M = 1。

➤ 例: I/O 模式选择。

```
CLR      P0M          ; 所有端口置为输入模式。
CLR      P4M
CLR      P5M

MOV      A, #0FFH    ; 所有端口置为输出模式。
B0MOV   P0M, A
B0MOV   P4M, A
B0MOV   P5M, A

B0BCLR  P4M.0        ; P4.0 置为输入模式。

B0BSET  P4M.0        ; P4.0 置为输出模式。
```

## 7.2 I/O 上拉电阻寄存器

0E0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P0UR</b>	-	-	-	-	P03R	P02R	P01R	P00R
读/写	-	-	-	-	W	W	W	W
复位后	-	-	-	-	0	0	0	0

0E4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P4UR</b>	-	-	-	P44R	P43R	P42R	P41R	P40R
读/写	-	-	-	W	W	W	W	W
复位后	-	-	-	0	0	0	0	0

0E5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P5UR</b>	-	-	-	P54R	P53R	-	-	-
读/写	-	-	-	W	W	-	-	-
复位后	-	-	-	0	0	-	-	-

\* 注：P0.4 是单向输入引脚，无上拉电阻，因此 P0UR.4 始终为“1”。

➤ 例：I/O 上拉电阻。

```
MOV      A, #0FFH      ; 使能 P0、4、5 上拉电阻。
B0MOV   P0UR, A
B0MOV   P4UR, A
B0MOV   P5UR, A
```

## 7.3 I/O 口数据寄存器

0D0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P0</b>	-	-	-	P04	P03	P02	P01	P00
读/写	-	-	-	R	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

0D4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P4</b>	-	-	-	P44	P43	P42	P41	P40
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

0D5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P5</b>	-	-	-	P54	P53	-	-	-
读/写	-	-	-	R/W	R/W	-	-	-
复位后	-	-	-	0	0	-	-	-

\* 注：使能外部复位时，P04 的值保持为“1”。

➤ 例：从输入端口读取数据。

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

➤ 例：写数据到输出端口。

```
MOV      A, #0FFH   ; 写 0FFH 到所有的端口。
B0MOV   P0, A
B0MOV   P4, A
B0MOV   P5, A
```

➤ 例：写 1 位数据到输出端口。

```
B0BSET  P4.0      ; P4.0 和 P5.3 设为“1”。
B0BSET  P5.3

B0BCLR  P4.0      ; P4.0 和 P5.3 为设“0”。
B0BCLR  P5.3
```

## 7.4 P4 口 ADC 共用引脚

P4 口和 ADC 的输入口共用，非施密特触发。同一时间只能设置 P4 口的一个引脚作为 ADC 的测量信号输入口（通过 ADM 寄存器来设置），其它引脚则作为普通 I/O 使用。具体应用中，当输入一个模拟信号到 CMOS 结构端口，尤其当模拟信号为  $1/2 V_{DD}$  时，将可能产生额外的漏电流。同样，当 P4 口外接多个模拟信号时，也会产生额外的漏电流。在睡眠模式下，上述漏电流会严重影响到系统的整体功耗。P4CON 为 P4 口的配置寄存器。将 P4CON[7:0]置“1”，其对应的 P4 口将被设置为纯模拟信号输入口，从而避免上述漏电流的情况。

0AEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P4CON</b>	-	-	-	P4CON4	P4CON3	P4CON2	P4CON1	P4CON0
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

Bit[4:0] **P4CON[4:0]**: P4.n 控制位。  
 0 = P4.n 作为模拟信号输入或普通 I/O 引脚；  
 1 = P4.n 作为仅作模拟信号输入引脚。

\* 注：当 P4.n 作为普通 I/O 口而不是 ADC 输入引脚时，P4CON.n 必须置为 0，否则 P4.n 的普通 I/O 信号会被隔离开来。

P4 的 ADC 模拟输入由寄存器 ADM 的 GCHS 和 CHSn 位控制，若 GCHS = 0，P4.n 为普通的 I/O 引脚，若 GCHS = 1，CHSn 所对应的 P4.n 用作 ADC 模拟信号输入引脚。

0B1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>ADM</b>	ADENB	ADS	EOC	GCHS	-	CHS2	CHS1	CHS0
读/写	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W
复位后	0	0	0	0	-	0	0	0

Bit 4 **GCHS**: ADC 输入通道控制位。  
 0 = 禁止 AIN 通道；  
 1 = 开启 AIN 通道。

Bit[2:0] **CHS[2:0]**: ADC 输入通道选择位。  
 000 = AIN0; 001 = AIN1; 010 = AIN2; 011 = AIN3; 100 = AIN4; 101 = AIN5。

\* 注：在设置 P4.n 为普通的 I/O 引脚时，必须保证 P4.n 的 ADC 功能已经被禁止。否则当 GCHS = 1 时，CHS[2:0]所指向的 P4.n 会被自动设为 ADC 输入引脚。

➤ 例：设置 P4.1 为普通的输入引脚，P4CON.1 必须置为 0。

；检查 GCHS 和 CHS[2:0]的状态。

```
B0BCLR          FGCHS          ; CHS[2:0]指向 P4.1 (CHS[2:0] = 001B)，则 GCHS=0。
                  ; CHS[2:0]没指向 P4.1 (CHS[2:0] ≠ 001B)，则忽略 GCHS 的状态。
```

；清 P4CON。

```
B0BCLR          P4CON.1        ; 使能 P4.1 的普通 I/O 功能。
```

；P4.1 设为输入模式。

```
B0BCLR          P4M.1          ; 设置 P4.1 为输入模式。
```

➤ 例：设置 P4.1 为普通的输出模式，P4CON.1 必须置为 0。

；检查 GCHS 和 CHS[2:0]的状态。

```
B0BCLR          FGCHS          ; CHS[2:0]指向 P4.1 (CHS[2:0]=001B)，则 GCHS=0。
                  ; CHS[2:0]不指向 P4.1 (CHS[2:0]≠001B)，则忽略 GCHS 的状态。
```

；清 P4CON。

```
B0BCLR          P4CON.1        ; 使能 P4.1 的普通 I/O 功能。
```

；设置 P4.1 为输出模式以避免误操作。

```
B0BSET          P4.1           ; 设置 P4.1 为 1。
```

或

```
B0BCLR          P4.1           ; 设置 P4.1 为 0。
```

；P4.1 设为输出模式。

```
B0BSET          P4M.1          ; P4.1 设为输入模式。
```

P4.0 可作为普通的 I/O 引脚, ADC 输入 (AIN0) 和 ADC 外部参考电压的高电平输入端。VERFH 寄存器的 EVHENB 位是 ADC 的外部参考电压的高电平输入控制位。若使能 EVHENB, P4.0 的普通 I/O 功能和 ADC 输入 (AIN0) 功能被禁止。P4.0 和 ADC 的参考电压输入端直接相连。

\* 注: 若想使能 P4.0 的普通 I/O 功能和 AIN0 功能, 必须将 EVHENB 设置为“0”。

0AFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VREFH	EVHENB	-	-	-	-	-	VHS1	VHS0
读/写	R/W	-	-	-	-	-	R/W	R/W
复位后	0	-	-	-	-	-	0	0

Bit 7 **EVHENB**: ADC 外部参考电压的高电平输入控制位。

0 = 禁止 ADC 参考电压的高电平输入;

1 = 允许 ADC 参考电压的高电平输入。

➤ 例: 设置 P4.0 为普通输入模式, **EVHENB** 和 **P4CON.0** 必须置 0。

; 检查 EVHENB 状态。

```
B0BTS0      FEVHENB      ;
B0BCLR      FEVHENB      ; EVHENB = 1, 禁止 ADC 外部参考电压的高电平输入。
; EVHENB = 0, 执行下一段程序。
```

; 检查 GCHS 和 CHS[2:0] 的状态。

```
B0BCLR      FGCHS      ; CHS[2:0] 指向 P4.0 (CHS[2:0] = 000B), 则 GCHS=0。
; CHS[2:0] 不指向 P4.0 (CHS[2:0] ≠ 000B), 则忽略 GCHS 的状态。
```

; 清 P4CON。

```
B0BCLR      P4CON.0    ; 使能 P4.0 的普通 I/O 功能。
```

; 设置 P4.0 为输入模式。

```
B0BCLR      P4M.0      ; P4.0 置为输入模式。
```

➤ 例: 设置 P4.0 为普通输出模式, **EVHENB** 和 **P4CON.0** 位必须置 0。

; 检查 EVHENB 的状态。

```
B0BTS0      FEVHENB      ;
B0BCLR      FEVHENB      ; EVHENB=1, 清 EVHENB 并禁止 ADC 外部参考电压的高电平输入。
; EVHENB = 0, 执行下一段程序。
```

; 检查 GCHS 和 CHS[2:0] 的状态。

```
B0BCLR      FGCHS      ; CHS[2:0] 指向 P4.0 (CHS[2:0] = 000B), 设置 GCHS=0。
; CHS[2:0] 不指向 P4.0 (CHS[2:0] ≠ 000B), 则忽略 GCHS 的状态。
```

; 清 P4CON。

```
B0BCLR      P4CON.0    ; 使能 P4.0 的普通 I/O 功能。
```

; 设置 P4.0 为输出模式。

```
B0BSET      P4.0      ; 设置 P4.0 为 1。
```

; 或

```
B0BCLR      P4.0      ; 设置 P4.0 为 0。
```

; 设置 P4.0 为输出模式。

```
B0BSET      P4M.0      ;
```



# 8 定时器

## 8.1 看门狗定时器

看门狗定时器 WDT 是一个 4 位二进制计数器，用于监控程序的正常执行。如果由于干扰，程序进入了未知状态，看门狗定时器溢出，系统复位。看门狗的工作模式由编译选项控制，其时钟源由内部低速 RC 振荡器（16KHz @3V, 32KHz @5V）提供。

看门狗溢出时间 = 8192 / 内部低速振荡器周期 (sec)

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

\* 注：如果看门狗被置为“Always\_On”模式，那么看门狗在睡眠模式和绿色模式下仍然运行。

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

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

➤ 例：如下是对看门狗定时器的操作，在主程序开头对看门狗清零。

```

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

```

看门狗定时器应用注意事项如下：

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

➤ 例：如下是对看门狗定时器的操作，在主程序开头对看门狗清零。

```

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

Correct:
...          ; I/O 和 RAM 正常，看门狗清零。
...
MOV         A, 5AH
B0MOV      WDTR, A
...
CALL       SUB1
CALL       SUB2
...
...
JMP        MAIN

```

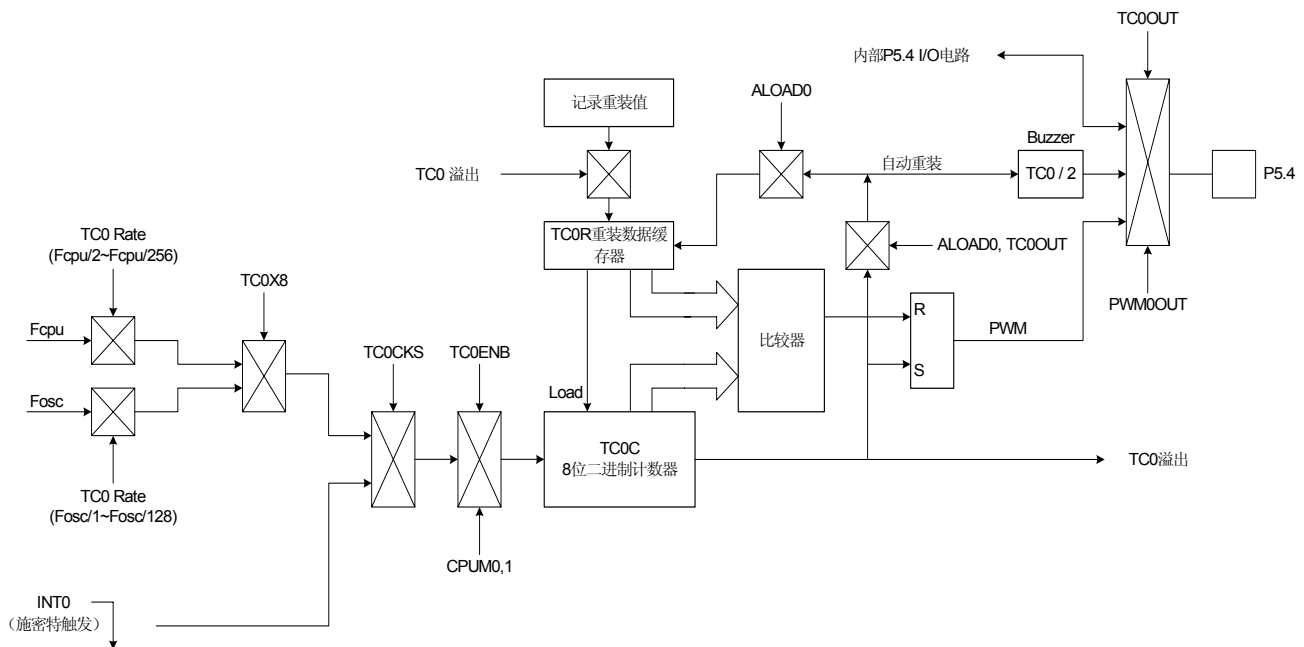
## 8.2 定时/计数器 TC0

### 8.2.1 概述

定时/计数器 TC0 具有双时钟源,可根据实际需要选择内部时钟或外部时钟作为计时标准。其中,内部时钟源来自  $F_{cpu}$  或  $F_{osc}$  ( $F_{osc}$  由 TC0X8 标志控制)。外部时钟源 INT0 从 P0.0 端输入(下降沿触发)。寄存器 TC0M 控制 TC0 时钟源的选择。当 TC0 从 0FFH 溢出到 00H 时,TC0 在继续计数的同时产生一个溢出信号,触发 TC0 中断请求。在 PWM 模式,TC0 的溢出时间由寄存器 ALOAD0 和 TC0OUT 位控制。

TC0 的主要功能如下:

- ☞ **8 位可编程定时器:** 根据选择的时钟频率信号,产生周期中断;
- ☞ **外部事件计数器:** 对外部事件计数;
- ☞ **绿色模式唤醒功能:** TC0 可以将系统从绿色模式下唤醒;
- ☞ **Buzzer 输出;**
- ☞ **PWM 输出。**



## 8.2.2 TC0M 模式寄存器

0DAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>TC0M</b>	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS	ALOAD0	TC0OUT	PWM0OUT
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

- Bit 0 **PWM0OUT**: PWM 输出控制。  
0 = 禁止 PWM 输出;  
1 = 使能 PWM 输出, PWM 输出占空比由 T0OUT 和 ALOAD0 控制。
- Bit 1 **TC0OUT**: TC0 溢出信号输出控制位。仅当 **PWM0OUT = 0** 时有效。  
0 = 禁止, P5.4 作为输入/输出口;  
1 = 允许, P5.4 输出 TC0OUT 信号。
- Bit 2 **ALOAD0**: 自动装载控制位。仅当 **PWM0OUT = 0** 时有效。  
0 = 禁止 TC0 自动重装;  
1 = 允许 TC0 自动重装。
- Bit 3 **TC0CKS**: TC0 时钟信号控制位。  
0 = 内部时钟 (Fcpu 或 Fosc);  
1 = 外部时钟, 由 P0.0/INT0 输入。

Bit [6:4] **TC0RATE[2:0]**: TC0 分频选择位。

TC0RATE [2:0]	TC0X8 = 0	TC0X8 = 1
000	Fcpu / 256	Fosc / 128
001	Fcpu / 128	Fosc / 64
010	Fcpu / 64	Fosc / 32
011	Fcpu / 32	Fosc / 16
100	Fcpu / 16	Fosc / 8
101	Fcpu / 8	Fosc / 4
110	Fcpu / 4	Fosc / 2
111	Fcpu / 2	Fosc / 1

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

\* 注: 若 TC0CKS=1, 则 TC0 用作外部事件计数器, 此时不需要考虑 TC0RATE 的设置, P0.0 口无中断信号 (P00IRQ=0)。

## 8.2.3 TC1X8, TC0X8, TC0GN 标志

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>T0M</b>	-	-	-	-	TC1X8	TC0X8	TC0GN	-
读/写	-	-	-	-	R/W	R/W	R/W	-
复位后	-	-	-	-	0	0	0	-

- Bit 1 **TC0GN**: TC0 绿色模式唤醒功能控制位。  
0 = 禁止 TC0 的唤醒功能;  
1 = 允许 TC0 的唤醒功能。
- Bit 2 **TC0X8**: TC0 内部时钟选择控制位。  
0 = TC0 内部时钟来自 Fcpu, TC0RATE = Fcpu/2~Fcpu/256;  
1 = TC0 内部时钟来自 Fosc, TC0RATE = Fosc/1~Fosc/128。
- Bit 3 **TC1X8**: TC1 内部时钟选择控制位。  
0 = TC1 内部时钟来自 Fcpu, TC0RATE = Fcpu/2~Fcpu/256;  
1 = TC1 内部时钟来自 Fosc, TC0RATE = Fosc/1~Fosc/128。

\* 注: TC0CKS = 1 时, TC0X8 和 TC0RATE 可以忽略不计。

## 8.2.4 TC0C 计数寄存器

TC0C 控制 TC0 的时间间隔。

0DBH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>TC0C</b>	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

TC0C 初始值计算公式如下：

$$\text{TC0C 初始值} = N - (\text{TC0 中断间隔时间} * \text{输入时钟})$$

N 为 TC0 二进制计数范围。各模式下参数的设定如下表所示：

TC0CKS	TC0X8	PWM0	ALOAD0	TC0OUT	N	TC0C 有效值	TC0C 二进制计数范围	备注
0	0 (Fcpu/2~ Fcpu/256)	0	x	x	256	00H~0FFH	00000000b~11111111b	每计数 256 次溢出
		1	0	0	256	00H~0FFH	00000000b~11111111b	每计数 256 次溢出
		1	0	1	64	00H~3FH	xx000000b~xx111111b	每计数 64 次溢出
		1	1	0	32	00H~1FH	xxx00000b~xxx11111b	每计数 32 次溢出
		1	1	1	16	00H~0FH	xxxx0000b~xxxx1111b	每计数 16 次溢出
	1 (Fosc/1~ Fosc/128)	0	x	x	256	00H~0FFH	00000000b~11111111b	每计数 256 次溢出
		1	0	0	256	00H~0FFH	00000000b~11111111b	每计数 256 次溢出
		1	0	1	64	00H~3FH	xx000000b~xx111111b	每计数 64 次溢出
		1	1	0	32	00H~1FH	xxx00000b~xxx11111b	每计数 32 次溢出
		1	1	1	16	00H~0FH	xxxx0000b~xxxx1111b	每计数 16 次溢出
1	-	-	-	-	256	00H~0FFH	00000000b~11111111b	每计数 256 次溢出

- 例：TC0 的间隔时间为 10ms，时钟源来自 Fcpu (TC0CKS = 0, TC0X8 = 0)，无 PWM 输出 (PWM0 = 0)，高速时钟 = 4MHz，Fcpu=Fosc/4，TC0RATE=010 (Fcpu/64)。

$$\begin{aligned} \text{TC0C 初始值} &= N - (\text{TC0 中断间隔时间} * \text{输入时钟}) \\ &= 256 - (10\text{ms} * 4\text{MHz} / 4 / 64) \\ &= 256 - (10^{-2} * 4 * 10^6 / 4 / 64) \\ &= 100 \\ &= 64\text{H} \end{aligned}$$

TC0 中断时间对应表 (TC0X8 = 0)

TC0RATE	TC0CLOCK	高速模式(fcpu = 4MHz / 4)		低速模式(fcpu = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/256	最大溢出间隔时间	单步间隔时间 = max/256
000	Fcpu/256	65.536 ms	256 us	8000 ms	31250 us
001	Fcpu/128	32.768 ms	128 us	4000 ms	15625 us
010	Fcpu/64	16.384 ms	64 us	2000 ms	7812.5 us
011	Fcpu/32	8.192 ms	32 us	1000 ms	3906.25 us
100	Fcpu/16	4.096 ms	16 us	500 ms	1953.125 us
101	Fcpu/8	2.048 ms	8 us	250 ms	976.563 us
110	Fcpu/4	1.024 ms	4 us	125 ms	488.281 us
111	Fcpu/2	0.512 ms	2 us	62.5 ms	244.141 us

TC0X8 = 1

TC0RATE	TC0CLOCK	高速模式(fcpu = 4MHz / 4)		低速模式(fcpu = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/256	最大溢出间隔时间	单步间隔时间 = max/256
000	Fosc/128	8.192 ms	32 us	1000 ms	7812.5 us
001	Fosc/64	4.096 ms	16 us	500 ms	3906.25 us
010	Fosc/32	2.048 ms	8 us	250 ms	1953.125 us
011	Fosc/16	1.024 ms	4 us	125 ms	976.563 us
100	Fosc/8	0.512 ms	2 us	62.5 ms	488.281 us
101	Fosc/4	0.256 ms	1 us	31.25 ms	244.141 us
110	Fosc/2	0.128 ms	0.5 us	15.625 ms	122.07 us
111	Fosc/1	0.064 ms	0.25 us	7.813 ms	61.035 us

## 8.2.5 TC0R 自动装载寄存器

TC0 的自动重装功能由 TC0M 的 ALOAD0 位控制。当 TC0C 溢出时，TC0R 的值自动装入 TC0C 中。这样，用户使用的过程中就不需要在中断中复位 TC0C。

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

\* 注：在 PWM 模式下，系统自动开启重装功能，ALOAD0 用于控制溢出范围。

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

TC0R 初始值计算公式如下：

$$\text{TC0R 初始值} = N - (\text{TC0 中断间隔时间} * \text{输入时钟})$$

N 是 TC0 最大溢出值。TC0 的溢出时间和有效值见下表：

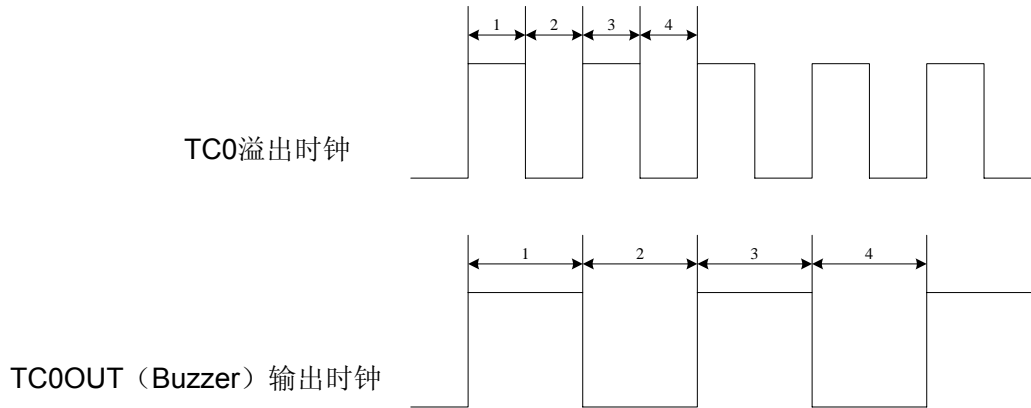
TC0CKS	TC0X8	PWM0	ALOAD0	TC0OUT	N	TC0R 有效值	TC0R 二进制有效范围
0	0 (Fcpu/2~ Fcpu/256)	0	x	x	256	00H~0FFH	0000000b~1111111b
		1	0	0	256	00H~0FFH	0000000b~1111111b
		1	0	1	64	00H~3FH	xx00000b~xx11111b
		1	1	0	32	00H~1FH	xxx0000b~xxx1111b
		1	1	1	16	00H~0FH	xxxx000b~xxxx111b
	1 (Fosc/1~ Fosc/128)	0	x	x	256	00H~0FFH	0000000b~1111111b
		1	0	0	256	00H~0FFH	0000000b~1111111b
		1	0	1	64	00H~3FH	xx00000b~xx11111b
		1	1	0	32	00H~1FH	xxx0000b~xxx1111b
		1	1	1	16	00H~0FH	xxxx000b~xxxx111b
1	-	-	-	-	256	00H~0FFH	0000000b~1111111b

➤ 例：TC0 中断间隔时间设置为 10ms，时钟源选 Fcpu (TC0KS=0, TC0X8 = 0)，无 PWM 输出 (PWM0=0)，高速时钟为外部 4MHz，Fcpu=Fosc/4，TC0RATE=010 (Fcpu/64)。

$$\begin{aligned} \text{TC0R} &= N - (\text{TC0 中断间隔时间} * \text{输入时钟}) \\ &= 256 - (10\text{ms} * 4\text{MHz} / 4 / 64) \\ &= 256 - (10^{-2} * 4 * 10^6 / 4 / 64) \\ &= 100 \\ &= 64\text{H} \end{aligned}$$

## 8.2.6 TC0 时钟频率输出（蜂鸣器输出）

对 TC0 时钟频率进行适当设置可得到特定频率的蜂鸣器输出（TC0OUT），并通过引脚 P5.4 输出。单片机内部设置 TC0 的溢出频率经过 2 分频后作为 TC0OUT 的频率，即 TC0 每溢出 2 次 TC0OUT 输出一个完整的脉冲，此时，P5.4 的 I/O 功能自动被禁止。TC0OUT 输出波形如下：



若外部高速时钟选择 4MHz，系统时钟源采用外部时钟  $F_{osc}/4$ ，程序中设置  $TC0RATE2 \sim TC0RATE1 = 110$ ， $TC0C = TC0R = 131$ ，则 TC0 的溢出频率为 2KHz，TC0OUT 的输出频率为 1KHz。下面给出范例程序。

### ➤ 例：设置 TC0OUT（P5.4）。

```

MOV      A,#01100000B
B0MOV    TC0M,A           ; TC0 速率 = Fcpu/4。

MOV      A,#131
B0MOV    TC0C,A           ; 自动装载参考值设置。
B0MOV    TC0R,A

B0BSET   FTC0OUT          ; TC0 的输出信号由 P5.4 输出，禁止 P5.4 的普通 I/O 功能。
B0BSET   FALOAD0          ; 使能 TC0 自动重装功能。
B0BSET   FTC0ENB          ; 开启 TC0 定时器。

```

\* 注：蜂鸣器的输出有效时，“PWM0OUT”必须被置为“0”。

## 8.2.7 TC0 操作流程

TC0 定时器可用于定时器中断、事件计数、TC0OUT 和 PWM。下面分别举例说明。

### ☞ 停止 TC0 计数，禁止 TC0 中断，并清 TC0 中断请求标志。

```
B0BCLR    FTC0ENB    ; 停止 TC0 计数、TC0OUT 和 PWM。
B0BCLR    FTC0IEN    ; 禁止 TC0 中断。
B0BCLR    FTC0IRQ    ; 清 TC0 中断请求标志。
```

### ☞ 设置 TC0 的速率 (不包含事件计数模式)。

```
MOV       A, #0xxx0000b    ; TC0M 的 bit4~bit6 控制 TC0 的速率为 x000xxxxb~x111xxxxb。
B0MOV     TC0M,A          ; 禁止 TC0 中断。
```

### ☞ 设置 TC0 的时钟源。

; 选择 TC0 内部/外部时钟源。

```
B0BCLR    FTC0CKS    ; 内部时钟。
```

或

```
B0BSET    FTC0CKS    ; 外部时钟。
```

;选择 TC0 Fcpu/Fosc 内部时钟源。

```
B0BCLR    FTC0X8    ; Fcpu 内部时钟。
```

或

```
B0BSET    FTC0X8    ; Fosc 内部时钟。
```

**\* 注：在 TC0 外部时钟模式下，TC0X8 可以忽略不计。**

### ☞ 设置 TC0 的自动装载模式。

```
B0BCLR    FALOAD0    ; 禁止 TC0 自动装载功能。
```

或

```
B0BSET    FALOAD0    ; 使能 TC0 自动装载功能。
```

### ☞ 设置 TC0 中断间隔时间，TC0OUT (Buzzer) 频率或 PWM 占空比。

; 设置 TC0 中断间隔时间，TC0OUT (Buzzer) 频率或 PWM 占空比。

```
MOV       A, #7FH      ; TC0 的模式决定 TC0C 和 TC0R 的值。
B0MOV     TC0C,A      ; 设置 TC0C 的值。
B0MOV     TC0R,A      ; 在自动装载模式或 PWM 模式下设置 TC0R 的值。
```

;PWM 模式下设置 PWM 的周期。

```
B0BCLR    FALOAD0    ; ALOAD0, TC0OUT = 00, PWM 周期 = 0~255。
B0BCLR    FTC0OUT
```

或

```
B0BCLR    FALOAD0    ; ALOAD0, TC0OUT = 01, PWM 周期 = 0~63。
B0BSET    FTC0OUT
```

或

```
B0BSET    FALOAD0    ; ALOAD0, TC0OUT = 10, PWM 周期 = 0~31。
B0BCLR    FTC0OUT
```

或

```
B0BSET    FALOAD0    ; ALOAD0, TC0OUT = 11, PWM 周期 = 0~15。
B0BSET    FTC0OUT
```

### ☞ 设置 TC0 的模式。

```
B0BSET    FTC0IEN    ; 使能 TC0 中断。
```

或

```
B0BSET    FTC0OUT    ; 使能 TC0OUT (Buzzer) 功能。
```

或

```
B0BSET    FPWM0OUT    ; 使能 PWM。
```

或

```
B0BSET    FTC0GN      ; 使能 TC0 的绿色模式下的唤醒功能。
```

### ☞ 开启 TC0 定时器。

```
B0BSET    FTC0ENB    ;
```

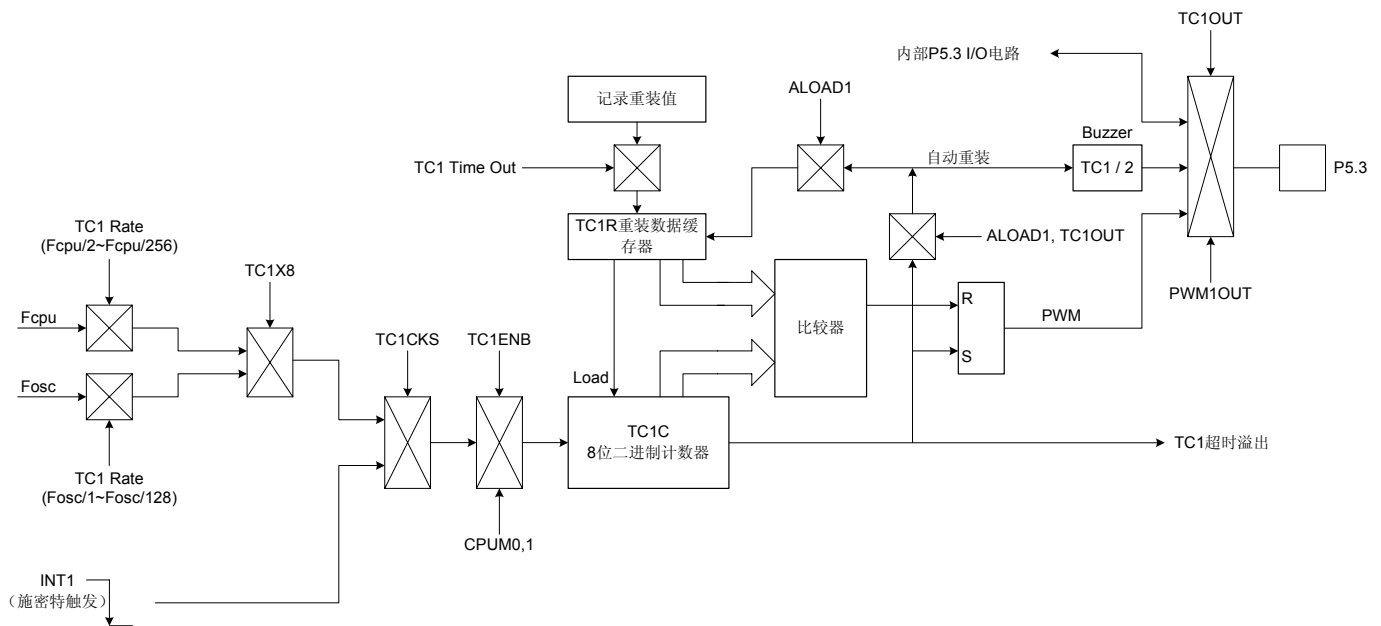
## 8.3 定时/计数器 TC1

### 8.3.1 概述

定时/计数器 TC1 具有双时钟源，可根据实际的需要选择内部时钟或外部时钟作为计时标准。其中，内部时钟源来自  $F_{cpu}$  或  $F_{osc}$  ( $F_{osc}$  由 TC1X8 标志控制)。外部时钟源 INT1 从 P0.1 端输入（下降沿触发）。寄存器 TC1M 控制 TC1 时钟源的选择。当 TC1 从 0FFH 溢出到 00H 时，TC1 在继续计数的同时产生一个超时信号，触发 TC1 中断请求。在 PWM 模式，TC1 的溢出由寄存器 ALOAD1 的位 TC1OUT 控制。

TC1 的主要功能如下：

- ☞ **8 位可编程定时器：** 根据选择的时钟信号，产生周期中断；
- ☞ **外部事件计数器：** 对外部事件计数；
- ☞ **Buzzer 输出；**
- ☞ **PWM 输出。**





### 8.3.2 TC1M 模式寄存器

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

Bit 0 **PWM1OUT**: PWM 输出控制位。  
0 = 禁止 PWM 输出;  
1 = 使能 PWM 输出, PWM 输出占空比由 TC1OUT 和 ALOAD1 控制。

Bit 1 **TC1OUT**: TC1 超时输出信号控制。仅当 **PWM1OUT = 0** 时有效。  
0 = 禁止, P5.3 作为输入/输出口;  
1 = 使能, P5.3 输出 TC1OUT 信号。

Bit 2 **ALOAD1**: 自动装载控制位。仅当 **PWM1OUT = 0** 时有效。  
0 = 禁止 TC1 自动装载;  
1 = 使能 TC1 自动装载。

Bit 3 **TC1CKS**: TC1 时钟源控制位。  
0 = 内部时钟 (Fcpu 或 Fosc);  
1 = 外部时钟, 由 P0.1/INT1 输入。

Bit [6:4] **TC1RATE[2:0]**: TC1 分频选择位。

TC1RATE [2:0]	TC1X8 = 0	TC1X8 = 1
000	Fcpu / 256	Fosc / 128
001	Fcpu / 128	Fosc / 64
010	Fcpu / 64	Fosc / 32
011	Fcpu / 32	Fosc / 16
100	Fcpu / 16	Fosc / 8
101	Fcpu / 8	Fosc / 4
110	Fcpu / 4	Fosc / 2
111	Fcpu / 2	Fosc / 1

Bit 7 **TC1ENB**: TC1 启动控制位。  
0 = 禁止 TC1 定时器;  
1 = 开启 TC1 定时器。

\* 注: 若 TC1CKS=1, 则 TC1 用作外部事件计数器, 此时不需要考虑 TC1RATE 的设置, P0.1 口无中断信号 (P0.1IRQ=0)。

### 8.3.3 TC1X8 标志

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>TOM</b>	-	-	-	-	TC1X8	-	-	-
读/写	-	-	-	-	R/W	-	-	-
复位后	-	-	-	-	0	-	-	-

Bit 3 **TC1X8**: TC1 内部时钟选择控制位。  
0 = TC1 内部时钟来自 Fcpu, TC1RATE = Fcpu/2~Fcpu/256;  
1 = TC1 内部时钟来自 Fosc, TC1RATE = Fosc/1~Fosc/128。

\* 注: TC1CKS = 1 时, TC1X8 和 TC1RATE 可以忽略不计。

## 8.3.4 TC1C 计数寄存器

TC1C 控制 TC1 的时间间隔。

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

TC1C 初始值的计算公式如下：

$$\text{TC1C 初始值} = N - (\text{TC1 中断间隔时间} * \text{输入时钟})$$

N 为 TC1 二进制计数范围。各模式下参数的设定如下表所示：

TC1CKS	TC1X8	PWM1	ALOAD1	TC1OUT	N	TC1C 有效值	TC1C 二进制计数范围	备注
0	0 (Fcpu/2~ Fcpu/256)	0	x	x	256	00H~0FFH	0000000b~1111111b	每计数 256 次溢出
		1	0	0	256	00H~0FFH	0000000b~1111111b	每计数 256 次溢出
		1	0	1	64	00H~3FH	xx00000b~xx11111b	每计数 64 次溢出
		1	1	0	32	00H~1FH	xxx0000b~xxx1111b	每计数 32 次溢出
		1	1	1	16	00H~0FH	xxxx000b~xxxx111b	每计数 16 次溢出
	1 (Fosc/1~ Fosc/128)	0	x	x	256	00H~0FFH	0000000b~1111111b	每计数 256 次溢出
		1	0	0	256	00H~0FFH	0000000b~1111111b	每计数 256 次溢出
		1	0	1	64	00H~3FH	xx00000b~xx11111b	每计数 64 次溢出
		1	1	0	32	00H~1FH	xxx0000b~xxx1111b	每计数 32 次溢出
		1	1	1	16	00H~0FH	xxxx000b~xxxx111b	每计数 16 次溢出
1	-	-	-	-	256	00H~0FFH	0000000b~1111111b	每计数 256 次溢出

- 例：TC1 的间隔时间为 10ms，时钟源来自 Fcpu (TC1CKS = 0, TC1X8 = 0)，无 PWM 输出 (PWM1 = 0)，高速时钟 = 4MHz，Fcpu=Fosc/4，TC1RATE=010 (Fcpu/64)。

$$\begin{aligned} \text{TC1C 初始值} &= N - (\text{TC1 中断间隔时间} * \text{输入时钟}) \\ &= 256 - (10\text{ms} * 4\text{MHz} / 4 / 64) \\ &= 256 - (10^{-2} * 4 * 10^6 / 4 / 64) \\ &= 100 \\ &= 64\text{H} \end{aligned}$$

TC1 中断时间对应表，TC1X8 = 0

TC1RATE	TC1CLOCK	高速模式(fcpu = 4MHz / 4)		低速模式(fcpu = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/256	最大溢出间隔时间	单步间隔时间 = max/256
000	Fcpu/256	65.536 ms	256 us	8000 ms	31250 us
001	Fcpu/128	32.768 ms	128 us	4000 ms	15625 us
010	Fcpu/64	16.384 ms	64 us	2000 ms	7812.5 us
011	Fcpu/32	8.192 ms	32 us	1000 ms	3906.25 us
100	Fcpu/16	4.096 ms	16 us	500 ms	1953.125 us
101	Fcpu/8	2.048 ms	8 us	250 ms	976.563 us
110	Fcpu/4	1.024 ms	4 us	125 ms	488.281 us
111	Fcpu/2	0.512 ms	2 us	62.5 ms	244.141 us

TC1 中断时间对应表，TC1X8 = 1

TC1RATE	TC1CLOCK	高速模式(fcpu = 4MHz / 4)		低速模式(fcpu = 32768Hz / 4)	
		最大溢出间隔时间	单步间隔时间 = max/256	最大溢出间隔时间	单步间隔时间 = max/256
000	Fosc/128	8.192 ms	32 us	1000 ms	7812.5 us
001	Fosc/64	4.096 ms	16 us	500 ms	3906.25 us
010	Fosc/32	2.048 ms	8 us	250 ms	1953.125 us
011	Fosc/16	1.024 ms	4 us	125 ms	976.563 us
100	Fosc/8	0.512 ms	2 us	62.5 ms	488.281 us
101	Fosc/4	0.256 ms	1 us	31.25 ms	244.141 us
110	Fosc/2	0.128 ms	0.5 us	15.625 ms	122.07 us
111	Fosc/1	0.064 ms	0.25 us	7.813 ms	61.035 us

### 8.3.5 TC1R 自动装载寄存器

TC1 的自动装载功能由 TC1M 的 ALOAD1 位控制。当 TC1C 溢出时，TC1R 的值自动装入 TC1C 中。这样，用户在使用过程中就不需要在中断中复位 TC1C。

TC1 为双重缓存器结构。若程序对 TC1R 进行了修改，那么修改后的 TC1R 值首先被暂存在 TC1R 的第一个缓存器中，TC1 溢出后，TC1R 的新值就会被存入 TC1R 缓存器中，从而避免 TC1 中断时间出错以及 PWM 和蜂鸣器误动作。

\* 注：在 PWM 模式下，系统自动开启自动重装功能，ALOAD1 用于控制溢出范围。

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

TC1R 初始值计算公式如下：

$$\text{TC1R 初始值} = N - (\text{TC1 中断间隔时间} * \text{输入时钟})$$

N 是 TC1 最大溢出值。TC1 的溢出时间和有效值见下表：

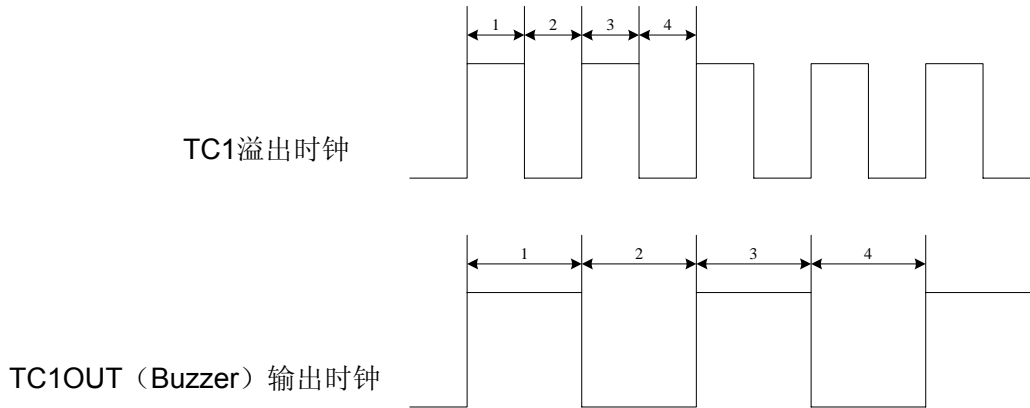
TC1CKS	TC1X8	PWM1	ALOAD1	TC1OUT	N	TC1R 有效值	TC1R 二进制有效范围
0	0 (Fcpu/2~ Fcpu/256)	0	x	x	256	00H~0FFH	00000000b~11111111b
		1	0	0	256	00H~0FFH	00000000b~11111111b
		1	0	1	64	00H~3FH	xx000000b~xx111111b
		1	1	0	32	00H~1FH	xxx00000b~xxx11111b
		1	1	1	16	00H~0FH	xxxx0000b~xxxx1111b
	1 (Fosc/1~ Fosc/128)	0	x	x	256	00H~0FFH	00000000b~11111111b
		1	0	0	256	00H~0FFH	00000000b~11111111b
		1	0	1	64	00H~3FH	xx000000b~xx111111b
		1	1	0	32	00H~1FH	xxx00000b~xxx11111b
		1	1	1	16	00H~0FH	xxxx0000b~xxxx1111b
1	-	-	-	256	00H~0FFH	00000000b~11111111b	

➤ 例：TC1 中断间隔时间设置为 10ms，时钟源选 Fcpu (TC1CKS=0, TC1X8 = 0)，无 PWM 输出 (PWM1=0)，高速时钟为外部 4MHz，Fcpu=Fosc/4，TC1RATE=010 (Fcpu/64)。

$$\begin{aligned} \text{TC1R 有效值} &= N - (\text{TC1 中断间隔时间} * \text{输入时钟}) \\ &= 256 - (10\text{ms} * 4\text{MHz} / 4 / 64) \\ &= 256 - (10^{-2} * 4 * 10^6 / 4 / 64) \\ &= 100 \\ &= 64\text{H} \end{aligned}$$

### 8.3.6 TC1 时钟频率输出（蜂鸣器输出）

对 TC1 时钟频率进行适当设置可得到特定频率的蜂鸣器输出（TC1OUT），并通过引脚 P5.3 输出。单片机内部设置 TC1 的溢出频率经过 2 分频后作为 TC1OUT 的频率，即 TC1 每溢出 2 次 TC1OUT 输出一个完整的脉冲，此时，P5.3 的 I/O 功能自动被禁止。TC1OUT 输出波形如下：



若外部高速时钟选择 4MHz，系统时钟源采用外部时钟  $F_{osc}/4$ ，程序中设置  $TC1RATE2 \sim TC1RATE1 = 110$ ， $TC1C = TC1R = 131$ ，则 TC1 的溢出频率为 2KHz，TC1OUT 的输出频率为 1KHz。下面给出范例程序。

➤ 例：设置 TC1OUT（P5.3）。

```

MOV      A,#60H
B0MOV   TC1M,A           ; TC1 速率= Fcpu/4。

MOV      A,#131
B0MOV   TC1C,A           ; 自动装载参考值设置。
B0MOV   TC1R,A

B0BSET  FTC1OUT          ; TC1 的输出信号由 P5.3 输出，禁止 P5.3 的普通 I/O 功能。
B0BSET  FALOAD1         ; 使能 TC1 自动重装功能。
B0BSET  FTC1ENB         ; 开启 TC1 定时器。

```

\* 注：蜂鸣器的输出有效时，“PWM1OUT”必须被置为“0”。

### 8.3.7 TC1 操作流程

TC1 定时器可用于定时器中断、事件计数、TC1OUT 和 PWM。下面分别举例说明。

☞ 停止 TC1 计数，禁止 TC1 中断并清 TC1 中断请求标志。

```
B0BCLR    FTC1ENB    ; 停止 TC1 计数、TC1OUT 和 PWM。
B0BCLR    FTC1IEN    ; 禁止 TC1 中断。
B0BCLR    FTC1IRQ    ; 清 TC1 中断请求标志。
```

☞ 设置 TC1 的速率 (不包含事件计数模式)。

```
MOV      A, #0xxx0000b    ;TC1M 的 bit4~bit6 控制 TC1 的速率在 x000xxxxb~x111xxxxb。
B0MOV    TC1M,A          ; 禁止 TC1 中断。
```

☞ 设置 TC1 的时钟源。

; 选择 TC1 内部/外部时钟源。

```
B0BCLR    FTC1CKS    ; 内部时钟。
```

或

```
B0BSET    FTC1CKS    ; 外部时钟。
```

;选择 TC1 Fcpu/Fosc 内部时钟源。

```
B0BCLR    FTC1X8    ; Fcpu 内部时钟。
```

或

```
B0BSET    FTC1X8    ; Fosc 内部时钟。
```

**\* 注：在 TC1 外部时钟模式下，TC1X8 可以忽略不计。**

☞ 设置 TC1 的自动装载模式。

```
B0BCLR    FALOAD1    ; 禁止 TC1 自动装载功能。
```

或

```
B0BSET    FALOAD1    ; 使能 TC1 自动装载功能。
```

☞ 设置 TC1 中断间隔时间，TC1OUT (Buzzer) 频率或 PWM 占空比。

; 设置 TC1 中断间隔时间，TC1OUT (Buzzer) 频率或 PWM 占空比。

```
MOV      A,#7FH      ; TC1 的模式决定 TC1C 和 TC1R 的值。
```

```
B0MOV    TC1C,A      ; 设置 TC1C 的值。
```

```
B0MOV    TC1R,A      ; 在自动装载模式或 PWM 模式下设置 TC1R 的值。
```

;PWM 模式下设置 PWM 周期。

```
B0BCLR    FALOAD1    ; ALOAD1, TC1OUT = 00, PWM 周期 = 0~255。
```

```
B0BCLR    FTC1OUT
```

或

```
B0BCLR    FALOAD1    ; ALOAD1, TC1OUT = 01, PWM 周期 = 0~63。
```

```
B0BSET    FTC1OUT
```

或

```
B0BSET    FALOAD1    ; ALOAD1, TC1OUT = 10, PWM 周期 = 0~31。
```

```
B0BCLR    FTC1OUT
```

或

```
B0BSET    FALOAD1    ; ALOAD1, TC1OUT = 11, PWM 周期 = 0~15。
```

```
B0BSET    FTC1OUT
```

☞ 设置 TC1 的模式。

```
B0BSET    FTC1IEN    ; 使能 TC1 中断。
```

或

```
B0BSET    FTC1OUT    ; 使能 TC1OUT (Buzzer) 功能。
```

或

```
B0BSET    FPWM1OUT   ; 使能 PWM。
```

☞ 开启 TC1 定时器。

```
B0BSET    FTC1ENB    ;
```

## 8.4 PWM 功能说明

### 8.4.1 概述

PWM 信号输出到 PWMnOUT (P5.3/P5.4 引脚)，位 TCnOUT 和 ALOADn 控制 PWM 输出的阶数 (256、64、32 和 16)。8 位计数器 TCnC 计数过程中不断与 TCnR 相比较，当 TCnC 计数到两者相等时，PWM 输出低电平，当 TCnC 再次从零开始计数时，PWM 被强制输出高电平。PWMn 输出占空比 = TCnR/阶数 (阶数= 256、64、32 或 16)。

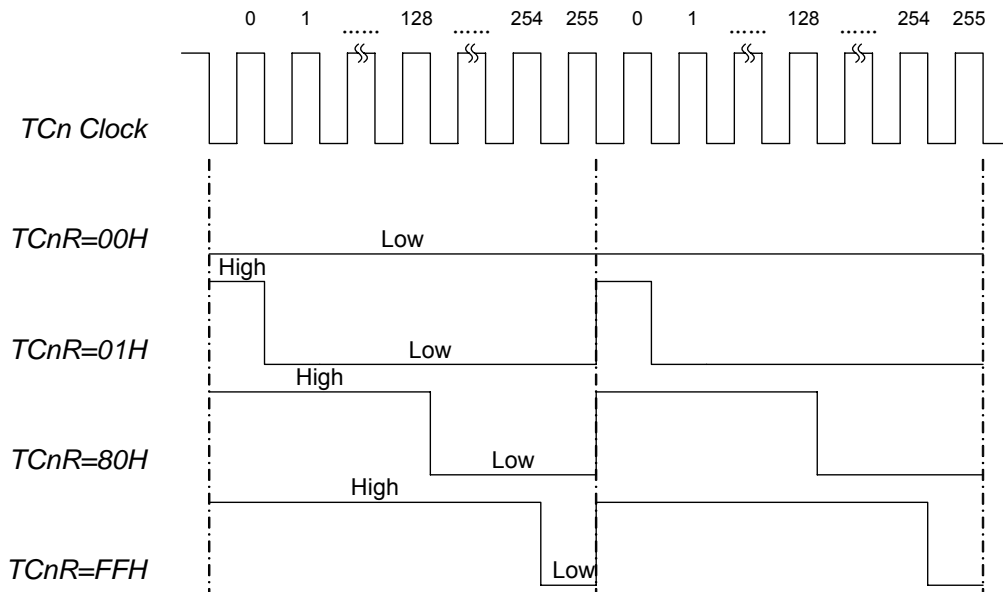
参考寄存器保持输入 00H 可使 PWM 的输出长时间维持在低电平，通过修改 TCnR 可改变 PWM 输出占空比。

\* 注：TCn、TCnC 的“n”的值只能是 0 或 1。n = 0 时为 TC0 模式，n = 1 时为 TC1 模式。

\* 注：TCn 为双重缓存器结构，调整 TCnR 的值可以改变 PWM 的输出占空比。用户可随时改变 TCnR 的值，但是只有在 TCn 溢出后，这一修改值才真正被写入 TCnR 中。

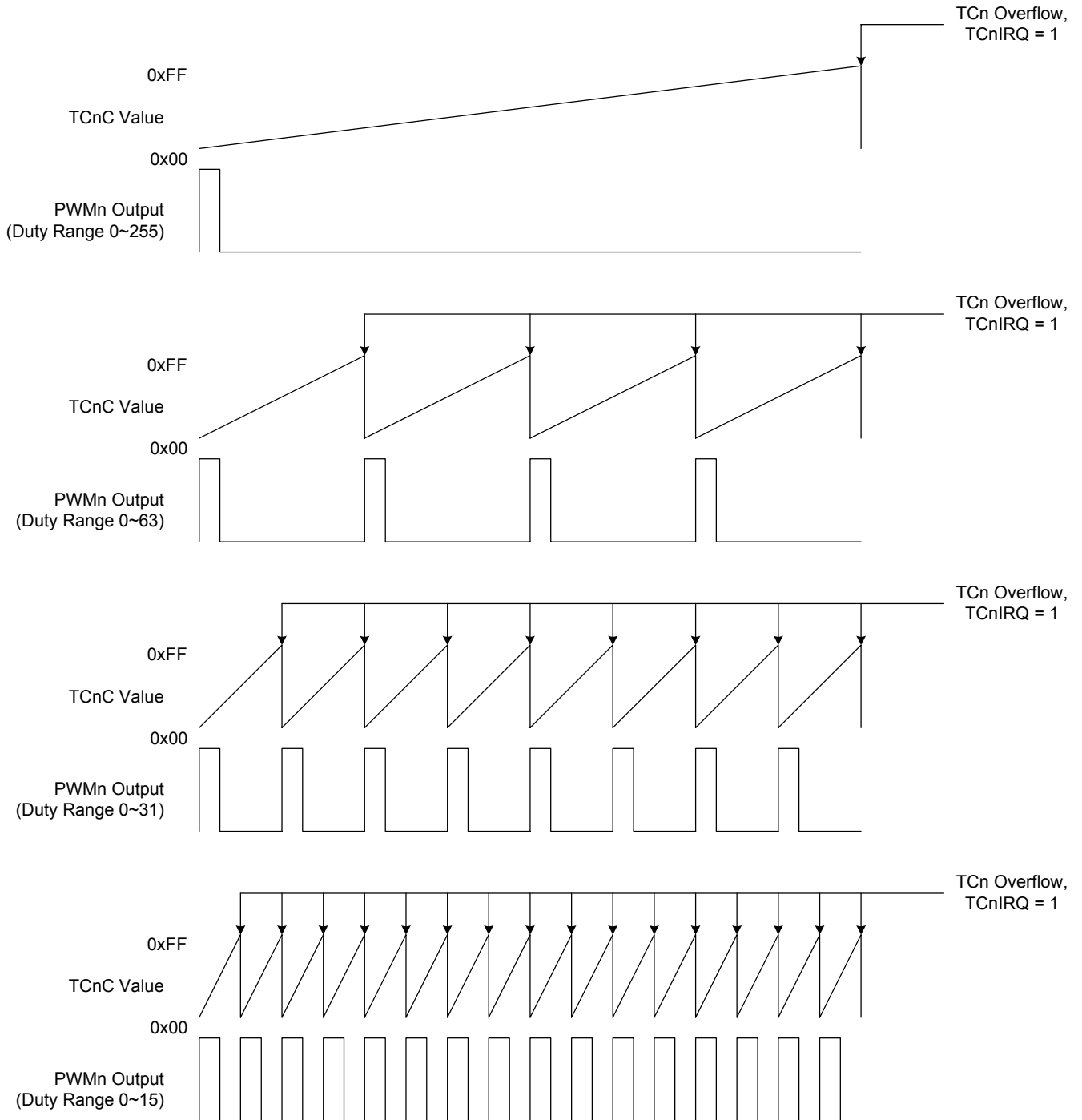
ALOADn	TCnOUT	PWM 占空比范围	TCnC 有效值	TCnR 有效范围	MAX. PWM 输出频率 (Fcpu = 4MHz)	备注
0	0	0/256~255/256	00H~0FFH	00H~0FFH	7.8125K	每计数 256 次溢出
0	1	0/64~63/64	00H~3FH	00H~3FH	31.25K	每计数 64 次溢出
1	0	0/32~31/32	00H~1FH	00H~1FH	62.5K	每计数 32 次溢出
1	1	0/16~15/16	00H~0FH	00H~0FH	125K	每计数 16 次溢出

PWM 输出占空比随 TCnR 的变化而变化：0/256~255/256。



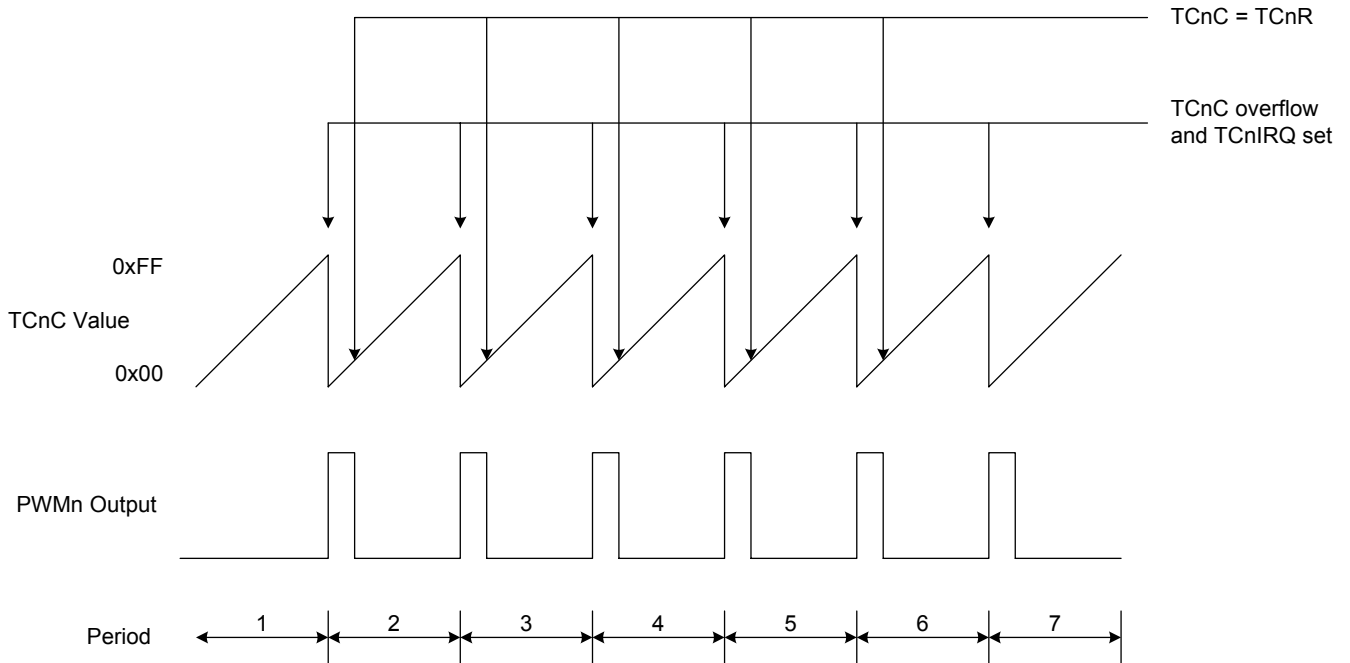
## 8.4.2 TCnIRQ 和 PWM 输出占空比

在 PWM 模式下，TCnIRQ 的频率与 PWM 的占空比有关，具体情况如下图所示：

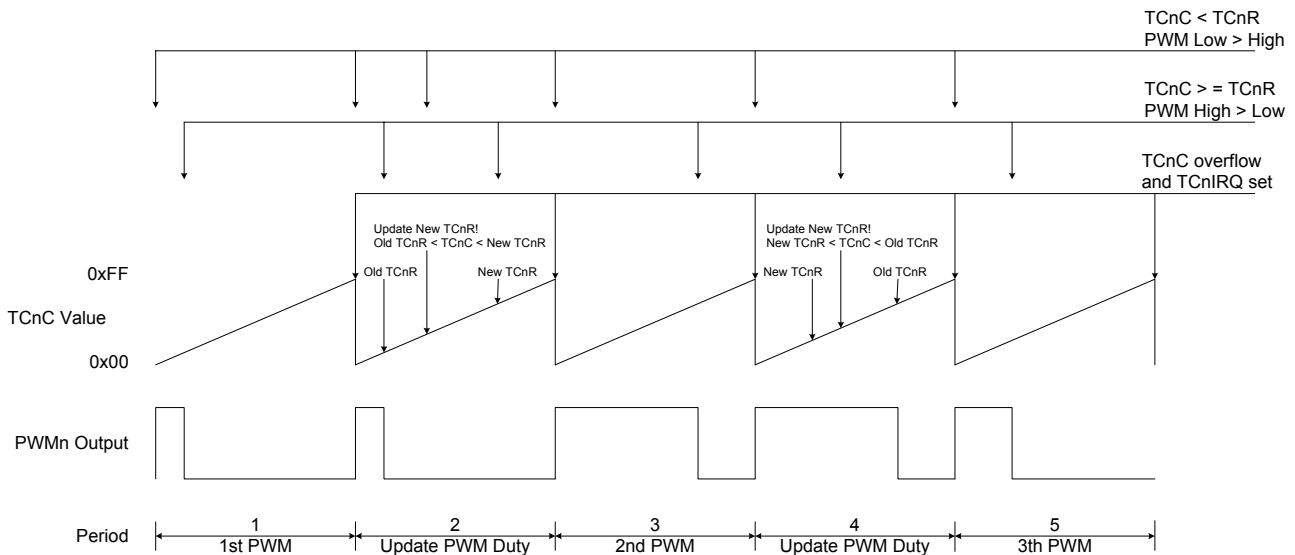


### 8.4.3 PWM 输出占空比和 TCnR 的变化

在 PWM 模式下，系统随时比较 TCnC 和 TCnR 的异同。若  $TCnC < TCnR$ ，PWM 输出高电平，反之则输出低电平。当 TCnC 发生改变的时候，PWM 将在下一周期改变输出占空比。如果 TCnR 保持恒定，那么 PWM 输出波形也保持稳定波形。



上图所示是 TCnR 恒定时的波形。每当 TCnC 溢出时，PWM 都输出高电平， $TCnC \geq TCnR$  时，PWM 即输出低电平。下面所示是 TC0R 发生变化时对应的波形图：



在 period 2 和 period 4 中，显示新的占空比 (TC0R)，但 PWM 在 period 2 和 period 4 的占空比要在下一个 period 才会改变。这样，可以避免 PWM 不随设定改变或在同一个周期内改变两次，从而避免系统发生不可预知的误动作。



## 8.4.4 PWM 编程举例

- 例：PWM 输出设置。外部高速振荡器输出频率 = 4MHZ,  $F_{cpu} = F_{osc}/4$ , PWM 输出占空比 = 30/256, 输出频率 1KHZ, PWM 时钟源来自外部时钟, TC0 速率 =  $F_{cpu}/4$ ,  $TC0RATE2 \sim TC0RATE1 = 110$ ,  $TC0C = TC0R = 30$ 。

```

MOV      A,#01100000B
B0MOV   TC0M,A           ; TC0 速率= $F_{cpu}/4$ 。

MOV      A,#30
B0MOV   TC0C,A           ; PWM 输出占空比=30/256。
B0MOV   TC0R,A

B0BCLR  FTC0OUT           ; 占空比变化范围: 0/256~255/256。
B0BCLR  FALOAD0
B0BSET  FPWM0OUT         ; PWM0 输出至 P5.4, 禁止 P5.4 I/O 功能。
B0BSET  FTC0ENB

```

\* 注：TCnR 为只写寄存器，不能用 INCMS 和 DECMS 指令对其进行操作。

- 例：改变 TC0R 的内容。

```

MOV      A, #30H
B0MOV   TC0R, A

INCMS   BUF0
NOP
B0MOV   A, BUF0
B0MOV   TC0R, A

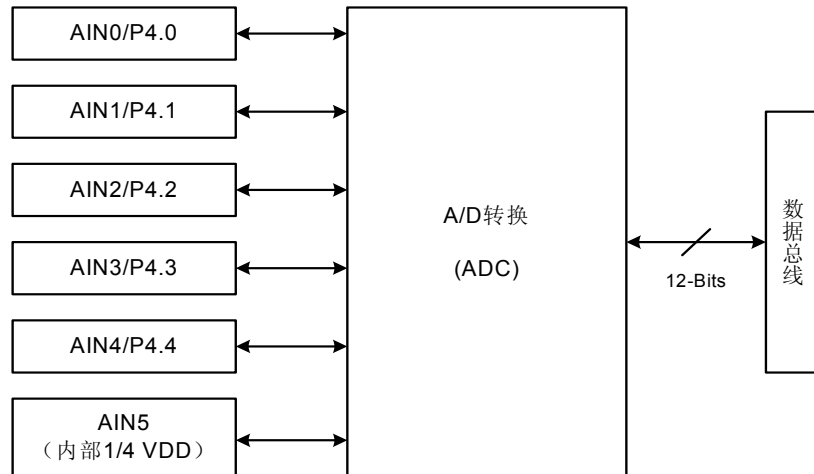
```

\* 注：PWM 可以在中断下工作。

# 9 5+1 通道 ADC

## 9.1 概述

SN8P2711A 模数转换模块共有 5 条外部通道 (AIN0~AIN4) 和一条内部通道 (AIN5: 内部 1/4VDD)，4096 阶分辨率的 A/D 转换器，可以将模拟信号转换成 12 位数字信号。进行 AD 转换时，首先要选择输入通道 (AIN0~AIN5)，然后把 GCHS 和 ADS 位置“1”，启动 AD 转换。转换结束后，系统自动将 EOC 设置为“1”，并将转换结果存入寄存器 ADB 和寄存器 ADR 中。



## 9.2 ADM 寄存器

0B1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>ADM</b>	ADENB	ADS	EOC	GCHS	-	CHS2	CHS1	CHS0
读/写	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W
复位后	0	0	0	0	-	0	0	0

Bit 7 ADENB: ADC 控制位。

0 = 禁止;  
1 = 使能。

Bit 6 ADS: ADC 启动位。

0 = 停止;  
1 = 开始。

Bit 5 EOC: ADC 状态控制位。

0 = 转换过程中;  
1 = 转换结束, ADS 复位。

Bit 4 GCHS: 通道选择位。

0 = 禁止 AIN 通道;  
1 = 使能 AIN 通道。

Bit[2:0] CHS[2:0]: ADC 输入通道选择位。

000 = AIN0; 001 = AIN1; 010 = AIN2; 011 = AIN3; 100 = AIN4; 101 = AIN5。

AIN5 是内部 1/4 VDD 输入通道，外部没有输入引脚。AIN5 可以作为电池系统的电池检测器。为了选择合适的内部 VREFH 电平并进行比较，系统内置了这个高性能、廉价的低电池检测器。

\* 注: 若 ADENB = 1, 用户应设置 P4.n/AINn 为无上拉电阻的输入模式。系统不会自动设置。若已经设置了 P4CON.n, P4.n/AINn 的数字 I/O 功能 (包括上拉电阻) 都是隔离开来的。

### 9.3 ADR 寄存器

0B3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>ADR</b>	-	ADCKS1	-	ADCKS0	ADB3	ADB2	ADB1	ADB0
读/写	-	R/W	-	R/W	R	R	R	R
复位后	-	0	-	0	X	X	X	X

Bit[6,4] ADCKS1, ADCKS0: **ADC 时钟源选择位。**

ADCKS1	ADCKS0	ADC 时钟源
0	0	Fcpu/16
0	1	Fcpu/8
1	0	Fcpu
1	1	Fcpu/2

Bit[3:0] ADB[3:0]: ADC 12 位分辨率的低字节数据缓存器。

\* 注: ADC 缓存器 ADR [3:0]复位后的初始值是未知的。

### 9.4 ADB 寄存器

0B2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>ADB</b>	ADB15	ADB14	ADB13	ADB12	ADB11	ADB10	ADB9	ADB8
读/写	R	R	R	R	R	R	R	R
复位后	X	X	X	X	X	X	X	X

Bit[7:0] ADB[7:0]: ADC 12 位分辨率的高字节数据缓存器。

8 位数据缓存器 ADB 用来保存 AD 转换结果的高 8 位 (bit4~bit11)，转换结果的低 4 位则保存在 ADR 寄存器中。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。

AIN 的输入电压 v.s. ADB 的输出数据

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

针对不同的应用，用户可能需要精度介于 8 位到 12 位之间的 AD 转换器。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	○	○	○	○	○	○	○	○	x	x	x	x
9-bit	○	○	○	○	○	○	○	○	○	x	x	x
10-bit	○	○	○	○	○	○	○	○	○	○	x	x
11-bit	○	○	○	○	○	○	○	○	○	○	○	x
12-bit	○	○	○	○	○	○	○	○	○	○	○	○

○ = 可选位, x = 未使用的位

\* 注: ADC 缓存器 ADB 在复位后的初始值是未知的。

## 9.5 P4CON 寄存器

P4 口和 ADC 的输入口共享。同一时间只能设置 P4 口的一个引脚作为 ADC 的测量信号输入口（通过 ADM 寄存器来设置），其它引脚则作为普通 I/O 使用。具体应用中，当向 CMOS 结构端口输入一个模拟信号，尤其当模拟信号为 1/2 VDD 时，将可能产生额外的漏电流。同样，当 P4 口外接多个模拟信号时，也会产生额外的漏电流。在睡眠模式下，上述漏电流会严重影响到系统的整体功耗。P4CON 为 P4 口的配置寄存器。将 P4CON[7:0]置"1"，其对应的 P4 口将被设置为纯模拟信号输入口，从而避免上述漏电流的情况。

0AEH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>P4CON</b>	-	-	-	P4CON4	P4CON3	P4CON2	P4CON1	P4CON0
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

Bit[4:0] P4CON[4:0]: P4.n 配置控制位。

0 = P4.n 作为模拟输入（ADC 输入）引脚或者数字 I/O 引脚；

1 = P4.n 只能作为单纯的模拟输入引脚，不能作为数字 I/O 引脚。

\* 注：当 P4.n 为普通 I/O 而不是 ADC 通道时，P4CON.n 必须置“0”，否则 P4.n 的数字 I/O 信号会被隔离。

## 9.6 VREFH 寄存器

0AFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>VREFH</b>	EVHENB	-	-	-	-	-	VHS1	VHS0
读/写	R/W	-	-	-	-	-	R/W	R/W
复位后	0	-	-	-	-	-	0	0

Bit[1:0] VHS[1:0]: ADC 内部参考电压选择位。

VHS1	VHS0	内部 VREFH 电压
1	1	VDD
1	0	4.0V
0	1	3.0V
0	0	2.0V

\* 注：若由 VHS[1:0]控制选择的内部 VREFH 电平高于 VDD，内部 VREFH 为 VDD。例：VHS[1:0] = 10（内部 VREFH = 4.0V），VDD = 3.0V，则实际内部 VREFH = 3.0V。

Bit[7] EVHENB: ADC 内部参考电压控制位。

0 = 允许 ADC 内部 VREFH 功能，VREFH 引脚是 P4.0/AIN0 引脚；

1 = 禁止 ADC 内部 VREFH 功能，P4.0/AIN0/VREFH 引脚来自外部 VREFH 输入引脚。

\* 注：若 EVHENB = 1，P4.0/AIN0 引脚就是外部 VREFH 的输入引脚，P4.0 的 I/O 功能和 AIN0 功能被隔离。此时，该引脚处于悬浮状态。

\* 注：当选择单片机内部 4V/3V/2V 参考电压时，ADC 的分辨率为 8 位。  
当选择单片机内部 VDD 和外部参考源时，ADC 的分辨率位 12 位。

## 9.7 AD 转换时间

$$12 \text{ 位 AD 转换时间} = 1/(\text{ADC clock}/4) * 16 \text{ sec}$$

高速时钟 (Fosc) = 4MHz

Fcpu	ADCKS1	ADCKS0	ADC 时钟	AD 转换时间	
Fosc/1	0	0	Fcpu/16	$1/((4\text{MHz}/1)/16/4)$	x16= 256 us
	0	1	Fcpu/8	$1/((4\text{MHz}/1)/8/4)$	x16= 128 us
	1	0	Fcpu	$1/((4\text{MHz}/1)/1/4)$	x16= 16 us
	1	1	Fcpu/2	$1/((4\text{MHz}/1)/2/4)$	x16= 32 us
Fosc/2	0	0	Fcpu/16	$1/((4\text{MHz}/2)/16/4)$	x16= 512 us
	0	1	Fcpu/8	$1/((4\text{MHz}/2)/8/4)$	x16= 256 us
	1	0	Fcpu	$1/((4\text{MHz}/2)/1/4)$	x16= 32 us
	1	1	Fcpu/2	$1/((4\text{MHz}/2)/2/4)$	x16= 64 us
Fosc/4	0	0	Fcpu/16	$1/((4\text{MHz}/4)/16/4)$	x16= 1024 us
	0	1	Fcpu/8	$1/((4\text{MHz}/4)/8/4)$	x16= 512 us
	1	0	Fcpu	$1/((4\text{MHz}/4)/1/4)$	x16= 64 us
	1	1	Fcpu/2	$1/((4\text{MHz}/4)/2/4)$	x16= 128 us
Fosc/8	0	0	Fcpu/16	$1/((4\text{MHz}/8)/16/4)$	x16= 2048 us
	0	1	Fcpu/8	$1/((4\text{MHz}/8)/8/4)$	x16= 1024 us
	1	0	Fcpu	$1/((4\text{MHz}/8)/1/4)$	x16= 128 us
	1	1	Fcpu/2	$1/((4\text{MHz}/8)/2/4)$	x16= 256 us
Fosc/16	0	0	Fcpu/16	$1/((4\text{MHz}/16)/16/4)$	x16= 4096 us
	0	1	Fcpu/8	$1/((4\text{MHz}/16)/8/4)$	x16= 2048 us
	1	0	Fcpu	$1/((4\text{MHz}/16)/1/4)$	x16= 256 us
	1	1	Fcpu/2	$1/((4\text{MHz}/16)/2/4)$	x16= 512 us

## 9.8 ADC 操作实例

➤ 例：设置 AIN0 为 ADC 的输入并执行 12 位 ADC，VREFH 为内部 3.0V，ADC 时钟源为 Fcpu。

；允许 ADC 功能，延迟 100us 等待转换。

ADC0:

```
B0BSET      FADENB      ; 使能 ADC 电路。
CALL        Delay100uS ; 延迟 100us 等待 ADC 电路开始转换。
```

；设置 P4 I/O 模式。

```
MOV         A, #0FEH
B0MOV      P4UR, A      ; 禁止 P4.0 上拉电阻。
B0BCLR     FP40M       ; 设置 P4.0 为输入模式。
```

；或

```
MOV         A, #01H
B0MOV      P4CON, A     ; 设置 P4.0 为单纯的模拟输入。
```

；设置 VREFH 为内部 3.0V。

```
MOV         A, #01H
B0MOV      VREFH, A    ; 设置内部 3.0V VREFH。
```

；设置 ADC 时钟源为 Fcpu。

```
MOV         A, #40H
B0MOV      ADM, A      ; 设置 ADC 时钟源为 Fcpu。
```

；使能 AIN0 (P4.0)。

```
MOV         A, #90H
B0MOV      ADM, A      ; 允许 ADC 并设置 AIN0 输入。
```

；开始 AD 转换。

```
B0BSET      FADS      ; 开始转换。
```

WADC0:

```
B0BTS1     FEOC      ;
JMP        WADC0     ;
B0MOV      A, ADB     ;
B0MOV      Adc_Buf_Hi, A
B0MOV      A, ADR     ;
AND        A, 0FH
B0MOV      Adc_Buf_Low, A
```

End\_ADC:

```
B0BCLR     FADENB    ; 禁止 AD 转换。
```

➤ 例：设置 AIN1 为 ADC 输入并执行 12 位 ADC。VREFH 来自 VREFH 引脚（P4.0/AIN0）的外部输入电源，ADC 的时钟源为 Fcpu。使用 ADC 中断。

; 允许 ADC 功能，延迟 100us 等待 AD 转换。

ADC0:

```
B0BSET      FADENB      ; 使能 ADC 电路。
CALL        Delay100uS ; 延迟 100us 等待 ADC 电路开始转换。
```

; 设置 P4 I/O 模式。

```
MOV         A, #0FDH
B0MOV      P4UR, A      ; 禁止 P4.1 的上拉电阻。
B0BCLR     FP41M        ; 设置 P4.1 为输入模式。
```

; 或

```
MOV         A, #02H
B0MOV      P4CON, A     ; 设置 P4.1 为单纯的模拟输入。
```

; 设置 VREFH 为外部输入电压。

```
B0BSET     FEVHENB     ; 允许外部 VREFH 输入。
```

; 设置 ADC 时钟源为 Fcpu。

```
MOV         A, #40H
B0MOV      ADR, A      ; 设置 ADC 时钟源为 Fcpu。
```

; 允许 AIN0（P4.1）。

```
MOV         A, #91H
B0MOV      ADM, A      ; 允许 ADC 和 AIN1 输入。
```

; 设置 ADC 中断。

```
B0BCLR     FADCIRQ     ; 清除 ADC 中断请求标志。
B0BSET     FADCIE      ; 使能 ADC 中断功能。
B0BSET     FGIE        ; 使能 GIE 功能。
```

; 开始 AD 转换。

```
B0BSET     FADS        ; 开始转换。
...
...
...
```

ADC\_INT\_SR:

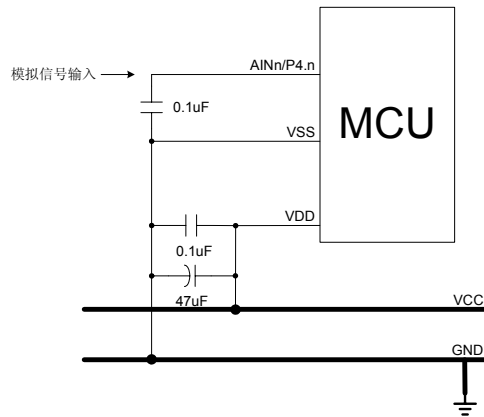
```
PUSH

B0BTS1     FADCIRQ     ; 检查是否有 ADC 中断标志。
JMP        ADC_INT_EXIT
B0BCLR     FADCIRQ     ; 清除 ADC 中断请求标志。
B0MOV      A, ADB      ;
B0MOV      Adc_Buf_Hi, A
B0MOV      A, ADR      ;
AND        A, 0FH
B0MOV      Adc_Buf_Low, A
B0BCLR     FADENB      ; 禁止 ADC 电路。
```

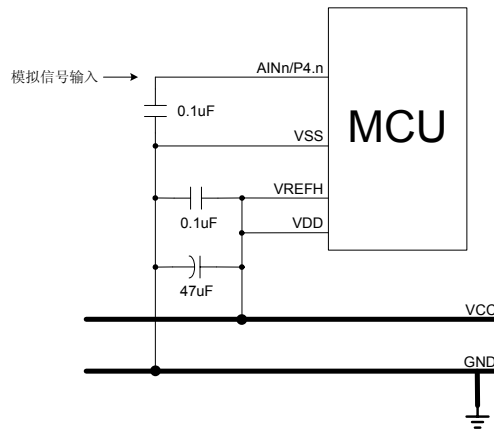
ADC\_INT\_EXIT:

```
POP
RETI
```

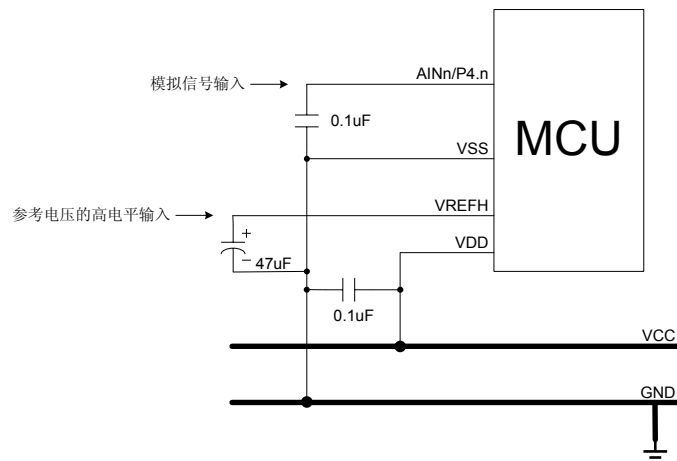
## 9.9 ADC 电路



ADC 参考电压为内部参考电压，VREFH 引脚是 AIN0/P4.0。AINn/P4.n 和 VSS 之间的电容（0.1uF）有助于模拟信号的稳定。



ADC 参考电压来自 VDD，AIN0/P4.0 为 VREFH 输入引脚。VREFH 应该来自单片机的 VDD，而不是其它 VDD。



ADC 参考电压来自外部电压，AIN0/P4.0 是 VREFH 的输入引脚。VREFH 和 VSS 之间的电容（47uF）有助于 VREFH 电压的稳定。



## 10 指令集

指令	指令格式	描述	C	DC	Z	周期	
MOV	A,M	$A \leftarrow M$ 。	-	-	√	1	
	M,A	$M \leftarrow A$ 。	-	-	-	1	
	B0MOV	A,M	$A \leftarrow M$ (bank 0)。	-	-	√	1
	B0MOV	M,A	$M$ (bank 0) $\leftarrow A$ 。	-	-	-	1
	A,I	$A \leftarrow I$	-	-	-	1	
	M,I	$M \leftarrow I$ 。(M 仅适用于系统寄存器 R、Y、Z、RBANK、PFLAG。)	-	-	-	1	
	A,M	$A \leftrightarrow M$ 。	-	-	-	1+N	
	A,M	$A \leftrightarrow M$ (bank 0)。	-	-	-	1+N	
		R, $A \leftarrow ROM$ [Y,Z]。	-	-	-	2	
ADC	A,M	$A \leftarrow A + M + C$ ，如果产生进位则 C=1，否则 C=0。	√	√	√	1	
	M,A	$M \leftarrow A + M + C$ ，如果产生进位则 C=1，否则 C=0。	√	√	√	1+N	
	A,M	$A \leftarrow A + M$ ，如果产生进位则 C=1，否则 C=0。	√	√	√	1	
	M,A	$M \leftarrow A + M$ ，如果产生进位则 C=1，否则 C=0。	√	√	√	1+N	
	M,A	$M$ (bank 0) $\leftarrow M$ (bank 0) + A，如果产生进位则 C=1，否则 C=0。	√	√	√	1+N	
	A,I	$A \leftarrow A + I$ ，如果产生进位则 C=1，否则 C=0。	√	√	√	1	
	A,M	$A \leftarrow A - M / C$ ，如果产生借位则 C=0，否则 C=1。	√	√	√	1	
	M,A	$M \leftarrow A - M / C$ ，如果产生借位则 C=0，否则 C=1。	√	√	√	1+N	
	A,M	$A \leftarrow A - M$ ，如果产生借位则 C=0，否则 C=1。	√	√	√	1	
	M,A	$M \leftarrow A - M$ ，如果产生借位则 C=0，否则 C=1。	√	√	√	1+N	
	A,I	$A \leftarrow A - I$ ，如果产生借位则 C=0，否则 C=1。	√	√	√	1	
AND	A,M	$A \leftarrow A$ 与 $M$ 。	-	-	√	1	
	M,A	$M \leftarrow A$ 与 $M$ 。	-	-	√	1+N	
	A,I	$A \leftarrow A$ 与 $I$ 。	-	-	√	1	
	A,M	$A \leftarrow A$ 或 $M$ 。	-	-	√	1	
	M,A	$M \leftarrow A$ 或 $M$ 。	-	-	√	1+N	
	A,I	$A \leftarrow A$ 或 $I$ 。	-	-	√	1	
	A,M	$A \leftarrow A$ 异或 $M$ 。	-	-	√	1	
	M,A	$M \leftarrow A$ 异或 $M$ 。	-	-	√	1+N	
	A,I	$A \leftarrow A$ 异或 $I$ 。	-	-	√	1	
SWAP	M	$A$ (b3~b0, b7~b4) $\leftarrow M$ (b7~b4, b3~b0)。	-	-	-	1	
	M	$M$ (b3~b0, b7~b4) $\leftarrow M$ (b7~b4, b3~b0)。	-	-	-	1+N	
	M	$A \leftarrow M$ 带进位右移。	√	-	-	1	
	M	$M \leftarrow M$ 带进位右移。	√	-	-	1+N	
	M	$A \leftarrow M$ 带进位左移。	√	-	-	1	
	M	$M \leftarrow M$ 带进位左移。	√	-	-	1+N	
	M	$M \leftarrow 0$ 。	-	-	-	1	
	M.b	$M.b \leftarrow 0$ 。	-	-	-	1+N	
	M.b	$M.b \leftarrow 1$	-	-	-	1+N	
	M.b	$M$ (bank 0).b $\leftarrow 0$ 。	-	-	-	1+N	
M.b	$M$ (bank 0).b $\leftarrow 1$ 。	-	-	-	1+N		
CMPRS	A,I	比较，如果相等则跳过下一条指令 C 与 ZF 标志位可能受影响。	√	-	√	1+S	
	A,M	比较，如果相等则跳过下一条指令 C 与 ZF 标志位可能受影响。	√	-	√	1+S	
	M	$A \leftarrow M + 1$ ，如果 $A = 0$ ，则跳过下一条指令。	-	-	-	1+S	
	M	$M \leftarrow M + 1$ ，如果 $M = 0$ ，则跳过下一条指令。	-	-	-	1+N+S	
	M	$A \leftarrow M - 1$ ，如果 $A = 0$ ，则跳过下一条指令。	-	-	-	1+S	
	M	$M \leftarrow M - 1$ ，如果 $M = 0$ ，则跳过下一条指令。	-	-	-	1+N+S	
	M.b	如果 $M.b = 0$ ，则跳过下一条指令。	-	-	-	1+S	
	M.b	如果 $M.b = 1$ ，则跳过下一条指令。	-	-	-	1+S	
	M.b	如果 $M$ (bank 0).b = 0，则跳过下一条指令。	-	-	-	1+S	
	M.b	如果 $M$ (bank 0).b = 1，则跳过下一条指令。	-	-	-	1+S	
	d	跳转指令， $PC_{15/14} \leftarrow RomPages_{1/0}$ ， $PC_{13} \sim PC_0 \leftarrow d$ 。	-	-	-	2	
	d	子程序调用指令， $Stack \leftarrow PC_{15} \sim PC_0$ ， $PC_{15/14} \leftarrow RomPages_{1/0}$ ， $PC_{13} \sim PC_0 \leftarrow d$ 。	-	-	-	2	
		子程序跳出指令， $PC \leftarrow Stack$ 。	-	-	-	2	
		中断处理程序跳出指令， $PC \leftarrow Stack$ ，使能全局中断控制位。	-	-	-	2	
		进栈指令，保存 ACC 和工作寄存器。	-	-	-	1	
		出栈指令，恢复 ACC 和工作寄存器。	√	√	√	1	
		空指令，无特别意义。	-	-	-	1	

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

# 11 电气特性

## 11.1 极限参数

Supply voltage (Vdd).....	- 0.3V ~ 6.0V
Input in voltage (Vin).....	Vss - 0.2V ~ Vdd + 0.2V
Operating ambient temperature (Topr)	
SN8P2711AP, SN8P2711AS, SN8P2711AX .....	0°C ~ + 70°C
SN8P2711APD, SN8P2711ASD, SN8P2711AXD .....	-40°C ~ + 85°C
Storage ambient temperature (Tstor) .....	-40°C ~ + 125°C

## 11.2 电气特性

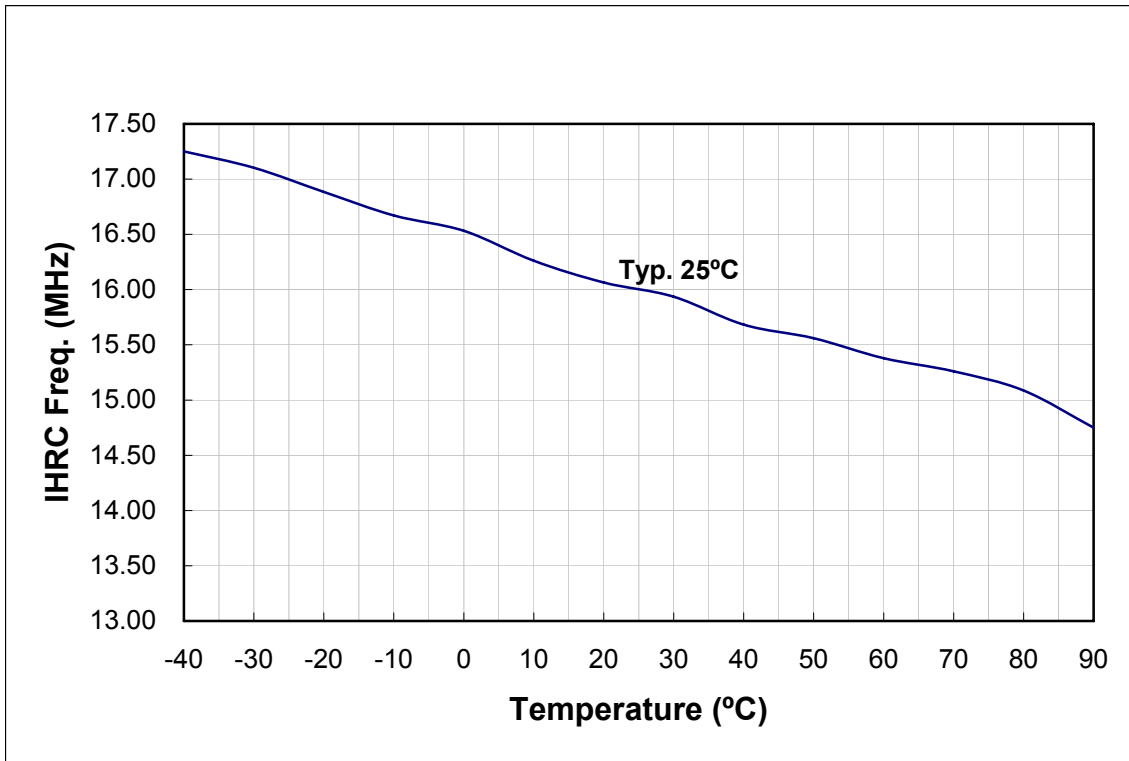
(All of voltages refer to Vss, Vdd = 5.0V, fosc = 4MHz, fcpu=1MHz, ambient temperature is 25°C unless otherwise note.)

PARAMETER	SYM.	DESCRIPTION	MIN.	TYP.	MAX.	UNIT	
Operating voltage	Vdd	Normal mode, Vpp = Vdd, 25°C	2.4	5.0	5.5	V	
		Normal mode, Vpp = Vdd, -40°C~85°C	2.5	5.0	5.5	V	
RAM Data Retention voltage	Vdr		1.5	-	-	V	
Vdd rise rate	Vpor	Vdd rise rate to ensure internal power-on reset	0.05	-	-	V/ms	
Input Low Voltage	ViL1	All input ports	Vss	-	0.3Vdd	V	
	ViL2	Reset pin	Vss	-	0.2Vdd	V	
	ViL3	P4 ADC shared pin.	Vss	0.5Vdd	-	V	
Input High Voltage	ViH1	All input ports	0.7Vdd	-	Vdd	V	
	ViH2	Reset pin	0.9Vdd	-	Vdd	V	
	ViH3	P4 ADC shared pin.	-	0.5Vdd	Vdd	V	
Reset pin leakage current	Ilekg	Vin = Vdd	-	-	2	uA	
I/O port pull-up resistor	Rup	Vin = Vss, Vdd = 3V	100	200	300	KΩ	
		Vin = Vss, Vdd = 5V	50	100	150		
I/O port input leakage current	Ilekg	Pull-up resistor disable, Vin = Vdd	-	-	2	uA	
I/O output source current	IoH	Vop = Vdd - 0.5V	8	12	-	mA	
	IoL	Vop = Vss + 0.5V	8	15	-		
INTn trigger pulse width	Tint0	INT0 interrupt request pulse width	2/fcpu	-	-	cycle	
Supply Current (Disable ADC)	Idd1	Run Mode (No loading, Fcpu = Fosc/4)	Vdd= 5V, 4Mhz	-	2.5	5	mA
			Vdd= 3V, 4Mhz	-	1	2	mA
	Idd2	Slow Mode (Internal low RC, Stop high clock)	Vdd= 5V, 32Khz	-	20	40	uA
			Vdd= 3V, 16Khz	-	5	10	uA
	Idd3	Sleep Mode	Vdd= 5V, 25°C	-	0.8	1.6	uA
			Vdd= 3V, 25°C	-	0.7	1.4	uA
			Vdd= 5V, -40°C~ 85°C	-	10	21	uA
			Vdd= 3V, -40°C~ 85°C	-	10	21	uA
	Idd4	Green Mode (No loading, Fcpu = Fosc/4 Watchdog Disable)	Vdd= 5V, 4Mhz	-	0.6	1.2	mA
			Vdd= 3V, 4Mhz	-	0.25	0.5	mA
Vdd=5V, ILRC 32Khz			-	15	30	uA	
		Vdd=3V, ILRC 16Khz	-	3	6	uA	
Internal High Oscillator Freq.	Fihrc	Internal High RC (IHRC)	25°C, Vdd= 5V, Fcpu = 1MHz	15.68	16	16.32	Mhz
			-40°C~85°C, Vdd= 2.4V~5.5V, Fcpu = 1MHz~16 MHz	13	16	19	Mhz
LVD Voltage	Vdet0	Low voltage reset level.	1.7	2.0	2.3	V	
	Vdet1	Low voltage reset level. Fcpu = 1 MHz. Low voltage indicator level. Fcpu = 1 MHz.	2.0	2.3	3	V	
	Vdet2	Low voltage indicator level. Fcpu = 1 MHz	2.9	3.4	4.5	V	
VREFH input voltage	Vrefh1	External reference voltage, Vdd = 5.0V.	2V	-	Vdd	V	
	Vrefh2	Internal VDD reference voltage, Vdd = 5V.	-	Vdd*	-	V	
	Vrefh3	Internal 4V reference voltage, Vdd = 5V.	-	4*	-	V	
	Vrefh4	Internal 3V reference voltage, Vdd = 5V.	-	3*	-	V	
	Vrefh5	Internal 2V reference voltage, Vdd = 5V.	-	2*	-	V	
AIN0 ~ AIN5 input voltage	Vani	Vdd = 5.0V	0	-	Vrefh1~5	V	
ADC enable time	Tast	Ready to start convert after set ADENB = "1"	100	-	-	us	
ADC current consumption	IADC	Vdd=5.0V	-	0.6	-	mA	
		Vdd=3.0V	-	0.4	-	mA	
ADC Clock Frequency	FADCLK	VDD=5.0V	-	-	8M	Hz	
		VDD=3.0V	-	-	5M	Hz	
ADC Conversion Cycle Time	FADCYL	VDD=2.4V~5.5V	64	-	-	1/FADCLK	
ADC Sampling Rate (Set FADS=1 Frequency)	FADSMP	VDD=5.0V	-	-	125	K/sec	
		VDD=3.0V	-	-	80	K/sec	
Differential Nonlinearity	DNL	VDD=5.0V, AVREFH=3.2V, FADSMP =7.8K	±1	±2	±16	LSB	
Integral Nonlinearity	INL	VDD=5.0V, AVREFH=3.2V, FADSMP =7.8K	±2	±4	±16	LSB	
No Missing Code	NMC	VDD=5.0V, AVREFH=3.2V, FADSMP =7.8K	8	10	12	Bits	

\*These parameters are for design reference, not tested.

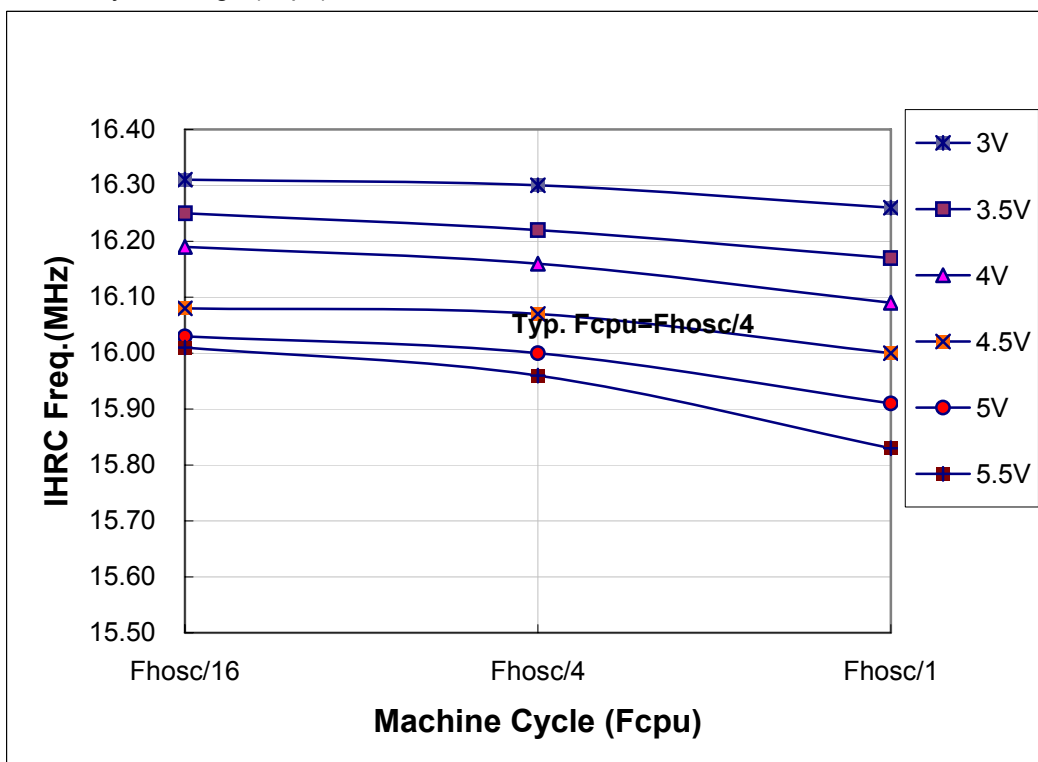
➤ **Internal 16MHz Oscillator RC Type Temperature Characteristic.**

Power Voltage (VDD) = 5V.  
Machine Cycle (Fcpu) = Fhosc/4.  
Typical Temperature = 25°C.  
Typical Internal 16MHz Oscillator RC Type Frequency = 16MHz.  
Testing Temperature Range = -40°C ~ + 90°C



➤ **Internal 16MHz Oscillator RC Type Power Voltage and Machine Cycle Characteristic.**

Temperature = 25°C.  
Typical Power Voltage (VDD) = 5V.  
Typical Machin Cycle (Fcpu) = Fhosc / 4.  
Typical Internal 16MHz Oscillator RC Type Frequency = 16MHz.  
Testing Power Voltage Range (VDD) = 3V~5.5V.  
Testing Machine Cycle Range (Fcpu) = Fhosc/1~Fhosc/16.



# 12 开发工具

## 12.1 在线仿真器 (ICE)

- **SN8ICE 2K ICE:** 完全仿真 SN8P2711A 的所有功能。

- \* 注:
1. ICE 工作电压: 3.0V ~ 5.0V;
  2. 5V 时最高仿真速度: 8 MIPS (如, 16Mhz 晶振,  $F_{cpu} = F_{osc}/2$ );
  3. 内部 16M RC 振荡器的精度低于芯片的精度;
  4. 建议采用 SN8P2711 EV-KIT 仿真 LVD 和 ADC 参考电压;

- \* 注: S8KD-2 ICE 不能对 SN8P2711 进行仿真。

## 12.2 OTP 烧录器

- **MP WriterIII:** 支持 SN8P2711 的联机烧录, 也支持大批量的脱机烧录。

## 12.3 SN8IDE

SONiX 8 位单片机的集成开发环境包括编译器、ICE 调试器和 OTP 的烧录软件。

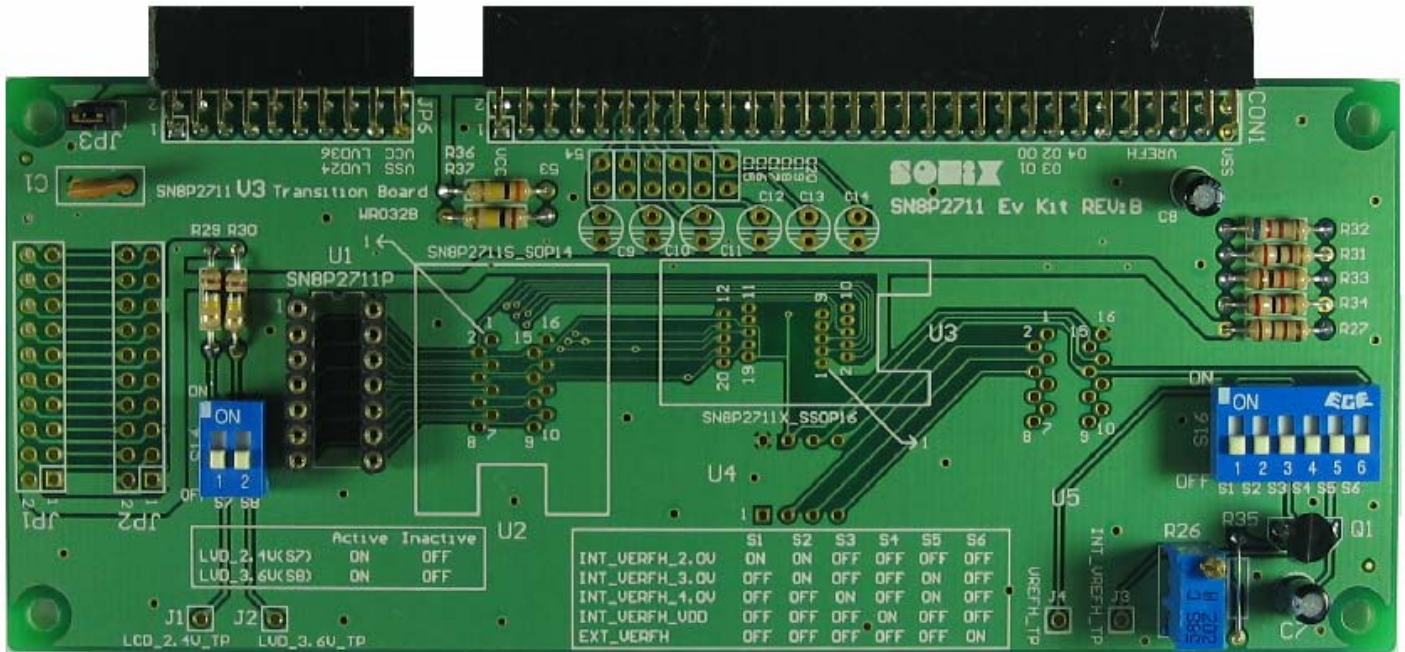
- **SN8ICE 2K:** M2IDE\_V115 及更新版本。
- **MP WriterIII:** M2IDE\_V115 或更新的版本。

- \* 注: SN8IDE (SN8IED\_1.99R...) 和 SN8WTxxx 不能仿真 SN8P2711。

## 12.4 SN8P2711 EV KIT

### 12.4.1 PCB 说明

SONiX 提供 SN8P2711 的 EV-kit Ver.A, 能够对 SN8P2711 的功能进行仿真, 对于 ICE 仿真, EV kit 提供 ADC 内部参考电压和 LVD2.4V/3.6V 选择电路。

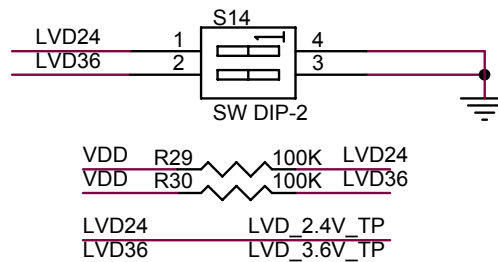


**CON1:** I/O 端口和 ADC 参考输入, 与 SN8ICE 2K CON1 相连;

**JP6:** LVD 2.4V、3.6V 输入引脚, 与 SN8ICE 2K JP6 相连;

**S14:** LVD 2.4V/3.6V 控制开关, 仿真 LVD 2.4V 标志/复位功能和 LVD3.6V 标志功能;

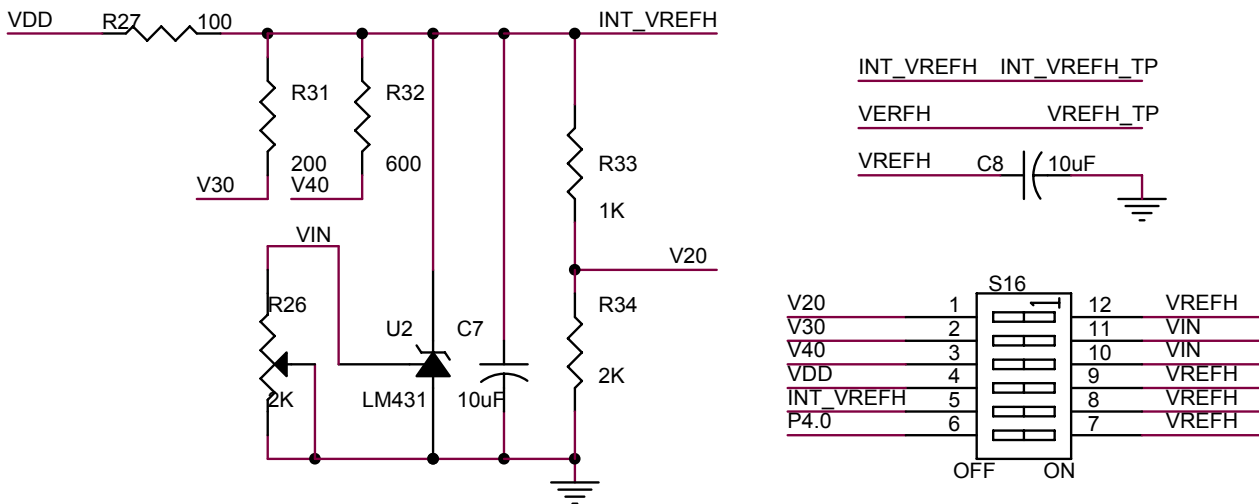
开关编号	On	Off
S7	LVD 2.4V 开	LVD 2.4V 关
S8	LVD 3.6V 开	LVD 3.6V 关



**S16:** ADC 参考电压选择开关, 此参考电压与 CON1 引脚的 VREFH 相连。最大参考值为 VDD, 当  $VDD < INT\_VREFH\_4.0V$  的时候, ADC 参考电压就等于 VDD。EXT\_VREFH 可选外部参考电压, 由 P4.0 口输入, 选择内部参考电压时, P4.0 作为普通 I/O 引脚或 ADC 输入引脚。

开关编号	S1	S2	S3	S4	S5	S6
INT_VREFH_2.0V	ON	ON	OFF	OFF	OFF	OFF
INT_VREFH_3.0V	OFF	ON	OFF	OFF	ON	OFF
INT_VREFH_4.0V	OFF	OFF	ON	OFF	ON	OFF
INT_VREFH_VDD	OFF	OFF	OFF	ON	OFF	OFF
EXT_VREFH	OFF	OFF	OFF	OFF	OFF	ON

**R26:** 2K 欧姆可变电阻, 用来调节 ADC 内部参考电压。若 S16 选为 INT\_VREFH\_4.0V, 那么输入电压  $VDD=5V$ , 通过 J3 测量内部参考电压, 调节 R26 使得 J3 电压为 4.0V。



**R36, P37:**  $R36=300K$  欧姆,  $R37=100K$  欧姆, 偏压等于  $1/4VDD$ 。通过 ADC 通道 5 仿真低电压检测功能。

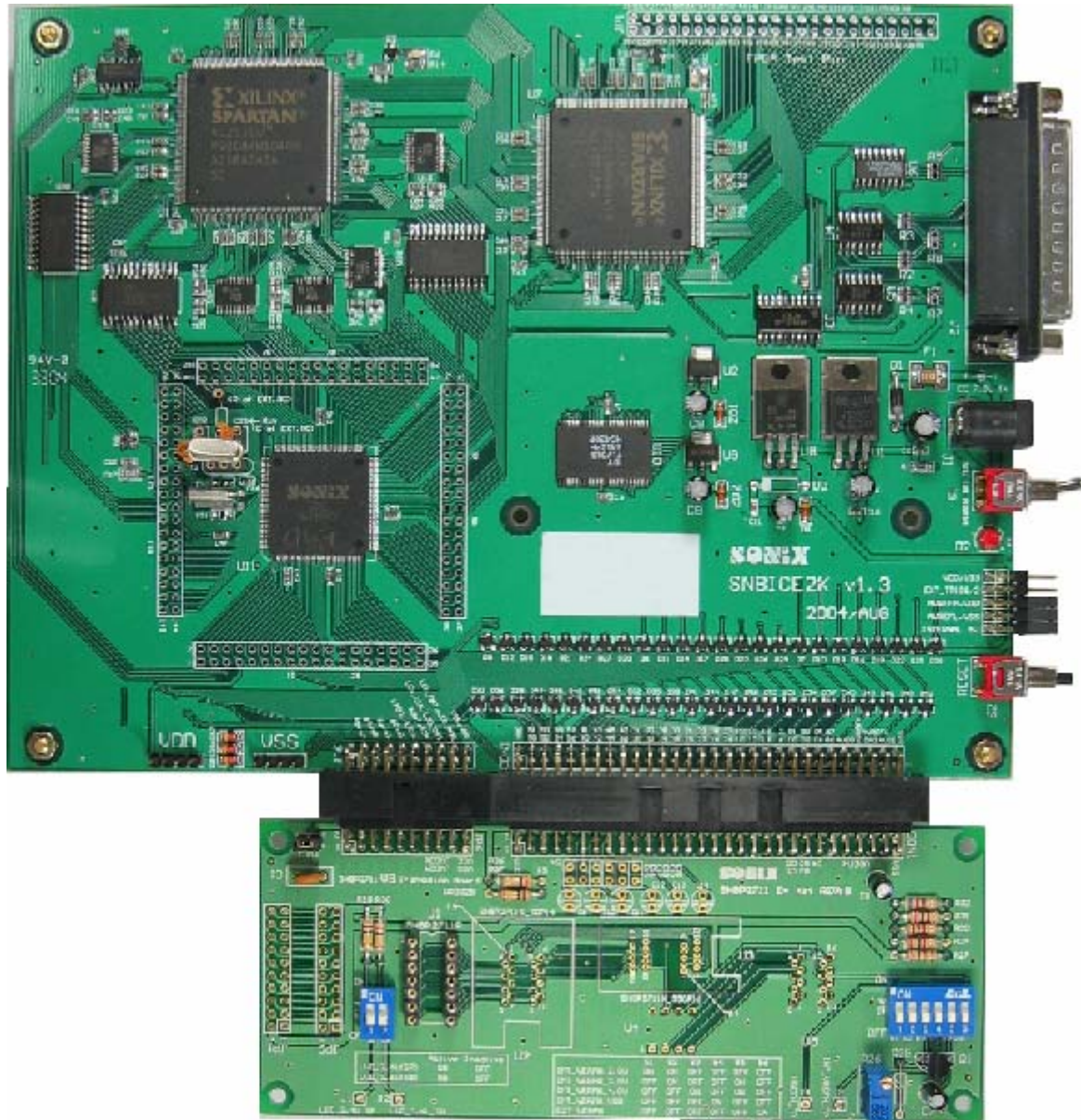
**C9~C14:** 47uF 电容, 连接至 ADC 的 0~5 通道旁路电容, 即 AIN0~AIN5 输入口。

**C15~C20:** 0.1uF 电容, 连接至 ADC 的 0~5 通道旁路电容, 即 AIN0~AIN5 输入口。



## 12.4.2 SN8P2711 EV KIT 与 SN8ICE 2K 的连接

SN8P2711 EV KIT 与 SN8ICE 2K 之间的具体连接如下图所示。ADC 参考电压由 SN8P2711 EV KIT 提供。SN8ICE 2K 的 AVREFH/VDD 跳线必须断开。



## 12.5 OTP 烧录信息

### 12.5.1 烧录转接板信息

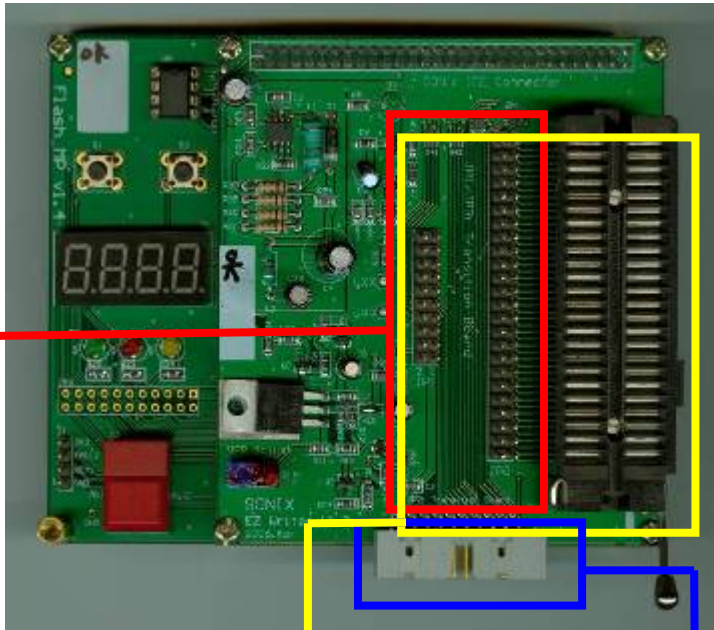


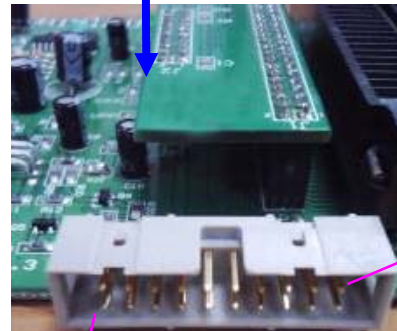
图 1 MPIII Writer 的内部结构



Writer 上板 JP1/JP3



Writer 上板 JP1/JP3



Pin 1 (Down)

Pin 20 (UP)

Writer 上板 JP2

注 1: JP1 连接 MP 烧录转接板, JP3 连接 OTP MCU。

注 2: JP2 连接外部烧录转接板。当 OTP MCU 的 PIN 超过 48PIN, 或者烧录 Dice MCU 时, 请采用外部烧录转接板, 连接到 JP2 进行烧录。

下面两个图演示了如何焊接烧录转接板。

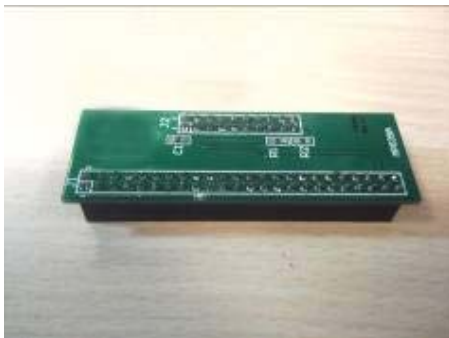


图 2

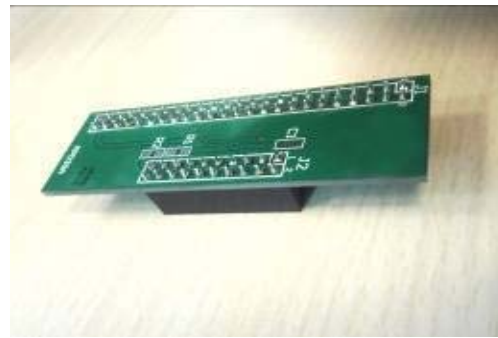


图 3



注:

- 1、印有 IC 型号的这一面为转接板的正面。
- 2、180 度的母座必须焊接在 MP 转接板的背面。请参考图 2 和图 3。

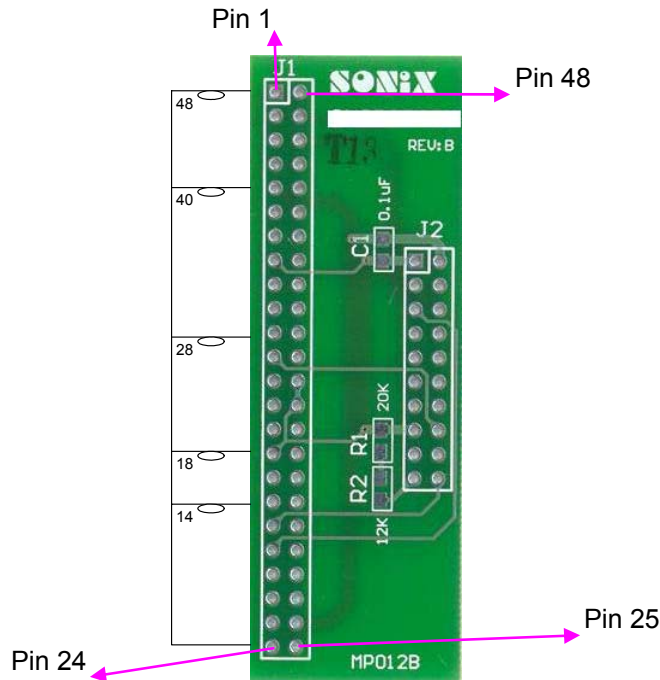


图 4 MP 转接板 (连接到 JP1&JP3)

**JP3 (连接 48-pin text tool)**

DIP 1	1	48	DIP48
DIP 2	2	47	DIP47
DIP 3	3	46	DIP46
DIP 4	4	45	DIP45
DIP 5	5	44	DIP44
DIP 6	6	43	DIP43
DIP 7	7	42	DIP42
DIP 8	8	41	DIP41
DIP 9	9	40	DIP40
DIP10	10	39	DIP39
DIP11	11	38	DIP38
DIP12	12	37	DIP37
DIP13	13	36	DIP36
DIP14	14	35	DIP35
DIP15	15	34	DIP34
DIP16	16	33	DIP33
DIP17	17	32	DIP32
DIP18	18	31	DIP31
DIP19	19	30	DIP30
DIP20	20	29	DIP29
DIP21	21	28	DIP28
DIP22	22	27	DIP27
DIP23	23	26	DIP26
DIP24	24	25	DIP25

**JP1/JP2**

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

JP1 连接 MP 烧录转接板

JP2 连接外部转接板

## 12.5.2 烧录引脚信息

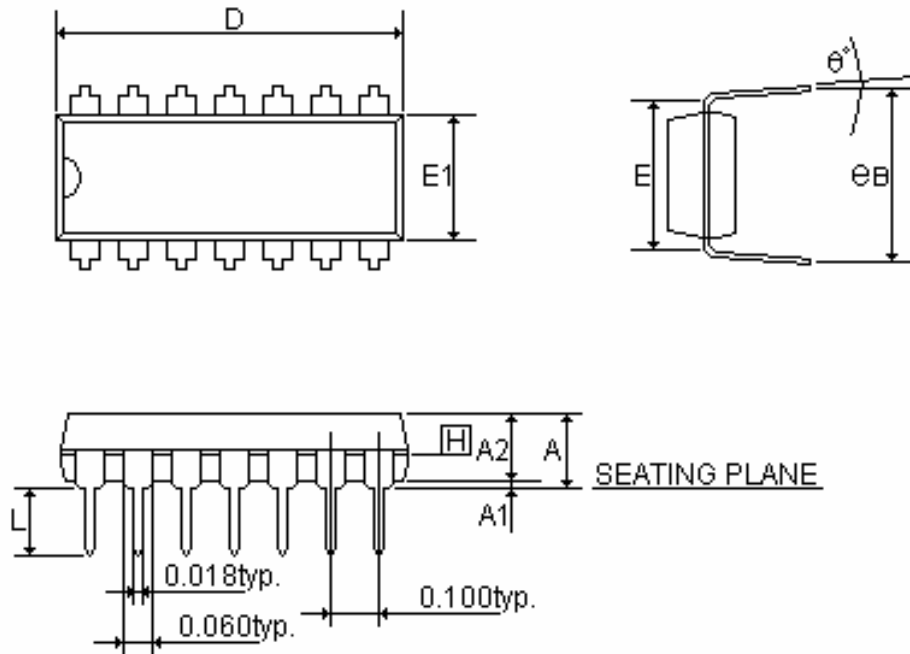
SN8P2711A 系列烧录信息							
单片机名称		SN8P2711AP,S			SN8P2711AX		
MPIO Writer		OTP IC / JP3 引脚配置					
JP1/JP2 Pin Number	JP1/JP2 Pin Name	IC Pin Number	IC Pin Number	JP3 Pin Number	IC Pin Number	IC Pin Number	JP3 Pin Number
1	VDD	1	VDD	18	1	VDD	17
2	GND	14	VSS	31	16	VSS	32
3	CLK	9	P4.0	26	11	P4.0	27
4	CE	-	-	-	-	-	-
5	PGM	13	P4.4	30	15	P4.4	31
6	OE	10	P4.1	27	12	P4.1	28
7	D1	-	-	-	-	-	-
8	D0	-	-	-	-	-	-
9	D3	-	-	-	-	-	-
10	D2	-	-	-	-	-	-
11	D5	-	-	-	-	-	-
12	D4	-	-	-	-	-	-
13	D7	-	-	-	-	-	-
14	D6	-	-	-	-	-	-
15	VDD	1	VDD	18	1	VDD	17
16	VPP	4	RST	21	4	RST	20
17	HLS	-	-	-	-	-	-
18	RST	-	-	-	-	-	-
19	-	-	-	-	-	-	-
20	ALSB/PDB	3	P0.2	20	3	P0.2	19

## \* 注:

1. 采用 M2IDE V1.16 (和更新版本) 仿真;
2. 用 16M Hz 晶振仿真内部 16MHz RC 振荡器;

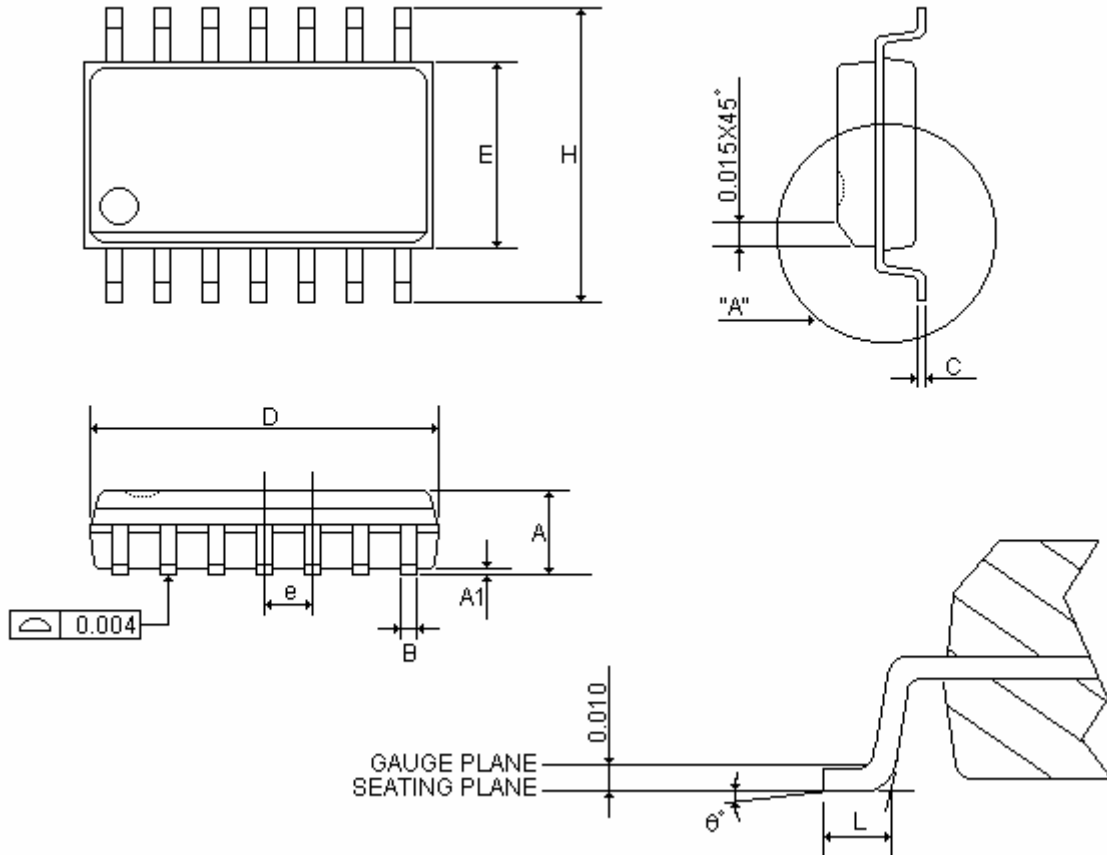
# 13 封装信息

## 13.1 P-DIP 14 PIN



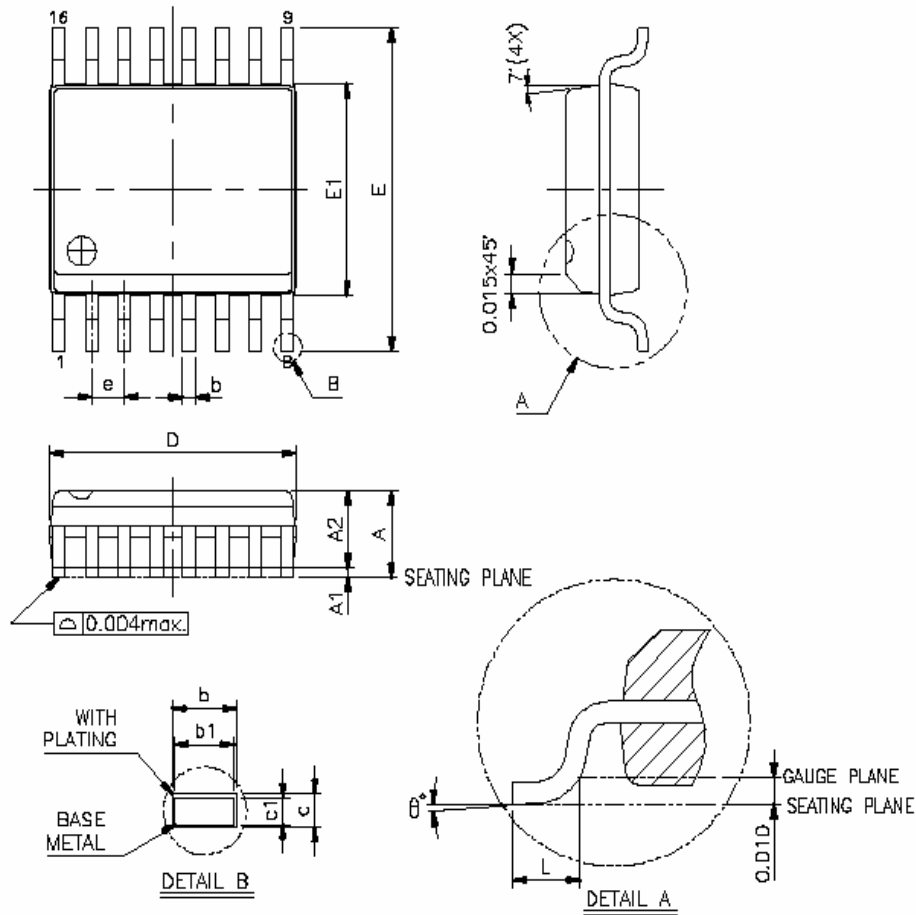
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.210	-	-	5.334
A1	0.015	-	-	0.381	-	-
A2	0.125	0.130	0.135	3.175	3.302	3.429
D	0.735	0.075	0.775	18.669	1.905	19.685
E	0.300			7.62		
E1	0.245	0.250	0.255	6.223	6.35	6.477
L	0.115	0.130	0.150	2.921	3.302	3.810
eB	0.335	0.355	0.375	8.509	9.017	9.525
θ°	0°	7°	15°	0°	7°	15°

### 13.2 SOP 14 PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.058	0.064	0.068	1.4732	1.6256	1.7272
A1	0.004	-	0.010	0.1016	-	0.254
B	0.013	0.016	0.020	0.3302	0.4064	0.508
C	0.0075	0.008	0.0098	0.1905	0.2032	0.2490
D	0.336	0.341	0.344	8.5344	8.6614	8.7376
E	0.150	0.154	0.157	3.81	3.9116	3.9878
e	-	0.050	-	-	1.27	-
H	0.228	0.236	0.244	5.7912	5.9944	6.1976
L	0.015	0.025	0.050	0.381	0.635	1.27
$\theta^\circ$	0°	-	8°	0°	-	8°

### 13.3 SSOP 16 PIN



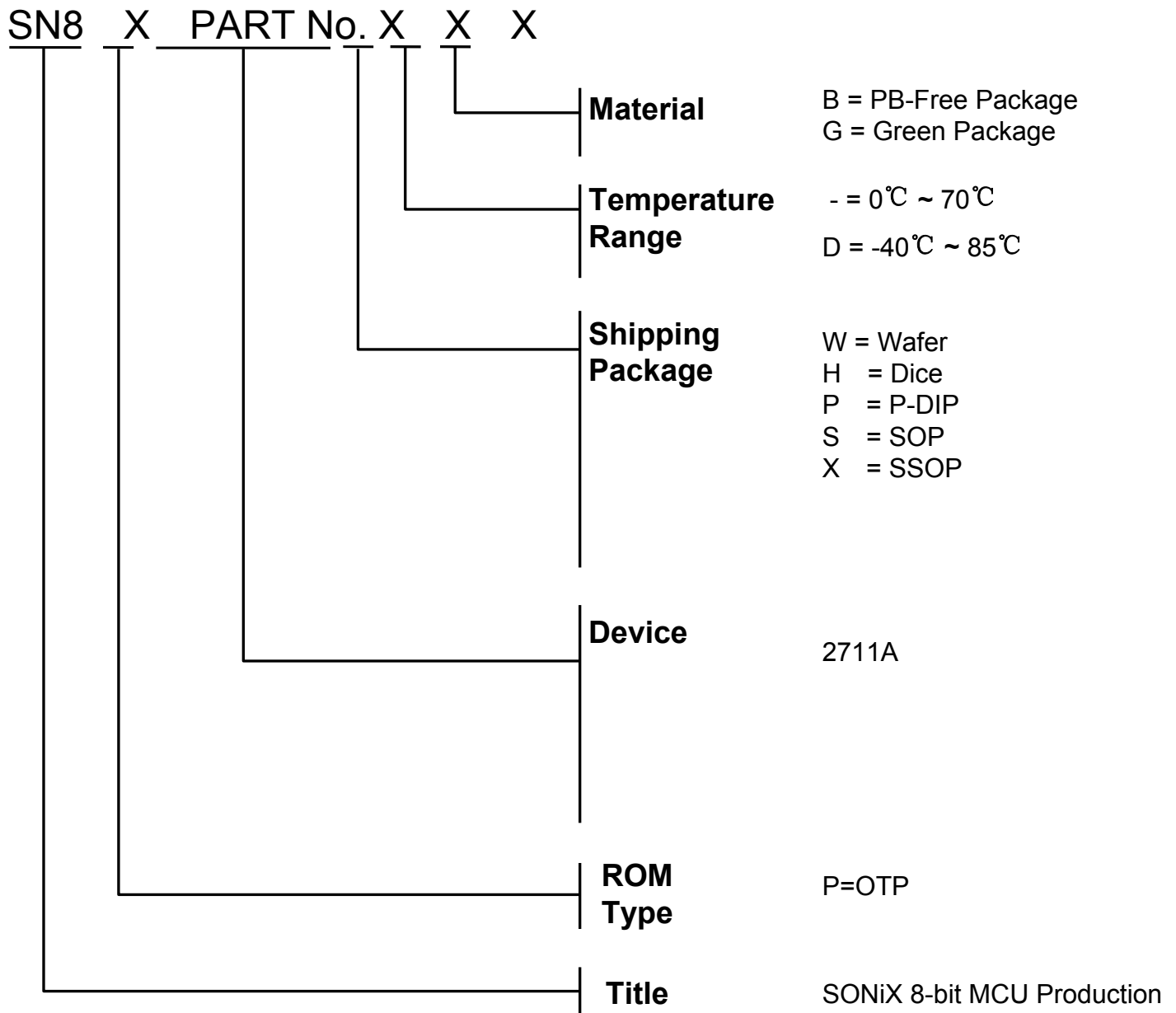
SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.053	-	0.069	1.3462	-	1.7526
A1	0.004	-	0.010	0.1016	-	0.254
A2	-	-	0.059	-	-	1.4986
b	0.008	-	0.012	0.2032	-	0.3048
b1	0.008	-	0.011	0.2032	-	0.2794
c	0.007	-	0.010	0.1778	-	0.254
c1	0.007	-	0.009	0.1778	-	0.2286
D	0.189	-	0.197	4.8006	-	5.0038
E1	0.150	-	0.157	3.81	-	3.9878
E	0.228	-	0.244	5.7912	-	6.1976
L	0.016	-	0.050	0.4064	-	1.27
e	0.025 BASIC			0.635 BASIC		
θ°	0°	-	8°	0°	-	8°

# 14 芯片正印命名规则

## 14.1 概述

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

## 14.2 芯片型号说明





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

**总公司:**

地址：台湾新竹县竹北市台元街 36 号 10 楼之一

电话：886-3-5600-888

传真：886-3-5600-889

**台北办事处:**

地址：台北市松德路 171 号 15 楼之 2

电话：886-2-2759 1980

传真：886-2-2759 8180

**香港办事处:**

地址：香港新界沙田沙田乡宁会路 138 # 新城市中央广场第一座 7 楼 705 室

电话：852-2723 8086

传真：852-2723 9179

**松翰科技（深圳）有限公司**

地址：深圳市南山区高新技术产业园南区 T2-B 栋 2 层

电话：86-755-2671 9666

传真：86-755-2671 9786

**技术支持:**

Sn8fae@SONiX.com.tw