YF2602
8位微控制器

产品说明书

说明书发行履历：

版本	发行时间	新制/修订内容
2016-03-A1	2016-03	新制

1、概 述

YF2602 是一款 8 位高性能精简指令集 OTP-ROM 单片机,具有 1K OTP ROM。其高抗干扰性能在小家电领域有广泛应用。主要特性如下：

功能特性：

- 存储器：
  - ROM：OTP、1K*16位
  - RAM：48字节
- I/O引脚配置

  输入/输出口：P0, P1, P5.

  输入口：P1.5.

  可编程漏极开路端：P1.0.

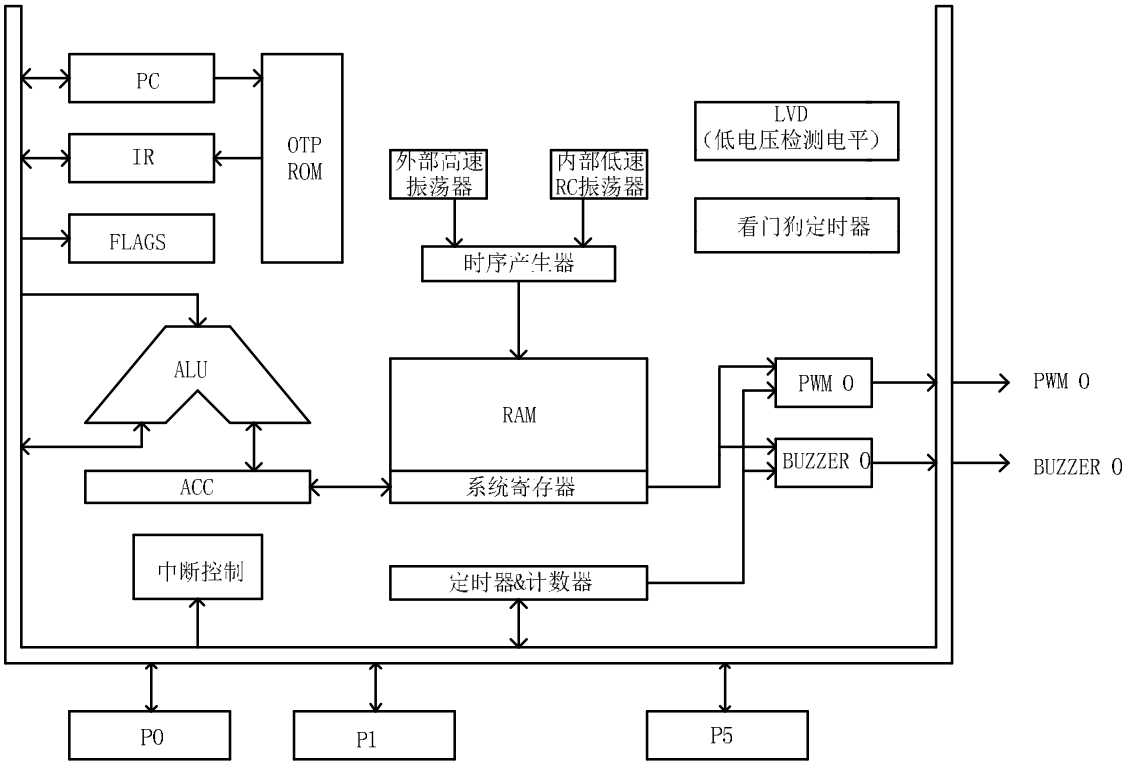
  唤醒功能端：P0, P1 电平触发

  内部上拉电阻：P0, P1, P5.

  外部中断信号沿触发：P0.0（寄存器PEDGE 控制）
- 3级低电压检测系统(LVD)
  - 可监控系统电源(若低于设定LVD则复位)
- 强大的指令系统
  - 单周期指令系统（1T）
  - 大部分指令仅需一个周期
  - 跳转指令JMP可在整个ROM区执行
  - 子程序条用指令CALL可在整个ROM区执行
  - 查表指令MOVC可寻址整个ROM区
- 工作电压：
  - VDD: 2.4V~5.5V
- 双时钟系统：
  - 外部高速时钟：RC模式高达10MHz
  - 外部高速时钟：晶振模式高达16MHz
  - 内部高速模式：16MHz RC（Fcpu=Fosc/4~Fosc/16）
  - 内部低速模式：RC振荡器，16KHz(3V),32KHz(5V)
- 3个中断源：
  - 2个内部中断源：T0、TC0
  - 1个外部中断源：INT0
- 两个8位定时/计数器：
  - TC0：自动装载定时器/计数器/蜂鸣器(BUZZER)输出
  - T0：基本定时/计数器，具有0.5sec实时时钟功能（RTC）
- 内置看门狗定时器，其时钟源由内部低速RC振荡提供（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：有T0周期性的唤醒
- 封装形式
  - SOP18/DIP18

2、功能框图及引脚说明

2.1、功能框图



2.2、引脚排列图

Ⅰ SOP18/DIP18

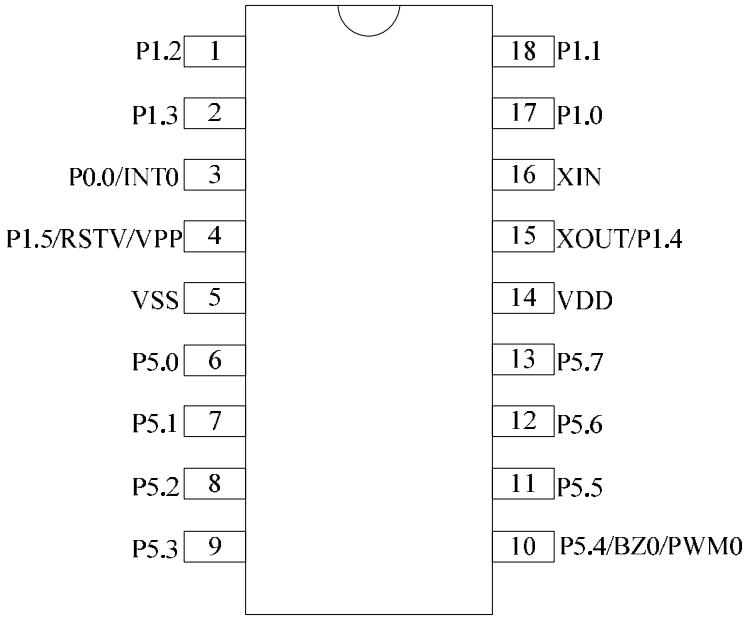


图 1 引脚排列

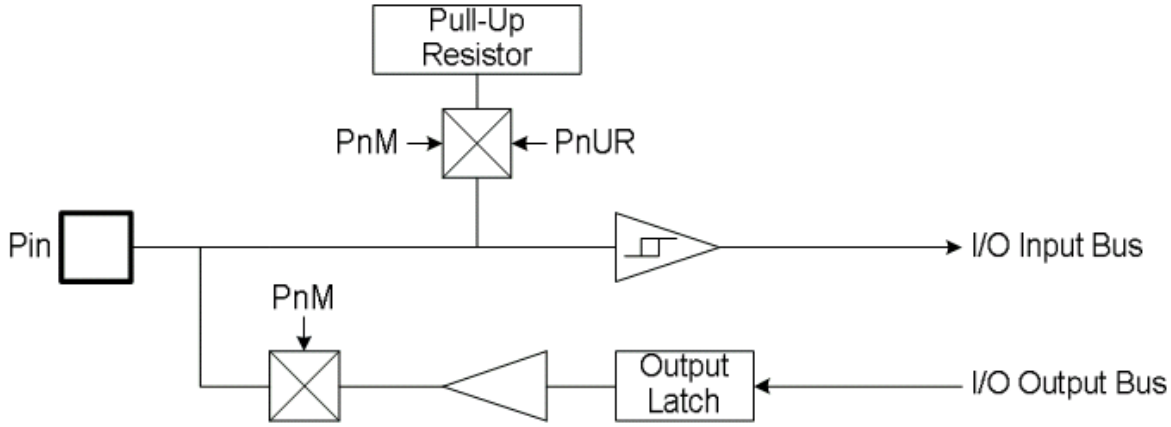
2.3、引脚说明

引脚	引脚名称	类型	说明
1	P1.2	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻

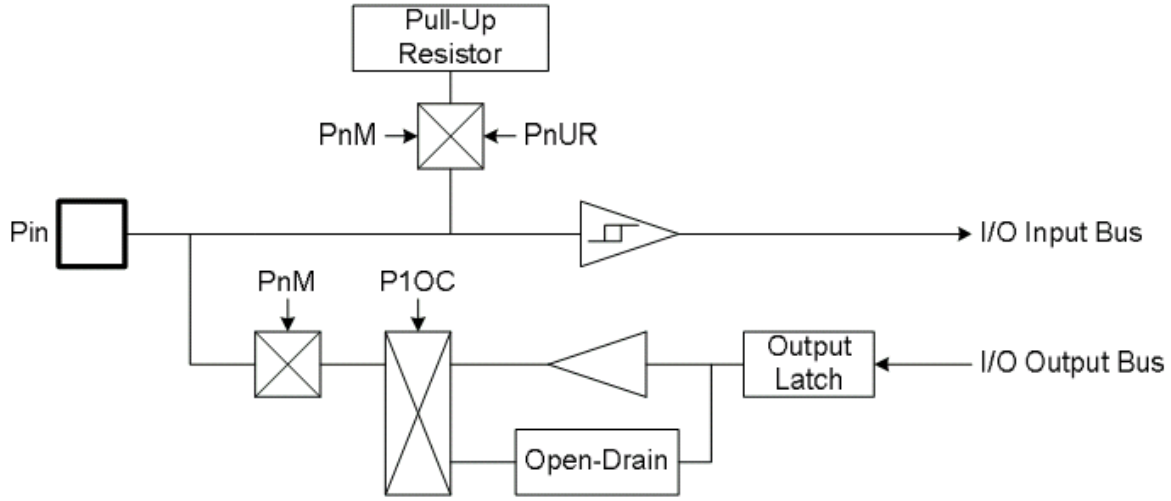
2	P1.3	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻
3	P0.0/INT0	I/O	输入/输出端，作输入端使用时为施密特触发器结构 具有唤醒功能 内置上拉电阻 INT0 触发端 (施密特触发结构) TC0 事件计数信号输入端
4	P1.5/RST/VPP	I,P	P1.5：外部复位无效时仅可作输入口使用 (施密特触发器结构) P1.5 无内置上拉电阻 具有唤醒功能 RST：系统复位输入端，施密特触发器结构，低电平有效，通常保持高电平 VPP：OTP 烧录引脚
5	VSS	P	数字电路电源输入端
6	P5.0	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻
7	P5.1	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻
8	P5.2	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻
9	P5.3	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻
10	P5.4/BZ0/PWM0	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻 蜂鸣器及 PWM 输出端
11	P5.5	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻
12	P5.6	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻
13	P5.7	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻
14	VDD	P	数字电路电源输入端
15	XOUT/P1.4	I/O	输入/输出端，作输入端使用时为施密特触发器结构 具有唤醒功能 XOUT: 采用外部振荡器的时晶振输出端
16	XIN	I/O	外部振荡信号输入端
17	P1.0	I/O	输入/输出端，漏极开路，作输入端使用时为施密特触发结构 内置上拉电阻
18	P1.1	I/O	输入/输出端，作输入端使用时为施密特触发结构,内置上拉电阻

2.4 端口结构

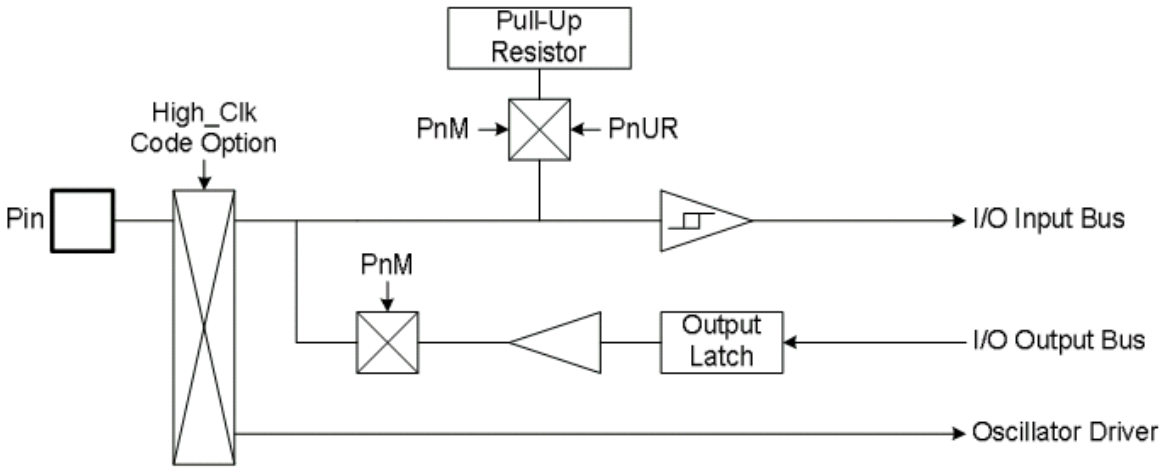
- P0、P1.P5.4~P5.7 端口结构



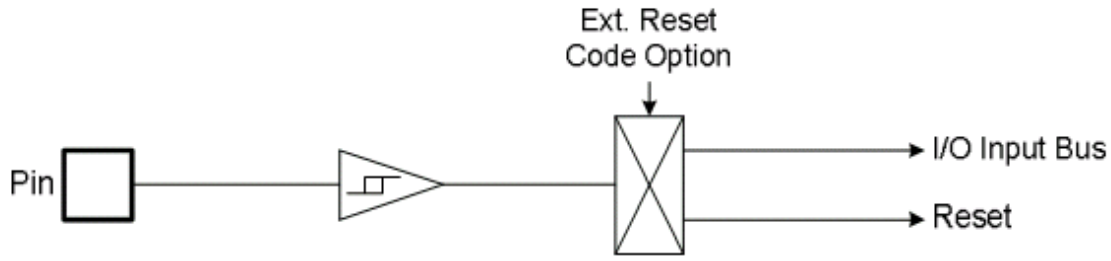
P1.0 端口结构



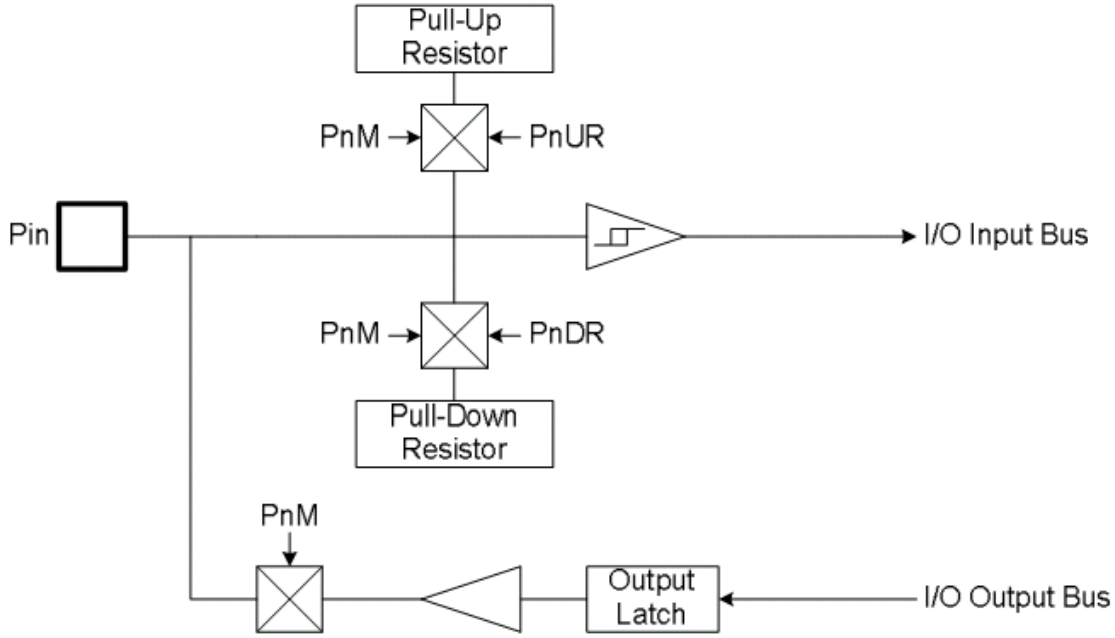
P1.4，P1.6 端口结构



P1.5 端口结构



- P5.0~P5.3 端口结构



3、电特性

3.1、 极限参数

除非另有规定，T_{amb}=25℃

参 数 名 称	符 号	条 件		额 定 值	单 位
电源电压	V_{CC}			-0.3~ +6.0	V
端口输入电压	V_{IN}			V_{SS}-0.2~V_{DD}+0.2	V
工作环境温度	T_{amb}			-40~85	℃
贮存温度	T_{stg}			-50~125	℃
焊接温度	T_L	10 秒	DIP	245	℃
			SOP	250	

3.2、电气特性

3.2.1 直流参数（ 除非另有规定，T_{amb}=25℃，Vdd=5.0V，Fcpu=1MHz，Fosc=4MHz）

符号	参数	测试条件		最小值	典型值	最大值	单位
Vdd	工作电压	正常模式，Vpp=Vdd，Fcpu=1MHz		2.2	-	5.5	V
Vdr	RAM 数据滞留电压			1.5	-	-	V
*Vpor	Vdd 上升率	Vdd 上升率确保内部接通复位		0.05	-	-	V/ms
ViL1	输入低电压	所有输入端口		Vss	-	0.3Vdd	V
ViL2		复位端口		Vss	-	0.3Vdd	V
ViH1	输入高电压	所有输入端口		0.7Vdd	-	Vdd	V
ViH2		复位端口		0.9Vdd	-	Vdd	V
Llekg	复位端口漏电流	Vin=Vdd		-	-	2	uA
Llekg	I/O 端口输入漏电流	上拉电阻禁用，Vin=Vdd		-	-	2	uA
Rup	I/O 端口上拉电阻	Vin=Vss，Vdd=3V		100	200	300	KΩ
		Vin=Vss，Vdd=5V		50	100	150	
LoH	I\O 端口灌拉电流	Vop=Vdd-0.5V		8	15	-	mA
LoL1		Vop=Vss+0.5V		8	15	-	
LoL2		Vop=Vss+0.5V，P5.0~P5.3，P5.5		20	40	-	
Tint0	INTn 触发脉冲宽度	INTn 中断请求脉冲宽度		2/fcpu	-	-	cycle
Idd1	电源电流	运行模式（低功率禁用）	Vdd=3V，Fcpu=16MHz	-	2.8	-	mA
			Vdd=5V，Fcpu=16MHz	-	5.8	-	mA
			Vdd=3V，Fcpu=4MHz	-	1.5	-	mA
			Vdd=5V，Fcpu=4MHz	-	3	-	mA
			Vdd=3V，Fcpu=1MHz	-	1.1	-	mA

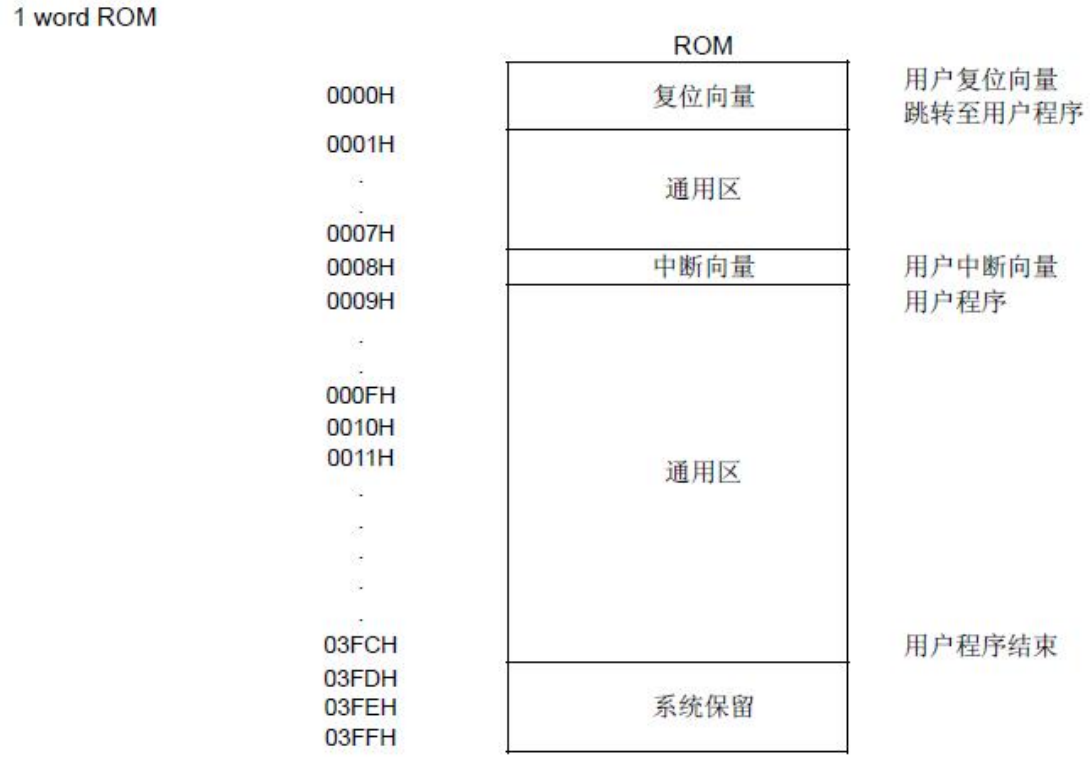
			Vdd=5V，Fcpu=1MHz	-	2.3	-	mA
			Vdd=3V，Fcpu=32MHz	-	20	-	uA
			Vdd=5V，Fcpu=32MHz	-	45	-	uA
Idd2		运行模式（低功率使用）	Vdd=3V，Fcpu=4MHz	-	1.3	-	mA
			Vdd=5V，Fcpu=4MHz	-	2.2	-	mA
			Vdd=5V，Fcpu=1MHz	-	0.7	-	mA
			Vdd=3V，Fcpu=16MHz	-	1	-	mA
Idd3		缓慢模式（内部低抗RC，停止高时钟）	Vdd=3V，Fcpu=16MHz	-	2.5	-	uA
			Vdd=5V，Fcpu=32MHz	-	7.8	-	uA
Idd4		睡眠模式	Vdd=5V/3V	-	1	2	uA
Idd5		绿色模式（下载，看门狗禁用）	Vdd=3V，IHRC=16MHz				
			Vdd=5V，IHRC =16MHz				
			Vdd=3V，Ext.32KHzX’tal				
			Vdd=5V，Ext.32KHzX’tal				
			Vdd=3V，ILRC=16MHz				
			Vdd=3V，ILRC=16MHz				
Fihrc	内部高振荡器频率	内部高频 RC (IHRC)	Vdd=2.2V~5.5V Fcpu=Fhosc/1~Fhosc/128	15.68	16	16.32	MHz

4、功能介绍

4.1、CPU 处理器

4.1.1 处理器

4.1.1.1 程序存储器



4.1.1.1.1 复位向量（0000H）

一个字长的系统复位向量。

- 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)。

系统执行完上电复位、外部复位或看门狗定时器溢出复位后，程序将从0000H 处开始重新执行，系统寄存器也都将重置为默认值。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状况。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               ; 0000H
        JMP     START           ; 跳至用户程序
        …
        ORG     10
START:                          ; 0010H, 用户程序首地址
        …                       ; 用户程序
        …
        ENDP                    ; 程序结束
```

4.1.1.1.2 中断向量（00008H）

长度为1-word，用于执行中断请求。当系统响应某个中断请求时，程序计数器（PC）的当前值存

入堆栈缓冲器后转至中断向量0008H。用户可以自己定义中断向量，下面的程序说明了如何在程序中定义中断向量。

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，中断服务程序就在ORG 8 之后。

```
.CODE
                ORG         0                   ; 0000H
                JMP         START               ; 跳至用户程序
                …
                ORG         8                   ; 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，中断程序位于用户程序之后。

```
.CODE
                ORG         0                   ; 0000H
                JMP         START               ; 跳至用户程序
                …
                ORG         8                   ; 中断向量
                JMP         MY_IRQ              ; 0008H,跳至中断程序
                ORG         10H
START:                                          ; 0010H, 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。

4.1.1.1.3 查表

对ROM 数据进行查找，寄存器Y 指向查找数据地址的高字节（bit8~bit15），寄存器Z 指向地址的低字节（bit0~bit7）。执行完MOVC 指令后，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 bits) 数据
                DW        5105H
                DW        2012H
                …
```

注：当寄存器Z 溢出（从0xFF 变为0x00）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，下面的宏指令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 bits) 数据
                DW      5105H
                DW      2012H
                …
```

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

- 例：由指令B0ADD/ADD 对Y 和Z 寄存器加1。

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

                B0MOV   A, BUF          ; Z = Z + BUF.
                B0ADD   Z, A
                B0BTS1  FC              ; 检查进位标志
                JMP     GETDATA         ; FC = 0
                INCMS   Y               ; FC = 1
                NOP
GETDATA: ;
                MOVC                    ; 存储数据，如果BUF = 0，数据为
                0x0035
                                        ; 如果BUF = 1，数据＝0x5105
                                        ; 如果BUF = 2，数据＝0x2012
                …
TABLE1:         DW      0035H           ; 定义数据表 (16 bits) 数据
                DW      5105H
                DW      2012H
                …
```

4.1.1.1.4 跳转表

跳转表能够实现多地址跳转功能。PCL 和ACC 的值相加即可得到新的PCL。如果PCL＋ACC 后发生溢出，PCH 则自动加1。由此得到的新的PC 值指向一系列跳至指令列表。如此可通过修改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     0X0100          ; 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 提供一个宏程序以保证可靠执行跳转表功能，它将检测ROM 边界并自动将跳转表移至适

当的位置。但采用该宏程序会占用部分ROM 空间。

❙ 例：如果跳转表在ROM 中跨段，则程序可能出错。

```
@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：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（0x00FF~0x0100），宏“@JMP_A”将调整跳转表到适当的位置（0x0100)。

❙ 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
0X00FD          JMP       A0POINT              ; ACC = 0，跳至A0POINT
0X00FE          JMP       A1POINT              ; ACC = 1，跳至A1POINT
0X00FF          JMP       A2POINT              ; ACC = 2，跳至A2POINT
0X0100          JMP       A3POINT              ; ACC = 3，跳至A3POINT
0X0101          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
0X0100          JMP       A0POINT              ; ACC = 0，跳至A0POINT
0X0101          JMP       A1POINT              ; ACC = 1，跳至A1POINT
0X0102          JMP       A2POINT              ; ACC = 2，跳至A2POINT
0X0103          JMP       A3POINT              ; ACC = 3，跳至A3POINT
0X0104          JMP       A4POINT              ; ACC = 4，跳至A4POINT
```

4.1.1.1.5 CHECKSUM 计算

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

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

```
                MOV         A,#END_USER_CODE$L
                B0MOV       END_ADDR1, A          ; low end address 地址存入end_addr1
                MOV         A,#END_USER_CODE$M
                B0MOV       END_ADDR2, A ; middle end address 存入 end_addr2
                CLR         Y                     ; Y 清零
                CLR         Z                     ; Z 清零
@@:
                MOVC
                B0BCLR      FC                    ; 标志位C 清零
                ADD         DATA1, A              ;
                MOV         A, R
                ADC         DATA2, A ;
                JMP         END_CHECK             ; 检验YZ 地址 = 代码结束地址?
AAA:
                INCMS       Z                     ; Z=Z+1
                JMP         @B                    ; 如果Z != 00H，进行下一个计算
                JMP         Y_ADD_1               ; 如果Z = 00H，Y 加1
END_CHECK:
                MOV         A, END_ADDR1
                CMPRS       A, Z                  ; 检验Z 地址 = 代码结束地址？
                JMP         AAA                   ; 否，则进行checksum 计算
                MOV         A, END_ADDR2
                CMPRS       A, Y                  ; 是，则检测是否Y = middle end address
                JMP         AAA                   ; 不等则跳至checksum 计算
                JMP         CHECKSUM_END          ; 相等，则 checksum 计算已结束
Y_ADD_1:
                INCMS       Y
                NOP
                JMP         @B                    ; 转至checksum 计算
CHECKSUM_END:
                …
                …
END_USER_CODE:
```

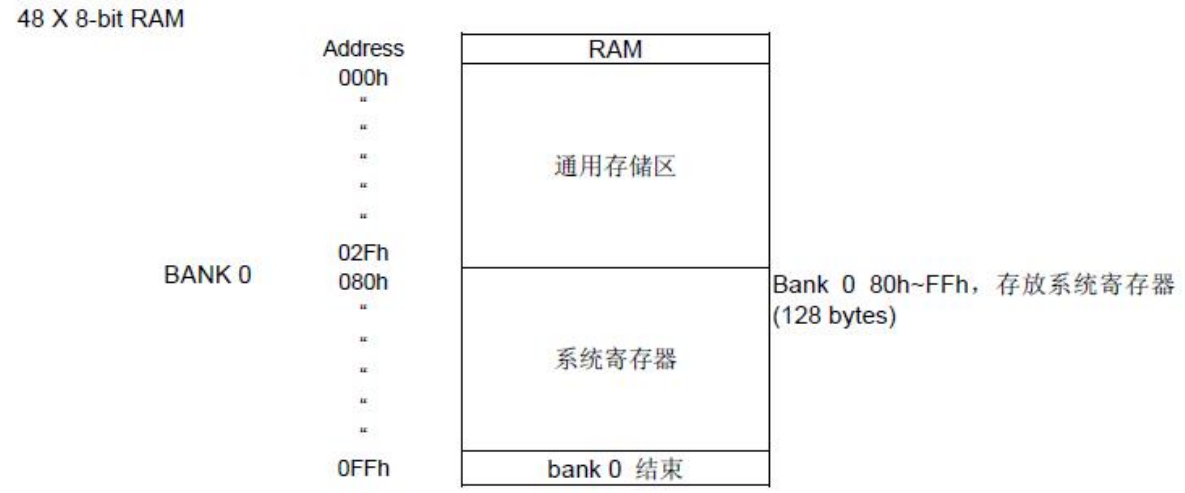
编译选项	内容	功能说明
Noise_Filter	Enable	开启杂讯滤波功能，Fcpu = Fosc/4~Fosc/8。
	Disable	关闭杂讯滤波功能，Fcpu = Fosc/1~Fosc/8。
Fcpu	Fhosc/1	指令周期 = 时钟周期； 注：若选择 Fosc/1，则必须关闭杂讯滤波功能。

	Fhosc/2	指令周期 = 2 个时钟周期; 注：若选择 Fosc/2，则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
High_Clk	RC	廉价RC 振荡电路提供内部高速时钟， XOUT 为P1.4 输入/输出引脚
	32K X'tal	低频，省电石英振荡器（如32.768KHz）作为外部高速时钟源。
	12M X'tal	高速石英/陶瓷振荡器（如12MHz）作为外部高速时钟源。
	4M X'tal	标准石英/陶瓷振荡器（如4M）作为外部高速时钟源。
Watch_Dog	Always_On	看门狗定时器始终处于有效状态，即使在睡眠模式和绿色模式下。
	Enable	开启看门狗定时器，在睡眠模式和绿色模式下关闭。
	Disable	关闭看门狗。
Reset_Pin	Reset	外部复位引脚有效。
	P15	作为P1.5 输入引脚，无上拉电阻。
Low Power	Enable	开启省电功能以减小工作电流。
	Disable	一般情况。
LVD	LVD_L	VDD 低于2.0V 时，LVD 复位系统。
	LVD_M	VDD 低于2.0V 时，LVD 复位系统; LVD的24-bit PFLAG 寄存器作为2.4V低电压监测器。
	LVD_H	VDD 低于2.4V 时，LVD 复位系统; LVD 的36-bit PFLAG 寄存器作为3.6V 低电压监测器。
Security	Enable	允许ROM 代码加密。
	Disable	禁止ROM 代码加密。

注:

1. 在干扰严重的情况下，建议开启杂讯滤波功能，此时Fcpu = Fosc/4 ~ Fosc/8，并将Watch_Dog 设置为 "Always_On";
2. 如果用户定义watchdog 为"Always_On",编译器将自动开启看门狗定时器;
3. 编译选项Fcpu 仅针对外部振荡信号，在进入内部低速模式时，Fcpu = FILRC/4。

4.1.1.3 数据存储器



4.1.1.4 系统寄存器

4.1.1.4.1 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	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	P0M	-	-	-	-	-	-	PEDGE
C	P1W	P1M	-	-	-	P5M	-	-	INTRQ	INTEN	OSCM	-	WDTR	T0R	PCL	PCH
D	P0	P1	-	-	-	P5	-	-	T0M	T0C	TC0M	TC0C	-	-	-	STKP
E	P0UR	P1UR	-	-	-	P5UR	-	@YZ	-	P1OC	-	-	-	-	-	-
F	-	-	-	-	-	-	-	-	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

寄存器说明

PFLAG = ROM 页及特殊寄存器　　R = 工作寄存器和ROM 查表数据缓存器

P1W = P1 唤醒功能寄存器　　Y, Z = 工作寄存器, @YZ 和ROM 寻址寄存器

PEDGE = P0.0 触发方向寄存器　　@YZ = RAM YZ 间接寻址指针

PnM = Pn 输入/输出模式寄存器　　Pn = Pn 数据缓存器

P1OC = P1 漏极开路控制寄存器　　PnUR = Pn 上拉电阻控制寄存器

INTRQ = 中断请求寄存器　　INTEN = 中断使能寄存器

OSCM = 振荡器模式寄存器　　PCH, PCL = 程序计数器

T0M = T0 模式寄存器　　T0C = T0 计数寄存器

TC0M = TC0 模式寄存器　　TC0C = TC0 计数寄存器

TC0R = TC0 自动加载数据缓存器　　WDTR = 看门狗定时器清零寄存器

STKP = 堆栈指针缓存器　　STK0~STK3 = 堆栈缓存器

4.1.1.4.2 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
0B8H	-	-	-	-	-	-	-	P00M	R/W	P0M
0BFH	-	-	-	P00G1	P00G0	-	-	-	R/W	PEDGE
0C0H	-	-	P15W	P14W	P13W	P12W	P11W	P10W	W	P1W 唤醒功能寄存器
0C1H	-	-	-	P14M	P13M	P12M	P11M	P10M	R/W	P1M I/O 模式寄存器
0C5H	P57M	P56M	P55M	P54M	P53M	P52M	P51M	P50M	R/W	P5M I/O 模式寄存器
0C8H	-	-	T0IRQ	T0IRQ	-	-	-	P00IRQ	R/W	INTRQ
0C9H	-	-	T0IEN	T0IEN	-	-	-	P00IEN	R/W	INTEN
0CAH	-	-	-	CPUM1	CPUM0	CLKMD	STPHX	-	R/W	OSCM
0CCH	WDTR7	WDTR6	WDTR5	WDTR4	WDTR3	WDTR2	WDTR1	WDTR0	W	WDTR
0CDH	T0R7	T0R6	T0R5	T0R4	T0R3	T0R2	T0R1	T0R0	W	T0R
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	-	-	PC9	PC8	R/W	PCH
0D0H	-	-	-	-	-	-	-	P00	R/W	P0 数据缓存器
0D1H	-	-	P15	P14	P13	P12	P11	P10	R/W	P1 数据缓存器
0D5H	P57	P56	P55	P54	P53	P52	P51	P50	R/W	P5 数据缓存器
0D8H	T0ENB	T0rate2	T0rate1	T0rate0	-	-	-	-	R/W	T0M
0D9H	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0	R/W	T0C
0DAH	T0ENB	T0rate2	T0rate1	T0rate0	T0CKS	ALOAD0	T0OUT	PWM0OUT	R/W	T0M
0DBH	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0	R/W	T0C
0DFH	GIE	-	-	-	-	-	STKPB1	STKPB0	R/W	STKP 堆栈指针
0E0H	-	-	-	-	-	-	-	P00R	W	P0 上拉电阻寄存器
0E1H	-	-	-	P14R	P13R	P12R	P11R	P10R	W	P1 上拉电阻寄存器
0E5H	P57R	P56R	P55R	P54R	P53R	P52R	P51R	P50R	W	P5 上拉电阻寄存器
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ
0E9H	-	-	-	-	-	-	-	P10OC	W	P1OC 漏极开路寄存器
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. 为了避免系统出错，请确认系统寄存器各位的值应与上表所示一致；
2. 所有系统寄存器的名称都已经在SN8ASM 编译器中宣告过；
3. 寄存器各位的名称都已经以“F”为前缀在SN8ASM 宣告过；
4. 指令“b0bset”, “b0bclr”, ”bset”, ”bclr”仅对“R/W”寄存器有效；
5. 详细信息请参考“系统寄存器快速参考表”。

4.1.1.4.3 累加器

8-bit 数据寄存器ACC 在ALU 和数据存储器之间进行数据的传送和操作。如果操作结果为零（Z）或有进位产生（C或DC），标志寄存器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

```
; 读取ACC 中的数据并存入BUF
                MOV         BUF, A
; 数据写入ACC
                MOV         A, #0FH
; 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，必须将中断入口处的ACC 和PFLAG 中的数据送入存储器进行保存。由程序通过“PUSH”和“POP”指令，对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        ; PFLAG 和ACC 数据送入缓冲器.
        ….
        …
        POP         ; 恢复ACC 和PFLAG
        RETI        ; 退出中断
```

4.1.1.4.4 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显示了芯片供电电源状况。

086H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PFLAG	NT0	NPD	LVD36	LVD24		C	DC	Z
读/写	R/W	R/W	R	R		R/W	R/W	R/W
复位后			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：LVD 3.6V 工作标志， LVD 编译选项为LVD_H 时有效。

0 = 无效(VDD > 3.6V)；

1 = 有效(VDD≤3.6V)。

Bit 4 LVD24：LVD 2.4V 工作标志，仅支持LVD 编译选项为LVD_M。

0 =无效(VDD > 2.4V)；

1 =有效(VDD≤2.4V)。

Bit 2 C：进位标志。

1 = 加法运算后有进位、减法运算没有借位发生或移位后移出逻辑“1”或比较运算的结果≥0；

0 = 加法运算后没有进位、减法运算有借位发生或移位后移出逻辑“0”或比较运算的结果<0。

Bit 1 DC：辅助进位标志。

1 = 加法运算时低四位有进位，或减法运算后没有向高四位借位；

0 = 加法运算时低四位没有进位，或减法运算后有向高四位借位。

Bit 0 Z：零标志。

1 = 算术/逻辑/分支运算的结果为零；

0 = 算术/逻辑/分支运算的结果非零。

注：关于标志位C、DC 和Z 的更多信息请参阅指令集相关内容。

4.1.1.4.5 程序计数器

10-bit 程序计数器PC 分为2-bit 高字节和8-bit 低字节，用于定位程序执行的指令。程序运行过程中，每执行一条指令，程序计数器PC 将自动加1。

但是，如果程序执行CALL 或JMP 指令时，PC 则指向特定地址。

	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PC	-	-					PC0	PC0	PC0	PC0	PC0	PC0	PC0	PC0	PC0	PC0
复位后							0	0	0	0	0	0	0	0	0	0
	PCH								PCL							

- 单地址跳转

共有9 条单地址跳转指令：CMPRS, INCS, INCMS, DECS, DECMS, BTS0, BTS1, B0BTS0, B0BTS1，如果这些指令执行的结果为真，PC 值加2 跳过待执行的下一条指令。

```
        B0BTS1      FC              ; 如果Carry_flag = 1 则跳过下一条指令
        JMP         C0STEP          ; 否则跳至C0STEP.
        …
C0STEP: NOP
        B0MOV       A, BUF0         ; BUF0 的值送入ACC.
        B0BTS0      FZ              ; 如果Zero flag = 0，跳过下一条指令
        JMP         C1STEP          ; 否则转至 C1STEP.
        …
C1STEP: NOP
```

如果 ACC 的值与立即数或存储器的值相等，则PC 值将加2 以跳过下一条指令。

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

如果PC 值加1 时从 0xFF 溢出到0x00, 则PC 值将加2 以跳过下一条指令。

INCS 指令：

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

INCMS 指令：

```
        INCMS       BUF0
```

```
                JMP         C0STEP
                …
                C0STEP:     NOP
```

如果PC 值减1 时从0x00 溢出到0xFF 则PC 值将加2 以跳过下一条指令。

DECS 指令：

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

DECMS 指令：

```
                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。PCL 溢出并向PCH 产生进位时，可采用指令“ADDM,A”, ”ADC M,A”和 “B0ADD M,A”。对于跳转表及其它应用，可采用以上三条指令计算PC 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 执行完后有进位时，PCH 会自动加1。若 PCL-ACC 执行完后有借位发生，此时PCH 的值保持不变。

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

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

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

```
; PC = 0323H
                B0ADD       PCL, A          ; PCL = PCL + ACC, PCH 不变
                JMP         A0POINT         ; ACC = 0，跳转至A0POINT
                JMP         A1POINT         ; ACC = 1，跳转至A1POINT
                JMP         A2POINT         ; ACC = 2，跳转至A2POINT
                JMP         A3POINT         ; ACC = 3，跳转至A3POINT
```

4.1.1.4.6 Y, Z 寄存器

寄存器Y 和Z 都是8 位缓存器，主要用途如下：

- 普通工作寄存器；

- RAM 数据寻址指针@YZ；
- 配合指令MOVC 对ROM 数据进行查表。

084H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Y	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	-	-	-	-	-

083H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Z	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	-	-	-	-	-

- 例：用 Y, Z 作为数据指针寻址RAM 中bank0 的025H。

```
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, bank 0
B0MOV   Z, #07FH        ; 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: ;
…
```

4.1.1.4.7 R 寄存器

8 位缓冲器R 主要有以下两个功能：

- 作为工作寄存器使用；
- 存储查表函数的高字节数据。

(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	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
R	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	-	-	-	-	-

4.1.2 寻址模式

4.1.2.1 立即寻址

立即寻址就是把一个立即数送入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：立即数寻址范围必须介于RAM 的0x80~0x87 单元。

4.1.2.2 直接寻址

直接寻址即是实现RAM 单元与ACC 之间的数据传输。

- 例：RAM 0x12 中数据送入ACC。

```
B0MOV       A, 12H
```

- 例：将ACC 中数据送入RAM 的0x12 单元。

```
B0MOV       12H, A
```

2.2.3 间接寻址

间接寻址即是通过指针寄存器（Y/Z）访问RAM 数据。

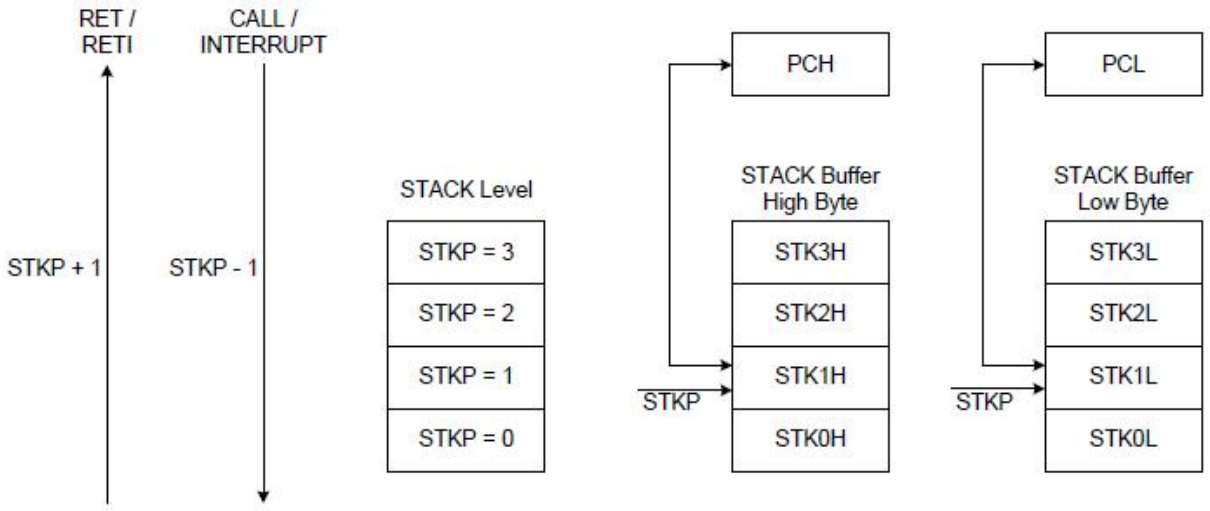
- 例：用 @YZ 实现间接寻址。

```
B0MOV       Y, #0           ; Y 清零以寻址RAM bank 0.
B0MOV       Z, #12H         ; 立即数12H 送入Z
B0MOV       A, @YZ
```

4.1.3 堆栈

4.1.3.1 概述

堆栈缓存器共4 层，当程序进入中断或执行CALL 指令时，用于存放PC 值。寄存器STKP 指向堆栈的当前层，STKnH和STKnL 具体存放PC 值。



4.1.3.2 堆栈寄存器

2-bit 堆栈指针STKP 用于存放当前访问的堆栈缓存器的地址，9-bit 数据存储器STKnH 和STKnL 用于临时存放堆栈操作地址。

入栈操作时STKP 的值减1，相反，执行出栈操作时，STKP 的值则加1。因此，STKP 总是指向堆栈最顶层的缓存器。程序进入中断或执行CALL 指令之前，程序计数器PC 的值被存入堆栈缓存器中。堆栈操作遵循LIFO（后进先出）的规则，堆栈指针STKP 和堆栈缓存器(STKnH 和STKnL) 位于系统寄存器的Bank 0。

0DFH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STKP	GIE	-	-	-	-	-	STKPB1	STKPB0
读/写	R/W	-	-	-	-	-	R/W	R/W
复位后	0	-	-	-	-	-	1	1

Bit[2:0] STKPBn: 堆栈指针 (n = 0 ~ 2)

Bit 7 GIE: 全局中断控制位。

0 = 禁止

1 = 允许（详见中断章节）。

- 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另行程序设定，如下面所示：

```
MOV     A, #00000011B
B0MOV   STKP, A
```

0F0H～0F8H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STKnH	-	-	-	-	-	-	SnPC9	SnPC8
读/写	-	-	-	-	-	-	R/W	R/W
复位后	-	-	-	-	-	-	0	0

0F0H～0F8H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STKnL	SnPC7	SnPC6	SnPC5	SnPC4	SnPC3	SnPC2	SnPC1	SnPC0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

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

4.1.3.3 堆栈操作举例

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

堆栈层数	STKP		堆栈缓存器		备注
	STKPB1	STKPB0	高字节	低字节	
0	***1***	***1***	Free	Free	-
1	***1***	***0***	STK0H	STK0L	-
2	***0***	***1***	STK1H	STK1L	-
3	***0***	***0***	STK2H	STK2L	-
4	***1***	***1***	STK3H	STK3L	-
> 4	***1***	***0***	-	-	***溢出出错***

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

堆栈层数	STKP		堆栈缓存器		备注
	STKPB1	STKPB0	高字节	低字节	
4	***1***	***1***	STK3H	STK3L	-
3	***0***	***0***	STK2H	STK2L	-
2	***0***	***1***	STK1H	STK1L	-
1	***1***	***0***	STK0H	STK0L	-
0	***1***	***1***			-

4.2复位

4.2.1 概述

在如下 4 种情况下系统会复位：

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

当上述任何一种复位发生时，所有的系统寄存器被初始化，程序停止运行同时计数器PC 清零。复位结束后，系统重启，程序从ORG 0 处开始执行。标志位NT0 和NPD 用于指示系统的复位状态，可对这两个寄存器编程控制系统复位方式。

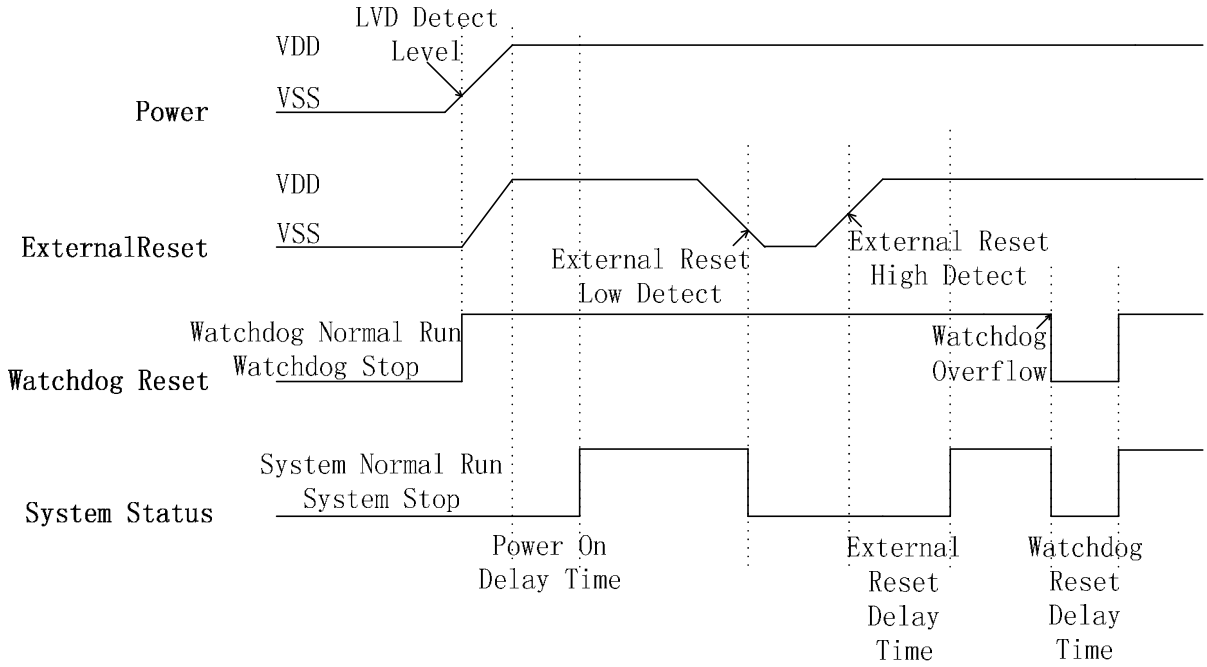
086H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PFLAG	NT0	NPD	-	-	-	C	DC	Z
读/写	R/W	R/W	-	-	-	R/W	R/W	R/W
复位后	-	-	-	-	-	0	0	0

Bit [7:6] NT0, NPD：复位状态标志

NTO	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，复位完成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。



4.2.2 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：系统检测电压值，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。

4.2.3 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：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。

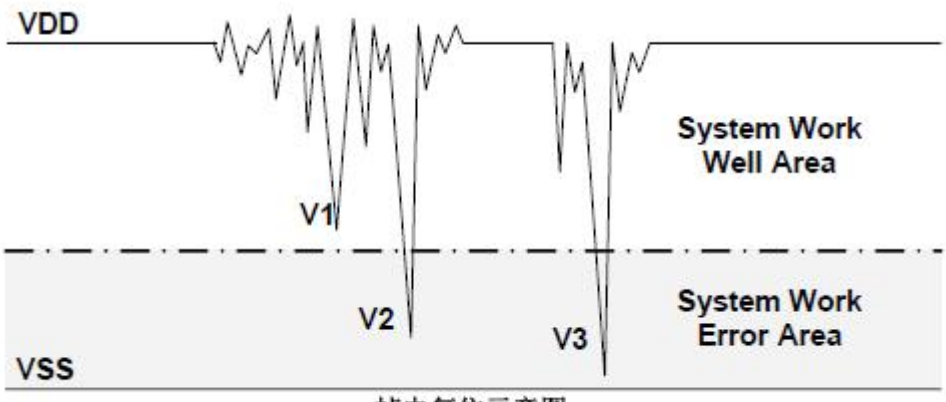
看门狗定时器应用注意事项：

- 看门狗定时器清零之前，请检查I/O 端口状态及RAM 数据；
- 不能在中断向量和中断程序中将看门狗定时器清零，否则无法起到侦测程序跑飞的目的；
- 程序中应该只在主程序中有一个清看门狗的动作，这种架构能够最大限度的发挥看门狗的保护功能。

4.2.4 掉电复位

4.2.4.1 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 应用中：

DC 应用中一般都采用电池供电，当电池电压过低或MCU 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系统复位电压，因此系统维持在死区。

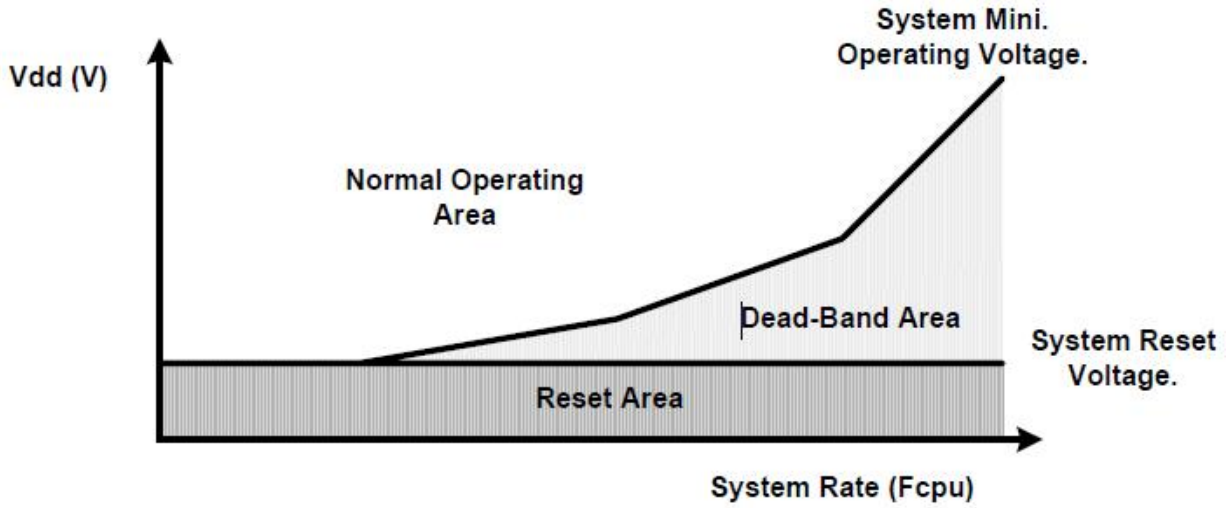
AC 应用中：

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，则系统工作电压会失去稳定状态。

在AC 应用中，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 应用中情形类似，AC 电源关断后，VDD 电压在缓慢下降的过程中易进入死区。

4.2.4.2 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亦不同。电气特性一章给出了系统工作电压与执行速度之间的关系。



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略高于系统复位电压，同时复位电压由LVD 检测电平决定。当系统执行速度高于复位电压时，系统最低工作电压值将会升高。复位电压与最低工作电压之间的区域即是系统工作的死区。

4.2.4.3 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：

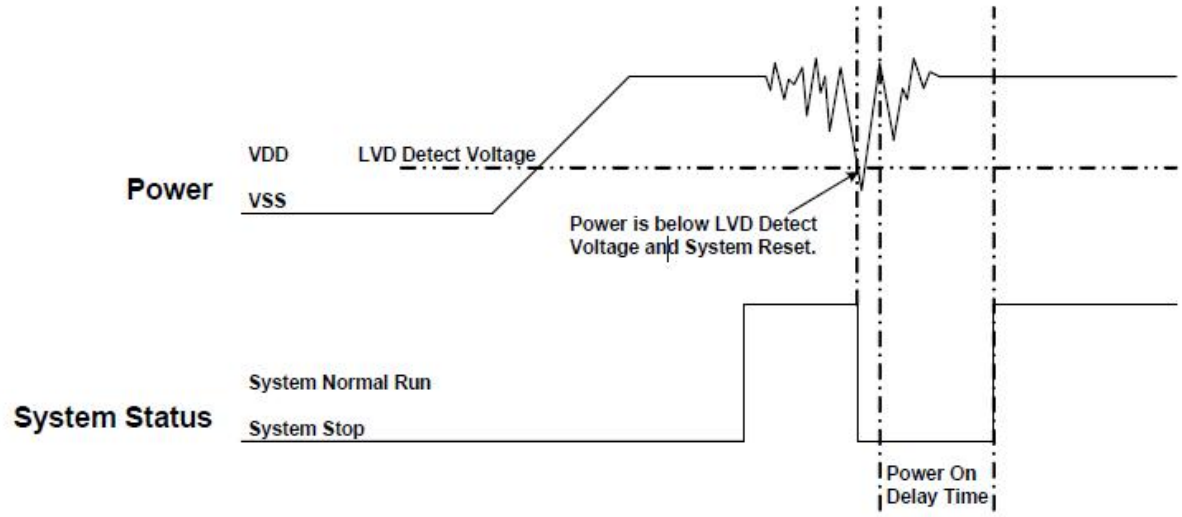
- LVD 复位；
- 看门狗复位；
- 降低系统工作速度；
- 采用外部复位电路 (齐纳二极管复位电路，电压偏移复位电路，外部IC 复位)。

注：

a）.“齐纳二极管复位电路”、“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；

b）. 对于AC 供电的系统，可提高EFT 性能，系统时钟采用4MHz 和外部复位（齐纳二极管复位电路，电压偏移复位电路，外部IC复位）。电路结构的优化可降低噪声的影响并获得更好的EFT 特性。

LVD 复位：



LVD(低电压检测)是SONiX 8-bit MCU 的内置掉电复位保护装置，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MCU 有不同的LVD 检测电平，LVD 检测电平值仅为一个点电压，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 却使得系统工作出错，那么LVD 就不能起到保护作用，就需要采用其它复位方法。电气特性一章中给出了更多关于LVD 的详细内容。

LVD 设计为三层结构（2.0V/2.4V/3.6V），由LVD 编译选项控制决定。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	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PFLAG	NT0	NPD	LVD36	LVD24	-	C	DC	Z
读/写	R/W	R/W	R	R	-	R/W	R/W	R/W
复位后	-	-	0	0	-	0	0	0

Bit 5 LVD36：LVD 3.6V 工作标志，仅当LVD 编译选项为LVD_H 时有效；

0 = LVD36 标志位无效 (VDD > 3.6V)；

1 = LVD36 标志位有效 (VDD≤3.6V)；

Bit 4 LVD24：LVD 2.4V 工作标志，仅当LVD 编译选项为LVD_M 时有效；

0 = LVD24 标志位无效 (VDD > 2.4V)；

1 = LVD24 标志位有效 (VDD≤2.4V)；

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 <= 2.4V，LVD24＝ “1”

LVD36 标志位无意义。

LVD2_H

如果VDD < 2.4V, 系统复位；

LVD24，如果VDD > 2.4V，LVD24＝“0”；如果VDD <= 2.4V， LVD24＝“1”

LVD36，如果VDD > 3.6V，LVD36＝“0”；如果VDD <= 3.6V， LVD36＝ “1”

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，系统复位并回到正常模式。如果看门狗复位后电源仍处于死区，则系统复位失败，一直处于复位状态直到电源回到正常水平。

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，直到电源恢复到正常值。

降低系统工作速度：

如果系统工作速度过快容易导致芯片最低工作电压值增高，从而加大工作死区的范围，那么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。因为系统工作速度较低则最小工作电压值相应较低。所以，调节电源电压选择恰当的电压值，在该电压下不会带来死区问题，从而找到合适的系统工作速度。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：

外部复位发也能够完全改善掉电复位性能。共有三种外部复位方式：齐纳二极管复位电路，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MCU 可靠复位。

4.2.5 外部复位

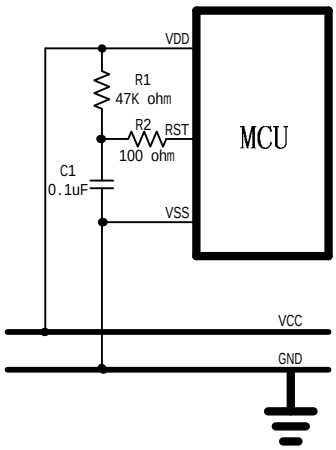
外部复位功能由“Reset_Pin”编译选项控制。将该编译选项置为“Reset”，从而使能外部复位功能。外部复位引脚为施密特触发器结构，低电平有效。复位引脚处于高电平时，系统正常运行。当复位引脚输入低电平信号时，系统复位。外部复位操作在上电和正常工作模式时有效。需要注意的是，在系统上电的过程中，外部复位引脚必须输入高电平，否则系统将一直保持在复位状态。外部复位的时序如下：

- 外部复位（当且仅当外部复位引脚为使能状态）：系统检测外部复位引脚的状态，如果外部复位引脚不为高电平，则系统保持在复位状态直到外部复位引脚被释放；
- 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。

4.2.6 外部复位电路

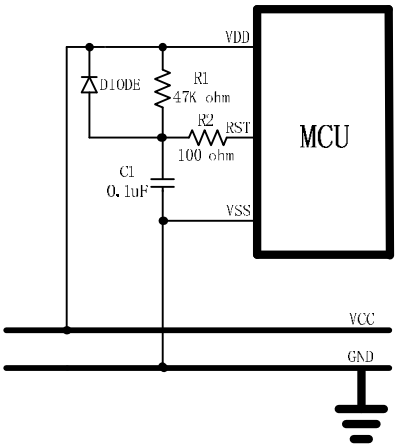
4.2.6.1 RC 复位电路



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间，当复位引脚达到高电平时，系统复位结束，进入正常工作状态。

注：RC 复位电路不能解决非正常上电和掉电复位问题。

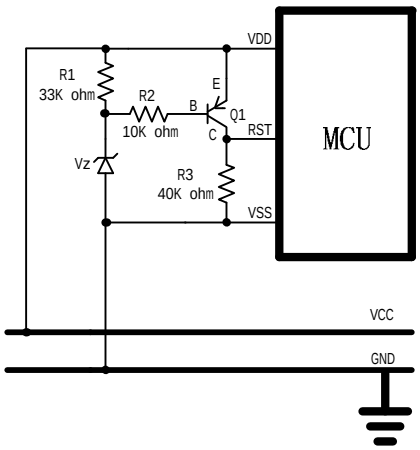
4.2.6.2 二极管&RC 复位电路



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

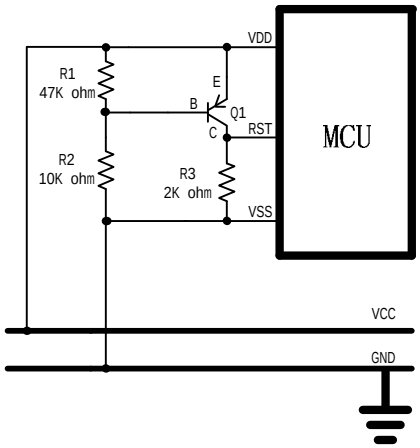
注：“基本RC 复位电路”和“二极管&RC 复位电路”中的电阻R2 都是必不可少的限流电阻，以避免复位引脚ESD（ElectrostaticDischarge）或EOS（Electrical Over-stress）)击穿。

4.2.6.3 齐纳二极管复位电路



齐纳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，利用齐纳管的击穿电压作为电路复位检测值，当VDD 高于“Vz + 0.7V”时，三极管集电极输出高电平，MCU 正常工作；当VDD 低于“Vz +0.7V”时，三极管集电极输出低电平，MCU 复位。齐纳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。

4.2.6.4 电压偏置复位电路

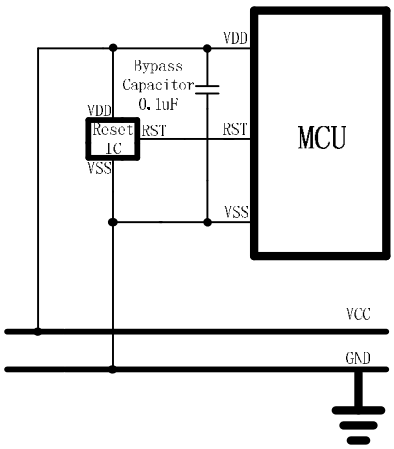


电压偏置复位电路是一种简单的LVD 电路，基本上可以完全解决掉电复位问题。与齐纳二极管复位电路相比，这种复位电路的检测电压值的精确度有所降低。电路中，R1 和R2 构成分压电路，当VDD 高于和等于分压值“0.7V x (R1 + R2)/ R1”时，三极管集电极C 输出高电平，MCU 正常工作；VDD 低于“0.7V x (R1 + R2) / R1”时，集电极C 输出低电平，MCU复位。

对于不同应用需求，选择适当的分压电阻。MCU 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 的电流稳定，在功耗电路如直流电源系统中，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，“齐纳二极管复位电路”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。

4.2.6.5 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。针对不用的应用要求选择适当的复位IC，如上图所示外部IC 复位电路，能够有效的降低电源变化对系统的影响。

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

4.3 系统时钟

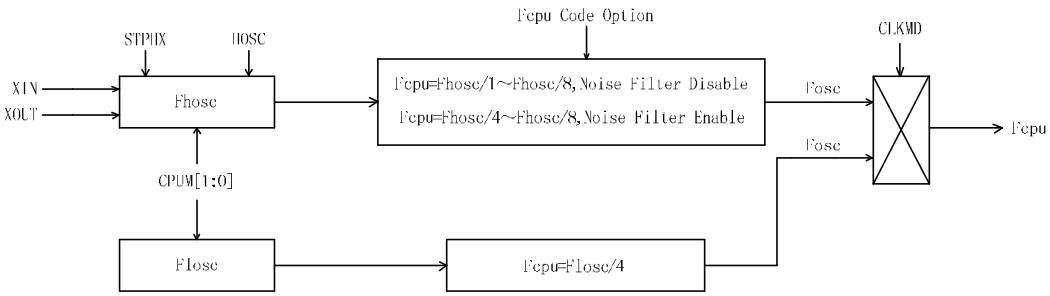
4.3.1 概述

双时钟系统包括高速时钟和低速时钟，高速时钟由外部振荡电路提供。低速时钟由内置的低速RC振荡电路（ILRC16KHZ@3V, 32KHz @5V）产生。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): Fcpu = Fhosc / N, N = 1 ~ 8, 由Fcpu 编译选项控制；
- 低速模式(低速时钟): Fcpu = Flosc/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。但在杂讯滤波器有效时，高速时钟的Fcpu 被限制为Fcpu＝Fhosc/4。

4.3.2 时钟框图



- HOSC：High_Clk 编译选项
- Fhosc：外部高速时钟频率
- Flosc：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

4.3.3 寄存器OSCM

寄存器OSCM 控制振荡器的状态和系统的工作模式。

0CAH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OSCM	0	0	0	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 = 外部高速时钟正常运行；

1 = 外部高速振荡器停止，内部低速RC 振荡器运行。

Bit 2 CLKMD：系统高/低速时钟模式控制位

0 = 普通模式，系统采用高速时钟；

1 = 低速模式，系统采用内部低速时钟。

Bit[4:3] CPUM[1:0]：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
```

- 例：进入睡眠模式时，停止高速及低速振荡器。

```
B0BSET      FCPUM0
```

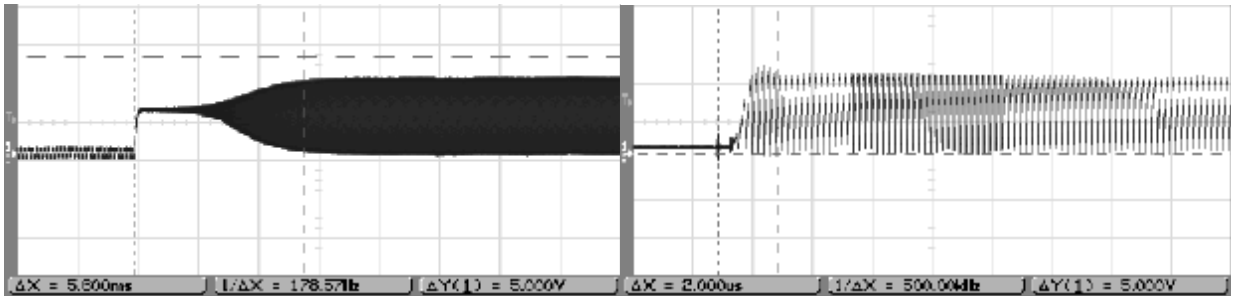
4.3.4 系统高速时钟

系统高速时钟源来自外部高速振荡器，由编译选项“High_Clk”控制。

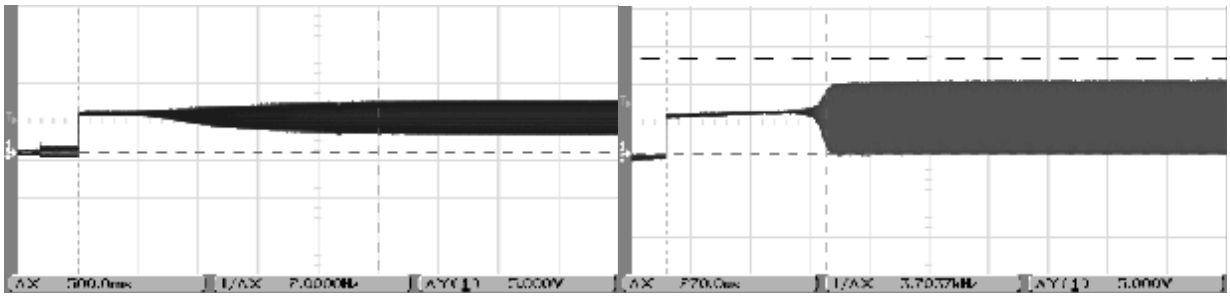
High_Clk	注释
RC	高速时钟为外部RC 振荡器，引脚XOUT 作为通用 I/O 端口。
32K	高速时钟为外部32768Hz 低速振荡器。
12M	高速时钟为外部高速振荡器，一般为10MHz ~ 16MHz。
4M	高速时钟为外部振荡器，输出频率2MHz ~ 10MHz。

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

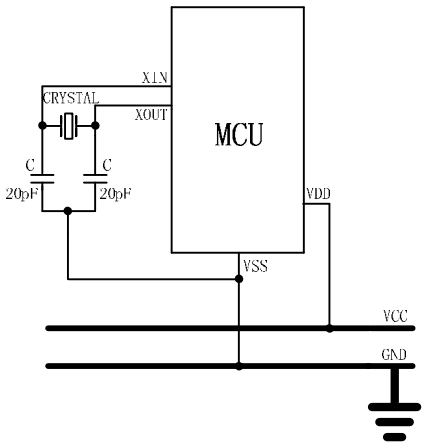


32768Hz Crystal 4MHz Ceramic



4.3.4.1 石英/陶瓷振荡器

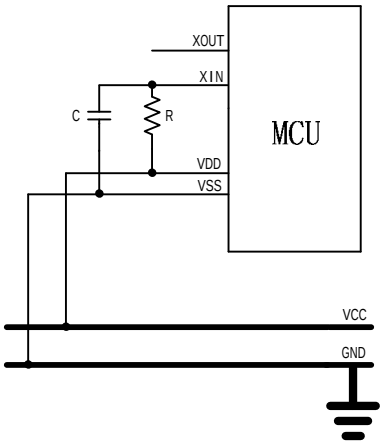
石英/陶瓷振荡器由XIN、XOUT 口驱动，对于高速、普通和低速三种不同工作模式，振荡器的驱动电流也不同。编译选项“High_Clk”亦支持不同的频率条件：高速模式下的12MHz 工作频率、普通模式的4MHz 工作频率以及低速模式下32KHz 工作频率。



注：上图中，XIN/XOUT/VSS 引脚与石英/陶瓷振荡器以及电容C 之间的线路越短越好。

4.3.4.2 RC 振荡器

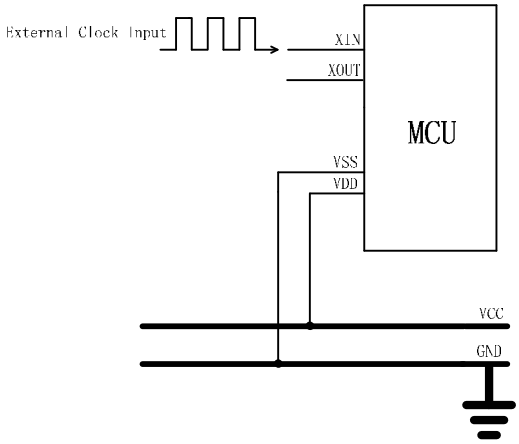
通过编译选项High_Clk 的设置可控制RC 振荡器的选择，RC 振荡器输出频率最高可达10MHZ。改变R 可改变输出频率的大小，电容C 的最佳容量为50P~100P，引脚XOUT 为通用I/O 口，如下图所示：



注：电容C 和电阻R 应尽可能的接近芯片的VDD。

4.3.4.3 外部时钟源

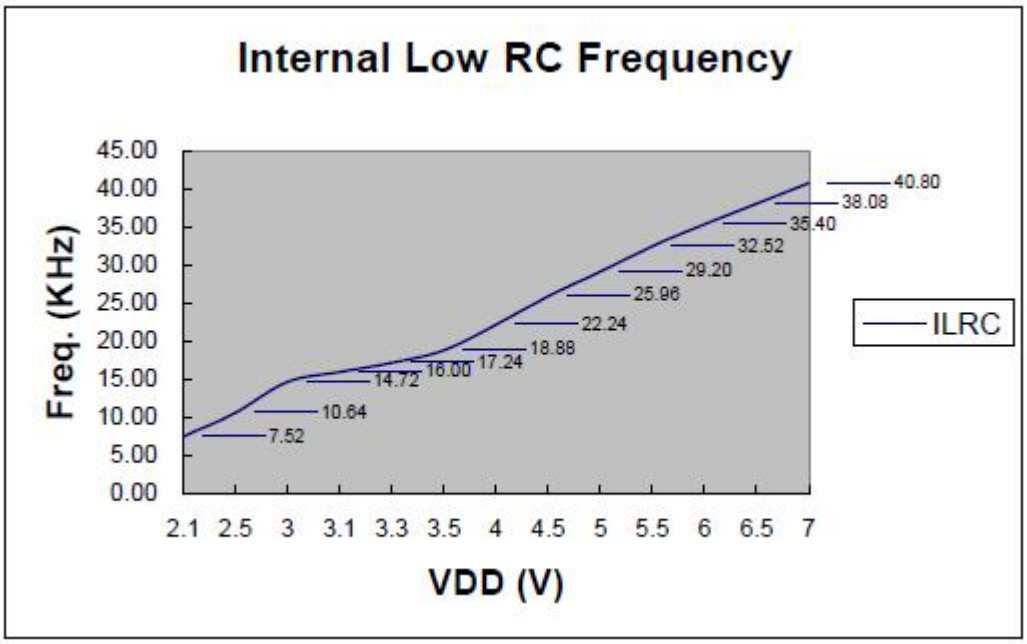
芯片可选择片外时钟信号作为系统时钟，由编译选项High_Clk 控制，从XIN 脚送入。XOUT 为普通的I/O 引脚。



注：外部振荡电路中的GND 必须尽可能的接近芯片的VSS 端口。

4.3.5 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

- Flosc = 内部低速RC 振荡器 (16KHz @3V, 32KHz @5V).
- 低速模式Fcpu = Flosc / 4

系统工作在睡眠模式和绿色模式下看门狗处于无效时，可以停止低速RC 振荡器。如果系统工作频率为32K 且看门狗无效，那么这种情况下只有32K 振荡器处于工作状态，系统功耗相应较低。

- 例：在睡眠模式下，停止内部低速振荡器。

```
B0BSET      FCPUM0
```

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

4.3.5.1 系统时钟测试

在设计过程中，用户可通过软件指令周期对系统时钟速度进行测试。

- 例：外部振荡器的Fcpu 指令周期测试。

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

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

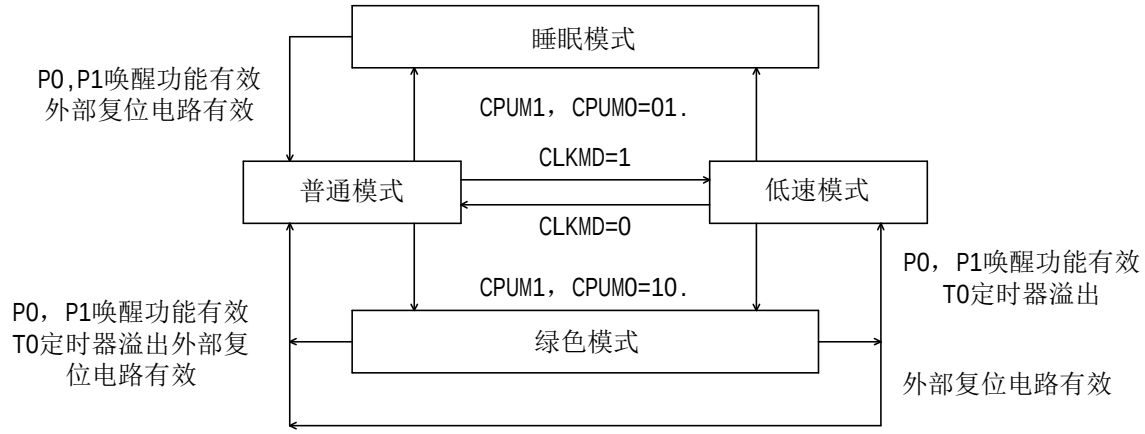
4.4 系统工作模式

4.4.1 概述

芯片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



系统模式切换框图

	普通模式	低速模式	睡眠模式	绿色模式	注释
EHOSC	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	
T0	*有效	*有效	*有效	无效	*T0ENB=1时有效
TC0	*有效	*有效	*有效	无效	*TC0ENB=1时有效
看门狗	由Watch_Dog编译选项控制	由Watch_Dog编译选项控制	由Watch_Dog编译选项控制	由Watch_Dog编译选项控制	
内部中断	都有效	都有效	T0，TC0	都无效	
外部中断	都有效	都有效	都有效	都无效	
唤醒功能	-	-	P0,P1,T0,复位	P0,P1,复位	

EHOSC:外部高速时钟

ILRC:内部低速时钟（3V时16K RC振荡器；5V时32K）

4.4.2 系统模式转换举例

❙ 例：系统由普通/低速模式转换到睡眠模式。

```
B0BSET      FCPUM0
```

注：系统进入睡眠模式后，只有具有唤醒功能的引脚和复位信号能够将系统唤醒并回到普通模式中。

❙ 例：系统由普通模式转换到低速模式。

```
B0BSET      FCLKMD   ; 置CLKMD=1
B0BSET      FSTPHX   ; 外部高速振荡器停振
```

❙ 例：低速模式转换到普通模式（外部高速振荡器始终处于工作状态）。

```
B0BCLR      FCLKMD
```

❙ 例：系统由低速模式转换到普通模式（外部高速振荡器停止工作）。

在外部高速时钟停振的情况下，系统回到普通模式时至少需要延迟 20ms以稳定振荡器

```
        B0BCLR      FSTPHX   ; 启动外部振荡器
        B0MOV       Z, #54   ; 若 VDD = 5V、内部 RC=32KHz，系统延迟0.125ms×162
                             =20.25ms
@@:     DECMS       Z
        JMP         @B
        B0BCLK      FCLKMD ; 系统回到普通模式
```

- 例：系统由普通模式/低速模式进入绿色模式。

```
        B0BSET      FCPUM1
```

注：绿色模式下如果禁止T0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返回到上一个工作模式。

- 例：系统由普通/低速模式进入绿色模式，并开启T0唤醒功能。

```
; 设置T0定时器的唤醒功能
        B0BCLR      FT0IEN   ; 禁止 T0 中断
        B0BCLR      FT0ENB   ; 关闭 T0 定时器
        MOV         A,#20H   ;
        B0MOV       T0M,A    ; T0 时钟＝ Fcpu / 64
        MOV         A,#74H
        B0MOV       T0C,A    ; T0C 初始值= 74H (T0 中断间隔= 10 ms)

        B0BCLR      FT0IEN   ; 禁止 T0 中断
        B0BCLR      FT0IRQ   ; T0中断请求寄存器清零
        B0BSET      FT0ENB   ; 开启T0
; 进入绿色模式
        B0BCLR      FCPUM0
        B0BSET      FCPUM1
```

注：绿色模式下如果允许T0的唤醒功能，则具有唤醒功能的引脚、复位引脚和T0都能够将系统唤醒回到上一工作模式。T0的唤醒周期可编程控制，请注意对T0ENB的设置。

4.4.3 唤醒时间

4.4.3.1 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、P1引脚的电平变化）和内部触发信号（T0溢出信号），具体为：

- 由睡眠模式唤醒到普通模式只能是外部触发；
- 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。

4.4.3.2 唤醒时间

系统进入睡眠模式后，高速时钟停止运行。把系统从睡眠模式下唤醒时，MCU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，等待的这一段时间就称为唤醒时间。唤醒时间结束后，系统才进入到普通模式。

注：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：

唤醒时间=1/Fosc*2048(sec)+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：

  唤醒时间=1/Fosc*2048=0.512ms (Fosc=4MHz)

  总的唤醒时间=0.512ms+振荡器启动时间

4.4.3.3 PWM唤醒控制寄存器

系统处于睡眠模式或绿色模式时，具有唤醒功能的 I/O端口能够将系统唤醒并进入到普通模式。P0和P1都具有上述唤醒功能，其中 P0的唤醒功能始终有效，而 P1则由寄存器 P1W控制。

0C0H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P1W	-	-	P15W	P14W	P13W	P12W	P11W	P10W
读/写	-	-	W	W	W	W	W	W
复位后	-	-	0	0	0	0	0	0

Bit[5:0] P10W~P15W:P1唤醒功能控制位

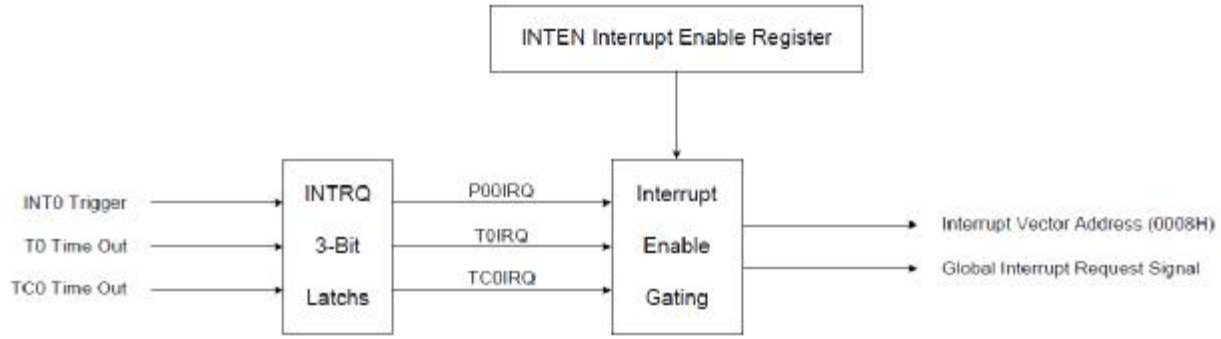
0=P1n 唤醒功能无效

1=P1n 唤醒功能有效

4.5 中断

4.5.1 概述

芯片提供三种中断源：两个内部中断（T0/TC0）和一个外部中断（ INT0）。系统从睡眠模式进入高速普通模式时，外部中断能够将芯片唤醒。一旦程序进入中断，寄存器 STKP的位GIE将被清零以避免再次响应其它中断。系统退出中断后，GIE置“1”，以响应下一个中断。中断请求存放在寄存器INTRQ中。



注：程序响应中断时，位 GIE 必须处于有效状态。

4.5.2 中断请求使能寄存器INTEN

中断请求控制寄存器 INTEN包括2个内部中断和1个外部中断，当有效位被置为1后，系统进入该中断服务程序，程序计数器入栈，程序转至ORG 8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	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INTEN	-	-	TC0IEN	T0IEN	-	-	-	P00IEN
读/写	-	-	R/W	R/W	-	-	-	R/W
复位后	-	-	0	0	-	-	-	0

Bit 0 P00IEN： P0.0外部中断(INT0) 控制位
0 = 禁止
1 = 允许

Bit 4 T0IEN： T0中断控制位
0 = 禁止
1 = 允许

Bit 5 TC0IEN： TC0中断控制位
0 = 禁止
1 = 允许

4.5.3 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	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INTRQ	-	-	TC0IRQ	T0IRQ	-	-	-	P00IRQ
读/写	-	-	R/W	R/W	-	-	-	R/W
复位后	-	-	0	0	-	-	-	0

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

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

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

4.5.4 GIE全局中断

只有当全局中断控制寄存器 GIE置1的时候程序才能响应中断请求。

0DFH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STKP	GIE	-	-	-	-	-	STKPB1	STKPB0
读/写	R/W	-	-	-	-	-	R/W	R/W
复位后	-	-	-	-	-	-	1	1

Bit 7 GIE： 全局中断控制位

0 = 全局中断无效

1 = 使能全局中断

❙ 例：设置全局中断控制位 (GIE)。

```
B0BSET      FGIE    ; 使能 GIE
```

注：在所有中断中，GIE都必须处于使能状态。

4.5.5 PUSH，POP处理

有中断请求发生并被响应后，程序转至ORG 8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     8
        JMP     INT_SERVICE
        ORG     10H
START:
        …
INT_SERVICE:
        PUSH        ; 保存ACC和PFLAG
        …
        …
        POP         ; 恢复ACC和PFLAG
        RETI        ; 退出中断
        …
        ENDP
```

4.5.6 INT0(P0.0)中断

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

注：P0.0的中断触发方式由PEDGE控制。

0BFH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PEDGE	-	-	-	P00G1	P00G0	-	-	-

读/写	-	-	-	R/W	R/W	-	-	-
复位后	-	-	-	1	0	-	-	-

Bit[4:3] P00G[1:0]: P0.0中断触发控制位

00=保留

01=上升沿触发

10=下降沿触发

11= 上升/下降沿触发（电平触发）

- 例：INT0中断请求设置，电平触发。

```
MOV       A, #18H
B0MOV     PEDGE, A     ; INT0置为电平触发
B0BSET    FP00IEN      ; INT0中断开放
B0BCLR    FP00IRQ      ; INT0中断请求标志清零
B0BSET    FGIE         ; 允许GIE
```

- 例：INT0中断服务程序。

```
        ORG       8          ;
        JMP       INT_SERVICE
INT_SERVICE:
        …                    ; ACC和PFLAG入栈保护
        B0BTS1    FP00IRQ    ; 检测P00IRQ
        JMP       EXIT_INT   ; P00IRQ = 0, 退出中断

        B0BCLR    FP00IRQ    ; P00IRQ 清零
        …                    ;
        …
EXIT_INT:
        …                    ; ACC和PFLAG出栈恢复
        RETI                 ; 退出中断
```

4.5.7 T0中断

计数器T0C溢出时，不管T0IEN 处于何种状态，T0中断请求寄存器T0IRQ都置“1”。此时，如果T0IEN＝1 ，那么系统将响应T0中断进入相应的中断服务子程序。如果这时T0IEN=0，那么系统并不响应该T0中断请求。对于多中断情形，尤其需要注意上述条件。

- 例：设置T0中断。

```
B0BCLR    FT0IEN       ; 禁止T0中断
B0BCLR    FT0ENB       ; 关闭T0
MOV       A, #20H      ;
B0MOV     T0M, A       ; 设置T0时钟= Fcpu / 64
MOV       A, #74H      ; 初始化T0C = 74H
B0MOV     T0C, A       ; 设置T0间隔时间= 10 ms
```

```
                B0BSET      FT0IEN      ; 开启T0中断
                B0BCLR      FT0IRQ      ; T0IRQ清零
                B0BSET      FT0ENB      ; 开启定时器T0

                B0BSET      FGIE        ; 开放GIE
```

❙ 例： T0中断服务程序。

```
                ORG         8
                JMP         INT_SERVICE
INT_SERVICE:
                …                       ; ACC和PFLAG入栈保护
                B0BTS1      FT0IRQ      ; 检查是否有T0中断请求标志
                JMP         EXIT_INT    ; T0IRQ = 0, 退出中断
                B0BCLR      FT0IRQ      ; 清T0IRQ
                MOV         A, #74H
                B0MOV       T0C, A      ;
                …
                …
EXIT_INT:
                …                       ; ACC和PFLAG出栈恢复

                RETI                    ; 退出中断
```

4.5.8 TCO中断

TC0C溢出时，不管TC0IEN是否开启，TC0IRQ都会被置“1”。如果 TC0IRQ=1且C0IEN=1 ，则系统将响应 TC0的中断请求；如果TC0IRQ=1 而TC0IEN=0，则系统并不会响应TC0的中断请求。对于多中断情形，尤其需要注意上述条件。

❙ 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, #74H     ; TC0C初始值＝74H
                B0MOV       TC0C, A     ; TC0间隔= 10 ms
                B0BSET      FTC0IEN     ; 允许TC0中断
                B0BCLR      FTC0IRQ     ; 清TC0中断请求标志
                B0BSET      FTC0ENB     ;
                B0BSET      FGIE        ; 允许GIE
```

❙ 例： TC0中断服务程序。

```
                ORG         8           ;
                JMP         INT_SERVICE
INT_SERVICE:
```

```
                …                       ; ACC和PFLAG入栈保护
                B0BTS1      FTC0IRQ     ; 检查是否有TC0中断请求标志
                JMP         EXIT_INT    ; TC0IRQ = 0, 退出中断
                B0BCLR      FTC0IRQ     ; 清TC0IRQ
                MOV         A, #74H
                B0MOV       TC0C, A     ; 清TC0C
                …                       ; TC0中断程序
                …
EXIT_INT:
                …                       ; ACC和PFLAG 出栈恢复
                RETI                    ; 退出中断
```

4.5.9 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。处理这种多中断情况时，必须预先对各中断请求设置不同的优先级别。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。各中断触发事件列表如下：

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

多个中断同时发生时，需要注意的是：首先，必须预先设定好各中断的优先级。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         8           ;
                JMP         INT_SERVICE
INT_SERVICE:
                …                       ; ACC 和 PFLAG 入栈保护

INTP00CHK:                              ; 检查是否有 INT0 中断请求
                B0BTS1      FP00IEN     ; 检查是否允许 P00 中断
                JMP         INTT0CHK    ; 进入下一个中断
                B0BTS0      FP00IRQ     ; 检查是否有 P00 中断请求
                JMP         INTP00      ; 进入 INT0 中断程序
INTT0CHK:                               ; 检查是否有 T0 中断请求
                B0BTS1      FT0IEN      ; 检查是否允许 T0 中断
                JMP         INTTC0CHK   ; 进入下一个中断
                B0BTS0      FT0IRQ      ; 检查是否有 T0 中断请求
                JMP INTT0 ; 进入 T0 中断程序
INTTC0CHK:                              ; 检查是否有 TC0 中断请求
                B0BTS1      FTC0IEN     ; 检查是否允许 TC0 中断
```

```
                JMP         INT_EXIT    ;
                B0BTS0      FTC0IRQ     ; 检查是否有 TC0 中断请求
                JMP         INTTC0      ; 进入 TC0 中断程序
INT_EXIT:
                …                       ; ACC 和 PFLAG 出栈恢复
                RETI                    ; 退出中断
```

4.6 I/O口

4.6.1I/O 模式

对寄存器PnM 编程设置可选择各端口的数据传送方向。

0B8H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P0M	-	-	-	-	-	-	-	P00M
读/写	-	-	-	-	-	-	-	R/W
复位后	-	-	-	-	-	-	-	0

0C1H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P1M	-	-	-	P14M	P13M	P12M	P11M	P10M
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

0C5H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P5M	P57M	P56M	P55M	P54M	P53M	P52M	P51M	P50M
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[7:0] PnM[7:0]：Pn 模式控制位 (n = 0~5)。

0 = Pn设置为输入模式；

1 = Pn 设置为输出模式。

- 注:
  1. 用户可通过位操作指令(B0BSET, B0BCLR)对I/O 口进行编程控制；
  2. P1.5 只能作为输入引脚，寄存器 P1M.5 的值保持为“1”。

- 例：I/O 模式设置。

```
                CLR         P0M             ; 所有端口设为输入模式
                CLR         P1M
                CLR         P5M
                MOV         A, #0FFH        ; 所有端口设为输出模式
                B0MOV       P0M, A
                B0MOV       P1M, A
```

```
B0MOV       P5M, A
B0BCLR      P1M.2          ; P1.2 设为输入模式
B0BSET      P1M.2          ; P1.2 设为输出模式
```

4.6.2 I/O 口上拉电阻寄存器

0E0H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P0UR	-	-	-	-	-	-	-	P00R
读/写	-	-	-	-	-	-	-	W
复位后	-	-	-	-	-	-	-	0

0E1H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P1UR	-	-	-	P14R	P13R	P12R	P11R	P10R
读/写	-	-	-	W	W	W	W	W
复位后	-	-	-	0	0	0	0	0

0E5H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P5UR	P57R	P56R	P55R	P54R	P53R	P52R	P51R	P50R
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

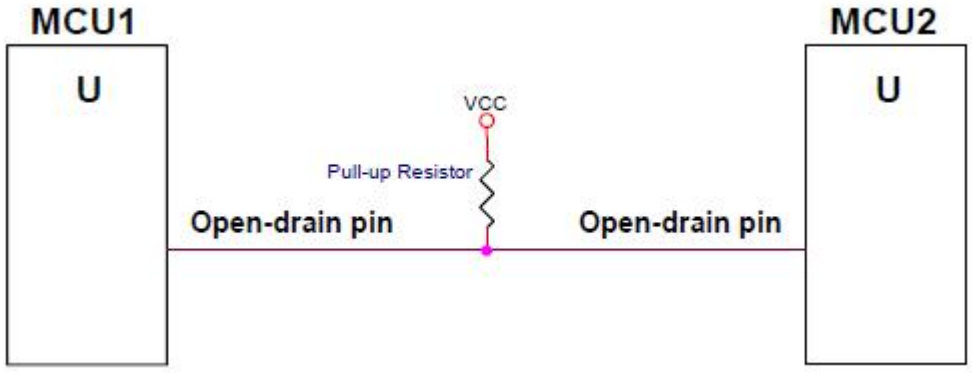
- 注：**P1.5** 只能作为输入模式，且无内部上拉电阻功能。因此**P1UR.5** 必须保持**“1“**。

- 例：I/O 上拉电阻寄存器设置。

```
MOV         A, #0FFH        ; 使能P0, 1, 5 上拉电阻。
B0MOV       P0UR, A
B0MOV       P1UR, A
B0MOV       P5UR, A
```

4.6.3 I/O 漏极开路寄存器

P1.0 有内置漏极开路功能，当使能该功能时，P1.0 必须被置为输出模式。漏极开路的外部电路如下，图中的上拉电阻必不可少，漏极开路的输出高电平由上拉电阻驱动，输出低电平时为灌电流。



0E0H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P1OC	-	-	-	-	-	-	-	P10OC

读/写	-	-	-	-	-	-	-	W
复位后	-	-	-	-	-	-	-	0

Bit 0 P10OC：P1.0 漏极开路控制位

0 = 禁止漏极开路

1 = 漏极开路

- 例：P1.0 设置为漏极开路模式，输出高电平。

```
B0BSET      P1.0        ; P1.0 置1
B0BSET      P10M        ; 设置P1.0 输出模式
MOV         A, #01H     ; P1.0 置为漏极开路模式
B0MOV       P1OC, A
```

- 注：P1OC 为只写寄存器，所以P10OC 只能用指令“MOV “进行设置。

- 例：禁止P1.0 漏极开路，输出低电平。

```
MOV         A, #0
B0MOV       P1OC, A
```

- 注：禁止P1.0 的漏极开路功能后，P1.0 返回上一个I/O 模式。

4.6.4 I/O 口数据寄存器

0D0H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P0	-	-	-	-	-	-	-	P00
读/写	-	-	-	-	-	-	-	R/W
复位后	-	-	-	-	-	-	-	0

0D1H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P1	-	-	P15	P14	P13	P12	P11	P10
读/写	-	-	R	R/W	R/W	R/W	R/W	R/W
复位后	-	-	0	0	0	0	0	0

0D5H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P5	P57	P56	P55	P54	P53	P52	P51	P50
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

- 注：当外部复位有效时，P15 的值保持为“1”。

- 例：从输入端口读取数据。

```
B0MOV       A, P0       ; 从P0 读数据
```

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

- 例：数据写到输出端口。

```
MOV       A, #0FFH
B0MOV     P0, A
B0MOV     P1, A
B0MOV     P5, A
```

- 例：数据写到输出端口。

```
B0BSET    P1.3           ; P1.3 和P5.5 置为 “1”.
B0BSET    P5.5
B0BCLR    P1.3           ; P1.3 和P5.5 置为 “0”.
B0BCLR    P5.5
```

4.7 定时器

4.7.1 看门狗定时器

看门狗定时器WDT 用于控制程序的正常执行。如果由于干扰程序进入了未知状态，WDT 溢出，MCU 复位。看门狗的工作模式由编译选项控制，其时钟源来自内部低速振荡器(16KHz @3V, 32KHz @5V)。

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。

寄存器WDTR 控制对看门狗的清零：WDTR 置为5AH 即可将看门狗清零。

0CCH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
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作用。

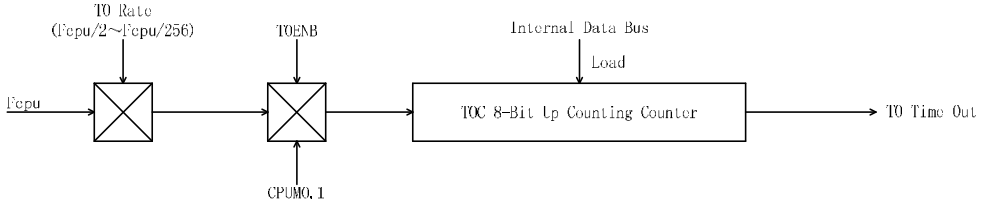
- 例：如下是对看门狗定时器的操作，在主程序开头对看门狗清零。

```
main:
        …                       ; 检测 I/O 口状态
        …                       ; 检测 RAM 内存
Err:    JMP $                   ; I/O 或RAM 出错，不对看门狗清零，并等待看门狗计时溢出
                                ; I/O 和RAM 正常，看门狗清零
                                ;
        @RST_WDT                ;
        …
        CALL SUB1
        CALL SUB2
        …
        …
        JMP MAIN
```

4.7.2 定时器T0

4.7.2.1 概述

8-bit 二进制计数器T0 可作为定时器使用。T0 溢出时（由FFH 计到00H），T0 在继续计数的同时给出一个超时信号，该信号即是T0 中断触发信号。计数器T0 主要有以下功能：



4.7.2.2 模式寄存器T0M

0D8H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

T0M	T0ENB	T0rate2	T0rate1	T0rate0	-	-	-	-
读/写	R/W	R/W	R/W	R/W	-	-	-	-
复位后	0	0	0	0	-	-	-	-

Bit [6:4] T0RATE[2:0]：T0 时钟频率控制位

000 = fcpu/256.

001 = fcpu/128.

…

110 = fcpu/4.

111 = fcpu/2.

Bit 7 T0ENB：T0 计数控制位

0 = 关闭T0 计数器

1 = 开启T0 计数器

4.7.2.3 计数寄存器T0C

T0C 用于控制T0 的间隔时间。

0D9H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T0C	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

T0C 初始值计算公式如下：T0C 初始值= 256 - (T0 中断间隔时间 * 输入时钟)

- 例：断间隔时间设置为10ms，高速时钟选择外部4MHz，Fcpu=Fosc/4，T0RATE=010 (Fcpu/64)。

T0C 初始值 = 256 - (T0 中断间隔时间 * 输入时钟)

= 256 - (10ms * 4MHz / 4 / 64)

= 256 - (10-2 * 4 * 106 / 4 / 64)

= 100

= 64H

T0 定时间隔列表

T0RATE	T0CLOCK	高速模式 (Fcpu = 4MHz / 4)		低速模式 (Fcpu = 32768Hz / 4)	
		最大溢出间隔 Max	One step = max/256	最大溢出间隔 Max	One step = 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

4.7.2.4 T0 操作时序

- T0 停止计数，禁止T0 中断并将T0 中断请求标志清零。

```
B0BCLR    FT0ENB
B0BCLR    FT0IEN
B0BCLR    FT0IRQ
```

- 设置T0 计时速率。

```
MOV       A,#0xxx00b    ;通过设置寄存器TOM 的bit4~bit6 可控制T0 的计数速率，设置范围x000xxxxb~x111xxxxb
B0MOV     T0M,A
```

- 设置T0 中断间隔时间。

```
MOV       A,#7FH
B0MOV     T0C,A
```

- 设置T0 工作模式。

```
B0BSET FT0IEN
```

- 开启T0。

```
B0BSET FT0ENB
```

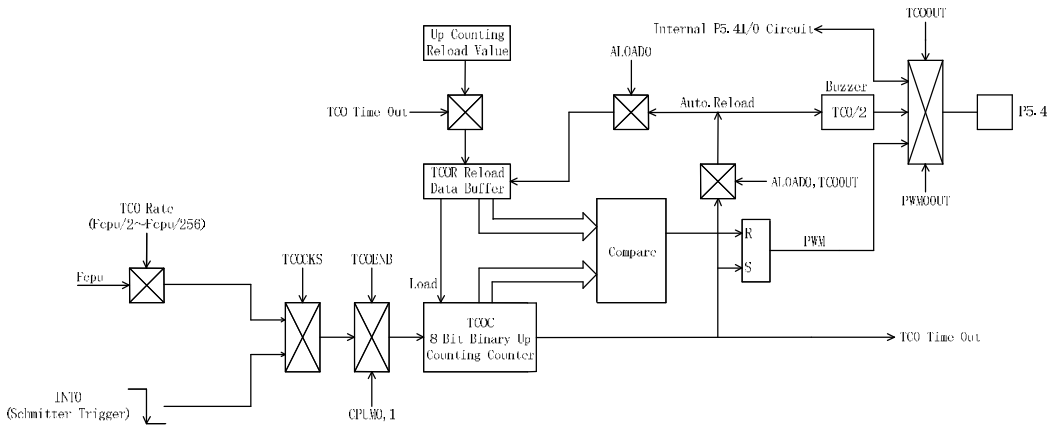
4.7.3 定时/计数器TC0

4.7.3.1 概述

8-bit 二进制增量计数器TC0 可根据实际需要，分别采用内部时钟和外部时钟两种时钟源作为计数时基。其中，内部时钟来自Fcpu，外部时钟INT0 由P0.0 引脚（下降沿触发）输入。寄存器TC0M 控制具体时钟源的选择。通常，TC0 计数到0xFF 变为0X00 时，系统给出溢出信号触发TC0 中断，同时开始重新计数。在PWM 模式下，TC0 的溢出由ALOAD0和TC0OUT 控制的PWM 周期决定。

TC0 的主要作用如下：

- 8-bit 可编程定时器：根据选定的时钟频率在特定时间产生中断信号；
- 外部事件计数：通过对INT0 口输入信号下降沿的检测，统计系统“事件”数；
- 蜂鸣器输出
- PWM 输出



4.7.3.2 模式寄存器TC0M

0DAH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TC0M	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 = 开放PWM 输出，PWM 输出占空比由TC0OUT 和 ALOAD0 控制。

Bit 1 TC0OUT：TC0 超时输出信号控制。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：自动重载控制。仅当PWM0OUT = 0 时有效。
0 = 禁止TC0 自动重载；
1 = 允许TC0 自动重载。

Bit 3 TC0CKS：TC0 时钟源控制位
0 = 内部时钟(Fcpu or Fosc)；
1 = 外部时钟，由P0.0/INT0 输入。

Bit [6:4] TC0RATE[2:0]：TC0 内部时钟选择控制
000 = fcpu/256；
001 = fcpu/128；
…
110 = fcpu/4；
111 = fcpu/2。

Bit 7 TC0ENB：TC0 计数控制位。
0 = 禁止TC0 定时器；
1 = 开放TC0 定时器。

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（P0.0IRQ＝0）。

4.7.3.3 计数寄存器TC0C

TC0C 控制TC0 的时间间隔。

0DBH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TC0C	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

TC0C 初始值的计算公式如下：TC0C 初始值 = N - (TC0 中断间隔时间 * 输入时钟)

N 为TC0 二进制计数范围，TC0 共有六种工作模式：TC0 定时器、TC0 事件计数器、TC0 Fcpu 时钟源、TC0 Fosc时钟源、PWM 模式和无PWM 模式。各模式下参数的设定如下表所示：

TC0CKS	PWM0	ALOAD0	TC0OUT	N	TC0C 范围	TC0C 二进制计数范	注释
0	0	x	x	256	0x00~0xFF	00000000b~ 11111111b	每计数256 次溢出
	1	0	0	256	0x00~0xFF	00000000b~ 11111111b	每计数256 次溢出
	1	0	1	64	0x00~0x3F	xx000000b~ xx111111b	每计数64 次溢出
	1	1	0	32	0x00~0x1F	xxx00000b~ xxx11111b	每计数32 次溢出
	1	1	1	16	0x00~0x0F	xxxx0000b~ xxxx1111b	每计数16 次溢出
1	-	-	-	256	0x00~0xFF	00000000b~ 11111111b	每计数256 次溢出

- 例：TC0 中断时间设为10ms，时钟源选择Fcpu(TC0KS = 0) ，无PWM 输出(PWM0=0)，高速时钟 = 4MHz. Fcpu= Fosc/4，TC0RATE = 010 (Fcpu/64)。

TC0C 初始值 = N - (TC0 中断时间 * 输入时钟)
= 256 - (10ms * 4MHz / 4 / 64)
= 256 - (10-2 * 4 * 106 / 4 / 64)
= 100
= 64H

T0 定时间隔列表

TC0RATE	TC0CLOCK	高速模式 (Fcpu = 4MHz / 4)		低速模式 (Fcpu = 32768Hz / 4)	
		最大溢出间隔 Max	单步间隔时间 = max/256	最大溢出间隔Max	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

4.7.3.4 自动重载寄存器TC0R

TC0 的自动重载功能由TC0M 的ALOAD0 位控制。当TC0C 溢出时，TC0R 的值自动装入TC0C 中。这样，用户在使用的过程中就不需要在中断中重新赋值。

TC0 为双重缓存器结构。若程序对TC0R 进行了修改，那么修改后的TC0R 值首先被暂存在TC0R 的第一个缓存器中，直到TC0 溢出后，才被真正存入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，系统自动开启自动重装功能。位寄存器ALOAD0 用于控制溢出范围。

0CDH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TC0R	TC0R7	TC0R6	TC0R5	TC0R4	TC0R3	TC0R2	TC0R1	TC0R0
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

TC0R 初始值计算公式如下：TC0R 初始值 = N - (TC0 中断间隔时间 * 输入时钟)

上式中，N 为TC0 的最大计数范围。TC0 的溢出时间有如下六种可能情况：

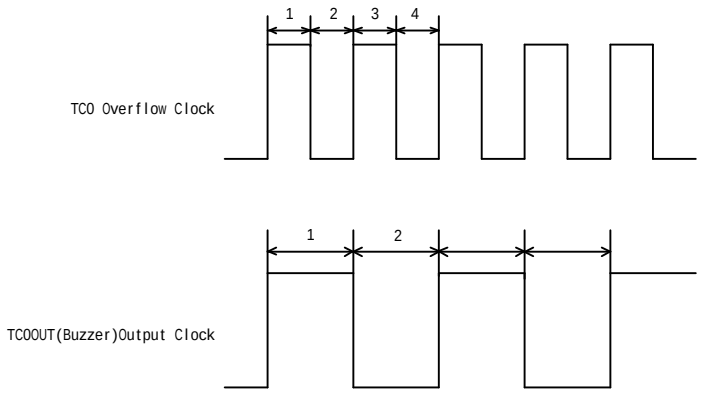
TC0CKS	PWM0	ALOAD0	TC0OUT	N	TC0C 范围	TC0C 二进制计数范
0	0	x	x	256	0x00~0xFF	00000000b~11111111b
	1	0	0	256	0x00~0xFF	00000000b~11111111b
	1	0	1	64	0x00~0x3F	xx000000b~xx111111b
	1	1	0	32	0x00~0x1F	xxx00000b~xxx11111b
	1	1	1	16	0x00~0x0F	xxxx0000b~xxxx1111b
1	-	-	-	256	0x00~0xFF	00000000b~11111111b

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

TC0R 初始值 = N - (TC0 中断间隔时间 * 输入时钟源)
= 256 - (10ms * 4MHz / 4 / 64)
= 256 - (10-2 * 4 * 106 / 4 / 64)
= 100
= 64H

4.7.3.5 TC0 时钟频率输出(蜂鸣器输出)

蜂鸣器输出即TC0 计数/定时器的频率输出功能。对TC0 时钟频率进行适当设置，TC0 经过2 分频后作为TC0OUT并输出到P5.4 引脚，同时，P5.4 的普通I/O 功能自动被屏蔽。TC0OUT 输出波形如下：



- 例：设置TC0OUT (P5.4)。其中，外部高速时钟 = 4MHz，TC0OUT 频率 = 0.5KHz，TC0 频率 = 1KHz，TC0 时钟源采用内部时钟Fcpu/4，TC0RATE2~TC0RATE1 = 110， TC0C = TC0R = 131。

```
MOV       A,#01100000B
B0MOV     TC0M,A            ; TC0 速率Fcpu/4
MOV       A,#131            ; 自动加载参考值设置
B0MOV     TC0C,A
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    FALOAD1           ; 允许TC0 自动重载功能
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。

4.7.3.6 TC0 操作举例

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。

- 例：停止TC0 计数器，禁止TC0 中断并将TC0 中断请求标志清零。

```
B0BCLR    FTC0ENB
B0BCLR    FTC0IEN
B0BCLR    FTC0IRQ
```

- 例：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 速率，设置范围为
                              x000xxxxb~x111xxxxb
B0MOV     TC0M,A            ; 禁止TC0 中断
```

- 例：设置TC0 的时钟源。

```
B0BCLR    FTC0CKS           ; 选择内部时钟
```

or

```
B0BSET      FTC0CKS          ; 选择外部时钟
```

❙ 例：TC0 自动重载模式设置。

```
B0BCLR      FALOAD0          ; 开放自动重载功能
```

or

```
B0BSET      FALOAD0          ; 禁止TC0 自动重载
```

❙ 例：**TC0** 中断间隔时间设置。

```
MOV         A,#7FH           ; TC0 模式决定TC0C 和TC0R 的值
B0MOV       TC0C,A           ; 设置 TC0C
B0MOV       TC0R,A           ; 设置TC0R
B0BCLR      FALOAD0          ; ALOAD0, TC0OUT = 00, PWM 周期 = 0~255
B0BCLR      FTC0OUT
```

or

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

or

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

or

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

❙ 例：设置**TC0** 模式。

```
B0BSET      FTC0IEN          ; 开放TC0 中断功能
```

or

```
B0BSET      FTC0OUT          ; 开放TC0OUT (蜂鸣器) 功能
```

or

```
B0BSET      FPWM0OUT         ; 开放PWM 功能
```

❙ 例：开启**TC0**。

```
B0BSET      FTC0ENB          ; 开放TC0
```

4.7.4 PWM0

4.7.4.1 概述

PWM 信号输出到PWM0OUT（P5.4 引脚），TC0OUT 和ALOAD0 控制PWM 输出的量程（256、64、32 和16）。8-bit 计数器TC0C 计数过程中不断与TC0R 相比较，当TC0C 计数到两者相等时，PWM 输出低电平，当TC0C 再次从零开始计数时，PWM 被强制输出高电平。PWM0 输出占空比 = TC0R/计数量程（计数量程 = 256、64、32 或16）。

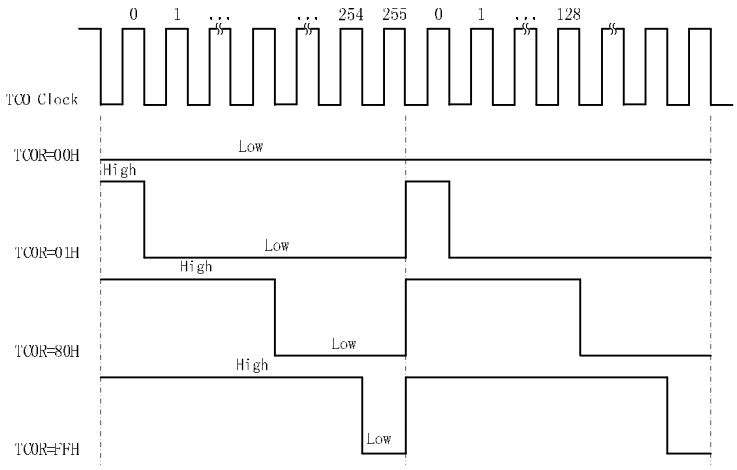
参考寄存器保持输入00H 可使PWM 的输出长时间维持在低电平，通过修改TC0R 可改变PWM 输出占空比。

❙ 注：TC0 为双重缓存器结构，调整TC0R 的值可以改变PWM 的输出占空比。用户可随时改变

TC0R 的值，但是只有在TC0 溢出后，这一修改值才真正被写入TC0R 中。

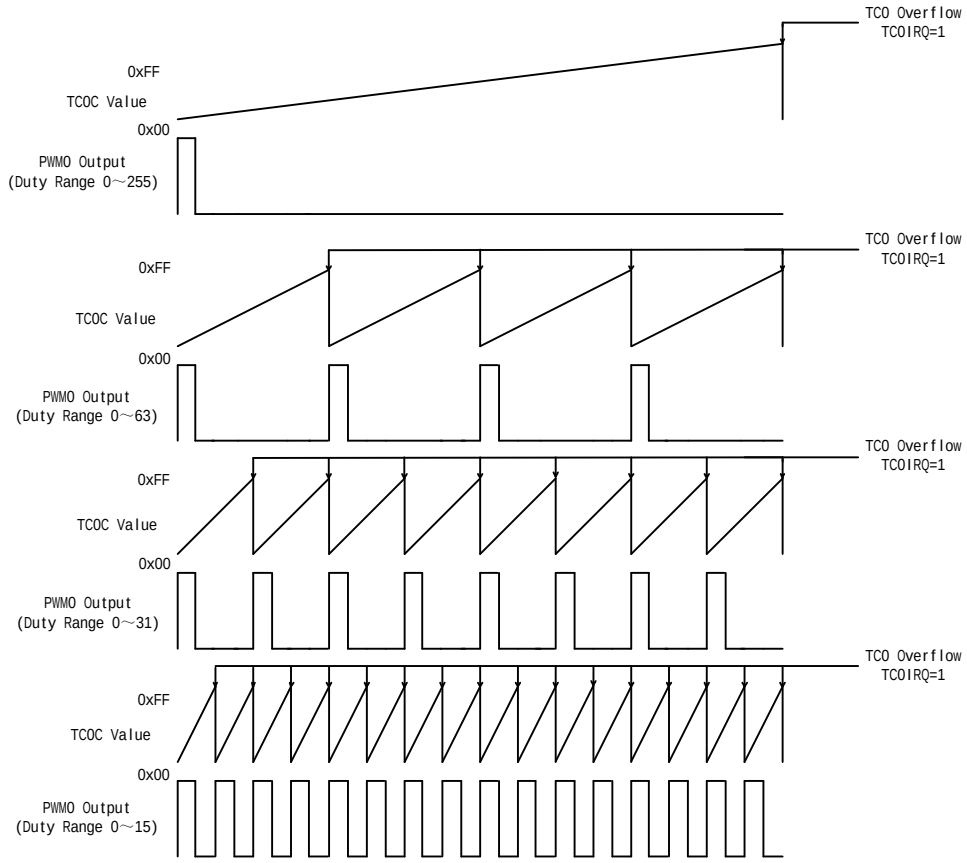
ALOAD0	TC0OUT	PWM 占空比范围	TC0C 有效值	TC0R 有效值	TC0C 范围	MAX. PWM 输出频率 (Fcpu = 4MHz)	注释
0	0	0/256~255/256	0x00~0xFF	0x00~0xFF	0x00~0xFF	7.8125K	每计数256 次溢出
0		0/64~63/64	0x00~0x3F	0x00~0x3F	0x00~0xFF	31.25K	每计数64次溢出
1	0	0/32~31/32	0x00~0x1F	0x00~0x1F	0x00~0x3F	62.5K	每计数32次溢出
1	1	0/16~15/16	0x00~0x0F	0x00~0x0F	0x00~0x1F	125K	每计数16次溢出

PWM 输出占空比随TC0R 的变化而变化：0/256~255/256。



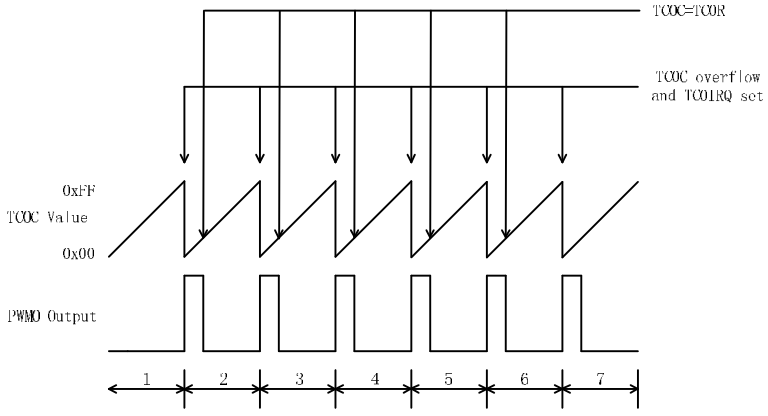
4.7.4.2 TC0IRQ 和PWM 输出占空比

在PWM 模式下，TC0IRQ 的频率与PWM 的占空比有关，具体情况如下图所示：

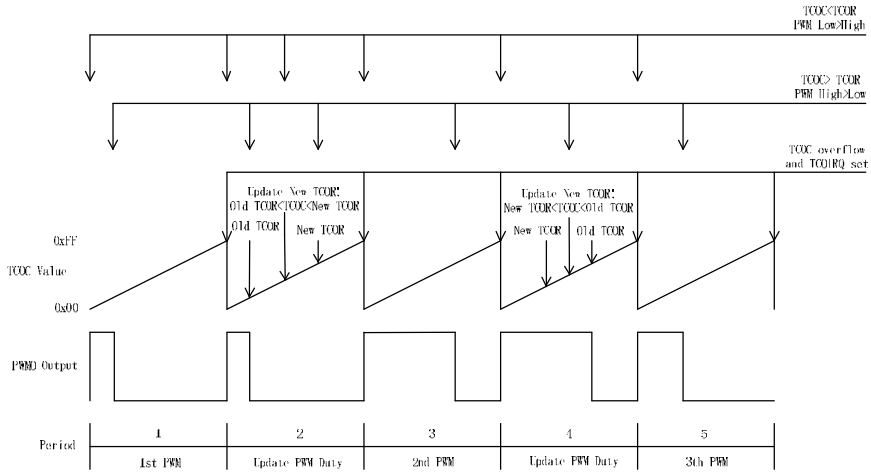


4.7.4.3 PWM 输出占空比与TC0R 的变化

在PWM 模式下，系统随时比较TC0C 和TC0R 的异同。若TC0C<TC0R，PWM 输出高电平，反之则输出低电平。当TC0C 发生改变的时候，PWM 将在下一周期改变输出占空比。如果TC0R 保持恒定，那么PWM 输出波形也保持稳定。



上图所示是TC0R 恒定时的波形。每当TC0C 溢出时，PWM都输出高电平，TC0C≧TC0R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。

4.7.4.4 PWM 编程举例

- 例：PWM 输出设置。外部高速振荡器输出频率 = 4MHZ，Fcpu = Fosc/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 = Fcpu/4，TC0RATE2~TC0RATE1 = 110，TC0C = TC0R = 30。

```
MOV       A,#01100000B
B0MOV     TC0M,A ; TC0 速率＝Fcpu/4
MOV       A,#30 ; PWM 输出占空比＝30/256
B0MOV     TC0C,A
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
```

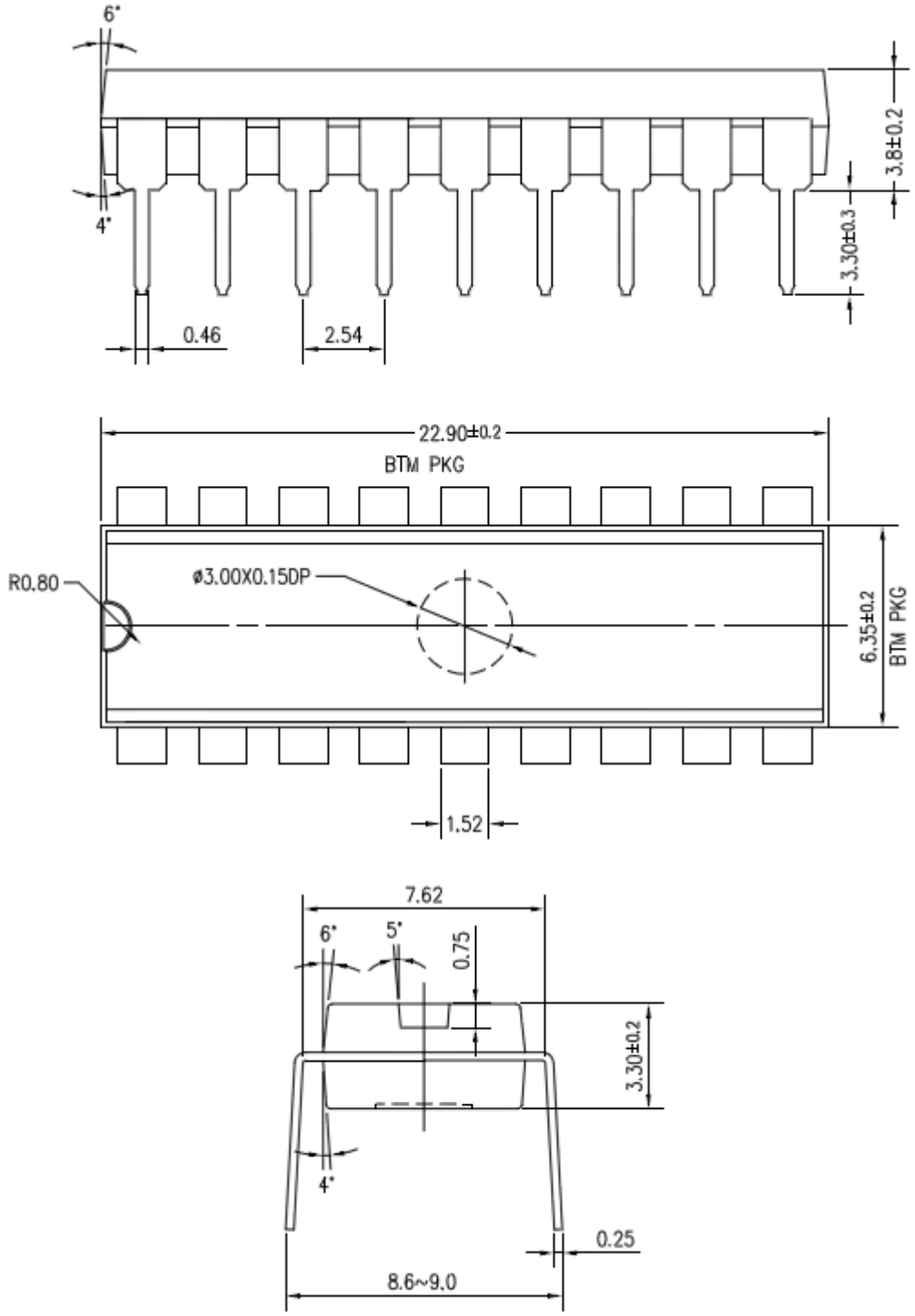
- 注：TC0R 为只写寄存器，不能用INCMS 和DECMS 指令对其进行操作。

- 例：改变TC0R 的内容。

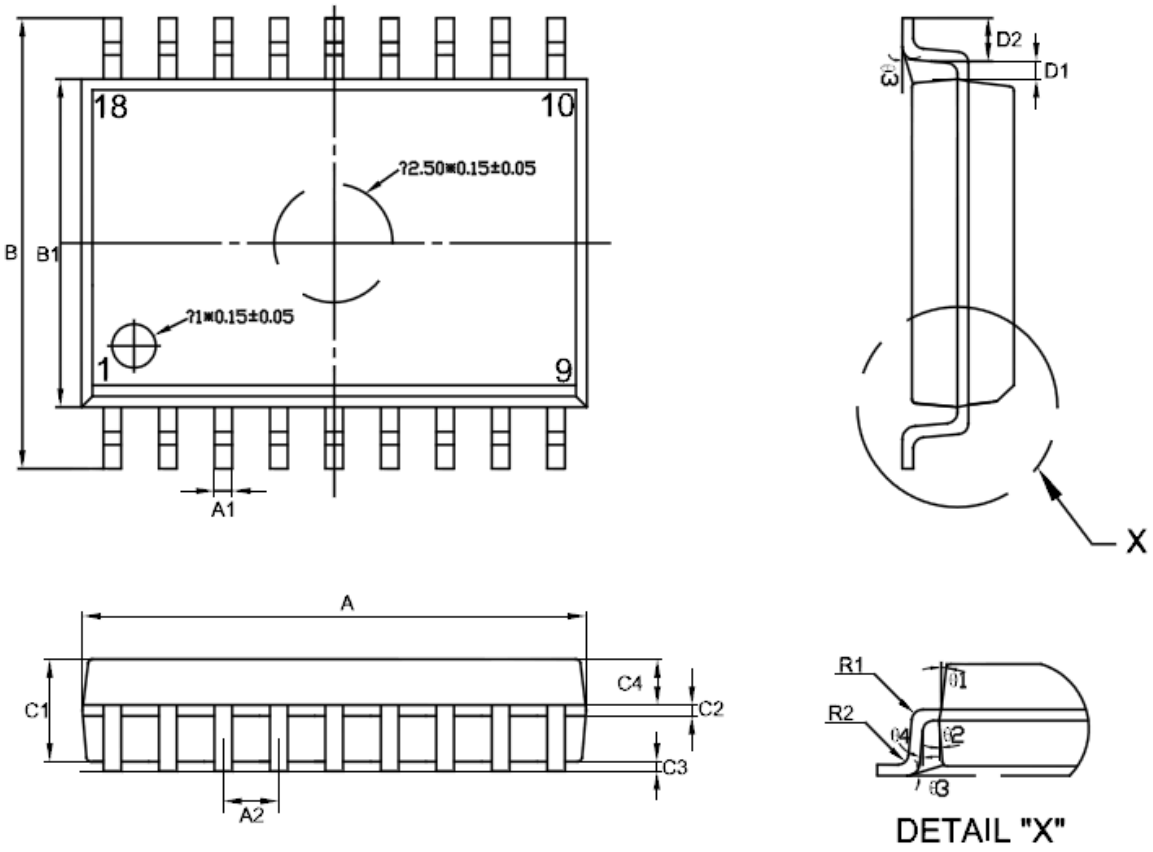
```
MOV       A, #30H
B0MOV     TC0R, A
INCMS     BUF0
NOP
B0MOV     A, BUF0
B0MOV     TC0R, A
```

5、封装尺寸与外形图

5. 1、DIP18 外形图与封装尺寸



5. 2、SOP18 外形图与封装尺寸



尺寸 / 标注	最小（mm）	最大（mm）	尺寸 / 标注	最小（mm）	最大（mm）
A	11.35	11.68	D1	0.5TYP	
A1	0.36	0.51	D2	0.7	0.97
A2	1.27TYP		R1	0.25TYP	
B	10.01	10.64	R2	0.25TYP	
B1	7.37	7.62	θ1	7° TYP	
C1	2.2	2.6	θ2	7° TYP	
C2	0.204~0.33		θ3	0°	8°
C3	0.10~0.25		θ4	10° TYP	
C4	1.0TYP				

6、声明及注意事项：

6.1、产品中有毒有害物质或元素的名称及含量

部件名称	有毒有害物质或元素

	铅（Pb）	汞（Hg）	镉（Cd）	六阶铬（Cr(Ⅵ)）	多溴联苯（PBBs）	多溴联苯醚(PBDEs）
引线框	○	○	○	○	○	○
塑封树脂	○	○	○	○	○	○
芯片	○	○	○	○	○	○
内引线	○	○	○	○	○	○
装片胶	○	○	○	○	○	○
说明	○：表示该有毒有害物质或元素的含量在 SJ/T11363-2006 标准的检出限以下。 ×：表示该有毒有害物质或元素的含量超出 SJ/T11363-2006 标准的限量要求。					

6.2 注意

在使用本产品之前建议仔细阅读本资料；

本资料中的信息如有变化，恕不另行通知；

本资料仅供参考，本公司不承担任何由此而引起的任何损失；

本公司也不承担任何在使用过程中引起的侵犯第三方专利或其它权利的责任。