SN8P2318 系列

用户参考手册

Version 1.3

SN8P2318
SN8P2317

SONiX 8 位单片机

修改记录

版本	实际	修改说明
VER 1.0	2010.05	初版。
VER1.1	2011.06	1、增加 SN8P2317 的相关信息。 2、修改开发工具的部分内容。 3、增加特性曲线图章节。 4、修正笔误。 5、增加-40°C~ +85°C 电气特性。
VER 1.2	2013.03	1、调整“电气特性”中工作温度范围：由 0~70°C 改为-20°C ~ + 85°C。
VER 1.3	2013.07	1、调整烧录引脚配置章节的引脚分配。 2、VLCD 和 VDD 引脚必须短接。

目 录

1 产品简介

1.1 功能特性

- **存储器配置**
  ROM：4K * 16 位。
  RAM：128 * 8 位。

- **8 层堆栈缓存器**
- **5 个中断源**
  3 个内部中断：T0，TC0，T1。
  2 个外部中断：INT0，INT1。

- **I/O 口引脚配置**
  双向输入输出端口：P0、P1、P5。
  双向输入输出端口，与 SEG 引脚共用：P2、P3。
  单向输入引脚，与复位引脚共用：P0.3。
  具有唤醒功能的端口引脚：P0、P1 电平变换。
  具有上拉电阻的端口：P0、P1、P2、P3、P5。
  外部中断引脚：
  P0.0 触发沿由 PEDGE 控制。
  P0.1 电平变换触发。
  事件计数器输入：
  TC0 事件计数器输入引脚：P0.0。
  T1 事件计数器输入引脚：P0.2。
  RFC 通道引脚：P1.0~P1.4。

- **3 级 LVD：2.0V/2.4V/3.6V**
- **功能强大的指令集**
  指令的长度为一个字。
  大多数指令仅需要一个周期。
  JMP 和 CALL 指令可寻址整个 ROM 区。
  查表指令（MOVC）可寻址整个 ROM 区。

- **Fcpu（指令周期）**
  Fcpu = Fosc/1，Fpsc/2，Fosc/4，Fosc/8，Fosc/16。

- **1 个 8 位基本定时器 T0**
- **1 个 8 位定时/计数器 TC0，具有占空比/周期可编程的 PWM 和 IR 载波信号**
- **1 个 16 位定时/计数器 T1，具有自动重装/计数器/捕捉定时器功能，支持 RFC 功能**
- **5 通道 RFC（电阻频率转换）**
- **4*32 LCD 驱动（128 点），支持内部 C 型和外部 R 型偏压**

- **内置看门狗定时器，时钟源为内部低速 RC 时钟 (16KHz @3V, 32KHz @5V)**

- **4 个系统时钟**
  外部高速时钟：RC，高达 10 MHz。
  外部高速时钟：晶体，高达 16MHz。
  内部高速时钟：PLL 16MHz，由 32K X'tal 提供。
  内部低速时钟：RC，16KHz(3V)，32KHz(5V)。

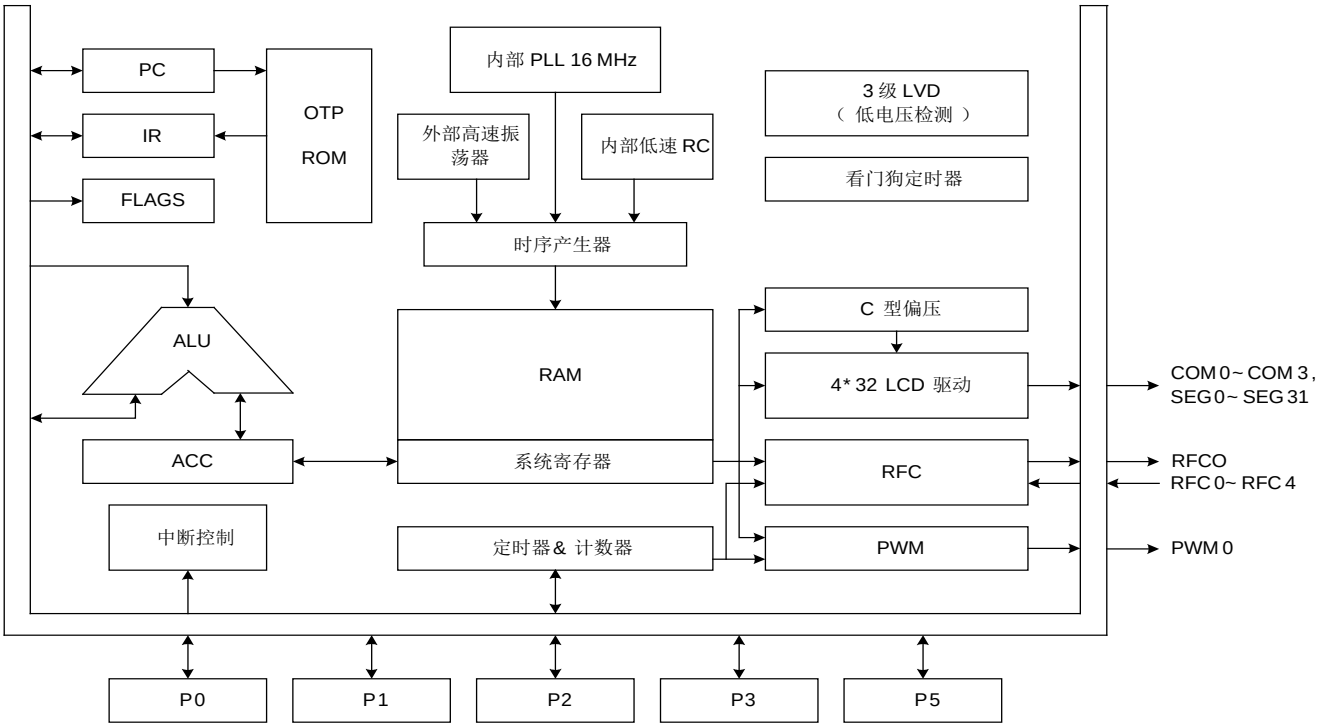
- **4 种工作模式**
  普通模式：高低速时钟都正常工作。
  低速模式：只有低速时钟正常功能。
  睡眠模式：高低速时钟都停止工作。
  绿色模式：由定时器周期性的唤醒。

- **封装形式**
  LQFP 64 pin
  LQFP 48 pin

☞ **产品选型表**

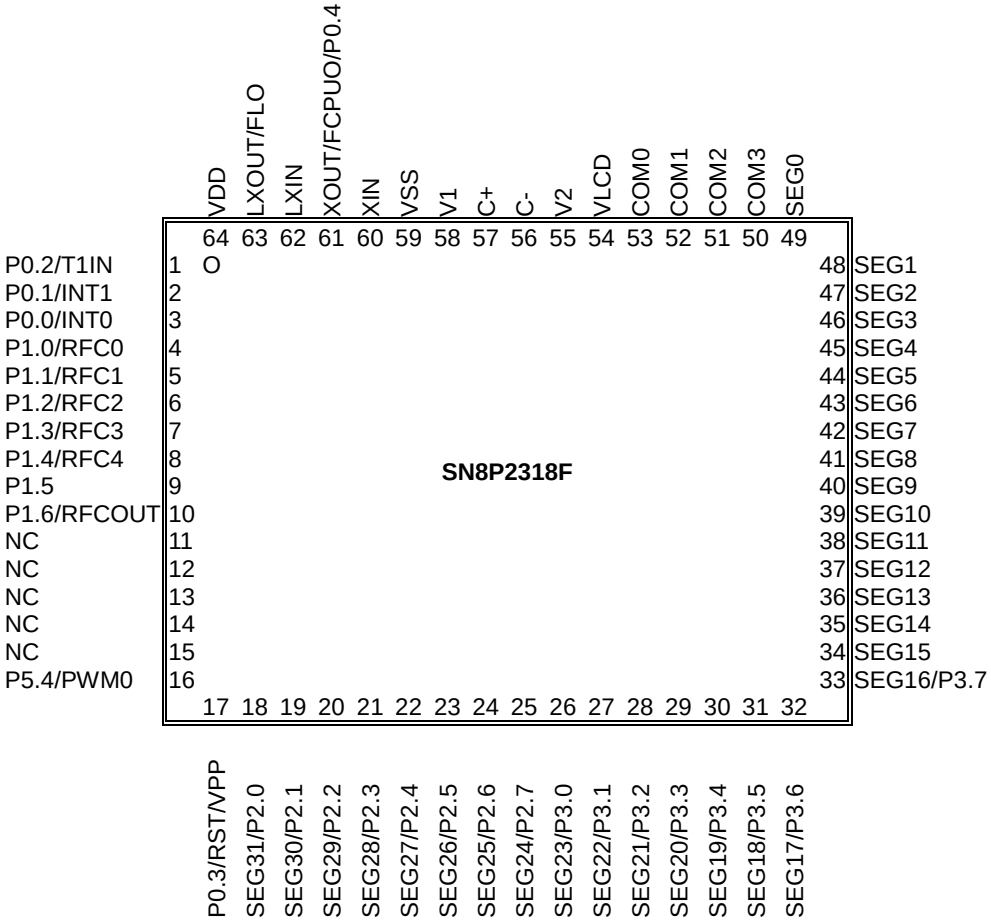
单片机名称	ROM	RAM	堆栈	定时器			PWM	RFC	外部中断	I/O	LCD 驱动			封装形式
				T0	TC0	T1					Dot	C-type	R-type	
SN8P2308	4K*16	128*8	8	V	V	1	1	2-ch	1	25	4*32	V	V	LQFP64
SN8P2318	4K*16	128*8	8	V	V	1	1	5-ch	2	29	4*32	V	V	LQFP64

1.2 系统框图

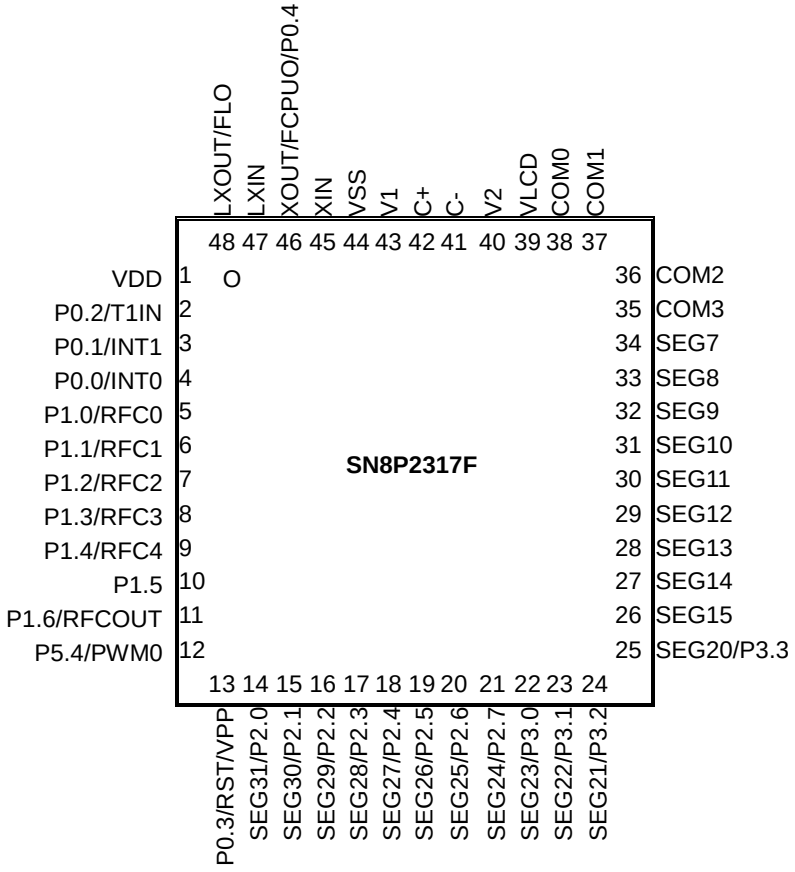

PC
IR
FLAGS
OTP
ROM
内部 PLL 16 MHz
外部高速振荡器
内部低速 RC
时序产生器
3 级 LVD
（ 低电压检测 ）
看门狗定时器
ALU
RAM
C 型偏压
4* 32 LCD 驱动
ACC
系统寄存器
RFC
中断控制
定时器& 计数器
PWM
COM0~COM3,
SEG0~SEG31
RFCO
RFC0~RFC4
PWM0
P0
P1
P2
P3
P5

1.3 引脚配置

SN8P2318F (LQFP 64 pin)



SN8P2317F (LQFP 48 pin)

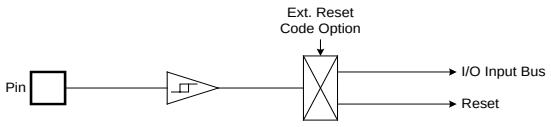


1.4 引脚说明

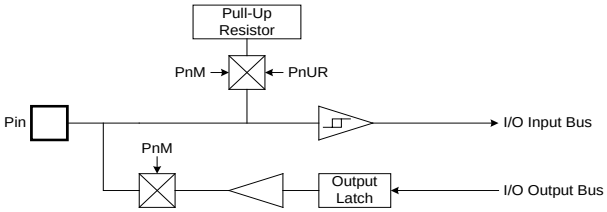
引脚名称	类型	功能说明
VDD, VSS	P	电源输入端。（**VLDE 和 VDD 引脚必须短接。**）
VLCD	P	LCD 电源输入端。（**VLDE 和 VDD 引脚必须短接。**）
C+, C-	P	LCD C-type charge pump 输出引脚。 LCD C-type：连接一个 0.1uF 的电容。 LCD R-type：低电压状态。
V1, V2	P	LCD 偏移电压。1/2 bias：V1 = V2。1/3 bias：V1 = 1/3*Vdd，V2 = 2/3*Vdd。 LCD R-type：连接外部电阻决定偏移电压和电流。
P0.3/RST/VPP	I, P	RST：系统外部复位输入引脚，施密特触发，低电平有效，通常保持高电平。
		VPP：OTP 12.3V 烧录引脚。
		P0.3：单向输入引脚，施密特触发，没有上拉电阻，具有唤醒功能。
XIN	I/O	XIN：外部振荡电路输入引脚（晶体或 RC）。 PLL_16M 模式下，XIN 与 VDD 之间串接一个 0.1uF 的电容。
XOUT/P0.4/FCPUO	I/O	XOUT：外部晶体振荡电路输出引脚。
		P0.4：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		FCPUO：Fcpu 时钟信号输出引脚（High_CLK 编译选项选择 RC 时）。
LXIN	I	LXIN：低速振荡器输入引脚。
LXOUT/FLO	O	LXOUT：低速振荡器输出引脚。
		FLO：低速 RC 振荡电路 Flosc 频率输出引脚（Low_Clk 编译选项选择 RC 时）。
P0.0/INT0	I/O	P0.0：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		INT0：外部中断 INT0 输入引脚。
		TC0 事件计数器输入引脚。
P0.1/INT1	I/O	P0.1：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		INT1：外部中断 INT1 输入引脚。
P0.2/T1IN	I/O	P0.2：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		T1 事件计数器输入引脚。
P1.0/RFC0	I/O	P1.0：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		RFC0：RFC 通道 0。
P1.1/RFC1	I/O	P1.1：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		RFC1：RFC 通道 1。
P1.2/RFC2	I/O	P1.2：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		RFC2：RFC 通道 2。
P1.3/RFC3	I/O	P1.3：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		RFC3：RFC 通道 3。
P1.4/RFC4	I/O	P1.4：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		RFC4：RFC 通道 4。
P1.5	I/O	P1.5：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
P1.6/RFCOUT	I/O	P1.6：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻，具有唤醒功能。
		RFCOUT：RFC 振荡信号输出引脚。
P5.4/ PWM0	I/O	P5.4：双向输入输出引脚，输入模式时施密特触发，内置上拉电阻。
		PWM0：可编程 PWM 输出引脚（由 TC0 输出）。
P2[7:0]/ SEG[31:24]	I/O	P2[7:0]：双向输入输出引脚，非施密特触发，内置上拉电阻。
		SEG[31:24]：LCD segment 输出引脚。
P3[7:0]/ SEG[23:16]	I/O	P3[7:0]：双向输入输出引脚，非施密特触发，内置上拉电阻。
		SEG[23:16]：LCD segment 输出引脚。
SEG[15:0]	O	SEG[15:0：LCD segment 输出引脚。
COM[3:0]	O	COM[3:0]：LCD common 0~3 输出引脚。

1.5 引脚电路结构图

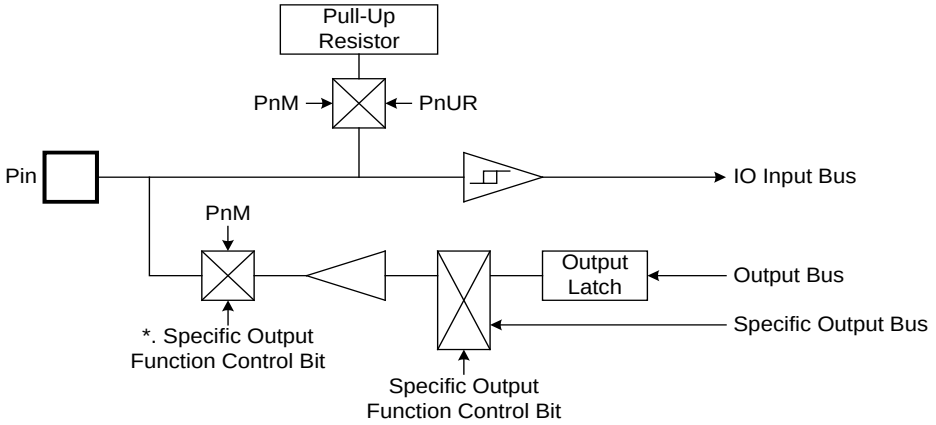
- 复位共用引脚：



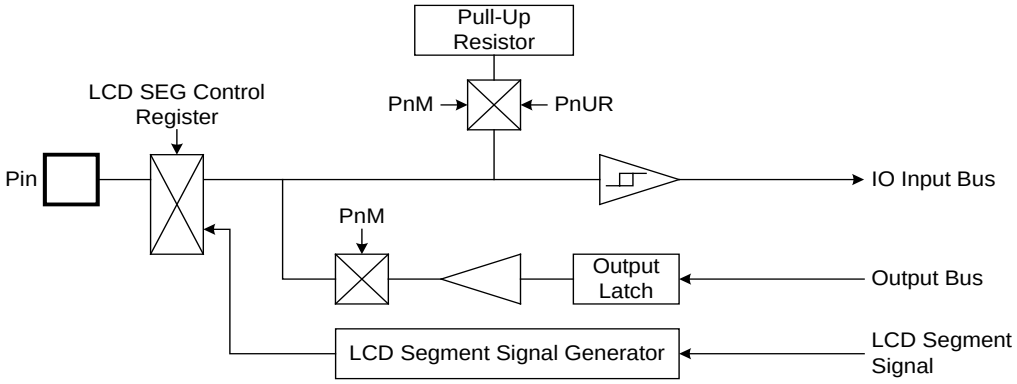
- GPIO 引脚：



- 特殊输出功能（如 **PWM、RFC.....**）共用引脚：



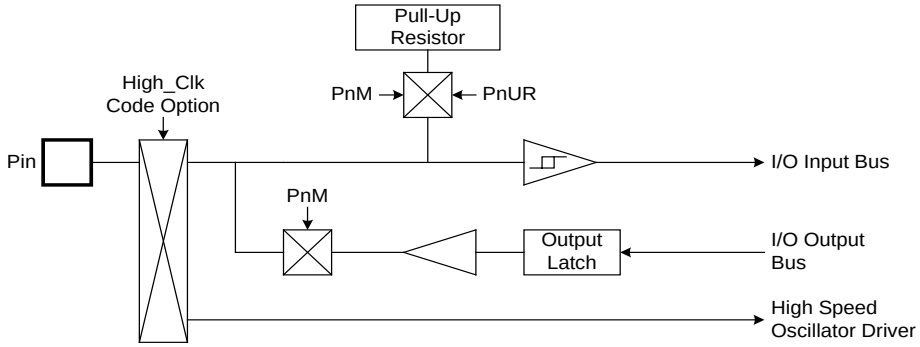
- LCD SEG 共用引脚：



- **振荡器共用引脚：**

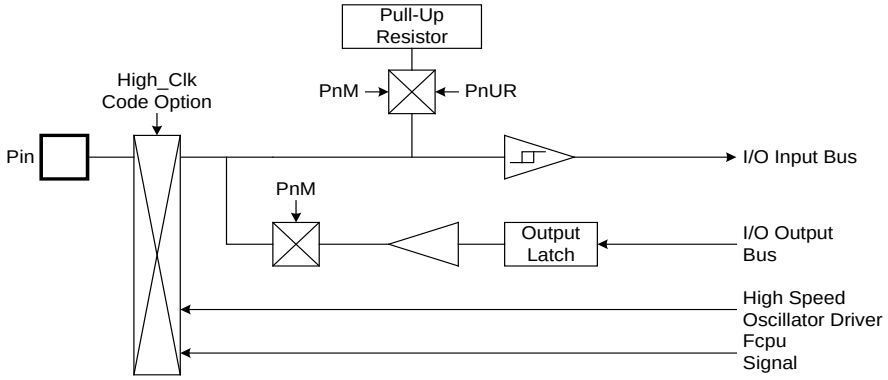
XIN:

引脚名称	振荡器编译选项	功能说明
XIN	RC, 4M X'tal, 12M X'tal, PLL_16M	振荡器输入引脚。



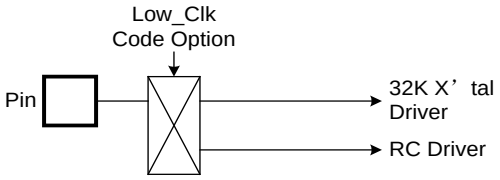
XOUT/FCPUO/P0.4:

引脚名称	振荡器编译选项	功能说明
XOUT	4M X'tal, 12M X'tal	振荡器输出引脚。
FCPUO	RC	Fcpu 信号输出引脚，测量 RC 频率，调整 RC 参数。
P0.4	PLL_16M	GPIO 模式。



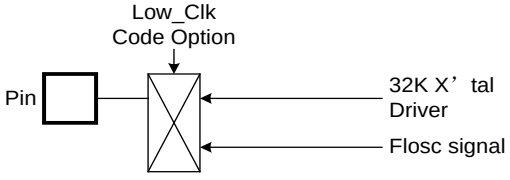
LXIN:

引脚名称	振荡器编译选项	功能说明
LXIN	RC, 32K X'tal	振荡器输入引脚。



LXOUT/FLO:

引脚名称	振荡器编译选项	功能说明
LXOUT	32K X'tal	振荡器输出引脚。
FLO	RC	Flosc 信号输出引脚，测量 RC 频率，调整 RC 参数。



2 中央处理器（CPU）

2.1 程序存储器（ROM）

☞ **ROM：4K**

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

2.1.1 复位向量（0000H）

具有一个字长的系统复位向量（0000H）。

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

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

➢ **例：定义复位向量。**

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

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

2.1.2 中断向量（0008H）

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

✷ **注：“PUSH”，“POP”指令用于存储和恢复 ACC/PFLAG（不包括 NT0、NTD）。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。

        ORG     10H
START:                          ; 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。

2.1.3 查表

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

➢ **例：查找 ROM 地址为“TABLE1”的值。**

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

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

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

➢ **例：宏 INC_YZ。**

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

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

➢ **例：通过“INC_YZ”对上例进行优化。**

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

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

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

➢ **例：由指令 B0ADD/ADD 对 Y 和 Z 寄存器加 1。**

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

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

                B0BTS1  FC              ; 检查进位标志。
                JMP     GETDATA         ; FC = 0。
                INCMS   Y               ; FC = 1。
                NOP
GETDATA:                                ;
                MOVC                    ; 存储数据，如果 BUF = 0，数据为 0035H。
                                        ; 如果 BUF = 1，数据=5105H。
                                        ; 如果 BUF = 2，数据=2012H。
                …
TABLE1:         DW      0035H           ; 定义数据表（16 位）数据。
                DW      5105H
                DW      2012H
                …
```

2.1.4 跳转表

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

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

➢ **例：跳转表。**

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

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

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

➢ **例：如果跳转表跨越 ROM 边界，将引起程序错误。**

```
@JMP_A  MACRO   VAL
        IF      (($+1) !& 0XFF00) !!= (($+(VAL)) !& 0XFF00)
        JMP     ($ | 0XFF)
        ORG     ($ | 0XFF)
        ENDIF
        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（00FFH~0100H），宏指令“@JMP_A”将调整跳转表到适当的位置（0100H）。

➢ **例：“@JMP_A”运用举例。**

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

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

2.1.5 CHECKSUM计算

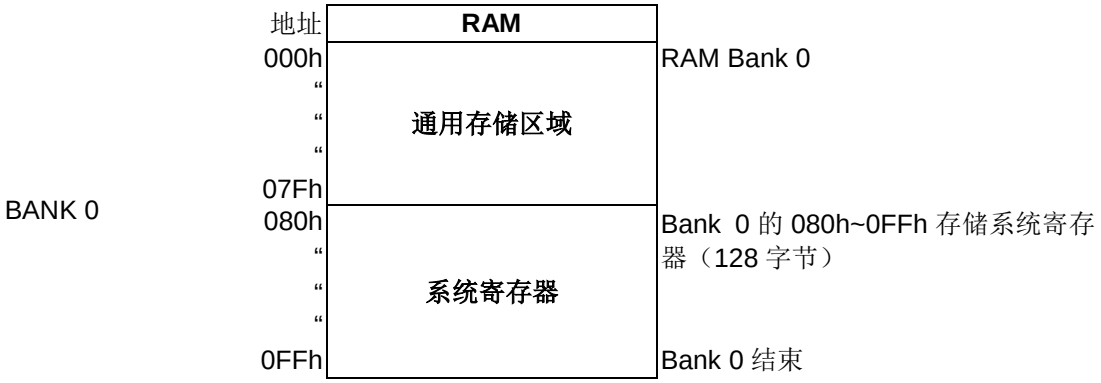
ROM 区末端位置的几个字限制使用，进行 Checksum 计算时，用户应避免对该单元的访问。

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

```
                MOV         A,#END_USER_CODE$L
                B0MOV       END_ADDR1, A        ; 用户程序结束地址低地址存入 end_addr1。
                MOV         A,#END_USER_CODE$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。
@@:
                MOVC
                B0BCLR      FC                  ; 清标志位 C。
                ADD         DATA1, A
                MOV         A, R
                ADC         DATA2, A
                JMP         END_CHECK           ; 检查 YZ 地址是否为代码的结束地址。
AAA:
                INCMS       Z                   ;
                JMP         @B                  ; 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。
```

2.2 数据存储器（RAM）

☞ **RAM：128 X 8 位**



128 字节通用存储区位于 RAM Bank 0，Sonix 提供 bank 0 型指令（如 B0MOV，B0ADD，B0BTS1，B0BSET……）在非零 RAM Bank 区直接控制 RAM Bank 0。

2.2.1 系统寄存器

2.2.1.1 系统寄存器列表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	L	H	R	Z	Y	-	PFLAG	RBANK	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	T1M	T1CL	T1CH	T1VCL	T1VCH	T1CKSM	RFCM	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	P0M	-	-	-	-	-	-	PEDGE
C	P1W	P1M	P2M	P3M	-	P5M	-	-	INTRQ	INTEN	OSCM	LCDM	WDTR	TC0R	PCL	PCH
D	P0	P1	P2	P3	-	P5	-	-	T0M	T0C	TC0M	TC0C	-	-	-	STKP
E	P0UR	P1UR	P2UR	P3UR	-	P5UR	@HL	@YZ	TC0D	-	-	-	-	-	-	-
F	STK7L	STK7H	STK6L	STK6H	STK5L	STK5H	STK4L	STK4H	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

2.2.1.2 系统寄存器说明

H, L = 工作寄存器，@HL 间接寻址寄存器
R = 工作寄存器，ROM 查表数据缓存器
RBANK = RAM bank 控制寄存器
T1CH,L = T1 计数寄存器
T1CKSM = T1 捕捉定时器控制寄存器
INTRQ = 中断请求寄存器
OSCM = 振荡模式控制寄存器
WDTR = 看门狗定时器清零寄存器
P1W = P1 唤醒功能控制寄存器
PnM = Pn 输入/输出模式控制寄存器
PnUR = Pn 上拉电阻控制寄存器
T0C = T0 计数寄存器
TC0C = TC0 计数寄存器
TC0D = TC0 占空比控制寄存器
@YZ = RAM YZ 间接寻址寄存器
STK0~STK7 = 堆栈缓存器

Y, Z = 工作寄存器，@YZ 间接寻址寄存器，ROM 寻址寄存器
PFLAG = 特殊标志寄存器
T1M = T1 模式寄存器
T1VCH,L = T1 事件计数器计数寄存器
RFCM = RFC 模式寄存器
INTEN = 中断使能寄存器
LCDM = LCD 模式寄存器
PEDGE = P0.0 触发方向控制寄存器
PCH, PCL = 程序计数器
Pn = Pn 数据缓存器
T0M = T0 模式寄存器
TC0M = TC0 模式寄存器
TC0R = TC0 自动重装数据缓存器
@HL = RAM HL 间接寻址寄存器
STKP = 堆栈指针

2.2.1.3 系统寄存器的位定义

地址	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	备注
080H	LBIT7	LBIT6	LBIT5	LBIT4	LBIT3	LBIT2	LBIT1	LBIT0	R/W	L
081H	HBIT7	HBIT6	HBIT5	HBIT4	HBIT3	HBIT2	HBIT1	HBIT0	R/W	H
082H	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0	R/W	R
083H	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0	R/W	Z
084H	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0	R/W	Y
086H	NT0	NPD	LVD36	LVD24		C	DC	Z	R/W	PFLAG
087H								RBNKS0	R/W	RBANK
0A0H	T1ENB	T1rate2	T1rate1	T1rate0	T1CKS				R/W	T1M
0A1H	T1CL7	T1CL6	T1CL5	T1CL4	T1CL3	T1CL2	T1CL1	T1CL0	R/W	T1CL
0A2H	T1CH7	T1CH6	T1CH5	T1CH4	T1CH3	T1CH2	T1CH1	T1CH0	R/W	T1CH
0A3H	T1VCL7	T1VCL6	T1VCL5	T1VCL4	T1VCL3	T1VCL2	T1VCL1	T1VCL0	R/W	T1VCL
0A4H							T1VCH1	T1VCH0	R/W	T1VCH
0A5H	CPTVC				CPTCKS	CPTStart	CPTG1	CPTG0	R/W	T1CKSM
0A6H	RFCENB		0	1	RFCOUT	RFCH2	RFCH1	RFCH0	R/W	RFCM
0B8H				P04M		P02M	P01M	P00M	R/W	P0M
0BFH				P00G1	P00G0				R/W	PEDGE
0C0H		P16W	P15W	P14W	P13W	P12W	P11W	P10W	W	P1W
0C1H		P16M	P15M	P14M	P13M	P12M	P11M	P10M	R/W	P1M
0C2H	P27M	P26M	P25M	P24M	P23M	P22M	P21M	P20M	R/W	P2M
0C3H	P37M	P36M	P35M	P34M	P33M	P32M	P31M	P30M	R/W	P3M
0C5H				P54M					R/W	P5M
0C8H		T1IRQ	TC0IRQ	T0IRQ			P01IRQ	P00IRQ	R/W	INTRQ
0C9H		T1IEN	TC0IEN	T0IEN			P01IEN	P00IEN	R/W	INTEN
0CAH				CPUM1	CPUM0	CLKMD	STPHX		R/W	OSCM
0CBH	CPCK1	CPCK0	VLCDCP	PSEG2	PSEG1	PSEG0	BIAS	LCDENB	R/W	LCDM
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					PC11	PC10	PC9	PC8	R/W	PCH
0D0H				P04	P03	P02	P01	P00	R/W	P0
0D1H		P16	P15	P14	P13	P12	P11	P10	R/W	P1
0D2H	P27	P26	P25	P24	P23	P22	P21	P20	R/W	P2
0D3H	P37	P36	P35	P34	P33	P32	P31	P30	R/W	P3
0D5H				P54					R/W	P5
0D8H	T0ENB	T0rate2	T0rate1	T0rate0				T0TB	R/W	T0M
0D9H	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0	R/W	T0C
0DAH	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS1	TC0CKS0		PWM0OUT	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DFH	GIE					STKPB2	STKPB1	STKPB0	R/W	STKP
0E0H				P04R		P02R	P01R	P00R	W	P0UR
0E1H		P16R	P15R	P14R	P13R	P12R	P11R	P10R	W	P1UR
0E2H	P27R	P26R	P25R	P24R	P23R	P22R	P21R	P20R	W	P2UR
0E3H	P37R	P36R	P35R	P34R	P33R	P32R	P31R	P30R	W	P3UR
0E5H				P54R					W	P5UR
0E6H	@HL7	@HL6	@HL5	@HL4	@HL3	@HL2	@HL1	@HL0	R/W	@HL
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ
0E8H	TC0D7	TC0D6	TC0D5	TC0D4	TC0D3	TC0D2	TC0D1	TC0D0	R/W	TC0D
0F0H	S7PC7	S7PC6	S7PC5	S7PC4	S7PC3	S7PC2	S7PC1	S7PC0	R/W	STK7L
0F1H					S7PC11	S7PC10	S7PC9	S7PC8	R/W	STK7H
0F2H	S6PC7	S6PC6	S6PC5	S6PC4	S6PC3	S6PC2	S6PC1	S6PC0	R/W	STK6L
0F3H					S6PC11	S6PC10	S6PC9	S6PC8	R/W	STK6H
0F4H	S5PC7	S5PC6	S5PC5	S5PC4	S5PC3	S5PC2	S5PC1	S5PC0	R/W	STK5L
0F5H					S5PC11	S5PC10	S5PC9	S5PC8	R/W	STK5H
0F6H	S4PC7	S4PC6	S4PC5	S4PC4	S4PC3	S4PC2	S4PC1	S4PC0	R/W	STK4L
0F7H					S4PC11	S4PC10	S4PC9	S4PC8	R/W	STK4H
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H					S3PC11	S3PC10	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH					S2PC11	S2PC10	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH					S1PC11	S1PC10	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH					S0PC11	S0PC10	S0PC9	S0PC8	R/W	STK0H

✷ 注：
1、所有寄存器名都已在 SN8ASM 编译器中做了宣告；
2、用户使用 SN8ASM 编译器对寄存器的位进行操作时，须在该寄存器的位前加“F”；
3、指令“b0bset”，“b0bclr”，“bset”，“bclr”只能用于可读写的寄存器（“R/W”）。

2.2.2 累加器

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。
```

2.2.3 程序状态寄存器PFLAG

寄存器 PFLAG 中包含 ALU 运算状态信息、系统复位状态信息和 LVD 检测信息，其中，位 NT0 和 NPD 显示系统复位状态信息，包括上电复位、LVD 复位、外部复位和看门狗复位；位 C、DC 和 Z 显示 ALU 的运算信息。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
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**：3.6V LVD 工作电压标志， 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**：2.4V LVD 工作电压标志，LVD 编译选项为 LVD_M 时有效。
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。**

2.2.4 程序计数器

程序计数器 PC 是一个 12 位二进制程序地址寄存器，分高 4 位和低 8 位。专门用来存放下一条需要执行指令的内存地址。通常，程序计数器会随程序中指令的执行自动增加。

若程序执行 CALL 和 JMP 指令时，PC 指向特定的地址。

	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PC	-	-	-	-	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
复位后	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
	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
                JMP         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          ;如果 ACC 不为“0”，则跳至 C0STEP。
                …
C0STEP:         NOP
```

INCMS：

```
                INCMS       BUF0
                JMP         C0STEP          ;如果 BUF0 不为“0”，则跳至 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          ;如果 ACC 不为“0”，则跳至 C0STEP。
                …
C0STEP:         NOP
```

DECMS：

```
                DECMS       BUF0
                JMP         C0STEP          ;如果 BUF0 不为“0”，则跳至 C0STEP。
                …
C0STEP:         NOP
```

☞ **多地址跳转**

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

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

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

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

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

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

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

2.2.5 H, L寄存器

寄存器 H 和 L 都是 8 位缓存器，主要有以下两个功能：

- **通用工作寄存器；**
- **RAM 数据寻址指针@HL。**

081H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
H	HBIT7	HBIT6	HBIT5	HBIT4	HBIT3	HBIT2	HBIT1	HBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	-	-	-	-	-

080H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
L	LBIT7	LBIT6	LBIT5	LBIT4	LBIT3	LBIT2	LBIT1	LBIT0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	-	-	-	-	-

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

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

➢ **例：对 bank 0 中的数据进行清零处理。**

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

        CLR     @HL
END_CLR:
        …
        …
```

2.2.6 Y, Z寄存器

寄存器 Y 和 Z 都是 8 位缓存器，主要用途如下：

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

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

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

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

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

➢ **例：利用数据指针@YZ 对 RAM 数据清零。**

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

CLR_YZ_BUF:
        CLR     @YZ             ; @YZ 清零。

        DECMS   Z               ;
        JMP     CLR_YZ_BUF      ; 不为零。

        CLR     @YZ
END_CLR:                        ;
        …
```

2.2.7 R寄存器

8 位缓存器 R 主要有以下两个功能：

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

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

✷ **注：关于 R 寄存器的查表功能，请参考“查表”章节的说明。**

2.3 寻址模式

2.3.1 立即寻址

将立即数直接送入 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：立即寻址模式中，指定的 RAM 单元必须是 80H~8FH 的工作寄存器。**

2.3.2 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 中 12H 单元。**

```
B0MOV   12H, A
```

2.3.3 间接寻址

通过数据指针（H/L，Y/Z）对数据存储单元进行读写。

- **例：通过指针@HL 间接寻址。**

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

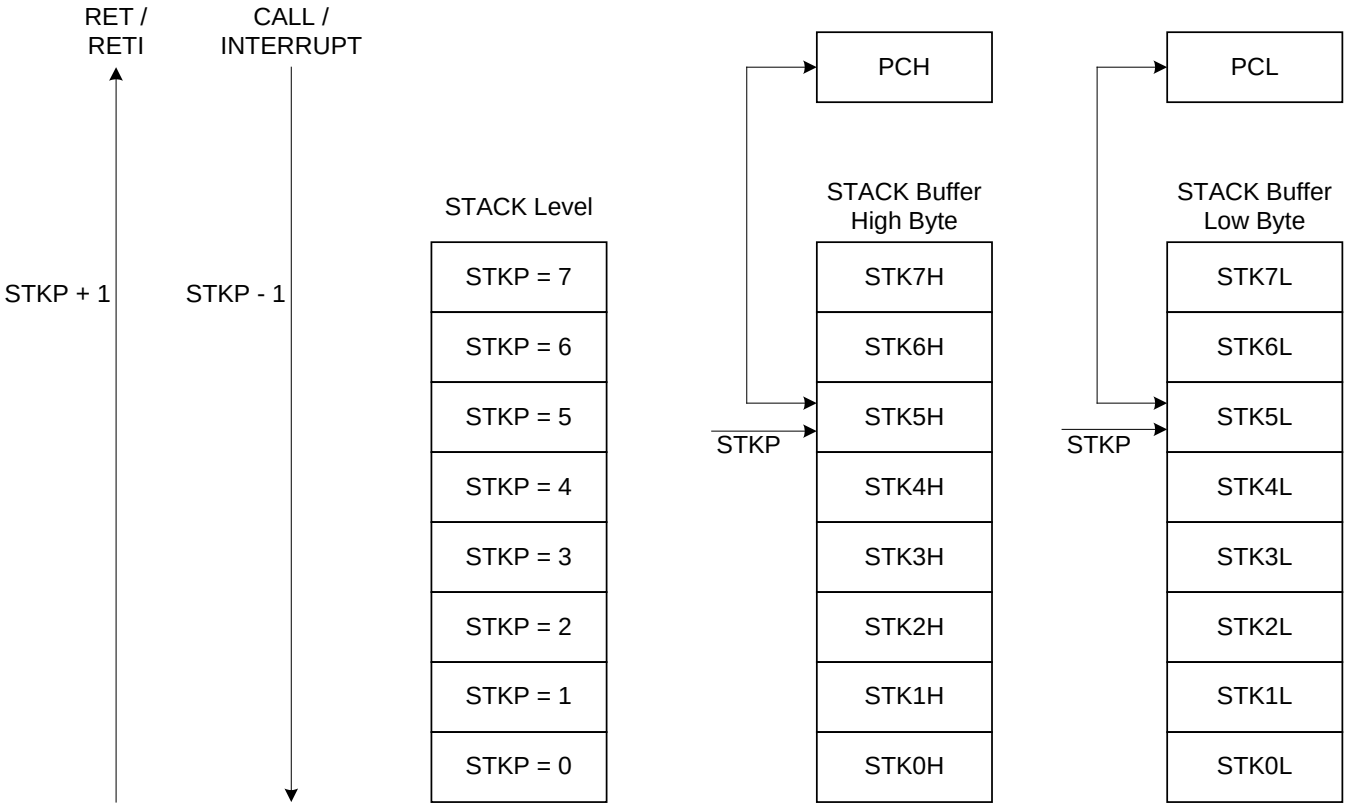
- **例：通过指针@YZ 间接寻址。**

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

2.4 堆栈

2.4.1 概述

SN8P2318 的堆栈缓存器共有 8 层，程序进入中断或执行 CALL 指令时，用来存储程序计数器 PC 的值。寄存器 STKP 为堆栈指针，指向堆栈缓存器顶层，STKnH 和 STKnL 分别是各堆栈缓存器高、低字节。



2.4.2 堆栈寄存器

堆栈指针 STKP 是一个 3 位寄存器，存放被访问的堆栈单元地址，13 位数据存储器 STKnH 和 STKnL 用于暂存堆栈数据。以上寄存器都位于 bank 0。

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
STKP	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0
读/写	R/W	-	-	-	-	R/W	R/W	R/W
复位后	0	-	-	-	-	1	1	1

Bit[2:0] **STKPBn**：堆栈指针（n = 0 ~ 2）。

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

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

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

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnH	-	-	-	SnPC12	SnPC11	SnPC10	SnPC9	SnPC8
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
复位后	-	-	-	0	0	0	0	0

0F0H~0FFH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STKnL	SnPC7	SnPC6	SnPC5	SnPC4	SnPC3	SnPC2	SnPC1	SnPC0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

STKn = STKnH , STKnL (n = 7 ~ 0)

2.4.3 堆栈操作举例

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

堆栈层数	STKP 寄存器			堆栈缓存器		注释
	STKPB2	STKPB1	STKPB0	高字节	低字节	
0	1	1	1	Free	Free	-
1	1	1	0	STK0H	STK0L	-
2	1	0	1	STK1H	STK1L	-
3	1	0	0	STK2H	STK2L	-
4	0	1	1	STK3H	STK3L	-
5	0	1	0	STK4H	STK4L	-
6	0	0	1	STK5H	STK5L	-
7	0	0	0	STK6H	STK6L	-
8	1	1	1	STK7H	STK7L	-
> 8	1	1	0	-	-	堆栈溢出，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存器。堆栈恢复操作如下表所示：

堆栈层数	STKP 寄存器			堆栈缓存器		注释
	STKPB2	STKPB1	STKPB0	高字节	低字节	
8	1	1	1	STK7H	STK7L	-
7	0	0	0	STK6H	STK6L	-
6	0	0	1	STK5H	STK5L	-
5	0	1	0	STK4H	STK4L	-
4	0	1	1	STK3H	STK3L	-
3	1	0	0	STK2H	STK2L	-
2	1	0	1	STK1H	STK1L	-
1	1	1	0	STK0H	STK0L	-
0	1	1	1	Free	Free	-

2.5 编译选项列表（CODE OPTION）

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

编译选项	配置项目	功能说明
High_Clk	PLL_16M	高速内部 16MHz PLL，外部低速 32K 振荡器正常运行，XOUT 引脚为 GPIO 引脚，XIN 引脚和 VDD 之间串接一个 0.1uF 的电容。
	RC	外部高速振荡器采用低成本的 RC 振荡电路，XIN 引脚连接 RC 电路，XOUT 引脚为 Fcpu 信号输出引脚。
	12M X'tal	外部高速振荡器采用高速陶瓷/石英振荡器（如 12MHz）。
	4M X'tal	外部高速振荡器采用标准陶瓷/石英振荡器（如 4MHz）。
Low_Clk	32K X'tal	外部低速振荡器采用低频、低功耗的晶体振荡器（如 32.768KHz），LXIN/LXOUT 引脚驱动外部 32768Hz 陶瓷/石英振荡器。
	RC	LXIN 引脚驱动外部 RC 振荡器，LXOUT 引脚输出 Flosc 时钟。
Fcpu	Fhosc/1	指令周期为 1 个振荡时钟。
	Fhosc/2	指令周期为 2 个振荡时钟。
	Fhosc/4	指令周期为 4 个振荡时钟。
	Fhosc/8	指令周期为 8 个振荡时钟。
	Fhosc/16	指令周期为 16 个振荡时钟。
Watch_Dog	Always_On	始终开启看门狗定时器，睡眠模式和绿色模式下也不例外。
	Enable	开启看门狗定时器，但在睡眠模式和绿色模式会处于关闭状态。
	Disable	关闭看门狗定时器。
Reset_Pin	Reset	使能外部复位引脚。
	P03	使能单向输入引脚 P0.3，无上拉电阻。
Noise_Filter	Enable	开启杂讯滤波器。
	Disable	关闭杂讯滤波器。
Security	Enable	ROM 代码加密。
	Disable	ROM 代码不加密。
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； 使能 PFLAG 寄存器的 LVD24 标志位以开启 2.4V 的低电压检测功能。
	LVD_H	VDD 低于 2.4V 时，LVD 复位系统； 使能 PFLAG 寄存器的 LVD36 标志位以开启 3.6V 的低电压检测功能。
	LVD_MAX	VDD 低于 3.6V 时，LVD 复位系统。

2.5.1 High_Clk编译选项

High_Clk 编译选项控制系统的高速振荡器，包括 PLL_16M，RC，4M X'tal 和 12M X'tal。PLL_16M 模式下，LXIN/LXOUT 连接 32KHz 晶振或者 RC 电路，XIN 引脚串接一个 0.1uF 电容，XOUT 为 GPIO 引脚。

2.5.2 Low_Clk编译选项

Low_Clk 编译选项控制系统的低速振荡器，包括 32K X'tal 和 RC。RC 模式下，LXIN 连接 RC 电路，LXOUT 输出 Flosc 信号测量 RC 频率。

2.5.3 Fcpu编译选项

Fcpu 指在高速操作模式下的指令周期。低速模式下，系统时钟源由外部低速 32KHz 振荡器提供或 RC 振荡器提供，与 LXIN/LXOUT 引脚相连接。低速模式下的 Fcpu 不受 Fcpu 编译选项的控制，固定为 Flosc/4。

2.5.4 Reset_Pin编译选项

复位引脚与单向输入引脚共用，由编译选项控制。

- **Reset**：使能外部复位引脚功能。当下降沿触发时，系统复位。
- **P03**：使能 P0.3 的单向输入功能，此时禁止外部复位引脚功能。

2.5.5 Security编译选项

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

2.5.6 Noise Filter编译选项

Noise Filter 编译选项是强杂讯滤除功能以减少杂讯对系统时钟的影响。如使能杂讯滤波器，在高干扰环境下，使能看门狗定时器且选择一个合适的 LVD 选项可以使整个系统更好的工作。

✷ 注：CODE OPTION High_CLK 选择 PLL_16M，Low_Clk 选择 32K X'tal，Watchdog 选择 Enable/Always_On 时，系统在睡眠模式下无效（支持绿色模式）。

3 复位

3.1 概述

SN8P2318 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（使能外部复位引脚时有效）。**

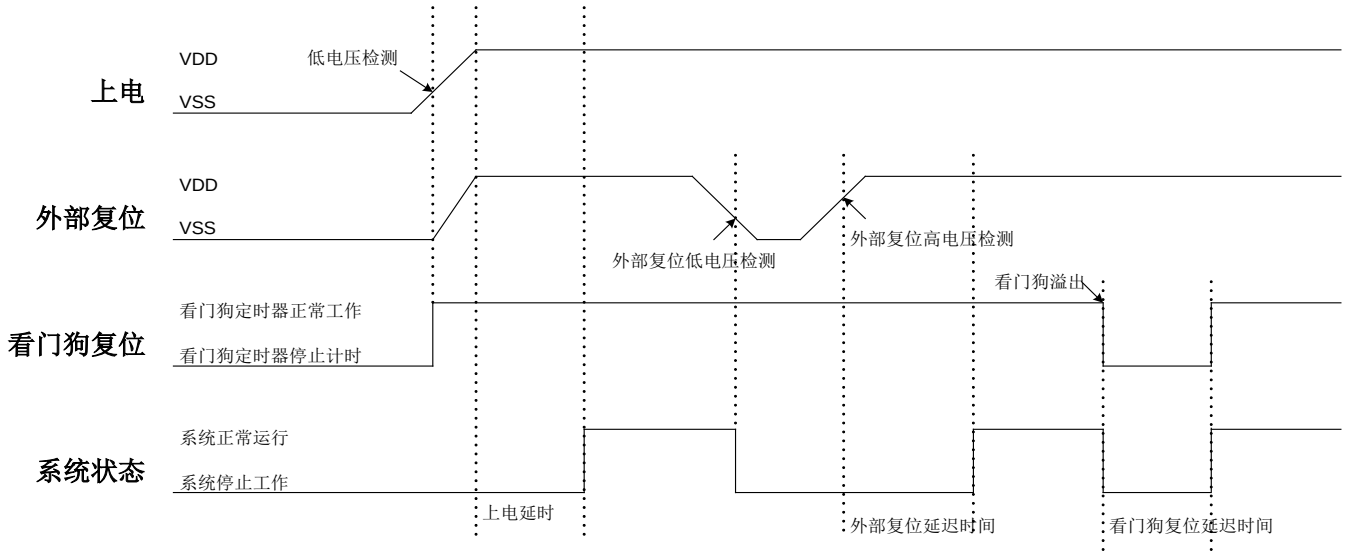
上述任一种复位发生时，所有的系统寄存器恢复默认状态，程序停止运行，同时程序计数器 PC 清零。复位结束后，系统从向量 0000H 处重新开始运行。PFLAG 寄存器的 NT0 和 NPD 标志位可以显示系统复位状态的信息。用户可以根据 NT0 和 NPD 的状态，编程控制系统的运行路径。

086H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PFLAG	NT0	NPD	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（使能外部复位引脚时才有效）：**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的复位结束。
- **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。

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。

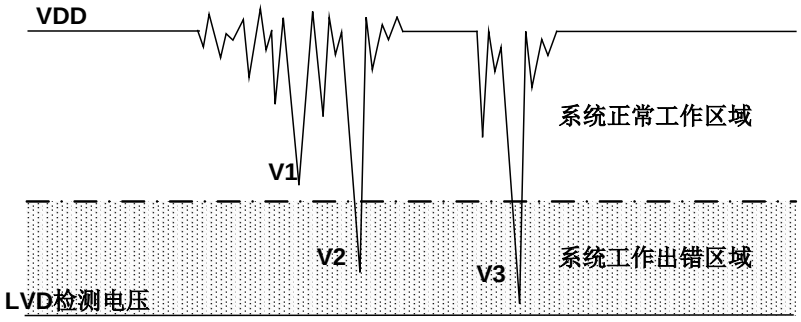
看门狗定时器应用注意事项如下：

- 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。

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

3.4 掉电复位

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



掉电复位示意图

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

DC 运用中：

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

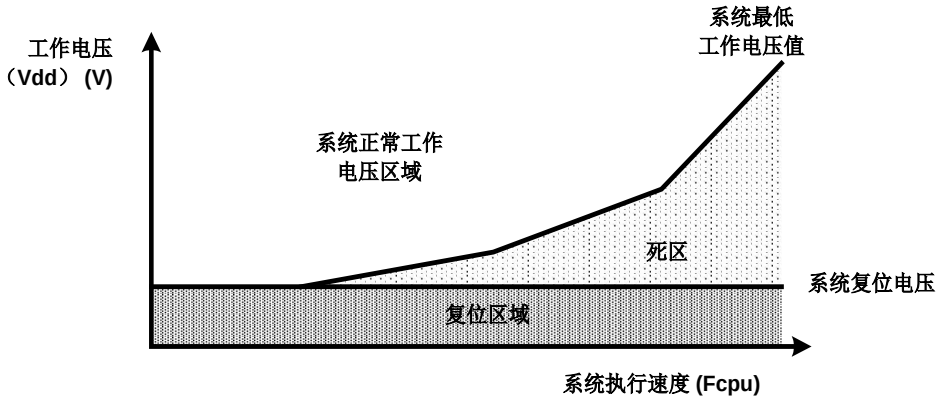
AC 运用中：

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

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

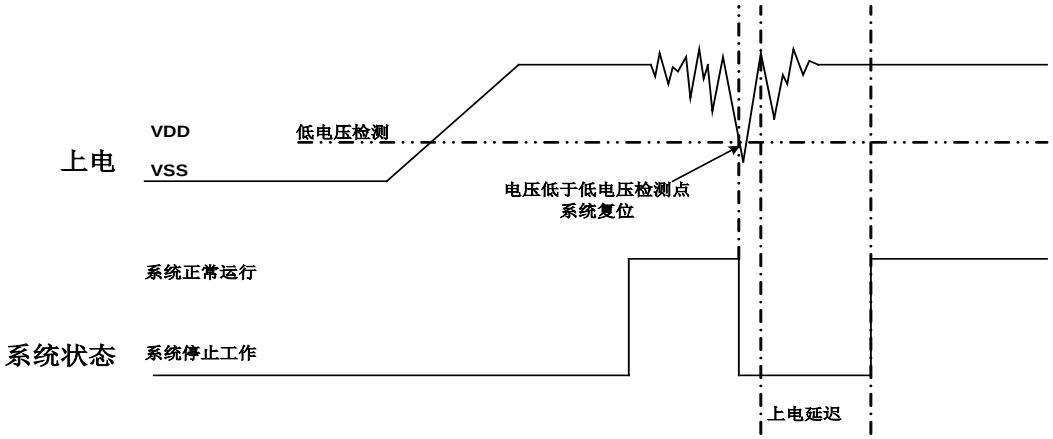
3.4.1 系统工作电压

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



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

3.4.2 低电压检测（LVD）



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

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
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 工作电压标志。
0 = Vdd > LVD36）；
1 = Vdd ≦ LVD36。

Bit 4 **LVD24**：LVD 2.4V 工作电压标志。
0 = Vdd > LVD24；
1 = Vdd ≦ LVD24。

LVD	LVD 编译选项			
	LVD_L	LVD_M	LVD_H	LVD_MAX
2.0V 复位	有效	有效	有效	有效
2.4V 标志	-	有效	-	-
2.4V 复位	-	-	有效	-
3.6V 标志	-	-	有效	-
3.6V 复位	-	-	-	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。
LVD_H
如果 VDD < 2.4V，系统复位；
LVD36：如果 VDD > 3.6V，LVD36＝0；如果 VDD <= 3.6V，LVD36＝1；
LVD24 标志位无意义。
LVD_MAX
如果 VDD<3.6V，系统复位。

✷ 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。

3.4.3 掉电复位性能改进

如何改善系统掉电复位性能，有以下几点建议：

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

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

看门狗复位：

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

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

降低系统工作速度：

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

附加外部复位电路：

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

3.5 外部复位

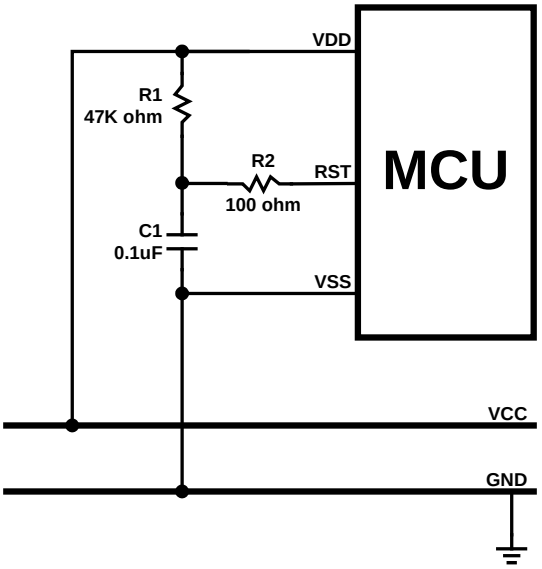
外部复位功能由编译选项“Reset_Pin”控制。将该编译选项置为“Reset”，可使能外部复位功能。外部复位引脚为施密特触发结构，低电平有效。复位引脚处于高电平时，系统正常运行。当复位引脚输入低电平信号时，系统复位。外部复位操作在上电和正常工作模式时有效。需要注意的是，在系统上电完成后，外部复位引脚必须输入高电平，否则系统将一直保持在复位状态。外部复位的时序如下：

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

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

3.6 外部复位电路

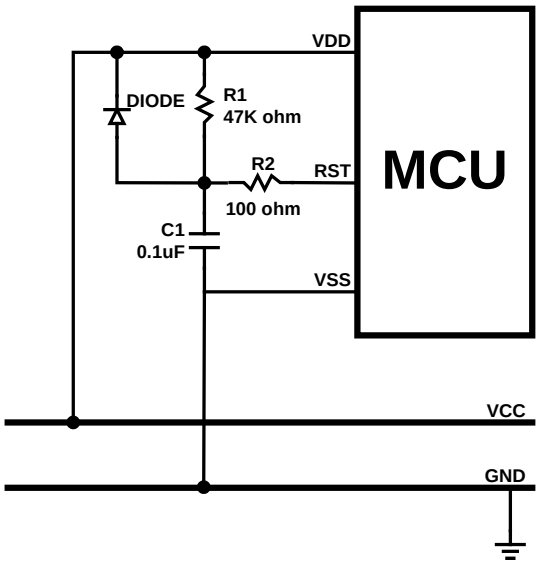
3.6.1 基本RC复位电路



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

✷ **注：此 RC 复位电路不能解决非正常上电和掉电复位问题。**

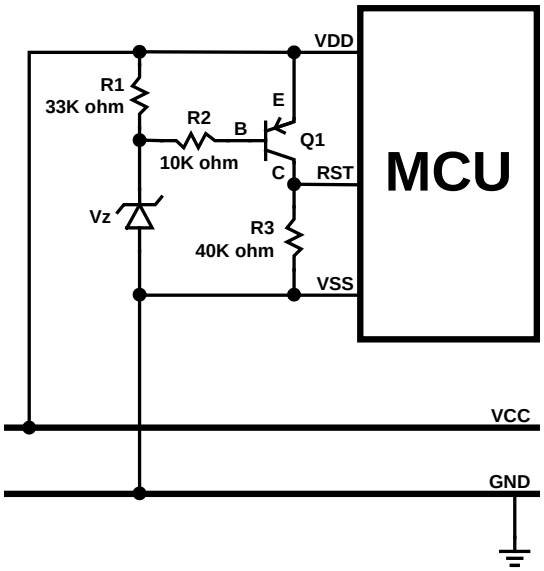
3.6.2 二极管&RC复位电路



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

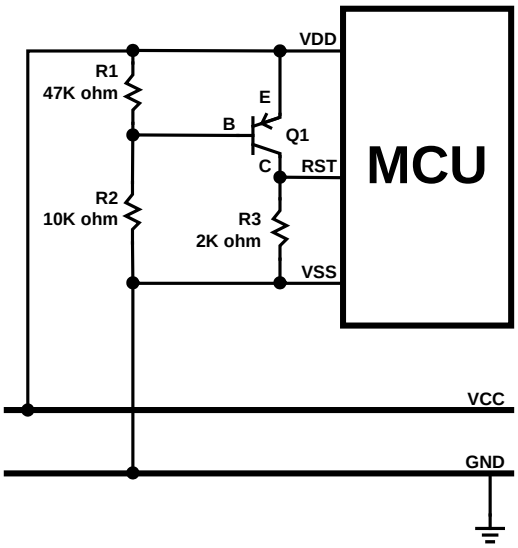
✷ **注：“基本 RC 复位电路”和“二极管及 RC 复位电路”中的电阻 R2 都是必不可少的限流电阻，以避免复位引脚 ESD（Electrostatic Discharge）或 EOS（Electrical Over-stress）击穿。**

3.6.3 齐纳二极管复位电路



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

3.6.4 电压偏置复位电路

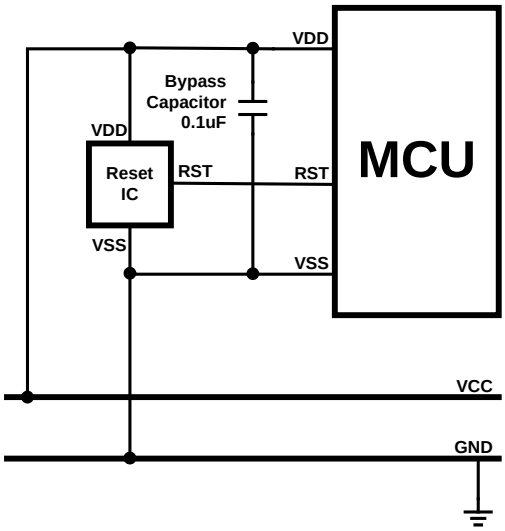


电压偏置复位电路是一种简单的 LVD 电路，基本上可以完全解决掉电复位问题。与稳压二极管复位电路相比，这种复位电路的检测电压值的精确度有所降低。电路中，R1 和 R2 构成分压电路，当 VDD 高于和等于分压值“0.7V x (R1 + R2) / R1”时，三极管集电极 C 输出高电平，单片机正常工作；VDD 低于“0.7V x (R1 + R2) / R1”时，集电极 C 输出低电平，单片机复位。

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

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

3.6.5 外部IC复位电路



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

4 系统时钟

4.1 概述

SN8P2318 内置双时钟系统：高速时钟和低速时钟。高速时钟由内部高速振荡时钟和外部高速振荡时钟提供，外部高速振荡时钟由编译选项“High_CLK”控制。低速时钟由外部低速振荡器提供。高、低速时钟都可以作为系统时钟源。

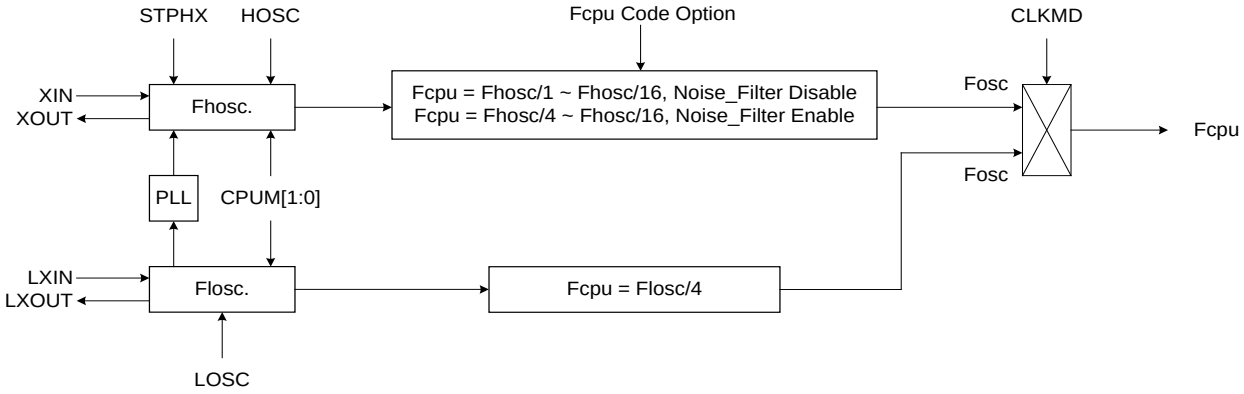
- **高速振荡器**

内部高速振荡器：16MHz PLL，称为 PLL_16M。
外部高速振荡器：包括石英/陶瓷振荡器（4MHz，12MHz）和 RC 振荡器。

- **低速振荡器**

内部低速振荡器：包括石英/陶瓷振荡器（32KHz）和 RC 振荡器。

- **系统时钟框图**



- HOSC：High_Clk 编译选项。
- LOSC：Low_Clk 编译选项。
- Fhosc：外部高速时钟/内部 PLL 时钟。
- Flosc：外部低速时钟。
- Fosc：系统时钟频率。
- Fcpu：指令周期。

4.2 指令周期Fcpu

系统时钟速率，即指令周期（Fcpu），从系统时钟源分离出来，决定系统的工作速率。Fcpu 的速率由 High_Fcpu 编译选项决定，正常模式下，Fcpu=Fhosc/1~Fhosc/16。当外部时钟选择 4MHz，且 Fcpu 编译选项选择 Fhosc/4 时，则 Fcpu 频率为 4MHz/4=1MHz。低速模式下，Fcpu 固定为 Flosc/4，即 Fcpu 的频率为 32KHz/4=8KHz。

4.3 系统高速时钟

系统高速时钟包括外部高速时钟和内部高速时钟。外部高速时钟又包括 4MHz、12MHz 晶体/陶瓷和 RC 振荡器。内部高速时钟支持指内部 PLL 16MHz 振荡器。高速振荡器由编译选项“High_Clk”选择控制。

4.3.1 HIGH_CLK编译选项

对应不同的时钟功能，SONIX 提供多种高速时钟选项，由 High_CLK 选项控制。High_CLK 选项可以选择 PLL_16M、RC、12M X'tal 和 4M X'tal，以支持不同带宽的振荡器。

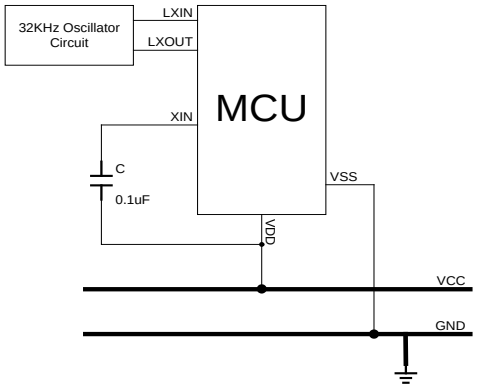
- **PLL_16M**：系统高速时钟源来自内部 PLL 高速 16MHz 振荡器，由外部低速 32K(LXIN\LXOUT)倍频得到。
- **RC**：系统高速时钟源来自廉价的 RC 振荡电路，RC 振荡电路只需和 XIN 连接，XOUT 为 Fcpu 信号的输出引脚。
- **12M X'tal**：系统高速时钟源来自外部高频晶体/陶瓷振荡器，其带宽为 10MHz~16MHz。
- **4M X'tal**：系统高速时钟源来自外部高频晶体/陶瓷振荡器，其带宽为 1MHz~10MHz。

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

4.3.2 内部高速振荡器

内部高速振荡器指 16MHz PLL，普通情况下，精确度为±2%，编译选项（Code Option）选择 PLL_16M 时，使能内部振荡器。

- **PLL_16M**：系统高速时钟来自内部 PLL 高速 16MHz 电路，LXIN/LXOUT 引脚连接外部低速 32KHz 振荡器，XIN 引脚串接一个 0.1uF 的电容到 VDD，XOUT 为 GPIO 引脚。
- **内部高速振荡器应用电路**

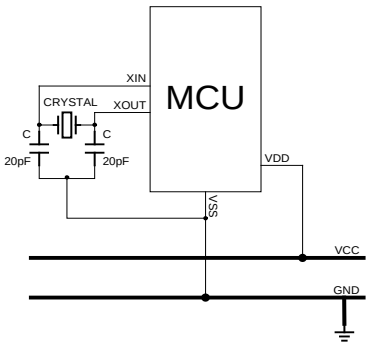


4.3.3 外部高速振荡器

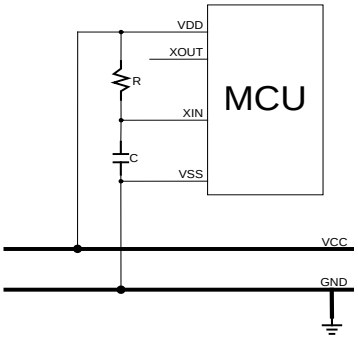
外部高速振荡器包括 4MHz，12MHz 和 RC，由编译选项“High_Clk”控制。4M 和 12M 可以使用晶体和陶瓷振荡器，XIN/XOUT 和 GND 之间需连接一个 20pF 的电容。廉价的 RC 振荡电路只需要和 XIN 引脚连接，电容的容值不能低于 100pF，电阻的阻值由频率决定。

- **外部高速振荡器应用电路**

CRYSTAL/CERAMIC：



RC：



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

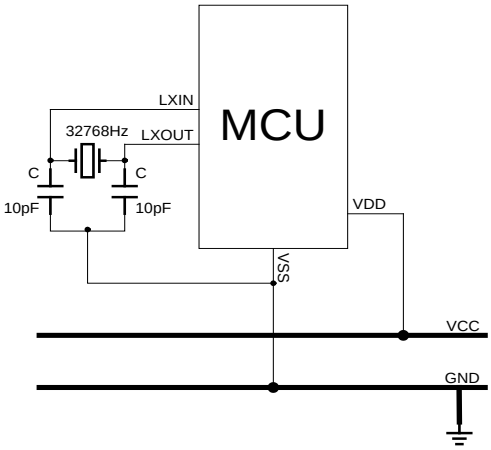
4.4 系统低速时钟

外部低速振荡器包括 32KHz 和 RC，由编译选项“Low_Clk”控制。32KHz 振荡器可以是石英和陶瓷，连接到 LXIN/LXOUT，并在与 GND 之间连接一个 10pF 的电容。RC 振荡器是一个廉价的 RC 电路，包括一个内置电阻，外接一个电容，连接到 LXIN 引脚。电容的容值为 32KHz 时 27pF @3V；39pF @5V。LXOUT 引脚输出 Flosc=32KHz 信号，通过调节 RC 的容值。

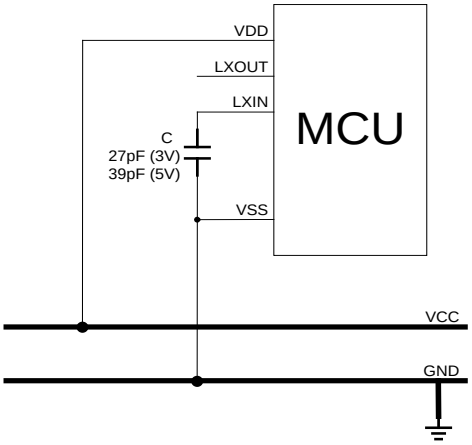
- **32K X’tal：**系统低速时钟源来自外部低速 32768Hz 振荡器。
- **RC：**系统低速时钟源来自外部低廉的 RC 振荡器，RC 振荡电路只需要连接一个电容到 LXIN 引脚，LXOUT 引脚输出 32KHz 信号。

- **外部低速振荡器应用电路**

32KHz CRYSTAL/CERAMIC：



RC：



✷ **注：晶振和电容 C 要尽可能的靠近单片机的 LXIN/LXOUT/VSS 引脚。**

4.5 OSCM寄存器

寄存器 OSCM 控制振荡器的状态和系统的工作模式。

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

Bit 1 **STPHX**：高速振荡器控制位。
0 = 高速时钟正常运行；
1 = 高速振荡器停止，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，系统采用低速时钟。

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

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

- **PLL_16M**：“STPHX=1”，禁止内部高速 PLL 振荡器。
- **RC, 4M, 12M**：“STPHX=1”，禁止外部振荡器。

4.6 系统时钟测量

RC 振荡器的频率可以通过时钟输出引脚进行测量。

- 高速 RC（Hign_Clk 选择 RC）模式下，系统时钟信号从 FCPUO 引脚输出。
- 低速 RC（Low_Clk 选择 RC）模式下，低速振荡器的频率（Flosc）信号从 FLO 引脚输出。
- 系统时钟还可以由软件进行测试。

➢ **例：外部振荡器的 Fcpu 指令周期测试。**

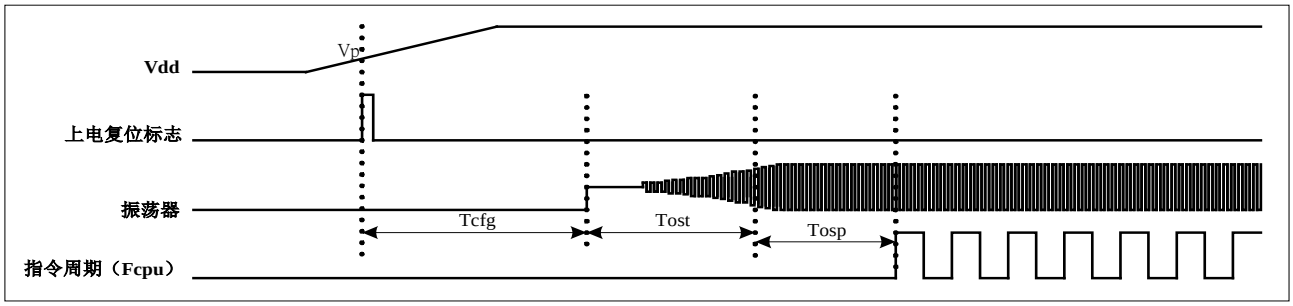
```
        B0BSET      P0M.0           ; P0.0 置为输出模式以输出 Fcpu 的触发信号。

@@:
        B0BSET      P0.0
        B0BCLR      P0.0
        JMP         @B
```

✷ **注：外部高速 RC 模式下，不能直接从 XIN 引脚测试 RC 振荡频率；外部低速 RC 模式下，不能直接从 LXIN 引脚测试 RC 振荡频率。因为探针的连接会影响测试的准确性。**

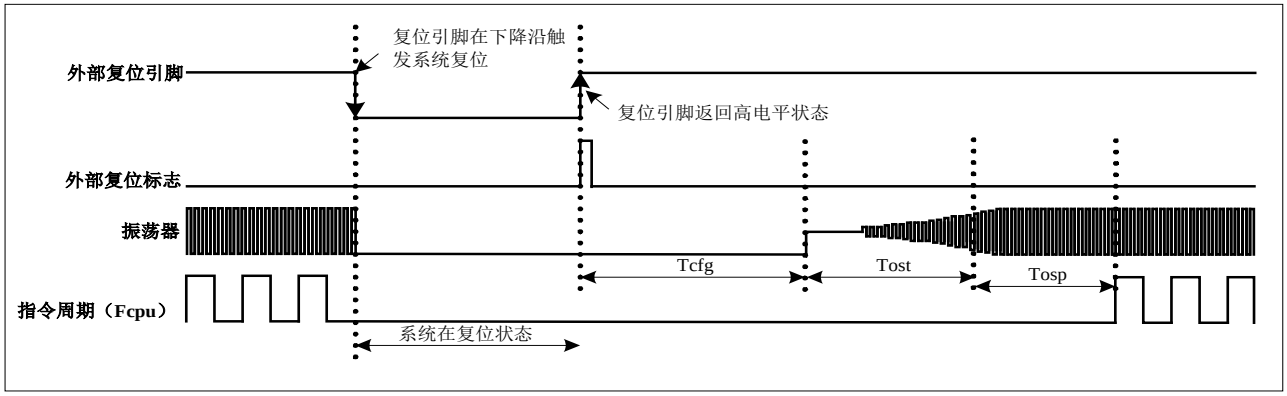
4.7 系统时钟时序

参数	符号	说明	典型值
硬件配置时间	Tcfg	2048*FILRC. （硬件配置时间时钟源由内部低速 RC 振荡器提供，频率为 16KHz @3V, 32KHz @5V）	64ms @ FILRC = 32KHz 128ms @ FILRC = 16KHz
振荡器启动时间	Tost	启动时间取决于振荡器的材料、工艺等。RC 振荡器的启动时间非常短，可以忽略。	-
振荡器起振时间	Tosp	复位情况（上电复位，LVD 复位，看门狗复位，外部复位和睡眠模式被唤醒复位）下，振荡器的起振时间： 外部高速晶振 4M/12M 模式：2048*Fhosc （复位模式） 外部高速晶振 4M/12M 模式：2560*Fhosc （唤醒模式） 外部高速 RC 模式：32*Fhosc 外部低速 32K 晶振模式：(214 +256)*Flosc 外部低速 RC 模式：32*Flosc 内部 PLL 16MHz 振荡器起振时间是外部低速振荡器起振时间+256 个低速时钟。 如： 外部低速晶振模式（PLL 16MHz）：(214 +256)*Flosc 外部低速 RC 模式（PLL 16MHz）：288*Flosc	高速振荡器： 8us @ Fhosc = 4MHz RC 128us @ Fhosc = 16MHz X'tal (Reset modes) 512us @ Fhosc = 4MHz X'tal (Reset modes) 160us @ Fhosc = 16MHz X'tal (Wake-up modes) 640us @ Fhosc = 4MHz X'tal (Wake-up modes) 低速振荡器： 0.52sec @ Flosc = 32KHz X'tal 1ms @ Flosc = 32KHz RC PLL 16MHz: 0.52sec @ Flosc = 32KHz X'tal 9ms @ Flosc = 32KHz RC

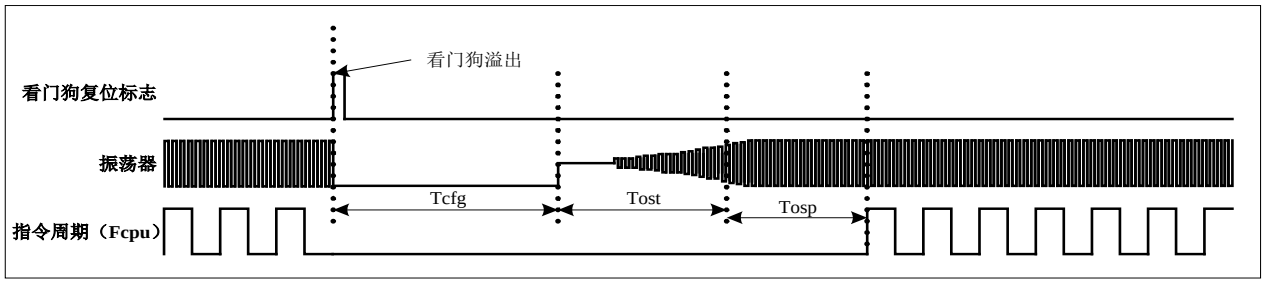
- 上电复位时序：



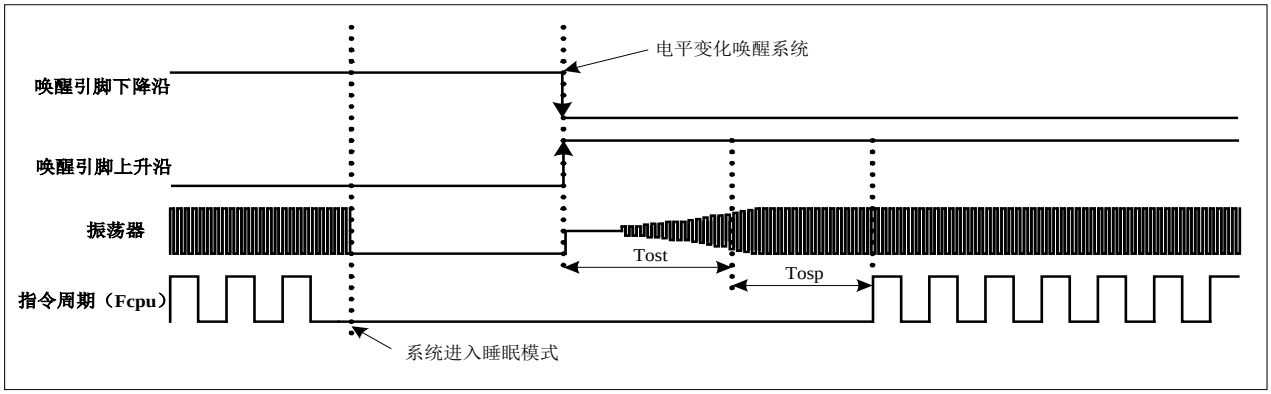
- 外部复位时序：



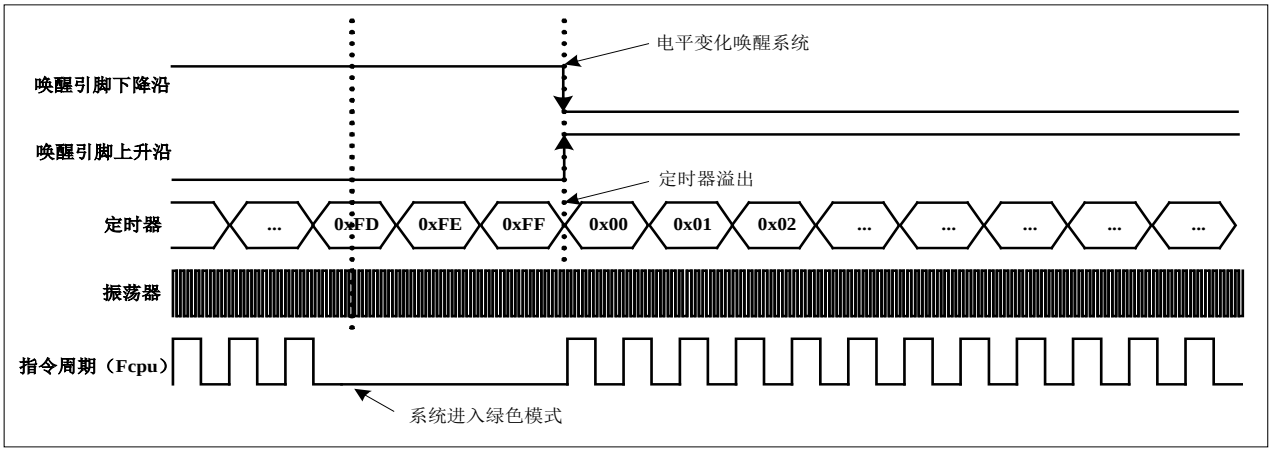
- 看门狗复位时序：



- 睡眠模式唤醒时序：

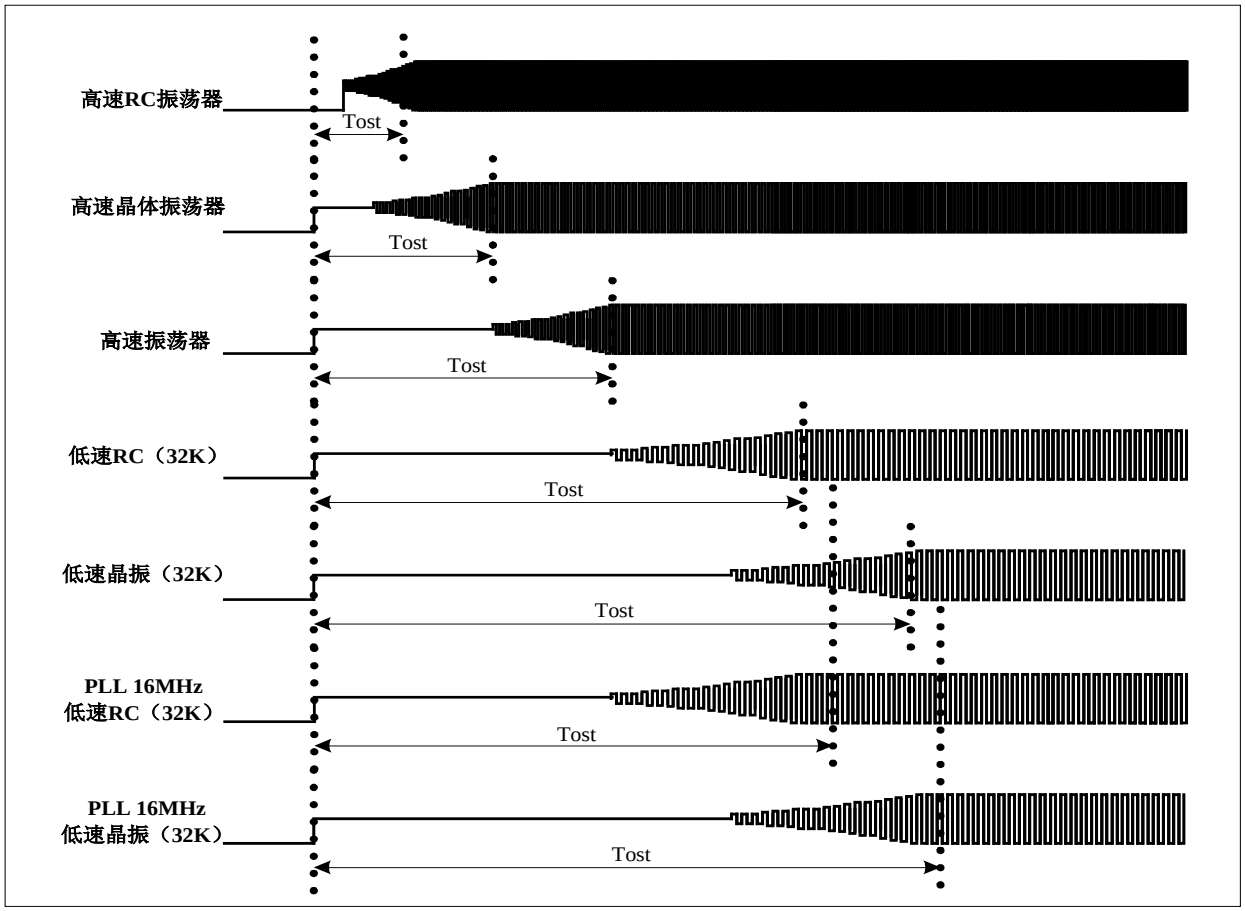


- 绿色模式唤醒时序：



- **振荡器启动时序：**

  启动时间取决于振荡器的材料、工艺等。RC 振荡器的启动时间非常短，可以忽略。



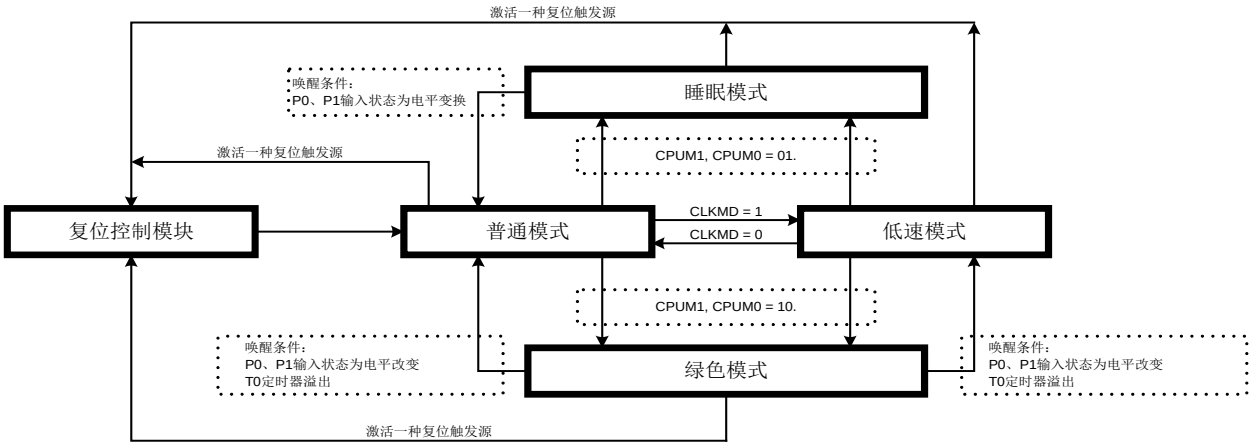
5 系统工作模式

5.1 概述

SN8P2318 可以在 4 种工作模式下以不同的时钟频率工作，这些模式可以控制振荡器的工作、程序的执行以及模拟电路的工作。

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

工作模式控制框图



工作模式时钟控制表：

工作模式	普通模式	低速模式	绿色模式	睡眠模式
EHOSC/IHOSC	运行	STPHX	STPHX	停止
ELOSC	运行	运行	运行	停止
CPU 指令	执行	执行	停止	停止
T0 定时器	T0ENB	T0ENB	T0ENB	无效
TC0 定时器	TC0ENB	TC0ENB	TC0ENB PWM 有效	无效
T1 定时器	T1ENB	T1ENB	无效	无效
LCD 驱动	LCDENB	LCDENB	LCDENB	无效
RFC 功能	RFCENB	RFCENB	RFCENB	无效
看门狗定时器	Watch_Dog 编译选项	Watch_Dog 编译选项	Watch_Dog 编译选项	Watch_Dog 编译选项
内部中断	全部有效	全部有效	T0	全部无效
外部中断	全部有效	全部有效	全部有效	全部无效
唤醒源	-	-	P0, P1, T0, 复位	P0, P1, 复位

- EHOSC：外部高速振荡器（XIN/XOUT）。
- IHOSC：内部高速 PLL 振荡器。
- ELOSC：内部低速振荡器（LXIN/LXOUT）。

5.2 普通模式

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

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

5.3 低速模式

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

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

5.4 睡眠模式

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

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

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

5.5 绿色模式

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

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

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

5.6 工作模式控制宏

Sonix 提供工作模式控制宏以方便系统工作模式的切换。

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

➢ **例：从普通模式/低速模式切换进入睡眠模式。**

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

➢ **例：从普通模式切换进入低速模式。**

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

➢ **例：从低速模式切换进入普通模式（外部高速振荡器停止工作）。**

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

➢ **例：从普通/低速模式切换进入绿色模式。**

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

➢ **例：从普通/低速模式切换进入绿色模式，并使能 T0 唤醒功能。**

```
; 设置定时器 T0 的唤醒功能。
    B0BCLR    FT0IEN            ; 禁止 T0 中断。
    B0BCLR    FT0ENB            ; 禁止 T0 定时器。
    MOV       A,#20H            ;
    B0MOV     T0M,A             ; 设置 T0 时钟= Fcpu / 64。
    MOV       A,#74H
    B0MOV     T0C,A             ; 设置 T0C 的初始值= 74H（设置 T0 间隔值 = 10 ms）。
    B0BCLR    FT0IEN            ; 禁止 T0 中断。
    B0BCLR    FT0IRQ            ; 清 T0 中断请求。
    B0BSET    FT0ENB            ; 使能 T0 定时器。

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

➢ **例：从普通/低速模式切换进入绿色模式，并使能 T0 的唤醒功能，带有 RTC 功能。**

```
    CLR       T0C               ; 清 T0 计数器。
    B0BSET    FT0TB             ; 使能 T0 RTC 功能。
    B0BSET    FT0ENB            ; 使能 T0 定时器。

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

5.7 系统唤醒

5.7.1 概述

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

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

5.7.2 唤醒时间

系统进入睡眠模式后，高速时钟振荡器停止运行。把系统从睡眠模式唤醒时，单片机需要等待 1 个时钟周期的时间，以等待振荡电路稳定工作。唤醒时间结束后，系统进入普通模式。

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

外部高速（12M_X'tal，4M_X'tal）振荡器的唤醒时间的计算如下：

唤醒时间 = 1/Fhosc * 2560（sec）+ 高速时钟启动时间

➢ **例：睡眠模式下，系统被唤醒进入普通模式。唤醒时间的计算如下：**

唤醒时间 = 1/Fhosc * 2560 = 0.64 ms（4MHz 晶振）
总的唤醒时间 = 0.64ms + 振荡器启动时间

外部高/低速 RC 振荡器的唤醒时间的计算如下：

唤醒时间 = 1/Fosc * 32 (sec) + 时钟启动时间

➢ **例：睡眠模式下，系统被唤醒进入普通模式。唤醒时间的计算如下：**

唤醒时间 = 1/Fhosc * 32 = 8 us（4MHz RC）
总的唤醒时间 = 8us + 振荡器启动时间

唤醒时间 = 1/Fhosc * 32 = 1 ms（32KHz RC）
总的唤醒时间 = 1ms + 振荡器启动时间

外部低速晶体振荡器的唤醒时间的计算如下：

唤醒时间（32K_X'tal）= 1/Flosc * (214+256) + 低速时钟启动时间

➢ **例：睡眠模式下，系统被唤醒进入普通模式。唤醒时间的计算如下：**

唤醒时间 = 1/Flosc * (214 +256) = 0.52 sec （32KHz 晶体振荡器）
总的唤醒时间 = 0.52 sec + 振荡器启动时间

内部高速 PLL 16MHz 振荡器的唤醒时间的计算如下：

唤醒时间（32KHz RC 振荡器模式）= 1/Flosc * 288 (sec) + 时钟启动时间

唤醒时间（32KHz 晶体振荡器模式）= 1/Flosc * (214+256) (sec) + 时钟启动时间

➢ **例：睡眠模式学，系统被唤醒进入普通模式。唤醒时间的计算如下：**

唤醒时间 = 1/Flosc * 288 = 9 ms (32KHz RC)
总的唤醒时间 = 9 ms + 振荡器启动时间

唤醒时间 = 1/Flosc * (214+256) = 0.52 sec (32KHz 晶体振荡器)
总的唤醒时间 = 0.52 sec + 振荡器启动时间

✷ **注：高速时钟的启动时间决定于 VDD 和振荡器的类型。**

5.7.3 P1W唤醒控制寄存器

在绿色模式和睡眠模式下，有唤醒功能的 I/O 口能够将系统唤醒到普通模式。P0 和 P1 都具有唤醒功能，二者区别在于：P0 的唤醒功能始终有效，而 P1 的唤醒功能则由寄存器 P1W 控制。

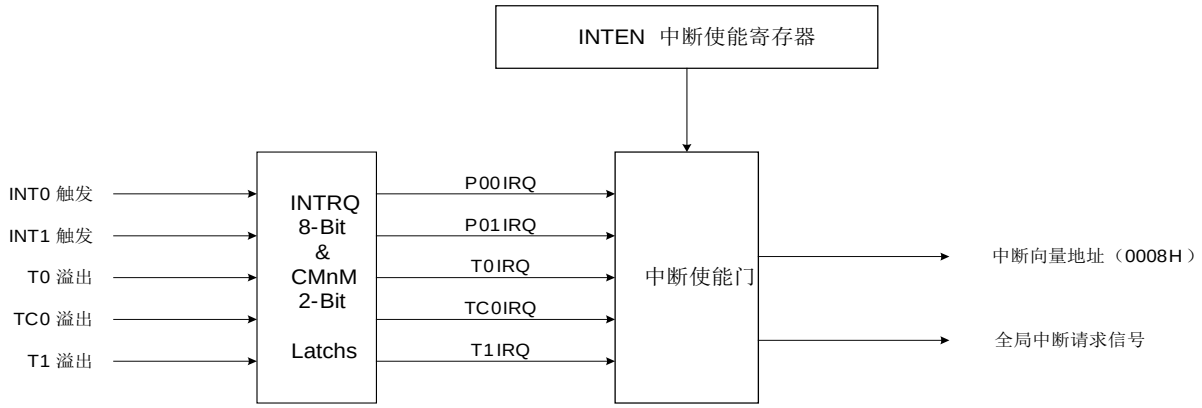
0C0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1W	-	P16W	P15W	P14W	P13W	P12W	P11W	P10W
读/写	-	W	W	W	W	W	W	W
复位后	-	0	0	0	0	0	0	0

Bit[7:0] **P10W~P16W**：P1 唤醒功能控制位。
0 = 禁止；
1 = 使能。

6 中断

6.1 概述

SN8P2318 提供 5 种中断源：3 个内部中断（T0/T1/TC0）和 2 个外部中断（INT0/INT1）。外部中断可以将系统从睡眠模式唤醒进入高速模式，在返回高速模式前，外部中断请求被锁定。一旦程序进入中断，寄存器 STKP 的位 GIE 将被硬件自动清零以避免再次响应其它中断。系统退出中断，即执行完 RETI 指令后，硬件自动将 GIE 置“1”，以响应下一个中断。中断请求存放在寄存器 INTRQ 中。



✷ **注：程序响应中断时，位 GIE 必须处于有效状态。**

6.2 中断使能寄存器INTEN

中断请求控制寄存器 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
INTEN	-	T1IEN	TC0IEN	T0IEN	-	-	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 = 禁止；
1 = 使能。

Bit 1 **P01IEN**：P0.1 外部中断（INT1）控制位。
0 = 禁止；
1 = 使能。

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

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

Bit 6 **T1IEN**：T1 中断控制位。
0 = 禁止；
1 = 使能。

6.3 中断请求寄存器INTRQ

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

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	-	T1IRQ	TC0IRQ	T0IRQ	-	-	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）请求标志位。
0 = INT0 无中断请求；
1 = INT0 有中断请求。

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

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

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

Bit 6 **T1IRQ**：T1 中断请求标志位。
0 = T1 无中断请求；
1 = T1 有中断请求。

6.4 全局中断GIE

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

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

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

➢ **例：设置全局中断控制位（GIE）。**

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

✷ **注：在所有中断中，GIE 都必须处于使能状态。**

6.5 PUSH, POP

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

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

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

```
        ORG     0
        JMP     START

        ORG     8H
        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
```

6.6 外部中断INT0

当外部中断 INT0 被触发时，不管外部中断控制位是否使能，外部中断请求标志位都被置 1。若使能外部中断使能控制位且外部中断请求位置 1 时，程序计数器跳转到中断向量地址 0008H 开始执行中断服务。若禁止外部中断使能控制位，则即使外部中断请求标志位仍然有效，也不会执行中断服务程序。

外部中断内置唤醒锁存功能，就是指系统从睡眠模式唤醒，唤醒源为中断源 P0.0。触发沿的方向与中断沿的配置匹配时，触发沿被锁存，则系统被唤醒后先执行中断服务程序。

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

Bit[4:3] **P00G[1:0]**：INT0 中断触发控制位。

00 = 保留；
01 = 上升沿；
10 = 下降沿；
11 = 上升/下降沿（电平变换触发）。

➢ **例：INT0 中断请求设置，电平触发。**

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

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

➢ **例：INT0 中断。**

```
        ORG     8H              ;
        JMP     INT_SERVICE
INT_SERVICE:

        …                       ; ACC 和 PFLAG 入栈保护。

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

        B0BCLR  FP00IRQ         ; P00IRQ 清零。
        …                       ; INT0 中断服务程序。
        …
EXIT_INT:
        …                       ; ACC 和 PFLAG 出栈恢复。

        RETI                    ; 退出中断。
```

6.7 外部中断INT1

当外部中断 INT1 被触发时，不管外部中断控制位是否使能，外部中断请求标志位都被置 1。若使能外部中断使能控制位且外部中断请求位置 1 时，程序计数器跳转到中断向量地址 0008H 开始执行中断服务。若禁止外部中断使能控制位，则即使外部中断请求标志位仍然有效，也不会执行中断服务程序。

若外部中断的触发方向与唤醒源的触发方向相同时，系统从睡眠模式或绿色模式被 P0.1 唤醒后，INT1 的中断请求被锁存，则立即执行中断服务程序。

✸ **注：INT1 中断请求被 P0.1 唤醒触发源锁存。**

✸ **注：P0.1 下降沿触发中断。**

➢ **例：INT1 中断请求设置。**

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

➢ **例：INT1 中断。**

```
        ORG         8H              ;
        JMP         INT_SERVICE
INT_SERVICE:
        …                           ; ACC 和 PFLAG 入栈保护。
        B0BTS1      FP01IRQ         ; 检测 P01IRQ。
        JMP         EXIT_INT        ; P01IRQ = 0，退出中断。
        B0BCLR      FP01IRQ         ; P01IRQ 清零。
        …                           ; INT1 中断服务程序。
        …
EXIT_INT:
        …                           ; ACC 和 PFLAG 出栈恢复。
        RETI                        ; 退出中断。
```

6.8 T0 中断

T0C 溢出时，无论 T0IEN 处于何种状态，T0IRQ 都会置“1”。若 T0IEN 和 T0IRQ 都置“1”，系统就会响应 T0 的中断；若 T0IEN = 0，则无论 T0IRQ 是否置“1”，系统都不会响应 T0 中断。尤其需要注意多种中断下的情形。

➢ **例：设置 T0 中断。Fcpu = 4MHz / 4。**

```
        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, #64H         ; 初始化 T0C = 64H。
        B0MOV       T0C, A          ; 设置 T0 间隔时间= 10 ms。

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

        B0BSET      FGIE            ; 使能 GIE。
```

➢ **例：T0 中断服务程序。**

```
        ORG         8H
        JMP         INT_SERVICE
INT_SERVICE:

        …                           ; 保存 ACC 和 PFLAG。

        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, #64H
        B0MOV       T0C, A          ;
        …
        …
EXIT_INT:
        …                           ; 恢复 ACC 和 PFLAG。


        RETI                        ; 退出中断。
```

6.9 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, # 74H        ; TC0C 初始值＝74H。
        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         ; 使能 TC0 定时器。

        B0BSET      FGIE            ; 使能 GIE。
```

➢ **例：TC0 中断服务程序。**

```
        ORG         8H              ;
        JMP         INT_SERVICE
INT_SERVICE:

        …                           ; 保存 ACC 和 PFLAG。

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

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

        RETI                        ; 退出中断。
```

6.10 T1 中断

T1C（T1CH、T1CL）溢出时，无论 T1IEN 处于何种状态，T1IRQ 都会置“1”。若 T1IEN 和 T1IRQ 都置“1”，系统就会响应 T1 的中断；若 T1IEN = 0，则无论 T1IRQ 是否置“1”，系统都不会响应 T1 中断。尤其需要注意多种中断下的情形。

➢ **例：T1 中断请求设置。**

```
        B0BCLR      FT1IEN          ; 禁止 T1 中断。
        B0BCLR      FT1ENB          ; 禁止 T1 定时器。
        MOV         A, #20H         ;
        B0MOV       T1M, A          ; 设置 T1 时钟 = Fcpu /32，下降沿触发。
        CLR         T1CH
        CLR         T1CL

        B0BCLR      FT1IRQ          ; 清 T1 中断请求。
        B0BSET      FT1IEN          ; 使能 T1 中断。
        B0BSET      FT1ENB          ; 使能 T1 定时器

        B0BSET      FGIE            ; 使能 GIE。
```

➢ **例：T1 中断服务程序。**

```
        ORG         8H
        JMP         INT_SERVICE
INT_SERVICE:

        PUSH                        ; 保存 ACC 和 PFLAG。

        B0BTS1      FT1IRQ          ; 检查是否有 T1 中断请求标志。
        JMP         EXIT_INT        ; T1IRQ = 0，退出中断。

        B0BCLR      FT1IRQ          ; 清 T1IRQ。
        B0MOV       A, T1CL
        B0MOV       T1CLBUF, A
        B0MOV       A, T1CH
        B0MOV       T1CHBUF, A      ; 保存脉宽。
        CLR         T1CH
        CLR         T1CL
        …                           ; T1 中断服务程序。
        …
EXIT_INT:
        POP                         ; 恢复 ACC 和 PFLAG。


        RETI                        ; 退出中断。
```

6.11 多中断操作

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

中断	有效触发
P00IRQ	由 PEDGE 控制
P01IRQ	P0.1 下降沿触发
T0IRQ	T0C 溢出
T1IRQ	T1CH、T1CL 溢出
TC0IRQ	TC0C 溢出

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

➢ **例：多中断条件下检测中断请求。**

```
                ORG             8               ; 中断向量。
                JMP             INT_SERVICE
INT_SERVICE:

                …                               ; 保存 ACC 和 PFLAG。

INTP00CHK:                                      ; 检查有无 INT0 中断请求。
                B0BTS1          FP00IEN         ; 检查是否使能 P00IEN 中断。
                JMP             INTT0CHK        ; 跳转到下一个中断。
                B0BTS0          FP00IRQ         ; 检查有无 P00IRQ。
                JMP             INTP00          ; 执行 INT0 中断。
INTP01CHK:                                      ; 检查有无 INT1 中断请求。
                B0BTS1          FP01IEN         ; 检查是否使能 P01IEN 中断。
                JMP             INTT0CHK        ; 跳转到下一个中断。
                B0BTS0          FP01IRQ         ; 检查有无 P01IRQ。
                JMP             INTP01          ; 执行 INT1 中断。
INTT0CHK:                                       ; 检查有无 T0 中断请求。
                B0BTS1          FT0IEN          ; 检查是否使能 T0IEN 中断。
                JMP             INTTC0CHK       ; 跳转到下一个中断。
                B0BTS0          FT0IRQ          ; 检查有无 T0IRQ。
                JMP             INTT0           ; 执行 T0 中断。
INTTC0CHK:                                      ; 检查有无 TC0 中断请求。
                B0BTS1          FTC0IEN         ; 检查是否使能 TC0IEN 中断。
                JMP             INTT1CHK        ; 跳转到下一个中断。
                B0BTS0          FTC0IRQ         ; 检查有无 TC0IRQ。
                JMP             INTTC0          ; 执行 TC0 中断。
INTT1CHK:                                       ; 检查有无 T1 中断请求。
                B0BTS1          FT1IEN          ; 检查是否使能 T1IEN 中断。
                JMP             …               ; 跳转到下一个中断。
                B0BTS0          FT1IRQ          ; 检查有无 T1IRQ。
                JMP             INTT1           ; 执行 T1 中断。
                …
                …
INT_EXIT:
                …                               ; 恢复 ACC 和 PFLAG。

                RETI                            ; 退出中断。
```

7 I/O口

7.1 概述

SN8P2318 共有 29 个 I/O 引脚，大多数 I/O 引脚与模拟引脚和特殊功能的引脚共用，详见下表：

I/O 引脚		共用引脚		共用引脚控制条件
名称	类型	名称	类型	
P0.0	I/O	INT0	DC	P00IEN=1
P0.1	I/O	INT1	DC	P01IEN=1
P0.2	I/O	T1IN	DC	T1CKS=1
P0.3	I	RST	DC	Reset_Pin code option = Reset
		VPP	HV	OTP 烧录
P0.4	I/O	XOUT	AC	High_Clk code option = 4M, 12M
		FCPUO	DC	High_Clk code option = RC
P1.0	I/O	RFC0	AC	RFCENB=1, RFCH[2:0]=000b
P1.1	I/O	RFC1	AC	RFCENB=1, RFCH[2:0]=001b
P1.2	I/O	RFC2	AC	RFCENB=1, RFCH[2:0]=010b
P1.3	I/O	RFC3	AC	RFCENB=1, RFCH[2:0]=011b
P1.4	I/O	RFC4	AC	RFCENB=1, RFCH[2:0]=100b
P1.6	I/O	RFCOUT	DC	RFCENB=1, RFCOUT=1
P5.4	I/O	PWM0	DC	TC0ENB=1, PWMOUT=1
P2[3:0]	I/O	SEG[28:31]	DC	PSEG[2:0]=000b
P2[7:4]	I/O	SEG[24:27]	DC	PSEG[2:0]=000b/001b
P3[3:0]	I/O	SEG[20:23]	DC	PSEG[2:0]=000b/001b/010b
P3[7:4]	I/O	SEG[16:19]	DC	PSEG[2:0]=000b/001b/010b/011b

*** DC：数字特性；AC：模拟特性；HV：高压特性。**

7.2 I/O口模式

寄存器 PnM 控制 I/O 口的工作模式。

0B8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0M	-	-	-	P04M	-	P02M	P01M	P00M
读/写	-	-	-	R/W	-	R/W	R/W	R/W
复位后	-	-	-	0	-	0	0	0

0C1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1M	-	P16M	P15M	P14M	P13M	P12M	P11M	P10M
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

0C2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2M	P27M	P26M	P25M	P24M	P23M	P22M	P21M	P20M
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

0C3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P3M	P37M	P36M	P35M	P34M	P33M	P32M	P31M	P30M
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

0C5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5M	-	-	-	P54M	-	-	-	-
读/写	-	-	-	R/W	-	-	-	-
复位后	-	-	-	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：用户可通过位操作指令（B0BSET、B0BCLR）对 I/O 口进行编程控制；**
✸ **注 2：P0.3 是单向输入引脚，P0M.3 未定义。**

➢ **例：I/O 模式选择。**

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

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

B0BCLR  P1M.0       ; P1.0 设为输入模式。

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

➢ **例：LCD 共用引脚控制方式。**

```
; 使能 P2.0~P2.3 GPIO 功能。
        B0BCLR      FPSEG2      ; 设置 PSEG[2:0]=001b。
        B0BCLR      FPSEG1
        B0BSET      FPSEG0

        MOV         A, #0X0F    ; 设置 P2.0~P2.3 为输出模式。
        B0MOV       P2M, A
        …
        B0BCLR      P2M.0       ; 设置 P2.0 为输入模式。

; 使能 P2.0~P2.7 GPIO 功能。
        B0BCLR      FPSEG2      ; 设置 PSEG[2:0]=010b。
        B0BSET      FPSEG1
        B0BCLR      FPSEG0

        MOV         A, #0XFF    ; 设置 P2.0~P2.7 为输出模式。
        B0MOV       P2M, A
        …
        B0BCLR      P2M.0       ; 设置 P2.0 为输入模式。

; 使能 P2.0~P2.7 和 P3.0~P3.3 GPIO 功能。
        B0BCLR      FPSEG2      ; 设置 PSEG[2:0]=011b。
        B0BSET      FPSEG1
        B0BSET      FPSEG0

        MOV         A, #0XFF    ; 设置 P2.0~P2.7 为输出模式。
        B0MOV       P2M, A
        MOV         A, #0X0F    ; 设置 P3.0~P3.3 为输出模式。
        B0MOV       P3M, A
        …
        B0BCLR      P2M.0       ; 设置 P2.0 为输入模式。
        B0BCLR      P3M.0       ; 设置 P3.0 为输入模式。

; 使能 P2.0~P2.7 和 P3.0~P3.7 GPIO 功能。
        B0BSET      FPSEG2      ; 设置 PSEG[2:0]=100b。
        B0BCLR      FPSEG1
        B0BCLR      FPSEG0

        MOV         A, #0XFF    ; 设置 P2.0~P2.7 和 P3.0~P3.7 为输出模式。
        B0MOV       P2M, A
        B0MOV       P3M, A
        …
        B0BCLR      P2M.0       ; 设置 P2.0 为输入模式。
        B0BCLR      P3M.0       ; 设置 P3.0 为输入模式。

; 禁止 P2.0~P2.7 和 P3.0~P3.7 GPIO 功能。
        B0BCLR      FPSEG2      ; 设置 PSEG[2:0]=000b。
        B0BCLR      FPSEG1
        B0BCLR      FPSEG0
```

7.3 I/O口上拉电阻寄存器

输入模式下内置上拉电阻。PnUR 寄存器可以控制上拉电阻，当 PnUR 为 0 时，禁止上拉电阻，为 1 时使能上拉电阻。

0E0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P0UR	-	-	-	P04R	-	P02R	P01R	P00R
读/写	-	-	-	W	-	W	W	W
复位后	-	-	-	0	-	0	0	0

0E1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1UR	-	P16R	P15R	P14R	P14R	P12R	P11R	P10R
读/写	-	W	W	W	W	W	W	W
复位后	-	0	0	0	0	0	0	0

0E2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2UR	P27R	P26R	P25R	P24R	P23R	P22R	P21R	P20R
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

0E3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P3UR	P37R	P36R	P35R	P34R	P33R	P32R	P31R	P30R
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

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

✷ **注：P0.3 为单向输入引脚，无上拉电阻，故 P0UR.3 未定义。**

➢ **例：I/O 口的上拉电阻。**

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

7.4 I/O口数据寄存器

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

0D1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1	-	P16	P15	P14	P13	P12	P11	P10
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

0D2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2	P27	P26	P25	P24	P23	P22	P21	P20
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

0D3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P3	P37	P36	P35	P34	P33	P32	P31	P30
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

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

✸ **注：当通过编译选项使能外部复位时，P03 保持为 1。**

➢ **例：读取输入口的数据。**

```
B0MOV   A, P0       ; 读取 P0 和 P1 口的数据。
B0MOV   A, P1
```

➢ **例：写入数据到输出端口。**

```
MOV     A, #0FFH    ; 写入数据 FFH 到 P0 和 P1。
B0MOV   P0, A
B0MOV   P1, A
```

➢ **例：写入 1 位数据到输出端口。**

```
B0BSET  P1.0        ; P1.0 置 1。

B0BCLR  P1.0        ; P1.0 清 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 振荡器提供，它是将内部 RC 振荡器经 512 分频后送往看门狗定时器进行计数。

看门狗溢出时间 = 8192 /内部低速振荡器周期（sec）

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

看门狗定时器的 3 种工作模式由编译选项“WatchDog”控制：

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

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

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

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

➢ **例：下面是对看门狗定时器操作的示例程序，在主程序的开始清看门狗定时器。**

```
Main:
        MOV     A, #5AH         ; 清看门狗定时器。
        B0MOV   WDTR, A
        …
        CALL    SUB1
        CALL    SUB2
        …
        JMP     Main
```

➢ **例：用宏指令@RST_WDT 清看门狗定时器。**

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

看门狗定时器应用注意事项如下：

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

➢ **例：下面是对看门狗定时器操作的示例程序，在主程序的开始清看门狗定时器。**

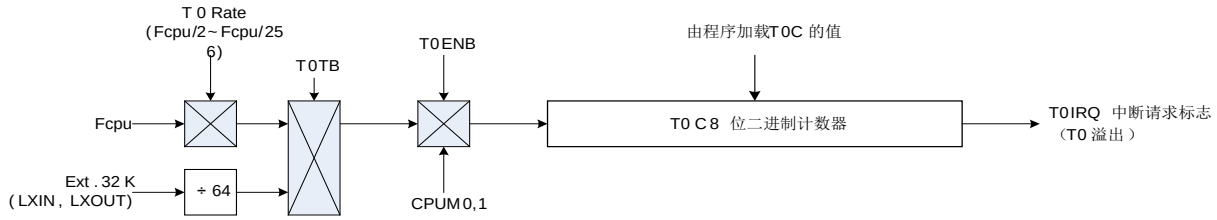
```
Main:
        …                       ; 检查 I/O 口的状态。
        …                       ; 检查 RAM 的内容。
Err:    JMP $                   ; I/O 口或 RAM 出错，不清看门狗等看门狗计时溢出。

Correct:                        ; I/O 口和 RAM 都正确，清看门狗定时器。
                                ;
        MOV     A, #5AH         ; 清看门狗定时器。
        B0MOV   WDTR, A
        …
        CALL    SUB1
        CALL    SUB2
        …
        JMP     Main
```

8.2 8 位基本定时器T0

8.2.1 概述

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

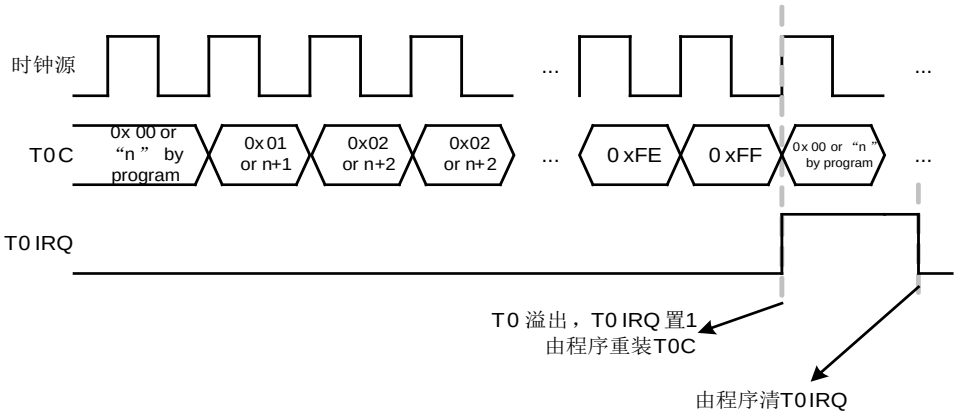
- **8 位可编程计数定时器：**根据选择的时钟频率周期性的产生中断请求。
- **中断功能：**T0 定时器支持中断功能，当 T0 溢出，T0IRQ 有效，程序计数器跳到中断向量地址执行中断。
- **RTC 功能：**T0 支持 RTC 功能，T0TB=1 时，RTC 时钟源由外部低速 32K 振荡时钟（LXIN/LXOUT）提供。
- **绿色模式唤醒功能：**T0 定时器在绿色模式下正常工作，溢出时将系统从绿色模式下唤醒。



✷ **注：RTC 模式下，T0 的间隔时间固定为 0.5S，T0C 计数 256 次。**

8.2.2 T0 定时器操作

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



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

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

8.2.3 T0M模式寄存器

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

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0rate2	T0rate1	T0rate0	-	-	-	T0TB
读/写	R/W	R/W	R/W	R/W	-	-	-	R/W
复位后	0	0	0	0	-	-	-	0

Bit 0 **T0TB**：RTC 时钟源控制位。
0 = 禁止 RTC（T0 时钟源由 Fcpu 提供）；
1 = 使能 RTC。

Bit [6:4] **T0RATE[2:0]**：T0 分频选择位。
000 = fcpu/256；
001 = fcpu/128；
…；
110 = fcpu/4；
111 = fcpu/2。

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

注：RTC 模式下 T0RATE 无效，T0 的间隔时间固定为 0.5S。

8.2.4 T0C计数寄存器

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

0D9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0C	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

T0C 初始值的计算公式如下：

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

- **例：计算 T0C，T0 的间隔时间为 10ms，T0 时钟源 Fcpu = 4MHz/4=1MHz，T0RATE = 001（Fcpu/128）。**
  T0 间隔时间=10ms，T0 时钟 Rate=4MHz/4/128

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

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

8.2.5 T0 操作举例

- **T0 定时器：**

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

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

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

; 清 T0IRQ。
        B0BCLR  FT0IRQ

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

- **T0 在 RTC 模式下工作：**

```
; 复位 T0 定时器。
        MOV     A, #0x00        ; 清 T0M。
        B0MOV   T0M, A

; 设置 T0 RTC 功能。
        B0BSET  FT0TB

; 清 T0C。
        CLR     T0C

; 清 T0IRQ。
        B0BCLR  FT0IRQ

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

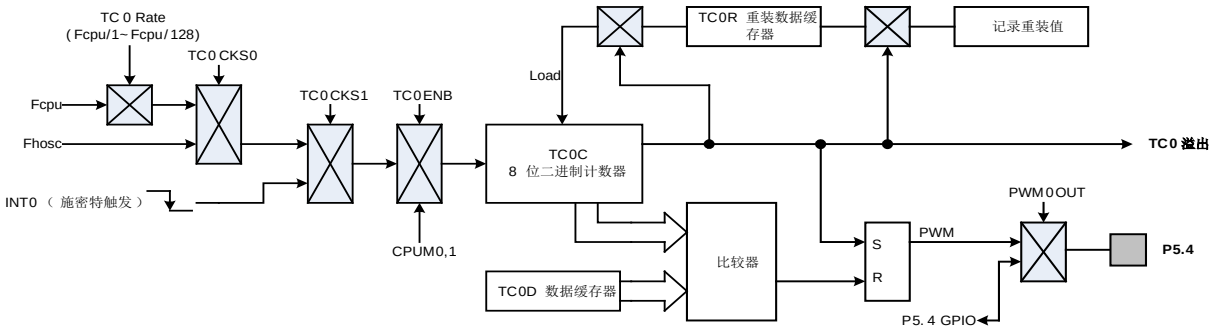
8.3 8 位定时/计数器TC0

8.3.1 概述

8 位二进制定时/计数器具有基本定时器、事件计数器和 PWM 功能。基本定时器功能可以支持中断请求标志的显示（TC0IRQ）和中断操作（中断向量）。由 TC0M、TC0C、TC0R 寄存器控制 TC0 的中断间隔时间。事件计数器可以将 TC0 时钟源由系统时钟（Fcpu/Fhosc）更改为外部时钟信号（如连续的脉冲、R/C 振荡信号等）。TC0 作为计数器时记录外部时钟数目以进行测量应用。TC0 还内置 PWM 功能，其周期/分辨率由 TC0 定时器时钟频率、TC0R 和 TC0D 寄存器控制，故具有良好性能的 PWM 可以处理 IR 载波信号、马达控制和光度调节等。

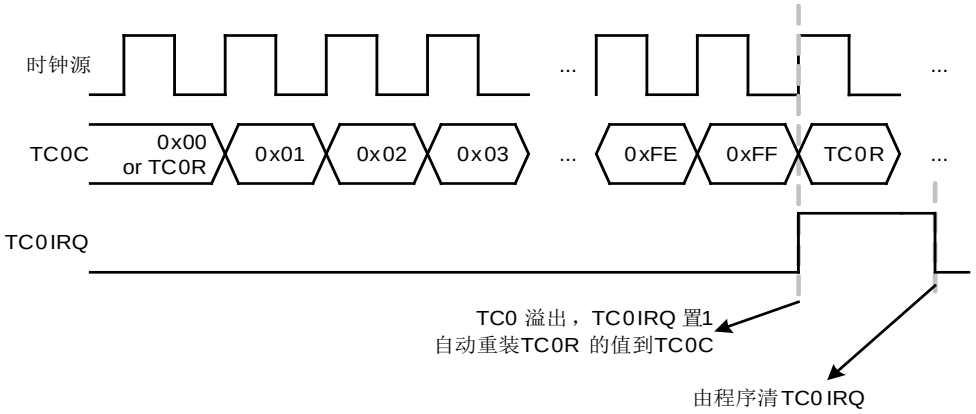
TC0 的主要通途如下：

- **8 位可编程定时器：**根据选择的时钟信号，产生周期中断；
- **中断功能：**TC0 定时器支持中断，当 TC0 溢出时，TC0IRQ 置 1，系统执行中断；
- **外部事件计数器：**对外部事件计数；
- **PWM 输出：**由 TC0R 和 TC0D 寄存器控制占空比/周期；
- **绿色模式功能：**绿色模式下，TC0 正常工作，但无唤醒功能。



8.3.2 TC0 定时器操作

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



TC0 根据不同的时钟源选择不同的应用模式，TC0 的时钟源由 Fcpu（指令周期）、Fhosc（高速振荡时钟）和外部引脚输入（P0.0）提供，由 TC0CKS[1:0]控制。TC0CKS0 选择时钟源来自 Fcpu 或者 Fhosc，当 TC0CKS0=0 时，TC0 时钟源来自 Fcpu，可以由 TC0Rate[2:0]选择不同的分频。当 TC0CKS0=1 时，TC0 时钟源来自 Fhosc。TC0CKS1 决定时钟源由外部引脚输入或者由 TC0CKS0 控制，TC0CKS1=0 时，TC0 的时钟源由 TC0CKS0 控制，TC0CKS1=1 时，TC0 时钟源由外部输入引脚提供，此时使能外部事件计数功能。TC0CKS0=1 或 TC0CKS1=1 时，TC0Rate[2:0]处于无效状态。

TC0CKS0	TC0rate[2:0]	TC0 时钟	TC0 间隔时间 Fhosc=16MHz, Fcpu=Fhosc/2	
			max. (ms)	Unit (us)
0	000b	Fcpu/128	4.096	16
0	001b	Fcpu/64	2.048	8
0	010b	Fcpu/32	1.024	4
0	011b	Fcpu/16	0.576	2.25
0	100b	Fcpu/8	0.256	1
0	101b	Fcpu/4	0.128	0.5
0	110b	Fcpu/2	0.064	0.25
0	111b	Fcpu/1	0.032	0.125
1	无效	Fhosc	0.016	0.0625

8.3.3 TC0M模式寄存器

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

0DAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TC0M	TC0ENB	TC0rate2	TC0rate1	TC0rate0	TC0CKS1	TC0CKS0	-	PWM0OUT
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 0 **PWM0OUT**：PWM 输出控制位。
0 = 禁止 PWM 输出，P5.4 为普通 I/O 引脚；
1 = 允许 PWM 输出，P5.4 输出 PWM 信号。

Bit 2 **TC0CKS 0**：TC0 时钟源选择位。。
0 = Fcpu；
1 = Fhosc。TC0rate[2:0]处于无效状态。

Bit 3 **TC0CKS1**：TC0 时钟源选择位。
0 = 内部时钟（Fcpu 或者 Fhosc，由 TC0CKS0 控制）；
1 = 外部时钟信号（P0.0/INT0），使能事件计数器功能，TC0Rate[2:0]处于无效状态。

Bit [6:4] **TC0RATE[2:0]**：TC0 分频选择位。
000 = Fcpu/128；001 = Fcpu/64；010 = Fcpu/32；011 = Fcpu/16；100 = Fcpu/8；101 = Fcpu/4；
110 = Fcpu/2；111 = Fcpu/1。

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

8.3.4 TC0C计数寄存器

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

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

TC0C 初始值的计算公式如下：

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

8.3.5 TC0R自动重装寄存器

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

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

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

TC0R 初始值计算公式如下：

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

➢ **例：计算 TC0C 和 TC0R 的值，TC0 的间隔时间为 10ms，时钟源来自 Fcpu= 16MHz/16=1MHz，TC0RATE=000（Fcpu/128）。**

TC0 间隔时间为 10ms，TC0 时钟 Rate=16MHz/16/128

TC0C/TC0R 初始值 = 256 -（TC0 中断间隔时间 * 输入时钟）
= 256 -（10ms * 16MHz / 16 / 128）
= 256 -（10-2 * 16 * 106 / 16 / 128）
= B2H

8.3.6 TC0D PWM占空比寄存器

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

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

TC0D 初始值的计算方法如下：

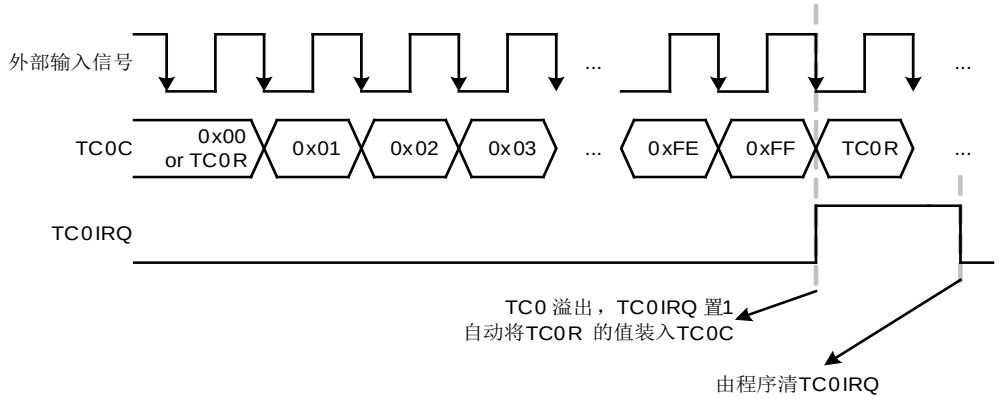
TC0D 初始值 = TC0R + （PWM 脉冲高电平宽度周期 / TC0 时钟 rate）

➢ **例：计算 TC0D 的值。1/3 占空比 PWM，TC0 时钟源 Fcpu=16MHz/16=1MHz，TC0RATE=000（Fcpu/128）。TC0R = B2H，TC0 间隔时间=10ms，PWM 周期频率为 100Hz，1/3 占空比条件下，PWM 高电平的宽度值约为 3.33ms。**

TC0D 初始值 = B2H +（PWM 脉冲高电平宽度值/TC0 时钟 Rate）
= B2H + (3.33ms * 16MHz / 16 / 128)
= B2H + 1AH
= CCH

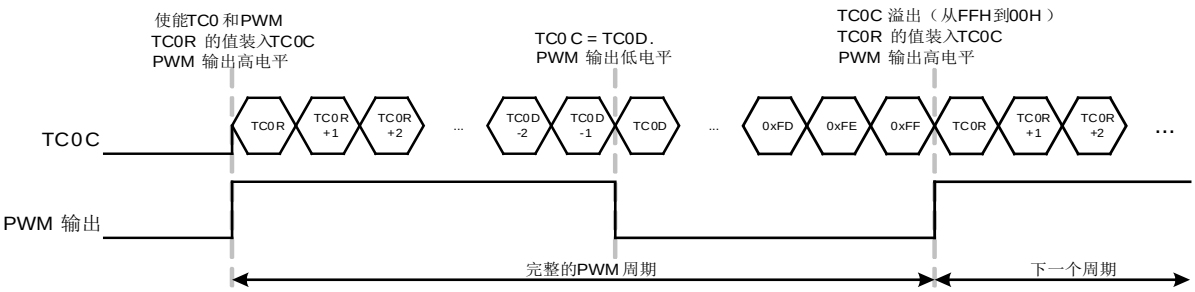
8.3.7 TC0 事件计数器

TC0 作为外部事件计数器时，其时钟源由外部输入引脚（P0.0）提供。当 TC0CKS1=1 时，TC0 的时钟源由外部输入引脚（P0.0）提供，下降沿触发。TC0C 溢出（从 FFH 到 00H）时， TC0 触发事件计数器溢出。使能外部事件计数功能，同时外部输入引脚的唤醒功能被禁止以避免外部事件的触发信号将系统唤醒而耗电。此时，P0.0 的外部中断功能也被禁止，即 P00IRQ=0。外部事件计数器通常用来测量外部连续信号的比率，如连续的脉冲信号，R/C 振荡信号等，外部信号的相位与 MCU 时钟的相位并不同步，通过 TC0 事件计数器的测量和计算以达到不同的应用。

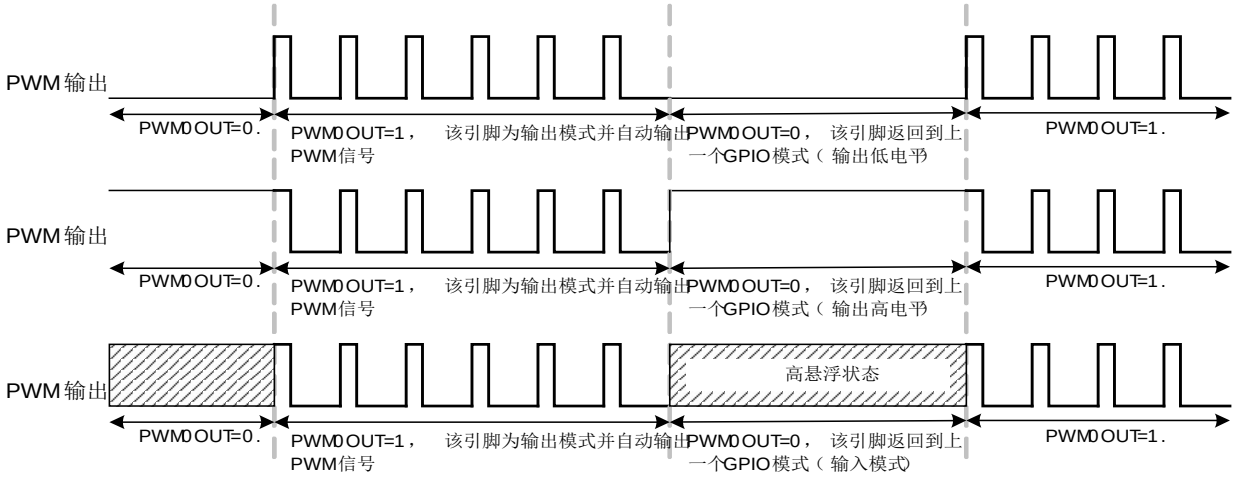


8.3.8 脉宽调制（PWM）

可编程控制占空比/周期的 PWM 可以提供不同的 PWM 信号。使能 TC0 定时器且 PWM0OUT=1 时，由 PWM 输出引脚（P5.4）输出 PWM 信号。PWM 首先输出高电平，然后输出低电平。TC0R 寄存器控制 PWM 的周期，TC0D 控制 PWM 的占空比（脉冲高电平的长度）。开启 TC0 定时器且定时器溢出后，TC0R 重装载 TC0C 的初始值。当 TC0C=TC0D 时，PWM 输出低电平；TC0 溢出时（TC0C 的值从 0FFH 到 00H），整个 PWM 周期完成，并进入下一个周期。TC0 溢出时，TC0R 的值自动装入 TC0C，PWM 的一个周期完成，以保持 PWM 的连贯性。在 PWM 输出的过程由程序更改 PWM 的占空比，则在下一个周期开始输出新的占空比的 PWM 信号。



PWM 的分辨率由 TC0R 决定，而 TC0R 的范围值为 00H~0FFH。当 TC0R=00H 时，PWM 的分辨率为 1/256；TC0R=80H 时，PWM 的分辨率为 1/128。TC0D 控制 PWM 高电平脉冲的宽度，即 PWM 的占空比。当 TC0C=TC0D 时，PWM 输出低电平，TC0D 的值必须大于 TC0R 的值，否则 PWM 的信号保持低电平状态。PWM 输出过程中，TC0 溢出时，TC0IRQ 有效，TC0IEN=1 时，则使能 TC0 中断。但强烈建议小心同时使用 PWM 和 TC0 定时器功能，保证两种功能都能正常工作。PWM 的输出引脚与 GPIO 共用，PWM0OUT=1 时，自动输出 PWM 信号；PWM0OUT=0，即禁止 PWM 时，该引脚自动返回到上一个 GPIO 模式。这样有利于处理 ON/OFF 操作的载波信号，而不控制 TC0ENB 位。



8.3.9 TC0 操作举例

- **TC0 定时器**

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

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

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

; 清 TC0IRQ。
        B0BCLR  FTC0IRQ

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

- **TC0 事件计数器**

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

; 使能 TC0 事件计数器。
        B0BSET  FTC0CKS1        ; 设置 TC0 的时钟源由外部输入引脚（P0.0）提供。

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

; 清 TC0IRQ。
        B0BCLR  FTC0IRQ

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

- **TC0 PWM**

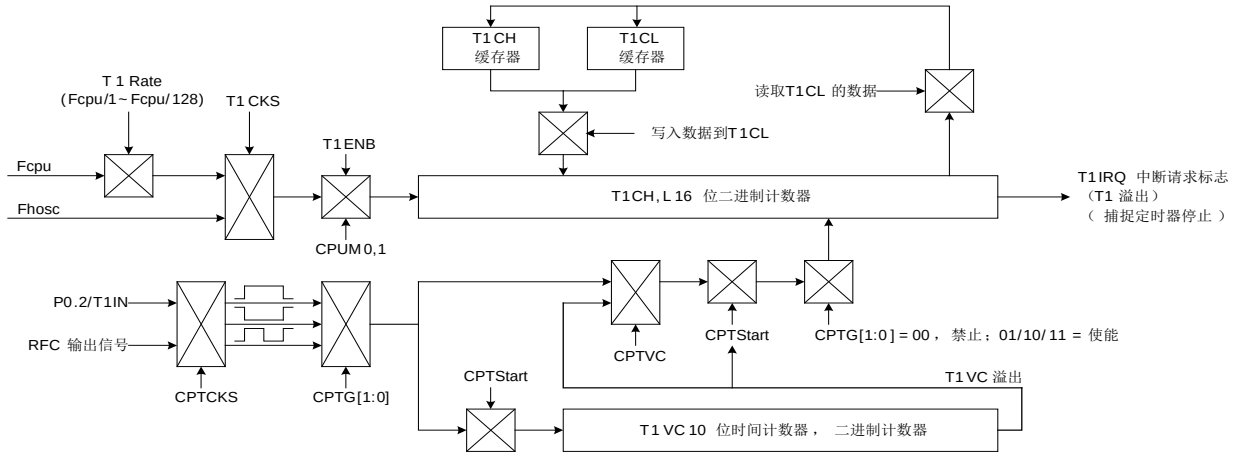
```
; 复位 TC0。
        CLR     TC0M            ; 清 TC0M。

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

; 设置 TC0C 和 TC0R 寄存器获得 PWM 周期。
        MOV     A, #value1
        B0MOV   TC0C, A
        B0MOV   TC0R, A

; 设置 TC0D 寄存器获得 PWM 占空比。
        MOV     A, #value2      ; TC0D 的值必须大于 TC0R 的值。
        B0MOV   TC0D, A

; 使能 PWM 和 TC0 定时器。
        B0BSET  FTC0ENB         ; 使能 TC0 定时器。
        B0BSET  FPWM0OUT        ; 使能 PWM。
```

8.4 16 位定时器T1

8.4.1 概述

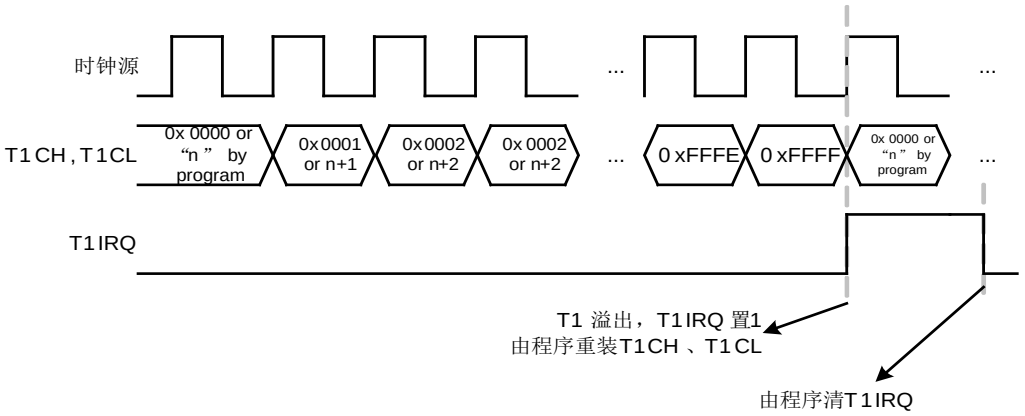
16 位二进制定时器 T1 具有基本定时器和捕捉定时器功能：基本定时器支持中断请求标志的显示（T1IRQ）和中断操作（中断向量），中断间隔时间由 T1M、T1CH/T1CL 计数寄存器控制；捕捉定时器支持脉冲高电平宽度测量、低电平宽度测量和周期测量（由 P0.2/T1IN）提供。T1 作为定时器使用时用来记录外部信号时间参数以进行测量应用。T1 的主要功能如下所示：

- **16 位可编程的计数定时器：**根据选择的时钟频率周期性的产生中断请求。
- **16 位捕捉定时器：**测量输入信号的脉冲宽度和周期，由根据决定捕捉定时器的分辨率的 T1 时钟决定。T1 内置可编程的触发边沿选择装置以决定开始-停止触发事件。
- **10 位事件定时器：**10 位事件计数器检测事件源以积累捕捉时间功能。事件计数器是一个向上计数的设计。当计数器溢出时，T1 停止计数，T1 计数缓存器记录事件计数器持续时间的周期。
- **中断功能：**T1 定时器和捕捉定时器支持中断功能。当 T1 溢出，T1IRQ 有效，程序计数器跳到中断向量地址执行中断。
- **绿色模式功能：**T1 定时器在绿色模式下正常工作，但不能将系统从绿色模式下唤醒。



8.4.2 T1 定时器操作

T1 定时器由 T1ENB 控制。当 T1ENB=0 时，T1 停止工作；当 T1ENB=1 时，T1 开始计数。使能 T1 之前，先设置 T1 的功能模式，如基本定时器、中断功能等。16 位计数器 T1（T1CH、T1CL）溢出（从 0FFFFH 到 0000H）时，T1IRQ 置 1 显示溢出状态并由程序清零。T1 无内置双重缓存器，故 T1 溢出后由程序装载 T1CH、T1CL 的值以确定合适的中断间隔时间。若使能 T1 中断（T1IEN=1），T1 溢出时，程序计数器跳到中断向量地址开始执行中断服务程序，在中断下必须由程序清 T1IRQ。T1 可以在普通模式、低速模式和绿色模式下工作。



T1 根据不同的时钟源选择不同的应用模式，T1 的时钟源由 Fcpu（指令周期）和 Fhosc（高速振荡时钟）提供，由 T1CKS 控制。T1CKS 选择时钟源来自 Fcpu 或者 Fhosc，当 T1CKS=0 时，T1 时钟源来自 Fcpu，可以由 T1Rate[2:0] 选择不同的分频。当 T1CKS=1 时，T1 时钟源来自 Fhosc。详见下表。

T1CKS	T1rate[2:0]	T1 Clock	T1 间隔时间			
			Fhosc=16MHz, Fcpu=Fhosc/4		Fhosc=4MHz, Fcpu=Fhosc/4	
			max. (ms)	Unit (us)	max. (ms)	Unit (us)
0	000b	Fcpu/128	2097.152	32	8388.608	128
0	001b	Fcpu/64	1048.576	16	4194.304	64
0	010b	Fcpu/32	524.288	8	2097.152	32
0	011b	Fcpu/16	262.144	4	1048.576	16
0	100b	Fcpu/8	131.072	2	524.288	8
0	101b	Fcpu/4	65.536	1	262.144	4
0	110b	Fcpu/2	32.768	0.5	131.072	2
0	111b	Fcpu/1	16.384	0.25	65.536	1
1	-	Fhosc/1	4.096	0.0625	16.384	0.25

8.4.3 T1M模式寄存器

模式寄存器 T1M 设置 T1 的工作模式，包括 T1 前置分频器、时钟源和捕捉参数等，这些设置必须在使能 T1 定时器之前完成。

0A0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1M	T1ENB	T1rate2	T1rate1	T1rate0	T1CKS			
读/写	R/W	R/W	R/W	R/W	R/W			
复位后	0	0	0	0	0			

Bit 7 **T1ENB**：T1 启动控制位。
0 = 禁止；
1 = 使能。

Bit [6:4] **T1RATE[2:0]**：T1 分频选择位。
000 = Fcpu/128；001 = Fcpu/64；010 = Fcpu/32；011 = Fcpu/16；100 = Fcpu/8；101 = Fcpu/4；
110 = Fcpu/2；111 = Fcpu/1。

Bit 3 **T1CKS**：T1 时钟源控制位。
0 = Fcpu，由 T1Rate[2:0]分频；
1 = Fhosc。

8.4.4 16 位计数寄存器T1CH, T1CL

16 位计数器 T1CH、T1CL 溢出时，T1IRQ 置 1 并由程序清零，用来控制 T1 的中断间隔时间。首先须写入正确的值到 T1CH 和 T1CL 寄存器，并使能 T1 定时器以保证第一个周期正确。T1 溢出后，由程序自动装入正确的值到 T1CH 和 T1CL 寄存器。

0A1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1CL	T1CL7	T1CL6	T1CL5	T1CL4	T1CL3	T1CL2	T1CL1	T1CL0
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

0A2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1CH	T1CH7	T1CH6	T1CH5	T1CH4	T1CH3	T1CH2	T1CH1	T1CH0
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

16 位定时计数器 T1 为双重缓存器设计，但核心总线只有 8 位，故处理 16 位数据时须锁存标志，以免瞬间状态影响到 16 位数据而出错。写入数据时，写 T1CH 是锁存控制标志；读取数据时，读 T1CL 是锁存控制标志。故写入数据到 T1 时，要先写入数据到 T1CH，再写入数据到 T1CL，即执行写入数据到 T1CL 时，所有数据都已经写入 16 位计数器 T1 中。反之，读取 T1 的数据时，要先读取 T1CL 的数据，再读取 T1CH 的数据，即执行读取 T1CH 时，T1 中的所有数据都已经被读取完毕。

- **读取 T1 计数缓存器中的数据时，要先读取 T1CL 中的数据，再读取 T1CH 中的数据。**
- **写入数据到 T1 计数缓存器中时，要先写入数据到 T1CH，再写入数据到 T1CL。**

16 位计数器 T1（T1CH，T1CL）初始值的计算公式如下：

T1CH，T1 初始值 = 65536 -（T1 中断间隔时间 * T1 时钟比率 T1Rate）

➢ **例：通过计算 T1CH 和 T1CL 的值，获得 T1 的中断间隔时间为 500ms，T1 时钟源为 Fcpu =16MHz/16=1MHz，T1RATE = 000（Fcpu/128）。**

T1 中断间隔时间=500ms，T1Rate=16MHz/16/128

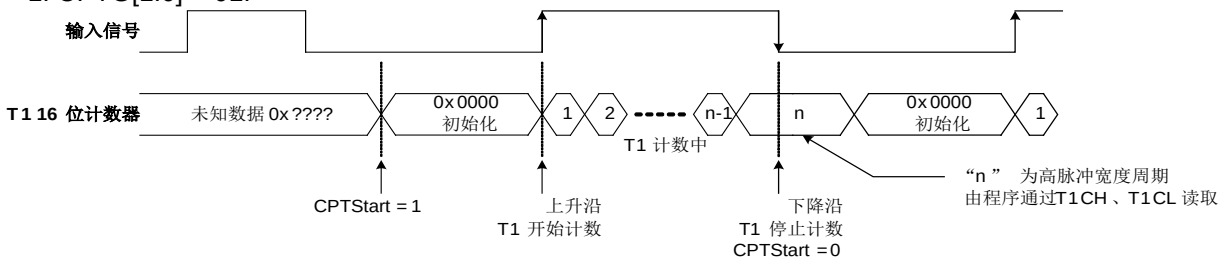
T1CH、T1CL 初始值 = 65536 -（T1 中断间隔时间 * 输入时钟）
= 65536 -（500ms * 16MHz / 16 / 128）
= 65536 -（500*10-3 * 16 * 106 / 16 / 128）
= F0BDH（T1CH = F0H，T1CL = BDH）

8.4.5 T1 捕捉定时器操作

16 位捕捉定时器用来测量输入信号脉冲宽度和周期。当满足触发条件时，T1 开始和停止计数，捕捉定时器功能由 CPTG[1:0]位控制，当 CPTG[1:0]=00 时，禁止捕捉定时器功能；当 CPTG[1:0]=01/10/11 时，使能捕捉定时器功能，但必须要先使能 T1ENB。捕捉定时器可以测量输入脉冲的高电平宽度，输入脉冲的低电平宽度和输入脉冲的周期，由 CPTG[1:0]决定测量内容：当 CPTG[1:0]=01 时，测量输入脉冲的高电平宽度，CPTG[1:0]=10 时，测量输入脉冲低电平宽度，CPTG[1:0]=11 时，测量输入脉冲的周期。CPTG[1:0]只能选择捕捉定时器功能，而不能执行该功能。CPTStart 位是该功能的执行控制位，CPTStart=1 时，捕捉定时器等待合适的触发沿，开始工作；等到第二个合适的触发沿到来时，捕捉定时器停止工作，此时 CPTStart 位被清零，T1IRQ 被置 1。在设置 CPTStart 位之前，16 位计数器 T1 被清零以自动初始化捕捉定时器。

- **高脉冲宽度测量**

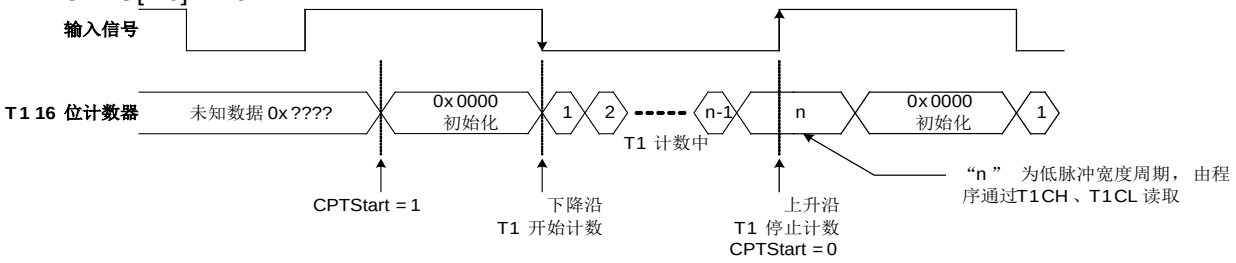
T1ENB = 1. CPTG[1:0] = 01.



测量脉冲高电平宽度：上升沿时，T1 开始计数；下降沿时，T1 停止计数。如果在高电平期间设置 CPTStart 位，捕捉定时器会在下个上升沿时重新测量高电平宽度。

- **低脉冲宽度测量**

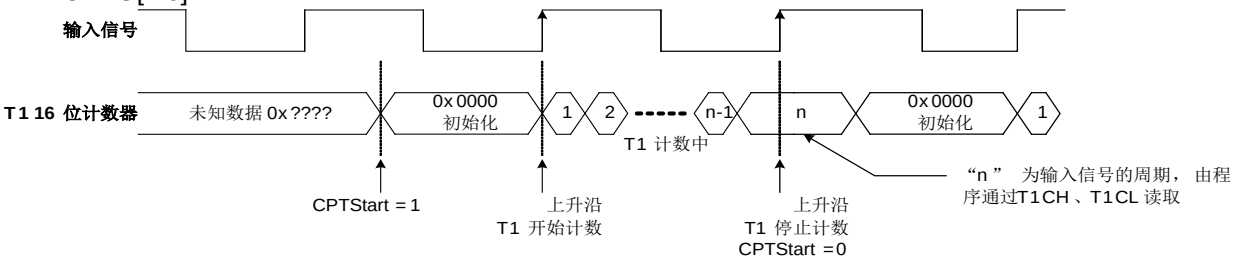
T1ENB = 1. CPTG[1:0] = 10.



测量脉冲低电平宽度：下降沿时，T1 开始计数；上升沿时，T1 停止计数。如果在低电平期间设置 CPTStart 位，捕捉定时器会在下个下降沿时重新测量低电平宽度。

- **输入周期测量**

T1ENB = 1. CPTG[1:0] = 11.



测量输入信号的周期：上升沿时，T1 开始计数；下一个上升沿时，T1 停止计数。如果在高电平和低电平期间设置 CPTStart 位，捕捉定时器会在下个上升沿时重新开始测量。

8.4.6 捕捉定时器控制寄存器

A5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1CKSM	CPTVC				CPTCKS	CPTStart	CPTG1	CPTG0
读/写	R/W				R/W	R/W	R/W	R/W
复位后	0				0	0	0	0

Bit 7 **CPTVC**：事件计数器功能控制位。
0 = 禁止；
1 = 使能。

Bit 3 **CPTCKS**：捕捉定时器时钟源控制位。
0 = 外部输入引脚，P0.2/T1IN；
1 = RFC 输出终端。

Bit 2 **CPTStart**：捕捉定时器计数控制位。
0 = 完成计数；
1 = 开始计数。

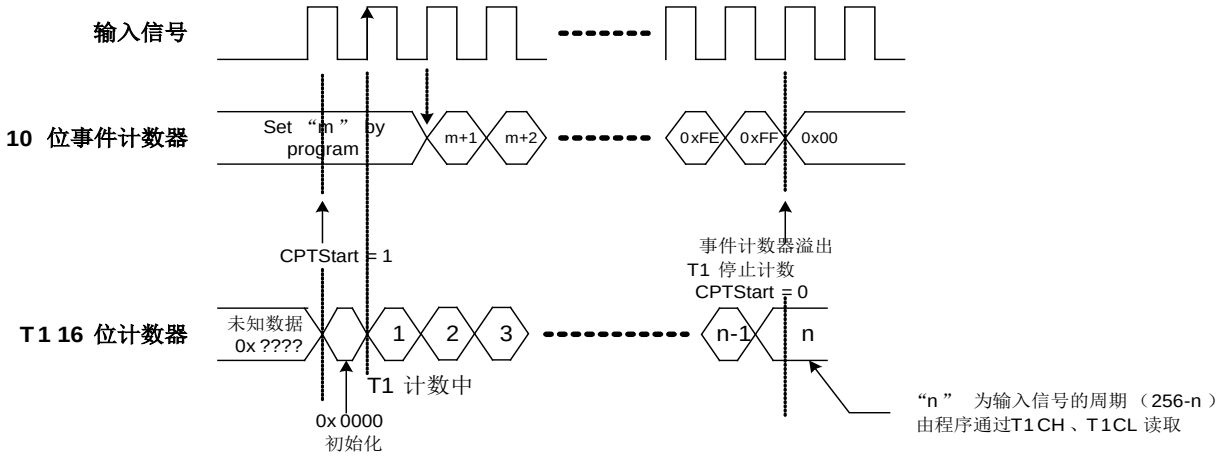
Bit [1:0] **CPTG[1:0]**：捕捉定时器功能控制位。
00 = 禁止捕捉定时器功能；
01 = 测量脉冲高电平宽度；
10 = 测量脉冲低电平宽度；
11 = 测量输入信号周期。

8.4.7 10 位事件计数器功能

10 位事件计数器通过 T1 的触发选择来测量连续输入信号的周期，由 CPTVC 位控制。CPTVC=0 时，禁止事件计数器；CPTVC=1 时，使能事件计数器，但必须使能 T1ENB。事件计数器为上升沿触发，并必须由程序设置 CPTG[1:0]=01。触发后，10 位事件计数器和 16 位定时器 T1 开始计数。T1 事件计数器溢出时，T1 事件计数器和 16 位定时器都停止计数，T1IRQ 置 1。由于执行 T1 事件计数器，16 位定时器必须初始化。

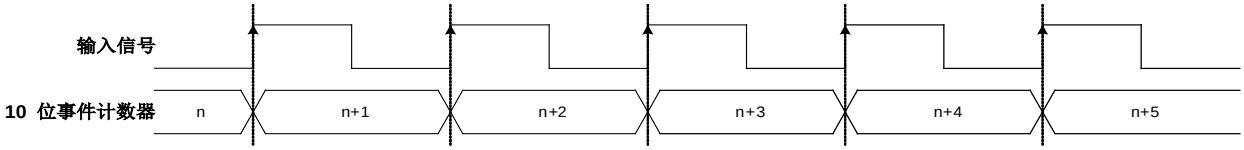
✷ **注：事件计数器的触发源为上升沿，必须由程序设置 CPTG[1:0] = 01。**

- **事件计数器操作**



- **事件计数器触发格式**

捕捉定时器的输入源是高脉冲、低脉冲和周期信号，但事件计数器的触发格式与之不同，事件计数器的触发源固定为上升沿格式。利用捕捉定时器的开始触发信号作为事件计数器的触发源。



8.4.8 10 位事件计数寄存器T1VCH, T1VCL

10 位 T1 事件计数器由 T1VCH 和 T1VCL 组成，T1 溢出时，T1IRQ 置 1 并由程序清零。T1VCH 和 T1VCL 决定 T1 事件计数器的数量。

0A3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1VCL	T1VCL7	T1VCL6	T1VCL5	T1VCL4	T1VCL3	T1VCL2	T1VCL1	T1VCL0
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

0A4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1VCH	-	-	-	-	-	-	T1VCH1	T1VCH0
读/写	-	-	-	-	-	-	R/W	R/W
复位后	-	-	-	-	-	-	0	0

10 位事件计数器 T1 的核心总线只有 8 位，故处理 10 位数据时须锁存标志，以免瞬间状态影响到 10 位数据而出错。写入数据时，写 T1VCH 是锁存控制标志；读取数据时，读 T1VCL 是锁存控制标志。故写入数据到 T1 时，要先写入数据到 T1VCH，再写入数据到 T1VCL，即执行写入数据到 T1VCL 时，所有数据都已经写入 10 位事件计数器 T1 中。反之，读取 T1 的数据时，要先读取 T1VCL 的数据，再读取 TV1CH 的数据，即执行读取 T1VCH 时，T1 中的所有数据都已经被读取完毕。

- **读取 T1 事件计数缓存器中的数据时，要先读取 T1VCL 中的数据，再读取 T1VCH 中的数据。**
- **写入数据到 T1 事件计数缓存器中时，要先写入数据到 T1VCH，再写入数据到 T1VCL。**

8.4.9 T1 操作举例

- **T1 定时器：**

```
; 复位 T1。
        MOV         A, #00H             ; 清 T1M。
        B0MOV       T1M, A

; 设置 T1Rate。
        MOV         A, #0nnn0000b
        B0MOV       T1M, A

; 设置 T1CH，T1CL 寄存器获取 T1 中断间隔时间。
        MOV         A, #value1          ; 先设置高字节。
        B0MOV       T1CH, A
        MOV         A, #value2          ; 设置低字节。
        B0MOV       T1CL, A

; 清 T1IRQ。
        B0BCLR      FT1IRQ

; 使能 T1 定时器和中断功能。
        B0BSET      FT1IEN              ; 使能 T1 中断功能。
        B0BSET      FT1ENB              ; 使能 T1 定时器。
```

- **T1 捕捉定时器，测量信号周期。**

```
; 复位 T1。
        MOV         A, #00H             ; 清 T1M。
        B0MOV       T1M, A

; 设置 T1Rate，选择输入源，选择/使能 T1 捕捉定时器功能。
        MOV         A, #0nnnm000b       ; “nnn”代表 T1rate[2:0]。
        B0MOV       T1M, A              ; “m”代表 T1 时钟源控制位。
        MOV         A,#000000mmb        ; “mm”代表 CPTG[1:0]，选择 T1 捕捉定时器的功能。
        B0MOV       T1CKSM,A            ; CPTG[1:0] = 01b，测量脉冲的高电平宽度。
                                        ; CPTG[1:0] = 10b，测量脉冲的低电平宽度。
                                        ; CPTG[1:0] = 11b，测量脉冲的周期。

; 选择 T1 捕捉源。
        B0BCLR      FCPTCKS             ; 捕捉源为 P0.0.。
; or
        B0BSET      FCPTCKS             ; 捕捉源为比较器输出。

; 清 T1CH，T1CL。
        CLR         T1CH                ; 先清除高字节。
        CLR         T1CL                ; 再清除低字节。

; 清 T1IRQ。
        B0BCLR      FT1IRQ

; 使能 T1 定时器和中断功能。
        B0BSET      FT1IEN              ; 使能 T1 中断功能。
        B0BSET      FT1ENB              ; 使能 T1 定时器。

; 设置捕捉定时器开始位。
        B0BSET      FCPTStart
```

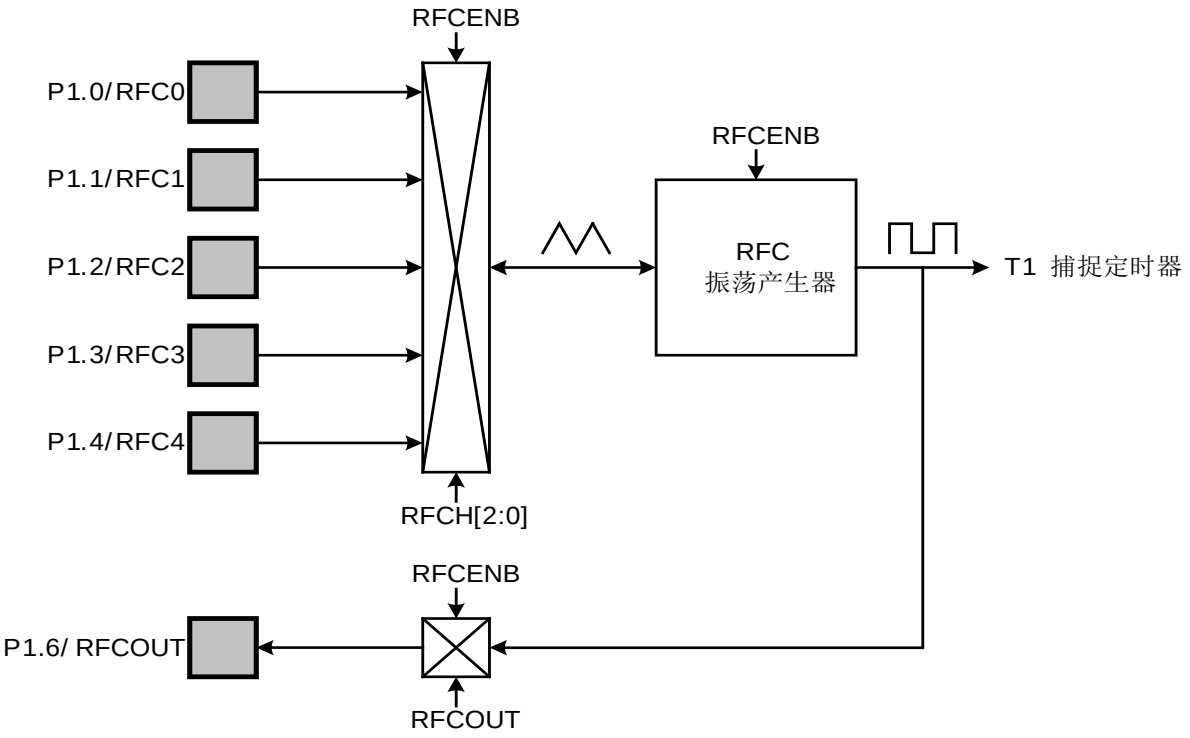
- **T1 事件计数器，测量连续的信号。**

```
; 复位 T1。
        CLR         T1M             ; 清 T1M。

; 设置 T1 时钟 Rate，选择/使能 T1 捕捉定时器。
        MOV         A, #0nnnm000b   ; “nnn”代表 T1rate[2:0]选择 T1 时钟 Rate。
        B0MOV       T1M, A          ; “m”代表 T1 时钟源控制位。
        MOV         A, #00000001b   ; 设置捕捉定时器功能。
        B0MOV       T1CKSM, A       ; CPTG[1:0]必须设置为 01。
                                    ; 测量高脉冲宽度。
; 清 T1CH, T1CL。
        CLR         T1CH            ; 先清除高字节。
        CLR         T1CL            ; 再清除低字节。

; 设置 10 位捕捉定时器 T1VCH, T1VCL 测量连续的信号。
        MOV         A, #value1      ; 先设置高字节。
        B0MOV       T1VCH, A
        MOV         A, #value2      ; 再设置低字节。
        B0MOV       T1VCL, A

; 清 T1IRQ。
        B0BCLR      FT1IRQ

; 使能 T1 定时器，中断功能和 T1 捕捉定时器功能。
        B0BSET      FT1IEN          ; 使能 T1 中断功能。
        B0BSET      FT1ENB          ; 使能 T1 定时器。
        B0BSET      FCPTVC          ; 使能 T1 事件计数器功能。
; 设置捕捉定时器起始位。
        B0BSET      FCPTStart
```

9 电阻频率转换（RFC）

9.1 概述

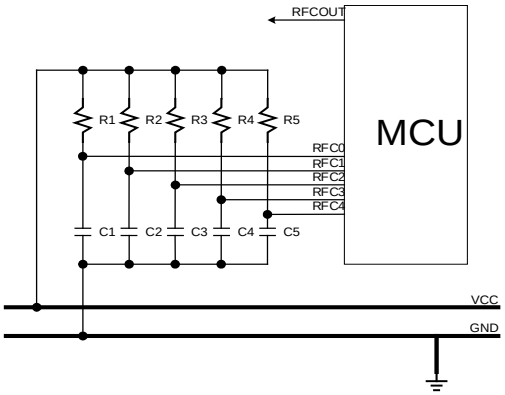
SN8P2318 内置阻频转换模块（RFC），该模块由电阻（R）和电容（C）构成的振荡电路产生，具体器件包括：一个阻抗传感器、一个参考电阻和一个电容。阻抗传感器的阻值或者参考电阻连接到 RFC 输入通道，通过 CR 振荡器转换成频率，时钟的数量由内置的测量定时/计数器 T1 记录。通过读取 T1 测量值可以获得电阻检测到的数字转换值。多种传感器电路容易实现 RFC 功能，如温度测量就采用一个温度调节器。RFC 共用 5 个输入通道：包括一个参考通道和 4 个传感器通道。RFC 的频率随阻抗传感器的不同而不同。T1 还提供脉宽测量和输入频率测量功能来测量 RFC 时钟的高/低速度。RFC 引脚与 P1 引脚共用，RFCENB=1 时，P1.0~P1.4 为 RFC0~RFC4 通道，由 RFCH[2:0]控制。P1.6 与 RFCOUT 引脚共用，由 RFCM 寄存器的 RFCOUT 位控制。RFC 引脚与 GPIO 引脚共用，由 RFCM 寄存器控制。RFC 在绿色模式下仍然工作，但没有唤醒功能。



- **RFC 通道：**RFC 通过与 GPIO 引脚共用，由 RFCENB=1 和 RFCH[2:0]选择输入通道。RFCENB 必须置 1，否则 RFCH[2:0]会将 RFC 输入通道设置为 GPIO 引脚。
- **RFCOUT 引脚：**RFC 输出引脚与 GPIO 引脚共用，由 RFCOUT 位控制。RFC 输出引脚输出 RFC 振荡信号，通过 RFC 振荡产生器产生。RFC 振荡信号也输入至 T1 捕捉定时器。

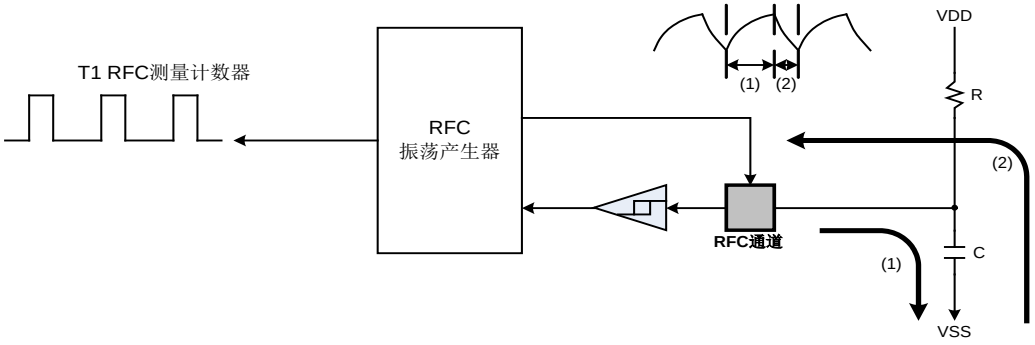
9.2 RFC应用电路

RFC 应用电路由一个电阻装置和一个电容装置组成。通常情况下，电阻是一个传感器设备，而电容则是一个静态的设备，其连接示意图如下所示。电阻连接 Vdd 到 RFC 通道，电容连接 Vss 到 RFC 通道。使能 RFC 功能时，内部 RFC 振荡产生器使 RFC 通道开始振荡，其频率由 R 和 C 的值决定。



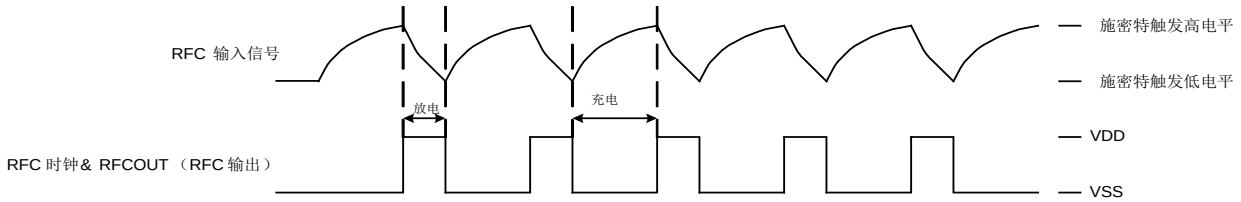
RC 电路由 RFCM 寄存器的 RFCH[2:0]选择切换，当选择其中一个 RFC 输入通道时，其他的 RFC 输入通道都会被设置为 GPIO 引脚。建议设置 RFC0~RFC4（P1.0~P1.4）为输入悬浮状态。RFC 数字转换时钟从 RFCOUT（P1.6）引脚输出，由 RFC 寄存器的 RFCOUT 位控制。

9.3 RFC操作



RFC 根据电容的充放电而转换时钟。当 RFCENB = 1 时使能 RFC 功能，RFC 通道会对外部 RC 电路充电，直至电压高于 RFC 通道的 ViH。然后 RFC 通道切换到放电模式，直至电压低于 ViL。充电/放电使 RFC 振荡。

上图中的循环（1）是一个充电模式：RFC 通道对电容充电，电压上升，当 RFC 通道检测到电压为施密特触发高电平时，RFC 通道切换到放电模式。循环（2）就是一个放电模式：RFC 通道对电容放电，电压下降。当 RFC 通道检测到电压为施密特触发低电平时，RFC 通道切换到充电模式。通过 RC 振荡控制电路将充电/放电的切换结果进行转换，产生数字时钟。



RFC 电路中的电容的值保持不变。RFC 时钟频率由电阻和充/放电的速率决定。根据 RFC 转换得到的频率结果和 RFC 的数字转换时钟，可以对阻抗传感器进行测量。RFC 可以做 T1 的时钟源，并通过频率测量和脉宽测量来测量 RFC 的转换结果。当 RFCOUT = 1 时，RFC 时钟还可以通过 RFCOUT 输出。

9.4 RFCM寄存器

0A6H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RFCM	RFCENB	-	0	1	RFCOUT	RFCH2	RFCH1	RFCH0
读/写	R/W	-	W	W	R/W	R/W	R/W	R/W
复位后	0	-	0	0	0	0	0	0

Bit 7 **RFCENB**：RFC 功能控制位。
0 = 禁止 RFC；
1 = 使能 RFC。

Bit 5 **RFCM.5** 必须由程序设置为 0。
Bit 4 **RFCM.4** 必须由程序设置为 1。

Bit 3 **RFCOUT**：RFC 输出控制位。
0 = 禁止 RFC 输出，P1.6 为 GPIO 引脚；
1 = 使能 RFC 输出，P1.6 为 RFCOUT 引脚。

Bit[2:0] **RFCH[2:0]**：RFC 输入通道选择控制位。
000 = 选择 RFC0 通道，禁止 P1.0 的 GPIO 功能，P1.1~P1.4 为 GPIO 引脚；
001 = 选择 RFC1 通道，禁止 P1.1 的 GPIO 功能，P1.0，P1.2~P1.4 为 GPIO 引脚；
010 = 选择 RFC2 通道，禁止 P1.2 的 GPIO 功能，P1.0，P1.1，P1.3，P1.4 为 GPIO 引脚；
011 = 选择 RFC3 通道，禁止 P1.3 的 GPIO 功能，P1.0~P1.2，P1.4 为 GPIO 引脚；
100 = 选择 RFC4 通道，禁止 P1.4 的 GPIO 功能，P1.0~P1.3 为 GPIO 引脚；
101~111 = 保留，P1.0~P1.4 为 GPIO 引脚。

✸ 注：
1、使能 RFC 之前，必须首先由程序将 P1.0~P1.4 设置为无上拉的输入模式。
2、由于 RFCM.4 为只写位，故必须通过指令 MOV/B0MOV 将该位设置为 1。
3、由于 RFCM.5 为只写位，故必须通过指令 MOV/B0MOV 将该位设置为 0。
4、强烈建议通过指令 MOV/B0MOV 设置 RFCM 寄存器。

9.5 RFC操作举例

- **例：通过 T1 事件计数器测量 RFC 信号。**

```
; 设置 T1 定时器模式。
        CLR     T1M             ; T1 时钟为 Fcpu，rate 为 Fcpu/128。
        MOV     A, #0           ; 清 T1 计数缓存器。
        B0MOV   T1CH, A
        B0MOV   T1CL, A

; 设置 T1 事件计数器模式。
        MOV     A, #10001000B   ; 使能 T1 捕捉定时器。
        B0MOV   T1CKSM, A
        MOV     A, #0           ; 设置 T1 事件计数缓存器。
        B0MOV   T1VCH, A
        MOV     A, #0xFF
        B0MOV   T1VCL, A

        B0BCLR  FT1IRQ          ; 清 T1 中断请求标志。

; 设置 RFC。
        CLR     P1UR            ; 禁止 P1 上拉电阻。
        MOV     A, #11100000B
        AND     P1M, A          ; 设置 P1.0~P1.4 为输入模式。
        MOV     A, #10010000B   ; 使能 RFC，并选择 RFC0 通道。
        B0MOV   RFCM, A

        B0BSET  FT1ENB          ; 使能 T1 定时器。
        B0BSET  FCPTStart       ; 开始测量 RFC 频率。

; 检测 T1IRQ=1。
Chk_T1IRQ:
        B0BTS1  FT1IRQ          ; 检测 T1IRQ=1。
        JMP     Chk_T1IRQ
        B0MOV   A, T1CL         ; RFC 测量结束。
        …
        B0MOV   A, T1CH
        …
```

10 4x32 LCD驱动

10.1 概述

SN8P2318 内置 4x32（4 个 com 口和 32 个 seg 口，共 128 点）的 LCD 驱动模块，支持 1/4 占空比和 1/2、1/3 偏压的 LCD 面板。LCD 帧速率为 64Hz，可选择外部 32768Hz 晶振或 RC 振荡电路作为时钟源。LCD 有 R 型和 C 型结构：C 型只支持 1/3 偏压 LCD 机构；R 型采用外部偏压电路提供 LCD 电源和偏压。共有 16 个 GPIO 引脚与 SEG 口共用，由寄存器控制。应用中针对不同的 LCD 面板，用户可以决定更多的 GPIO 引脚。

10.2 LCD寄存器

0CBH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LCDM	CPCK1	CPCK0	VLCDCP	PSEG2	PSEG1	PSEG0	BIAS	LCDENB
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 [7:6] **CPCK[1:0]**：VLCD charge-pump 时钟选择控制位。

CPCK[1:0]	Charge-pump 时钟
00	32KHz
01	16KHz
10	4KHz
11	1KHz

✷ **注：通常情况下，1KHz 的 charge-pump 时钟可以适用于大多数应用。低 charge-pump 时钟频率能更省电，高 charge-pump 时钟频率有更强的 VLCD 驱动能力。**

Bit 5 **VLCDCP**：VLCD charge-pump 控制位。
0 = 禁止，LCD 为 R 型；
1 = 使能，LCD 为 C 型。

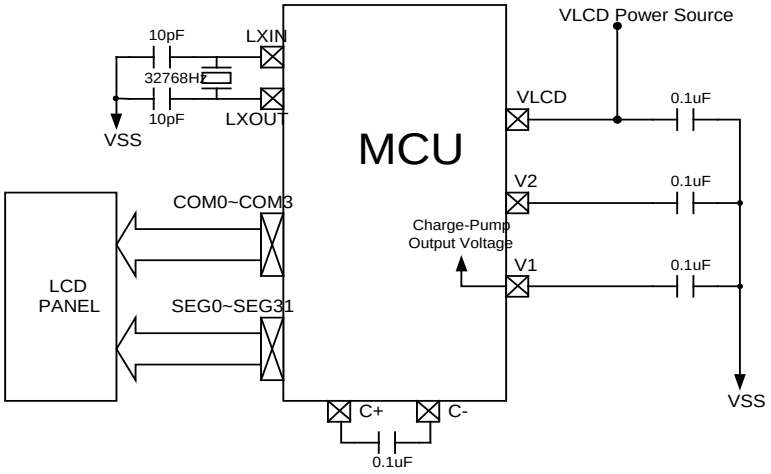
Bit [4:2] **PSEG2**：LCD 共用引脚控制位。
000 = 禁止所有 LCD 共用引脚的 GPIO 功能，SEG 引脚共 32 个；
001 = 使能 P2.0~P2.3（SEG28~SEG31）的 GPIO 功能，SEG 引脚共 28 个；
010 = 使能 P2.0~P2.7（SEG24~SEG31）的 GPIO 功能，SEG 引脚共 24 个；
011 = 使能 P2.0~P2.7 和 P3.0~P3.3（SEG20~31）的 GPIO 功能，SEG 引脚共 20 个；
100 = 使能 P2.0~P2.7 和 P3.0~P3.7（SEG16~31）的 GPIO 功能，SEG 引脚共 16 个；
101~111 = 保留。

Bit 1 **BIAS**：LCD 偏压控制位。
0 = 1/3 偏压（C 型和 R 型）；
1 = 1/2 偏压（R 型）。

Bit 0 **LCDENB**：LCD 控制位。
0 = 禁止；
1 = 使能。

10.3 C型LCD

C 型模式下只支持 1/3 偏压的 LCD 面板，LCD 的电源 VLCD 由外部 VDD 提供，C 型的功耗低于 R 型，这是因为没有外部 DC 偏压电路消耗功率电流。charge-pump 的电压电平由 VLCD 电压提供：V1 为 charge-pump 源，其电平为 1/3 * VLCD；V2 为 2*V1，其电平为 2/3 * VLCD。C 型 LCD 模式下，VLCD 寄存器的 VLCDCP 位必须置 1。



C 型 LCD 应用电路

- LCD 的 C 型模式只支持 1/3 偏压。
- C 型模式下，需在 C+和 C-引脚之间连接一个 0.1uF 的电容。
- 必须在 VLCD/V1/V2 与 VSS 之间连接一个 0.1uF 的电容，以使电源稳定。
- V1 的电压是 charge-pump 源电压，为 1/3 * VLCD。
- V2 的电压为 2 * V1 = 2/3 * VLCD。

✷ **注：VLCD 的电压源来自外部电压源，由 LCD 面板的电压电平决定，VLCD 的电压电平不能高于单片机的 Vdd。**

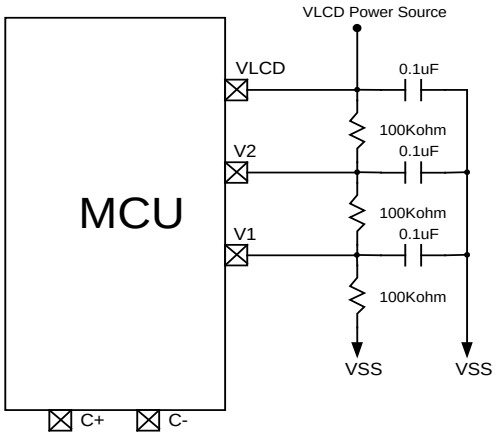
➢ **例：设置 C 型 LCD 模式。**

```
; 设置 C 型 LCD。
        MOV     A, #nn0mmm00B   ;“nn”选择 charge pump 的时钟 rate。
        B0MOV   LCDM            ;“mmm”控制 P2/P3 LCD 共用引脚。
                                ; BIAS = 0，选择 1/3 偏压结构。
; 使能 charge pump。
        B0BSET  FVLCDCP         ; 使能 LCD charge-pump，LCD 切换到 C 型。

; 使能 LCD 驱动。
        B0BSET  FLCDENB         ; 使能 LCD 驱动。

; LCD 图像处理。
        …
        …
```

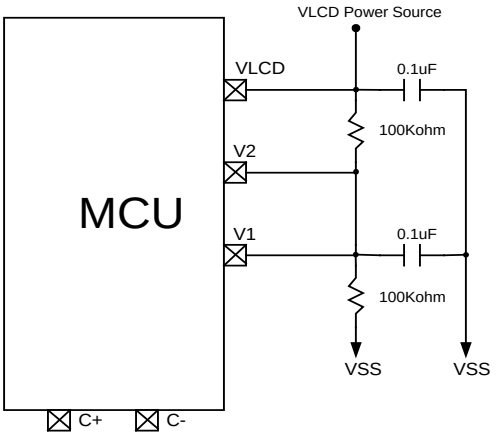
10.4 R型LCD

R 型模式下，LCD 的电源 VLCD 来自内部 VDD，V1、V2 的偏压由外部偏压电阻控制。偏压电阻决定 LCD V1、V2 的偏压和 LCD 的驱动电流。但过多的电流使 LCD 面板带有鬼影。一般情况下，建议外部偏压电阻的大小为 100K。在 R 型 LCD 模式下，VLCD 寄存器的 VLCDCP 位必须置 0。

LCD 偏压	VLCD	V2	V1
1/3 偏压（Bias=0）	VDD	2/3*VDD	1/3*VDD
1/2 偏压（Bias=1）	VDD	1/2*VDD	1/2*VDD



1/3 bias, 1/4 duty, R-type LCD Circuit.



1/2 bias, 1/4 duty, R-type LCD Circuit.

- R 型模式下，C+/C-不需要连接任何设备。
- 1/2 偏压下，V2 = V1。
- **必须在 VLCD/V1/V2 与 VSS 之间连接一个 0.1uF 的电容，以使电源稳定。**

✷ **注：VLCD 的电压源来自外部电压源，由 LCD 面板的电压电平决定，VLCD 的电压电平不能高于单片机的 Vdd。**

➢ **例：设置 R 型 LCD 模式。**

```
; 设置 R 型 LCD。
        MOV     A, #000mmmn0B   ;“n”选择偏压。
        B0MOV   LCDM            ;“mmm”控制 P2/P3 LCD 共用引脚。

; 使能 LCD 驱动。
        B0BSET  FLCDENB         ; 使能 LCD 驱动。

; LCD 图像处理。
        …
        …
```

10.5 LCD RAM分配

LCD dots 由 LCD RAM bank15 控制，使用间接寻址（bank0）或者直接寻址（bank15）访问 LCD RAM。由 LCD SEG 决定 LCD RAM 的分配放置。一个 SEG 地址包括 4 个 COM 数据。COM0 ~ COM3 是一个 LCD RAM 的低字节数据（bit0 ~ bit3）；LCD RAM 的高字节数据忽略不计。LCD RAM 的分配如下表所示：

RAM bank 15 地址 vs. Common/Segment 位置

RAM	Bit	0	1	2	3	4	5	6	7
地址	LCD	COM0	COM1	COM2	COM3	-	-	-	-
00h	SEG0	00h.0	00h.1	00h.2	00h.3	-	-	-	-
01h	SEG1	01h.0	01h.1	01h.2	01h.3	-	-	-	-
02h	SEG2	02h.0	02h.1	02h.2	02h.3	-	-	-	-
03h	SEG3	03h.0	03h.1	03h.2	03h.3	-	-	-	-
.	.	.	.	.	.	-	-	-	-
.	.	.	.	.	.	-	-	-	-
.	.	.	.	.	.	-	-	-	-
0Ch	SEG12	0Ch.0	0Ch.1	0Ch.2	0Ch.3	-	-	-	-
0Dh	SEG13	0Dh.0	0Dh.1	0Dh.2	0Dh.3	-	-	-	-
0Eh	SEG14	0Eh.0	0Eh.1	0Eh.2	0Eh.3	-	-	-	-
0Fh	SEG15	0Fh.0	0Fh.1	0Fh.2	0Fh.3	-	-	-	-
10h	SEG16	10h.0	10h.1	10h.2	10h.3	-	-	-	-
.	.	.	.	.	.	-	-	-	-
.	.	.	.	.	.	-	-	-	-
.	.	.	.	.	.	-	-	-	-
1Bh	SEG27	1Bh.0	1Bh.1	1Bh.2	1Bh.3	-	-	-	-
1Ch	SEG28	1Ch.0	1Ch.1	1Ch.2	1Ch.3	-	-	-	-
1Dh	SEG29	1Dh.0	1Dh.1	1Dh.2	1Dh.3	-	-	-	-
1Eh	SEG30	1Eh.0	1Eh.1	1Eh.2	1Eh.3	-	-	-	-
1Fh	SEG31	1Fh.0	1Fh.1	1Fh.2	1Fh.3	-	-	-	-

➢ **例：通过间接寻址@YZ bank0 设置 LCD RAM。**

```
B0MOV   Y, #0FH         ; 设置@YZ 指向 LCD RAM 的地址 1500H。
CLR     Z

MOV     A, #00001010B   ; 设置 SEG0 的 COM0=0，COM1=1，COM2=0，COM3=1。
B0MOV   @YZ, A

INCMS   Z               ; 指向下一个 segment 地址。
…
…
```

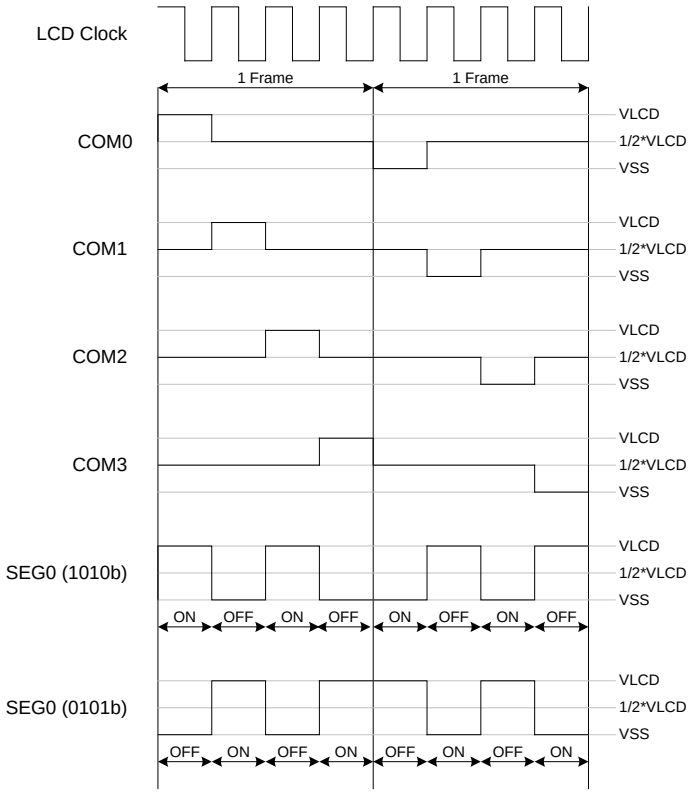
➢ **例：通过直接寻址 bank15 设置 LCD RAM。**

```
MOV     A, #01          ; 切换到 RAM bank 15。
B0MOV   RBANK, A

MOV     A, #00001010B   ; 设置 SEG0 的 COM0=0，COM1=1，COM2=0，COM3=1。
MOV     00H, A

BCLR    01H.0           ; 清 SEG 1 的 COM0=0。
BSET    01H.1           ; 设置 SEG 1 的 COM1=1。
…
…

MOV     A, #0           ; 切换到 RAM bank 0。
B0MOV   RBANK, A
```

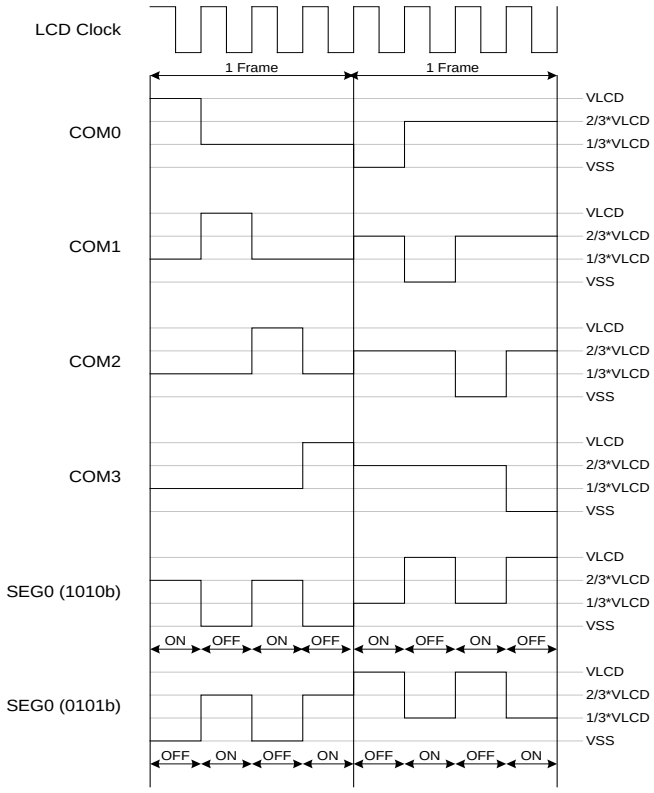
✷ **注：访问 bank0 RAM（系统寄存器和用户自定义的 RAM 0000H ~ 007FH）的数据时需使用指令“B0XXX”。**

10.6 LCD波形

1/2 bias, 1/4 duty



1/3 bias, 1/4 duty



11 指令集

指令	指令格式		描述	C	DC	Z	周期
	MOV	A,M	A ← M。	-	-	√	1
M	MOV	M,A	M ← A。	-	-	-	1
O	B0MOV	A,M	A ← M (bank 0)。	-	-	√	1
V	B0MOV	M,A	M (bank 0) ← A。	-	-	-	1
E	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I。（M 仅适用于系统寄存器 R、Y、Z、RBANK、PFLAG。）	-	-	-	1
	XCH	A,M	A ←→M。	-	-	-	1+N
	B0XCH	A,M	A ←→M (bank 0)。	-	-	-	1+N
	MOVC		R, A ← ROM [Y,Z]。	-	-	-	2
	ADC	A,M	A ← A + M + C，如果产生进位则 C=1，否则 C=0。	√	√	√	1
A	ADC	M,A	M ← A + M + C，如果产生进位则 C=1，否则 C=0。	√	√	√	1+N
R	ADD	A,M	A ← A + M，如果产生进位则 C=1，否则 C=0。	√	√	√	1
I	ADD	M,A	M ← A + M，如果产生进位则 C=1，否则 C=0。	√	√	√	1+N
T	B0ADD	M,A	M (bank 0) ← M (bank 0) + A，如果产生进位则 C=1，否则 C=0。	√	√	√	1+N
H	ADD	A,I	A ← A + I，如果产生进位则 C=1，否则 C=0。	√	√	√	1
M	SBC	A,M	A ← A - M - /C，如果产生借位则 C=0，否则 C=1。	√	√	√	1
E	SBC	M,A	M ← A - M - /C，如果产生借位则 C=0，否则 C=1。	√	√	√	1+N
T	SUB	A,M	A ← A - M，如果产生借位则 C=0，否则 C=1。	√	√	√	1
I	SUB	M,A	M ← A - M，如果产生借位则 C=0，否则 C=1。	√	√	√	1+N
C	SUB	A,I	A ← A - I，如果产生借位则 C=0，否则 C=1。	√	√	√	1
	DAA		将数据格式又十六进制换成十进制	√	√	√	1
	AND	A,M	A ← A 与 M。	-	-	√	1
L	AND	M,A	M ← A 与 M。	-	-	√	1+N
O	AND	A,I	A ← A 与 I。	-	-	√	1
G	OR	A,M	A ← A 或 M。	-	-	√	1
I	OR	M,A	M ← A 或 M。	-	-	√	1+N
C	OR	A,I	A ← A 或 I。	-	-	√	1
	XOR	A,M	A ← A 异或 M。	-	-	√	1
	XOR	M,A	M ← A 异或 M。	-	-	√	1+N
	XOR	A,I	A ← A 异或 I。	-	-	√	1
	SWAP	M	A (b3~b0, b7~b4) ←M(b7~b4, b3~b0)。	-	-	-	1
P	SWAPM	M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)。	-	-	-	1+N
R	RRC	M	A ← M 带进位右移。	√	-	-	1
O	RRCM	M	M ← M 带进位右移。	√	-	-	1+N
C	RLC	M	A ← M 带进位左移。	√	-	-	1
E	RLCM	M	M ← M 带进位左移。	√	-	-	1+N
S	CLR	M	M ← 0。	-	-	-	1
S	BCLR	M.b	M.b ← 0。	-	-	-	1+N
	BSET	M.b	M.b ← 1	-	-	-	1+N
	B0BCLR	M.b	M(bank 0).b ← 0。	-	-	-	1+N
	B0BSET	M.b	M(bank 0).b ← 1。	-	-	-	1+N
	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
B	CMPRS	A,M	比较，如果相等则跳过下一条指令 C 与 ZF 标志位可能受影响。	√	-	√	1 + S
R	INCS	M	A ← M + 1，如果 A = 0，则跳过下一条指令。	-	-	-	1+ S
A	INCMS	M	M ← M + 1，如果 M = 0，则跳过下一条指令。	-	-	-	1+N+S
N	DECS	M	A ← M - 1，如果 A = 0，则跳过下一条指令。	-	-	-	1+ S
C	DECMS	M	M ← M - 1，如果 M = 0，则跳过下一条指令。	-	-	-	1+N+S
H	BTS0	M.b	如果 M.b = 0，则跳过下一条指令。	-	-	-	1 + S
	BTS1	M.b	如果 M.b = 1，则跳过下一条指令。	-	-	-	1 + S
	B0BTS0	M.b	如果 M(bank 0).b = 0，则跳过下一条指令。	-	-	-	1 + S
	B0BTS1	M.b	如果 M(bank 0).b = 1，则跳过下一条指令。	-	-	-	1 + S
	JMP	d	跳转指令，PC15/14 ← RomPages1/0，PC13~PC0 ← d。	-	-	-	2
	CALL	d	子程序调用指令，Stack ← PC15~PC0，PC15/14 ← RomPages1/0，PC13~PC0 ← d。	-	-	-	2
M	RET		子程序跳出指令，PC ← Stack。	-	-	-	2
I	RETI		中断处理程序跳出指令，PC ← Stack，使能全局中断控制位。	-	-	-	2
S	PUSH		进栈指令，保存 ACC 和工作寄存器。	-	-	-	1
C	POP		出栈指令，恢复 ACC 和工作寄存器。	√	√	√	1
	NOP		空指令，无特别意义。	-	-	-	1

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

12 电气特性

12.1 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)
SN8P2318F, SN8P2317F..... -20°C ~ + 85°C
SN8P2318FD, SN8P2317FD.....-40°C~ +85°C
Storage ambient temperature (Tstor) ..... –40°C ~ + 125°C

12.2 电气特性

- **DC CHARACTERISTIC**

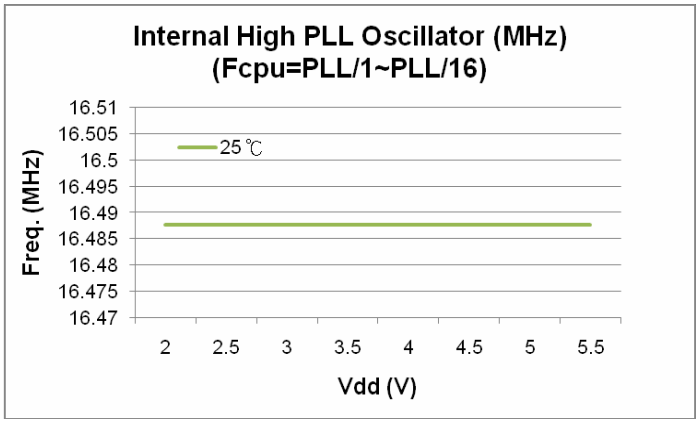
(All of voltages refer to Vss, Vdd = 5.0V, Fosc = 4MHz,Fcpu=1MHz,ambient temperature is 25°C unless otherwise note.)

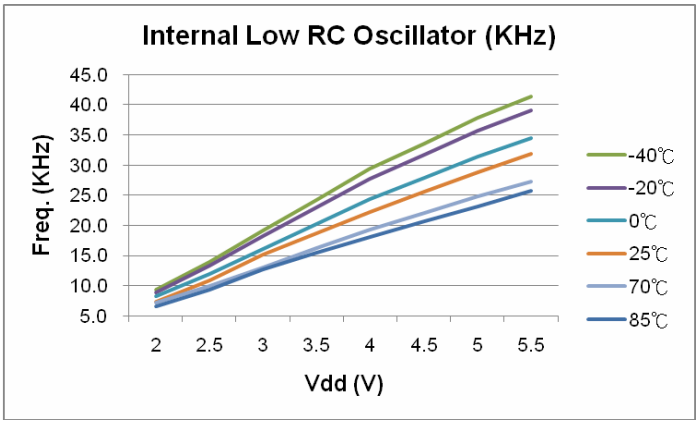
PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode. Fcpu = 1MHz		2.2	-	5.5	V
		Normal mode. Fcpu = 4MHz		2.4	-	5.5	
RAM Data Retention voltage	Vdr			1.5	-	-	V
*Vdd rise rate	Vpor	Vdd rise rate to ensure internal power-on reset		0.05	-	-	V/ms
Input Low Voltage	ViL1	All input ports		Vss	-	0.3Vdd	V
	ViL2	Reset pin		Vss	-	0.2Vdd	V
Input High Voltage	ViH1	All input ports		0.7Vdd	-	Vdd	V
	ViH2	Reset pin		0.8Vdd	-	Vdd	V
Reset pin leakage current	Ilekg	Vin = Vdd		-	-	2	uA
I/O port input leakage current	Ilekg	Pull-up resistor disable, 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 output source current	IoH	Vop = Vdd – 0.5V		8	-	-	mA
sink current	IoL	Vop = Vss + 0.5V		8	-	-	
*INTn trigger pulse width	Tint0	INT0 interrupt request pulse width		2/fcpu	-	-	cycle
Supply Current	Idd1	Run Mode (LCD disable)	Vdd= 3V, PLL/4 = 4MHz	-	2.5	-	mA
			Vdd= 5V, PLL/4 = 4MHz	-	4.5	-	mA
			Vdd= 3V, PLL/16 = 1MHz	-	1.5	-	mA
			Vdd= 5V, PLL/16 = 1MHz	-	3	-	mA
			Vdd= 3V, 16M X'tal/4 = 4MHz	-	2.5	-	mA
			Vdd= 5V, 16M X'tal/4 = 4MHz	-	5	-	mA
			Vdd= 3V, 4M X'tal/4 = 1MHz	-	1	-	mA
			Vdd= 5V, 4M X'tal/4 = 1MHz	-	2.5	-	mA
	Idd2	Slow Mode (LCD disable)	Vdd= 3V, Ext. 32K/4	-	5	-	uA
			Vdd= 5V, Ext. 32K/4	-	15	-	uA
	Idd3	Sleep Mode (LCD disable)	Vdd= 5V/3V	-	-	2	uA
	Idd4	Green Mode (LCD disable)	Vdd= 3V, PLL	-	0.6	-	mA
			Vdd= 5V, PLL	-	0.9	-	mA
			Vdd= 3V, 16M X'tal	-	0.6	-	mA
			Vdd= 5V, 16M X'tal	-	1.6	-	mA
			Vdd= 3V, 4M X'tal	-	0.2	-	mA
			Vdd= 5V, 4M X'tal	-	0.5	-	mA
			Vdd= 3V, Ext. 32KHz X'tal	-	2.5	-	uA
			Vdd= 5V, Ext. 32KHz X'tal	-	9	-	uA
	Idd5	Green Mode (LCD enable, no panel).	Vdd= 3V, Ext. 32KHz X'tal	-	5.5	-	uA
			Vdd= 5V, Ext. 32KHz X'tal	-	15	-	uA
Internal PLL Oscillator	Fpll	PLL 16MHz	25°C, Vdd=2.2V~ 5.5V Fcpu=Fhosc/1	16.11	16.78	17.45	MHz
			25°C, Vdd=2.2V~ 5.5V Fcpu=Fhosc/2~Fhosc/16	16.44	16.78	17.12	MHz
LVD Voltage	Vdet0	Low voltage reset level. 25°C		1.9	2.0	2.1	V
	Vdet1	Low voltage reset/indicator level. 25°C		2.3	2.4	2.5	V
	Vdet2	Low voltage reset/indicator level. 25°C		3.5	3.6	3.7	V

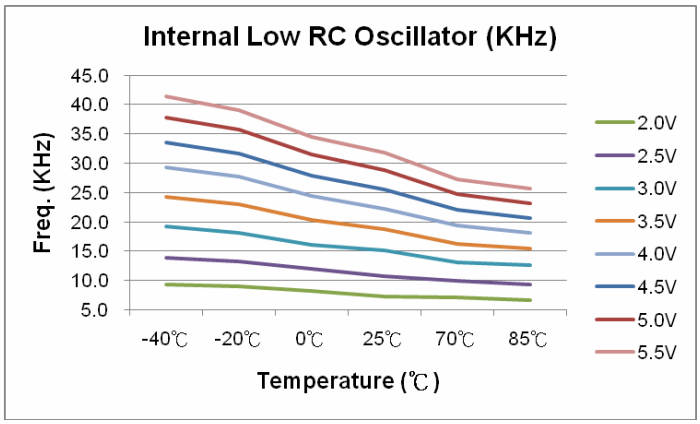
" *" These parameters are for design reference, not tested.

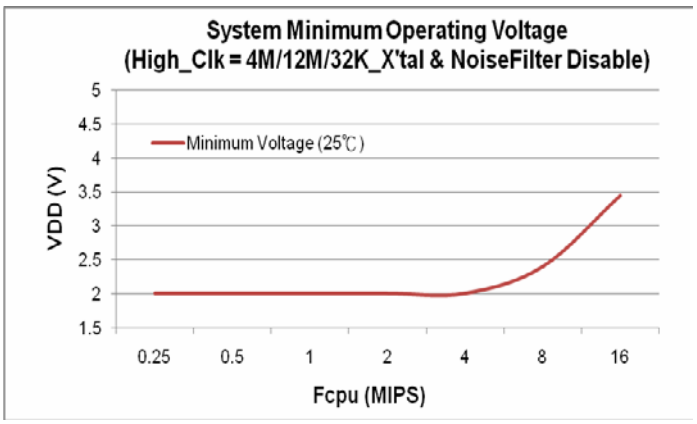
12.3 特性曲线图

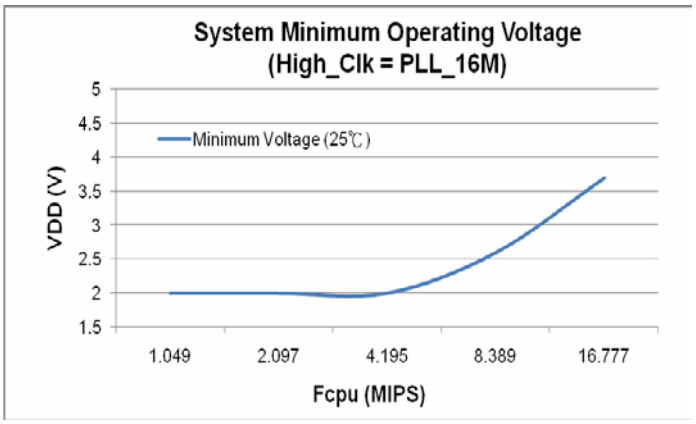
本章所列的各曲线图仅作设计参考，其中给出的部分数据可能超出了芯片指定的工作范围，为保证芯片的正常工作，请严格参照电气特性说明。

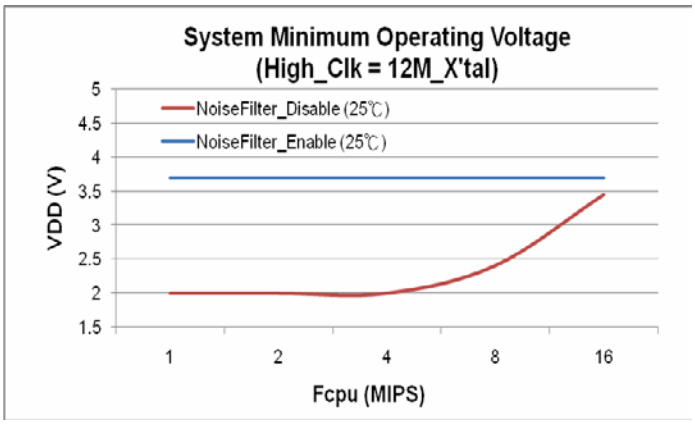












13 开发工具

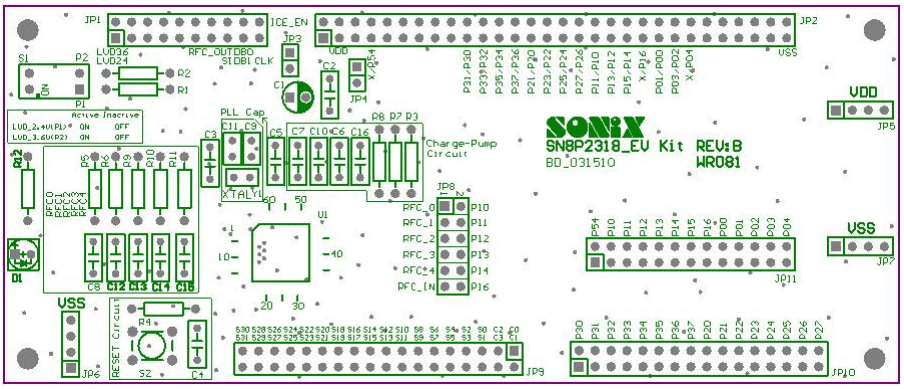
在进行 SN8P2318 的开发时，SONIX 提供 ICE（在线仿真器），IDE（集成开发环境）和 EV-Kit 开发工具。ICE 和 EV-Kit 为外部硬件装置，IDE 有一个友好的用户界面进行韧件开发与仿真。各工具的版本如下所示：

- **ICE：SN8ICE2K Plus 2。（仿真 PLL_16M 时需安装 16MHz 晶振）**
- **ICE 的最高仿真速度为：8MPIS @5V（如 16MHz 晶振，Fcpu=Fosc/2）。**
- **EV-kit：SN8P2318_EV kit Rev. B。**
- **IDE：SONiX IDE M2IDE_V129 或更晚的版本。**
- **Writer：MPIII writer。**
- **Writer 转接板：SN8P2318。**

13.1 SN8P2318 EV-kit

SN8P2318 包括多 LCD 和 RFC 功能，这些功能不能内置于 SN8ICE2K PLUS 2 中。故必须通过 SN8P2318 实际芯片来仿真这些功能，提供 EV-Kit 来仿真 LCD 和 RFC 功能。通过 SN8P2318 EV-Kit 可以仿真 LCD/RFC 功能和 LVD2.4/3.6V 切换电路。

SN8P2318 EV-Kit PCB 如下所示：

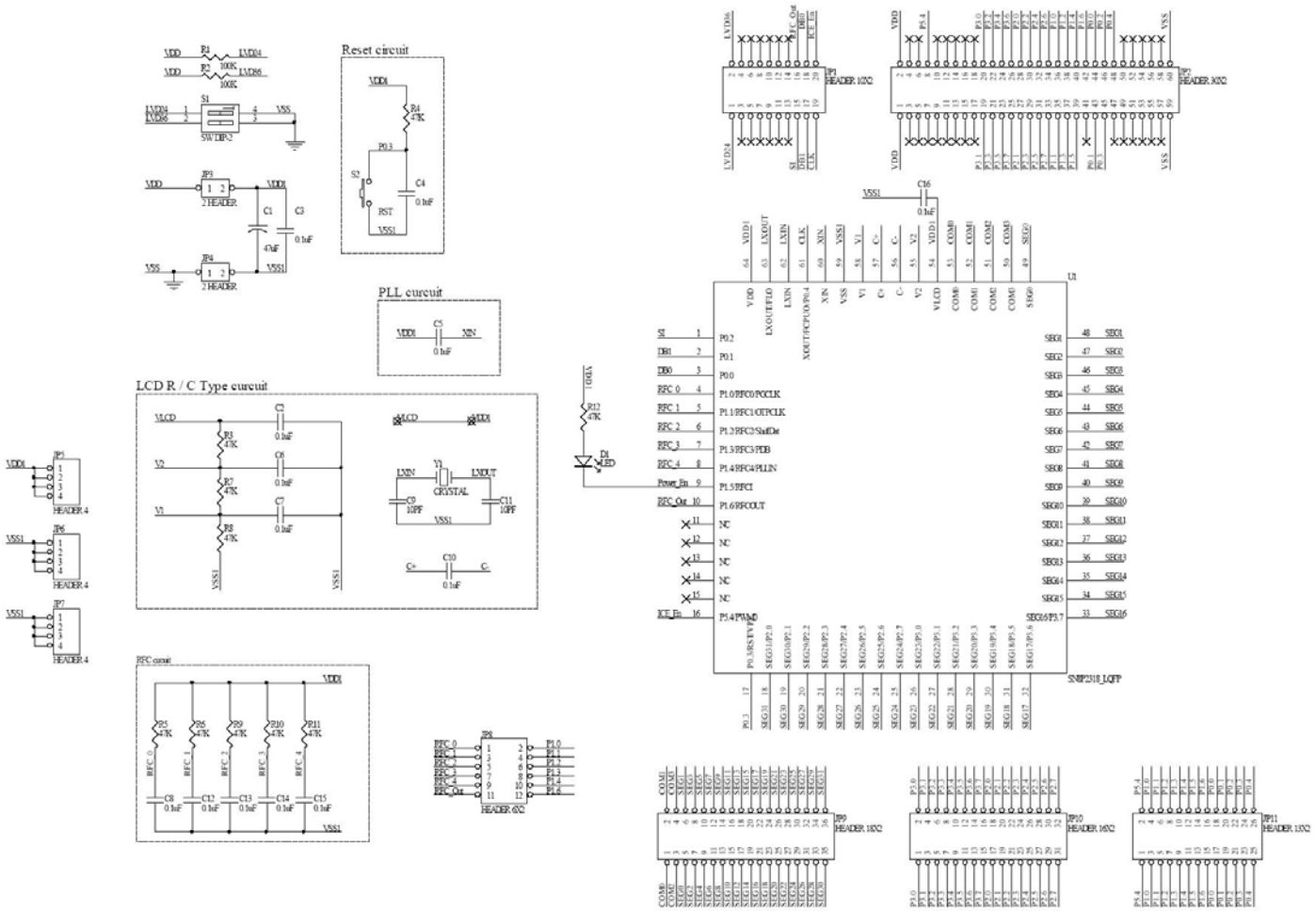


- JP2：连接到 SN8ICE2K Plus 2 CON1（包括 GPIO，EV-KIT 控制信号，及其他功能）。
- JP1：连接到 SN8ICE2K Plus 2 JP3（EV-KIT 与 ICE 的通讯总线，控制信号，及其他功能）。
- S1：LVD24V/LVD36V 控制开关。仿真 LVD2.4V 标志/复位功能和 LVD3.6V 标志/复位功能。

开关编号	ON	OFF
LVD24 (1)	LVD 2.4V 有效	LVD 2.4V 无效
LVD36 (2)	LVD 3.6V 有效	LVD 3.6V 无效

- U1：SN8P2318 EV-chip。
- S2：SN8P2318 EV-chip 复位按键。如果 EV-KIT 激活失败，按下 S2 复位 EV-KIT 实际芯片（U1）。
- JP8：保持 JP8 为打开状态。
- JP9：LCD 接口。
- JP10, JP11：GPIO 接口。
- R5, C8：RFC0。
- R6, C12：RFC1。
- R9, C13：RFC2。
- R10, C14：RFC3。
- R11, C15：RFC4。
- C7, C10, C6, C16：LCD 串联电容（0.1uF）。
- R8, R7, R3：R 型 LCD 电阻。
- C11, C9, Y1：SN8P2318 EV-chip 32KHz 晶振电路。

SN8P2318 EV-kit 原理图：

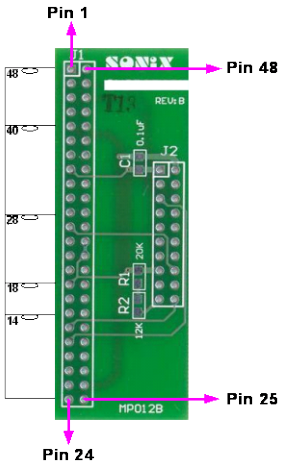


13.2 ICE和EV-KIT应用注意事项

- 连接 SN8P2318 EV-Kit 到 SN8ICE2K Plus2 之前，必须将 SN8ICE2K Plus2 上的电源开关关闭。
- 将 EV-KIT 上的 JP2/JP1 连接到 ICE 上的 CON1/JP3。
- 打开 SN8ICE2K Plus 2 的电源开始仿真。
- 如果电源指示灯（LED D1）不亮，则表示 EV-Kit 出现的问题，请联系 SONIX 的代理商。
- 在 ICE 上仿真 PLL_16M 功能时，必须连接 16MHz 的外部晶振。SN8ICE2K Plus 2 不支持超过 8-mips 指令周期，但实际芯片可以。
- JP11 的 P10~P14 和 P16 只支持 GPIO 功能，不支持 RFC 功能。
- JP10 只支持 P2/P3 的 GPIO 功能，不支持 LCD 功能。
- JP9 只支持 LCD 功能，插上 LCD 面板。

14 OTP烧录引脚

14.1 烧录器转接板引脚配置



JP3（连接 48-pin 接口）

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

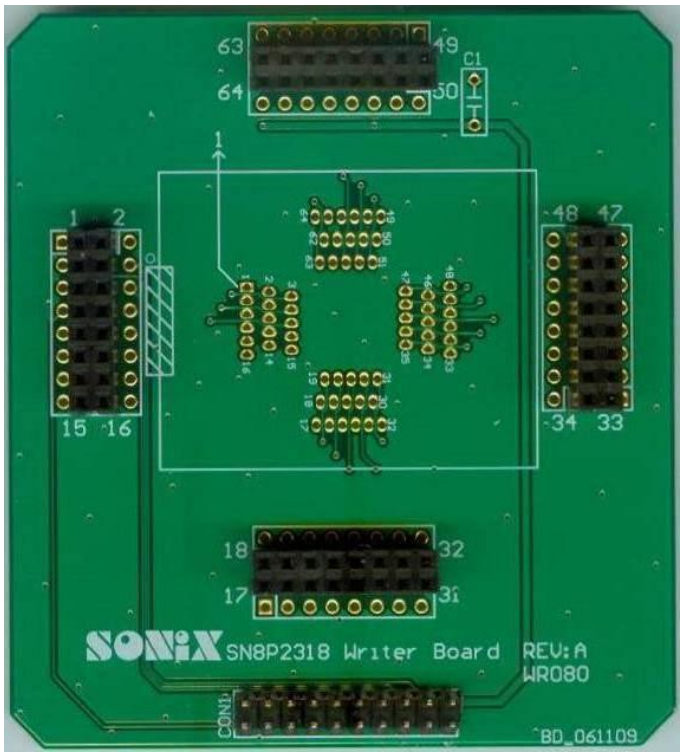
Writer JP1/JP2

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 连接烧录器转接板
JP2 连接 Dice 或大于 48pin 封装的芯片

14.2 SN8P2317/SN8P2318 OTP烧录工具

- WR080 用于 SN8P2318 LQFP64 OTP 的烧录。
- MP125 用于 SN8P2317 LQFP48 OTP 的烧录。



14.3 烧录引脚配置

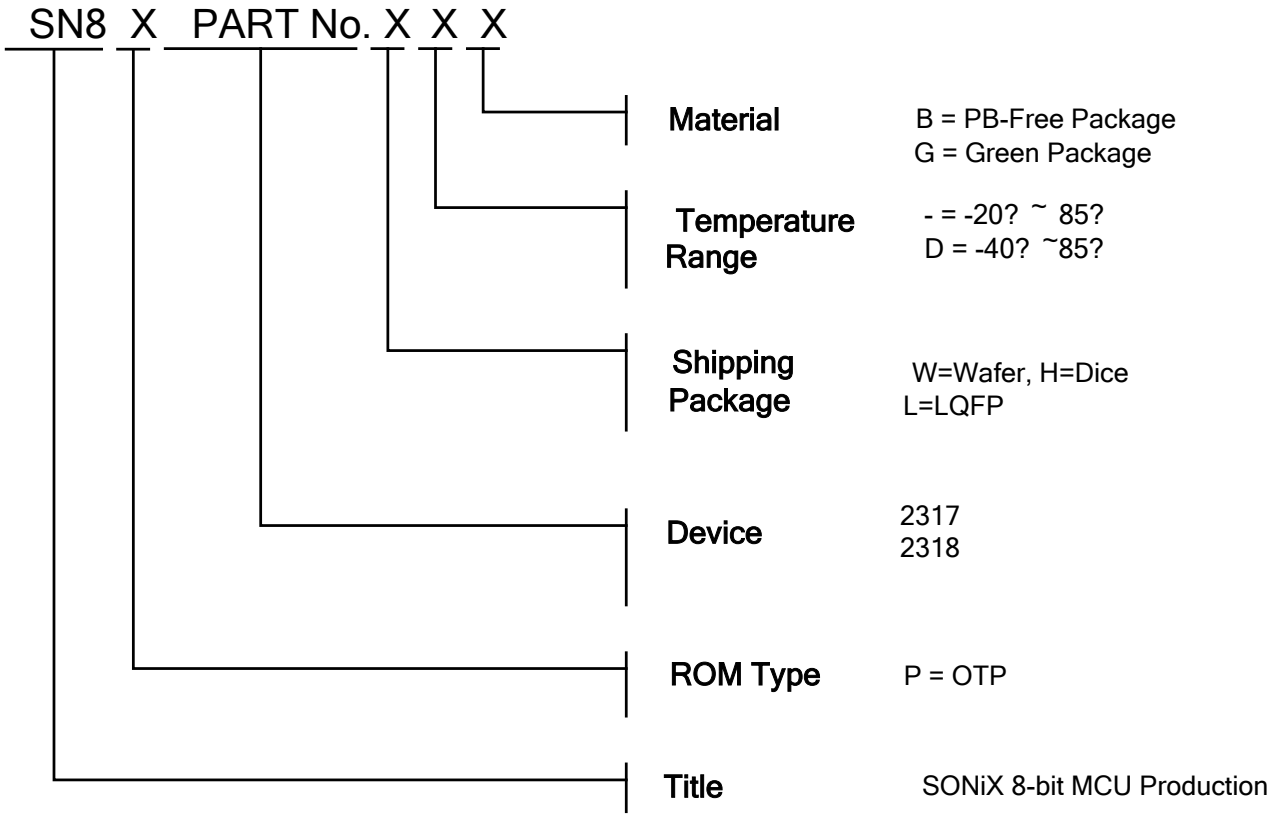
SN8P2318 的烧录引脚信息					
单片机名称		SN8P2318F(LQFP)		SN8P2317F (LQFP48)	
烧录器接口		IC 和 JP1 20-pin 引脚配置		IC 和 JP3 48-pin 锁紧引脚分配	
JP1/JP2 引脚编号	JP1/JP2 引脚名称	IC 引脚编号	IC 引脚名称	IC 引脚编号	IC 引脚名称
1	VDD	64/54	VDD/VLCD	1/39	VDD/VLCD
2	GND	59	VSS	44	VSS
3	CLK	4	P1.0	5	P1.0
4	CE	-	-	-	-
5	PGM	5	P1.1	6	P1.1
6	OE	6	P1.2	7	P1.2
7	D1	-	-	-	-
8	D0	-	-	-	-
9	D3	-	-	-	-
10	D2	-	-	-	-
11	D5	-	-	-	-
12	D4	-	-	-	-
13	D7	-	-	-	-
14	D6	-	-	-	-
15	VDD	-	-	-	-
16	VPP	17	RST	13	RST
17	HLS	-	-	-	-
18	RST	-	-	-	-
19	-	-	-	-	-
20	ALSB/PDB	7	P1.3	8	P1.3

15 单片机正印命名规则

15.1 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。

15.2 单片机型号说明



15.3 命名举例

- **Wafer, Dice：**

单片机名称	ROM 类型	设备（Device）	封装形式	温度范围	封装材料
S8P2318W	OTP	2318	Wafer	-20℃~85℃	-
SN8P2318H	OTP	2318	Dice	-20℃~85℃	-

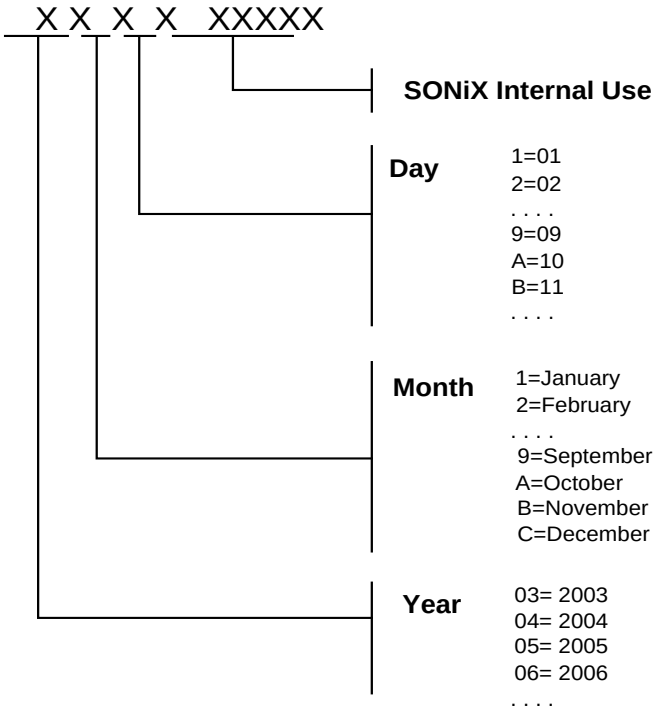
- **绿色封装：**

单片机名称	ROM 类型	设备（Device）	封装形式	温度范围	封装材料
SN8P2317FG	OTP	2318	LQFP	-20℃~85℃	绿色封装
SN8P2318FG	OTP	2318	LQFP	-20℃~85℃	绿色封装
SN8P2317FDG	OTP	2318	LQFP	-40℃~85℃	绿色封装
SN8P2318FDG	OTP	2318	LQFP	-40℃~85℃	绿色封装

- **无铅封装：**

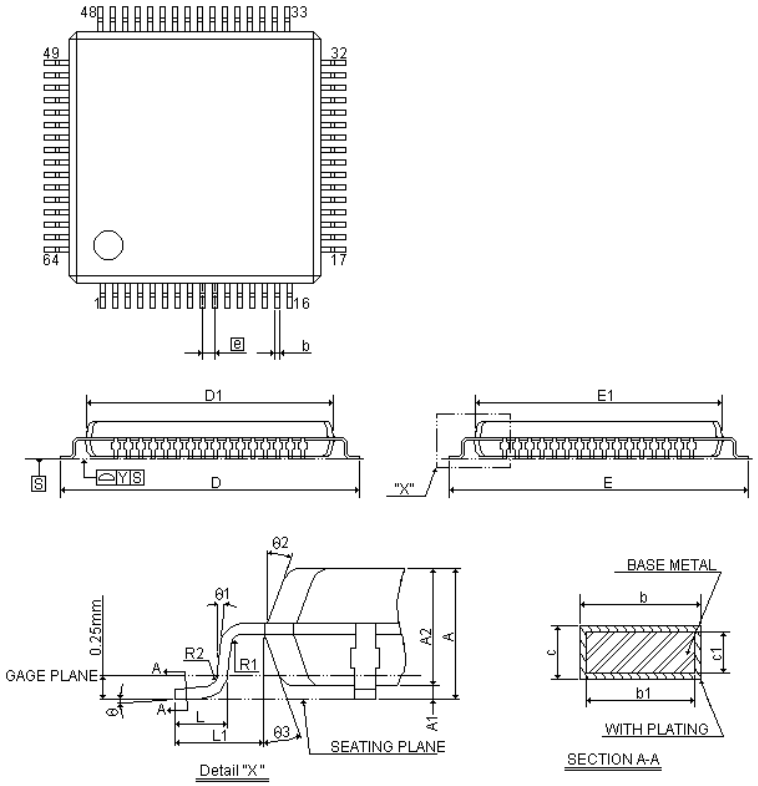
单片机名称	ROM 类型	设备（Device）	封装形式	温度范围	封装材料
SN8P2317FB	OTP	2318	LQFP	-20℃~85℃	绿色封装
SN8P2318FB	OTP	2318	LQFP	-20℃~85℃	绿色封装
SN8P2317FDB	OTP	2318	LQFP	-40℃~85℃	绿色封装
SN8P2318FDB	OTP	2318	LQFP	-40℃~85℃	绿色封装

15.4 日期码规则


X X X X XXXXX
SONiX Internal Use
Day
1=01
2=02
. . . .
9=09
A=10
B=11
. . . .
Month
1=January
2=February
. . . .
9=September
A=October
B=November
C=December
Year
03= 2003
04= 2004
05= 2005
06= 2006
. . . .

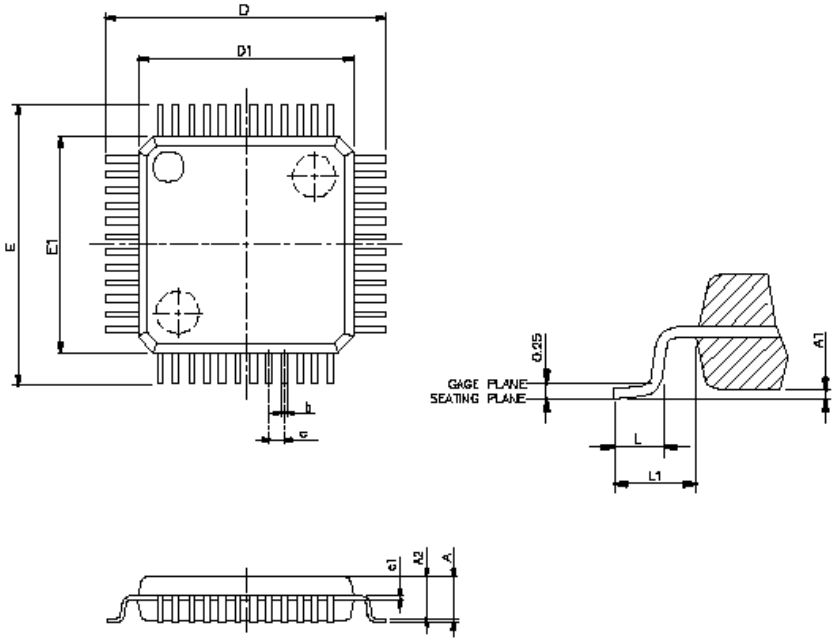
16 封装信息

16.1 LQFP 64 PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.063	-	-	1.600
A1	0.002	0.055	0.006	0.050	1.400	0.150
A2	0.054	0.009	0.057	1.360	0.220	1.450
b	0.007	0.009	0.011	0.170	0.220	0.270
b1	0.007	-	0.009	0.170	-	0.230
C	0.004	-	0.008	0.090	-	0.200
C1	0.004	-	0.006	0.090	-	0.160
D	0.472			12.000		
D1	0.394			10.000		
E	0.472			12.000		
E1	0.394			10.000		
[e]	0.020			0.500		
L	0.018	0.024	0.030	0.450	0.600	0.750
L1	0.039	-	-	1.000	-	-
R1	0.003	-	-	0.080	-	-
R2	0.003	-	0.008	0.080	-	0.200
Y	-	-	0.030	-	-	0.750
θ°	0°	3.5°	7°	0°	3.5°	7°
θ1°	0°	-	-	0°	-	-
θ2°	11°	12°	13°	11°	12°	13°
θ3°	11°	12°	13°	11°	12°	13°

16.2 LQFP 48 PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	-	-	0.06	-	-	1.6
A1	0.00	-	0.01	0.05	-	0.15
A2	0.05	-	0.06	1.35	-	1.45
C1	0.00	-	0.01	0.09	-	0.16
D	0.35 BSC			9.00 BSC		
D1	0.27 BSC			7.00 BSC		
E	0.35 BSC			9.00 BSC		
E1	0.27 BSC			7.00 BSC		
E	0.02 BSC			0.5 BSC		
B	0.01	-	0.01	0.17	-	0.27
L	0.02	-	0.03	0.45	-	0.75
L1	0.04REF			1REF		

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：香港新界沙田火炭禾盛街 11 号，中建电讯大厦 26 楼 03 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