SN8P1937

用户参考手册

Version 1.0

SONiX 8 位单片机

修改记录

版本	日期	修改说明
VER 1.0	2010.06	第一版。

目　　录

1 产品简介

1.1 产品选型表

名称	ROM	RAM	堆栈	LCD	定时器			I/O	ADC	PWM Buzzer	RTC	唤醒功能引脚数目	封装显示
					T0	TC0	TC1						
SN8P1907	2K*16	128*8	8	4*12	V	-	-	11	16-bit	-	-	5	SSOP48
SN8P1917	2K*16	128*8	8	4*12	V	-	-	13	16-bit	-	-	5	SSOP48
SN8P1927	2K*16	128*8	8	4*12	V	-	-	13	16-bit	-	-	5	SSOP48/LQFP48
SN8P1937	2K*16	128*8	8	4*12	V	V	-	13	16-bit	1	V	5	LQFP64

SN8P19x7 系列单片机性能比较表

1.2 产品资源配置改良表

项目	SN8P1917	SN8P1927	SN8P1937
PGIA 增益设置	1x, 12.5x, 50x, 100x, 200x	1x, 12.5x, 50x, 100x, 200x	1x, 16x, 32x, 64x, 128x (ADC Gain option ~ 4x)
PGIA 温度漂移	良好	良好	良好
AVE+低压	3.0V 或 1.5V	2.0V 或 1.5V	1.5V
CPR ON 负载后的 AVE+电流	双倍电流功耗	不是双倍电流功耗	不是双倍电流功耗
ADC 参考电压 V(R+, R-)	0.8V 或 0.4V	0.4V	0.6V 或 0.3V
电池检测方法	通过比较器或者 ADC	通过比较器或者 ADC	通过比较器或者 ADC
温度传感器	内置	内置	内置
ACM 电压	外灌电流时，电压不会变化	外灌电流时，电压不会变化	外灌电流时，电压不会变化
Charge pump 时钟频率（CPCKS）	4 位	4 位	无
Chopper 时钟频率（AMPCKS）	3 位	3 位	3 位
AVDDR/AVE+能否在低速模式下工作	能	能	能
高速模式下的功耗	少	少	非常少
低速模式下的功耗	少	少	非常少
LCD 偏压	1/3 或 1/2	1/3 或 1/2	1/3 或 1/2
LCD 类型	只有 R 型	只有 R 型	R 型和 C 型
VLCD 电压	固定	固定	可调节
内部 16M RC 振荡器	有	有	有
P2 [1:0] I/O	Fosc=IHRC 时有效	Fosc=IHRC 时有效	Fosc=IHRC 时有效
OTP 烧录方式	串行	串行	串行
ROM 在线烧录	无	有	有

SN8P1937 资源配置改良表

1.3 功能特性

◆ **存储器配置**
OTP ROM：2K * 16 位。
RAM:128 * 8 位（bank 0）。
8 层堆栈缓存器。
LCD RAM：4*12 位。

◆ **I/O 引脚配置**
单向输入口：P0。
双向输入输出口：P1，P2，P5。
具有唤醒功能的端口：P0，P1。
内置上拉电阻的端口：P0，P1，P2，P5。
外部中断端口：P0。

◆ **强大的指令系统**
一条指令需要 4 个时钟周期。
所有的指令均为一个字长。
绝大部分指令只需要一个周期。
指令的最长周期为 2 个指令周期。
JMP 指令可在整个 ROM 区执行。
查表指令（MOVC）可寻址整个 ROM 区。

◆ **2 个定时/计数器**
T0：基本定时器，具有 0.5S RTC 功能。
TC0：自动重装定时器/计数器/Buzzer/PWM 输出。

◆ **可编程增益放大器**
PGIA 增益可选选项：1x/16x/32x/64x/128x。

◆ **16 位 Delta-Sigma ADC，具有 14 位 noise free**
ADC 增益选项：1x，2x，4x。
2 个 ADC 通道配置。
1 个全差分通道。
2 个单端输入通道。

◆ **3 个中断源**
2 个内部中断：T0，TC0。
1 个外部中断：INT0。

◆ **单电源输入：2.4V ~5.5V**

◆ **内置看门狗定时器**

◆ **内置 2.4V 电压输出和 10mA 驱动电流的稳压器（AVDDR）**

◆ **内置 1.5V 输出电压的稳压器（Regulator）**
AVE+负载电流功耗不是双倍功耗

◆ **内置参考电压为 1.2V 的 Band Gap，用来监控电池电压**

◆ **内置电压比较器**

◆ **内置 ADC 参考电压 V(R+,R-)= 0.3V/0.6V**

◆ **1 Set Buzzer/ PWM**

◆ **内置 0.5S RTC 模式**

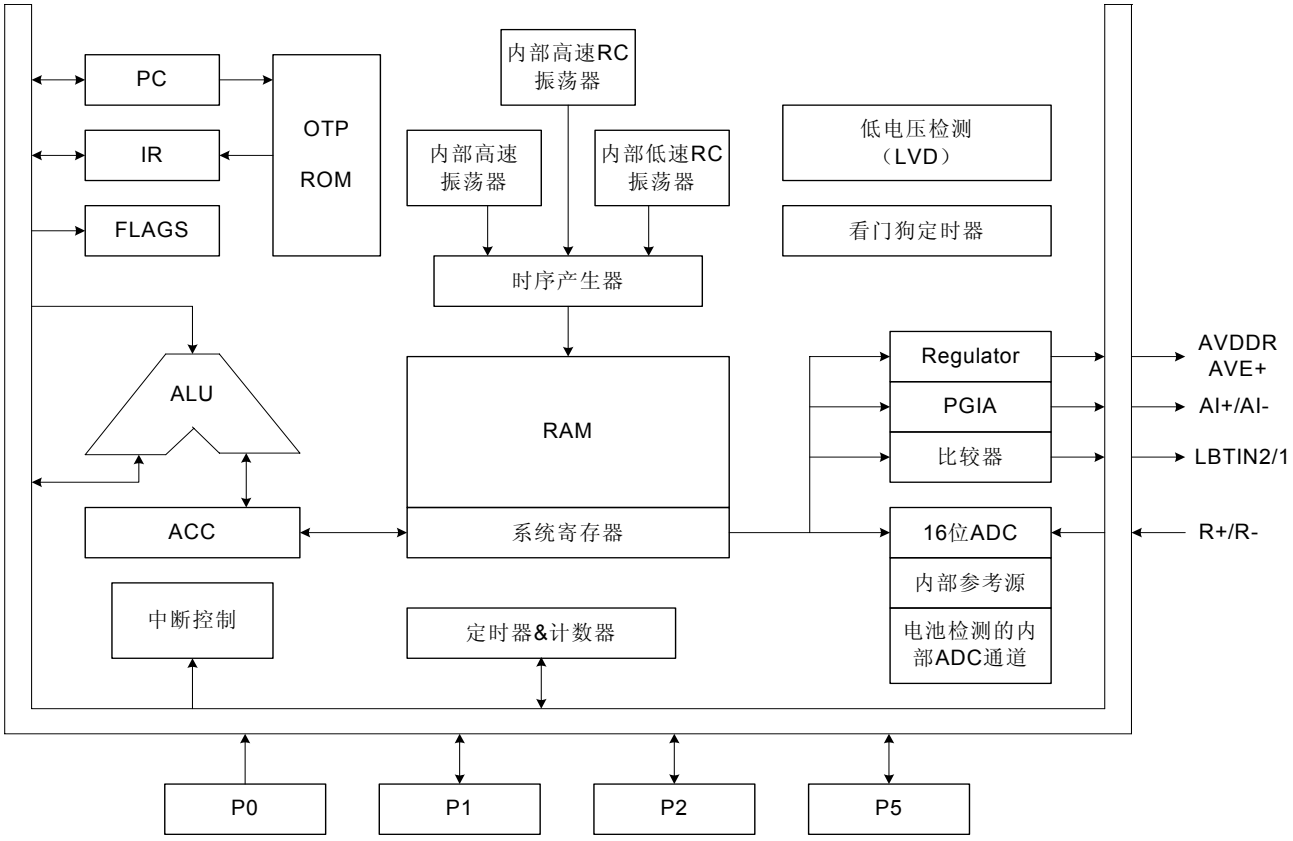
◆ **ROM 在线烧录**

◆ **LCD 驱动**
1/3 或 1/2 偏压。
4 common * 12 segment。
包括 R 型和 C 型 LCD。
C 型 LCD 电压：2.7 ~3.4V。

◆ **双时钟系统提供 4 中工作模式**
内部高速时钟：RC 模式，高达 16MHz。
外部高速时钟：晶体模式，高达 8MHz。
普通模式：高低速时钟都正常工作。
低速模式：仅低速时钟工作。
绿色模式：低速时钟正常工作，高速时钟可选择是否工作。
睡眠模式：高低速时钟都停止工作。

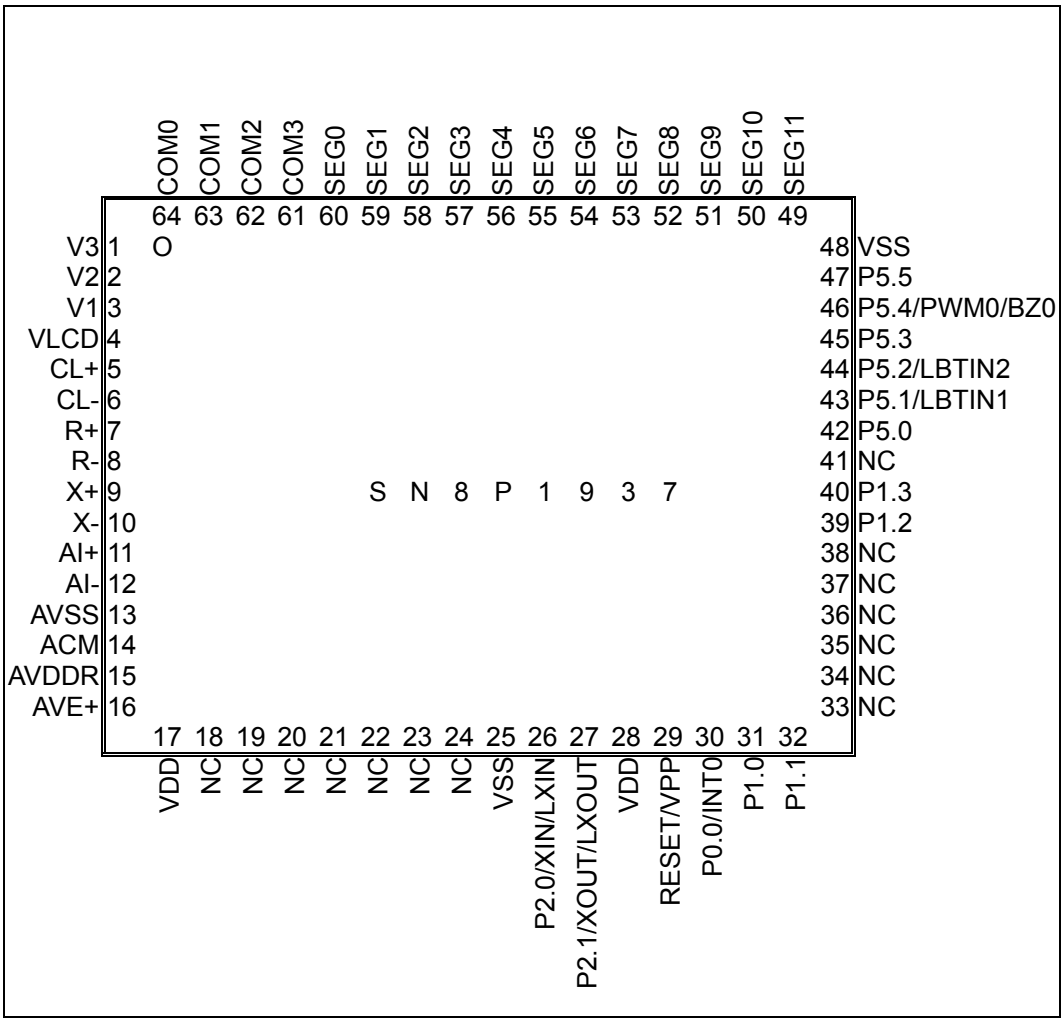
◆ **封装形式**
Dice / LQFP64

1.4 系统框图


PC
IR
FLAGS
OTP
ROM
内部高速RC
振荡器
内部高速
振荡器
内部低速RC
振荡器
低电压检测
（LVD）
看门狗定时器
时序产生器
ALU
RAM
ACC
系统寄存器
Regulator
PGIA
比较器
16位ADC
内部参考源
电池检测的内
部ADC通道
AVDDR
AVE+
AI+/AI-
LBTIN2/1
R+/R-
中断控制
定时器&计数器
P0
P1
P2
P5

1.5 引脚配置

SN8P1937 LQFP64

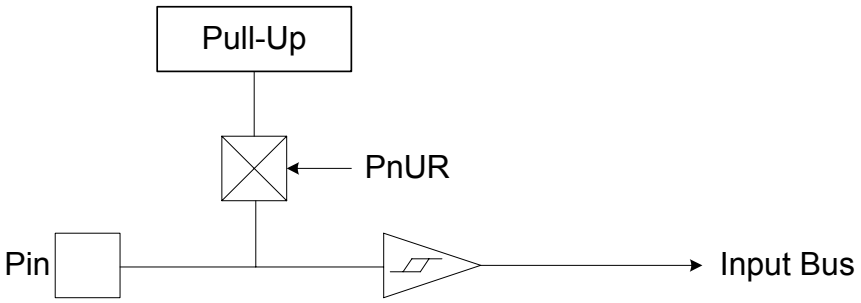


1.6 引脚说明

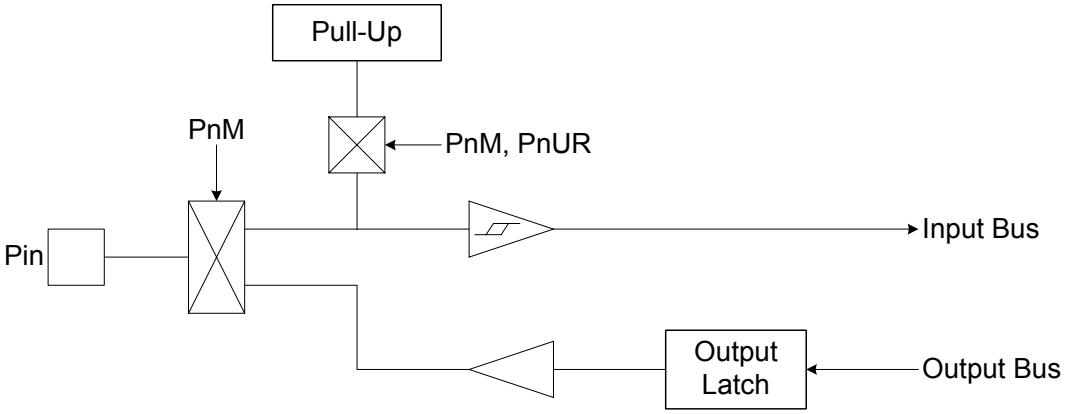
引脚名称	类型	功能说明
VDD, VSS, AVSS	P	电源输入端。
AVDDR	P	Regulator 电源输出引脚：电压为 2.4V。
AVE+	P	传感器 Regulator 输出 1.5V，最大输出电流为 10mA。
ACM	P	ACM 电压输出 0.4V。
R+	AI	ADC 参考源的正极输入端。
R-	AI	ADC 参考源的负极输入端。
X+	AI	ADC 差分的正极输入端。
X-	AI	ADC 差分的负极输入端。
AI+	AI	模拟输入通道的正极输入端。
AI-	AI	模拟输入通道的负极输入端。
VPP/ RST	P, I	OTP ROM 烧录引脚。 系统复位输入引脚，施密特触发，低电平有效，通常保持高电平。
XIN / LXIN / P20	I, O	外部高速时钟振荡器引脚（编译选项选择 4M Crystal）。 外部低速 32768Hz 振荡器引脚（编译选项选择 IHRC_RTC）。 与 GPIO 引脚 P2.0 共用。
XOUT / LXOUT/ P21	I, O	外部高速振荡器引脚（编译选项选择 4M Crystal）。 外部低速 32768Hz 振荡器引脚（编译选项选择 IHRC_RTC）。 与 GPIO 引脚 P2.1 共用。
P0.0 / INT0	I	P0.0 与 INT0 触发引脚共用，施密特触发，内置上拉电阻。
P1 [3:0]	I/O	双向输入输出引脚，具有唤醒功能，内置上拉电阻。
P2 [1:0]	I/O	双向输入输出引脚，内置上拉电阻，与 XIN/LXIN、XOUT/LXOUT 引脚共用。
P5 [5:0]	I/O	双向输入输出引脚，内置上拉电阻。P5.4 与 BZ0/PWM0 共用。
LBTIN1	I	电池低电压检测输入引脚，与 P5.1 共用。
LBTIN2	I	电池低电压检测输入引脚，与 P5.2 共用。
COM [3:0]	O	COM0~COM3 LCD 驱动 COM 端。
SEG0 ~ SEG11	O	LCD 驱动 SEG 引脚。
CL+, CL-	P	C 型 LCD charge pump 电容。
VLCD	P	LCD 驱动电源引脚（无需外接电源）。
V3	P	LCD 偏置电压。
V2	P	LCD 偏置电压
V1	P	LCD 偏置电流有效/无效控制点。

1.7 引脚电路结构图

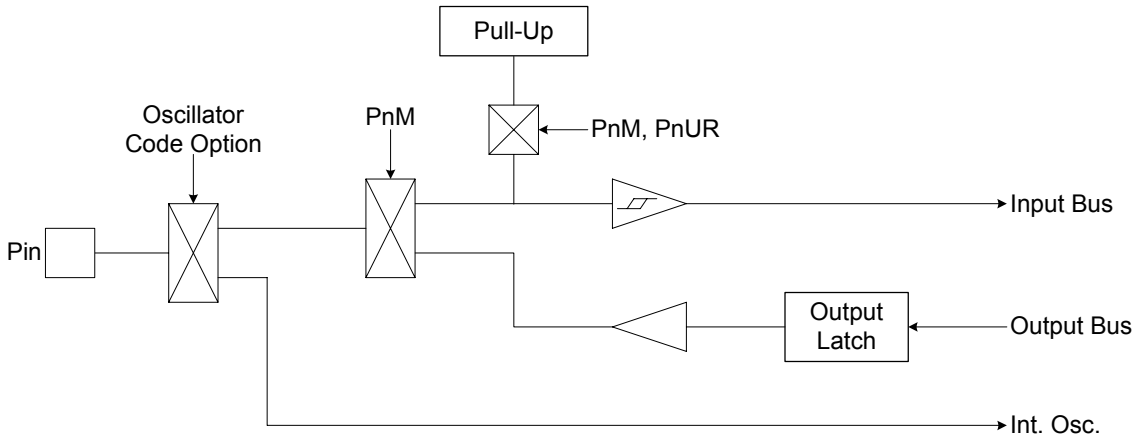
- **P0：**



- **P1/P5：**



- **P2：**

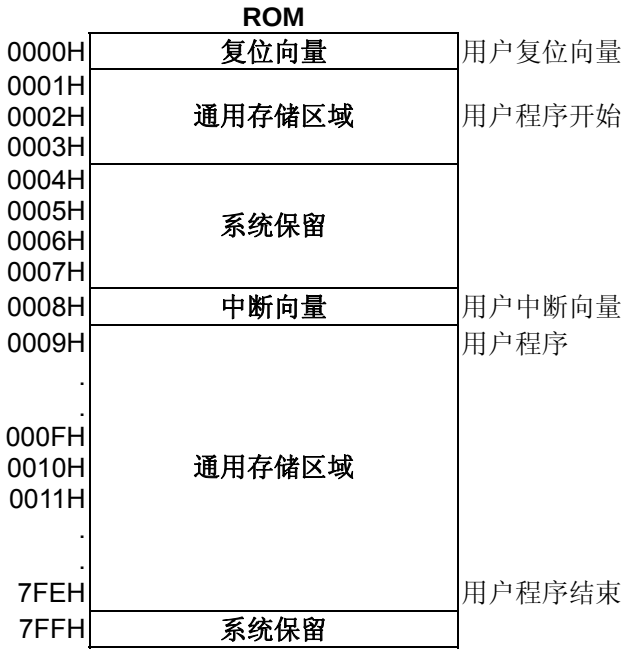


2 中央处理器（CPU）

2.1 存储器

2.1.1 程序存储器（ROM）

☞ **ROM：2K 字**



2.1.1.1 复位向量（0000H）

具有一个字长的系统复位向量（0000H）。

☞ **上电复位；**
☞ **看门狗复位；**
☞ **外部复位。**

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。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:                          ; 0010H，用户程序起始地址。
        …                       ; 用户程序。
        …

        ENDP                    ; 程序结束。
```

2.1.1.2 中断向量

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：在中断发生时，用户必须用程序保存 ACC 和 PFLAG。**

➢ **例：定义中断向量，中断服务程序紧随 ORG 8 之后。**

```
.DATA           ACCBUF      DS   1          ;
                PFLAGBUF    DS   1          ;
.CODE
                ORG         0               ; 0000H。
                JMP         START           ; 跳到用户程序。
                …
                ORG         8H              ;
                B0XCH       A, ACCBUF       ; 保存 ACC。
                B0MOV       A, PFLAG
                B0MOV       PFLAGBUF, A     ; 保存 PFLAG。
                …
                B0MOV       A, PFLAGBUF
                B0MOV       PFLAG, A        ; 恢复 PFLAG。
                B0XCH       A, ACCBUF       ; 恢复 ACC。
                RETI                        ; 中断返回。
                …
START:                                      ; 用户程序开始。
                …                           ; 用户程序。
                JMP         START           ; 用户程序结束。
                …
                ENDP                        ;
```

➢ **例：定义中断向量，中断服务程序位于用户程序的后面。**

```
.DATA           ACCBUF      DS   1
                PFLAGBUF    DS   1
.CODE
                ORG         0               ; 0000H。
                JMP         START           ; 跳到用户程序。
                …
                ORG         8H              ;
                JMP         MY_IRQ          ; 跳到中断服务程序。

                ORG         10H
START:                                      ; 用户程序开始。
                …                           ; 用户程序。
                JMP         START           ; 用户程序结束。
                …
MY_IRQ:                                     ; 中断服务程序开始。
                B0XCH       A, ACCBUF       ; 保存 ACC。
                B0MOV       A, PFLAG
                B0MOV       PFLAGBUF, A     ; 保存 PFLAG。
                …
                B0MOV       A, PFLAGBUF
                B0MOV       PFLAG, A        ; 恢复 PFLAG。
                B0XCH       A, ACCBUF       ; 恢复 ACC。
                RETI                        ; 中断返回。
                …
                ENDP                        ;
```

✷ **注：从上面的程序中容易得出 SONiX 的编程规则，有以下几点：**

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

2.1.1.3 查表

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

➢ **例：查找 ROM 地址为“TABLE1”的值。**

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

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

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

➢ **例：宏 INC_YZ。**

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

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

➢ **例：通过“INC_YZ”对上例进行优化。**

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

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

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

➢ **例：由指令 B0ADD/ADD 对 Y 和 Z 寄存器加 1。**

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

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

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

2.1.1.4 跳转表

跳转表能够实现多地址跳转功能。PCL 和 ACC 的值相加即可得到新的 PCL。由此得到的新的 PC 值指向一系列跳至指令列表。

执行“ADD PCL, A”后，如果有进位发生，结果并不会影响 PCH 寄存器。用户必须检查跳转表是否跨越了 ROM 的页边界。如果跳转表跨越了 ROM 页边界（例如从 xxFFH 到 xx00H），将跳转表移动到下一页程序存储区的顶部（xx00H）。
注：一页包含 256words。

✷ **注：在执行加法运算后，若 PCL 溢出，程序计数器 PC 并不会从 PCL 进位到 PCH。**

➢ **例：跳转表。**

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

        B0ADD   PCL, A      ; PCL = PCL + ACC，PCH 的值不会被改变。
        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。
```

在下面的例子中，跳转表从 00FDH 开始，执行 B0ADD PCL, A 后，如果 ACC = 0 或者 1，跳转表指向正确的地址，但如果 ACC 大于 1，因为 PCH 不能自动加一，程序就会出错。可以看到当 ACC = 2 时，PCL = 0，而 PCH 仍然保持为 0，则新的程序计数器 PC 将指向错误的地址 0000H，程序出错。因此，检查跳转表是否跨越边界（xxFFH 到 xx00H）非常重要。良好的编程风格是将跳转表放在 ROM 的开始边界（如 0100H）。

➢ **例：如果跳转表跨越 ROM 边界，将引起程序错误。**

```
ROM Address
…
…
…
00FDH   B0ADD   PCL, A      ; PCL = PCL + ACC，PCH 的值不会被改变。
00FEH   JMP     A0POINT     ; ACC = 0。
00FFH   JMP     A1POINT     ; ACC = 1。
0100H   JMP     A2POINT     ; ACC = 2。 ← 跳转表跨越边界
0101H   JMP     A3POINT     ; ACC = 3。
…
…
```

SONiX 提供一个宏程序以保证可靠执行跳转表功能，它将检测 ROM 边界并自动将跳转表移至适当的位置。但采用该宏程序会占用部分 ROM 空间。

➢ **例：如果跳转表跨越 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。
```

如果跳转表跨越了 ROM 的边界（0FFH~100H），宏指令“@JMP_A”会调整跳转表的位置使其从 ROM 的前端开始。

➢ **例：宏指令@JMP_A 操作。**

；编译前

```
ROM 地址
        B0MOV     A, BUF0       ; “BUF0”从 0 至 4。
        @JMP_A    5             ; 列表个数为 5。
0X00FD  JMP       A0POINT       ; ACC = 0，跳至 A0POINT。
0X00FE  JMP       A1POINT       ; ACC = 1，跳至 A1POINT。
0X00FF  JMP       A2POINT       ; ACC = 2，跳至 A2POINT。
0X0100  JMP       A3POINT       ; ACC = 3，跳至 A3POINT。
0X0101  JMP       A4POINT       ; ACC = 4，跳至 A4POINT。
```

；编译后

```
ROM 地址
        B0MOV     A, BUF0       ; “BUF0”从 0 至 4。
        @JMP_A    5             ; 列表个数为 5。
0X0100  JMP       A0POINT       ; ACC = 0, 跳至 A0POINT。
0X0101  JMP       A1POINT       ; ACC = 1, 跳至 A1POINT。
0X0102  JMP       A2POINT       ; ACC = 2, 跳至 A2POINT。
0X0103  JMP       A3POINT       ; ACC = 3, 跳至 A3POINT。
0X0104  JMP       A4POINT       ; ACC = 4, 跳至 A4POINT。
```

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

2.1.2 编译选项列表（CODE OPTION）

编译选项	内容	功能说明
High_Clk	IHRC	高速时钟采用内部 16MHz RC 振荡器，XIN/XOUT（P2.0/P2.1）作为普通 I/O 引脚。
	IHRC_RTC	高速时钟采用内部 16MHz RC 振荡器和外部 32768 振荡器。
	4M X'tal	外部高速振荡器采用标准振荡器（如 4MHz）。
Watch_Dog	Enable	开启看门狗定时器。
	Disable	关闭看门狗定时器。
Security	Enable	ROM 代码加密。
	Disable	ROM 代码不加密。
INT_32K_RC	Always_ON	强行设置内部 32K RC 作为看门狗定时器的时钟源。 看门狗定时器在省电模式和绿色模式下仍然处于运行状态。
	By_CPUM	由 CPUM 寄存器控制内部 32K（@3V）RC 时钟是否工作。
Low Power	Enable	使能低功耗功能以省电。
	Disable	禁止低功耗功能。

✷ **注：**

1.在高干扰环境下，强烈建议设置 Watch_Dog 为"Enable"，INT_32K_RC 设置为"Always_On"；
2.Fcpu 编译选项仅对高速时钟有效，低速模式下 Fcpu = Fosc/4；
3.在高干扰环境下，强烈建议禁止"Low Power"功能；
4.如果使能"Low Power"功能，会增加最低工作电压；
5.使能"Low Power"功能会减小工作电流，低速模式下除外。

2.1.3 数据存储器（RAM）

☞ **RAM：128 X 8 位**

		RAM	
BANK 0	000h	**通用存储区域**	; Bank 0 的 000H~07FH 为通用存储区域
		.	;（128 字节）
	07Fh	.	
	080h	**系统寄存器**	; Bank 0 的 080H~0FFH 为系统寄存器
		.	; （128 字节）
	0FFh	**Bank 0 结束**	
BANK 15	F00h	**LCD RAM 区域**	; Bank 15 为 LCD 数据显示区域
		.	; （12 字节）
	F0Bh	**LCD RAM 结束**	;

2.1.4 系统寄存器

2.1.4.1 系统寄存器列表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-	-	R	Z	Y	-	PFLAG	RBANK	-	LCDM1	LCDM2	-	-	-	-	-
9	AMPM	AMPCHS	AMPCKS	ADCM	ADCKS	CPM	CPCKS	DFM	ADCDL	ADCDH	LBTM	CPMTEST	-	-	-	-
A	ROMADRH	ROMADRL	ROMDAH	ROMDAL	ROMCNT	-	-	-	-	-	-	-	-	-	-	
B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	PEDGE
C	-	P1M	P2M	-	-	-	-	-	INTRQ	INTEN	OSCM	-	-	-	PCL	PCH
D	P0	P1	P2	-	P4	P5	-	-	T0M	T0C	TC0M	TC0C	-	-	-	STKP
E	-	P1UR	P2UR	-	-	-	-	@YZ	-	-	-	-	-	-	-	-
F	STK7L	STK7H	STK6L	STK6H	STK5L	STK5H	STK4L	STK4H	STK3L	STK3H	STK2L	STK2H	STK1L	STK1H	STK0L	STK0H

2.1.4.2 系统寄存器说明

Y, Z = 工作寄存器，@YZ 间接寻址寄存器，ROM 寻址寄存器
PFLAG = ROM 页和特殊标志寄存器
AMPM = PGIA 模式寄存器
AMPCKS = PGIA 时钟选择寄存器
ADCKS = ADC 时钟选择寄存器
CPCKS = Charge pump 时钟选择寄存器
ADCDL = ADC 低字节数据缓存器
PNM = Pn 输入/输出模式寄存器
PN = Pn 数据缓存器
INTEN = 中断使能寄存器
LCDM1= LCD 模式寄存器
LCDM2= LCD 模式寄存器
T0M = T0 模式寄存器
T0C = T0 计数寄存器
TC0M = TC0 模式寄存器
TC0C = TC0 计数寄存器
LBTM= 电池低电压检测寄存器

R = 工作寄存器，ROM 查表数据缓存器
AMPCHS = PGIA 通道选择寄存器
ADCM = ADC 模式寄存器
CPM = Charge pump 模式寄存器
DFM = 数字滤波模式寄存器
ADCDH = ADC 高字节数据缓存器
PNUR = Pn 上拉电阻寄存器
INTRQ = 中断请求寄存器
OSCM = 振荡器模式寄存器
PCH, PCL = 程序计数器
STK0~STK7 = 堆栈缓存器
@YZ = 间接寻址寄存器
STKP = 堆栈指针
ROMADRH/L= ISP ROM 地址
ROMDAH/L= ISP ROM 数据
ROMCNT= ISP ROM 计数器

2.1.4.3 系统寄存器的位定义

地址	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	寄存器名称
082H	RBIT7	RBIT6	RBIT5	RBIT4	RBIT3	RBIT2	RBIT1	RBIT0	R/W	R
083H	ZBIT7	ZBIT6	ZBIT5	ZBIT4	ZBIT3	ZBIT2	ZBIT1	ZBIT0	R/W	Z
084H	YBIT7	YBIT6	YBIT5	YBIT4	YBIT3	YBIT2	YBIT1	YBIT0	R/W	Y
086H	-	-	-	-	-	C	DC	Z	R/W	PFLAG
087H	-	-	-	-	RBNKS3	RBNKS2	RBNKS1	RBNKS0	R/W	RBANK
089H	LCDREF1	LCDREF0	LCDBNK	LCDTYPE	LCDENB	LCDBIAS	LCDRATE	LCDCLK	R/W	LCDM1
08AH	-	-	BGM	LCDCPK1	LCDCPK0	VCP2	VCP1	VCP0	R/W	LCDM2
090H	CHPENB	BGRENB	FDS1	FDS0	GS2	GS1	GS0	AMPENB	R/W	AMPM
091H	-	-	-	-	-	CHS2	CHS1	CHS0	R/W	AMPCHS
092H	-	-	-	-	-	AMPCKS2	AMPCKS1	AMPCKS0	R/W	AMPCKS
093H	ADG2	ADG1	ADG0	RVS	IRVS	DTENB	DTSEL	ADCENB	R/W	ADCM
094H	ADCKS7	ADCKS6	ADCKS5	ADCKS4	ADCKS3	ADCKS2	ADCKS1	ADCKS0	W	ADCKS
095H	ACMENB	AVDDRENB	AVESEL	AVENB	ACMSEL				R/W	CPM
097H	OSR2	OSR1	OSR0	ADCHPENB	-	-	-	DRDY	R/W	DFM
098H	ADCB7	ADCB6	ADCB5	ADCB4	ADCB3	ADCB2	ADCB1	ADCB0	R	ADCDL
099H	ADCB15	ADCB14	ADCB13	ADCB12	ADCB11	ADCB10	ADCB9	ADCB8	R	ADCDH
09AH	-	-	-	-	-	LBTO	P51IO	LBTENB	R/W	LBTM
0A0H	VPPCHK	-	-	-	-	ROMADR10	ROMADR9	ROMADR8	R/W	ROMADRH
0A1H	ROMADR7	ROMADR6	ROMADR5	ROMADR4	ROMADR3	ROMADR2	ROMADR1	ROMADR0	R/W	ROMADRL
0A2H	ROMDA15	ROMDA14	ROMDA13	ROMDA12	ROMDA11	ROMDA10	ROMDA9	ROMDA8	R/W	ROMDAH
0A3H	ROMDA7	ROMDA6	ROMDA5	ROMDA4	ROMDA3	ROMDA2	ROMDA1	ROMDA0	R/W	ROMDAL
0A4H	ROMCNT7	ROMCNT6	ROMCNT5	ROMCNT4	ROMCNT3	ROMCNT2	ROMCNT1	ROMCNT0	W	ROMCNT
0BFH	PEDGEN	-	-	P00G1	P00G0	-	-	-	R/W	PEDGE
0C1H	-	-	-	-	P13M	P12M	P11M	P10M	R/W	P1M
0C2H	-	-	-	-	-	-	P21M	P20M	R/W	P2M
0C5H	-	-	P55M	P54M	P53M	P52M	P51M	P50M	R/W	P5M
0C8H	-	-	T0CIRQ	T0IRQ	-	-	-	P00IRQ	R/W	INTRQ
0C9H	-	-	T0CIEN	T0IEN	-	-	-	P00IEN	R/W	INTEN
0CAH	WTCKS	WDRST	WDRATE	CPUM1	CPUM0	CLKMD	STPHX	-	R/W	OSCM
0CDH	T0CR7	T0CR6	T0CR5	T0CR4	T0CR3	T0CR2	T0CR1	T0CR0	W	T0CR
0CEH	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	R/W	PCL
0CFH	-	-	-	-	-	PC10	PC9	PC8	R/W	PCH
0D0H	-	-	-	-	-	-	-	P00	R	P0
0D1H	-	-	-	-	P13	P12	P11	P10	R/W	P1
0D2H	-	-	-	-	-	-	P21	P20	R/W	P2
0D5H	-	-	P55	P54	P53	P52	P51	P50	R/W	P5
0D8H	T0EN	T0RATE2	T0RATE1	T0RATE0	-	-	T0GN	T0TB	R/W	T0M
0D9H	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0	R/W	T0C
0DAH	TC0EN	TC0RATE2	TC0RATE1	TC0RATE0	TC0CKS	ALOAD0	TC0OUT	PWM0OUT	R/W	TC0M
0DBH	TC0C7	TC0C6	TC0C5	TC0C4	TC0C3	TC0C2	TC0C1	TC0C0	R/W	TC0C
0DFH	GIE	-	-	-	-	STKPB2	STKPB1	STKPB0	R/W	STKP
0E1H	-	-	-	-	P13R	P12R	P11R	P10R	W	P1UR
0E2H	-	-	-	-	-	-	P21R	P20R	W	P2UR
0E5H	-	-	P55R	P54R	P53R	P52R	P51R	P50R	W	P5UR
0E7H	@YZ7	@YZ6	@YZ5	@YZ4	@YZ3	@YZ2	@YZ1	@YZ0	R/W	@YZ
0F0H	S7PC7	S7PC6	S7PC5	S7PC4	S7PC3	S7PC2	S7PC1	S7PC0	R/W	STK7L
0F1H	-	-	-	-	-	S7PC10	S7PC9	S7PC8	R/W	STK7H
0F2H	S6PC7	S6PC6	S6PC5	S6PC4	S6PC3	S6PC2	S6PC1	S6PC0	R/W	STK6L
0F3H	-	-	-	-	-	S6PC10	S6PC9	S6PC8	R/W	STK6H
0F4H	S5PC7	S5PC6	S5PC5	S5PC4	S5PC3	S5PC2	S5PC1	S5PC0	R/W	STK5L
0F5H	-	-	-	-	-	S5PC10	S5PC9	S5PC8	R/W	STK5H
0F6H	S4PC7	S4PC6	S4PC5	S4PC4	S4PC3	S4PC2	S4PC1	S4PC0	R/W	STK4L
0F7H	-	-	-	-	-	S4PC10	S4PC9	S4PC8	R/W	STK4H
0F8H	S3PC7	S3PC6	S3PC5	S3PC4	S3PC3	S3PC2	S3PC1	S3PC0	R/W	STK3L
0F9H	-	-	-	-	-	S3PC10	S3PC9	S3PC8	R/W	STK3H
0FAH	S2PC7	S2PC6	S2PC5	S2PC4	S2PC3	S2PC2	S2PC1	S2PC0	R/W	STK2L
0FBH	-	-	-	-	-	S2PC10	S2PC9	S2PC8	R/W	STK2H
0FCH	S1PC7	S1PC6	S1PC5	S1PC4	S1PC3	S1PC2	S1PC1	S1PC0	R/W	STK1L
0FDH	-	-	-			S1PC10	S1PC9	S1PC8	R/W	STK1H
0FEH	S0PC7	S0PC6	S0PC5	S0PC4	S0PC3	S0PC2	S0PC1	S0PC0	R/W	STK0L
0FFH	-	-	-			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”）。

2.1.4.4 累加器 ACC

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。

➢ **例：ACC 和工作寄存器中断保护。**

```
.DATA           ACCBUF      DS   1          ; 定义 ACCBUF 为 ACC 数据存储单元。
                PFLAGBUF    DS   1          ; 定义 PFLAGBUF 为 PFLAG 数据存储单元。
.CODE
INT_SERVICE:
                B0XCH       A, ACCBUF       ;
                B0MOV       A, PFLAG
                B0MOV       PFLAGBUF, A     ; 存储 ACC 和 PFLAG。
                …
                B0MOV       A, PFLAGBUF
                B0MOV       PFLAG, A        ; 恢复 PFLAG。
                B0XCH       A, ACCBUF       ; 恢复 ACC。
                RETI                        ; 退出中断。
```

✸ **注：必须使用“B0XCH”指令进行 ACC 数据的中断恢复，否则 PFLAG 会被更改而导致出错。**

2.1.4.5 程序状态寄存器 PFLAG

寄存器 PFLAG 包含 ALU 运算状态信息，位 C、DC 和 Z 显示 ALU 的运算结果状态。

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

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。**

2.1.4.6 程序计数器 PC

程序计数器 PC 是一个 11 位二进制程序地址寄存器，分高 3 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	-	-	-	-	-	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
	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，跳过下一条指令
                JMP         C0STEP      ; 否则跳到 C0STEP.
                …
C0STEP:         NOP

                B0MOV       A, BUF0     ;
                B0BTS0      FZ          ; 若 Zero flag = 0，跳过下一条指令
                JMP         C1STEP      ; 否则跳到 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     ; 若 ACC = 12H，跳过下一条指令
                JMP         C0STEP      ; 否则跳到 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。若 PCL 溢出，PCH 不会自动进位。用户必须用程序来调整 PCH 的值。对于跳转表及其它应用，用户需计算 PC 的值以避免 PCL 溢出而导致程序出错。

✷ **注：PCL 溢出时，PCH 不能自动进位。用户必须用程序调整 PCH 的值以避免程序出错。**

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

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

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

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

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

2.1.4.7 Y, Z 寄存器

寄存器 Y 和 Z 都是 8 位缓存器，主要用途如下：

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

084H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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	X	X	X	X	X	X	X	X

083H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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	X	X	X	X	X	X	X	X

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

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

➢ **例：利用数据指针@YZ 对 RAM 数据清零。**

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

CLR_YZ_BUF:
        CLR     @YZ         ; @YZ 清零。

        DECMS   Z           ;
        JMP     CLR_YZ_BUF  ; 不为零。

        CLR     @YZ
END_CLR:                    ;
        …
```

2.1.4.8 R 寄存器

8 位缓存器 R 主要有以下两个功能：

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

082H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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	X	X	X	X	X	X	X	X

2.2 寻址模式

2.2.1 立即寻址

将立即数送入 ACC 或指定的 RAM 单元。

- **例：立即数 12H 送入 ACC。**

```
MOV     A, #12H
```

- **例：立即数 12H 送入寄存器 R。**

```
B0MOV   R, #12H
```

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

2.2.2 直接寻址

通过 ACC 对 RAM 单元数据进行操作。

- **例：地址 12H 处的内容送入 ACC。**

```
B0MOV   A, 12H
```

- **例：ACC 中数据写入 RAM 的 12H 单元。**

```
B0MOV   12H, A
```

2.2.3 间接寻址

通过指针寄存器（Y/Z）访问 RAM 数据。

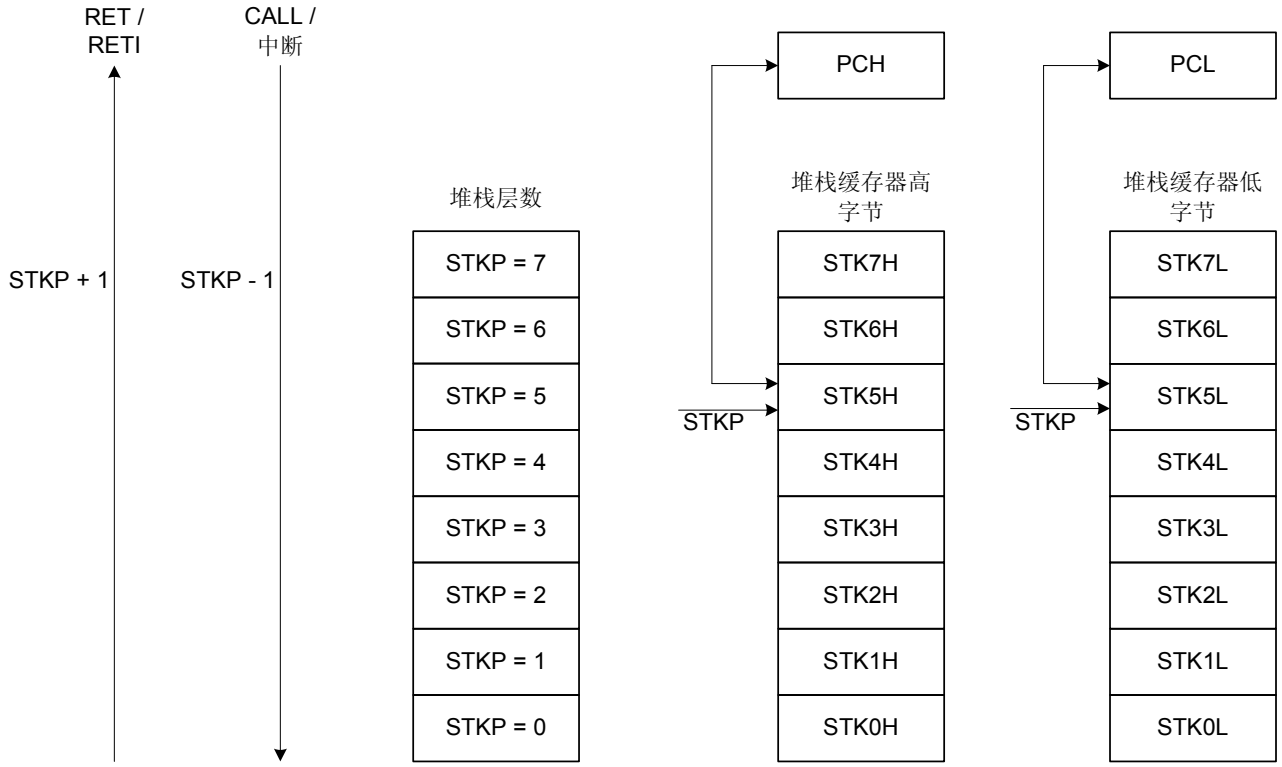
- **例：用 @YZ 实现间接寻址。**

```
B0MOV   Y, #0       ; Y 清零以寻址 RAM bank 0。
B0MOV   Z, #12H     ; 设定寄存器地址。
B0MOV   A, @YZ
```

2.3 堆栈操作

2.3.1 概述

SN8P1937 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。



2.3.2 堆栈寄存器

堆栈指针 STKP 是一个 3 位寄存器，存放被访问的堆栈单元地址，11 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 = 使能。

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

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

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

2.3.3 堆栈操作举例

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

堆栈层数	STKP			堆栈缓存器		说明
	STKPB2	STKPB1	STKPB0	高字节	低字节	
0	1	1	1	Free	Free	-
1	1	1	0	STK0H	STK0L	-
2	1	0	1	STK1H	STK1L	-
3	1	0	0	STK2H	STK2L	-
4	0	1	1	STK3H	STK3L	-
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

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

堆栈层数	STKP			堆栈缓存器		说明
	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	-

3 复位

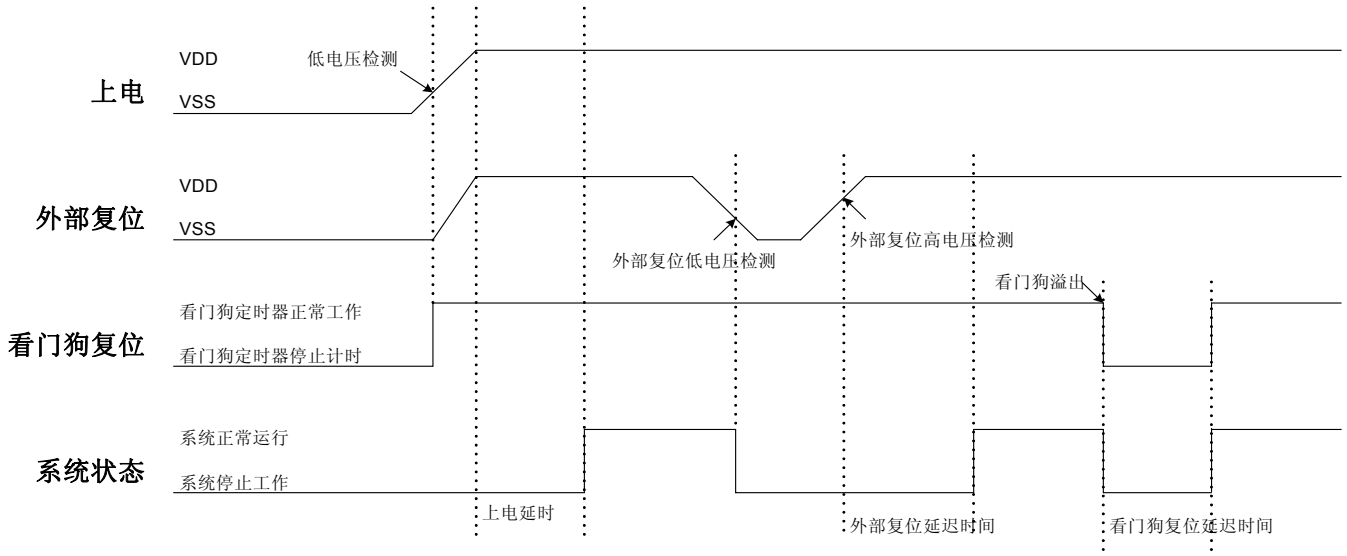
3.1 概述

SN8P1937 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。**

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。

系统复位需要一定的时间，并提供完整的上电复位规程。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 振荡器的启动时间非常短，晶体振荡器的启动时间则比较长。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。下面给出了复位时序图。



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。上电复位的时序如下：

- **上电：**系统检测到电源电压上升并等待其稳定；
- **外部复位：**系统检测外部复位引脚状态。如果不为高电平，系统保持复位状态直到外部复位引脚释放。
- **系统初始化：**所有的系统寄存器被置为默认状态；
- **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。

看门狗定时器运用注意事项：

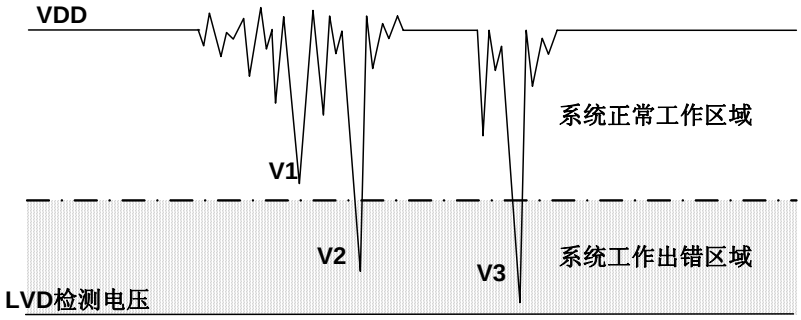
- 看门狗定时器清零之前，请检查 I/O 口的状态和 RAM 的内容可增强程序的可靠性；
- 不能在中断中对看门狗清零，否则无法侦测到主程序跑飞的状况；
- 程序中应该只在主程序中有一次清看门狗的动作，这种架构能够最大限度的发挥看门狗的保护功能。

✷ **注：关于看门狗定时器的详细内容，请参阅“看门狗定时器有关章节。**

3.4 掉电复位

3.4.1 概述

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



掉电复位电路图

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系统复位电压，因此系统维持在死区。

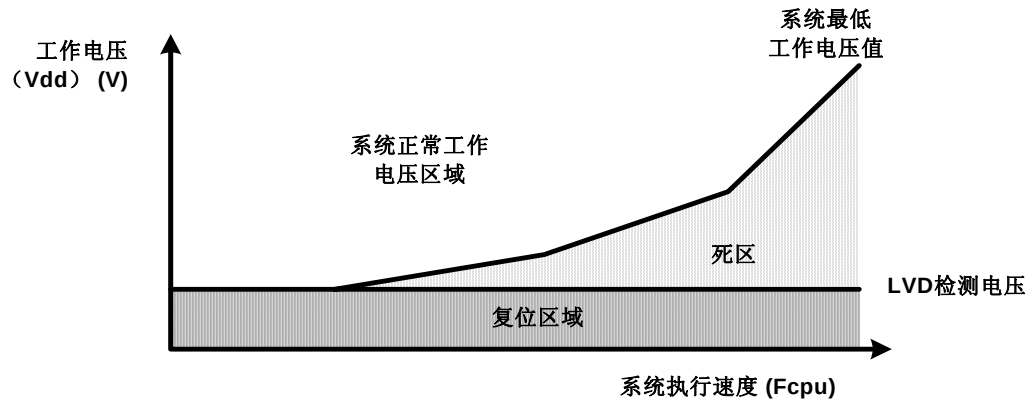
AC 运用中：

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，在系统将有可能进入不稳定工作状态。

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

3.4.2 系统工作电压

为了改善系统掉电复位的性能，首先必须明确系统具有的最低工作电压值。系统最低工作电压与系统执行速度有关，不同的执行速度下最低工作电压亦不同。电气特性一章给出了系统工作电压与执行速度之间的关系。



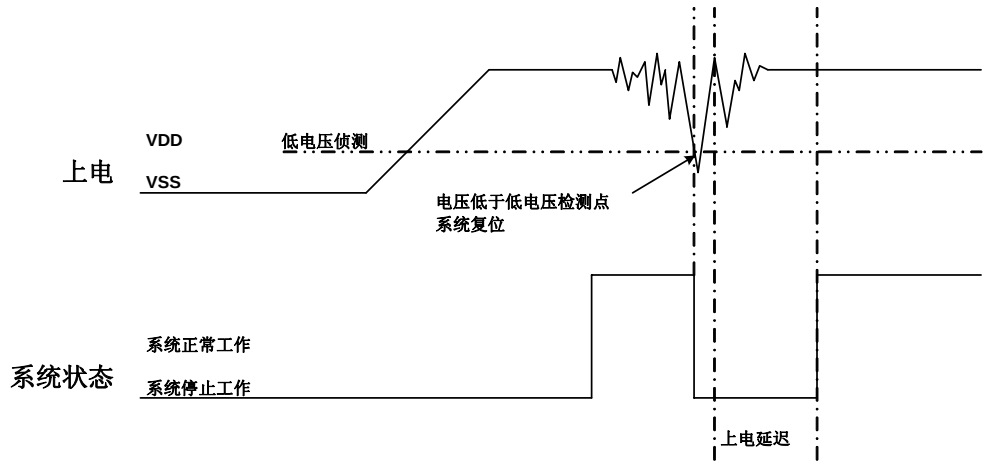
如上图所示，系统正常工作电压区域一般略高于系统复位电压，同时复位电压由 LVD 检测电平决定。当系统执行速度提高时，系统最低工作电压也相应提高。复位电压与最低工作电压之间的区域即是系统工作的死区。

3.4.3 掉电性能复位改进

如何改善系统掉电复位性能，有以下几点建议：

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

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



LVD 复位：

低电压检测（LVD）是 SONiX 8 位单片机的内置掉电复位保护装置，当 VDD 跌落并低于 LVD 检测电压值时，LVD 被触发，系统复位。不同的单片机有不同的 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，那么 LVD 就不能起到保护作用，就需要采用其它复位方法。电气特性一章中给出了更多关于 LVD 的详细内容。

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将看门狗清零。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大工作电压值越高，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。所有，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：

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

3.5 外部复位

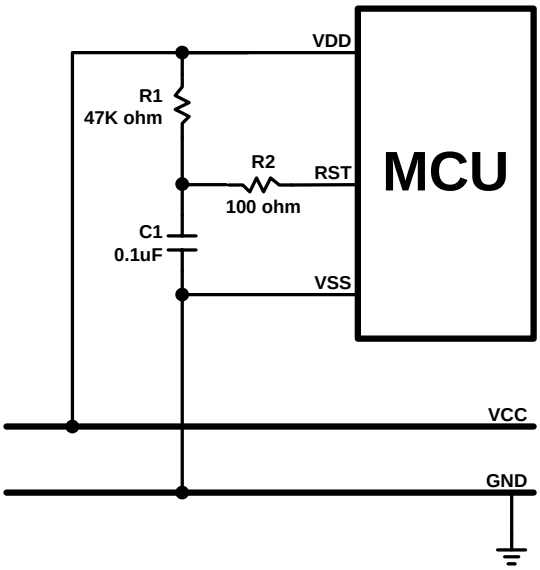
外部复位功能由编译选项“RESET PIN”控制。将该编译选项置为“Reset”，可使能外部复位功能。外部复位引脚为施密特触发器结构，低电平有效。复位引脚处于高电平时，系统正常运行。当复位引脚输入低电平信号时，系统复位。外部复位操作在上电和正常工作模式时有效。需要注意的是，在系统上电的过程中，外部复位引脚必须输入高电平，否则系统将一直保持在复位状态。外部复位的时序如下：

- **外部复位：**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。

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

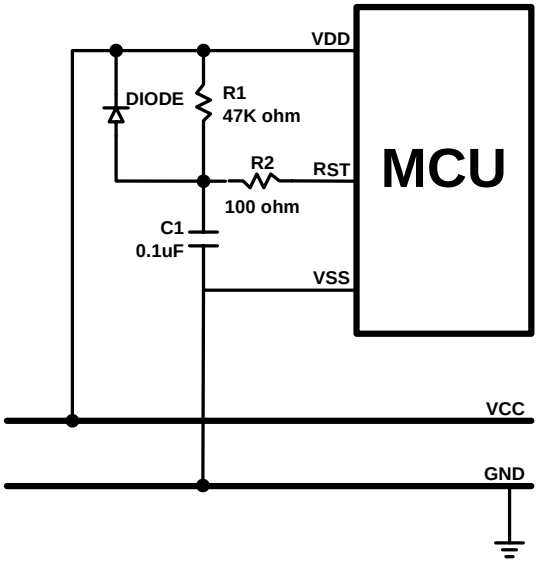
3.6.1 基本 RC 复位电路



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

✷ **注：外部复位电路不能解决非正常上电和掉电复位问题。**

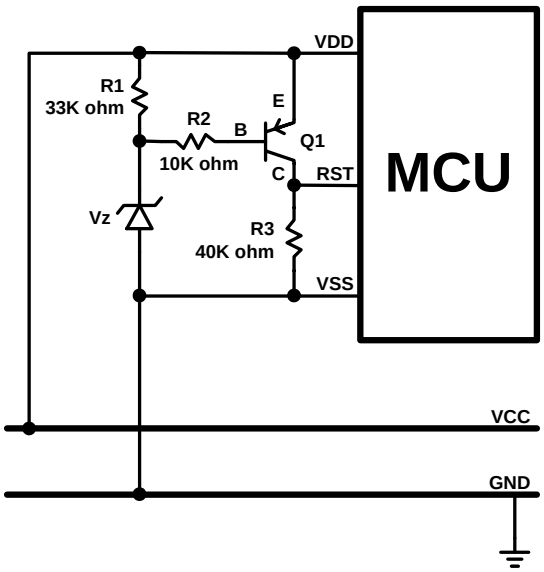
3.6.2 二极管&RC 复位电路



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。

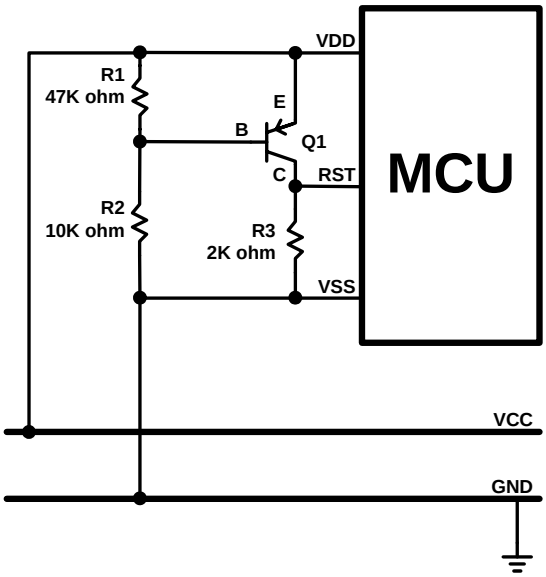
✷ **注：“基本 RC 复位电路”和“二极管及 RC 复位电路”中的电阻 R2 都是必不可少的限流电阻，以避免复位引脚 ESD（Electrostatic Discharge）或 EOS（Electrical Over-stress）)击穿 。**

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 偏置电压复位电路

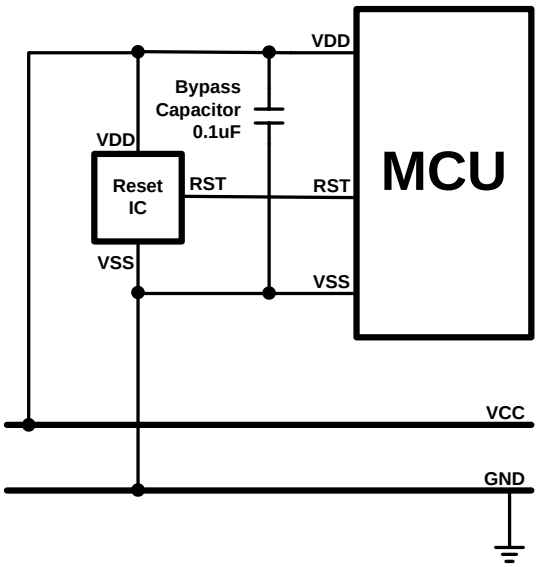


电压偏置复位电路是一种简单的 LVD 电路，基本上可以完全解决掉电复位问题。与稳压二极管复位电路相比，偏压复位电路的的检测电压值的精确度有所降低。电路中，R1 和 R2 构成分压电路，当 VDD 高于和等于分压值“0.7V x (R1 + R2) / R1”时，三极管集电极 C 输出高电平，单片机正常工作；VDD 低于“0.7V x (R1 + R2) / R1”时，集电极 C 输出低电平，单片机复位。

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 C 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



4 系统时钟

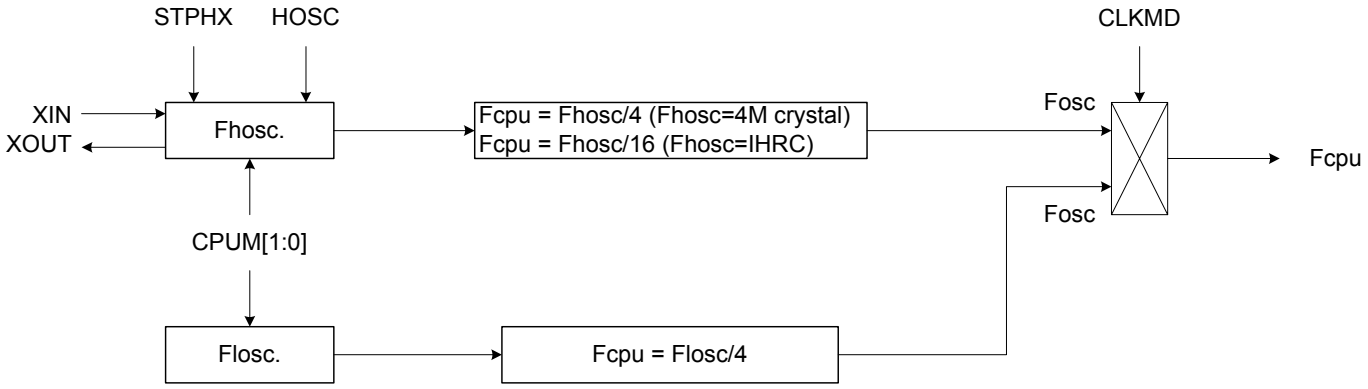
4.1 概述

SN8P1937 内带双时钟系统：高速时钟和低速时钟。高速时钟由外部振荡电路或内置 16MHz 高速 RC 振荡电路（IHRC 16MHz）提供，低速时钟则由外部 32768 晶体振荡电路（LXIN/LXOUT）或 RC 振荡电路提供。

高低速时钟都可以作为系统时钟，当系统在低速模式下工作时，时钟信号 4 分频之后作为系统指令周期 Fcpu。

- ☞ **普通模式（高速时钟）：Fcpu = Fhosc / 4，（Fhosc= 4M/8M）；**
  Fcpu = Fhosc / 16，（Fhosc=IHRC）。
- ☞ **低速模式（低速时钟）：Fcpu = Flosc/4。**

4.2 系统时钟框图



- **HOSC**：High_Clk 编译选项（code option）。
- **Fhosc**：外部高速时钟/内部高速 RC 时钟频率。
- **Flosc**：（1）内部低速 RC 时钟频率（编译选项现在 4M/IHRC）。
  （2）外部 32768 晶体振荡器电路（系统在 IHRC_RTC 模式下工作）。
- **Fosc**：系统时钟频率。
- **Fcpu**：指令执行频率。

4.3 OSCM 寄存器

寄存器 OSCM 控制振荡器的状态和系统模式。

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	WTCKS	WDRST	WDRATE	CPUM1	CPUM0	CLKMD	STPHX	-
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 1 **STPHX**：外部高速振荡器控制位。
0 = 运行；
1 = 停止。内部低速 RC 振荡器仍然运行。

Bit 2 **CLKMD**：高/低速时钟模式控制位。
0 = 普通（双时钟）模式，系统时钟来自高速时钟；
1 = 低速模式，系统时钟来自外部低速时钟或者内部低速 RC 时钟。

Bit[4:3] **CPUM0**：CPU 工作模式控制位。
0 = 普通模式；
1 = 睡眠模式。

Bit5 **WDRATE**：看门狗定时器速率选择控制位。
0 = $F_{CPU} \div 2^{14}$；
1 = $F_{CPU} \div 2^{8}$。

Bit6 **WDRST**：看门狗定时器复位控制位。
0 = 没有复位；
1 = 清看门狗定时器（详见看门狗定时器章节）。

Bit7 **WTCKS**：看门狗定时器时钟源选择控制位。
0 = F_{CPU}；
1 = 内部 RC 低速时钟（编译选项 INT_32K_RC 选择 Always_On 时，WTCKS 置 1）。

WTCKS	WTRATE	CLKMD	看门狗定时器溢出时间
0	0	0	1 / (fcpu ÷ 2^{14} ÷ 16) = 293 ms, Fosc=3.58MHz
0	1	0	1 / (fcpu ÷ 2^{8} ÷ 16) = 500 ms, Fosc=32768Hz
0	0	1	1 / (fcpu ÷ 2^{14} ÷ 16) = 32.7s, Fosc=32KHz@3V
0	1	1	1 / (fcpu ÷ 2^{8} ÷ 16) = 0.5s, Fosc=32KHz@3V
1	-	-	1 / (16K ÷ 512 ÷ 16) ~ 0.5s @3V

➢ **例：停止高速振荡器。**

```
B0BSET      FSTPHX          ; 停止外部高速振荡器。
```

➢ **例：系统进入睡眠模式后，高低速振荡器都停止运行。**

```
B0BSET      FCPUM0
```

4.4 系统高速时钟

内部 16MHZ RC 振荡器和外部振荡器都可作为系统高速时钟源，由代码选项“High_Clk”控制。

High_Clk 编译选项	说　　明
IHRC	内部 16MHz RC 振荡器作为系统时钟源，XIN/XOUT 作为普通的 I/O 口
IHRC_RTC	内部 16MHz RC 振荡器作为系统时钟源，XIN/XOUT 外接 32768Hz 晶振作为 RTC 时钟源。
4M	外部振荡器作为系统高速时钟，典型频率为 4MHz。

4.4.1 内部高速 RC 时钟

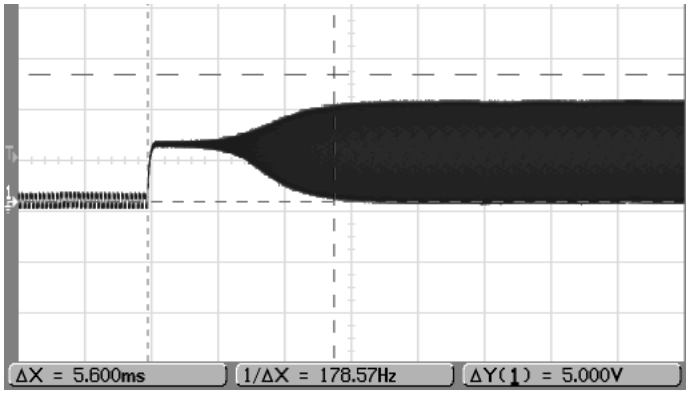
编译选项“IHRC”控制单片机的内置 RC 高速时钟（16MHZ）。IHRC 模式下，内置 16MHZ RC 振荡器作为系统时钟源，XIN 和 XOUT 引脚作为通用 I/O 口。

- **IHRC**：系统高速时钟来自内置 16MHz RC 振荡器，XIN/XOUT 引脚作为普通的 I/O 引脚。
- **IHRC_RTC**：系统高速时钟来自内置 16MHz RC 振荡器，XIN/XOUT 外接 32768Hz 晶振作为 RTC 时钟源。

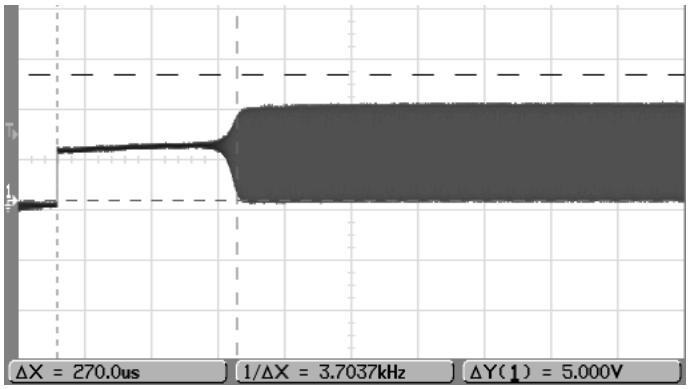
4.4.2 外部高速时钟

外部高速时钟共两种模式：石英/陶瓷振荡器和外部时钟源，由编译选项 High_Clk 控制具体模式的选择。振荡器上升时间与复位时间的长短密切相关。

4MHz Crystal

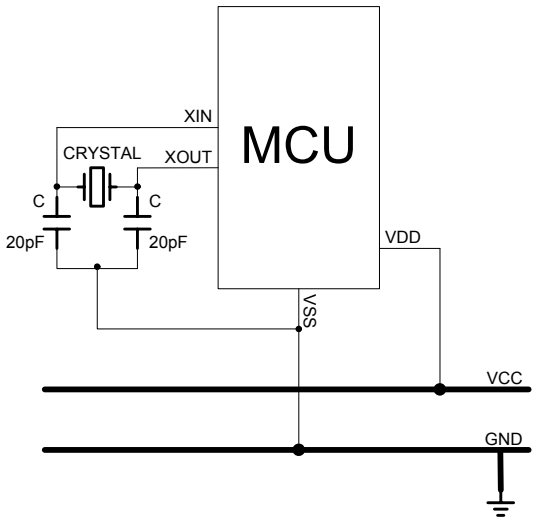


4MHz Ceramic



4.4.2.1 石英振荡时钟

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



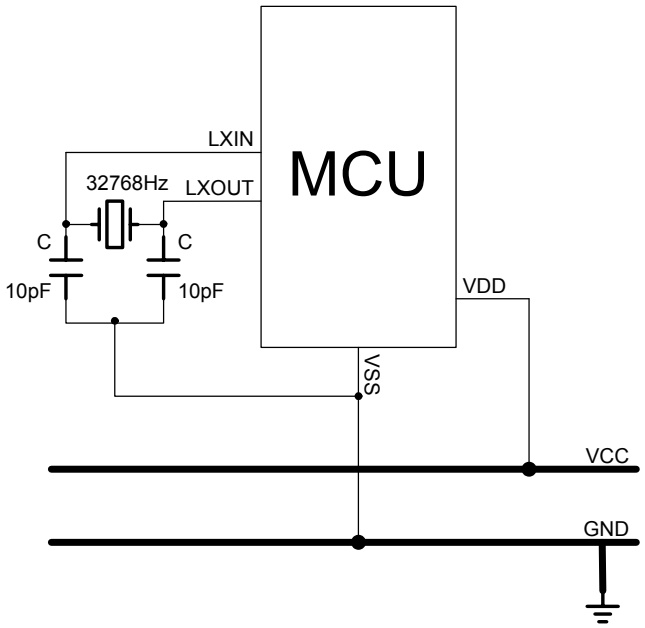
✷ **注：上图中， XIN/XOUT/VSS 引脚与石英/陶瓷振荡器以及电容 C 之间的距离越近越好。**

4.5 系统低速时钟

系统低速时钟源由内部低速 RC（ILRC）振荡器或外部 32768Hz 振荡器提供，在 IHRC_RTC 模式下，系统低速时钟源来自外部 32768Hz 振荡器。4MHz 和 IHRC 高速模式下，低速时钟源来自 ILRC。

4.5.1 晶体振荡时钟

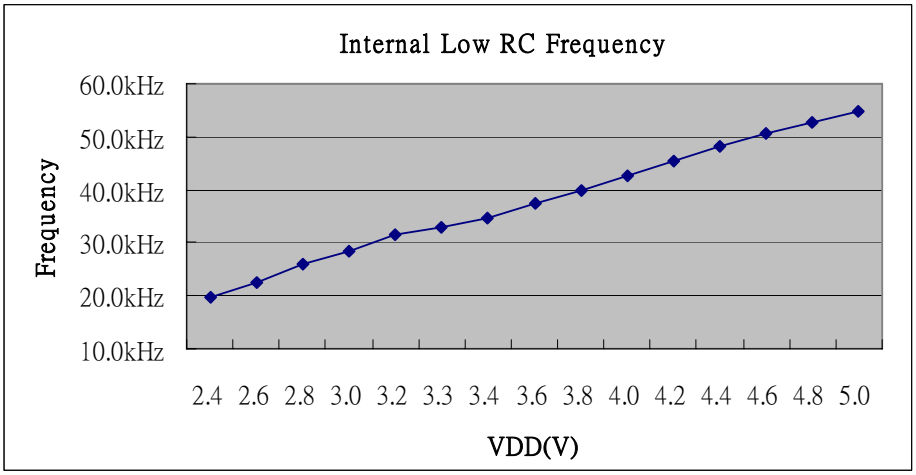
晶体振荡时钟由 LXIN、LXOUT 引脚驱动，32768Hz 晶体和 10uF 的电容要尽可能的靠近单片机。



- **Flosc = 32768Hz（晶体）**
- **低速模式下 Fcpu = Flosc / 4**

4.5.2 内部低速 RC 时钟（ILRC）

系统低速时钟源即内置的低速振荡器，采用 RC 振荡电路。低速时钟的输出频率受系统电压和环境温度的影响，通常为 3.3V 时输出 32KHz。输出频率与工作电压之间的关系如下图所示。



ILRC 时钟可作为看门狗定时器的时钟源。由 CLKMD 控制系统低速工作模式。

- **Flosc = ILRC。**
- **低速模式 Fcpu = Flosc / 4。**

睡眠模式下，外部低速时钟停止工作。

➢ **例：睡眠模式下停止内部低速振荡器。**

```
B0BSET      FCPUM0
```

✷ **注：不可以单独停止外部低速时钟；由寄存器 OSCM 的位 CPUM0、CPUM1 的设置决定外部低速时钟的状态。**

4.5.3 系统时钟测试

在设计过程中，用户可通过软件指令周期 Fcpu 对系统时钟速度进行测试。这种测试方法只在 RC 模式下有效。

➢ **例：外部振荡器的 Fcpu 指令周期测试。**

```
        B0BSET      P1M.0           ; 设置 P1.0 位输出模式以输出 Fcpu 的触发信号。
@@:
        B0BSET      P1.0
        B0BCLR      P1.0
        JMP         @B
```

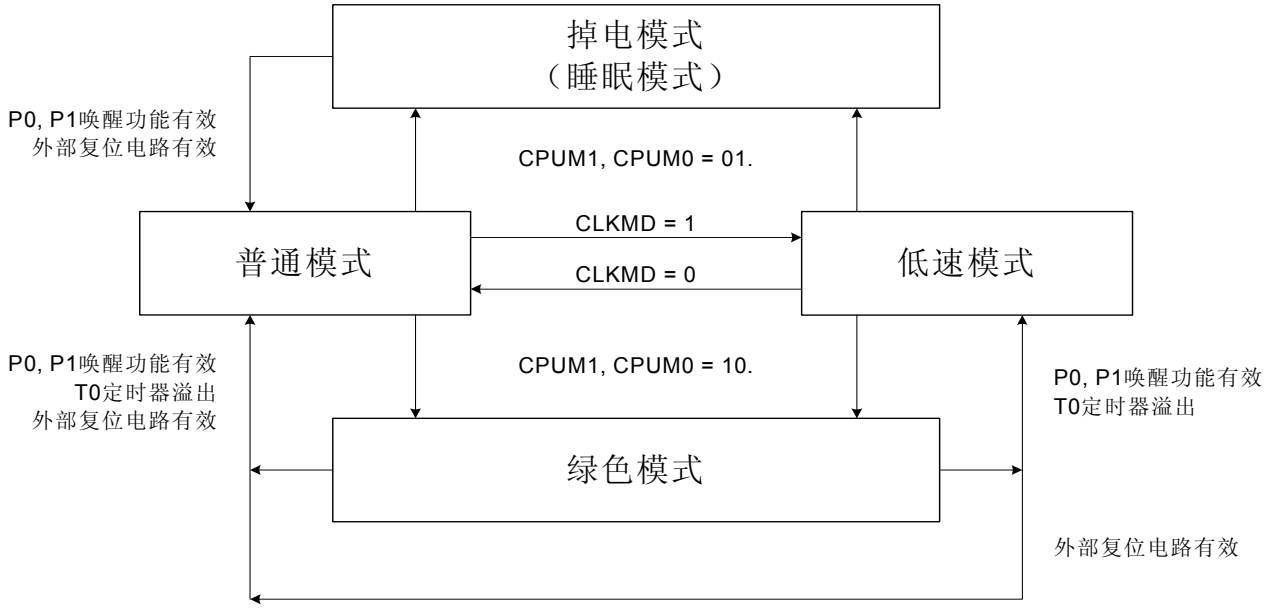
✷ **注：不能直接从 XIN 测量 RC 的频率，因为探针会影响 RC 频率的精度。**

5 系统工作模式

5.1 概述

SN8P1937 可以在如下 4 种工作模式之间切换：

- **高速模式；**
- **低速模式；**
- **睡眠模式；**
- **绿色模式。**



系统模式切换示意图

工作模式说明

工作模式	普通模式	低速模式	绿色模式	睡眠模式	备注
Fhosc	运行	STPHX	STPHX	停止	
Flosc	运行	运行	运行	停止	
CPU 指令	执行	执行	停止	停止	
T0 定时器	*有效	*有效	*有效	无效	* T0ENB=1 时有效
TC0 定时器	*有效	*有效	TC0GN	无效	* TC0ENB=1 时有效
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	参考编译选项说明
内部中断	全部有效	全部有效	T0, TC0	全部无效	TC0GN = 1
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, TC0, 复位	P0, P1, 复位	

5.2 系统模式切换举例

➢ **例：系统从普通/低速模式切换进入睡眠模式。**

```
B0BSET      FCPUM0      ;
```

✷ **注：睡眠模式下，只有具有唤醒功能的引脚和复位可以把系统唤醒进入普通模式。**

➢ **例：系统从普通模式切换进入低速模式。**

```
B0BSET      FCLKMD      ; 设置 CLKMD = 1，系统进入低速模式。
B0BSET      FSTPHX      ; 停止外部高速振荡器。
```

➢ **例：系统从低速模式切换进入普通模式（外部高速振荡器正常运行）。**

```
B0BCLR      FCLKMD      ;
```

➢ **例：系统从低速模式切换进入普通模式（外部高速振荡器停止运行）。**

外部高速时钟停止后，系统返回普通模式需要等待 20ms 的延迟时间等待外部时钟电路稳定。

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

        MOV         A, #54      ; 若 VDD = 3V，内部 RC=32KHz（典型值）
        B0MOV       Z, A
@@:     DECMS       Z           ; 需延迟 0.125ms X 162 = 20.25ms 等待外部时钟稳定。
        JMP         @B
                                ;
        B0BCLR      FCLKMD      ; 系统返回到普通模式。
```

➢ **例：从普通/低速模式切换进入绿色模式。**

```
B0BSET      FCPUM1      ;
```

✷ **注：绿色模式下禁止 T0/TC0 的唤醒功能，只有具有唤醒功能的引脚和复位引脚可以将系统唤醒。**

➢ **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。
        B0BCLR      FCPUM0      ; 设置 CPUMx=10。
        B0BSET      FCPUM1
```

✷ **注：绿色模式下使能 T0 的唤醒功能，即具有唤醒功能的引脚、复位引脚和 T0 定时器都可以将系统从绿色模式唤醒。T0 的唤醒周期由程序控制，T0ENB 必须置 1。**

5.3 系统唤醒

5.3.1 概述

系统在睡眠模式下时并不执行程序。唤醒触发信号可以将系统唤醒进入普通模式。唤醒触发信号来自外部触发信号（P0、P1 的电平变化）。

- 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）；

5.3.2 唤醒时间

系统进入睡眠模式后，高速时钟停止运行。把系统从睡眠模式下唤醒时，单片机需要等待 2048 个高速振荡器时钟周期以使振荡电路进入稳定工作状态，等待的这一段时间就称为唤醒时间。唤醒时间结束后，系统才进入到普通模式。

唤醒时间计算如下：

唤醒时间 = 1/Fosc * 2048 (sec) + 高速时钟启动时间

✷ **注：高速时钟的启动时间与 VDD 和振荡器类型有关。**

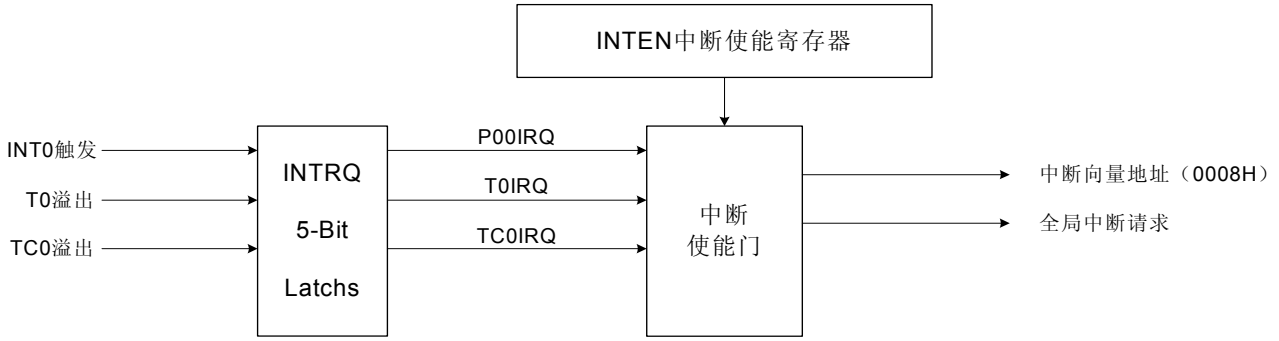
➢ **例：将系统从睡眠模式中唤醒，并设置系统进入普通模式。唤醒时间计算如下。**

唤醒时间= 1/Fosc * 2048 = 0.512 ms (Fosc = 4MHz)
总的唤醒时间 = 0.512 ms + 振荡器启动时间

6 中断

6.1 概述

SN8P1937 共有 3 个中断源：2 个内部中断 T0/TC0 和 1 个外部中断 INT0。外部中断可以将系统从睡眠模式中唤醒进入高速普通模式。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	-	-	TC0IEN	T0IEN	-	-	-	P00IEN
读/写	-	-	R/W	R/W	-	-	-	R/W
复位后	-	-	0	0	-	-	-	0

Bit 0 **P00IEN**：P0.0 外部中断(INT0) 控制位。
0 = 无效；
1 = 有效。

Bit 4 **T0IEN**：T0 中断控制位。
0 = 无效；
1 = 有效。

Bit 5 **TC0IEN**：TC0 中断控制位。
0 = 无效；
1 = 有效。

6.3 中断请求寄存器 INTRQ

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

0C8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTRQ	-	-	TC0IRQ	T0IRQ	-	-	-	P00IRQ
读/写	-	-	R/W	R/W	-	-	-	R/W
复位后	-	-	0	0	-	-	-	0

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

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

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

6.4 全局中断控制位 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。响应中断之前，必须保存 ACC 和 PFLAG 的内容。单片机没有提供特殊的指令来保护和恢复 ACC 和 PFLAG 的内容，因此用户必须使用指令“B0XCH”来保存/恢复 ACC；“B0MOV”保护/恢复 PFLAG，从而避免中断结束后主程序运行错误。

✷ **注：用户必须使用指令“B0XCH”来保护/恢复 ACC，否则 ACC 的操作会影响 PFLAG 的内容。**

➢ **例：执行中断时保存 ACC 和 PFLAG。**

```
.DATA           ACCBUF      DS 1
                PFLAGBUF    DS 1

.CODE
                ORG         0
                JMP         START

                ORG         8H
                JMP         INT_SERVICE

                ORG         10H
START:
                …

INT_SERVICE:
                B0XCH       A, ACCBUF       ; 保存 ACC。
                B0MOV       A, PFLAG
                B0MOV       PFLAGBUF, A     ; 保存 PFLAG。
                …
                …
                B0MOV       A, PFLAGBUF
                B0MOV       PFLAG, A        ; 恢复 PFLAG。
                B0XCH       A, ACCBUF       ; 恢复 ACC。

                RETI                        ; 退出中断。
                …
                ENDP
```

6.6 INT0（P0.0）中断

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

✷ **注：P0.0 的中断触发方式由 PEDGE 控制。**

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

Bit7 **PEDGEN**：中断和唤醒触发控制位。
0 = 禁止边沿触发功能；
P0：低电平唤醒触发，下降沿中断触发；
P1：低电平唤醒触发；（绿色模式唤醒时，P0、P1 下降沿触发中断）
1 = 使能边沿触发功能；
P0.0：由 P00G1 和 P00G0 位控制唤醒触发和中断触发；
P1：电平触发（上升/下降沿触发）唤醒功能。

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

➢ **例：INT0 中断请求设置，电平触发。**

```
        MOV     A, #98H
        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:

        …                       ; 保存 ACC 和 PFLAG。

        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:
        …                       ; 恢复 ACC 和 PFLAG。

        RETI                    ; 退出中断。
```

6.7 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。**

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

        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, #74H
        B0MOV       T0C, A          ;
        …
        …
EXIT_INT:
        …                           ; 恢复 ACC 和 PFLAG。

        RETI                        ; 退出中断。
```

✷ **注：1、RTC 模式下，必须延迟 1/2 RTC 时钟源（32768Hz）后才能清 T0IRQ，否则 RTC 间隔时间有误。延迟时间约 16uS，可以通过执行 T0 中断程序指令来达到此延时。**
2、RTC 模式下，不能在中断中复位 T0C。

➢ **例：T0 中断服务程序，带 RTC 功能。**

```
        ORG         8
        JMP         INT_SERVICE
INT_SERVICE:

        …                           ; 保存 ACC 和 PFLAG。

        B0BTS1      FT0IRQ          ; 检查是否有 T0 中断请求标志。
> 16us  JMP         EXIT_INT        ; T0IRQ = 0，退出中断。

        …                           ; T0 中断服务程序。
        …                           ; 中断时间必须长于 16uS。
        B0BCLR      FT0IRQ          ; 清 T0IRQ
        …
EXIT_INT:
        …                           ; 恢复 ACC 和 PFLAG。

        RETI                        ; 退出中断。
```

6.8 TC0 中断

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

➢ **例：TC0 中断请求设置。**

```
        B0BCLR      FTC0IEN         ; 禁止 TC0 中断。
        B0BCLR      FTC0ENB         ;
        MOV         A, #20H         ;
        B0MOV       TC0M, A         ; TC0 时钟＝Fcpu / 64。
        MOV         A, # 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         ;

        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。
```

6.9 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
T0IRQ	T0C 溢出
TC0IRQ	TC0C 溢出

多个中断同时发生时，需要注意的是：首先，必须预先设定好各中断的优先权；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     8H
        JMP     INT_SERVICE
INT_SERVICE:

        …                                   ; 保存 ACC 和 PFLAG。

INTP00CHK:                                  ; 检查是否有 INT0 中断请求。
        B0BTS1  FP00IEN                     ; 检查是否使能 INT0 中断。
        JMP     INTP01CHK                   ; 跳到下一个中断。
        B0BTS0  FP00IRQ                     ; 检查是否有 INT0 中断请求。
        JMP     INTP00                      ; 进入 INT0 中断。
INTT0CHK:                                   ; 检查是否有 T0 中断请求。
        B0BTS1  FT0IEN                      ; 检查是否使能 T0 中断。
        JMP     INTTC0CHK                   ; 跳到下一个中断。
        B0BTS0  FT0IRQ                      ; 检查是否有 T0 中断请求。
        JMP     INTT0                       ; 进入 T0 中断。
INTTC0CHK:                                  ; 检查是否有 TC0 中断请求。
        B0BTS1  FTC0IEN                     ; 检查是否使能 TC0 中断。
        JMP     INTTC1CHK                   ; 跳到下一个中断。
        B0BTS0  FTC0IRQ                     ; 检查是否有 TC0 中断请求。
        JMP     INTTC0                      ; 进入 TC0 中断。
INT_EXIT:
        …                                   ; 恢复 ACC 和 PFLAG。

        RETI                                ; 退出中断。
```

7 I/O 口

7.1 I/O 口模式

寄存器 PnM 控制 I/O 口的工作模式。

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

0C2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2M	-	-	-	-	-	-	P21M	P20M
读/写	-	-	-	-	-	-	R/W	R/W
复位后	-	-	-	-	-	-	0	0

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

Bit[7:0] **PnM[7:0]**：Pn 模式控制位（n = 0~5）。
0 = 输入模式；
1 = 输出模式。

✷ **注：**
1、 用户可以通过位操作指令（B0BSET、B0BCLR）对 I/O 口进行编程控制；
2、 P0 为单向输入端口；
3、 P2 口和 XIN/XOUT、LXIN/LXOUT 引脚共用。

➢ **例：I/O 模式选择。**

```
CLR     P1M         ; 设置所有端口为输入模式。
CLR     P2M

MOV     A, #0FFH    ; 设置所有端口为输出模式。
B0MOV   P1M,A
B0MOV   P2M, A

B0BCLR  P1M.0       ; 设置 P1.0 为输入模式。

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

7.2 I/O 口上拉电阻

0E1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1UR	-	-	-	-	P13R	P12R	P11R	P10R
读/写	-	-	-	-	W	W	W	W
复位后	-	-	-	-	0	0	0	0

0E2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2UR	-	-	-	-	-	-	P21R	P20R
读/写	-	-	-	-	-	-	W	W
复位后	-	-	-	-	-	-	0	0

0E5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5UR	-	-	P55R	P54R	P53R	P52R	P51R	P50R
读/写	-	-	W	W	W	W	W	W
复位后	-	-	0	0	0	0	0	0

✷ **注：P0 口的上拉电阻一直有效。**

➢ **例：使能 I/O 口的上拉电阻。**

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

7.3 I/O 口数据寄存器

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

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

0D2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P2	-	-	-	-	-	-	P21	P20
读/写	-	-	-	-	-	-	R/W	R/W
复位后	-	-	-	-	-	-	0	0

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

➢ **例：从输入端口读取数据。**

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

➢ **例：写入数据到输出端口。**

```
MOV     A, #0FFH    ; 所有的端口写 FFH。
B0MOV   P1, A
B0MOV   P5, A
```

➢ **例：写入 1 位数据到输出端口。**

```
B0BSET  P1.0    ; 设置 P1.0 为 1。

B0BCLR  P1.0    ; 设置 P1.0 为 0。
```

8 定时器

8.1 看门狗定时器 WDT

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。在一定的时间内执行指令“B0BSET FWDRST”可以清看门狗，如果没有定时将看门狗清零，则看门狗定时器溢出，系统复位。

0CAH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OSCM	WTCKS	WDRST	WDRATE	CPUM1	CPUM0	CLKMD	STPHX	0
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	-

Bit5 **WDRATE**：看门狗定时器分频选择位。
0 = FCPU ÷ 2^{14}；
1 = FCPU ÷ 2^{8}。

Bit6 **WDRST**：看门狗定时器复位控制位。
0 = 看门狗不复位；
1 = 看门狗清零。

Bit7 **WTCKS**：看门狗定时器时钟源选择位。
0 = FCPU；
1 = 内部低速 RC 时钟。

看门狗定时器溢出列表

WTCKS	WDRATE	CLKMD	看门狗定时器溢出时间
0	0	0	1 / (fcpu ÷ 214 ÷ 16) = 293 ms, Fosc=3.58MHz
0	1	0	1 / (fcpu ÷ 28 ÷ 16) = 500 ms, Fosc=32768Hz
0	0	1	1 / (fcpu ÷ 214 ÷ 16) = 32.7s, Fosc=32KHz@3V
0	1	1	1 / (fcpu ÷ 28 ÷ 16) = 0.5s, Fosc=32KHz@3V
1	-	-	1 / (16K ÷ 512 ÷ 16) ~ 0.5s @3V

✷ **注：由编译选项决定是否禁止看门狗定时器。**

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。**

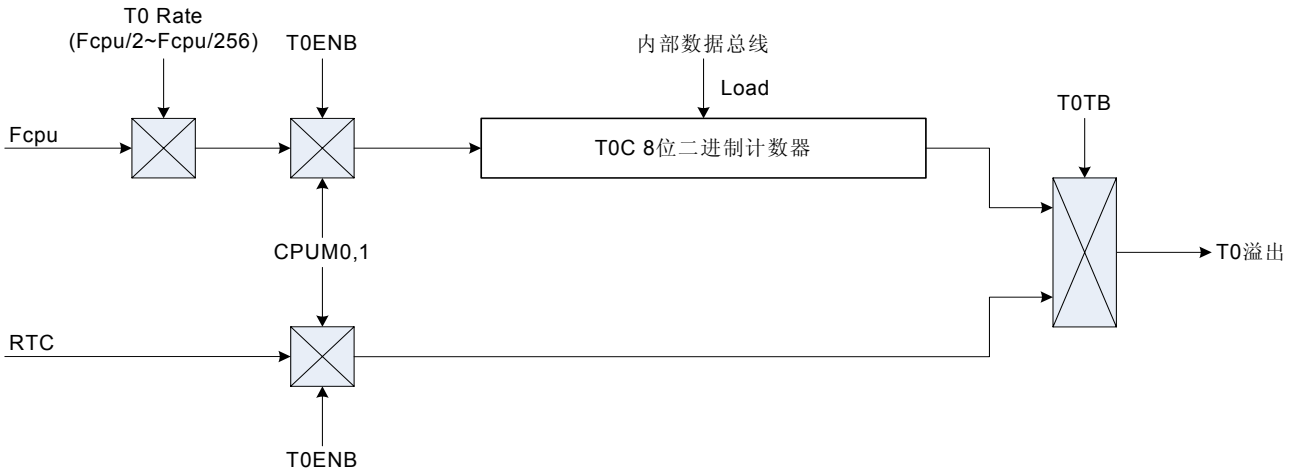
```
Main:
            …                           ; 检查 I/O 口的状态。
            …                           ; 检查 RAM 的内容。
Err:        JMP $                       ; I/O 口或 RAM 出错，不清看门狗等看门狗计时溢出。
                                        ;
Correct:                                ; I/O 口和 RAM 都正确，清看门狗定时器。
                                        ;
            B0BSET      FWDRST          ; 在整个程序中只有一处地方清看门狗。
            …
            CALL        SUB1
            CALL        SUB2
            …
            …
            JMP         MAIN
```

8.2 定时器 T0

8.2.1 概述

二进制定时/计数器 T0 溢出（从 0FFH 到 00H）时，T0 继续计数并给出一个溢出信号触发 T0 中断。定时器 T0 的主要用途如下：

- **8 位可编程定时计数器：**根据选择的时钟频率周期性的产生中断请求；
- **RTC 定时器：**根据时钟信号在实时的产生中断请求，仅当 T0TB=1 时 RTC 功能有效；
- **绿色模式唤醒功能：**T0ENB = 1 的条件下，T0 的溢出能够将系统从绿色模式下唤醒。



✷ **注：RTC 模式下，T0 的溢出时间间隔固定为 0.5S 而不受 T0C 控制。**

8.2.2 T0M 模式寄存器

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0rate2	T0rate1	T0rate0	-	-	TC0GN	T0TB
读/写	R/W	R/W	R/W	R/W	-	-	R/W	R/W
复位后	0	0	0	0	-	-	0	0

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

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

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 =关闭；
1 =开启。

✷ **注：RTC 模式下，T0RATE 的功能被屏蔽，T0 的间隔时间固定为 0.5 sec.**

8.2.3 T0C 计数寄存器

8 位计数寄存器 T0C 用于控制 T0 的中断间隔时间。

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 初始值计算公式如下：

T0C 初始值 = 256 -（T0 间隔时间 * 输入时钟）

➢ **例：T0 中断间隔时间置为 10ms，时钟信号 4MHz， Fcpu=Fosc/4， T0RATE=010 (Fcpu/64)。**

T0C 初始值 = 256 - (T0 间隔时间 * 时钟信号)
= 256 - (10ms * 4MHz / 4 / 64)
= 256 - (10-2 * 4 * 106 / 4 / 64)
= 100
= 64H

T0 间隔时间设置列表

T0RATE	T0CLOCK	高速模式（Fcpu = 4MHz / 4）		低速模式（Fcpu = 32768Hz / 4）	
		最大溢出间隔时间	单步间隔时间 = max/256	最大溢出间隔时间	单步间隔时间 = max/256
000	Fcpu/256	65.536 ms	256 us	8000 ms	31250 us
001	Fcpu/128	32.768 ms	128 us	4000 ms	15625 us
010	Fcpu/64	16.384 ms	64 us	2000 ms	7812.5 us
011	Fcpu/32	8.192 ms	32 us	1000 ms	3906.25 us
100	Fcpu/16	4.096 ms	16 us	500 ms	1953.125 us
101	Fcpu/8	2.048 ms	8 us	250 ms	976.563 us
110	Fcpu/4	1.024 ms	4 us	125 ms	488.281 us
111	Fcpu/2	0.512 ms	2 us	62.5 ms	244.141 us

✷ **注：RTC 模式下，T0C 设置无效，T0 的间隔时间固定为 0.5S。**

8.2.4 T0 操作流程

T0 的操作流程举例如下：

☞ **T0 计时停止，禁止 T0 中断，清 T0 中断请求标志。**

```
B0BCLR    FT0ENB    ; 关闭 T0 计数器。
B0BCLR    FT0IRQ    ; T0 中断请求标志位清零。
B0BCLR    FT0IEN    ; 禁止 T0 中断。
```

☞ **设置 T0 的计时速率。**

```
MOV       A, #0xxx0000b    ; T0M 的 bit4~bit6 将 T0 的速率控制在 x000xxxxb~x111xxxxb。
B0MOV     T0M,A            ; T0 定时器关闭。
```

☞ **设置 T0 的时钟信号（Fcpu 或 RTC）。**

```
B0BCLR    FT0TB    ; Fcpu 为时钟源。
```

或

```
B0BSET    FT0TB    ; RTC 为时钟源。
```

☞ **设置 T0 的间隔时间。**

```
MOV       A,#7FH
B0MOV     T0C,A    ; 设置 T0C 的值。
```

☞ **设置 T0 定时器的模式 。**

```
B0BSET    FT0IEN    ; 开启 T0 的中断功能。
```

☞ **开启 T0 定时器。**

```
B0BSET    FT0ENB    ; 开启 T0 定时器。
```

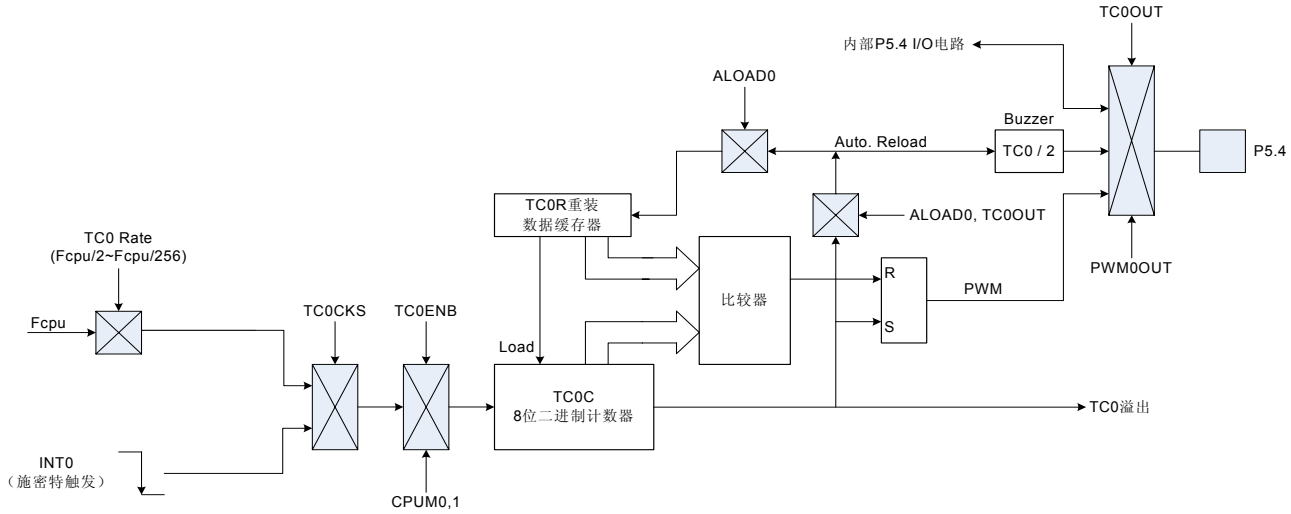
8.3 定时器/计数器 TC0

8.3.1 概述

定时/计数器TC0有2个时钟源，可根据实际需要选择内部时钟或外部时钟作为计时标准。其中，内部时钟源来自Fcpu。外部时钟源 INT0 从 P0.0 端输入（下降沿触发）。寄存器 TC0M 控制 TC0 时钟源的选择。当 TC0 从 0FFH 溢出到 00H 时，TC0 在继续计数的同时产生一个溢出信号，触发 TC0 中断请求。在 PWM 模式，TC0 的溢出时间由 ALOAD0 和 TC0OUT 位控制。

TC0 的主要功能如下：

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



8.3.2 TC0M 模式寄存器

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

Bit 0 **PWM0OUT**：PWM 输出控制。
0 = 禁止 PWM 输出；
1 = 使能 PWM 输出，PWM 输出占空比由 TC0OUT 和 ALOAD0 控制。

Bit 1 **TC0OUT**：TC0 溢出信号输出控制位。仅当 PWM0OUT = 0 时有效。
0 = 禁止，P5.4 作为输入/输出口；
1 = 允许，P5.4 输出 TC0OUT 信号。

Bit 2 **ALOAD0**：自动装载控制位。仅当 PWM0OUT = 0 时有效。
0 = 禁止 TC0 自动重装；
1 = 允许 TC0 自动重装。

Bit 3 **TC0CKS**：TC0 时钟信号控制位。
0 = 内部时钟；
1 = 外部时钟，由 P0.0/INT0 输入。

Bit [6:4] **TC0RATE[2:0]**：TC0 分频选择位。

TC0RATE [2:0]	TC0 时钟
000	Fcpu / 256
001	Fcpu / 128
010	Fcpu / 64
011	Fcpu / 32
100	Fcpu / 16
101	Fcpu / 8
110	Fcpu / 4
111	Fcpu / 2

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

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

8.3.3 TC0GN 标志

0D8H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0M	T0ENB	T0rate2	T0rate1	T0rate0	-	-	TC0GN	T0TB
读/写	R/W	R/W	R/W	R/W	-	-	R/W	R/W
复位后	0	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 1 **TC0GN**：TC0 绿色模式唤醒功能控制位。
0 = 禁止 TC0 的唤醒功能；
1 = 允许 TC0 的唤醒功能。

Bit [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 =关闭；
1 =开启。

✸ **注：TC0CKS = 1 时，TC0RATE 可以忽略不计。**

8.3.4 TC0C 计数寄存器

TC0C 控制 TC0 的时间间隔。

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

TC0C 初始值的计算公式如下：

TC0C 初始值 = N -（TC0 中断间隔时间 * 输入时钟）

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

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

TC0 中断间隔时间表

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

8.3.5 TC0R 自动装载寄存器

TC0 的自动装载功能由 TC0M 的 ALOAD0 位控制。当 TC0C 溢出时，TC0R 的值自动装入 TC0C 中。这样就能产生一个精确的时间值，而用户在使用的过程中就不需要在中断中复位 TC0C。

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

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：

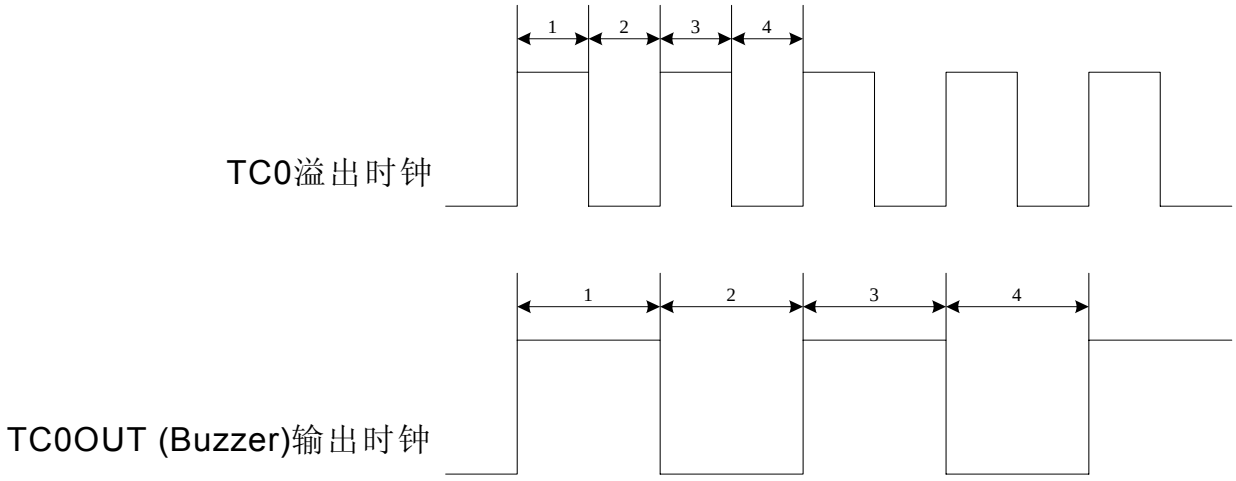
TC0R 初始值 = N -（TC0 中断间隔时间 * 输入时钟）

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

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

8.3.6 TC0 时钟频率输出（BUZZER）

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



➢ **例：设置 TC0OUT（P5.4）。若外部高速时钟选择 4MHz，系统时钟源采用外部时钟 Fcpu/4，程序中设置 TC0RATE2~TC0RATE1 = 110，TC0C = TC0R = 131，则 TC0 的溢出频率为 1KHz，TC0OUT 的输出频率为 0.5KHz。下面给出范例程序。**

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

MOV     A,#6            ; 自动装载参考值设置。
B0MOV   TC0C,A
B0MOV   TC0R,A

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

✷ **注：蜂鸣器的输出有效时，“PWM0OUT”必须被置为“0”。**

8.3.7 TC0 操作流程

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

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

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

☞ **设置 TC0 的速率 (不包含事件计数模式)。**

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

☞ **设置 TC0 的自动装载模式。**

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

或

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

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

```
; 设置 TC0 中断间隔时间，TC0OUT（Buzzer）频率或 PWM 占空比。
MOV         A,#7FH      ; TC0 的模式决定 TC0C 和 TC0R 的值。
B0MOV       TC0C,A      ; 设置 TC0C 的值。
B0MOV       TC0R,A      ; 在自动装载模式或 PWM 模式下设置 TC0R 的值。
```

☞ **设置 TC0 的模式。**

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

或

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

或

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

或

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

☞ **开启 TC0 定时器。**

```
B0BSET      FTC0ENB
```

9 LCD 驱动

SN8P1937 的 LCD 驱动包括 R 型和 C 型，具有 4 个 common 引脚和 12 个 segment 引脚，LCD 扫描的时序占用 1/4 占空比，1/2 或者 1/3 偏压。R 型和 C 型 LCD 均支持所有功能，都有 48 点驱动。R 型 LCD 驱动下，LCD 电源和偏置电压都可以通过内部/外部偏压电路进行调节；C 型 LCD 驱动下，通过内部 charge pump 进行调节。

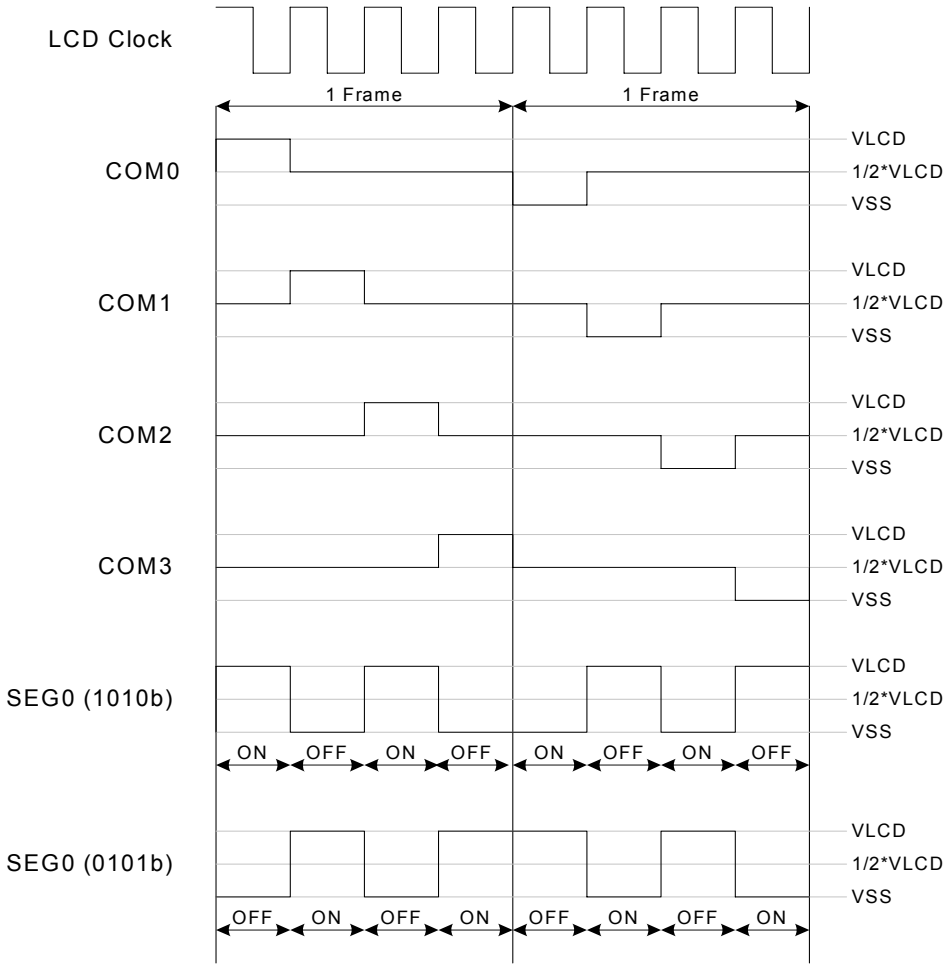
9.1 LCD 时序

LCD 时序表

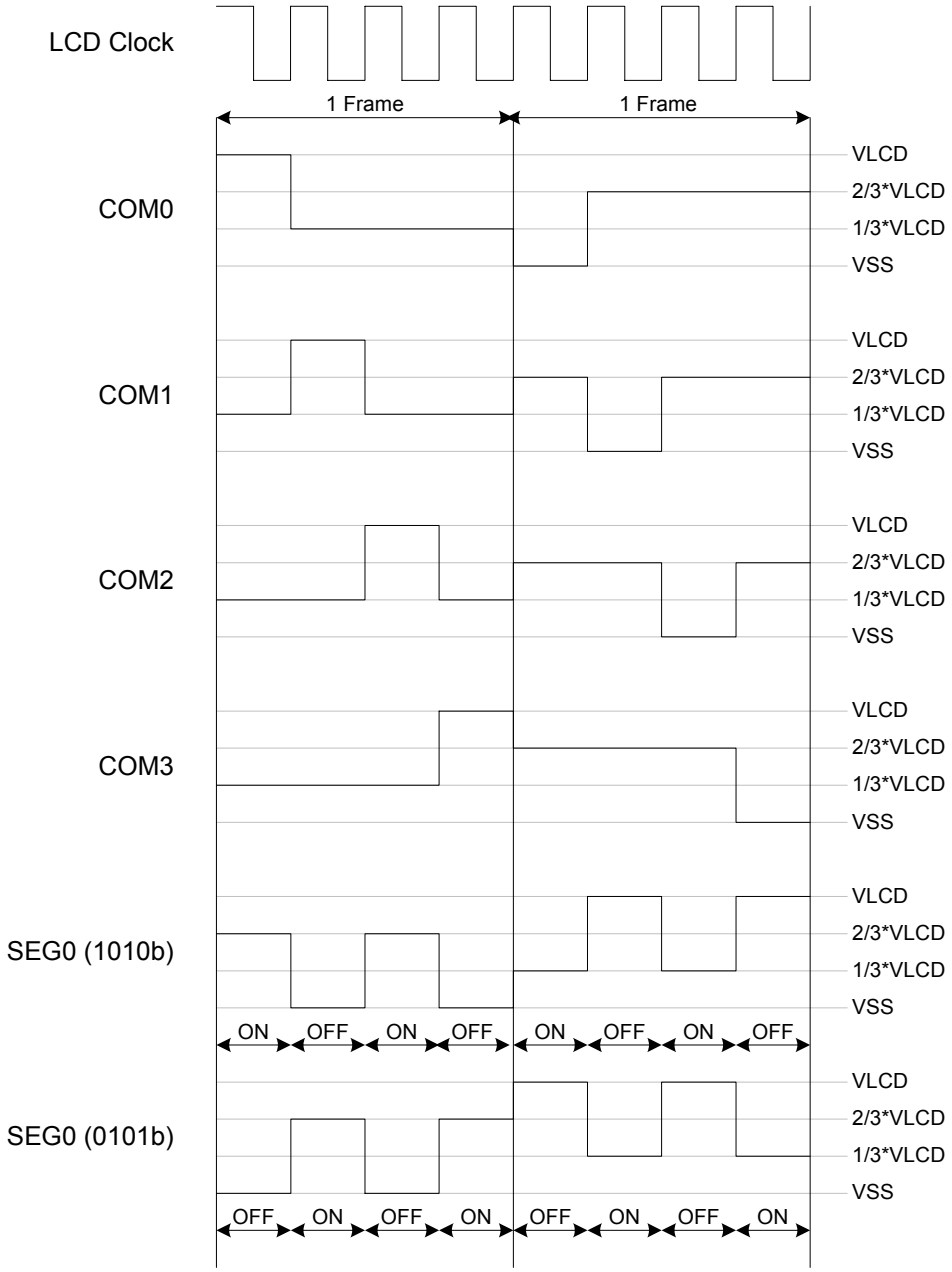
编译选项 High_Clk 选择 IHRC 或 4MHz				
LCDCLK	LCD clock source	LCDRATE	LCD Clock	Frame = LCD clock/4
0	Fhosc (4M X'tal)	X	4M/2^14 = 244.14Hz@4M	244.14/4 = 61.03Hz
0	Fhosc (IHRC)	X	4M/2^14 = 244.14Hz@4M	244.14/4 = 61.03Hz
1	Flosc	0	32K/64 = 500Hz@3V	500/4 = 125Hz
1	Flosc	1	32K/32 = 1000Hz@3V	1000/4 = 250Hz
1	Flosc	0	64K/64 = 1000Hz@5V	1000/4 = 250Hz
1	Flosc	1	64K/32 = 2000Hz@5V	2000/4 = 500Hz

✷ **注：系统处于 IHRC_RTC 模式时，LCD 帧 rate 固定为 64Hz，LCDCLK 位在 C 型 Charge pump 时钟设置时有效。**

● **LCD 驱动波形，1/4 duty，1/2 bias**



- **LCD 驱动波形，1/4 duty，1/3 bias**



9.2 LCDM1 寄存器

089H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LCDM1	LCDREF1	LCDREF0	LCDBNK	LCDTYPE	LCDENB	LCDBIAS	LCDRATE	LCDCLK
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	1	1

Bit0 **LCDCLK**：LCD 时钟源选择控制位。
0 = LCD 时钟 = 外部时钟 /2^14，帧 rate = LCD 时钟 / 4；
外部高速时钟为 4M，LCD 时钟为 244.14Hz，帧 Rate 为 244.14/4=61.03Hz
高速时钟为 3.58M，LCD 时钟为 218.51Hz，帧 rate 为 218.51/4＝54.62Hz
1 = LCD 时钟 = 内部 RC/32（LCDRATE=1）或内部 RC/64（LCDRATE=0），帧 Rate = LCD 时钟/4。

Bit1 **LCDRATE**：LCD 时钟 rate 控制位（LCDCLK=1）。
0 = LCD 时钟= 内部 RC/ 64；
1 = LCD 时钟= 内部 RC/ 32。

Bit2 **LCDBIAS**：LCD 偏压选择位。
0 = LCD 的偏压是 1/3；
1 = LCD 的偏压是 1/2。

Bit3 **LCDENB**：LCD 驱动使能控制位。
0 = 禁止；
1 = 使能。

Bit4 **LCDTYPE**：R 型/ C 型 LCD 驱动控制位。
0 = R 型；
1 = C 型。

Bit5 **LCDBNK**：LCD 显示控制位。
0 = 正常显示；
1 = 关闭 LCD。

Bit[7:6] **LCDREF[1:0]**：LCD 偏压分压电阻的选择值（R 型 LCD 驱动模式下）。
00 = 400K；
01 = 200K；
10 = 100K；
11 = 50K。

R 型和 C 型 LCD 驱动控制：

LCDENB	LCDTYPE	R 型驱动	C 型驱动	注释
0	0	禁止	禁止	LCD Charge Pump 关闭
0	1	禁止	禁止	LCD Charge Pump 开启
1	0	使能	禁止	LCD Charge Pump 关闭
1	1	禁止	使能	LCD Charge Pump 开启

9.3 LCDM2 寄存器

08AH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LCDM2	-	-	BGM	LCDCPK1	LCDCPK0	VCP2	VCP1	VCP0
读/写	-	-	R/W	R/W	R/W	R/W	R/W	R/W
复位后	-	-	0	0	0	0	1	1

Bit[0:2] **VCP[0:2]**：C 型 LCD charge pump 输出电压。

（LCDENB=1 且 LCDTYPE=1 时，LCD charge pump 开始 pumping）

	1/3 偏压条件下				1/2 偏压条件下			
VCP[2:0]	V1	V2	V3	VLCD	V1	V2	V3	VLCD
000	0V	0.900V	1.800V	2.7V	0V	1.350V	1.350V	2.7V
001	0V	0.933V	1.866V	2.8V	0V	1.400V	1.400V	2.8V
010	0V	0.966V	1.933V	2.9V	0V	1.450V	1.450V	2.9V
011	0V	1.000V	2.000V	3.0V	0V	1.500V	1.500V	3.0V
100	0V	1.033V	2.066V	3.1V	0V	1.550V	1.550V	3.1V
101	0V	1.066V	2.133V	3.2V	0V	1.600V	1.600V	3.2V
110	0V	1.100V	2.200V	3.3V	0V	1.650V	1.650V	3.3V
111	0V	1.133V	2.266V	3.4V	0V	1.700V	1.700V	3.4V

Bit[4:3] **LCDCPK[1:0]**：LCD charge pump 时钟选择位。

LCDCPK [1:0]	LCDCLK	Charge-pump 时钟源
00	1	Flosc / 1 = 32kHz
01	1	Flosc / 2 = 16kHz
10	1	Flosc / 8 = 4kHz
11	1	Flosc / 32 = 1kHz
00	0	Fhosc / 64 = 62.5kHz
01	0	Fhosc / 128 = 31.2kHz
10	0	Fhosc / 512 = 7.8kHz
11	0	Fhosc / 2048 = 1.9kHz

Bit5 **BGM**：Band Gap 模式选择位。

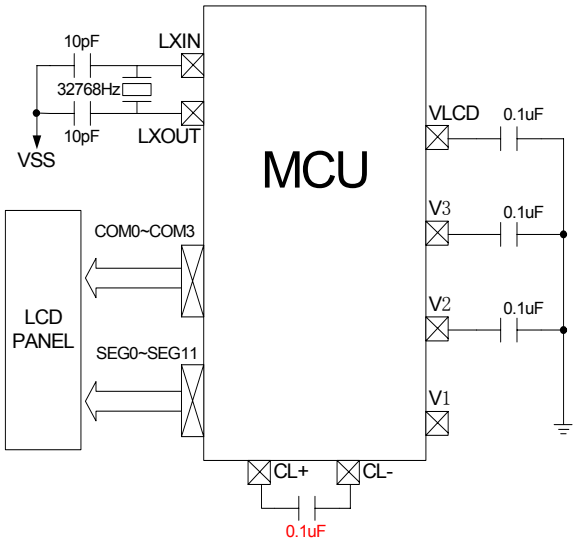
0 = 禁止；

1 =使能高精度 Band Gap 模式来提供准确的 VLCD 电压（BGM 必须置 1）

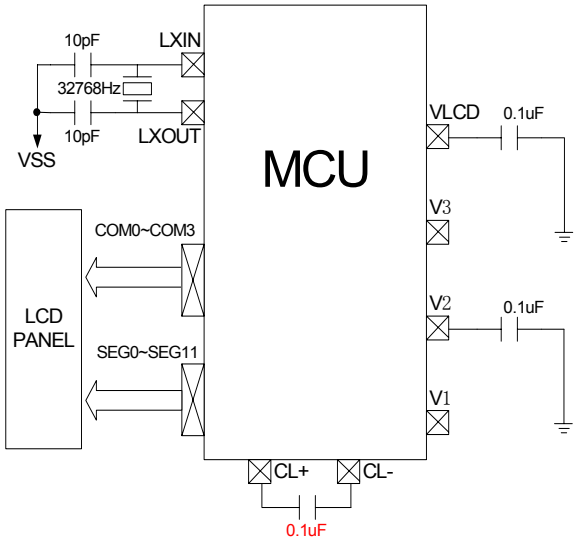
- **注 1：IHRC_RTC 模式下，Flocs 由 32768Hz 晶振提供，其他模式下，Flosc 由 ILRC 提供（ILRC：32kHz@3V, 64kHz@5V）。**
- **注 2：R 型 LCD 模式下，V1/V2/V3 引脚可以用来调整 LCD 偏压和驱动电流。**
- **注 3：C 型 LCD 模式下，引脚 VLCD/V2/V3 必须外接一个 0.1uf 的电容到 GND，以供电压 pumping。**
- **注 4：C 型 LCD 应用下，BGM 必须置 1。**
- **注 5：进入睡眠模式之前，将 LCDEBN、LCDTYPE 和 BGRENB 位清零以省电。**

9.4 C 型 LCD 驱动模式

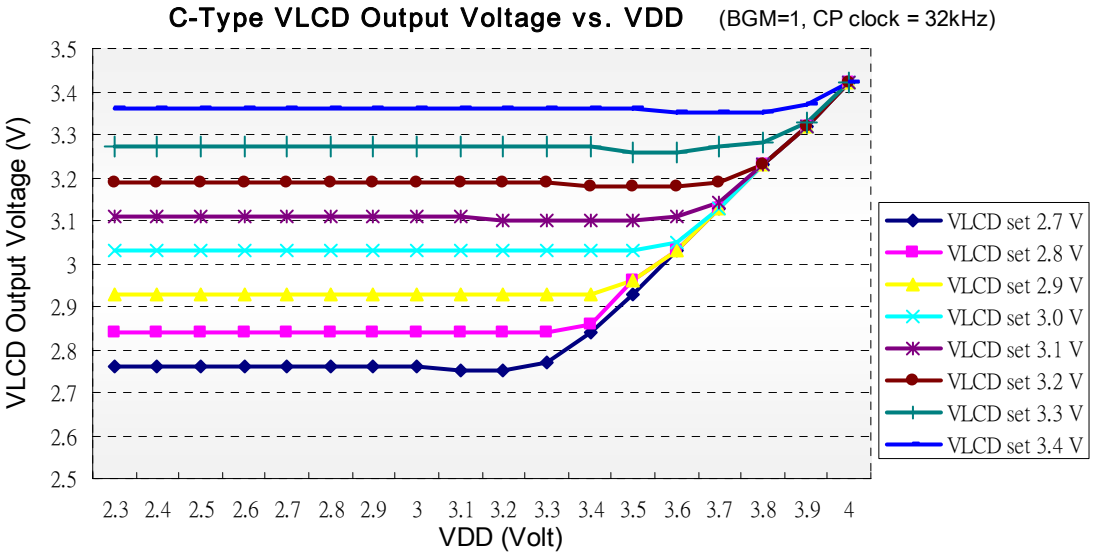
C 型 LCD 驱动模式支持 1/3 和 1/2 偏压 LCD 面板，LCD 电源（VLCD）由内部 LCD charge-pump 提供。C 型的电源功耗低于 R 型的电源功耗，这是由于没有额外的 DC 偏压电路消耗电流。Charge-pump 的电压电平为 VLCD 电压，V2 为 charge pump 的电压源，即 1/3 * VLCD；V3 是 2 * V2，即 2/3 * VLCD。C 型 LCD 模式下，LCDM1 寄存器的 LCDTYPE 位必须置 1。下图为 1/3 和 1/2 偏压 C 型 LCD 应用电路和 VLCD 输出电压曲线图。



C 型 LCD 应用电路（1/3 偏压）



C 型 LCD 应用电路（1/2 偏压）

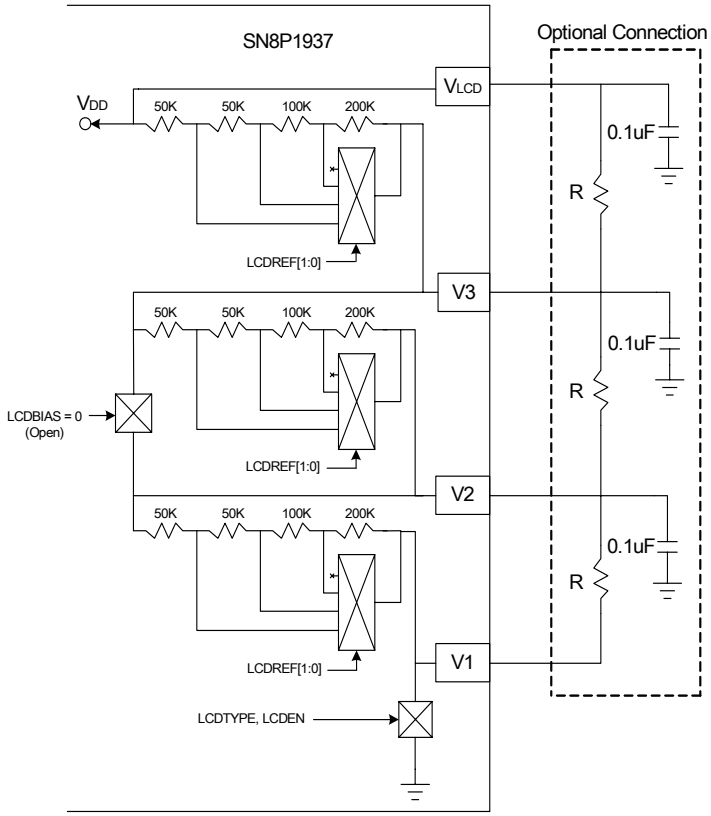


- **注 1：C 型 LCD 模式下，在 CL+和 CL-引脚之间、VLCD/V3/V2 和 VSS 之间连接一个 0.1uF 的电容。**
- **注 2：针对大面板 LCD 的应用，需要设置 LCDCPK[1:0]来增加 LCD 的 charge pump 时钟。**
- **注 3：C 型 LCD 模式下，在 VDD 小于 3.5V 的电源条件下，可以设置 VLCD 为 3.0V。**
- **注 4：VLCD 的输出电压范围为 2.7V~3.4V，精确度为±0.2V。**

9.5 R 型 LCD 驱动模式

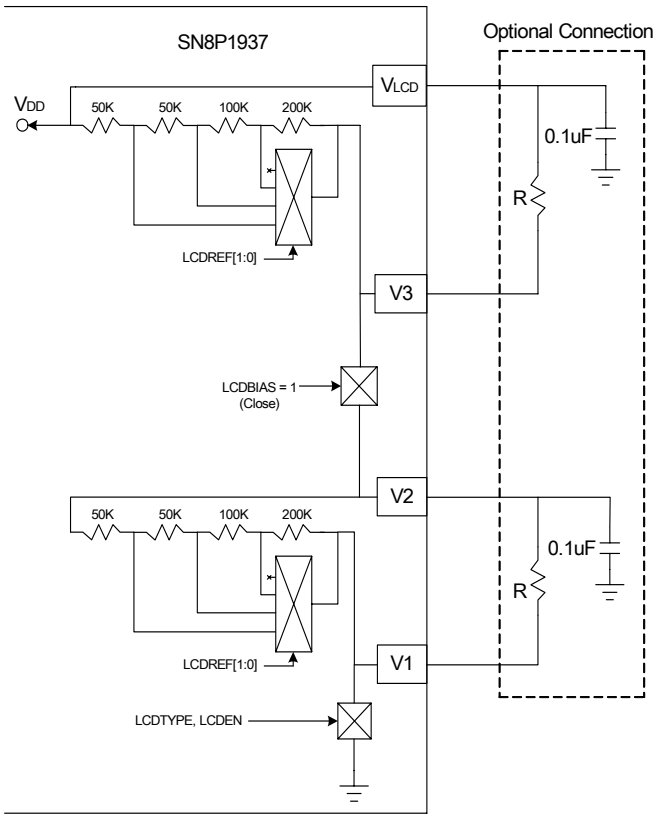
R 型 LCD 模式下，LCD 电源（VLCD）自动连接到内部 VDD，V3 和 V2 偏置电压来自内部电压，并由选择的电阻进行分压。根据选择的分压电阻来调节 LCD 的驱动电流，以适合不同的 LCD 面板应用，分压电阻可以在 400K、200K、100K 和 50K 之间选择，由 LCDREF[1:0]控制。另外，针对 LCD 大面板的应用，通过 VLCD、V3、V2 和 V1 引脚并联外部电阻增加驱动电流。下图为 1/4 占空比，1/3 和 1/2 偏压连接图。

- **1/4 占空比，1/3 偏压：**



LCD 电流功耗 = $\frac{VLCD}{\left(\frac{50k \times R}{50k + R}\right) \times 3}$ ，LCDREF= [11]，外部并联 R。

- **1/4 占空比，1/2 偏压：**



LCD 电流功耗 $= \dfrac{VLCD}{\left(\dfrac{50k \times R}{50k + R}\right) \times 2}$，LCDREF= [11]，外部并联 R。

✸ **注 1：C 型 LCD 驱动模式下，禁止 LCDREF [1:0]功能。**
✸ **注 2：R 型 LCD 驱动模式下，VLCD 电源自动连接到内部 VDD，VLCD 引脚不连接任何电源设备。**
✸ **注 3：外部电阻的大小根据 LCD 的尺寸来选择。**

9.6 LCD RAM 位置

RAM bank 15 的地址与 Common/Segment 引脚位置的关系：

	Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7
	COM0	COM1	COM2	COM3	-	-	-	-
SEG 0	00H.0	00H.1	00H.2	00H.3	-	-	-	-
SEG 1	01H.0	01H.1	01H.2	01H.3	-	-	-	-
SEG 2	02H.0	02H.1	02H.2	02H.3	-	-	-	-
SEG 3	03H.0	03H.1	03H.2	03H.3	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
SEG 11	11H.0	11H.1	11H.2	11H.3	-	-	-	-

- **例：开启 LCD 功能。**
  置 LCD 的控制位（LCDENB）和 LCD RAM 显示 LCD。

```
B0BSET      FLCDENB      ; LCD 驱动。
```

10 在线烧录（ISP）

10.1 概述

SN8P1937 具有在线烧录 ROM 功能（ISP ROM），为用户将数据存储在 ROM 中提供了一种简易的方式。选择 ROM 地址后，执行 ROM 烧录指令-ROMWDT，并向 VPP/RST 输入 12.5V 的电压，由寄存器 ROMCNT 控制烧录时间。烧录完成后，ROMDAH/ROMDAL 中的数据被烧录到 ROMADRH/ROMADRL 地址处。

10.2 ROMADRH/ROMADRL 寄存器

0A0H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ROMADRH	VPPCHK	-	-	-	-	ROMADR10	ROMADR9	ROMADR8
读/写	R	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

0A1H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ROMADRL	ROMADR7	ROMADR6	ROMADR5	ROMADR4	ROMADR3	ROMADR2	ROMADR1	ROMADR0
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

VPPCHK：VPP 引脚烧录电压检测。
0 = VPP 的电压未达到 12.5V，不能在线烧录；
1 = VPP 的电压达到 12.5V，可以在线烧录。

✷ **注：使用宏指令@B0BTS1_FVPPCHK 或@B0BTS0_FVPPCHK 检测 VPP 的电压。**

ROMADR[14:0]：ISP ROM 烧录地址。需要烧录的 ROM 地址。

10.3 ROMDAH/ROMDAL 寄存器

0A2H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ROMDAH	ROMDA15	ROMDA14	ROMDA13	ROMDA12	ROMDA11	ROMDA10	ROMDA9	ROMDA8
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

0A3H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ROMDAL	ROMDA7	ROMDA6	ROMDA5	ROMDA4	ROMDA3	ROMDA2	ROMDA1	ROMDA0
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

ROMDA[15:0]：ISP ROM 烧录数据。需要烧录到 ROM 中的数据。

10.4 ROMCNT 寄存器和 ROMWRT 指令

0A4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ROMCNT	ROMCNT7	ROMCNT6	ROMCNT5	ROMCNT4	ROMCNT3	ROMCNT2	ROMCNT1	ROMCNT0
读/写	W	W	W	W	W	W	W	W
复位后	-	-	-	-	-	-	-	-

Bit[7:0] **ROMCNT[7:0]**：ISP ROM 烧录时间计数器。
ISP ROM 烧录时间由 ROMCNT[7:0]控制；
烧录时间为（256-ROMCNT）*4/Fcpu；
建议烧录时间为 1ms。

Fcpu	ROMCNT	烧录时间
1MIPs	6	1ms

设置完成后，执行 ROMWRT 指令，将数据 ROMDA[15:0]烧录到 ROMADR[14:0]中。

- **注 1：在线烧录时，需保持 VDD=5V；**
- **注 2：执行 ROMWRT 指令后，须加 3 条 NOP 指令以延时；**
- **注 3：请在室温条件下（25℃）进行在线烧录。**

10.5 ISP ROM 示例程序

在线烧录示例程序。

```
; 保留 ISP ROM 区域为 0FFFFH。
ORG             0100H
@CALDATA:
                DW              0xFFFF
                ……..
; 烧录数据 0AA55H 到地址@CALDATA 中。
                MOV             A, #@CALDATA$L
                B0MOV           ROMADRL, A          ; 将低字节地址存入 ROMADRL。
                MOV             A, #@CALDATA$H
                B0MOV           ROMADRH, A          ; 将低字节地址存入 ROMADRH。
                MOV             A, #55H
                B0MOV           ROMDAL, A           ; 将低字节数据存入 ROMDAL。
                MOV             A, #0AAH
                B0MOV           ROMDAH, A           ; 将低字节数据存入 ROMADRH。
; 检测 VPP 电压。
                @B0BTS1_ FVPPCHK                    ; 检测 VPP 电压是否为 12.5V。
                JMP             $-1                 ; 若 VPP 电压不为 12.5V，继续等待。
; 设置烧录计数器，开始在线烧录。
@ROM_WRT:       MOV             A,#6                ; 设置烧录计数器。
                B0MOV           ROMCNT,A
                ROMWRT                              ; 开始在线烧录。
                NOP                                 ; NOP 指令延时。
                NOP                                 ; NOP 指令延时。
                NOP                                 ; NOP 指令延时。
; 检测 VPP 电压。                                   ; 设置 VPP=VDD。
                @B0BTS0_ FVPPCHK                    ; 检测 VPP 是否等于 VDD。
                JMP             $-1                 ; 若 VPP 仍为 12.5V，继续等待。

; 检查烧录数据。
                B0MOV           Z, #@CALDATA$L
                B0MOV           Y, #@CALDATA$H
                MOVC                                ; 在线烧录的数据存入 A 和 R 中。

                CMPRS           A,#55H
                JMP             @WRT_ERR
                B0MOV           A, R
                CMPRS           A,#0AAH
                JMP             @WRT_ERR            ; 对在线烧录的数据进行校验。
                ……..
```

11 Regulator, PGIA 和 ADC

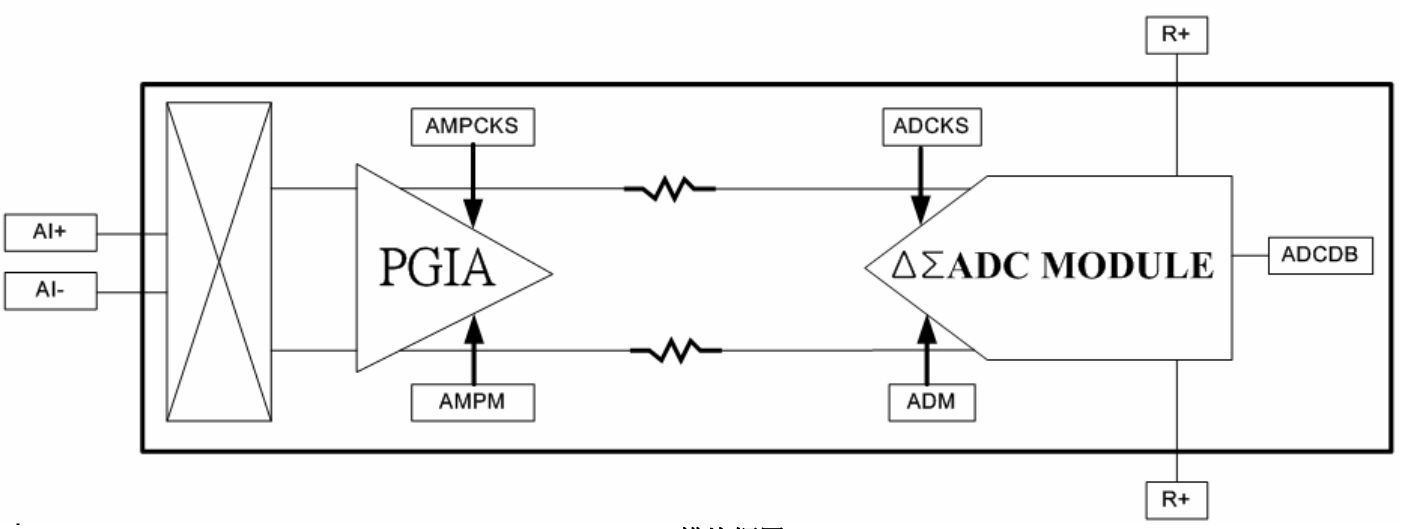
11.1 概述

SN8P1937 内置稳压器 Regulator，从 AVDDR 输出稳定的 2.4V，而从 AVE+输出稳定的 1.5V，最大的驱动电流为 10mA。AVDDR 给内部电路（PGIA、ADC）和外部传感器（压力传感器或热敏电阻）提供稳定的电压。SN8P1937 具有完整的 ΔΣ 模拟数字转换器（ADC），具有 16 位性能，高达 65536 阶的分辨率。ADC 有 2 个不同的输入信道模式：（1）1 个全差分输入；（2）2 个单端输入。ADC 在压力测量和医用仪表方面可以进行单/双极性的测量。内置增益可调的低噪声可编程增益放大器（PGIA），在应用时，可以选择 1x、16x、32x、64x 和 128x 五种增益。

11.2 模拟电路

下图是 PGIA 和 ADC 模块结构简图，由一个多路选择器（用于输入通道的选择），一个可编程增益放大器（PGIA）和 ΔΣ ADC 模块组成。

为了使 ADC 输出的范围达到最大，ADC 的输入信号电压 V（X+、X-）应该接近于但不能超过参考电压 V（R+、R-），选择一个合适的参考电压和合适的 PGIA 可以使 ADC 的输出范围很大。相关的控制位是 ADCM 寄存器的 RVS（参考电压选择）位和 AMPM 寄存器的 GS[2:0]（增益选择）位。



ADC 模块框图

11.3 电压稳压器

SN8P1937 内置一个电压稳压器，提供 2.4V 的稳定电源（来自 AVDDR）和 1.5V 的稳定电源（来自 AVE+），最大驱动电流 10mA。寄存器 CPM 可以控制 AVDDR、AVE+和 ACM 的电压输出状态。由于 PGIA 和 ADC 的电源来自 AVDDR，因此在使能 PGIA 和 ADC 之前要打开 AVDDR（AVDDRENB = 1），而 AVDDR 端的电压由 VDD 管控。

11.3.1CPM-Charge Pump 模式寄存器

095H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CPM	ACMENB	AVDDRENB	AVESEL	AVENB	ACMSEL	-	-	-
读/写	R/W	R/W	R/W	R/W	R/W	-	-	-
复位后	0	0	0	0	0	-	-	-

Bit3: **ACMSEL**：ACM 电压选择控制位。
0 = ACM 输出 0.4V。
1 = 保留。

Bit4: **AVENB**：AVE+电压输出控制位
0 = 禁止 AVE+输出电压；
1 = 使能 AVE+输出电压。

Bit5 **AVESEL**：AVE+电压选择控制位。
0 = AVE+输出 1.5V；
1 = 保留。

Bit6 **AVDDRENB**：Regulator（AVDDR）电压使能控制位。
0 = 关闭 Regulator，AVDDR 输出 0V 电压；
1 = 打开 Regulator，AVDDR 输出 2.4V 电压。

Bit7 **ACMENB**：模拟电路公共端（ACM）电压使能控制位。
0 = 关闭 ACM，ACM 的输出电压为 0V；
1 = 打开 ACM，ACM 的输出电压为 0.4V。

✷ 注 1：在使能下列功能之前，必须先打开 Band Gap 的参考电压（详见 AMPM 寄存器）。
1、 AVDDR、AVE+和 ACM 稳压器；
2、 PGIA 功能；
3、 ADC 功能；
4、 低电压检测功能。
✷ 注 2：PGIA 和 ADC 可以在低速模式下工作，但必须重新设置 AMPCKS 寄存器的值。
✷ 注 3：来自 AVE+（不包括压力传感器）或 AVDDR 的电流功耗都不是双倍的。

11.4 PGIA –可编程增益放大器

SN8P1937 内置一个增益可调的低噪声可编程增益放大器（PGIA），通过寄存器 AMPM 可以选择 1x、16x、32x、64x 和 128x 增益。PGIA 还提供 2 种信道选择模式：（1）1 个全差分输入；（2）2 个单端输入，由寄存器 AMPCHS 控制。

11.4.1AMPM- 放大器模式寄存器

090H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
AMPM	CHPENB	BGRENB	FDS1	FDS0	GS2	GS1	GS0	AMPENB
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	1	0	1	1	1	1	1	0

Bit0 **AMPENB**：PGIA 功能使能控制位。
0 = 禁止 PGIA 功能；
1 = 使能 PGIA 功能。

Bit[3:1] **GS [2:0]**：PGIA 增益选择控制位。

GS [2:0]	PGIA 增益
000	16
001	32
010	64
011	128
100,101,110	保留
111	1

Bit[5:4] **FDS [1:0]**：Chopper 低频设置位。
11 = 全部应用。

Bit6 **BGRENB**：Band Gap 参考电压使能控制位。
0 = 禁止 Band Gap 参考电压；
1 = 使能 Band Gap 参考电压。

Bit7 **CHPENB**：Chopper 使能控制位。
0 = 禁止 PGIA Chopper；
1 = 使能 PGIA Chopper。

✸ **注：当选择增益 1x 时可以禁止 PGIA（AMPENB = 0）以省电。**
✸ **注：在所有的应用中请设置 FDS[1:0] = “11”。**

11.4.2AMPCKS- PGIA 时钟选择寄存器

092H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
AMPCKS	-	-	-	-	-	AMPCKS1	AMPCKS1	AMPCKS0
读/写	-	-	-	-	-	W	W	W
复位后	-	-	-	-	-	0	0	0

Bit[2:0] **AMPCKS [2:0]**：寄存器设置PGIA Chopper的工作时钟，建议Chopper的时钟选择为：31.25KHz @4MHz。

PGIA 时钟 = Fcpu / 32 / (2^AMPCKS)

在不同的 Fosc 频率下，AMPCKS[2:0]寄存器的值请参阅下表：

AMPCKS2	AMCKS1	AMPCKS0	高速时钟			
			2M	3.58M	4M/IHRC	8M
0	0	0	15.62K	27.96K	31.25K	62.5K
0	0	1	7.81K	13.98K	15.62K	31.25K
0	1	0	3.90K	6.99K	7.812K	15.62K
0	1	1	1.95K	3.49K	3.90K	7.81K
1	0	0	976Hz	1.748K	1.95K	3.90K
1	0	1	488Hz	874Hz	976Hz	1.95K
1	1	0	244Hz	437Hz	488Hz	976Hz
1	1	1	122Hz	218Hz	244Hz	488Hz

✷ **注：在一般应用时，PGIA Chopper 的工作时钟应该设置为 31.25KHz，但是在高速 32768 晶振时钟模式或者内部低速时钟模式下则应该设置为 250Hz。**

11.4.3AMPCHS-PGIA 通道选择寄存器

091H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
AMPCHS	-	-	-	-	-	CHS2	CHS1	CHS0
读/写	-	-	-	-	-	R/W	R/W	R/W
复位后	-	-	-	-	-	0	0	0

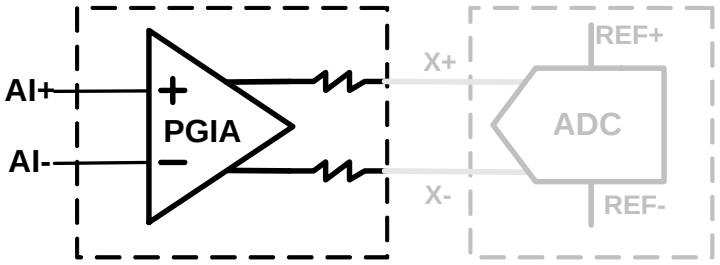
Bit[2:0] **CHS [2:0]**：PGIA 通道选择控制位。

PGIA 通道选择表：

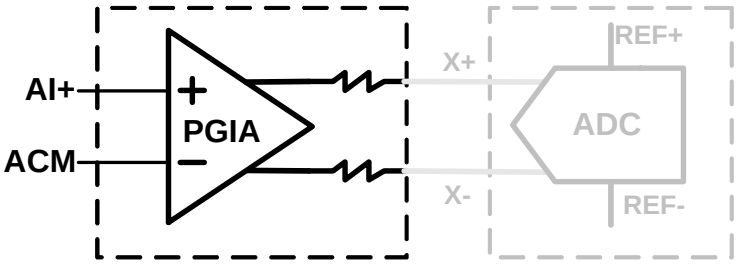
CHS [2:0]	选择通道	V (X+, X-)输出	输入信号的类型
000	AI+, AI-	V (AI+, AI-) × PGIA 增益	差分输入
001	AI+, ACM	V (AI+, ACM) × PGIA 增益	单端输入
010	AI-, ACM	V (AI-, ACM) × PGIA 增益	单端输入
011	ACM, ACM	V (ACM, ACM) × PGIA 增益	输入电路
100	保留	-	-
101	温度传感器	V (VTS, 0.6V) × 1	N/A
110	Voltage Detection	V (3/16VDD, 2/16VDD) × PGIA 增益 V (3/16VLDD, 2/16VLDD) × PGIA 增益	差分输入
110,111	保留	-	-

✷ **注 1：V（AI+, AI-）=（AI+电压 - AI-电压）**
✷ **注 2：V（AI-, ACM）=（AI-电压 – ACM 电压）**
✷ **注 3：输入短路模式仅用来测试 PGIA 的偏移量。**

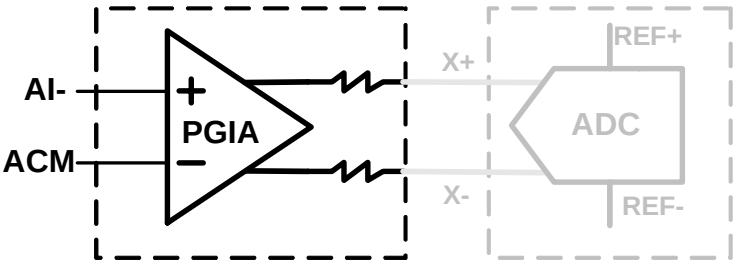
AMPCHS[2:0]=”000”



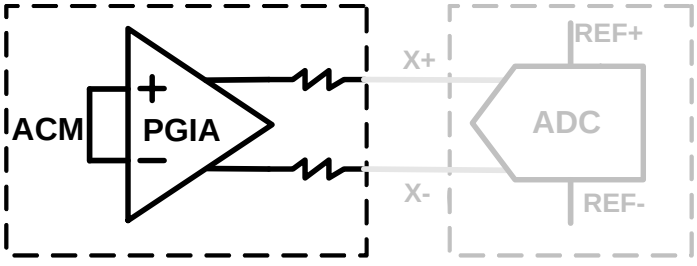
AMPCHS[2:0]=”001”



AMPCHS[2:0]=”010”



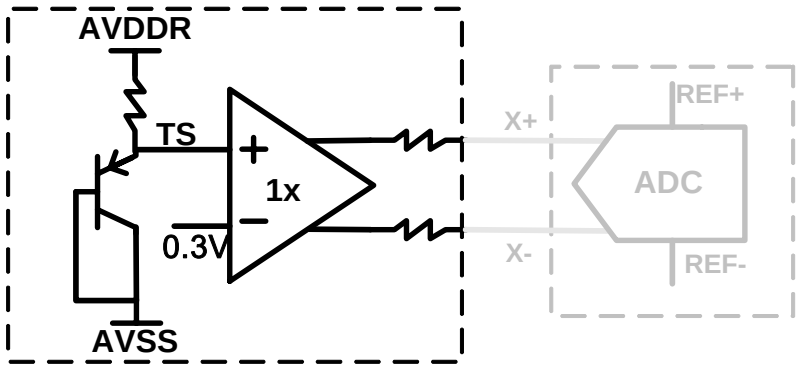
AMPCHS[2:0]=”011”



11.4.4温度传感器（TS）

在应用中，不同的环境温度会使传感器的特性也有所不同，为了得到不同的温度信息，SN8P1937 内置了一个温度传感器（TS）来测量工作环境温度。通过相对应的 PGIA 通道达到测量环境温度的目的。

AMPCHS [2:0] =”101”



✸ **注 1：当选择温度传感器时，PGIA 的增益要选择为 1x，否则会出错。**
✸ **注 2：这样设置后，X+的电压就是 V（TS），X-的电压是 0.3V。**
✸ **注 3：这里的温度传感器只是一个参考数据而不是真实的室温，在精确的应用中，请选用外部的温度传感器。**

在 25^{0}C 的环境下，V（TS）大约是 0.8V。如果温度上升 10C，V（TS）就会下降 15mV；相反，若温度下降 10C，V（TS）则会上升 15mV。

例：

温度	V(TS)	V(REF+,REF-)	ADC 输出
15	0.769V	0.6V	25601
25	0.754V	0.6V	24782
35	0.738V	0.6V	23908

通过 V（TS）ADC 输出，可以得到温度信息和系统补偿。

✸ **注 1：每颗单片机的 V（TS）和温度曲线都是有差异的，建议在应用温度传感器时进行室内温度校准。**
✸ **注 2：温度传感器的典型温度参数是 1.5mV/C。**

➢ **例：PGIA 设置（Fosc = 4M X'tal)）。**

```
@CPREG_Init:
            XB0BSET   FBGRENB       ; 使能 Band Gap 参考电压。

@ACM_Enable:
            XB0BCLR   FACMSEL       ; 设置 ACM 输出 0.4V。
            XB0BSET   FACMENB       ; 使能 ACM 电压输出。

@AVDDR_Enable:
            XB0BSET   FAVDDRENB     ; 设置 AVDDR 电压为 2.4V。

@AVE_Enable:
            XB0BCLR   FAVESEL       ; 设置 AVE+输出 1.5V。
            XB0BSET   FAVEENB       ; 使能 AVE+电压输出。
@PGIA_Init:
            MOV       A, #11110110B
            XB0MOV    AMPM, A       ; 使能 Band Gap，设置 FDS= 11、PGIA 增益 =128x。
            MOV       A, #00000000B
            XB0MOV    AMPCKS, A     ; 设置 AMPCKS = 000，PGIA 工作时钟 = 31.2K @ 4M X'tal。
            MOV       A, #00h
            XB0MOV    AMPCHS, A     ; 选择 PGIA 差分输入通道 = AI+，AI-。
@PGIA_Enable:
            XB0BSET   FAMPENB       ; 使能 PGIA 功能。
            …                       ; V（X+, X-）输出 = V（AI+, AI-）x 128。
```

注 1：在 PGIA 工作之前使能 AVDDR Regulator。
注 2：设置 PGIA 相关寄存器后再使能 PGIA 功能位。

➢ **例：设置 PGIA 通道。**

```
@PGIA_Init:
            MOV       A, #11110110B
            XB0MOV    AMPM, A       ; 使能 Band Gap，设置 FDS= 11、PGIA 增益 =128x。
            MOV       A, #00000000B
            XB0MOV    AMPCKS, A     ; 设置 AMPCKS = 000，PGIA 工作时钟 = 31.2K @ 4M X'tal。
            MOV       A, #00H
            XB0MOV    AMPCHS, A     ; 选择 PGIA 差分输入通道 = AI+, AI-。
@PGIA_Enable:
            XB0BSET   FAMPENB       ; 使能 PGIA 功能。
            …                       ; V（X+, X-）输出 = V（AI+, AI-）x 128。
@PGIA_single-end:
            MOV       A, #11110111B ; 切换 PGIA 通道时不需要禁止 PGIA。
            XB0MOV    AMPM, A       ; 使能 Band Gap 设置 FDS= 11，PGIA 增益=128x。
            MOV       A, #00000001B
            XB0MOV    AMPCHS, A     ; 选择 PGIA 作为差分输入通道。
            …                       ; V (X+, X-)输出 = V(AI+,ACM) x 128。

@PGIA_TS:
            MOV       A, #11111111B ; 切换 PGIA 信道时不用禁止 PGIA 功能。
            XB0MOV    AMPM, A       ; 使能 Band Gap，设置 FDS= 11、PGIA 增益 =1x。
            MOV       A, #00000101B
            XB0MOV    AMPCHS, A     ; 选择 PGIA 为温度传感器通道。
            …                       ; V（X+, X-）输出 = V（TS, 0.3）x 1。
```

11.5 16 位 ADC

11.5.1ADCM- ADC 模式寄存器

093H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCM	ADG2	ADG1	ADG0	RVS	IRVS	DTENB	DTSEL	ADCEN
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	0	0	0	1	1	0	0	0

Bit0 **ADCENB**：ADC 功能控制位。
0 = 禁止 16 位 ADC；
1 = 使能 16 位 ADC。

Bit1 **DTSEL**：电压侦测源选择控制位。
0 = 选择 VDD 作为电压侦测源；
1 = 选择 VLCD 作为电压侦测源。

Bit2 **DTENB**：侦测功能使能位。
0 = 从 X+、X-选择 ADC 为普通操作；
1 = 使能 ADC 为 VDD 或 VLCD 电压侦测功能。

Bit3 **IRVS**：内部参考电压选择位。
0 = 内部参考电压 V(REF+,REF-)为 AVE+ /5（当 AVE+ = 1.5V 时，V(REF+, REF-) = 0.3V）；
1 = 内部参考电压 V(REF+,REF-)为 AVE+ /2.5（当 AVE+ = 1.5V 时，V(REF+, REF-) = 0.6V）。

Bit4 **RVS**：ADC 参考电压选择控制位。
0 = 选择 ADC 参考电压源来自外部参考源 R+、R-；
1 = 选择 ADC 参考电压源来自内部参考源。

Bit[7:5] **ADG[2:0]**：ADC 增益选择表如下：

ADG[2:0]	ADC 增益
0xx	1x
100	2x
101	4x
110, 111	保留

ADC 参考电压配置表：

RVS	IRVS	DTENB	DTSEL	ADC 参考电压		AD 输入通道		备注
				REF+	REF-	ADCIN+	ADCIN-	
0	X	0	x	R+	R-	X+	X-	外部参考电压
1	0	0	x	0.6V	0.3V			V (X+, X-) < 0.3V, (AVE+=1.5V)
1	1	0	x	0.9V	0.3V			V (X+, X-) < 0.6V, (AVE+=1.5V)
0	X	1	0	R+	R-	VDD*3/16	VDD*2/16	ADC 输入 = 1/16 VDD，监控电池电压（AVE+=1.5V）
1	0	1	0	0.6V	0.3V			
1	1	1	0	0.9V	0.3V			
0	x	1	1	R+	R-	VLCD*3/16	VLCD*2/16	ADC 输入 = 1/16 VLCD，监控 VLCD
1	0	1	1	0.6V	0.3V			
1	1	1	1	0.9V	0.3V			

✷ **注 1：AD 转换的数据存放在 ADCDH 和 ADCDL 寄存器中，其中 ADCB15 是 ADC 数据的符号位。关于 AD 转换数据值的计算方法请参考如下公式：**
✷ **注 2：内部参考电压为 AVE+/X。**

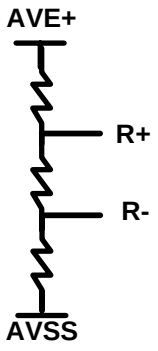
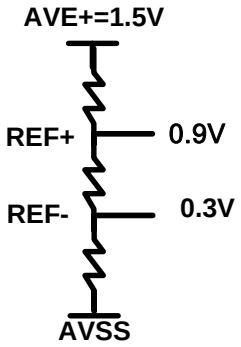
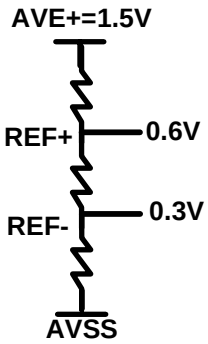
$$(ADCIN+) > (ADCIN-) \Rightarrow ADCConversionData = +\frac{[(\text{ADCIN+}) - (\text{ADCIN-})]}{(\text{REF+}) - (\text{REF-})} \times \text{ADC\_Gain} \times 32768 \times \left(\frac{\text{OSR} - 1}{\text{OSR}}\right)^2$$

$$(ADCIN+) < (ADCIN-) \Rightarrow ADCConversionData = (ADCIN+) > (ADCIN-) \Rightarrow ADCConversionData = -\frac{[(\text{ADCIN+}) - (\text{ADCIN-})]}{(\text{REF+}) - (\text{REF-})} \times \text{ADC\_Gain} \times 32768 \times \left(\frac{\text{OSR} - 1}{\text{OSR}}\right)^2$$

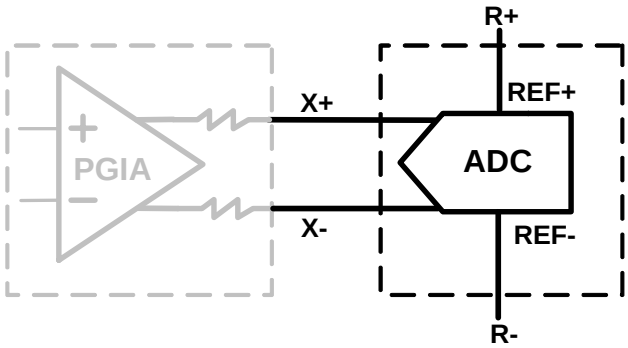
注 1：OSR 的值请参考 11.5.4 章节的 OSR 列表（ADCHPENB=0）。
注 2：例如 OSR[2:0]=111，OSR 的值为 4.96，ADC full counts = 32768x [(4096-1)/4096]^2 = 32752。
注 3：ADC full counts 与 OSR 设置的关系列表如下：

OSR[2:0]	ADC Full Counts	OSR[2:0]	ADC Full Counts
000	30752	100	32640
001	31752	101	32704
010	32258	110	32736
011	32513	111	32752

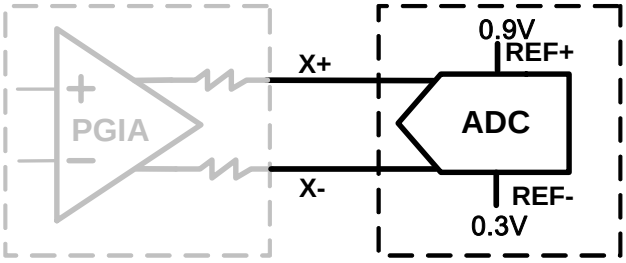
外部和内部参考电路图如下：

外部参考电路	内部参考电路	
RVS=0	RVS=1	
	IRVS=1, AVE+=1.5V	IRVS=0,AVE+=1.5V
		

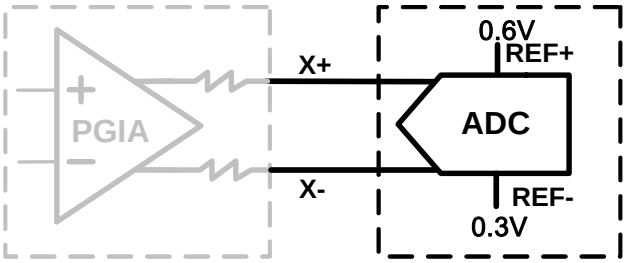
ADCM=#xxx0x0xxB，V(REF+, REF-) = V(R+, R-)，ADC 参考电压来自外部 R+、R-。



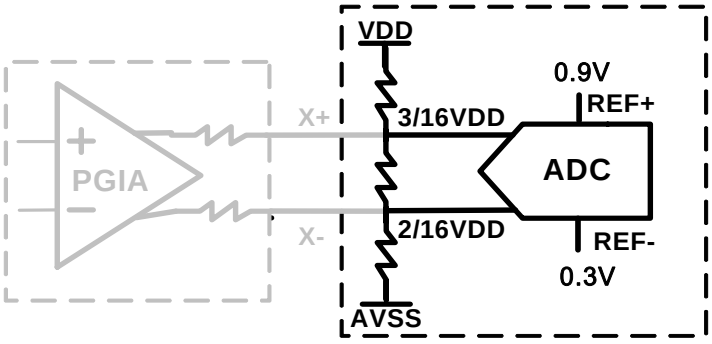
ADCM=#xxx1100xB，V(REF+, REF-) = V(0.9V, 0.3V) = 0.6V，ADC 参考电压来自内部 0.9V 和 0.3V。



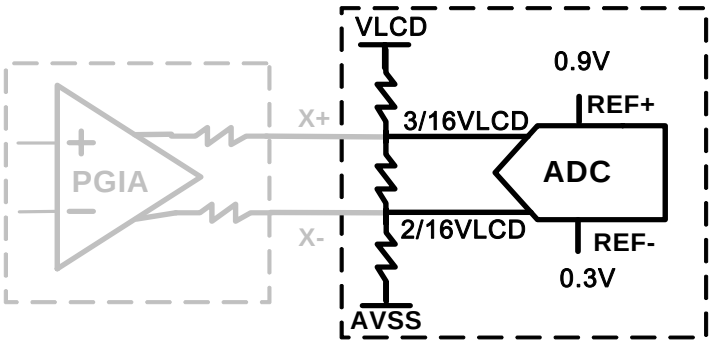
ADCM=#xxx1000xB，V(REF+, REF-) = V(0.6V, 0.3V)=0.3V，ADC 参考电压来自内部 0.6V 和 0.3V。



ADCM=#xxx1110xB，V(REF+, REF-) = V(0.9V, 0.3V)=0.6V，ADC 参考电压来自内部 0.9V 和 0.3V，ADC 输出电压测量结果。



ADCM=#xxx1111xB，V(REF+, REF-) = V(0.9V, 0.3V)=0.6V，ADC 参考电压来自内部 0.9V 和 0.3V，ADC 输出电压测量结果。



11.5.2ADCKS- ADC 时钟寄存器

094H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCKS	ADCKS7	ADCKS6	ADCKS5	ADCKS4	ADCKS3	ADCKS2	ADCKS1	ADCKS0
读/写	W	W	W	W	W	W	W	W
复位后	0	0	0	0	0	0	0	0

Bit[7:0] **ADCKS[7:0]：**寄存器设置 ADC 的工作时钟频率，建议值为 100KHz。

关于 ADCKS [7:0]寄存器在不同的 Fosc 频率下的值请参阅下表：

ADC 时钟 = (Fosc / (256-ADCKS [7:0]))/2

ADCKS [7:0]	FOSC	ADC 工作时钟频率
246	4M	(4M / 10)/2 = 200K
236	4M	(4M / 20)/2 = 100K
243	4M	(4M / 13)/2 = 154K
231	4M	(4M / 25)/2 = 80K

ADCKS [7:0]	FOSC	ADC 工作时钟频率
236	8M	(8M / 20)/2 = 200K
216	8M	(8M / 40)/2 = 100K
231	8M	(8M / 25)/2 = 160K
206	8M	(8M / 50)/2 = 80K

✷ **注：在一般应用中，ADC 的工作时钟频率设置为 100KHz。**

11.5.3ADC 数据寄存器

099H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCDH	ADCB15	ADCB14	ADCB13	ADCB12	ADCB11	ADCB10	ADCB8	ADCB9
读/写	R	R	R	R	R	R	R	R
复位后	0	0	0	0	0	0	0	0

098H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCDL	ADCB7	ADCB6	ADCB5	ADCB4	ADCB3	ADCB2	ADCB1	ADCB0
读/写	R	R	R	R	R	R	R	R
复位后	0	0	0	0	0	0	0	0

ADCDH [7:0]：输出 ADC 数据的高字节。
ADCDL [7:0]：输出 ADC 数据的低字节。

ADC 参考数据（十六进制）	十进制
0x7FF0	32752
…	…
0x4000	16384
…	…
0x1000	4096
…	…
0x0002	2
0x0001	1
0x0000	0
0xFFFF	-1
0xFFFE	-2
…	…
0xF000	-4096
…	…
0xC000	-16384
…	…
0x8010	-32752

- **注 1：ADCDL [7:0]和 ADCDH [7:0]都是只读寄存器。**
- **注 2：ADC 的数据存放在 ADCDH、ADCDL 寄存器中，其中 ADCB15 位是 ADC 数据的符号位。**
  ADCB15 = 0 表示数据为正值，ADCB15 = 1 表示数据为负值。
- **注 3：ADC 输出的最大正值是 7FFFH。**
- **注 4：ADC 输出的最小负值是 8000H。**
- **注 5：由于 ADC 的设计限制，ADC 的线形范围为+29491~-29491（十进制），故 ADC 的值必须在该范围内。**

11.5.4DFM-ADC 数字滤波模式寄存器

097H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DFM	OSR2	OSR1	OSR0	ADCHPENB	-	-	-	DRDY
读/写	R/W	R/W	R/W	R/W	-	-	-	R/W
复位后	0	0	0	1	-	-	-	0

Bit0 **DRDY**：ADC 数据就绪位。
1 = ADC 输出新的转换数据到 ADCDH 和 ADCDL；
0 = ADCDH 和 ADCDL 的转换数据还未就绪。

Bit4 **ADCHPENB**：ADC Chopper 控制位。
1 = 使能；
0 = 禁止。

Bit[7:5] **OSR[2:0]**：ADC Over 采样 rate 选择位。
OSR 列表：

OSR [2:0]	ADCHPENB = 1	ADCHPENB = 0
	OSR	
000	128	32
001	256	64
010	512	128
011	1024	256
100	2048	512
101	4096	1024
110	8192	2048
111	16384	4096

ADC 输出字 Rate = ADC 时钟 / OSR / ADC_Gain

- **例 1：ADC 时钟 = 100K，OSR = 16384，ADC_Gain = 1。（ADCHPENB = 1）**
  ADC 输出字 Rate = 100k/16384/1 = 6.1Hz.
- **例 2：ADC 时钟 = 100K，OSR = 16384，ADC_Gain = 2。（ADCHPENB = 1）**
  ADC 输出字 Rate = 100k/16384/2 = 3Hz
- **例 3：ADC 时钟 = 250K，OSR = 64，ADC_Gain = 2。（ADCHPENB = 0）**
  ADC 输出字 Rate = 250k/64/2 = 1.95 kHz （快速称重时 ADC 转换速率参考设置）。

- **注 1：调节 ADC 时钟（ADCKS）和 OSR 可以得到合适的 ADC 输出字 rate。**
- **注 2：在高分辨率应用中，建议设置 OSR 的最大值 16384。**
- **注 3：读取 ADC 数据后将 DRDY 位清零，否则会一直保持高电平。**
- **注 4：针对普通 ADC 应用设置 ADCHPENB = 1，针对 Fast ADC 应用设置 ADCHPENB = 0。**
- **注 5：Fast ADC 的噪音滤波分辨率降至 10 位。**

➢ **例：Regulator，PGIA 和 ADC 设置（Fosc = 4M X'tal）。**

```
@CPREG_Init:        XB0BSET     FBGRENB         ; 使能 Band Gap 参考电压。

@ACM_Enable:        XB0BCLR     FACMSEL         ; 设置 ACM 输出 0.4V。
                    XB0BSET     FACMENB         ; 使能 ACM 电压。

@AVE_Enable:        XB0BCLR     FAVESEL         ; 设置 AVE+输出 1.5V。
                    XB0BSET     FAVENB          ; 使能 AVE+电压。

@AVDDR_Enable:      XB0BSET     FAVDDRENB       ; 使能 AVDDR 电压为 2.4V。

@PGIA_Init:         MOV         A, #11110110B
                    XB0MOV      AMPM, A         ; 使能 Band Gap，设置 FDS=11，PGIA 增益=128。
                    MOV         A, #00000000B
                    XB0MOV      AMPCKS, A       ; 设置 AMPCKS = 000，PGIA 工作时钟为 31.25KHz。
                    MOV         A, #00h
                    XB0MOV      AMPCHS, A       ; 选择 PGIA 差分输入通道 = AI+, AI。
                    XB0BSET     FAMPENB         ; 使能 PGIA 功能。
                                                ; V (X+, X-)输出 = V (AI+, AI-) x 128

@ADC_Init:          MOV         A, #10011000B
                    XB0MOV      ADCM, A         ; ADC 参考电压为 0.6V，ADC_Gain = 2x。
                    MOV         A, #246
                    XB0MOV      ADCKS, A        ; 设置 ADCKS=246，ADC 工作时钟为 200K@4MHz。
                    MOV         A, #11110000B   ; 设置 ADC OSR = 16384。
                    XB0MOV      DFM, A          ; 设置 ADC 转换 rate =200k / 16384 / 2 = 6.1 Hz。
                    Call        Wait_300uS      ; 延时 300us 等待 Regulators 和模拟功能稳定。
                    XB0BSET     FADCENB         ; 使能 ADC 功能。

@ADC_Wait:          XB0BTS1     FDRDY           ; 检查 ADC 是否输出新的数据。
                    JMP         @ADC_Wait       ; 等待位 DRDY = 1.
                                                ; 输出 ADC 数据。
@ADC_Read:          XB0BCLR     FDRDY
                    XB0MOV      A, ADCDH
                    B0MOV       Data_H_Buf, A   ; 将 ADC 高字节数据存入数据缓存器。
                    XB0MOV      A, ADCDL
                    B0MOV       Data_L_Buf, A   ; 将 ADC 低字节数据存入数据缓存器。
```

✷ **注 1：请先设置 ADC 的相关寄存器，再使能 ADC 功能。**
✷ **注 2：使能 ADC 功能之前，请先设置模拟功能（Regulators，PGIA 和 ADC）并延时 300us 等待所有功能稳定。**

➢ **例：VDD/VLCD 电压检测。**

```
@CPREG_Init:        XB0BSET     FBGRENB         ; 使能 Band Gap 参考电压。

@ACM_Enable:        XB0BCLR     FACMSEL         ; 设置 ACM 输出 0.4V。
                    XB0BSET     FACMENB         ; 使能 ACM 电压。

@AVE_Enable:        XB0BCLR     FAVESEL         ; 设置 AVE+输出 1.5V。
                    XB0BSET     FAVENB          ; 使能 AVE+电压。

@AVDDR_Enable:      XB0BSET     FAVDDRENB       ; 使能 AVDDR 电压为 2.4V。

@PGIA_Init:         MOV         A, #11111110B
                    XB0MOV      AMPM, A         ; 使能 Band Gap，设置 FDS=11，PGIA 增益=128。
                    MOV         A, #00000000B
                    XB0MOV      AMPCKS, A       ; 设置 AMPCKS = 000，PGIA 工作时钟为 31.25KHz。
                    MOV         A, #00000110B
                    XB0MOV      AMPCHS, A       ; 选择 PGIA 电压检测通道 VDD 或 VLCD。
                    XB0BSET     FAMPENB         ; 使能 PGIA 功能。

@VDD_Detection:     MOV         A, #00011100B   ; ADC 内部参考电压，ADC 增益为 1，使能电压检测。
                    XBMOV       ADCM, A         ; 使能 VDD 检测功能。
```

```
                MOV         A, #0236
                XB0MOV      ADCKS, A        ; 设置 ADCKS=236，ADC 工作时钟为 100K @ 4MHz。
                MOV         A, #11110000B   ; 设置 ADC OSR = 16384。
                XB0MOV      DFM, A          ; 设置 ADC 转换 rate =100k / 16384 / 1 = 6.1 Hz。
                Call        Wait_300uS      ; 延时 300us 等待 Regulators 和模拟功能稳定。
                XB0BSET     FADCENB         ; 使能 ADC 功能。

@ADC_Wait:      XB0BTS1     FDRDY           ; 检查 ADC 是否输出新的数据。
                JMP         @ADC_Wait       ; 等待位 DRDY = 1。
@ADC_Read:      XB0BCLR     FDRDY           ; 输出 ADC 数据。
                XB0MOV      A, ADCDH
                B0MOV       Data_H_Buf, A   ; 将 ADC 高字节数据存入数据缓存器。
                XB0MOV      A, ADCDL
                B0MOV       Data_L_Buf, A   ; 将 ADC 低字节数据存入数据缓存器。
                …
                …
@VLCD_Detection: MOV        A, #00011110B   ;ADC 内部参考电压，ADC 增益为 1，使能电压检测。
                XBMOV       ADCM, A         ; 使能 VDD 检测功能。
                MOV         A, #0236
                XB0MOV      ADCKS, A        ; 设置 ADCKS=236，ADC 工作时钟为 100K @ 4MHz。
                MOV         A, #11110000B   ; 设置 ADC OSR = 16384。
                XB0MOV      DFM, A          ; 设置 ADC 转换 rate =100k / 16384 / 1 = 6.1 Hz。
                Call        Wait_300uS      ; 延时 300us 等待 Regulators 和模拟功能稳定。
                XB0BSET     FADCENB         ; 使能 ADC 功能。

@ADC_Wait:      XB0BTS1     FDRDY           ; 检查 ADC 是否输出新的数据。
                JMP         @ADC_Wait       ; 等待位 DRDY = 1。
@ADC_Read:      XB0BCLR     FDRDY           ; 输出 ADC 数据。
                XB0MOV      A, ADCDH
                B0MOV       Data_H_Buf, A   ; 将 ADC 高字节数据存入数据缓存器。
                XB0MOV      A, ADCDL
                B0MOV       Data_L_Buf, A   ; 将 ADC 低字节数据存入数据缓存器。
                …
                …
```

➢ **例：快速 ADC 转换速率设置。**

```
@ADC_Init:
                MOV         A, #10011000B   ;ADC 内部参考电压，ADC 增益为 2。
                XBMOV       ADCM, A
                MOV         A, #0248
                XB0MOV      ADCKS, A        ; 设置 ADCKS=248，ADC 工作时钟为 250K @ 4MHz。
                MOV         A, #00100000B   ; 设置 ADC OSR = 64，ADCHPENB =0。
                XB0MOV      DFM, A          ; 设置 ADC 转换 Rate =250k / 64 / 2 = 1.95k Hz。
                Call        Wait_300uS      ; 延时 300us 等待 Regulators 和模拟功能稳定。
                XB0BSET     FADCENB         ; 使能 ADC 功能。

@ADC_Wait:      XB0BTS1     FDRDY           ; 检查 ADC 是否输出新的数据。
                JMP         @ADC_Wait       ; 等待位 DRDY = 1。
@ADC_Read:
                XB0BCLR     FDRDY           ; 输出 ADC 数据。
                XB0MOV      A, ADCDH
                B0MOV       Data_H_Buf, A   ; 将 ADC 高字节数据存入数据缓存器。
                XB0MOV      A, ADCDL
                B0MOV       Data_L_Buf, A   ; 将 ADC 低字节数据存入数据缓存器。
                …
                …
                JMP         @ADC_Wait:      ; 应用中第二个 ADC 数据有效。
```

✷ **注 1：使能 ADC 功能之前先设置 ADC 相关寄存器。**
✷ **注 2：使能 ADC 功能之前，先设置模拟功能（Regulators，PGIA 和 ADC）并延时 300us 等待所有功能稳定。**
✷ **注 3：Fast ADC 转换应用时，第二个 ADC 数据有效。**
✷ **注 4：为增加 Fast ADC 转换精度，建议取几次 ADC 数据的平均值。**

11.5.5LBTM：电池低电压检测寄存器

SN8P1937 提供 2 种方式测量电源电压：一种是通过选择 ADC 参考电压，这种方法比较精确但是比较费时且比较复杂；另外一种是通过内置的电压比较器，电源电压经过外部分压电路连接到 P5.1，与 Band Gap 参考电压（1.2V）进行比较，比较结果在 LBTO 位。

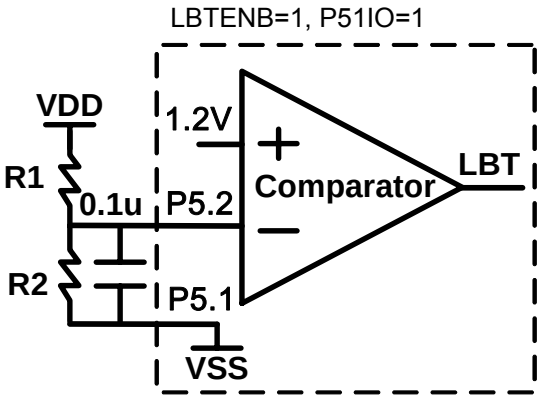
09AH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LBTM	-	-	-	-	-	LBTO	P51IO	LBTENB
读/写	-	-	-	-	-	R	R/W	R/W
复位后	-	-	-	-	-	0	0	0

Bit0 **LBTENB**：电池低电压检测模式控制位。
0 = 禁止电池低电压检测功能；
1 = 使能电池低电压检查功能。

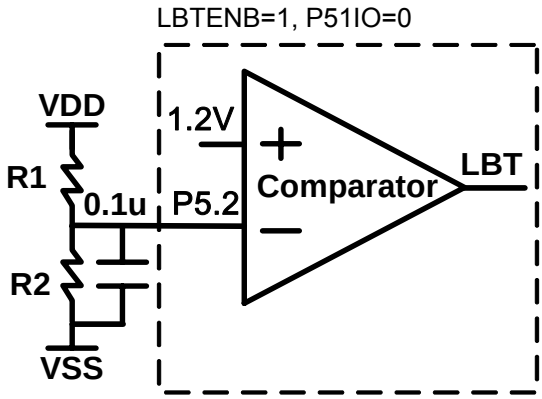
Bit1: **P51IO**：P5.1 输入/LBT 功能控制位。
0 = P51 为输入口；
1 = P51 为 LBT 功能。

Bit2: **LBTO**：电池低电压检测输出位。
0 = P5.2/LBT 电压高于 Band Gap 参考电压（1.2V）；
1 = P5.2/LBT 电压低于 Band Gap 参考电压（1.2V）。

下图是 LBT 应用的两种电路连接方式：一种使用 P5.2 和 P5.1，这样在睡眠模式下不会产生漏电流；另外一种只使用 P5.2，这种方式会在省电模式下产生一些漏电流，但是可以把 P5.1 作为输入口用。



P5.1 作为 LBT 功能时，睡眠模式下没有漏电流



P5.1 作为输入引脚，睡眠模式下产生漏电流

电池低电压	R1	R2	LBTO=1
2.4V	1MΩ	1MΩ	VDD<2.4V
3.6V	1.33MΩ	0.66MΩ	VDD<3.6V
4.8V	1.5MΩ	0.5MΩ	VDD<4.8V

✷ **注：在一段时间内（如 20ms 或更长的时间），建议连续多次判断（多于 10 次）LBTO 是否置 1 以保证电池电压确实为低电压。**

11.5.6模拟电路的设置和应用

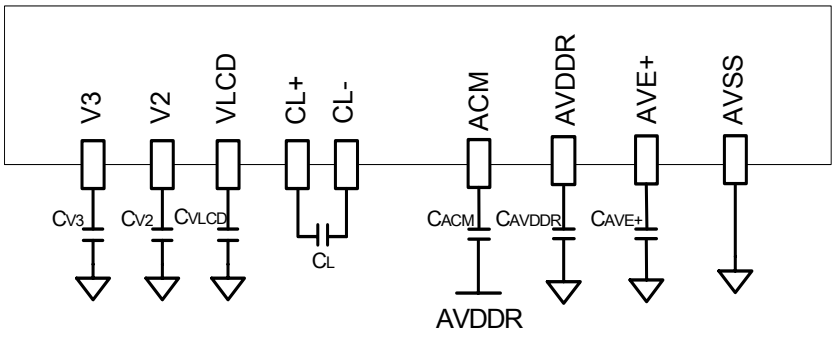
SN8P1937 主要应用于 DC 测量，如体重秤、压力测量等。下表列出了不同应用方案的建议，单片机的电源分别由 CR2032 电池、AA/AAA 干电池或者外部稳压电路供电。

电容列表：

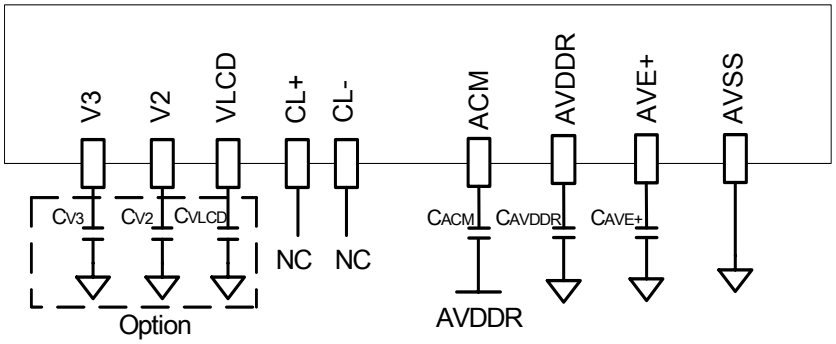
电源类型	AI+	AI-	R+/R-	ACM	AVDDR	AVE+	CL+/CL-	VDD (Pin17)	VDD (Pin28)
	CAI+	CAI-	CR	CACM	CAVDDR	CAVE+	CL	CAVDD	CDVDD
CR2032 (2.4~3V)	0.01uF	0.01uF	0.1uF	0.1uF	0.47uF	0.47uF	0.1uF	10uF	0.1uF
CR2032 ((4.4~6V))	0.1uF	0.1uF	0.1uF	0.1uF	0.47uF	0.47uF	0.1uF	10uF	0.1uF
AA/AAA Bat.(2.4~3V)	0.1uF	0.1uF	0.1uF	0.1uF	0.47uF	0.47uF	0.1uF	10uF	0.1uF
AA/AAA Bat.(4.4~6V)	0.1uF	0.1uF	0.1uF	0.1uF	0.47uF	0.47uF	0.1uF	10uF	0.1uF
External 5V Reg.	0.1uF	0.1uF	0.1uF	0.1uF	0.47uF	0.47uF	0.1uF	10uF	0.1uF

✸ 注：R 型 LCD 驱动模式下，CL 不需要连接到 MCU。

VDD=2.4V ~ 5.5V 模拟电路的连接（C 型 LCD 驱动）：

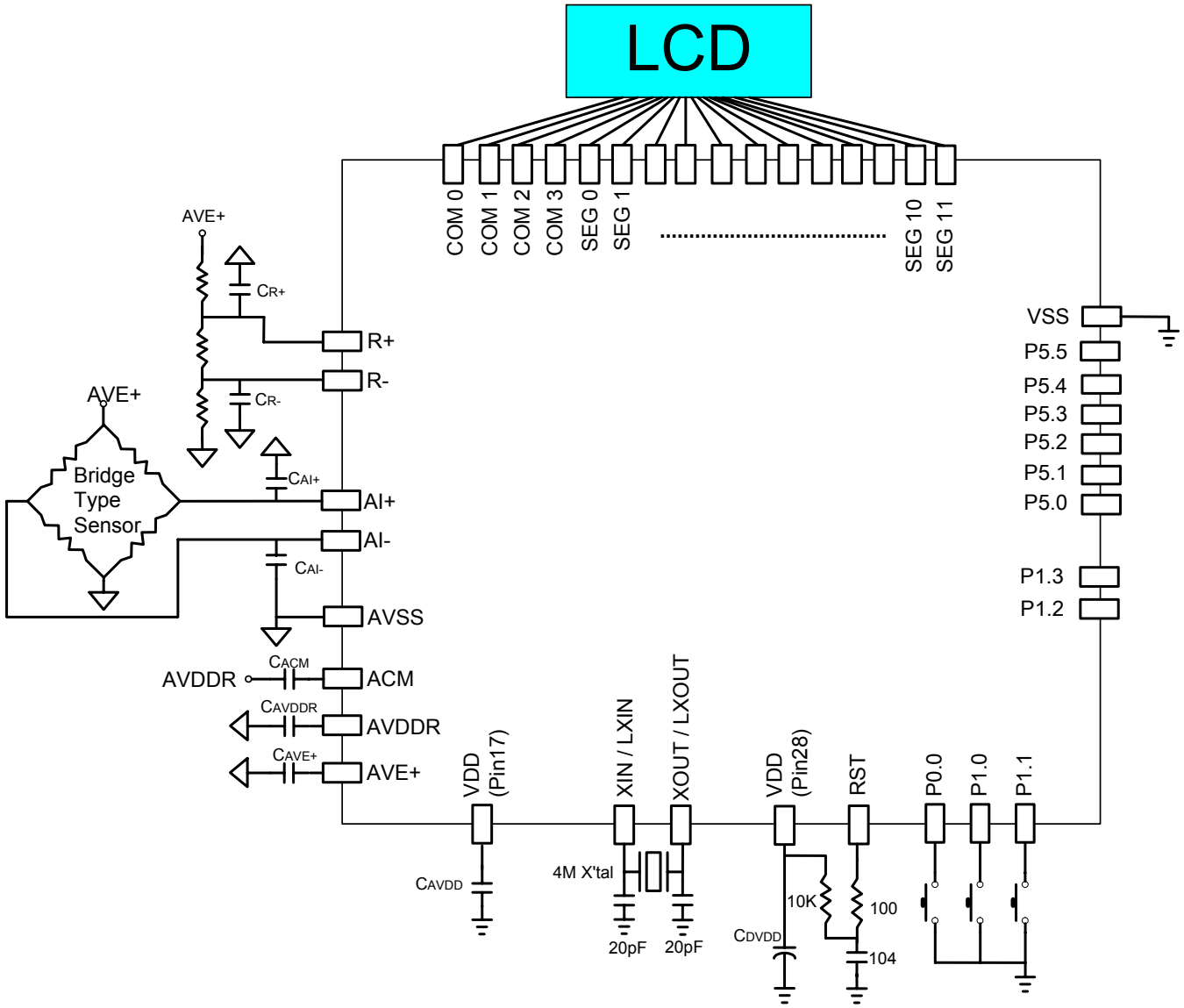


VDD=2.4V ~ 5.5V 模拟电路的连接（R 型 LCD 驱动）：



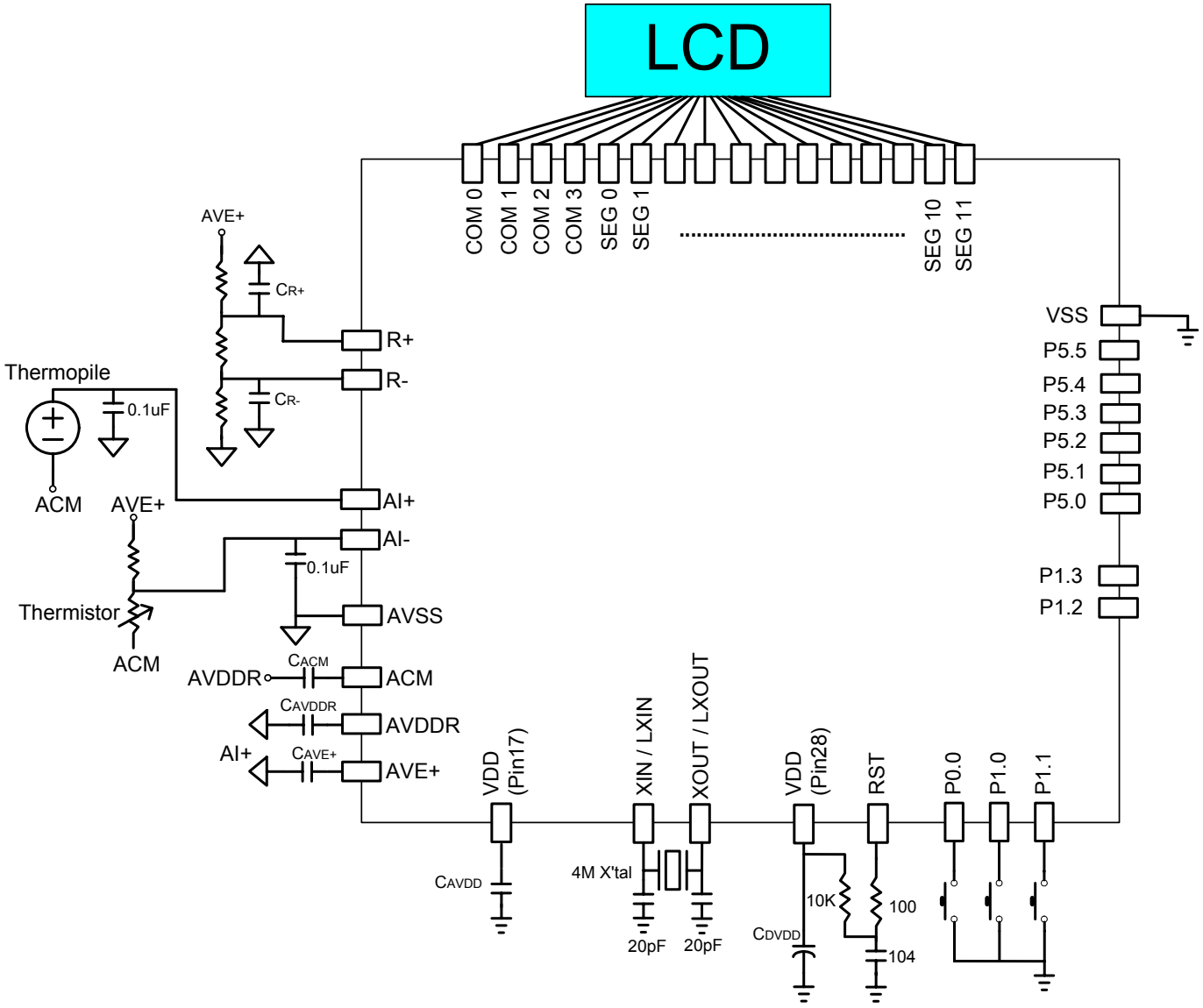
12 应用电路

12.1 电子秤（Load Cell）应用电路



✷ 注：电容的设置请参考 11.5.6 章节。

12.2 温度计应用电路



✷ **注：电容的设置请参考 11.5.6 章节。**

13 指令集

指令	指令格式		指令说明	C	DC	Z	周期
	MOV	A,M	A ← M	-	-	√	1
M	MOV	M,A	M ← A	-	-	-	1
O	B0MOV	A,M	A ← M (bnak 0)	-	-	√	1
V	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
E	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I，（M 指工作寄存器 R、Y、Z、RBANK 和 PFLAG 等。）	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	1
	B0XCH	A,M	A ←→M (bank 0)	-	-	-	1
	MOVC		R, A ← ROM [Y,Z]	-	-	-	2
	ADC	A,M	A ← A + M + C，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
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
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
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, if occur borrow, then C=0, else C=1	√	√	√	1
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
C	SUB	A,I	A ← A - I，如果产生借位，则 C=0，否则 C=1。	√	√	√	1
	DAA		将 ACC 中的数据由十六进制转换成十进制格式。	√	-	-	1
	AND	A,M	A ← A 与 M	-	-	√	1
L	AND	M,A	M ← A 与 M	-	-	√	1
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
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
	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
R	RRC	M	A ← RRC M	√	-	-	1
O	RRCM	M	M ← RRC M	√	-	-	1
C	RLC	M	A ← RLC M	√	-	-	1
E	RLCM	M	M ← RLC M	√	-	-	1
S	CLR	M	M ← 0	-	-	-	1
S	BCLR	M.b	M.b ← 0	-	-	-	1
	BSET	M.b	M.b ← 1	-	-	-	1
	B0BCLR	M.b	M(bank 0).b ← 0	-	-	-	1
	B0BSET	M.b	M(bank 0).b ← 1	-	-	-	1
	CMPRS	A,I	ZF,C ← A – I，如果 A = I，跳转到下一条指令。	√	-	√	1 + S
B	CMPRS	A,M	ZF,C ← A – M，如果 A = M，跳转到下一条指令。	√	-	√	1 + S
R	INCS	M	A ← M + 1，如果 A = 0，跳转到下一条指令。	-	-	-	1 + S
A	INCMS	M	M ← M + 1，如果 M = 0，跳转到下一条指令。	-	-	-	1 + S
N	DECS	M	A ← M – 1，如果 A = 0，跳转到下一条指令。	-	-	-	1 + S
C	DECMS	M	M ← M – 1，如果 M = 0，跳转到下一条指令。	-	-	-	1 + S
H	BTS0	M.b	如果 M.b = 0，跳转到下一条指令。	-	-	-	1 + S
	BTS1	M.b	如果 M.b = 1，跳转到下一条指令。	-	-	-	1 + S
	B0BTS0	M.b	如果 M(bank 0).b = 0，跳转到下一条指令。	-	-	-	1 + S
	B0BTS1	M.b	如果 M(bank 0).b = 1，跳转到下一条指令。	-	-	-	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		中断程序跳出指令，PC ← Stack，使能全局中断。	-	-	-	2
S	NOP		空指令。	-	-	-	1
C							

注： 条件跳转指令中，满足条件则 S=1，否则 S=0。

14 开发工具

14.1 开发工具版本

14.1.1在线仿真器 ICE

SN8ICE 1K（S8KD-2）：支持 SN8P1937 所有功能的仿真。

SN8ICE1K ICE 仿真时需注意：
ICE 的工作电压：3.0V~5.0V。
5V 工作电压的最大仿真速率建议为：4 MIPS（如 16MHZ 晶振时 Fcpu = Fhosc/4）。
使用 SN8P1937 EV-KIT 仿真模拟功能。

注：SN8ICE2K 不支持 SN8P1937 系列的仿真。

14.1.2OTP 烧录器

MPIII Writer：支持 SN8P1937 大批量的脱机/联机烧录。

14.1.3集成开发环境（IDE）

SONiX 8 位单片机的集成开发环境包括编译器、ICE 调试器和 OTP 的烧录软件。
SN8ICE 1K：SN8IDE 1.99Z05 或更新的版本。
MPIII Writer：SN8IDE 1.99Z05 或更新的版本。
M2IDE V1.2X 不支持 SN8P1937 的编译。

14.2 OTP 烧录引脚和转接板配置

14.2.1烧录引脚配置和烧录转接板引脚配置

JP1/JP2 引脚配置

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

JP1 连接 MP 转接板
JP2 连接多于 48PIN 或 DICE 烧录转接板

JP3 引脚配置

DIP1	1	48	DIP48
DIP2	2	47	DIP47
DIP3	3	46	DIP46
DIP4	4	45	DIP45
DIP5	5	44	DIP44
DIP6	6	43	DIP43
DIP7	7	42	DIP42
DIP8	8	41	DIP41
DIP9	9	40	DIP40
DIP10	10	39	DIP39
DIP11	11	38	DIP38
DIP12	12	37	DIP38
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

JP3 连接 MP 烧录转接板

14.2.2SN8P1937 烧录引脚配置

SN8P1937 烧录信息			
单片机名称		SN8P1937	
MPIII Writer		OTP IC / JP3 引脚配置	
引脚编号	引脚名称	引脚编号	引脚名称
1	VDD	17,28	VDD
2	GND	13,25,48	VSS
3	CLK	31	P1.0
4	CE	-	-
5	PGM	32	P1.1
6	OE	39	P1.2
7	D1	-	-
8	D0	-	-
9	D3	-	-
10	D2	-	-
11	D5	-	-
12	D4	-	-
13	D7	-	-
14	D6	-	-
15	VDD	17,28	VDD
16	VPP	29	RST
17	HLS	-	-
18	RST	-	-
19	-	-	-
20	ALSB/PDB	40	P1.3

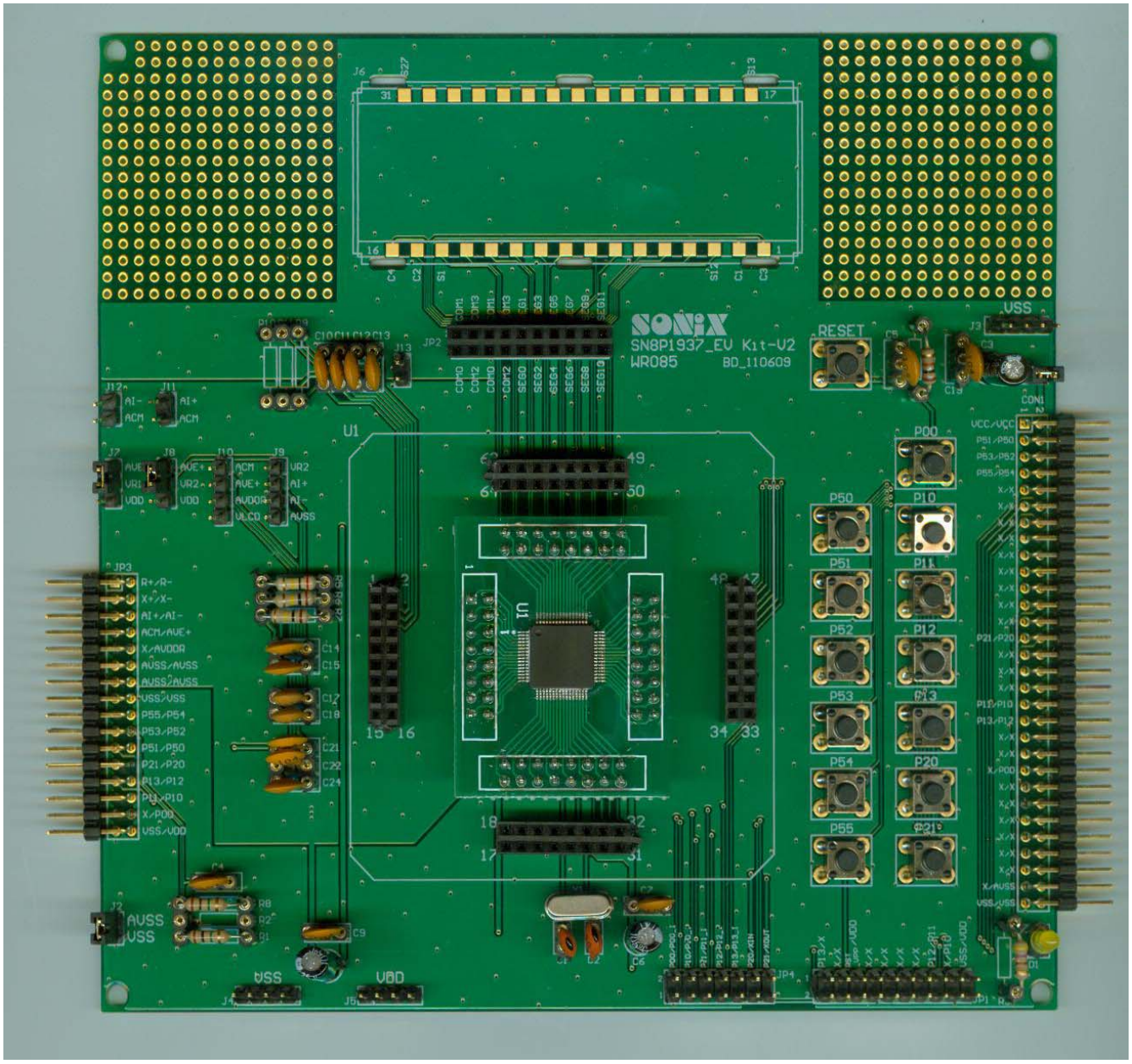
14.3 SN8P1937 EV-KIT

14.3.1简述

Sonix 提供一套完整的 SN8P1937 EV-Kit，包括 ICE（S8KD）、SN8P1937 EV Board，SONiX 编译器。用户可以在电脑上编写程序，之后通过软件或者 ICE 进行模拟仿真。一方面，执行程序时可以监控 RAM 的状态，用户还可以使用不同的功能如：断点、单步操作等，可以更方便的调试程序。而且，系统还内置 5V 电源。

14.3.2PCB 说明

Sonix 提供 SN8P1937 EV board 来仿真所有的功能，如下图所示：



SN8P1937 EV board

14.3.3EV BOARD 设置

CON1：连接到 ICE。
J1：置于短路状态，则由 ICE 提供电源；置于开路状态，则使用其他电源。
D1：电源指示灯。
RESET：复位开关。
JP1：OTP 烧录引脚。
JP2：LCD PORT
JP3：连接用户目标板。
JP4：当 ICE 和 EV Board 分开工作时，须将 JP4 设为短路状态，否则相关 IO 口不工作。
J2：AVSS 和 VSS 短路跳线引脚。
J6：LCD COM/SEG 连接引脚。
J7：从 VDD 或 AVE+选择外部参考电压 V(R+, R-)。
J8：从 VDD 或 AVE+选择 Load cell 电源。
J9：模拟差分(AI+, AI-)输入引脚。
J10：模拟电压输出引脚（ACM/AVE+/AVDDR/VLCD）。
J11/J12：模拟单端 AI+/ACM 和 AI-/ACM 输入引脚。
R5/R6/R7：ADC 外部参考电压分压电阻。
R1/R2/R8：LBT 功能电阻。
R9/R10/R11：R 型 LCD 外部分压电阻，用户需在 VLCD/V3/V2/V1 连接电阻以获得更大驱动电流。
C10/C11/C12/C13：C 型 LCD 外部电容。

	EV board 连接 ICE	EV board 连接 WRITER	EV board
JP1	不连接	烧录 SN8P1937	不连接
JP2(LCD)	无功能	-	根据实际情况设置
JP3	根据实际情况设置	不连接	根据实际情况设置
JP4	都不连接	都不连接	都连接
J6/7/8/9/10/11	根据实际情况设置	-	根据实际情况设置
J1	根据实际情况设置	短路	短路
J2	短路	短路	短路
U1	SN8P1937 EV-Link IC	空白 SN8P1937 IC，准备烧录	SN8P1937 IC，已经烧录系统应用程序
R1/R2/R8(LBT)	根据实际情况设置	-	根据实际情况设置
C10~C13	-	-	C 型 LCD 驱动

14.3.4SN8P1937 EV BOARD 与 ICE 的连接

1. EV Board 上的 U1 连接烧录程序的 IC Board。
2. 由 ICE 提供电源时，J1 必须短路。
3. 连接到开发系统时，JP4 必须开路，否则系统不能复位。
4. EV Board 和 ICE 必须使用通用的振荡器。
5. 用 EV_Board 仿真和开发时，J2 需要短路（数字 GND 和模拟 GND 短接）。
6. R1/R2/R8 为 LBT 功能电阻。
7. J7：ADC 外部参考电压源。.
8. J8：Load cell 电压源。

14.3.5STAND ALONG EV BOARD

1. EV board 和 ICE 分隔开来。
2. EV Board U1 须贴上已经烧录好应用程序的 SN8P1937 芯片。
3. 当使用独立的 EV Board 做仿真开发时，JP4 应该设为短路状态，否则相关 IO 口不能正常工作。
4. 确认 J2 的连接状态。J2 应该设置为短路状态（数字 GND 和模拟 GND 短接）。
5. C 型 LCD 驱动需要外接电容 C10~C13。

14.3.6SONIX 编译器

SONIX 提供 SN8ASM 编译器进行程序的编译，及 SN8P1937 相关软件的开发。将 SONIX ICE 和 SN8P1937 EV Board 结合起来进行开发，可以节省成本和时间。

14.3.7系统需求

操作系统：

SONIX 编译软件支持 WIN95、WIN98、WINME、WIN2000 和 WINXP 操作系统。必须在 WIN2000 和 WINXP 下安装驱动。

文件说明：

SN8IDE_xxxx.EXE	编译器的安装软件包，xxx 指版本号；
SN8ASMxxxx.EXE	编译器的应用程序，xxx 指版本号；
MACRO1.H	宏指令集；
MACRO2.H	宏指令集；
MACRO3.H	宏指令集；
SN8P1937.INC	定义 SN8P1937 所有功能；
1937Ev.H	SN8P1937 EV Kit 仿真代码的宏定义和常数；
1937Ev.ASM	SN8P1937 EV Kit 子程序，用户必须将此子程序包含在 EV Kit 中；
1937_EV_Demo.ASM	SN8P1937 Demo 程序。

14.3.8软件安装注意事项

1. 检查 SN8P1937.INC 是否包含在 SONIX 编译软件的指定文件夹中（默认路径： C:\Sonix\Sn8IDE_xxxx\use_inc2）；
2. 检查主程序中是否包含 1937Ev.H 和 1937Ev.ASM，详细指令请查阅 1937_TEMPLATE.ASM；
3. 主程序的第一行为：

```
ICE_Mode EQU 1
CHIP SN8P1937；
```

4. 当进行 OTP 烧录时，用户必须将主程序中第一行设定为实际芯片烧录，如下所示：

```
ICE_Mode EQU 0
CHIP SN8P1937。
```

警告：

1. 当监控 ICE 上 XB0MOV 的特殊寄存器时，建议用户将 XB0MOV 拷贝到用户的 RAM 区域中。
2. 在前面的章节没有进行说明，则不要使用特殊指令表（如 XB0MOV），以避免出现不可预料的错误。

14.3.9程序架构

```
;*****************************************************************************
; FILENAME : 1937_Demo.ASM
; AUTHOR : SONiX
; PURPOSE : Demo Code for SN8P1937
;*****************************************************************************
;* (c) Copyright 2009, SONiX TECHNOLOGY CO., LTD.
;*****************************************************************************
      ICE_Mode EQU 1              ; 1 for ICE , 0 for real chip
      ;ICE_Mode EQU 0
CHIP SN8P1937                     ; Select the CHIP
;-----------------------------------------------------------------------------
; Include Files
;-----------------------------------------------------------------------------
.nolist                           ; do not list the macro file
      INCLUDESTD MACRO1.H
      INCLUDESTD MACRO2.H
      INCLUDESTD MACRO3.H
      INCLUDESTD 1937Ev.h         ; for ICE linking emulation board
.list                             ; Enable the listing function
;-----------------------------------------------------------------------------
; Constants Definition
;-----------------------------------------------------------------------------
;     ONE EQU 1
;-----------------------------------------------------------------------------
; Variables Definition
;-----------------------------------------------------------------------------
.DATA
      org 0h                      ; Bank 0 data section start from RAM address 0x000
      Wk00B0      DS 1            ;Temporary buffer for main loop
      Iwk00B0     DS 1            ;Temporary buffer for ISR
      AccBuf   DS 1               ; Accumulater buffer
      PflagBuf DS 1               ;PFLAG buffer
;-----------------------------------------------------------------------------
; Bit Flag Definition
;-----------------------------------------------------------------------------
Wk00B0_0    EQU     Wk00B0.0      ;Bit 0 of Wk00B0
Iwk00B0_1   EQU     Iwk00B0.1     ;Bit 1 of Iwk00
;-----------------------------------------------------------------------------
; Code section
;-----------------------------------------------------------------------------
.CODE
      ORG         0               ;Code section start
      jmp         Reset           ;Reset vector
                                  ;Address 4 to 7 are reserved
      ORG         8
      Jmp         Isr             ;Interrupt vector
      ORG         10h
;-----------------------------------------------------------------------------
; Program reset section
;-----------------------------------------------------------------------------
Reset:
      mov         A,#07Fh         ;Initial stack pointer and
      b0mov       STKP,A          ;disable global interrupt
      b0mov       PFLAG,#00h      ;pflag = x,x,x,x,x,c,dc,z
      b0mov       RBANK,#00h      ;Set initial RAM bank in bank 0
      call        ClrRAM          ;Clear RAM
      call        SysInit         ;System initial
;-----------------------------------------------------------------------------
      INIT_1937Ev                 ; for ICE linking emulation board
;-----------------------------------------------------------------------------
      b0bclr   FGIE               ;Enable global interrupt
;-----------------------------------------------------------------------------
```

```
; Main routine
;--------------------------------------------------------------------------
Main:
        b0bset      FWDRST          ;Clear watchdog timer
        mov         a, #11010000B;
        XB0MOV      CPM,a           ;using XB0MOV command for CPM setting
        Call        @Delay_300us    ;Delay 300us for Analog voltage stable.
        jmp         Main
;--------------------------------------------------------------------------
INCLUDESTD 1937Ev.asm               ; SN8P1937 Ev. Kit interface code
;--------------------------------------------------------------------------
ENDP
```

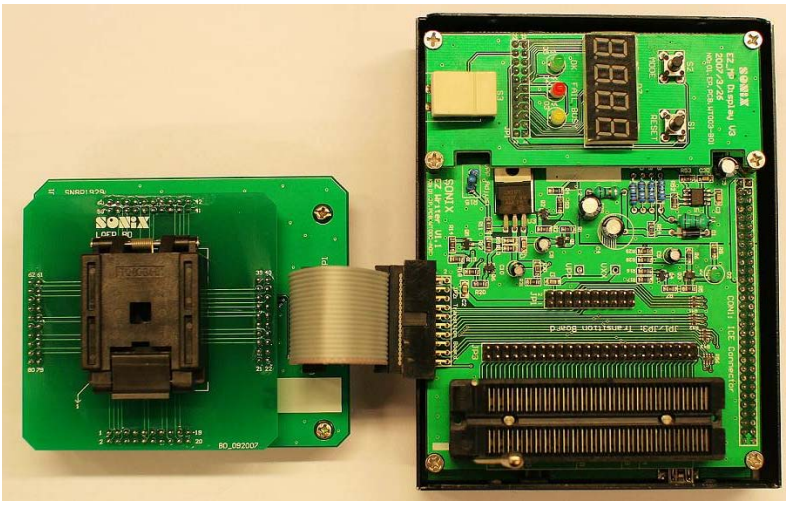
Please be aware of the position of the listed file names marked in red.

14.3.10 OTP 烧录步骤

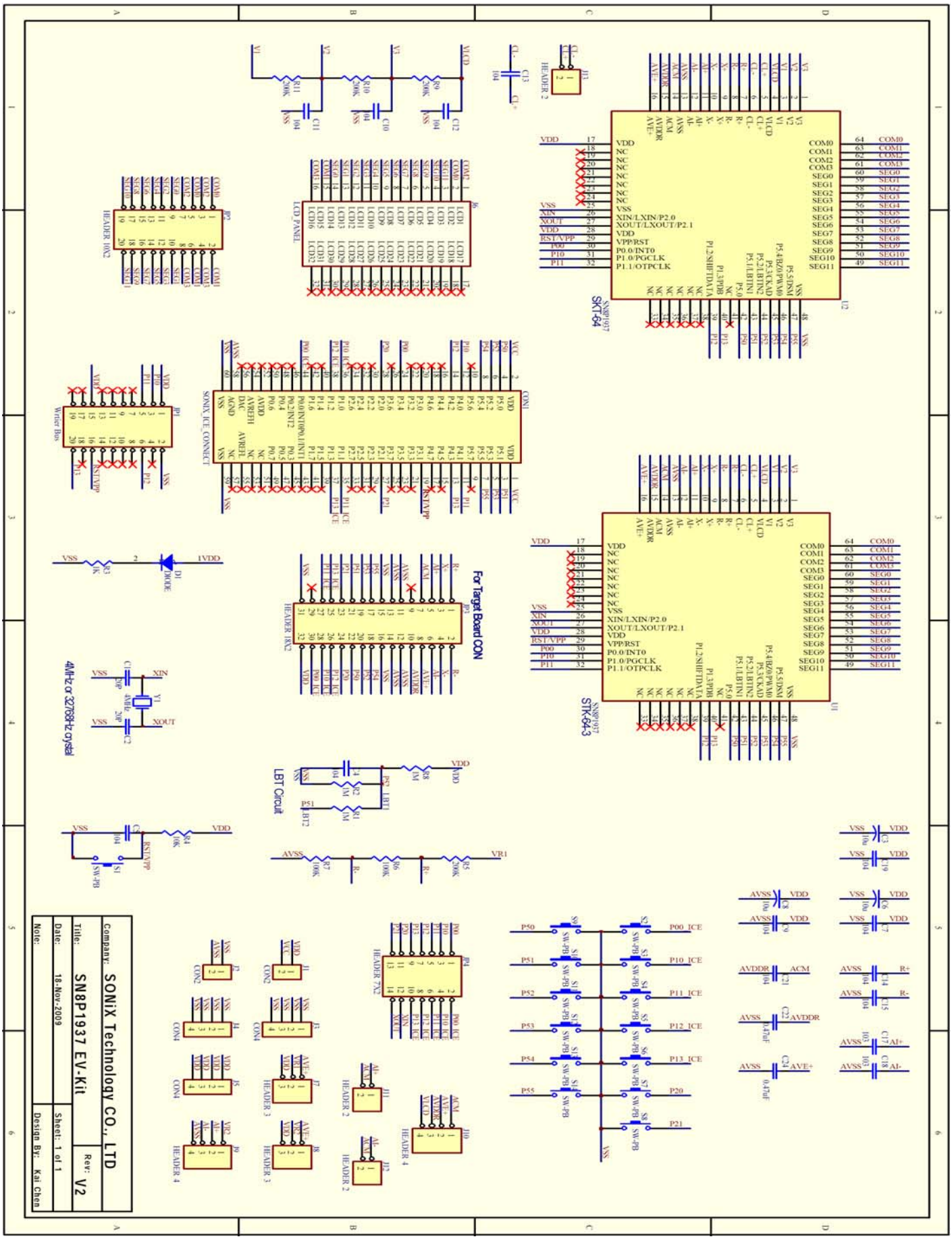
使用 SN8P1937 EV-board 烧录的步骤如下：

1. 将 EV board 和 ICE 分隔开来；
2. 在 U1 处放入空白的 OTP 芯片；
3. JP1 连接到 MPIII writer；
4. 确认 J2 的连接状态，要求 J2 处于短接状态。

14.3.11 SN8P1937 烧录转接板与 MPIII WRITER 的连接



14.3.12 附录 A：EV-KIT 电路图



15 电气特性

15.1 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)..... 0°C ~ + 70°C
Storage ambient temperature (TSTOR)..... –40°C ~ + 125°C

15.2 电气特性

(All of voltages refer to VSS, VDD = 5.0V,FOSC = 4MHz,Fcpu=1MHZ, ambient temperature is 25°C unless otherwise note.)

PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	Vdd	Normal mode, Vpp = Vdd		2.4	5.0	5.5	V
RAM Data Retention voltage	Vdr			-	1.5	-	V
VDD rise rate	VPOR	VDD rise rate to ensure power-on reset		0.05	-	-	V/ms
Input Low Voltage	ViL1	All input pins		Vss	-	0.3Vdd	V
	ViL2	Reset pin		Vss	-	0.2Vdd	V
Input High Voltage	ViH1	All input pins		0.7Vdd	-	Vdd	V
	ViH2	Reset pin		0.9Vdd	-	Vdd	V
Reset pin leakage current	Ilekg	Vin = Vdd		-	-	5	uA
I/O port pull-up resistor	Rup	Vin = Vss , Vdd = 3V		100	200	300	KΩ
		Vin = Vss , Vdd = 5V		50	100	180	KΩ
I/O port input leakage current	Ilekg	Pull-up resistor disable, Vin = Vdd		-	-	2	uA
I/O Port source current	IoH	Vop = Vdd - 0.5V		8	12	-	mA
sink current	IoL	Vop = Vss + 0.5V		8	15	-	
INTn trigger pulse width	Tint0	INT0 interrupt request pulse width		2/fcpu	-	-	cycle
Supply Current	Idd1	normal Mode (Low Power Disable, Analog Parts OFF)	Vdd= 5V 4MHz	-	1.5	3	mA
			Vdd= 5V IHRC	-	1.5	3	mA
			Vdd= 3V 4MHz	-	0.6	1.2	mA
			Vdd= 3V IHRC	-	0.8	1.2	mA
	Idd2	normal Mode (Low Power Enable, Analog Parts OFF)	Vdd= 5V 4MHz	-	1	2	mA
			Vdd= 5V IHRC	-	1	2	mA
			Vdd= 3V 4MHz	-	0.5	1	mA
			Vdd= 3V IHRC	-	0.7	1.4	mA
	Idd3	normal Mode (Low Power Disable, Analog Parts ON)	Vdd= 5V 4MHz	-	2	4	mA
			Vdd= 5V IHRC	-	2	4	mA
			Vdd= 3V 4MHz	-	1.2	2.4	mA
			Vdd= 3V IHRC	-	1.2	2.4	mA
	Idd4	normal Mode (Low Power Enable, Analog Parts ON)	Vdd= 5V 4MHz	-	1.6	3.2	mA
			Vdd= 5V IHRC		1.6	3.2	mA
			Vdd= 3V 4MHz	-	1.2	2.4	mA
			Vdd= 3V IHRC		1.2	2.4	mA
	Idd5	Slow mode (Stop High Clock, LCD OFF)	Vdd= 5V Ext.32768Hz	-	20	40	uA
			Vdd= 5V	-	12	24	uA
			Vdd= 3V Ext.32768Hz	-	7	14	uA
			Vdd= 3V	-	4	8	uA
	Idd6	Slow mode (Stop High Clock, R-LCD ON 200K)	Vdd= 5V Ext.32768Hz		28	56	uA
			Vdd= 5V		15	30	uA
			Vdd= 3V Ext.32768Hz	-	16	32	uA
			Vdd= 3V	-	12	24	uA
	Idd7	Slow mode (Stop High Clock, C-LCD) CP clock =32k , LCD no loading	Vdd= 5V Ext.32768Hz	-	45	90	uA
			Vdd= 5V	-	40	80	uA
			Vdd= 3V Ext.32768Hz	-	36	72	uA
			Vdd= 3V	-	28	56	uA
	Idd8	Green mode *Stop High Clock *LCD OFF	Vdd= 5V Ext.32768Hz	-	15	30	uA
			Vdd= 5V	-	6	12	uA
			Vdd= 3V Ext.32768Hz	-	5	10	uA
			Vdd= 3V	-	2	4	uA
	Idd9	Green mode *Stop High Clock *R-Type LCD ON 200K	Vdd= 5V Ext.32768Hz	-	25	50	uA
			Vdd= 5V	-	15	30	uA
			Vdd= 3V Ext.32768Hz	-	10	10	uA
			Vdd= 3V	-	7	14	uA

	Idd10	Green mode *Stop High Clock *C-Type LCD ON, BGM = 0. *LCDCPK=32k,no LCD loading, C+-= 0.1uF	Vdd= 3V Ext.32768Hz	-	35	70	uA
			Vdd= 3V	-	30	30	uA
	Idd11	Green mode *High Clock Non-stop *C-Type LCD ON, BGM = 1. *LCDCPK=32k,no LCD loading, C+-= 0.1u	Vdd= 3V Ext.32768Hz	-	200	400	uA
			Vdd= 3V	-	200	400	uA
	Idd13	Sleep Mode	Vdd= 5V	-	1	2	uA
			Vdd= 3V	-	0.7	1.5	uA
LVD Detect Level	VLVD	Internal POR detect level	0℃~70℃	1.7	2.07	2.3	V
Internal High Clock Freq.	FIHRC	Internal High RC Oscillator Frequency		14	16	18	MHz

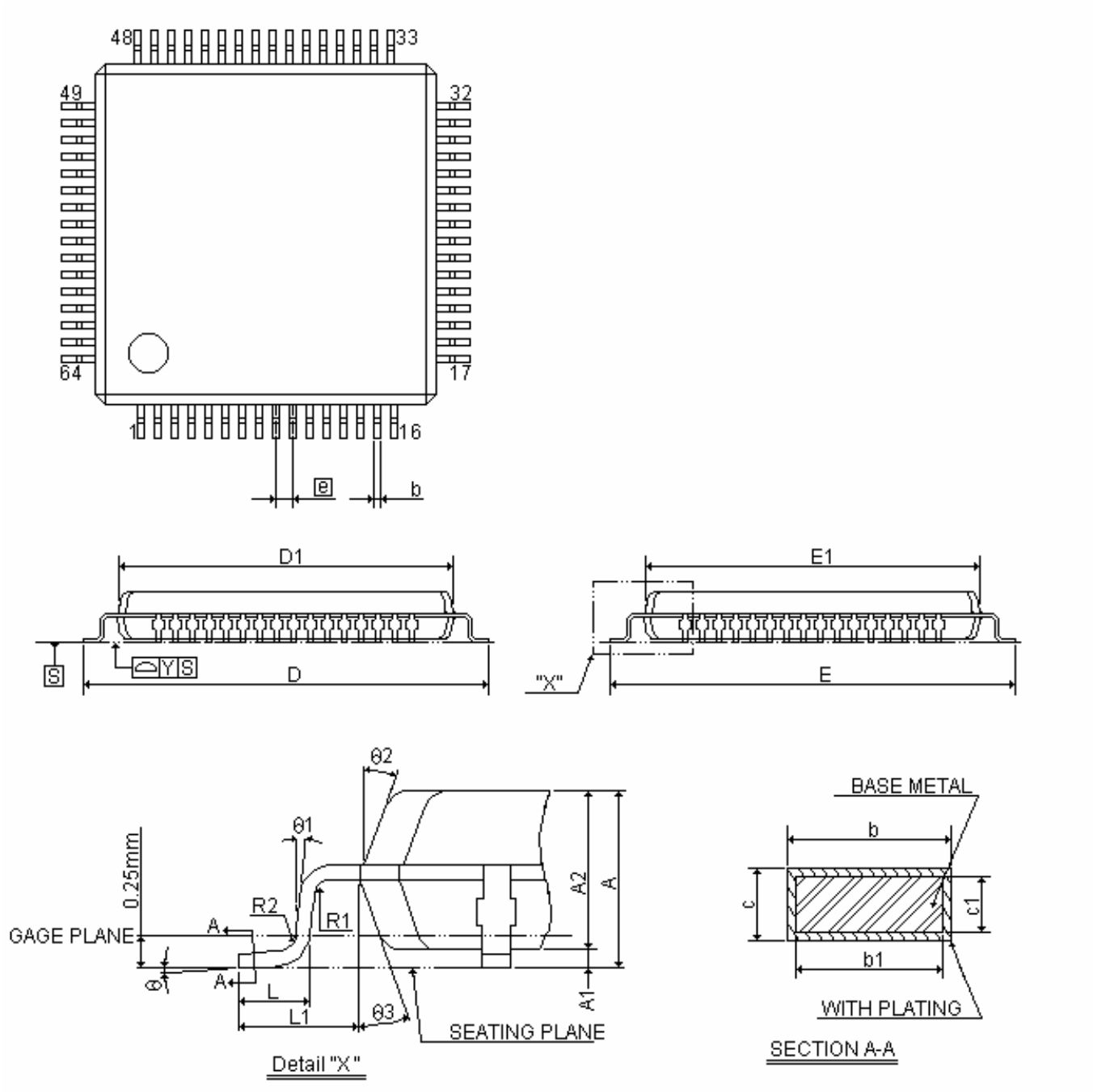
Note: Analog Parts including Regulator, PGIA and ADC.

(All of voltages refer to Vdd=3V FOSC = 4MHz, ambient temperature is 25°C unless otherwise note.)

PARAMETER	SYM.	DESCRIPTION	MIN.	TYP.	MAX.	UNIT
Analog to Digital Converter						
Operating current	IDD_ADC	Run mode @ 2.4V		250	300	uA
Power down current	IPDN	Stop mode @ 2.4V		0.1	1	μA
Conversion rate	FSMP	ENOB 14-bit			25	Sps
		ENOB 10-bit			1000	Sps
Reference Voltage Input Voltage	Vref	R+, R- Input Range (External Ref.)	0.3		0.9	V
		R+, R- Input Range (Internal Ref.)	0.3		0.9	V
Differential non-linearity	DNL	ADC range ± 29491		±0.5	±0.5	LSB
Integral non-linearity	INL	ADC range ± 29491		±1	±4	LSB
No missing code	NMC	ADC range ± 29491	16			bit
Noise free code	NFC	ADC range ± 29491		14	16	bit
Effective number of bits	ENOB	ADC range ± 29491		14	16	bit
ADC Input range	VAIN		0.4		1.05	V
Temperature Sensor inaccuracy	ETS	Inaccuracy range vs. real Temp.		±8		℃
PGIA						
Current consumption	IDD_PGIA	Run mode @ 2.4V		200	250	uA
Power down current	IPDN	Stop mode @ 2.4V			0.1	uA
Input offset voltage	Vos			25	50	uV
Bandwidth	BW				100	Hz
PGIA Gain Range (Gain=128x)	GR	VDD = 2.4V	110	128	150	
PGIA Input Range	Vopin	AI+,AI- input range. (AVDDR = 2.4V)	0.4		1.05	V
PGIA Output Range	Vopout	X+,X- output range. (AVDDR = 2.4V)	0.4		1.05	V
Band gap Reference (Refer to ACM)						
Band gap Reference Voltage	VBG		1.18	1.23	1.28	V
Reference Voltage Temperature Coefficient	TACM			50*		PPM/℃
Operating current	IBG	Run mode @ 2.4V		200	250	uA
Regulator						
Regulator output voltage AVDDR	VAVDDR		2.25	2.4	2.5	V
Regulator output voltage AVE+	VAVE+	AVE+ set as 1.5V	1.4	1.5	1.6	V
Analog common voltage	VACM	VACM = 0.4	0.35	0.4	0.45	V
Regulator output current capacity	IVA+		10			mA
Quiescent current	IQI	ACM + AVDDR + AVE		80	100	uA
VACM driving capacity	ISRC		10	-	-	μA
VACM sinking capacity	ISNK		1	-	-	mA

16 封装信息

16.1 LQFP 64 PIN



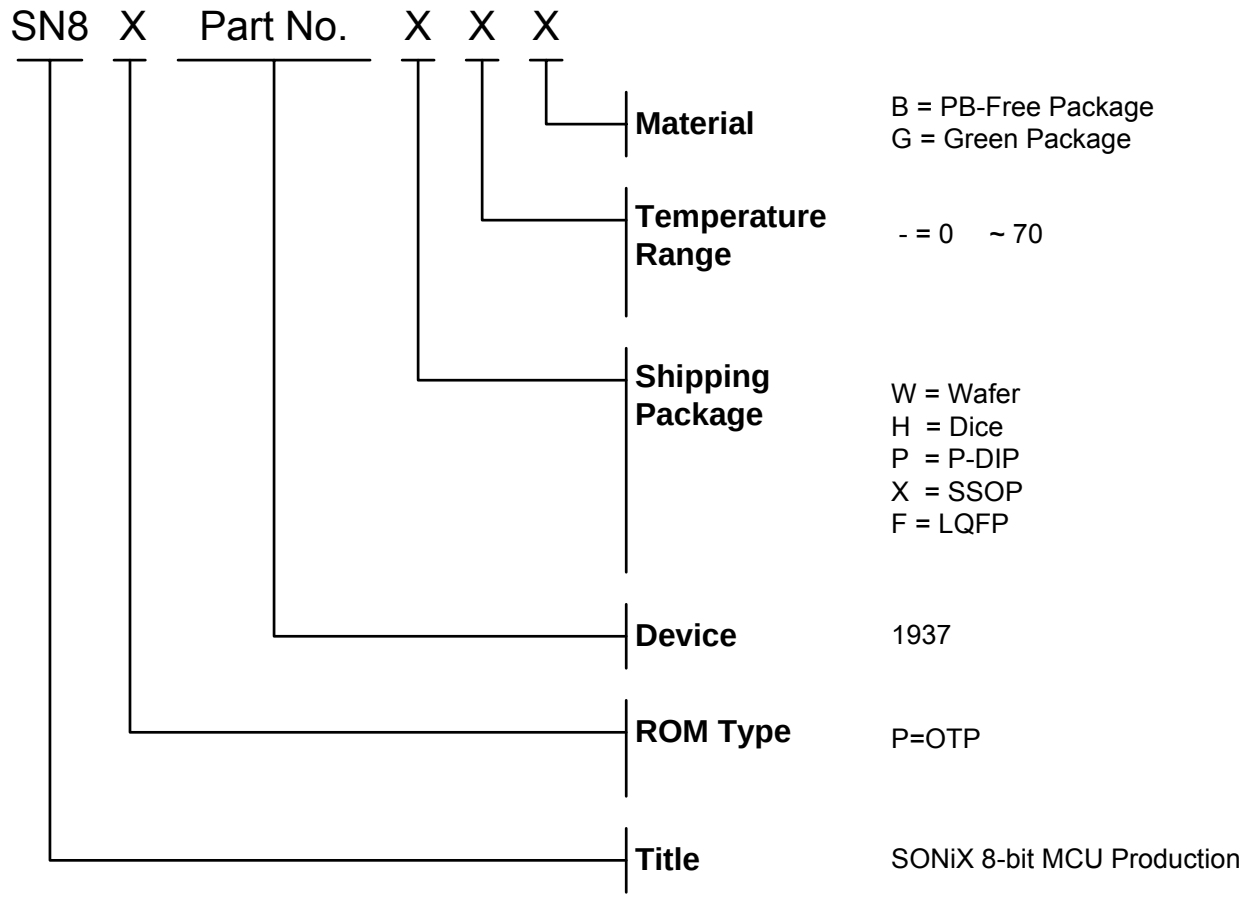
SYMBLE	DIMENSION (MM)			DIMENSION (MIL)		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A			1.60			63
A1	0.05	1.40	0.15	2	55	6
A2	1.36	0.22	1.45	35	9	57
b	0.17	0.22	0.27	7	8	11
b1	0.17		0.23	7		12
c	0.09		0.20	4		8
c1	0.09		0.16	4		6
D	11.75	12.00	12.25	463	473	483
D1	9.95	10.00	10.05	392	394	396
E	11.75	12.00	12.25	463	473	483
E1	9.95	10.00	10.05	392	394	396
[e]		0.50			20	
L	0.45	0.60	0.75	18	24	30
L1	0.9	1	1.1		39	
R1	0.08			3		
R2	0.08		0.20	3		8
Y			0.075			3
θ	0°	3.5°	7°	0°	3.5°	7°
θ 1	0°			0°		
θ 2	11°	12°	13°	11°	12°	13°
θ 3	11°	12°	13°	11°	12°	13°

17 单片机正印命名规则

17.1 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。

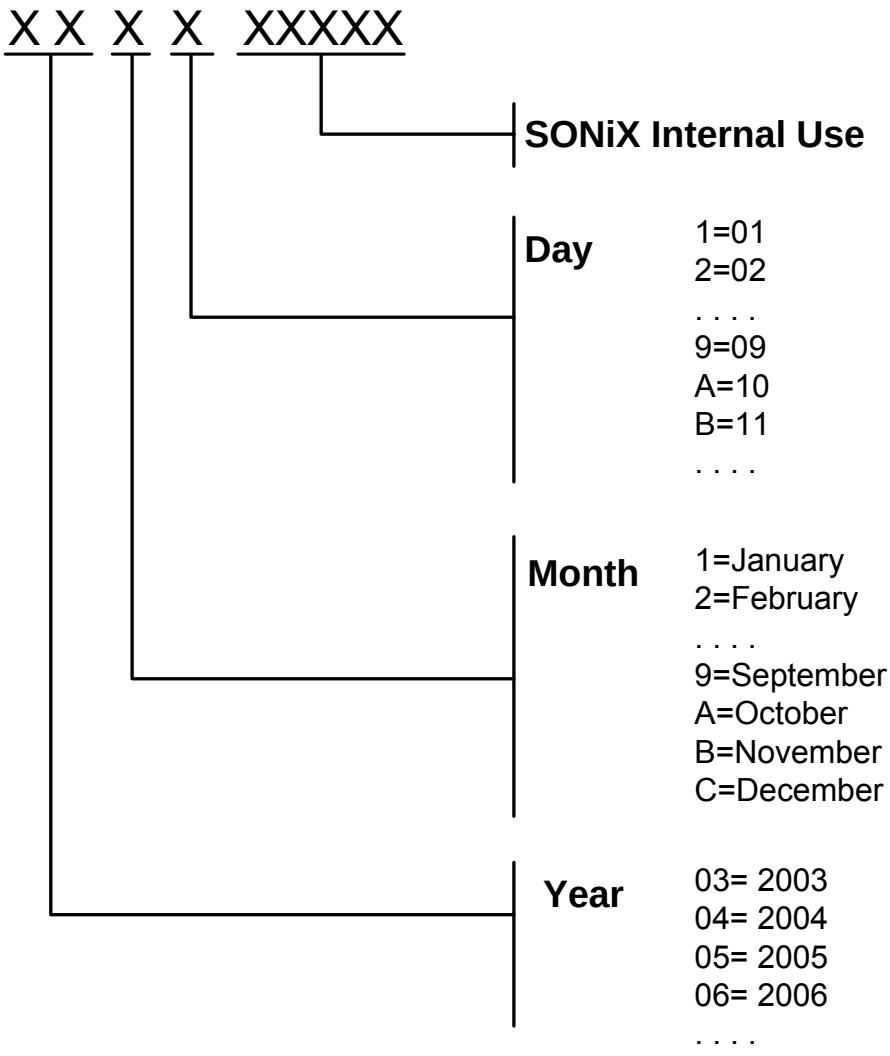
17.2 单片机型号说明



17.3 命名举例

单片机名称	ROM 类型	设备（Device）	封装显示	温度范围	封装材料
SN8P1937FB	OTP	1937	LQFP	0℃~70℃	无铅封装
SN8P1937FG	OTP	1937	LQFP	0℃~70℃	绿色封装

17.4 日期码规则



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

总公司：
地址：台湾新竹县竹北市台元街 36 号 10 楼之一
电话：886-3-5600-888
传真：886-3-5600-889
台北办事处：
地址：台北市松德路 171 号 15 楼之 2
电话：886-2-2759 1980
传真：886-2-2759 8180
香港办事处：
地址：香港新界沙田沙田乡宁会路 138＃新城市中央广场第一座 7 楼 705 室
电话：852-2723 8086
传真：852-2723 9179
松翰科技（深圳）有限公司
地址：深圳市南山区高新技术产业园南区 T2-B 栋 2 层
电话：86-755-2671 9666
传真：86-755-2671 9786
技术支持：
Sn8fae@SONiX.com.tw