SN8P1927

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	2008 年 3 月	初版。
	2008 年 5 月	修改烧录信息章节内容。

目　录

1 产品简介

1.1 产品性能表

单片机型号	ROM	RAM	堆栈	LCD	定时器			I/O	ADC	PWM Buzzer	SIO	唤醒功能引脚数目	封装形式
					T0	TC0	TC1						
SN8P1907	2K*16	128*8	8	4*12	V	-	-	11	16 位	-	-	5	SSOP48
SN8P1917	2K*16	128*8	8	4*12	V	-	-	13	16 位	-	-	5	SSOP48
SN8P1927	2K*16	128*8	8	4*12	V	-	-	13	16 位	-	-	5	SSOP48/LQFP48

SN8P1927 性能比较表

1.2 产品资源配置改良表

项目	SN8P1907	SN8P1917	SN8P1927
PGIA 增益设置	1x，16x，32x，64x，128x	1x，12.5x，50x，100x，200x	1x，12.5x，50x，100x，200x
PGIA 温度漂移	一般	好	好
AVE+电压	只有 3.0V	3.0V 或 1.5V	2.0V 或 1.5V
在 CPR ON 负载后的 AVE+电流	双倍电流功耗	双倍电流功耗	不是双倍电流功耗
ADC 参考电压 V(R+, R-)	只有 0.8V	0.8V/ 0.4V	0.8V/ 0.6V/ 0.4V
电池检测方法	ADC	通过比较器或 ADC	通过比较器或 ADC
温度传感器	外置	内置	内置
ACM（1.2V）电压	当外灌电流时，电压发生变化	当外灌电流时，电压不会变化	当外灌电流时，电压不会变化
Charge pump 时钟频率（CPCKS）	2 位	4 位	4 位
Chopper 时钟频率（AMPCKS）	2 位	3 位	3 位
AVDDR/AVE+能否在低速模式下工作	不能	能	能
高速模式下的功耗	多	少	少
低速模式下的功耗	多	少	少
LCD 偏压	1/2	1/3 或 1/2	1/3 或 1/2
内部 16M RC 振荡器	无	有	有
P2 [1:0] I/O	无	当 Fosc=IHRC 时有效	当 Fosc=IHRC 时有效
OTP 烧录方式	并行	串行	串行
In-System Programmer ROM	无	无	有
ADC 精确度	14 位	14 位	13 位

SN8P1927 资源配置改良表

1.3 功能特性

- **存储器配置**
  OTP ROM 空间：2K * 16 位。
  RAM 空间：128 字节（Bank0）。
  8 层堆栈缓存器。
  LCD RAM 空间：4*12 位。
- **I/O 引脚配置**
  单向输入端口：P0，P4。
  双向输入输出端口：P1，P2。
  单向输出端口：P5。
  具有唤醒功能的端口：P0，P1。
  内置上拉电阻的端口：P0，P1，P2，P4。
  外部中断引脚：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。
  查表功能（MOVC）可寻址整个 ROM 区。

- **可编程增益放大器**
  增益可选选项：1x/12.5x/50x/100x/200x。

- **16 位 Delta-Sigma ADC，具有 13 位精度**
  2 个 ADC 信道配置：
  1 个全差分通道。
  2 个单端输入通道。

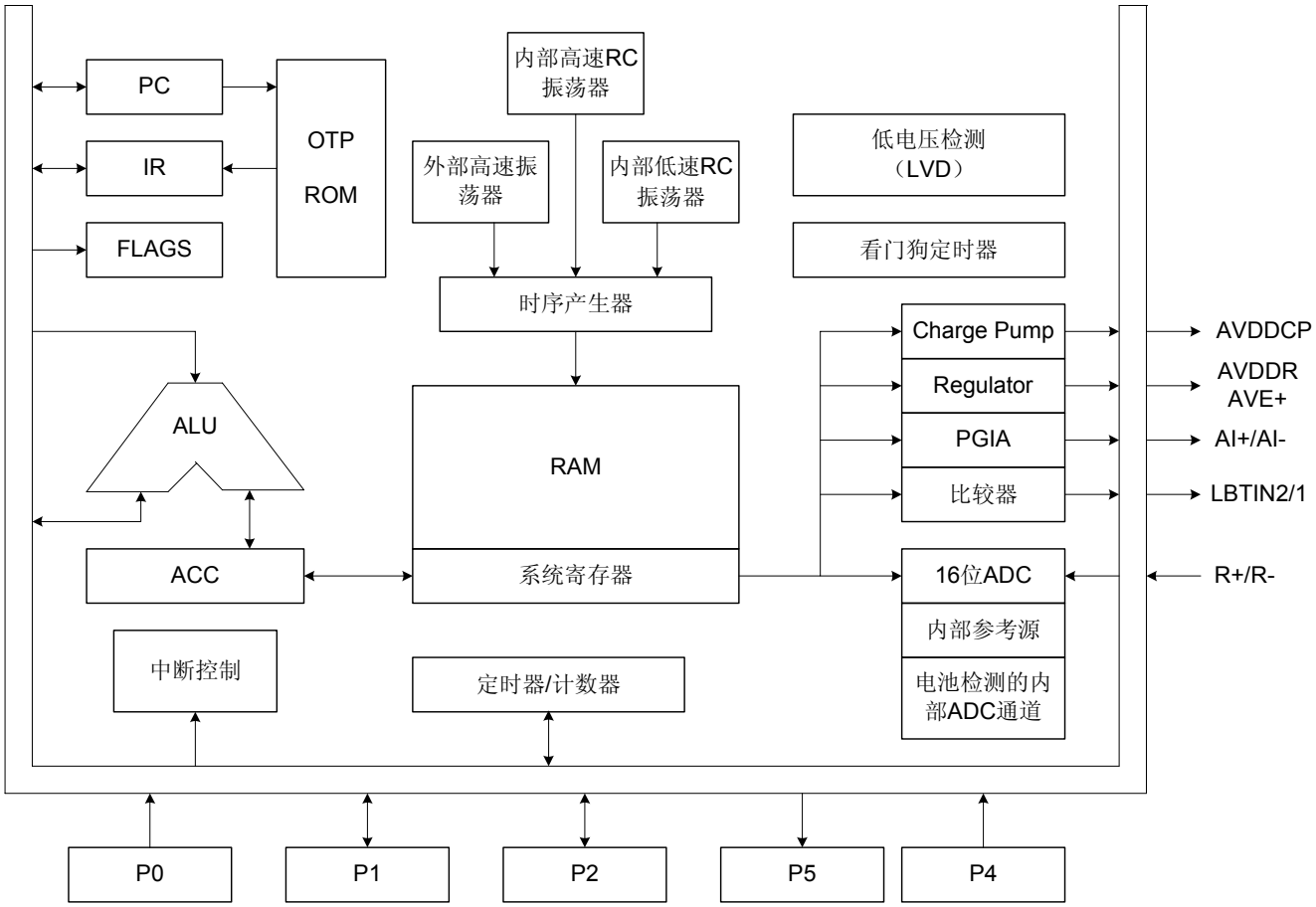
- **2 个中断源**
  1 个内部中断：T0。
  1 个外部中断：INT0。

- **单电源输入：2.4V ~5.5V**
- **内置看门狗定时器**
- **内置具有 3.8V 电压输出和 10mA 驱动电流的内置升压稳压器（charge-pump regulator）**
- **内置 2.0V/1.5V 电压输出的 regulator**
- **当 CPR ON 时 AVE+的负载电流功耗不是双倍**
- **内置参考电压 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-)= 0.8V/0.6V/0.4V**
- **具有在线烧录功能（In-System Programmer ROM）.**

- **LCD 驱动**
  1/3 或 1/2 偏压。
  4 common * 12 segment。

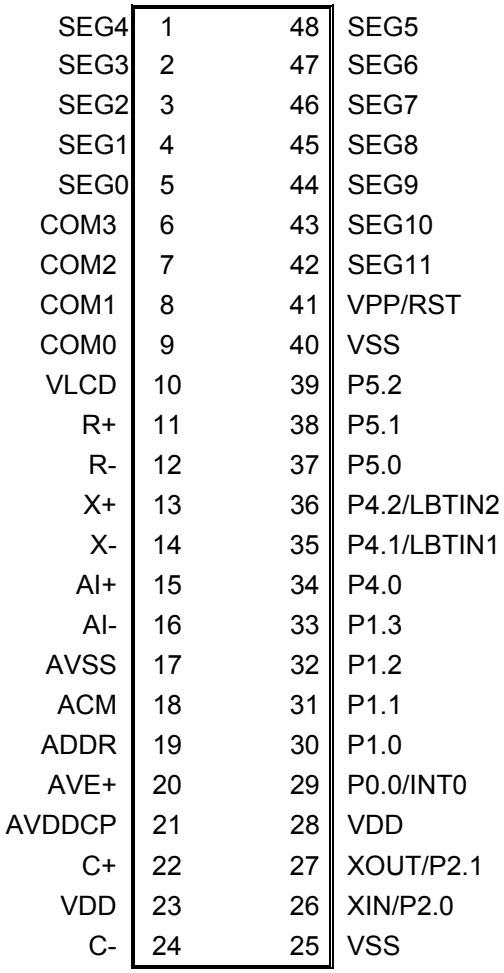
- **双时钟系统提供 3 种操作模式**
  外部高速时钟：RC 模式，高达 10MHz。
  外部高速时钟：晶体模式，高达 8MHz。
  普通模式：高、低速时钟同时运行。
  低速模式：仅低速时钟运行。
  睡眠模式：高、低速时钟均停止运行。
- **封装形式**
  SSOP48/LQFP48

1.4 系统框图


PC
IR
FLAGS
OTP
ROM
内部高速RC
振荡器
外部高速振
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
Charge Pump
Regulator
PGIA
比较器
AVDDCP
AVDDR
AVE+
AI+/AI-
LBTIN2/1
ALU
RAM
ACC
系统寄存器
16位ADC
R+/R-
内部参考源
中断控制
定时器/计数器
电池检测的内
部ADC通道
P0
P1
P2
P5
P4

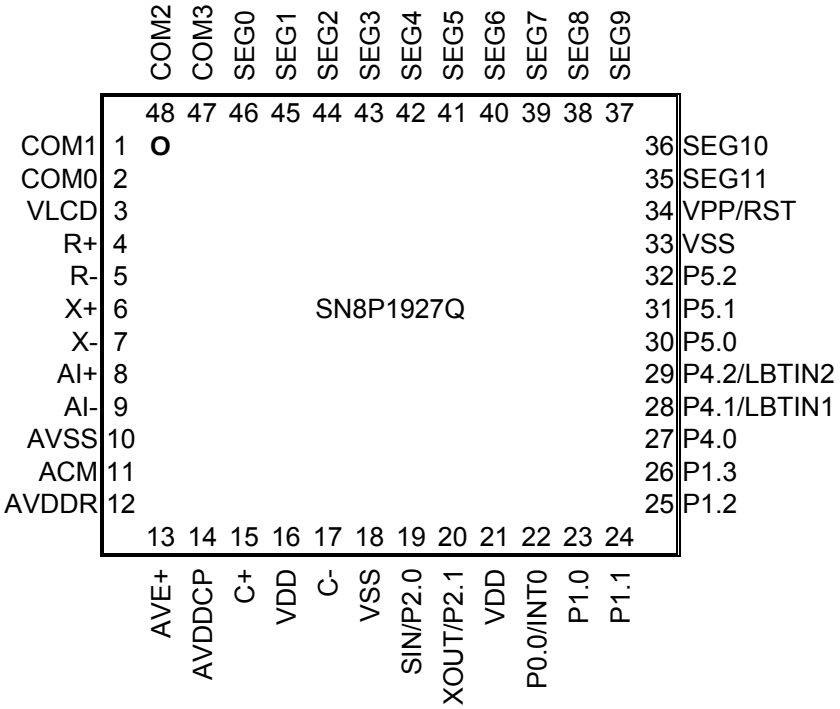
1.5 引脚配置

SN8P1927X



SN8P1927X

SN8P1927Q

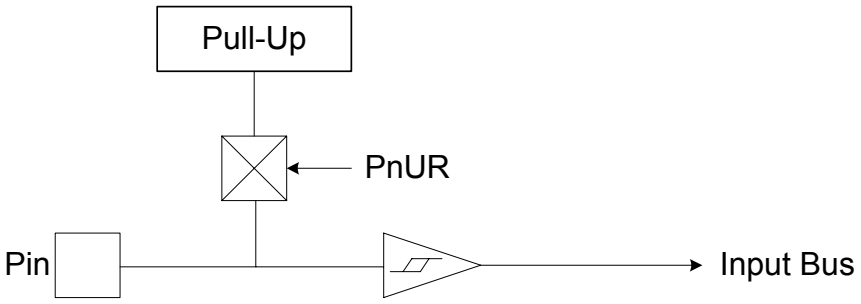


1.6 引脚说明

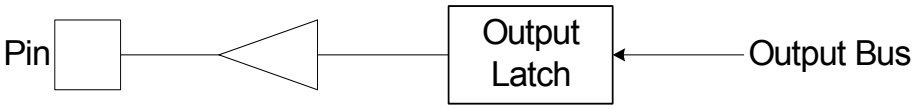
引脚名称	类型	功能说明
VDD, VSS, AVSS	P	电源输入端。
VLCD	P	LCD 电源输入端。
AVDDR	P	Regulator 电源输出引脚，V=3.8V。
AVE+	P	传感器的 Regulator 输出 = 2.0V/1.5V，最大输出电流为 10 mA。
ACM	P	Band Gap 电源输出为 1.2V。
AVDDCP	P	Charge Pump 电压输出引脚（和 VDD 之间连接一个 10uF 或者更大的电容）。
R+	AI	参考源的正极输入端。
R-	AI	参考源的负极输入端。
X+	AI	ADC 差分的正极输入端，和 X-引脚之间连接一个 0.1uF 的电容。
X-	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。
C+	A	charge pump regulator 电极电容的正极。
C-	A	charge pump regulator 电极电容的负极。
VPP/ RST	P, I	OTP ROM 的烧录引脚。 系统复位输入端，施密特触发，低电平有效，通常保持高电平。
XIN, XOUT	I, O	振荡信号的输入输出引脚。
P0.0 / INT0	I	P 0.0 和 INT0 共用引脚，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，和 XIN/XOUT 引脚共用。
P4 [2:0]	I	单向输入引脚，内置上拉电阻。
P5 [2:0]	O	单向输出引脚。
LBTIN1/2	I I	电池低电压检测输入引脚，和 P4.1、P4.2 引脚共用。
COM [3:0]	O	COM0~COM3 LCD 驱动 COM 端。
SEG0 ~ SEG11	O	LCD 驱动 segment 引脚。

1.7 引脚电路结构图

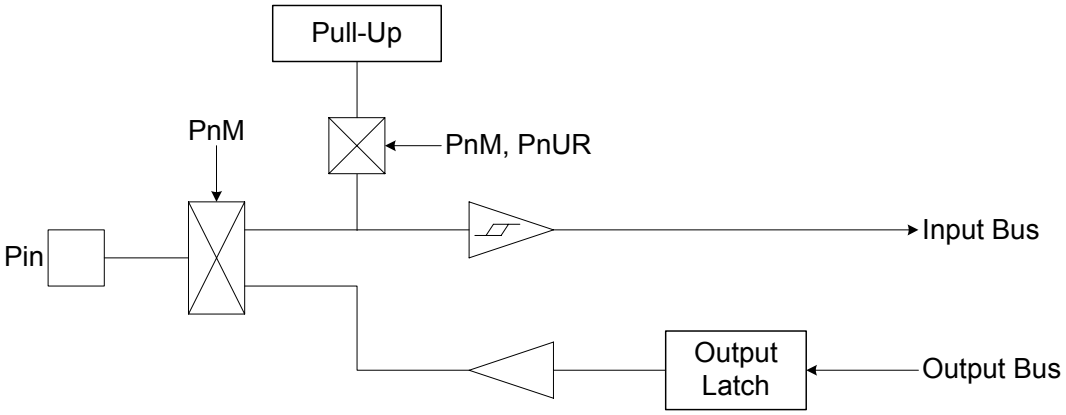
P0、P4：



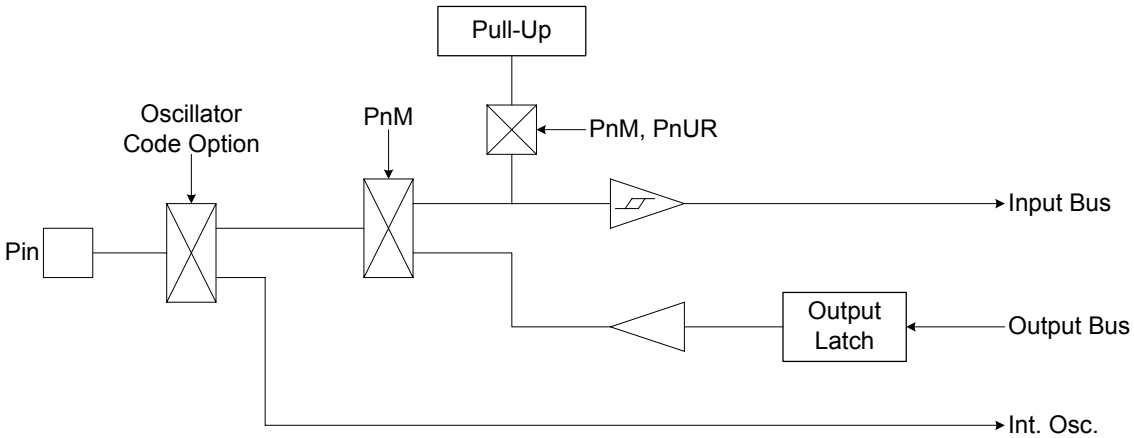
P5：



P1：



P2：



2 中央处理器（CPU）

2.1 存储器

2.1.1 程序存储器（ROM）

☞ ***ROM：2K***

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（0008H）

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

✷ **注：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 address
            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 address
            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（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。
	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_16K_RC	Always_ON	强行设置内部 16K RC 作为看门狗定时器的时钟源。 看门狗定时器在省电模式下仍然处于运行状态。
	By_CPUM	由 CPUM 寄存器控制内部 16K（@3V）RC 时钟是否工作。
Low Power	Enable	使能低功耗功能。
	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_16K_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 = Flosc/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 字节***

		RAM	
BANK 0	000H	**通用存储区**	; Bank0 的 000H~07FH（128 字节）是通用存储区
	07FH	.	
	080H	**系统寄存器**	; Bank0 的 080H~0FFH（128 字节）是系统寄存器区域
	0FFH	**Bank0 的结束区**	
BANK 15	0F00H	**LCD RAM**	; Bank 15 存储 LCD 显示数据区域（12 字节）
		.	;
	0F0BH	**LCD Ram 的结束区**	;

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	-	-	-	-	-	-
9	AMPM	AMPCHS	AMPCKS	ADCM	ADCKS	CPM	CPCKS	DFM	ADCDL	ADCDH	LBTM	-	-	-	-	-
A	ROMADRH	ROMADRL	ROMDAH	ROMDAL	ROMCNT	-	-	-	-	-	-	-	-	-	-	
B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	PEDGE
C	-	P1M	P2M	-	-	-	-	-	INTRQ	INTEN	OSCM	-	-	-	PCL	PCH
D	P0	P1	P2	-	P4	P5	-	-	T0M	T0C	-	-	-	-	-	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 = 专用寄存器，@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 时钟寄存器
ADCKS = ADC 时钟选项
CPCKS = Charge pump 时钟选项
ADCDL = ADC 低字节缓存器
P_NM = PN 输入/输出模式寄存器
P_N = PN 数据缓存器
INTEN = 中断使能寄存器
LCDM1 = LCD 模式寄存器
T0M = T0 模式寄存器
T0C = T0 计数寄存器
LBTM = 电池低电压检测器

R = 工作寄存器和 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
P_NUR = 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	-	-	LCDBNK	-	LCDENB	LCDBIAS	LCDRATE	LCDCLK	R/W	LCDM1
090H	-	BGRENB	FDS1	FDS0	GS2	GS1	GS0	AMPENB	R/W	AMPM
091H	-	-	-	-	-	CHS2	CHS1	CHS0	R/W	AMPCHS
092H	-	-	-	-	-	AMPCKS2	AMPCKS1	AMPCKS0	W	AMPCKS
093H	-	-	-	-	IRVS	RVS1	RVS0	ADCENB	R/W	ADCM
094H	ADCKS7	ADCKS6	ADCKS5	ADCKS4	ADCKS3	ADCKS2	ADCKS1	ADCKS0	W	ADCKS
095H	ACMENB	AVDDRENB	AVESEL	AVENB	CPSTS	CPAUTO	CPON	CPRENB	R/W	CPM
096H	-	-	-	-	CPCKS3	CPCKS2	CPCKS1	CPCKS0	W	CPCKS
097H			-	-	-	WRS0		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	P41IO	LBTENB	R/W	LBTM
0A0H	VPPCHK	ROMADR14	ROMADR13	ROMADR12	ROMADR11	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
0C8H	-	-	-	T0IRQ	-	-	-	P00IRQ	R/W	INTRQ
0C9H	-	-	-	T0IEN	-	-	-	P00IEN	R/W	INTEN
0CAH	WTCKS	WDRST	WDARTE	-	CPUM0	CLKMD	STPHX	-	R/W	OSCM
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
0D4H	-	-	-	-	-	P42	P41	P40	R	P4
0D5H	-	-	-	-	-	P52	P51	P50	R/W	P5
0D8H	T0ENB	T0RATE2	T0RATE1	T0RATE0	-	-	-	-	R/W	T0M
0D9H	T0C7	T0C6	T0C5	T0C4	T0C3	T0C2	T0C1	T0C0	R/W	T0C
0DFH	GIE	-	-	-	STKPB3	STKPB2	STKPB1	STKPB0	R/W	STKP
0E1H	-	-	-	-	P13R	P12R	P11R	P10R	W	P1UR
0E2H	-	-	-	-	-	-	P21R	P20R	W	P2UR
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

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。

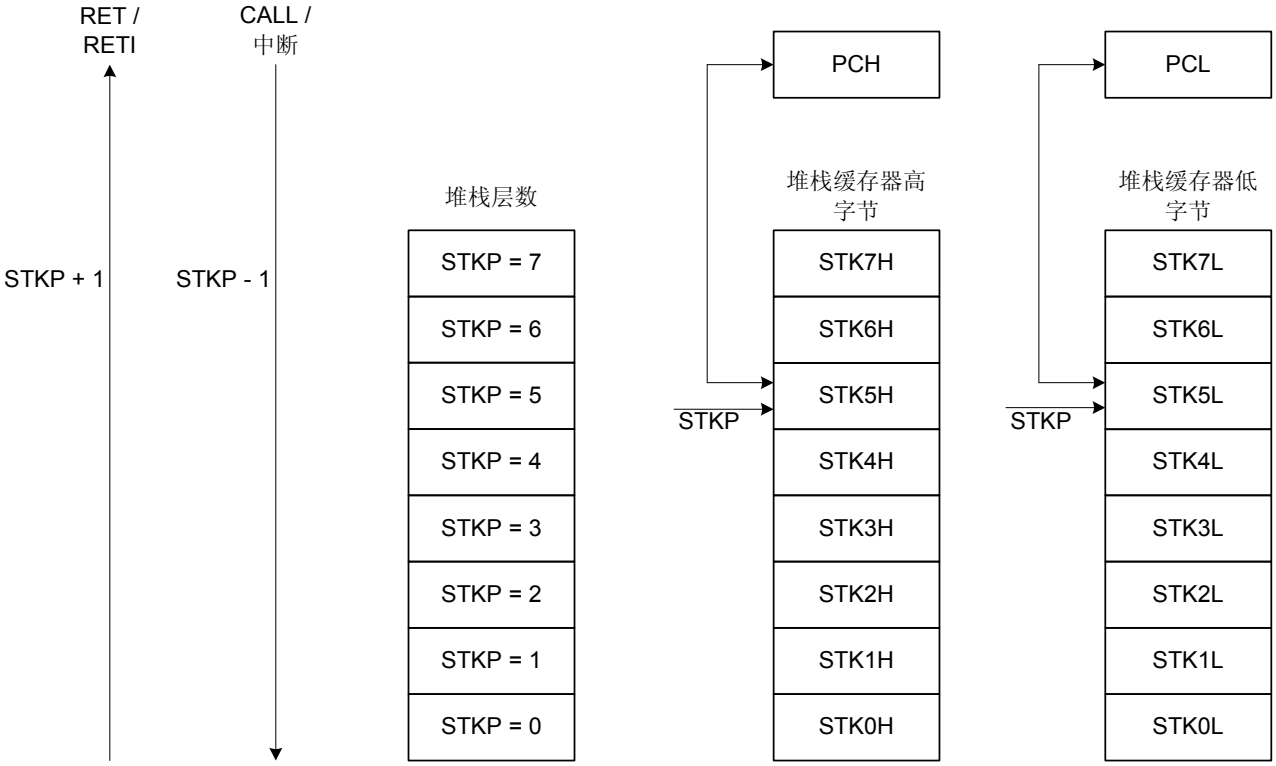
- **例：用 @YZ 实现间接寻址。**

```
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

2.3.1 概述

SN8P1927 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

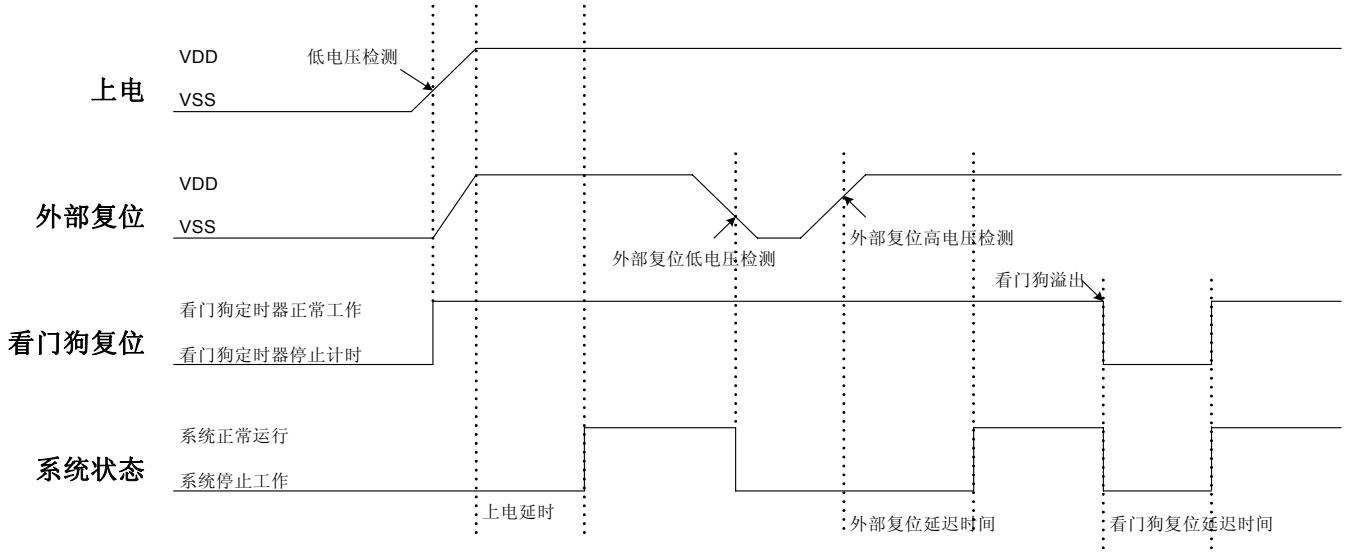
3.1 概述

SN8P1927 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。晶体振荡器类型不同则复位时间亦存在差别，这使得 VDD 上升时间和启动时间不是确定值。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：

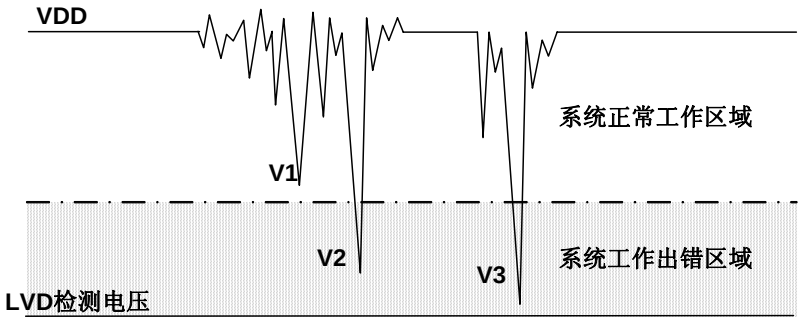
- 看门狗定时器清零之前，请检查 I/O 口的状态和 RAM 的内容可增强程序的可靠性；
- 不能在中断中对看门狗清零，否则无法侦测到主程序跑飞的状况；
- 程序中应该只在主程序中有一次清看门狗的动作，这种架构能够最大限度的发挥看门狗的保护功能。

✷ **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 掉电

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。

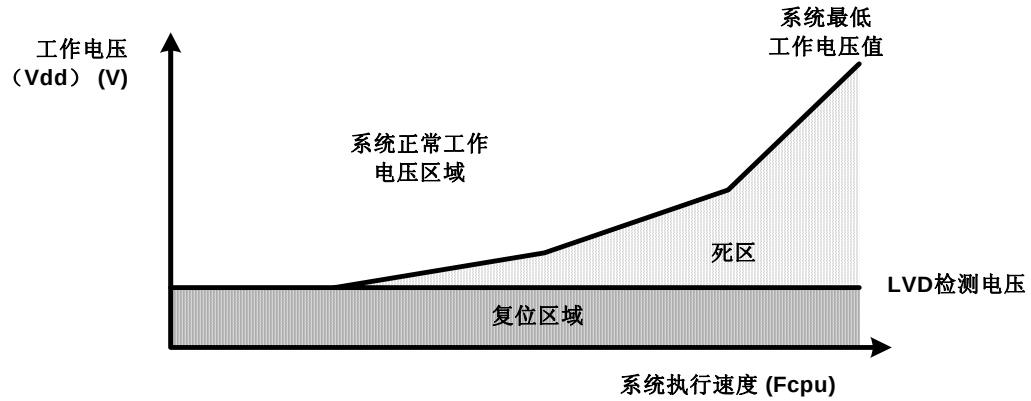
AC 运用中：

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。

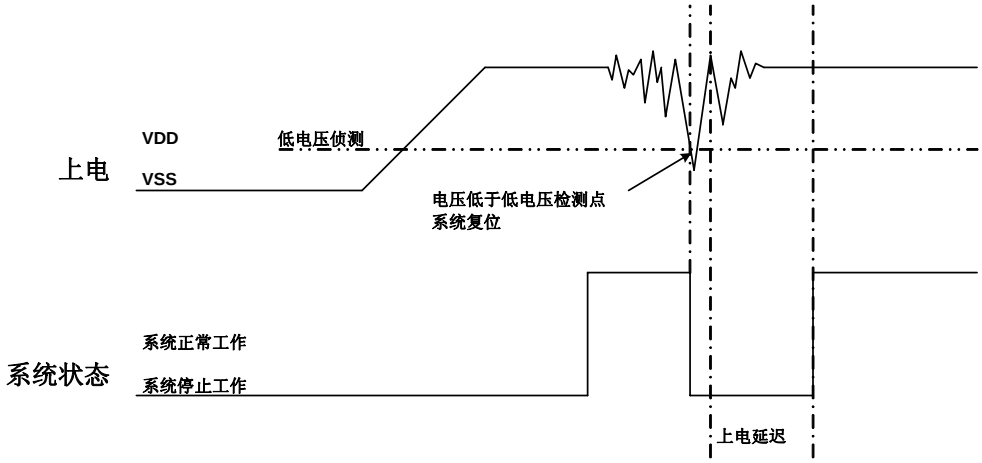
3.4.3 掉电复位性能改进

如何改善系统掉电复位性能，有以下几点建议：

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

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

LVD 复位：



低电压检测（LVD）是 SONiX 8 位单片机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

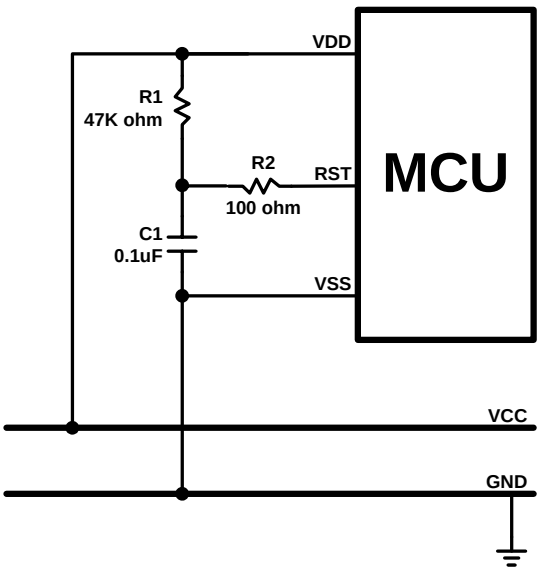
外部复位功能由编译选项“RESET PIN”控制。将该编译选项置为“Reset”，可使能外部复位功能。外部复位引脚为施密特触发器结构，低电平有效。复位引脚处于高电平时，系统正常运行。当复位引脚输入低电平信号时，系统复位。外部复位操作在上电和正常工作模式时有效。需要注意的是，在系统上电的过程中，外部复位引脚必须输入高电平，否则系统将一直保持在复位状态。外部复位的时序如下：

- **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

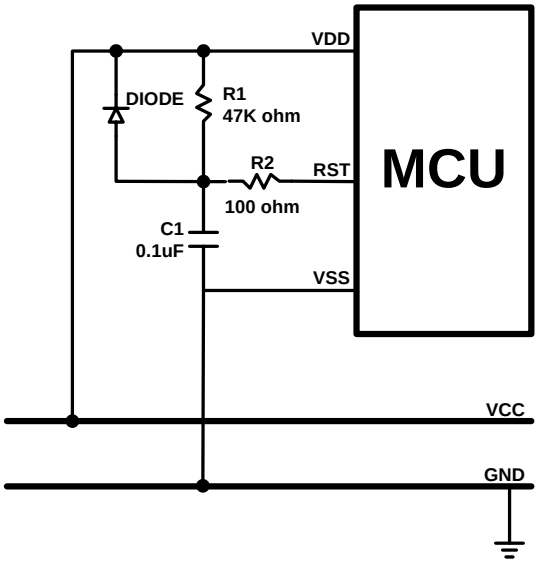
3.6.1 基本 RC 复位电路



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。**

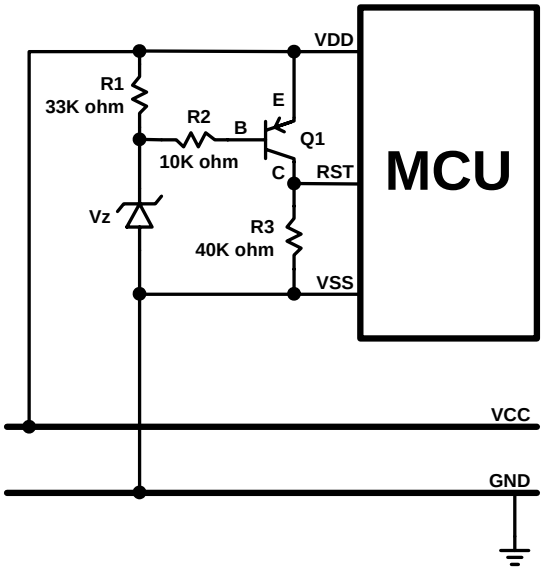
3.6.2 二极管及 RC 复位电路



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

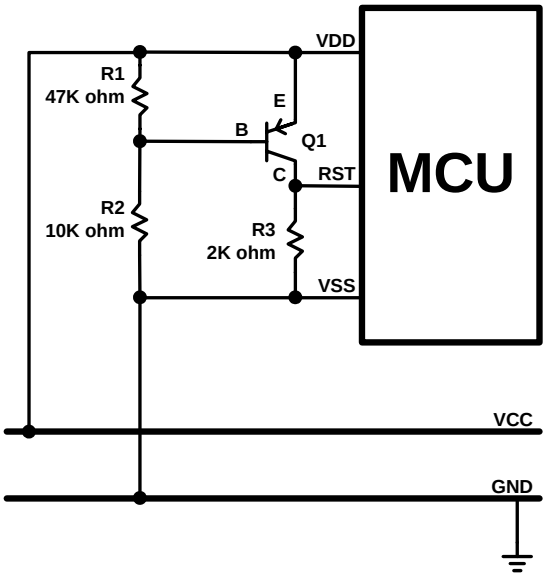
✷ **注：“基本 RC 复位电路”和“二极管及 RC 复位电路”中的电阻 R2 都是必不可少的限流电阻，以避免复位引脚 ESD（Electrostatic Discharge）或 EOS（Electrical Over-stress）)击穿 。**

3.6.3 稳压二极管复位电路



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

3.6.4 电压偏移复位电路

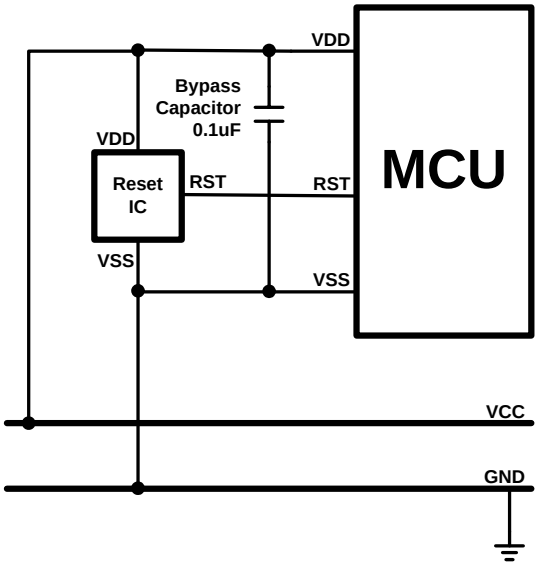


电压偏置复位电路是一种简单的 LVD 电路，基本上可以完全解决掉电复位问题。与稳压二极管复位电路相比，偏压复位电路的的检测电压值的精确度有所降低。电路中，R1 和 R2 构成分压电路，当 VDD 高于和等于分压值“0.7V x (R1 + R2) / R1”时，三极管集电极 C 输出高电平，单片机正常工作；VDD 低于“0.7V x (R1 + R2) / R1”时，集电极 C 输出低电平，单片机复位。

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


VDD
Bypass
Capacitor
0.1uF
VDD
Reset
IC
RST
RST
MCU
VSS
VSS
VCC
GND

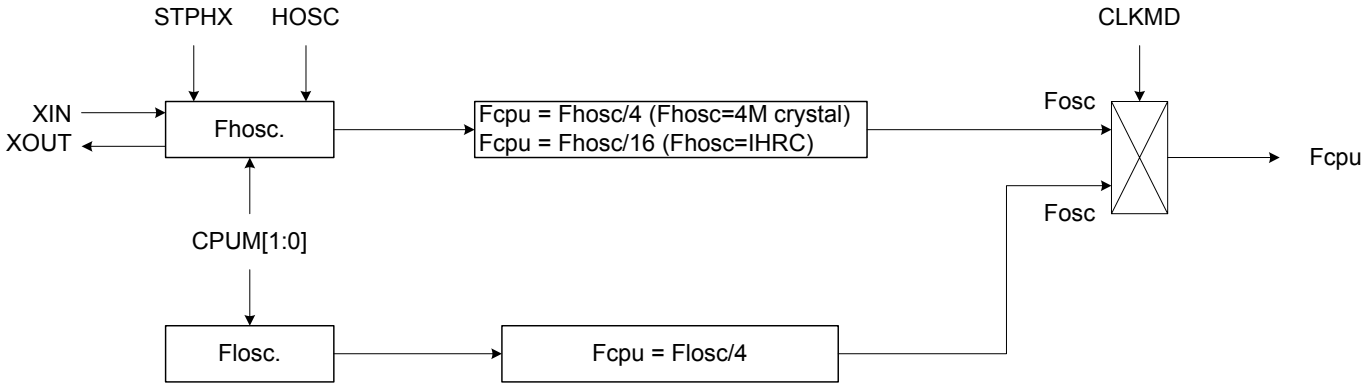
4 系统时钟

4.1 概述

SN8P1927 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，低速时钟则由内置低速 RC 振荡电路（ILRC 16KHzZ@3V，32KHz@5V）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 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：内部低速 RC 时钟频率（16KHz@3V，32KHz@5V）。
Fosc：系统时钟频率。
Fcpu：指令执行频率。

4.3 OSCM 振荡器

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	-	CPUM0	CLKMD	STPHX	-
读/写	R/W	R/W	R/W	-	R/W	R/W	R/W	-
复位后	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，系统时钟来自低速时钟。

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。

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

➢ **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
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

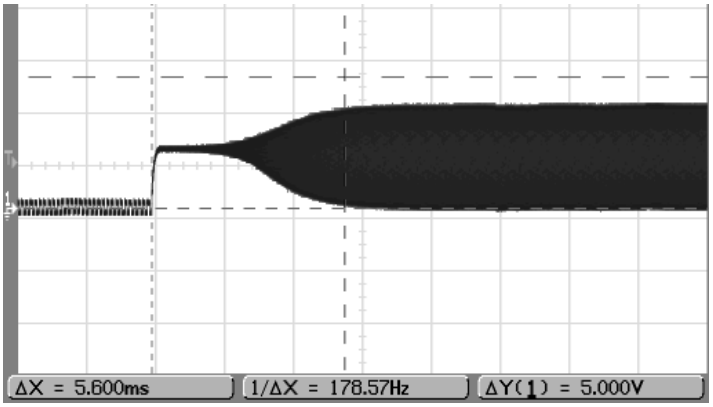
代码选项“IHRC_16M”和控制单片机的内置 RC 高速时钟（16MHZ）。若选择“IHRC_16M”，则内置 16MHZ RC 振荡器作为系统时钟源，XIN 和 XOUT 引脚作为通用 I/O 口。

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

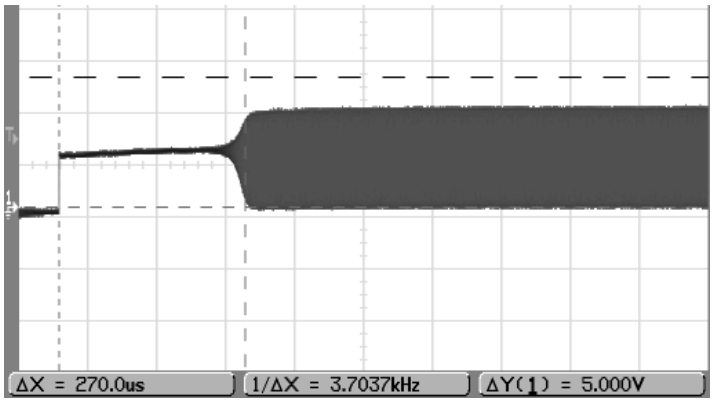
4.4.2 外部高速时钟

外部高速时钟共两种模式：石英/陶瓷振荡器和外部时钟源，由代码选项 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

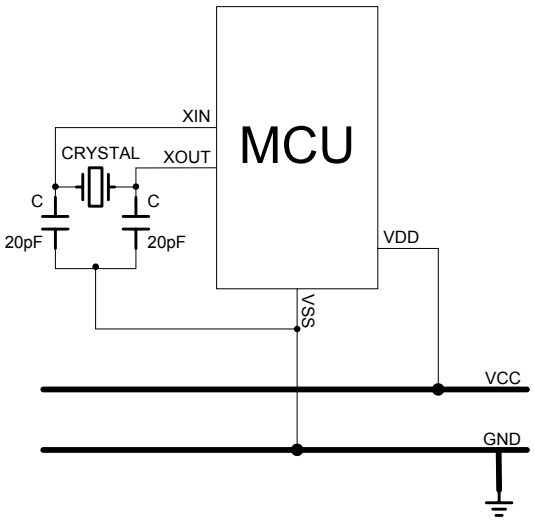


4MHz Ceramic



4.4.2.1 石英/陶瓷振荡器

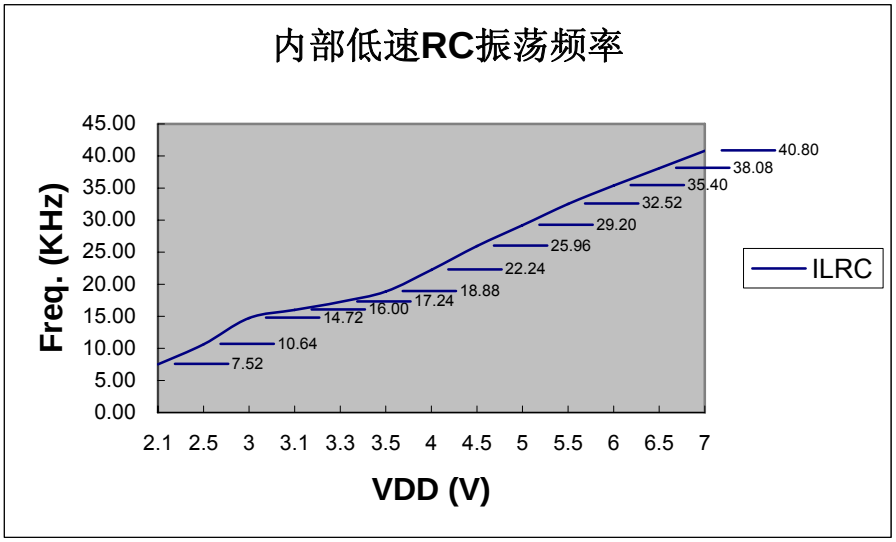
石英/陶瓷振荡器由 XIN/XOUT 口驱动，对于高速、普通和低速三种不同工作模式，振荡器的驱动电流也不同。不同的工作模式下，代码选项“High_Clk”支持不同的频率条件：如普通模式下 4MHz 工作频率。



✷ **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

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



低速时钟可作为看门狗定时器的时钟源。由 CLKMD 控制系统低速工作模式。

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

系统工作在睡眠模式下禁止看门狗时，可以停止低速 RC 振荡器。如果系统工作频率为 32K 且禁止看门狗，那么这种情况下只有 32K 振荡器处于工作状态，系统功耗相应较低。

➢ **例：停止内部低速振荡器。**

```
B0BSET      FCPUM0
```

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

4.5.1 系统时钟测试

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

➢ **例：外部振荡器的 Fcpu 指令周期测试。**

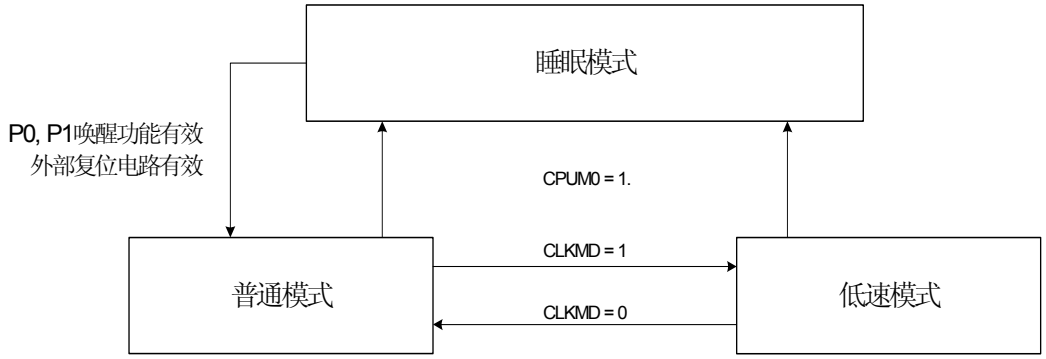
```
B0BSET      P1M.0       ; 设置 P1.0 位输出模式以输出 Fcpu 的触发信号。
@@:
B0BSET      P1.0
B0BCLR      P1.0
JMP         @B
```

5 系统工作模式

5.1 概述

SN8P1927 可以在如下 3 种工作模式之间切换：

- **高速模式；**
- **低速模式；**
- **睡眠模式。**



系统模式切换示意图

工作模式

工作模式	普通模式	低速模式	睡眠模式	备注
EHOSC	运行	由 STPHX 控制	停止	
IHRC	运行	由 STPHX 控制	停止	
ILRC	运行	运行	停止	
CPU 指令	执行	执行	停止	
T0 定时器	*有效	*有效	无效	* T0ENB=1 时有效
看门狗定时器	由 Watch_Dog 编译选项控制	由 Watch_Dog 编译选项控制	由 Watch_Dog 编译选项控制	参阅编译选项章节
内部中断	全部有效	全部有效	全部无效	
外部中断	全部有效	全部有效	全部无效	
唤醒功能	-	-	P0，P1，复位	

EHOSC：外部高速时钟；
IHRC：内部高速时钟（16M RC 振荡器）；
ILRC：内部低速时钟（RC 振荡器：3V 时 16K，5V 时 32K）；

5.2 系统模式切换

➢ **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 10ms 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, #27      ; 若 VDD = 5V，内部 RC=32KHz（典型值）
        B0MOV       Z, A
@@:     DECMS       Z           ; 需延迟 0.125ms X 81 = 10.125ms 等待外部时钟稳定。
        JMP         @B
                                ;
        B0BCLR      FCLKMD      ; 系统返回到普通模式。
```

5.3 唤醒时间

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 个外部高速振荡器时钟周期以使振荡电路进入稳定工作状态，等待的这一段时间就称为唤醒时间。唤醒时间结束后，系统才进入到普通模式。

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。**

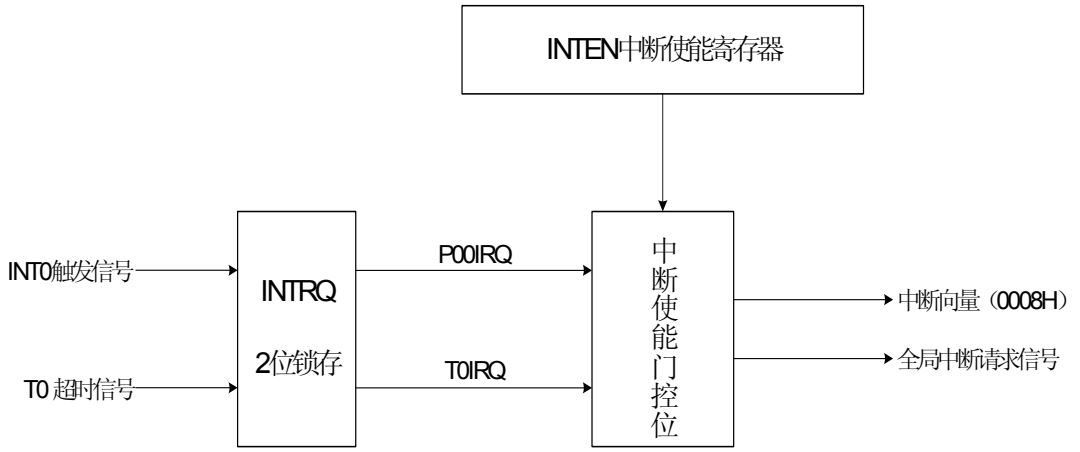
➢ **例：将系统从睡眠模式中唤醒，并设置系统进入普通模式。唤醒时间计算如下。**

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

SN8P1927 共有 2 个中断源：1 个内部中断 T0 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 中断请求使能寄存器 INTEN

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

0C9H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTEN	-	-	-	T0IEN	-	-	-	P00IEN
读/写	-	-	-	R/W	-	-	-	R/W
复位后	-	-	-	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。

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	-	-	-	T0IRQ	-	-	-	P00IRQ
读/写	-	-	-	R/W	-	-	-	R/W
复位后	-	-	-	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。

6.4 GIE 全局中断

只有当全局中断控制寄存器 GIE 置“1”的时候程序才能响应中断请求。 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 处理

有中断请求发生并被响应后，程序转至 0008H 执行中断子程序。响应中断之前，必须保存 ACC 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可能的程序运行错误。

✷ **注：必须使用“B0XCH”指令来保护和恢复 ACC 的内容。否则 ACC 操作可能会改变 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；
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, #64H         ; 初始化 T0C = 64H。
        B0MOV       T0C, A          ; 设置 T0 间隔时间= 10 ms。

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

        B0BSET      FGIE            ; 使能 GIE。
```

➢ **例：T0 中断服务程序。**

```
        ORG         8H
        JMP         INT_SERVICE
INT_SERVICE:

        …                           ; 保存 ACC 和 PFLAG。

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

        B0BCLR      FT0IRQ          ; 清 T0IRQ。
        MOV         A, #64H
        B0MOV       T0C, A          ;
        …
        …
EXIT_INT:
        …                           ; 恢复 ACC 和 PFLAG。

        RETI                        ; 退出中断。
```

6.8 多中断操作举例

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

中断	有效触发
P00IRQ	由 PEDGE 控制
T0IRQ	T0C 溢出

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

➢ **例：多中断条件下检测中断请求。**

```
                ORG             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             INTT0CHK        ; 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             INT_EXIT        ;
                B0BTS0          FT0IRQ          ; 检查是否有 T0 中断请求。
                JMP             INTT0           ; 进入 T0 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

Bit[7:0] **PnM[7:0]**：Pn 模式控制位（n = 1~2）。
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. **P4 是单向输入端口。**
3. **P5 是单向输出端口。**
4. **P2 口和 XIN/XOUT 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 口上拉电阻寄存器

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

✷ **注：P0 和 P4 口的上拉电阻是始终存在的。**

➢ **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
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

0D4H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P4	-	-	-	-	-	P42	P41	P40
读/写	-	-	-	-	-	R	R	R
复位后	-	-	-	-	-	0	0	0

0D5H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P5	-	-	-	-	-	P52	P51	P50
读/写	-	-	-	-	-	W	W	W
复位后	-	-	-	-	-	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, P4   ; 读 P4 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，如果不能将看门狗定时清零，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	-	CPUM0	CLKMD	STPHX	-
读/写	R/W	R/W	R/W	-	R/W	R/W	R/W	-
复位后	0	0	0	-	0	0	0	-

Bit5 **WDRATE**：看门狗定时器分频选择位。
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 = 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。

看门狗定时器溢出时间表

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

✷ **注：看门狗定时器由编译选项（Code Option）控制。**

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。**

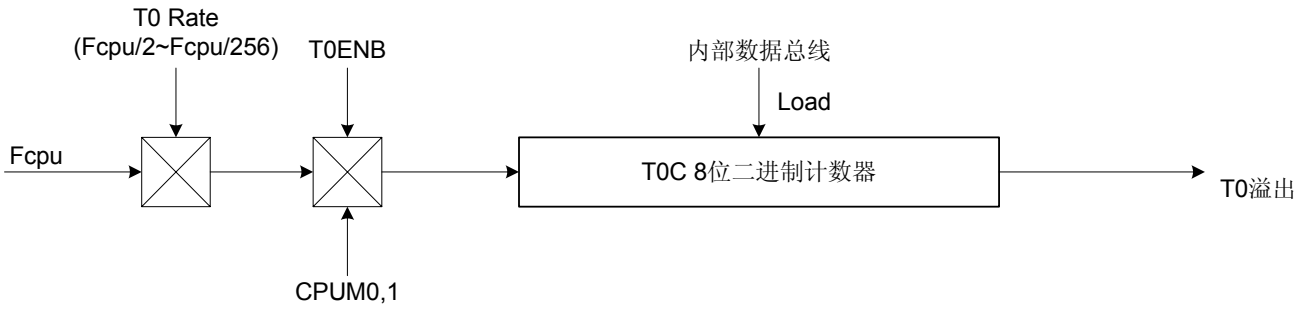
```
Main:
        …                       ; 检查 I/O 口的状态。
        …                       ; 检查 RAM 的内容。
Err:    JMP $                   ; I/O 或 RAM 出错，不清看门狗等看门狗计时溢出。

Correct:                        ; I/O 和 RAM 正常，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

8 位二进制计数器 T0 溢出时（由 0FFH 计到 00H），T0 在继续计数的同时给出一个溢出信号触发 T0 中断。计数器 T0 主要有以下功能：

- **8 位可编程定时器：**根据选定的时钟频率定时产生中断；
- **绿色模式唤醒功能：**在 T0ENB=1 的条件下，T0 的溢出可将系统从绿色模式中唤醒。



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

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 T0 计数器；
1 = 开启 T0 计数器。

8.2.3 T0C 计数寄存器

T0C 用于控制 T0 的间隔时间。

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 中断间隔时间 * 输入时钟）

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

T0C 初始值 = 256 -（T0 中断间隔时间 * 输入时钟）
= 256 -（10ms * 4MHz / 4 / 64）
= 256 -（10^{-2} * 4 * 10^{6} / 4 / 64）
= 100
= 64H

T0 定时间隔列表

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

8.2.4 T0 操作流程

☞ **T0 停止计数，禁止 T0 中断并将 T0 中断请求标志清零。**

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

☞ **设置 T0 计时速率。**

```
MOV       A,#0xxx00b      ; T0M 的 bit4~bit6 位可控制 T0 的计数速率为 x000xxxxb~x111xxxxb。
B0MOV     T0M,A
```

☞ **设置 T0 中断间隔时间。**

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

☞ **设置 T0 工作模式。**

```
B0BSET    FT0IEN
```

☞ **开启 T0。**

```
B0BSET    FT0ENB
```

9 LCD 驱动

SN8P1927 的 LCD 驱动包括 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，共有 48 点驱动。

9.1 LCDM1 寄存器

LCDM1 初始值= xx0x 0011

089H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LCDM1	-	-	LCDBNK	-	LCDENB	LCDBIAS	LCDRATE	LCDCLK
读/写			R/W	-	R/W	R/W	R/W	R/W
复位后	-	-	0	-	0	0	1	1

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

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

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

Bit1 **LCDRATE**：LCD 时钟速率控制位（LCD CLK = 1）。
0 = LCD 时钟 = 内部 RC/ 64；
1 = LCD 时钟 = 内部 RC/ 32。

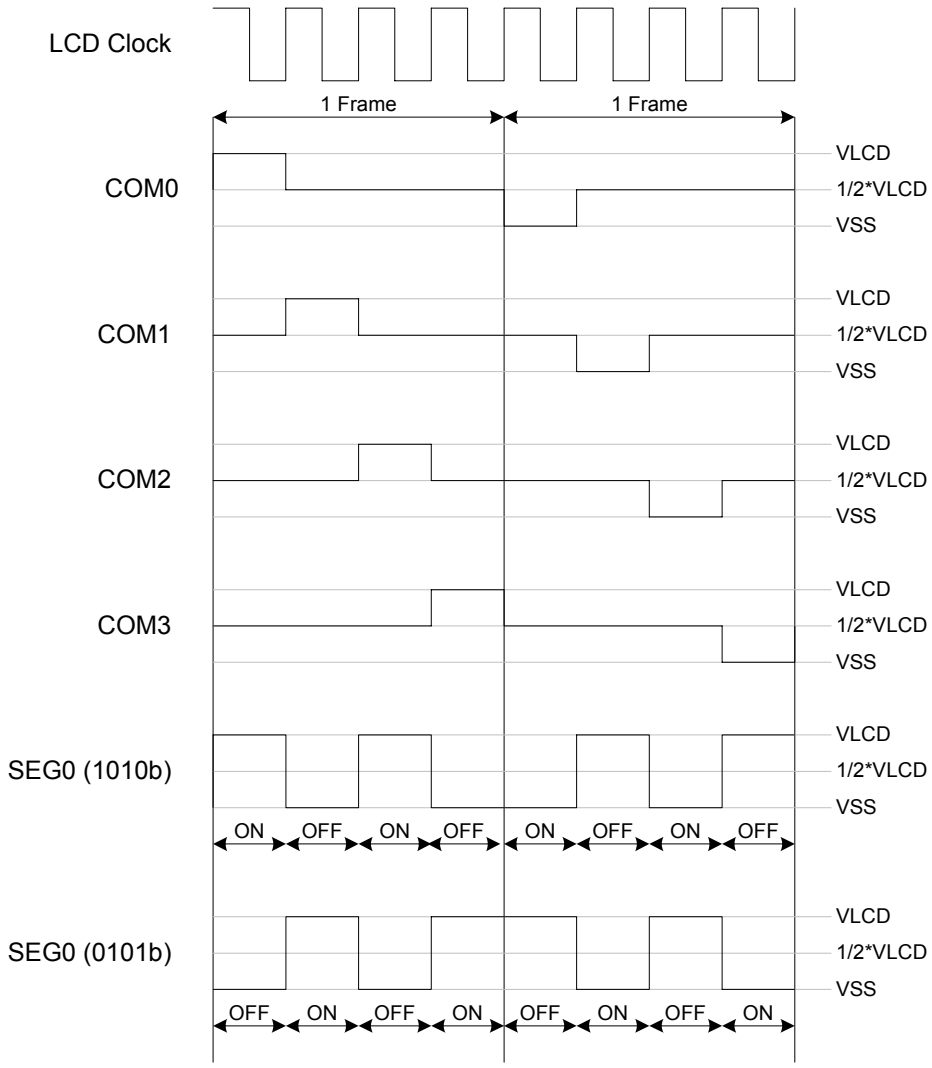
Bit0 **LCDCLK**：LCD 时钟源选择控制位。
0 = LCD 时钟 = 外部/2^14，帧比率 = LCD 时钟/ 4；
高速时钟 = 4M，LCD 时钟 = 244.14Hz，帧比率 = 244.14/4=61.03
高速时钟 = 3.58M，LCD 时钟 = 218.51Hz，帧比率 = 218.51/4=54.62
1 = LCD 时钟 = 内部 RC /32（LCDRATE=1）或者内部 RC = 64（LCDRATE=0）。
帧比率 = LCD 时钟/4

✷ **注 1：当 SN8P1927 的封装形式是 Dice 时，另外 2 个引脚 V1/V2 可以用来调节 LCD 的驱动能力。**
✷ **注 2：在 1/3 偏压时设置 V1 = 1/3VLCD，V2 = 2/3VLCD；1/2 偏压时将 V1 和 V2 短路，即设置为 V1 = V2 = 1/2VLCD。**
✷ **注 3：引脚 V1/V2 只在 Dice 的形式下才有，封装片是没有的。**

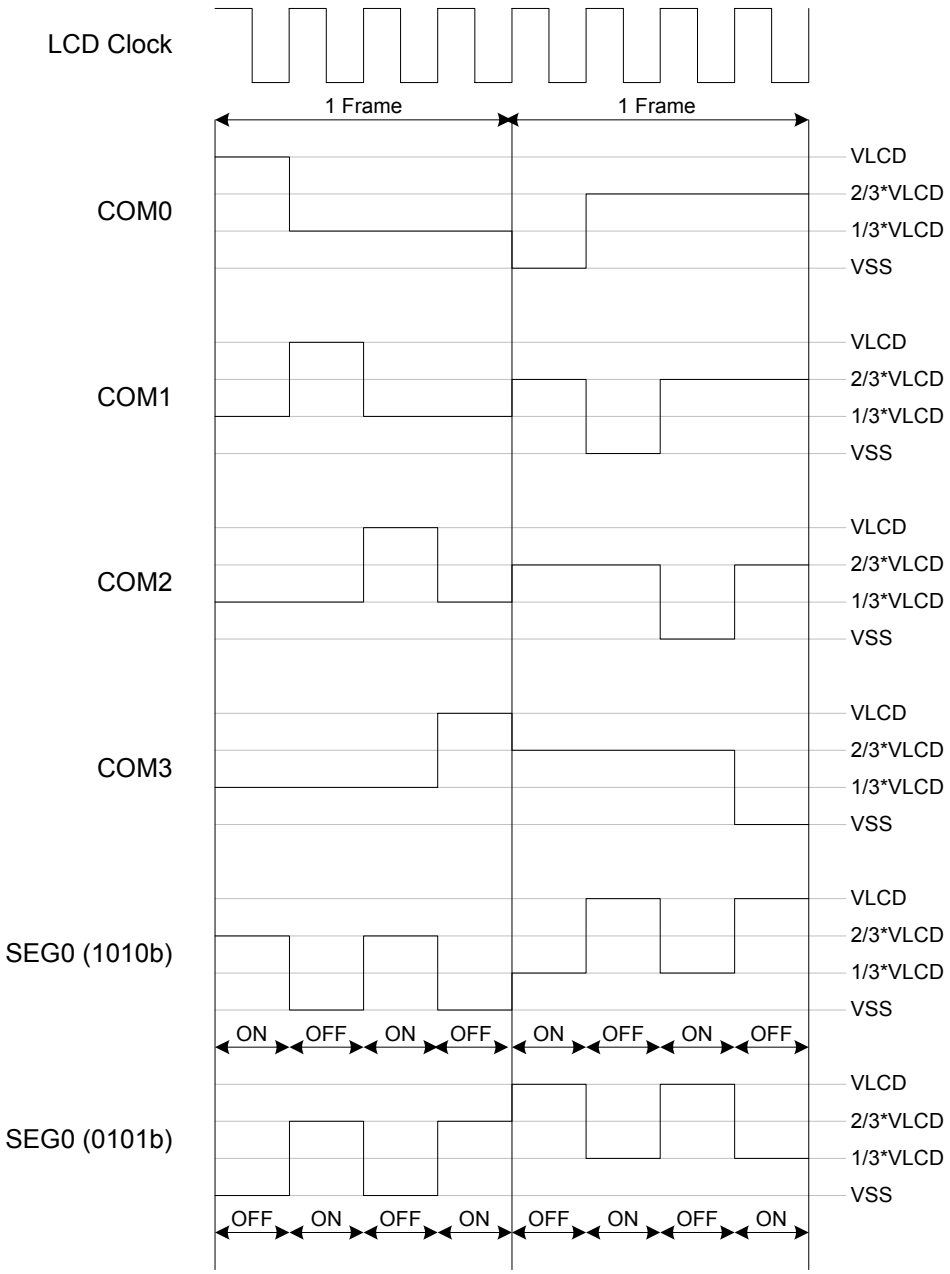
9.2 LCD 时序

LCD 时序表

LCDCLK	LCD 时钟源	LCDRATE	LCD Clock	Frame=LCD clock/4
0	Fhosc	X	4M/2^14=244.14Hz@4M	244.14/4=61.03Hz
0	Fhosc	X	3.58M/2^14=218.51Hz@3.58M	218.51/4=54.6Hz
1	Flosc	0	16K/64=250Hz@3V	250/4=62.5Hz
1	Flosc	1	16K/32=500Hz@3V	500/4=125Hz
1	Flosc	0	32K/64=500Hz@5V	500/4=125Hz
1	Flosc	1	32K/32=1000Hz@3V	1000/4=250Hz



LCD 驱动波形，1/4 占空比，1/2 偏压



LCD 驱动波形，1/4 占空比，1/3 偏压

9.3 LCD RAM 地址

RAM bank 15 的地址与 Common/Segment 引脚地址的关系

	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。

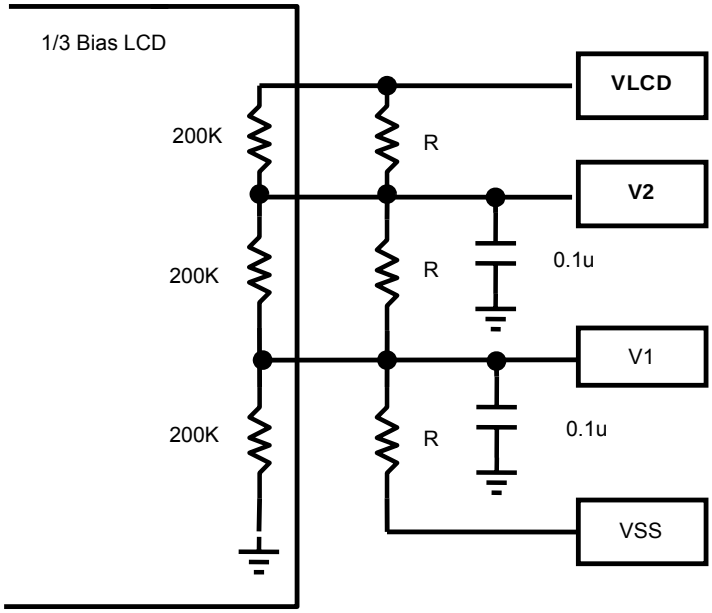
```
B0BSET      FLCDENB      ; LCD 驱动。
```

9.4 LCD 电路

在 LCD 电路中，SN8P1927 内置 200K 欧姆的分压电阻。用户可在 VLCD/V2/V1/VSS 之间添加电阻以获得更大的驱动电流。

注：V1，V2 只在 Dice 形式下存在，在封装片上是不存在的。

LCD 电路，1/4 占空比，1/3 偏压

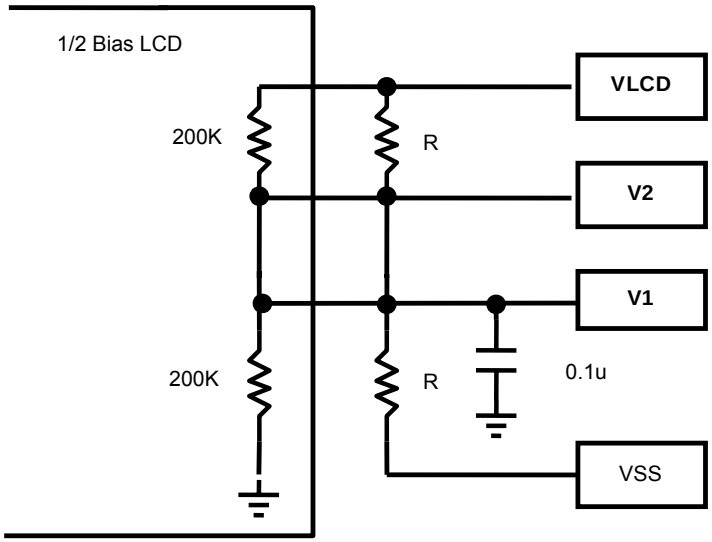


$$\text{LCD Current consumption} = \frac{\text{VLCD}}{\left(\frac{200\text{K}\times\text{R}}{200\text{K}+\text{R}}\right)\times 3}。$$

注：使用外部电阻时，VLCD 的 LCD 电流功耗始终存在，即使在睡眠模式下也存在。

注：V1=1/3*VLCD、V2=2/3*VLCD。

LCD 电路，1/4 占空比，1/2 偏压：



$$\text{LCD Current consumption} = \frac{\text{VLCD}}{\left(\frac{200\text{K}\times\text{R}}{200\text{K}+\text{R}}\right)\times 2}$$

10 在线烧录（ISP）

10.1 概述

SN8P1927 具有在线烧录功能（ISP ROM），它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-ROMWRT，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 中的数据将被存入 ROMADRH/ROMADRL 中。

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	ROMADR14	ROMADR13	ROMADR12	ROMADR11	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。
0 = VPP 电压与 12.5V 相接，不能使用在线烧录；
1 = VPP 电压与 12.5V 相接，可以使用在线烧录。

✷ **注：使用宏指令@B0BTS1_FVPPCHK 和 @B0BTS0_FVPPCHK 可以检测 VPP 电压状态。**

ROMADR[14:0]：ISP ROM 烧录地址。
烧录 ROM 地址。

10.3 ROMDAH/ROMADL 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。
烧录 ROM 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	X	X	X	X	X	X	X	X

Bit[7:0] ROMCNT**[7:0]**：ISP ROM 烧录时间计数器。
ISP ROM 在线烧录烧录时间由 ROMCNT**[7:0]**位控制。
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 ROMDA[15:0]中数据烧入 ROMADR[14:0]中。

✷ **注 1：在线烧录时，需保持 VDD=5V。**
✷ **注 2：ROMWRT 执行后，程序中还应该再加一条 NOP 指令以延时。**

10.5 ISP ROM 操作范例

➢ **例 ：**

```
; 保留在线烧录 ROM 空间为 0FFFFH。
ORG             0100H
@CALDATA:
                DW              0FFFFH
                ……..
;烧录数据 0AA55H 存入地址@CALDATA。
                MOV             A, #@CALDATA$L
                B0MOV           ROMADRL, A      ;低字节地址存入 ROMADRL。
                MOV             A, #@CALDATA$H
                B0MOV           ROMADRH, A      ;高字节地址存入 ROMADRH。
                MOV             A, #0X55
                B0MOV           ROMDAL, A       ;低字节数据存入 ROMDAL。
                MOV             A, #0XAA
                B0MOV           ROMDAH, A       ;高字节数据存入 ROMDAH。
;VPP 电压检测。
                @B0BTS1_ FVPPCHK                ;检查 VPP 电压是否为 12.5V。
                JMP             $-1             ;如果 VPP 电压未达到 12.5V，继续等待。

;设置烧录时间控制计数器的初始值，并开始 ISP ROM。
@ROM_WRT:       MOV             A,#6
                B0MOV           ROMCNT,A
                ROMWRT                          ;在线烧录。
                NOP                             ;NOP 延迟。
                NOP                             ;NOP 延迟。
                NOP                             ;NOP 延迟。
;VPP 电压检测。
                @B0BTS0_ FVPPCHK                ;检测 VPP 电压是否为 VDD。
                JMP             $-1             ;如果 VPP 仍然为 12.5V，继续等待。
;检测烧录数据。
                B0MOV           Z, #@CALDATA$L
                B0MOV           Y, #@CALDATA$H
                MOVC                            ;在线烧录的数据存入 A 和 R 寄存器。

                CMPRS           A,#0x55
                JMP             @WRT_ERR
                B0MOV           A, R
                CMPRS           A,#0xAA
                JMP             @WRT_ERR        ;检查在线烧录数据方向。
                ……..
```

11 Charge-Pump，PGIA 和 ADC

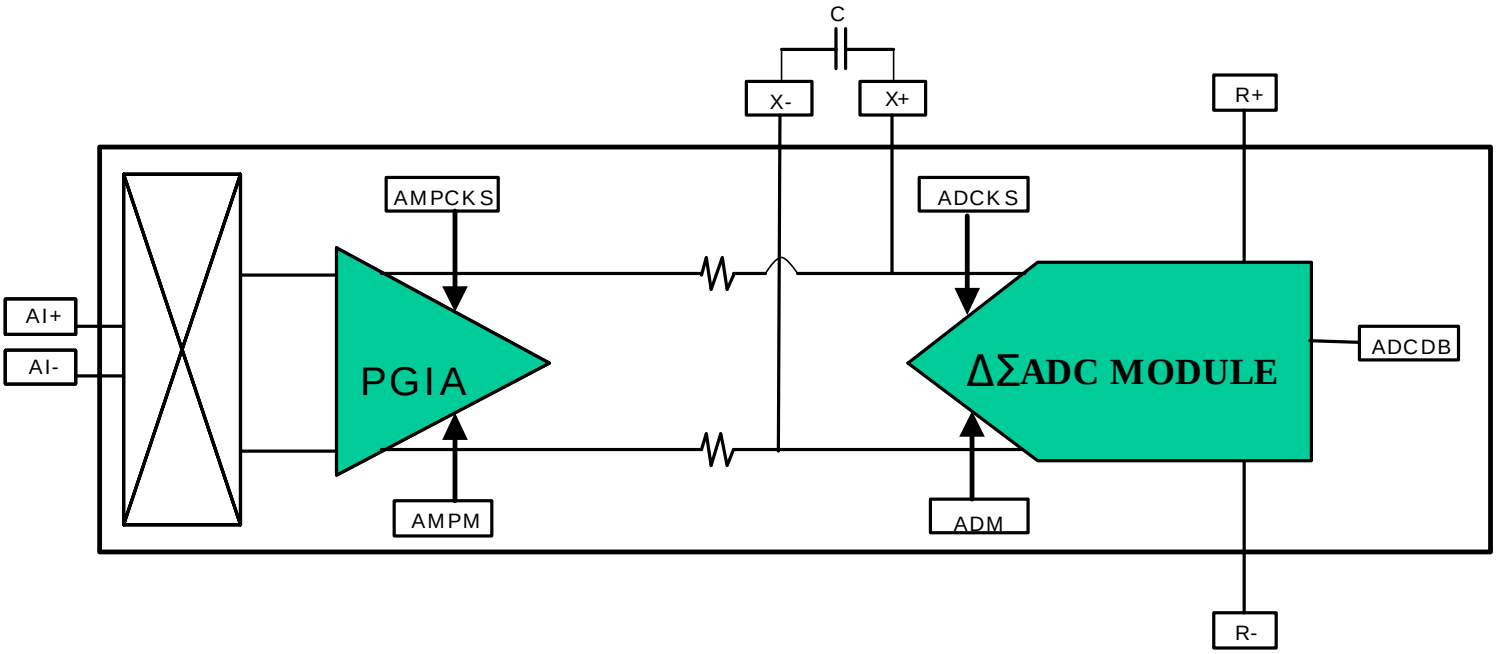
11.1 概述

SN8P1927 内置升压/稳压器 Charge-Pump/Regulator（CPR），从 AVDDR 输出稳定的 3.8V，而从 AVE+输出稳定的 2.0V/1.5V，最大的驱动电流为 10mA。CPR 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。SN8P1927 具有完整的 ΔΣ 模拟数字转换器（ADC），具有 16 位性能，高达 62500 阶的分辨率。ADC 有 2 个不同的输入信道模式：（1）1 个差分输入；（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、12.5x、50x、100x 和 200x 五种增益。

11.2 模拟信号输入

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[1:0]]（参考电压选择）位和 AMPM 寄存器的 GS[2:0]（增益选择）位。



ADC 模块的结构简图

✷ **注 1：低通滤波器（R、R 和 C）可以滤除 PGIA Chopper Clock 带来的干扰。**
✷ **注 2：电容 C 的建议值是 0.01uF，电容 C 要尽可能的靠近单片机。**

11.3 Charge Pump / Regulator (CPR)

SN8P1927 内置一个 CPR 电路，提供 3.8V 的电源（来自 AVDDR）和 2.0V/1.5V（来自 AVE+）电源，最大驱动电流 10mA。寄存器 CPM 可以控制 CPR 的工作状态和工作模式，而寄存器 CPCKS 决定 CPR 的工作频率，建议值为 4KMz。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 端的电压是来自 AVDDCP，通过稳压器稳定为 3.8V。另外，在设置 CPRENB 为高电平后，最少要等待 10ms，等待 VDD 稳定。

11.3.1 CPM-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	CPSTS	CPAUTO	CPON	CPRENB
读/写	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W
复位后	0	0	0	0	0	0	0	0

Bit0: **CPRENB**：Charge Pump / Regulator 功能使能控制位。
0 = 关闭 charge pump / regulator；
1 = 打开 charge pump / regulator。

Bit1: **CPON**：Change Pump 始终开启（ON）功能控制位（CPRENB 必须置“1”）。
0 = 由 CPAUTO 位控制 Charge Pump 的 On / Off 状态；
1 = 始终开启 charge pump regulator。

Bit2: **CPAUTO**：Charge Pump Auto 模式功能控制位。
0 = 禁止 charge pump auto 模式；
1 = 使能 charge pump auto 模式。

Bit3: **CPSTS**：Charge-Pump 在 AUTO 模式下的状态位（仅在 CPAUTO = 1 时有效）。
0 = Charge-Pump 在 Auto 模式下为 OFF 状态；
1 = Charge-Pump 在 Auto 模式下为 ON 状态。

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 = AVE+输出 2.0V。

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 3.8V 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 1.2V。

✷ **注 1：在置 CPRENB = 1 后，请延迟 30ms 等待电压稳定。**
✷ **注 2：在打开 Charge Pump 后，AVDDR（包括 PGIA 和 ADC）和 AVE+端负载的功耗都是没有开启时的 2 倍。**
✷ **注 3：来自 AVE+（如 Load cell）的所有功耗不是 2 倍。**
✷ **注 4：在打开 Charge pump / Regulator 之前，必须先打开 Band Gap（BGRENB = 1）。**
✷ **注 5：在打开 PGIA 和 ADC 前，必须先打开 Band Gap（BGRENB = 1）,ACM（ACMENB = 1）和 AVDDR（AVDDRENB）。**
✷ **注 6：CPR、PGIA 和 ADC 都可以在低速模式下工作，但是 CPCKS、AMPCKS 寄存器的值必须重新设置。**

CPRENB、CPON 和 CPAUTO 位是 Charge-Pump 工作模式的控制位，通过它们，Charge-Pump 设置为 OFF、Always ON 和 Auto 模式。

CPRENB	CPON	CPAUTO	AVDDRENB	Charge-Pump 状态	Regulator状态	CPSTS	AVDDR	PGIA, ADC功能
0	X	X	0	OFF	OFF	N/A	0V	无效
1	0	0	1	OFF	ON	N/A	参照注1	参照注1
1	0	1	1	Auto Mode	ON	0/1	3.8V	有效
1	1	0	1	ON	ON	N/A	3.8V	有效

在 Auto Mode 下，Charge-Pump ON/OFF 的工作模式由 VDD 电压决定。

Auto-Mode 说明：

CPRENB	CPON	CPAUTO	AVDDRENB	VDD	Charge-Pump状态	CPSTS	Regulator状态	AVDDR输出	PGIA, ADC功能
1	0	1	1	>4.1V	OFF	0	ON	3.8V	有效
				≦4.1V	ON	1	ON	3.8V	有效

✷ **注：当 Charge-Pump 处于 OFF 状态，Regulator 处于 ON 状态时，VDD 的电压必须高于 4.1V 以保证 AVDDR 输出正常的压，否则 PGIA 和 ADC 的功能会不正常。**

CPRENB	CPON	CPAUTO	AVDDRENB	VDD	Charge-Pump状态	Regulator状态	AVDDR输出	PGIA, ADC功能
1	0	0	1	>4.1V	OFF	ON	3.8V	有效
				≦4.1V	OFF	ON	**VDD**	无效

✷ **注 1：一般应用时，强烈建议设置 CP 为 Auto 模式（CPAUTO = 1）。**
✷ **注 2：如果 VDD 为 5V，不要将 Charge-Pump 设置为 Always ON。**
✷ **注 3：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. **Charge pump /Regulator；**
2. **PGIA 功能；**
3. **16 位 ADC 功能；**
4. **低电压检测功能。**

11.3.2 CPCKS-Charge Pump 时钟寄存器

096H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CPCKS	-	-	-	-	CPCKS3	CPCKS2	CPCKS1	CPCKS0
读/写	-	-	-	-	W	W	W	W
复位后	-	-	-	-	0	0	0	0

CPCKS [3:0]寄存器设置 Charge-Pump 的工作时钟，建议在普通模式下，Charge-Pump 的时钟设定为 13～15.6KHz；在低速模式下设定为 2KHz。

Charge-Pump 时钟 = Fcpu / 4 / (2^CPCKS[3:0])

在不同的 Fosc 频率下，CPCKS[3:0]寄存器的值的设置见参考下表：

CPCKS3	CPCKS2	CPCKS1	CPCKS0	High Clock			
				2M	3.58M	4M/IHRC	8M
0	0	0	0	125K	223.75K	250K	500K
0	0	0	1	62.5K	111.88K	125K	250K
0	0	1	0	31.25K	55.94K	62.5K	125K
0	0	1	1	15.625K	27.97K	31.25K	62.5K
0	1	0	0	7.8125K	13.985K	15.625K	31.25K
0	1	0	1	3.90625K	6.99K	7.8125K	15.625K
0	1	1	0	1.953215K	3.495K	3.90625K	7.8125K
0	1	1	1	0.976K	1.75K	1.953215K	3.90625K
1	0	0	0	0.488K	0.875K	0.976K	1.953215K
1	0	0	1	0.244K	0.438K	0.488K	0.976K
1	0	1	0	0.122K	0.219K	0.244K	0.488K
1	0	1	1	0.61K	0.11K	0.122K	0.244K
1	1	0	0	0.3K	0.055K	0.061K	0.122K
1	1	0	1	0.15K	0.028K	0.03K	0.61K
1	1	1	0	0.075K	0.014K	0.015K	0.3K
1	1	1	1	0.037K	0.007K	0.008K	0.15K

- **注 1：在打开 Charge Pump 时，建议先将 Charge Pump 时钟设为“1011”以避免 VDD 跌落。**
- **注 2：在一般的应用中，CP 工作时钟的建议值是：普通模式下 13~15KHz；低速模式（内部低速时钟模式）下 2KHz。**
- **注 3：Charge Pump 的时钟越快，AVE+的负载能力更强。**

➢ **例：设置 Charge-Pump（Fosc = 4M X'tal）**

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

        MOV         A, #00001011b
        XB0MOV      CPCKS, A    ; 设置 CPCKS 为最低的时钟以避免 VDD 跌落。

        MOV         A, #00100100B ;
        XB0MOV      CPM, A      ; 设置 AVE+=3.0V，CP 为 Auto 模式并在使能 Charge Pump 之前禁
                                ; 止 AVDDR、AVE+和 ACM 的电压。
@CP_Enable:
        XB0BSET     FCPRENB     ; 使能 Charge-Pump。
        CALL        @Wait_200ms ; 延迟 200ms 等待 Charge-Pump 稳定。

        MOV         A, #0000100b
        XB0MOV      CPCKS, A    ; 设置 CPCKS 为 15.6K，负载能力为 10mA。
        CALL        @Wait_100ms ; 延迟 5ms 等待 ACM 电压稳定。

@ACM_Enable:
        XB0BSET     FACMENB     ; 设置 ACM 电压 = 1.2V。
        CALL        @Wait_5ms   ; 延迟 5ms 等待 ACM 电压稳定。

@AVDDR_Enable:
        XB0BSET     FAVDDRENB   ; 设置 AVDDR 电压 = 3.8V。
        CALL        @Wait_50ms  ; 延迟 50ms 等待 AVDDR 稳定。

@AVE_Enable:
        XB0BSET     FAVENB      ; 设置 AVE+电压 = 3.0V/1.5V。
        CALL        @Wait_50ms  ; 延迟 50ms 等待 AVE+稳定。
        …
        …
```

✷ **注 1：当使用单颗 CR2032 电池供电时，Charge-Pump 应该延迟 200ms 或者 100ms 以避免 VDD 跌落；如果使用 AA 或者 AAA 干电池供电时，延迟时间应该少于 50ms。**

11.4 PGIA –可编程增益放大器

SN8P1927 内建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、12.5x、50x、100x 和 200x 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。

11.4.1 AMPM-放大器工作模式寄存器

090H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
AMPM	-	BGRENB	FDS1	FDS0	GS2	GS1	GS0	AMPENB
读/写	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位后	-	0	0	0	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	12.5
001	50
010	100
011	200
100,101,110	保留
111	1

✷ **注：当选择增益 1x 时可以禁止 PGIA（AMPENB = 0）以省电。**

Bit[5:4] **FDS [1:0]**：Chopper 低频设置位。

✷ **注：在所有的应用中请设置 FDS[1:0] = “11”。**

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。

✷ **注 1：在使能下列功能之前必须先使能 Band Gap 参考电压（FBGRENB）：**
1. **Charge pump /Regulator；**
2. **PGIA 功能；**
3. **16 位 ADC 功能；**
4. **电池低电压检测功能。**

✷ **注 2：PGIA 不能在低速模式下工作，除了增益选择 1x 外。**

11.4.2 AMPCKS- 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：1.95KHz @4MHz，1.74KHz @3.58MHz。

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	High Clock			
			2M	3.58M	4M/IHRC	8M
0	0	0	15.625K	27.968K	31.25K	62.5K
0	0	1	7.8125K	13.98K	15.625K	31.25K
0	1	0	3.90625K	6.99K	7.8125K	15.625K
0	1	1	1.953125K	3.49K	3.90625K	7.8125K
1	0	0	976Hz	1.748K	1.953125K	3.90625K
1	0	1	488Hz	874Hz	976Hz	1.953125K
1	1	0	244Hz	437Hz	488Hz	976Hz
1	1	1	122Hz	218Hz	244Hz	488Hz

✷ **注：在一般应用时，PGIA Chopper 的时钟应该设置为~2KHz，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。**

11.4.3 AMPCHS-PGIA 通道选择

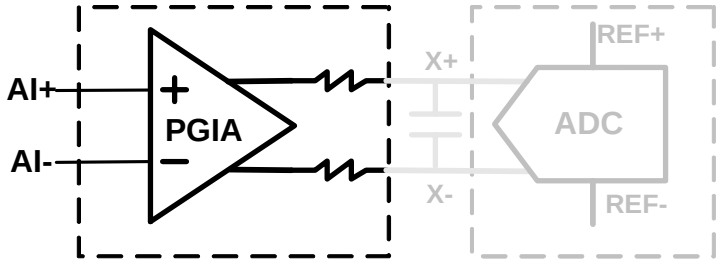
091H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
AMPCHS	-	-	-	-	-	CHS2	CHS1	CHS0
	-	-	-	-	-	R/W	R/W	R/W

CHS [2:0]：PGIA 通道选择

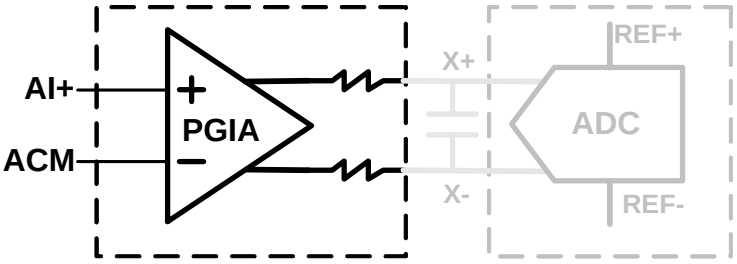
CHS [2:0]	选择通道	V (X+, X-) 输出	输入信号的类型
000	AI+, AI-	V (AI+, AI-) × PGIA 增益	差分输入
001	AI+, ACM	V (AI+, ACM) × PGIA 增益	单端输入
010	AI-, ACM	V (AI-, ACM) × PGIA 增益	单端输入
011	ACM, ACM	V (ACM, ACM) × PGIA 增益	输入短路
100	保留	-	-
101	温度传感器	V (VTS, 0.4V) × 1	N/A
110,111			

- **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。**
- **注 4：当禁止 CPR 或者系统停止运行时，模拟信号的输入引脚都必须置 0（0V，包括 AI+、AI-、X+、X-、R+和 R-），否则这些引脚会造成漏电。**

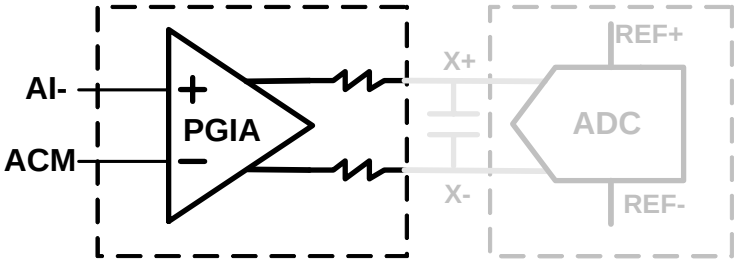
AMPCHS[2:0]=”000”



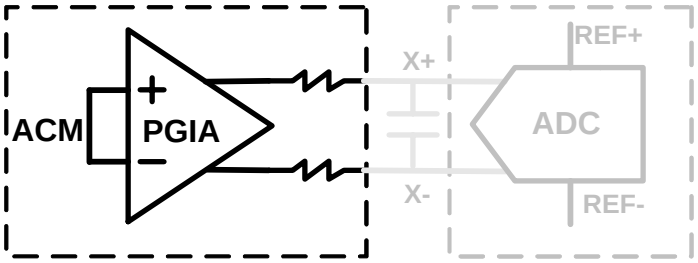
AMPCHS[2:0]=”001”



AMPCHS[2:0]=”010”

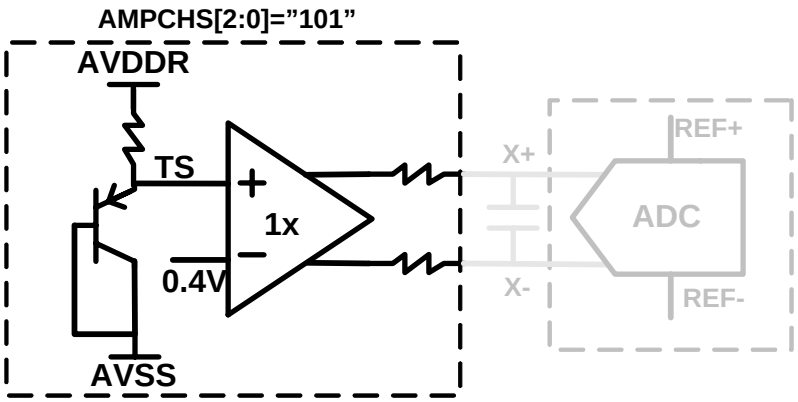


AMPCHS[2:0]=”011”



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，SN8P1927 内置了一个温度传感器（TS）来测量工作环境温度。通过不同的通道达到测量环境温度的目的。



- 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 0.4V。
- 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 25C 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	0.815V	0.8V	16211
25	0.800V	0.8V	15625
35	0.785V	0.8V	15039

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。

        MOV       A, #00001011b
        XB0MOV    CPCKS, A      ; 设置 CPCKS 为最低时钟模式以避免 VDD 跌落。

        MOV       A, #00100100B ;
        XB0MOV    CPM, A        ; 设置 AVE+=2.0V，设置 CP 为自动模式并在使能 Charge Pump 之前
@CP_Enable:                     ; 禁止 AVDDR、AVE+、ACM 电压。
        XB0BSET   FCPRENB       ; 使能 Charge-Pump。
        CALL      @Wait_200ms   ; 延迟 200ms 等待 Charge-Pump 稳定。

        MOV       A, #0000100b
        XB0MOV    CPCKS, A      ; 设置 CPCKS 为 15.6K，负载能力为 10mA。
        CALL      @Wait_100ms   ; 延迟 100ms 等待 CPCKS 稳定。
@ACM_Enable:
        XB0BSET   FACMENB       ; 设置 ACM 电压为 1.2V。
        CALL      @Wait_5ms     ; 延迟 5ms 等待 ACM 稳定。
@AVDDR_Enable:
        XB0BSET   FAVDDRENB     ; 设置 AVDDR 电压为 3.8V。
        CALL      @Wait_50ms    ; 延迟 50ms 等待 AVDDR 稳定。
@AVE_Enable:
        XB0BSET   FAVENB        ; 设置 AVE+电压为 2.0V/1.5V。
        CALL      @Wait_50ms    ; 延迟 50ms 等待 AVE+稳定。
@PGIA_Init:
        MOV       A, #01110110B
        XB0MOV    AMPM, A       ; 使能 Band Gap，设置 FDS= 11、PGIA 增益 =200x。
        MOV       A, #00000100B
        XB0MOV    AMPCKS, A     ; 设置 AMPCKS = 100，PGIA 工作时钟 = 1.9K @ 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 200。
```

➢ **注 1：在 PGIA 工作之前使能 Charge-Pump/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, #01110110B
        XB0MOV    AMPM, A       ; 使能 Band Gap，设置 FDS= 11、PGIA 增益 =200x。
        MOV       A, #00000100B
        XB0MOV    AMPCKS, A     ; 设置 AMPCKS = 100，PGIA 工作时钟 = 1.9K @ 4M X'tal。
        MOV       A, #00000000B
        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 200。
@PGIA_Sensor:
        MOV       A, #01110111B ; 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=200x。
        MOV       A, #00000000B
        XB0MOV    AMPCHS, A     ; 选择 PGIA 作为差分输入通道。
        …                       ; V (X+, X-)输出 = V(AI+,AI-) x 200

@PGIA_TS:
        MOV       A, #01110001B ; 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, 0.4）x 1。
```

11.5 16 位 ADC

11.5.1 ADCM- ADC 模式寄存器

093H	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADCM	-	-	-	-	IRVS	RVS1	RVS0	ADCENB
读/写	-	-	-	-	R/W	R/W	R/W	R/W
复位后	-	-	-	-	0	0	0	0

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

Bit1: **RVS 0**：ADC 信号输入选择位。
0 = 信号从 X+、X-输入；
1 = 信号从 VDD 输入。

Bit2: **RVS 1**：ADC 参考电压选择位。
0 = ADC 参考电压来自外部参考源 R+、R-；
1 = ADC 参考电压来自内部参考源。

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+ = 2.0V 时，V(REF+, REF-) = 0.4V）；
1 = 内部参考电压 V(REF+,REF-)为 AVE+/2.5（当 AVE+ = 2.0V 时，V(REF+, REF-) = 0.8V）。

IRVS	RVS1	RVS0	AD参考大于		AD输入通道		备注
			REF+	REF-	ADCIN+	ADCIN-	
X	0	0	R+	R-	X+	X-	V (X+, X-) < V (R+, R-)
0	1	0	0.8V	0.4V			V (X+, X-) < 0.4V (AVE+=2.0V)
0	1	0	0.6V	0.3V			V (X+, X-) < 0.3V (AVE+=1.5V)
1	1	0	1.2V	0.4V			V (X+, X-) < 0.8V (AVE+=2.0V)
1	1	0	0.9V	0.3V			V (X+, X-) < 0.6V (AVE+=1.5V)
X	0	1	R+	R-	VDD *3/16	VDD* 2/16	ADC 输入 = 1/16 VDD 电池监控 (AVE+=2.0V)
0	1	1	0.8V	0.4V			
1	1	1	1.2V	0.4V			

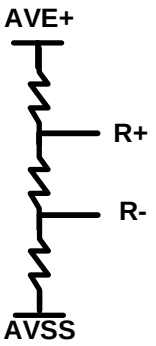
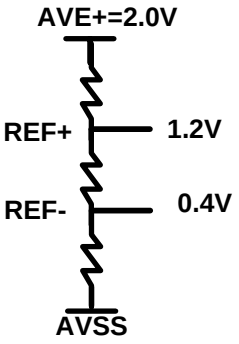
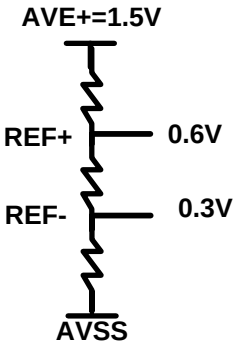
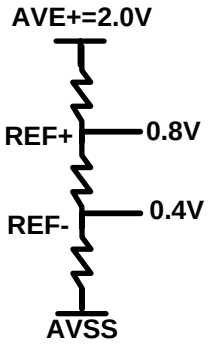
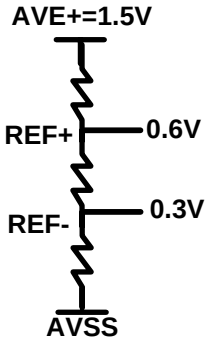
✷ **注 1：AD 转换的数据存放在 ADCDH 和 ADCDL 寄存器中，其中 ADCB15 是 ADC 数据的符号位。关于 AD 转换数据值的计算方法请参考如下公式：**
✷ **注 2：内部参考电压除以 AVE+，因此所得到的电压值随 AVE+（2.0V/1.5V）的不同而不同。**

$$(ADCIN+) > (ADCIN-) \Rightarrow ADCConversionData = +\frac{(ADCIN+)-(ADCIN-)}{(REF+)-(REF-)} X\,31250$$

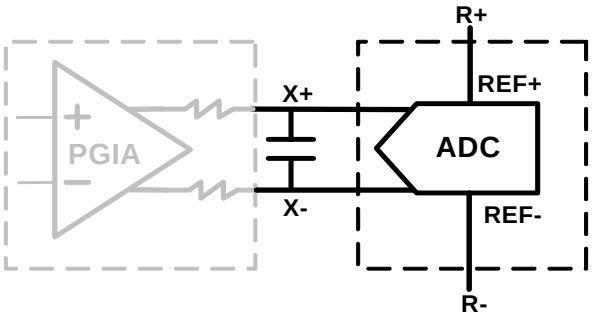
$$(ADCIN+) < (ADCIN-) \Rightarrow ADCConversionData = -\frac{(ADCIN+)-(ADCIN-)}{(REF+)-(REF-)} X\,31250$$

✷ **注 1：内部 1.2V、0.8V 和 0.4V 参考电压来自 Band Gap 参考电压。**

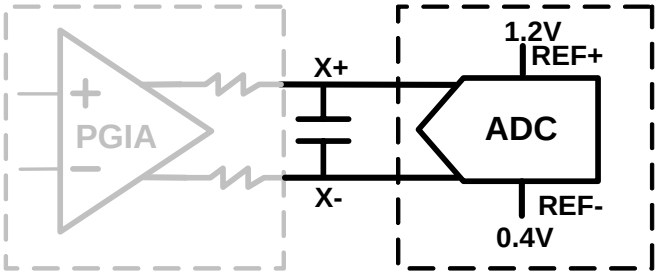
外部参考电路和内部参考电路列表如下：

外部参考电路	内部参考电路			
RVS1=0	RVS1=1,			
	IRVS=1, AVE+=2.0V	IRVS=1, AVE+=1.5V	IRVS=0,AVE+=2.0V	IRVS=0,AVE+=1.5V
				

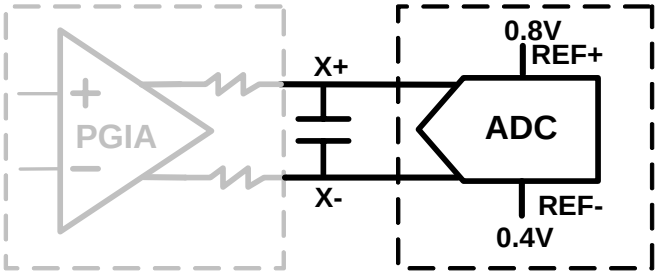
ADCM=#xxxxx00xB，V(REF+, REF-) = V(R+, R-)，ADC 参考电压来自外部 R+、R-：



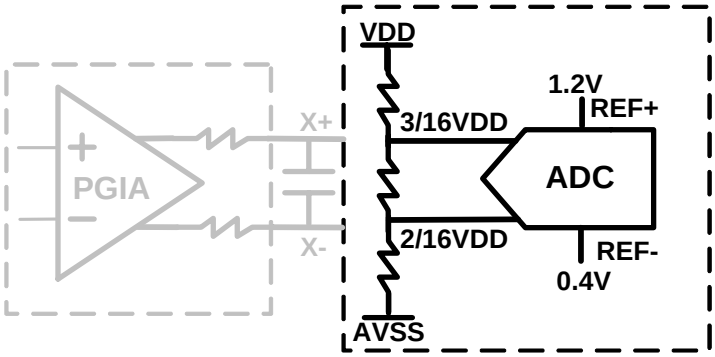
ADCM=#xxxx110xB，V（REF+, REF-） = V（1.2V, 0.4V）=0.8V（AVE+=2.0V），ADC 参考电压来自内部 1.2V/0.4V：



ADCM=#xxxx010xB，V（REF+, REF-） = V（0.8V, 0.4V）=0.4V（AVE+ = 2.0V），ADC 参考电压来自内部 0.8V/0.4V：



ADCM=#xxxx111xB，V（REF+, REF-） = V（1.2V, 0.4V）=0.8V（AVE+ = 2.0V），ADC 参考电压来自内部 1.2V/0.4V，ADC 的输出结果是电源电压分压之后的值。



11.5.2 ADCKS- 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

ADCKS [7:0]寄存器设置 ADC 的工作时钟，建议值是 100K Hz。

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

ADC 时钟频率 = (Fosc / (256-ADCKS [7:0]))/2

ADCKS [7:0]	F_{osc}	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]	F_{osc}	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。**

11.5.3 ADCDL- ADC 低字节数据寄存器

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

11.5.4 ADCDH- ADC 高字节数据寄存器

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

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

.

- 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 7A12H。**
- 注 4：**ADC 输出的最小负值是 85EEH。**
- 注 5：**由于 ADC 的设计限制，ADC 的线形范围为+28125~-28125（十进制），故 ADC 的值必须在该范围内。**

ADC 数据（十六进制）	十进制
7A12H	31250
…	…
4000H	16384
…	…
1000H	4096
…	…
0002H	2
0001H	1
0000H	0
0FFFFH	-1
0FFFEH	-2
…	…
xF000H	-4096
…	…
0C000H	-16384
…	…
85EEH	-31250

11.5.5 DFM-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	-	-	-	-	-	WRS0	-	DRDY
	-	-	-	-	-	R/W	-	R/W

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。

Bit2 **WRS0**：ADC 输出频率选择位。

WRS0	输出 Word Rate		
	ADC 时钟频率 = 200K	ADC 时钟频率 = 100K	ADC 时钟频率 = 80K
0	50Hz	25 Hz	20 Hz
1	25Hz	12.5 Hz	10 Hz

- **注 1：当输出 word rate = 25Hz，AC 50Hz 杂波会被滤除掉。**
- **注 2：当输出 word rate = 20Hz，AC 60Hz 杂波会被滤除掉。**
- **注 3：当输出 word rate = 10Hz，AC 50Hz 和 AC 60Hz 杂波都会被滤除掉。**
- **注 4：在读取 ADC 数据后要将 DRDY 位清零，否则它会一直保持高电平。**
- **注 5：调节 ADC 的时钟频率（ADCKS）和 WRS0，可以得到合适的 word rate。**

➢ **例：设置 Charge-Pump、PGIA 和 ADC（Fosc = 4M X'tal）。**

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

        MOV         A, #00001011b
        XB0MOV      CPCKS, A    ; 设置 CPCKS 为最低时钟工作模式以避免 VDD 跌落。

        MOV         A, #00100100B ;
        XB0MOV      CPM, A      ; 设置 AVE+=2.0V，CP 为自动模式并在使能 Charge Pump 之前禁止
                                AVDDR、AVE+和 ACM 电压。
@CP_Enable:
        XB0BSET     FCPRENB     ; 使能 Charge-Pump。
        CALL        @Wait_200ms ; 延迟 200ms 等待 Charge-Pump 稳定。

        MOV         A, #0000100b
        XB0MOV      CPCKS, A    ; 设置 CPCKS = 15.6K，负载能力为 10mA。
        CALL        @Wait_100ms ; 延迟 100ms 等待 CPCKS 稳定。
@ACM_Enable:
        XB0BSET     FACMENB     ; 设置 ACM 电压 =1.2V。
        CALL        @Wait_5ms   ; 延迟 5ms 等待 ACM 稳定。
@AVDDR_Enable:
        XB0BSET     FAVDDRENB   ; 设置 AVDDR 电压 =3.8V。
        CALL        @Wait_50ms  ; 延迟 50ms 等待 AVDDR 稳定。
@AVE_Enable:
        XB0BSET     FAVENB      ; 设置 AVE+电压 =2.0V/1.5V。
        CALL        @Wait_50ms  ; 延迟 50ms 等待 AVE+稳定。
@PGIA_Init:
        MOV         A, #01110110B
        XB0MOV      AMPM, A     ; 使能 Band Gap，设置 FDS=11 和 PGIA 增益 =200。
        MOV         A, #00000100B
        XB0MOV      AMPCKS, A   ; 设置 AMPCKS = 100，PGIA 工作时钟 = 1.9K @ 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 200。
@ADC_Init:
        MOV         A, #00000110B
        XB0MOV      ADCM, A     ; 选择 ADC 的参考电压 = V(R+, R-)。
        MOV         A, #0236
```

```
                XB0MOV      ADCKS, A        ; 设置 ADCKS = 236，ADC 的工作时钟= 100K @ 4M X'tal。
                MOV         A, #00H
                XB0MOV      DFM, A          ; 设置 ADC 为连续工作模式，WRS0 = 0。
@ADC_Enable:                                ;  ADC rate =25 Hz。
                XB0BSET     FADCENB         ; 使能 ADC 功能。
@ADC_Wait:
                XB0BTS1     FDRDY           ; 检查 ADC 是否输出新值。
                JMP         @ADC_Wait       ; 等待 DRDY = 1。
@ADC_Read:                                  ; 输出 ADC 数据。
                XB0BCLR     FDRDY
                XB0MOV      A, ADCDH
                B0MOV       Data_H_Buf, A   ; 把 ADC 的高字节数据存放在数据缓存器中。
                XB0MOV      A, ADCDL
                B0MOV       Data_L_Buf, A   ; 把 ADC 的低字节数据存放在数据缓存器中。
                …
```

✷ **注：请先设置 ADC 的相关寄存器，再使能 ADC 功能。**

➢ **例：设置 ADC 参考电压。**

```
@ADC_Init:
                MOV         A, #00000110B
                XB0MOV      ADCM, A         ; 选择 ADC 参考电压 = V(R+, R-)。
                MOV         A, #0236
                XB0MOV      ADCKS, A        ; 设置 ADCKS = 236，ADC 工作时钟= 100K @ 4M X'tal。
                MOV         A, #00H
                XB0MOV      DFM, A          ; 设置 ADC 为连续工作模式，WRS0 = "0" 25 Hz。
@ADC_Enable:
                XB0BSET     FADCENB         ; 使能 ADC 功能。
@ADC_Wait:
                XB0BTS1     FDRDY           ; 检查 ADC 是否输出新值。
                JMP         @ADC_Wait       ; 等待 DRDY = 1。
@ADC_Read:                                  ; 输出 ADC 数据。
                XB0BCLR     FDRDY
                XB0MOV      A, ADCDH
                B0MOV       Data_H_Buf, A   ; 把 ADC 的高字节数据存放在数据缓存器中。
                XB0MOV      A, ADCDL
                B0MOV       Data_L_Buf, A   ; 把 ADC 的低字节数据存放在数据缓存器中。
                …
@ADC_RVS1:
                MOV         A, #00011011B   ; 在设置参考电压是不用禁止 ADC。
                XB0MOV      ADCM, A         ; 选择 ADC 的参考电压来自内部 V(1.2V,0.4V)。
@@:
                XB0BTS1     FDRDY           ; 检查 ADC 是否输出新值。
                JMP         @B              ; 等待 DRDY = 1。
                                            ; 输出 ADC 数据。
                XB0BCLR     FDRDY
                XB0MOV      A, ADCDH
                B0MOV       Data_H_Buf, A   ; 把 ADC 的高字节数据存放在数据缓存器中。
                XB0MOV      A, ADCDL
                B0MOV       Data_L_Buf, A   ; 把 ADC 的低字节数据存放在数据缓存器中。
                …
@ADC_RVS2:
                MOV         A, #00011001B   ; 设置参考电压时不用禁止 ADC。
                XBMOV       ADCM, A         ; 设置 ADC 用作电压测量。
@@:
                XB0BTS1     FDRDY           ; 检查 ADC 是否输出新值。
                JMP         @B              ; 等待 DRDY = 1。
                                            ; 输出 ADC 数据。
                XB0BCLR     FDRDY
                XB0MOV      A, ADCDH
                B0MOV       Data_H_Buf, A   ; 把 ADC 的高字节数据存放在数据缓存器中。
                XB0MOV      A, ADCDL
                B0MOV       Data_L_Buf, A   ; 把 ADC 的低字节数据存放在数据缓存器中。
                …
```

11.5.6 LBTM -电池低电压检测寄存器

SN8P1927 提供 2 种方式测量电源电压：一种是通过 16 位的 ADC，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 P4.2，与 ACM（1.2V）进行比较，比较结果在 LBTO 位。

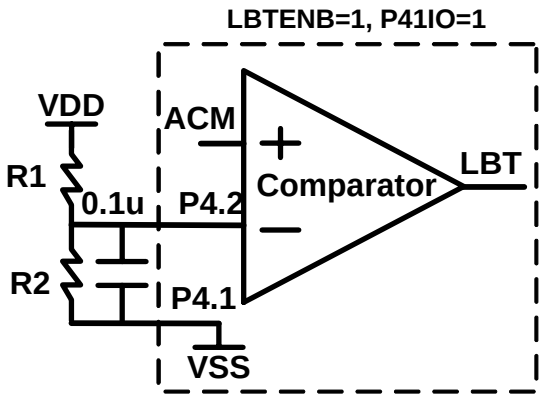
09AH	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LBTM	-	-	-	-	-	LBTO	P41IO	LBTENB
读/写	-	-	-	-	-	R	R/W	R/W
复位后	-	-	-	-	-	0	0	0

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。

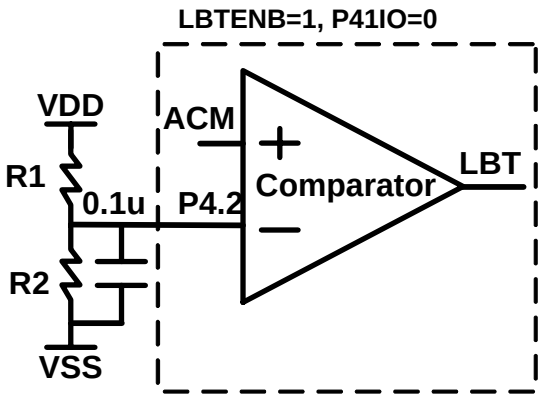
Bit1: **P41IO**：P4.1 输入/LBT 功能控制位。
0 = P41 为输入口；
1 = P41 为 LBT 功能。

Bit2: **LBTO**：电池低电压检测输出位。
0 = P4.2/LBT 电压高于 ACM(1.2V)；
1 = P4.2/LBT 电压低于 ACM(1.2V)。

下图是 LBT 应用的两种电路连接方式：一种使用 P4.2 和 P4.1，这样在睡眠模式下不会产生漏电流；另外一种只使用 P4.2，这种方式会在省电模式下产生一些漏电流，但是可以把 P4.1 作为输入口用。



P4.1 设为 LBT 功能时，在睡眠模式下没有漏电流



P4.1 设为输入模式时，在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。**

11.5.7 模拟部分电路设置和应用

SN8P1927 主要应用于 DC 测量，如体重秤、压力测量等。为了防止开启 Charge Pump 而造成 VDD 跌落，同时兼顾材料成本，所以不同的应用方案，模拟部分的电容配置不一样。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。

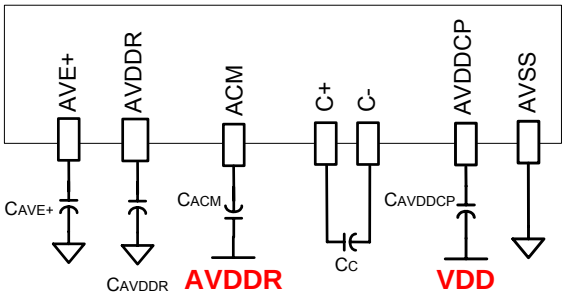
电容列表：

电源类型	AI+	AI-	X+/X-	R+/R-	ACM	AVDDR	AVE+	AVDDCP	C+/C-	VDD(Pin23)	VDD(Pin28)
	C_{AI+}	C_{AI-}	C_X	C_R	C_{ACM}	C_{AVDDR}	C_{AVE+}	C_{AVDDCP}	C_C	C_{AVDD}	C_{DVDD}
CR2032 (2.4~3V)	0.1uF	0.1uF	0.01uF	0.1uF	1uF	1uF	2.2uF	10uF	1uF	10uF	0.1uF
CR2032 ((4.4~6V))	0.1uF	0.1uF	0.01uF	0.1uF	1uF	1uF	2.2uF	No	No	10uF	0.1uF
AA/AAA Bat.(2.4~3V)	0.1uF	0.1uF	0.01uF	0.1uF	1uF	1uF	4.7uF	10uF	1uF	10uF	0.1uF
AA/AAA Bat.(4.4~6V)	0.1uF	0.1uF	0.01uF	0.1uF	1uF	1uF	4.7uF	No	No	10uF	0.1uF
外部 5V 电源	0.1uF	0.1uF	0.01uF	0.1uF	1uF	1uF	4.7uF	No	No	10uF	0.1uF

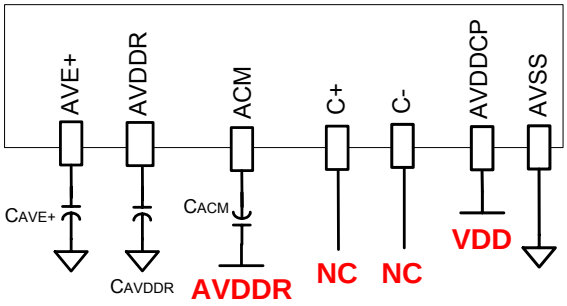
- 注 1：当单片机由 CR2032 电池供电时，AVE+端的负载不要超过 3mA。即当 AVE+=2.0V 时，Loadcell 的阻值不要低于 1K。
- 注 2：当单片机由 AA/AAA 干电池供电时，AVE+的负载可以达到 10mA，即当 AVE+=2.0V 时，loadcell 的阻值可以低至 330Ω。
- 注 3：当单片机供电电压恒高于 4.2V 时，可以将 ChargePump 设为自动模式或者关闭模式，这样 ChargePump 被关闭，AVDDR 和 AVE+端的负载电流不会加倍，AVDDCP 和 C+，C-之间的电容可以省去不接，将 AVDDCP 直接和 VDD 连接。

- 注 1：电容 C_{AVDDCP} 的正极应该和 AVDDCP 相连，负极则应该连接到 VDD。
- 注 2：电容 C_{ACM} 的正极应该和 AVDDR 相连，负极则应该连接到 ACM。

VDD = 2.4V~4.2V 时电容的连接方法：



VDD = 4.2V~5.5V 时电容的连接方法：



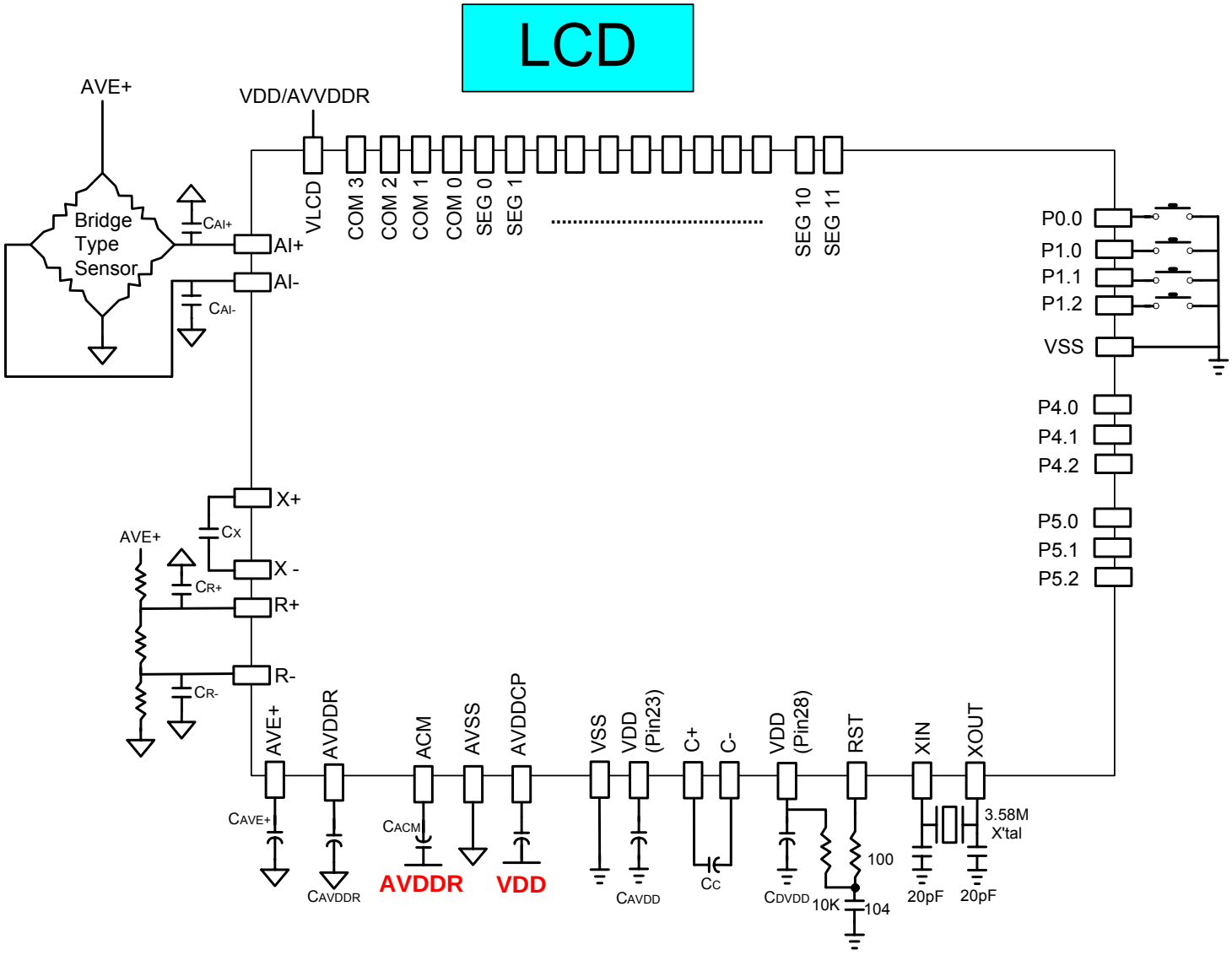
延迟时间

电源类型	Charge Pump 使能延迟时间		使能 ACM	使能 AVDDR	使能 AVE+
	Step1（PCKS=#00001011B）	Step2（CPCKS=#00000100B）			
CR2032 (2.4~3V)	200ms	100ms	5ms	50ms	50ms
CR2032 ((4.4~6V))	-	-	5ms	50ms	50ms
AA/AAA Bat.(2.4~3V)	100ms	50ms	5ms	50ms	50ms
AA/AAA Bat.(4.4~6V)	-	-	5ms	50ms	50ms
外部 5V 电压	-	-	5ms	50ms	50ms

- 注 1：在 CR2032 供电情况下，需要足够长的延迟时间，防止开启 Charge Pump 时 VDD 跌落；
- 注 2：当 VDD 恒高于 4.2V 时，可以将 Charge Pump 设置为自动或者禁止模式，这样可以关闭 Charge Pump；
- 注 3：在 AA/AAA 干电池供电情况下的延迟时间要比 CR2032 供电情况下的延迟时间短。

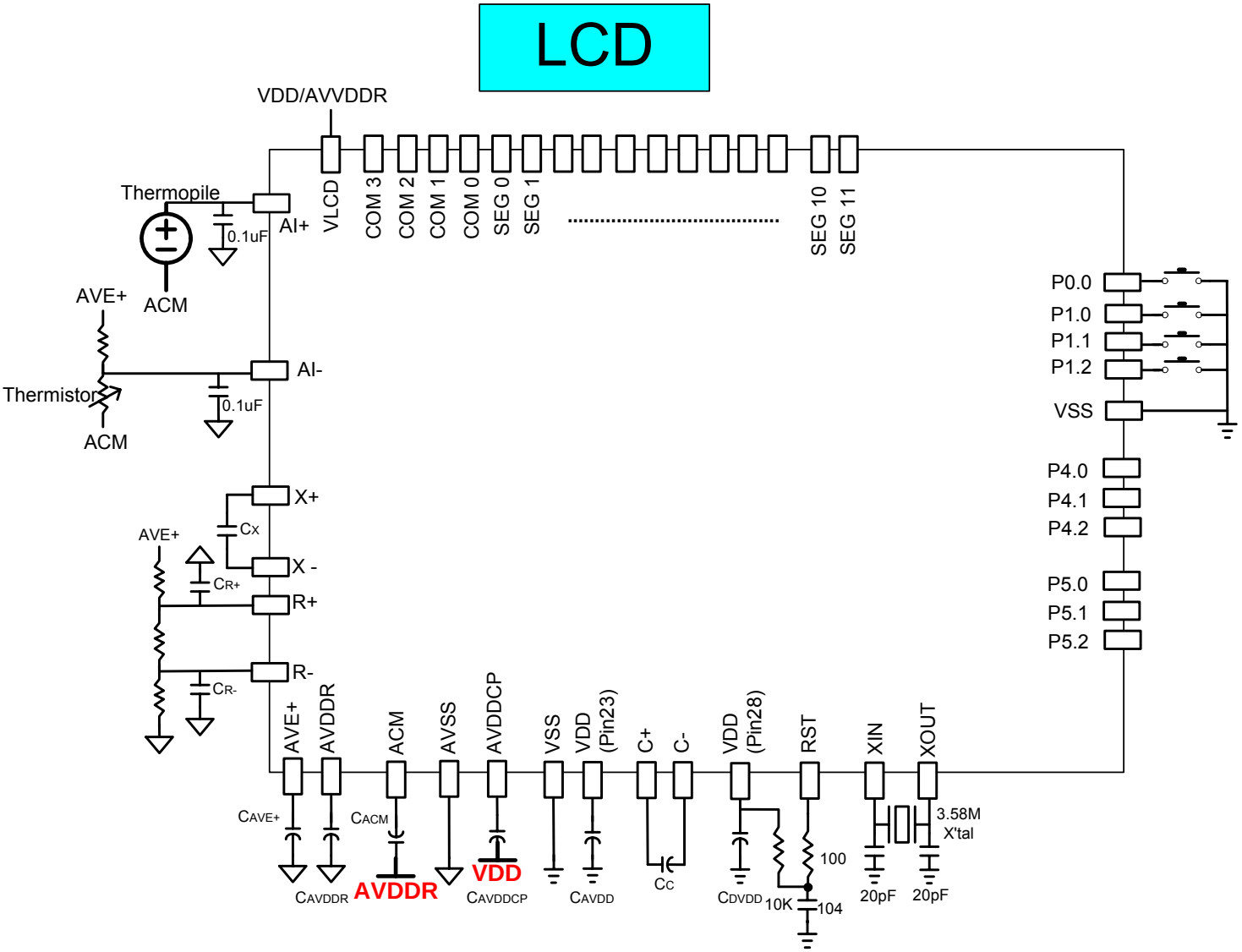
12 应用电路

12.1 电子秤（Load Cell）应用电路



✷ 注：电容的设置请参考 10.5.7。

12.2 温度计应用电路



✷ **注：电容的设置请参考 10.5.7。**

13 指令集

指令	指令格式		说明	C	DC	Z	周期
MOVE	MOV	A,M	A ← M	-	-	√	1
	MOV	M,A	M ← A	-	-	-	1
	B0MOV	A,M	A ← M (bank 0)	-	-	√	1
	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I（M 仅适用地址是 80H~87H 的系统寄存器，如 R、Y、Z、RBANK 和 PFLAG）	-	-	-	1
	XCH	A,M	A ←→M	-	-	-	1
	B0XCH	A,M	A ←→M (bank 0)	-	-	-	1
	MOVC		R, A ← ROM [Y,Z]	-	-	-	2
ARITHMETIC	ADC	A,M	A ← A + M + C，如果产生进位则 C = 1，否则 C = 0	√	√	√	1
	ADC	M,A	M ← A + M + C，如果产生进位则 C = 1，否则 C = 0	√	√	√	1
	ADD	A,M	A ← A + M，如果产生进位则 C = 1，否则 C = 0	√	√	√	1
	ADD	M,A	M ← A + M，如果产生进位则 C = 1，否则 C = 0	√	√	√	1
	B0ADD	M,A	M (bank 0) ← M (bank 0) + A，如果产生进位则 C = 1，否则 C = 0	√	√	√	1
	ADD	A,I	A ← A + I，如果产生进位则 C = 1，否则 C = 0	√	√	√	1
	SBC	A,M	A ← A - M - /C，如果产生借位则 C=0，否则 C=1	√	√	√	1
	SBC	M,A	M ← A - M - /C，如果产生借位则 C=0，否则 C=1	√	√	√	1
	SUB	A,M	A ← A - M，如果产生借位则 C=0，否则 C=1	√	√	√	1
	SUB	M,A	M ← A - M，如果产生借位则 C=0，否则 C=1	√	√	√	1
	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改为十进制格式	√	-	-	1
LOGIC	AND	A,M	A ← A 与 M	-	-	√	1
	AND	M,A	M ← A 与 M	-	-	√	1
	AND	A,I	A ← A 与 I	-	-	√	1
	OR	A,M	A ← A 或 M	-	-	√	1
	OR	M,A	M ← A 或 M	-	-	√	1
	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
PROCESS	SWAP	M	A (b3~b0, b7~b4) ←M(b7~b4, b3~b0)	-	-	-	1
	SWAPM	M	M(b3~b0, b7~b4) ← M(b7~b4, b3~b0)	-	-	-	1
	RRC	M	A ← M 带进位右移	√	-	-	1
	RRCM	M	M ← M 带进位右移	√	-	-	1
	RLC	M	A ← M 带进位左移	√	-	-	1
	RLCM	M	M ← M 带进位左移	√	-	-	1
	CLR	M	M ← 0	-	-	-	1
	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
BRANCH	CMPRS	A,I	比较，如果相等则跳过下一条指令 C 与 ZF 标志位可能受影响	√	-	√	1 + S
	CMPRS	A,M	比较，如果相等则跳过下一条指令 C 与 ZF 标志位可能受影响	√	-	√	1 + S
	INCS	M	A ← M + 1，如果 A = 0，则跳过下一条指令	-	-	-	1 + S
	INCMS	M	M ← M + 1，如果 M = 0，则跳过下一条指令	-	-	-	1 + S
	DECS	M	A ← M - 1，如果 A = 0，则跳过下一条指令	-	-	-	1 + S
	DECMS	M	M ← M - 1，如果 M = 0，则跳过下一条指令	-	-	-	1 + S
	BTS0	M.b	如果 M.b = 0，则跳过下一条指令	-	-	-	1 + S
	BTS1	M.b	如果 M.b = 1，则跳过下一条指令	-	-	-	1 + S
	B0BTS0	M.b	如果 M(bank 0).b = 0，则跳过下一条指令	-	-	-	1 + S
	B0BTS1	M.b	如果 M(bank 0).b = 1，则跳过下一条指令	-	-	-	1 + S
	JMP	d	跳转指令，PC15/14 ← RomPages1/0，PC13~PC0 ← d	-	-	-	2
	CALL	d	子程序调用指令，Stack ← PC15~PC0，PC15/14 ← RomPages1/0，PC13~PC0 ← d	-	-	-	2
MISC	RET		子程序跳出指令，PC ← Stack	-	-	-	2
	RETI		中断处理程序跳出指令，PC ← Stack，使能全局中断控制位	-	-	-	2
	NOP		空指令，无特别意义	-	-	-	1

14 开发工具

14.1 开发工具版本

14.1.1 ICE（在线仿真器）

- **SN8ICE 1K（S8KD-2）**：支持 SN8P1927 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 SN8P1927 EV-KIT 仿真模拟功能。**

✷ **注：SN8ICE2K 不支持 SN8P1927 系列的仿真。**

14.1.2 OTP 烧录器

- **MPIII Writer**：支持 SN8P1927 大批量的脱机烧录。

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 1.99Z01 或更新的版本。
- **MPIII Writer**：SN8IDE 1.99Z01 或更新的版本。
- **M2IDE V1.1X 不支持 SN8P1927 的编译。**

14.2 SN8P1927 DEMO/EV KIT

14.2.1 PCB 说明

SONIX 提供 SN8P1927 DEMO/EV KIT 来仿真所有的功能，如下图所示：

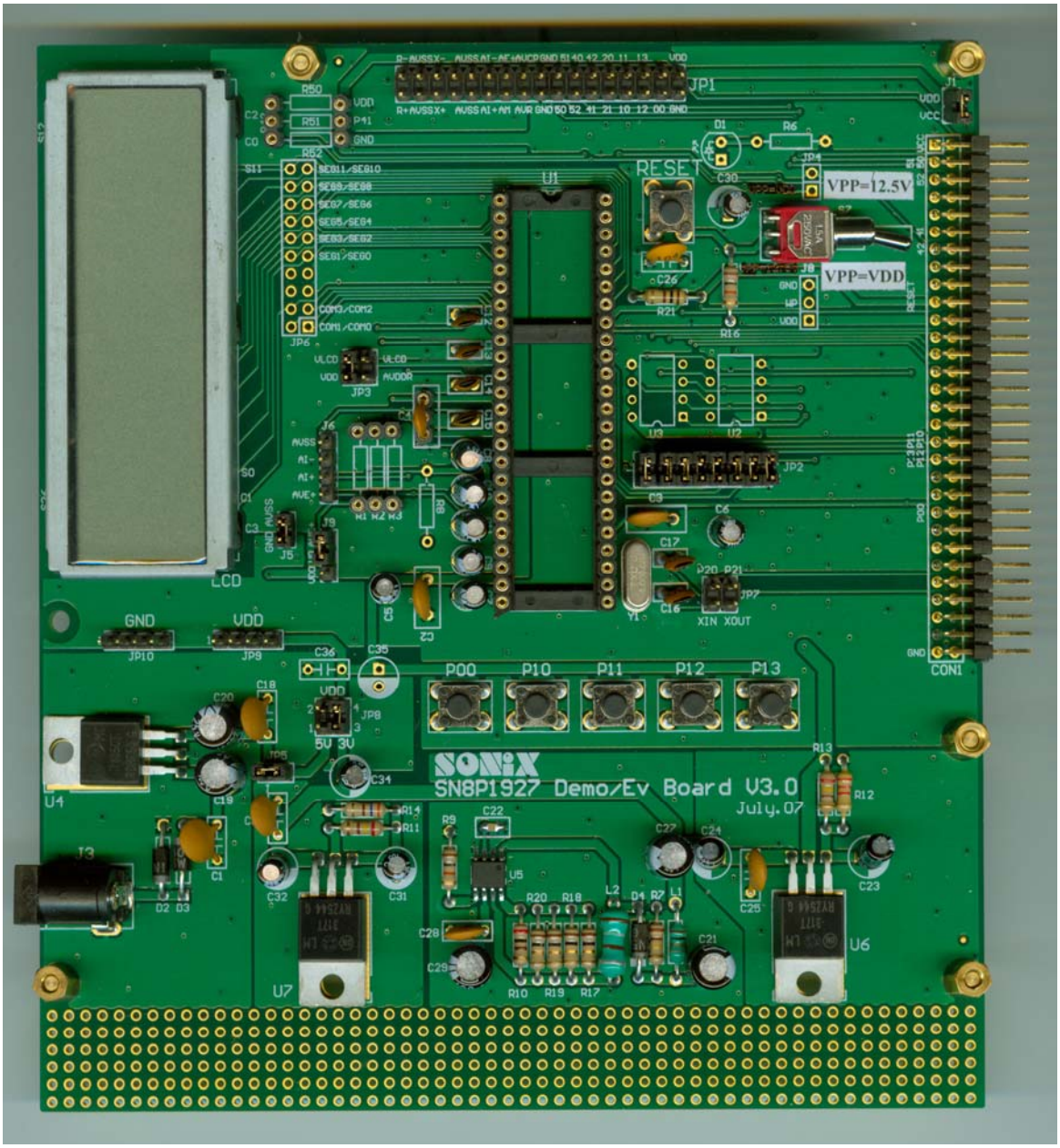


Fig.1 SN8P1927 DEMO/EV KIT

CON1：连接到 ICE。

J1：当电源由 ICE 提供时，将 J1 设置为“SHORT”状态；当电源由其他电源提供时，将 J1 设置为“OPEN”状态。

D1：电源指示灯。

JP2：当 ICE 和 EV-Kit 分开工作时，必须将所有的跳线都设为“SHORT”状态，否则实际单片机的 P1.0~P1.3、P4.2 和 P5.0~P5.2 都不能工作。

JP3：设置不同的 DC 提供给 VLCD。

J5：AVSS 和 GND 的“SHORT”跳线引脚。

JP6：LCD 连接口。

J6：Load cell 信号输入端。

J9：ADC 外部参考电压的选择端，可以选择 VDD 和 AVDDR。

JP7：当 ICE 和 EV-Kit 分开工作且使用的 IHRC 电路时，将所有的跳线都设为“SHORT”状态。

JP1：1927 单片机的 I/O 引脚。

J8：EEPROM 选择功能引脚。24 系列的写保护或普通的读/写操作功能，93 系列的内部结构（8 位/16 位）。

U2/U3：EEPROM 24 series and 93 series placement, respectively.

J3：7.5V 直流电源输入端。

JP5：提供 5V 电压给 U5。当连接到 ICE 时，将 JP5 设为“OPEN”状态。

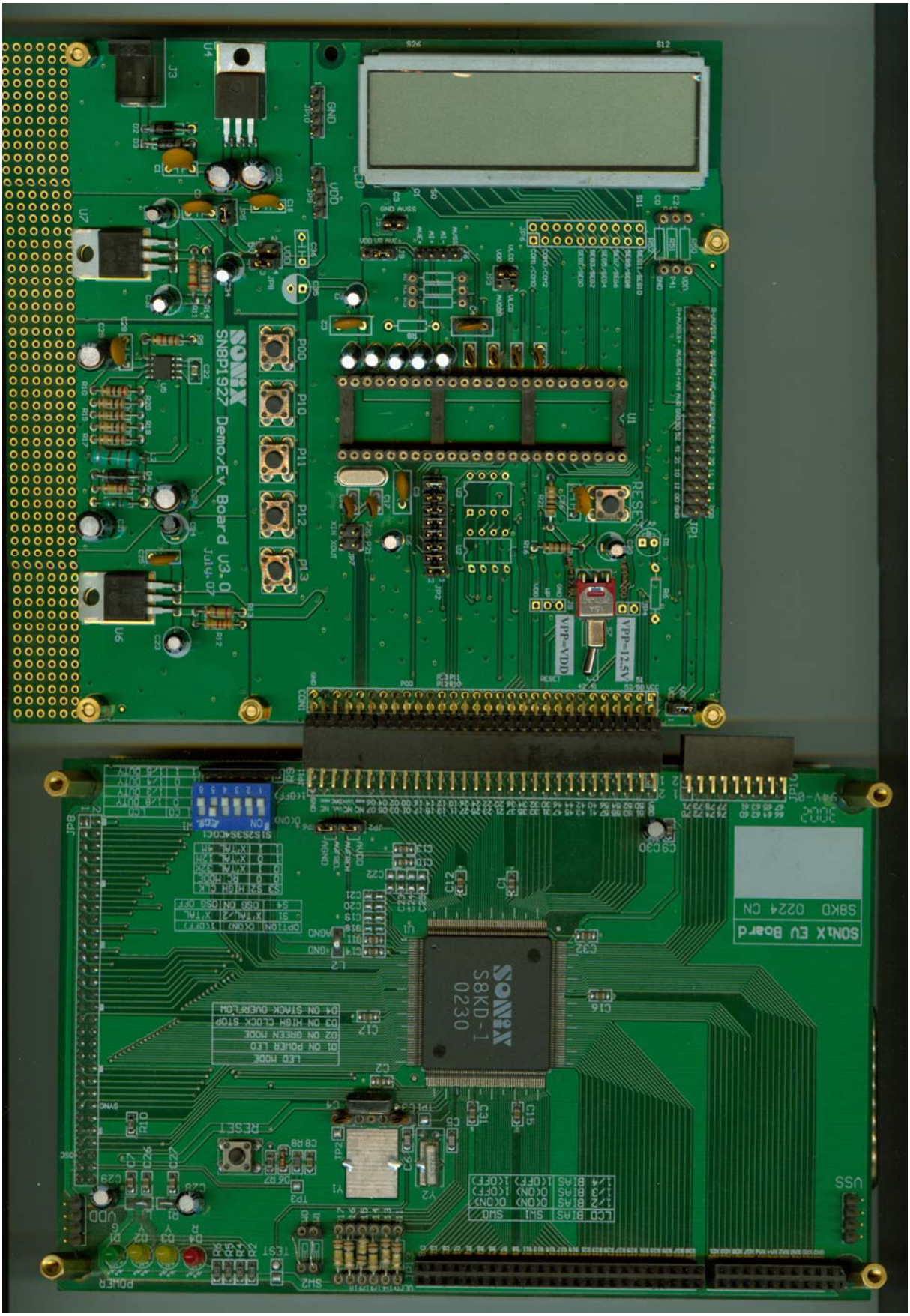
JP8：提供 5V/3V 的电压到 U1 VDD。当连接到 ICE 时，将 JP8 设为“OPEN”状态。

S7：提供“VPP=VDD”或“VPP=12.5V”的电压到 RST/VPP，该切换开关用来控制 RST/VPP 的电压。在上电或正常工作的状态下，S7 必须保持“**VPP=VDD**”的状态；当用户指向 ISP 功能时，S7 就该设为“**VPP=12.5V**”状态，提供 12.5V 的高电压给 VPP/RST 引脚，然后迅速地返回到“**VPP=5V**”的状态；当连接到 ICE 时，S7 提供“**VPP=VDD**”的电压给 RST/VPP 引脚。

	EV board（连接到 ICE）	**Demo board（没有连接到 ICE）**
JP3 (VLCD)	-	由实际情况决定
J1	连接	断开
JP2	所有的跳线都断开	所有的跳线都短接
JP4	由实际情况决定	由实际情况决定
JP6 (LCD)	没有功能	由实际情况决定
U1	SN8P1927 单片机，带有 EV-kit 功能	SN8P1927 单片机，带有应用功能
JP5	断开	连接
J4	连接：EV-Kit 的电源由 ICE 提供（5V） 断开：EV-Kit 的电源由其他电源端提供	-
J5	连接	连接
J9	由实际情况决定	由实际情况决定
JP8	所有的跳线都断开	由实际情况决定是连接 5V 还是连接 3V

14.2.2 SN8P1927 DEMO/EV BOARD 连接到 SN8ICE 1K

SN8P1927 DEMO/EV BOARD 与 SN8ICE 1K 的连接如下图所示：



15 OTP 烧录信息

15.1 烧录转接板信息

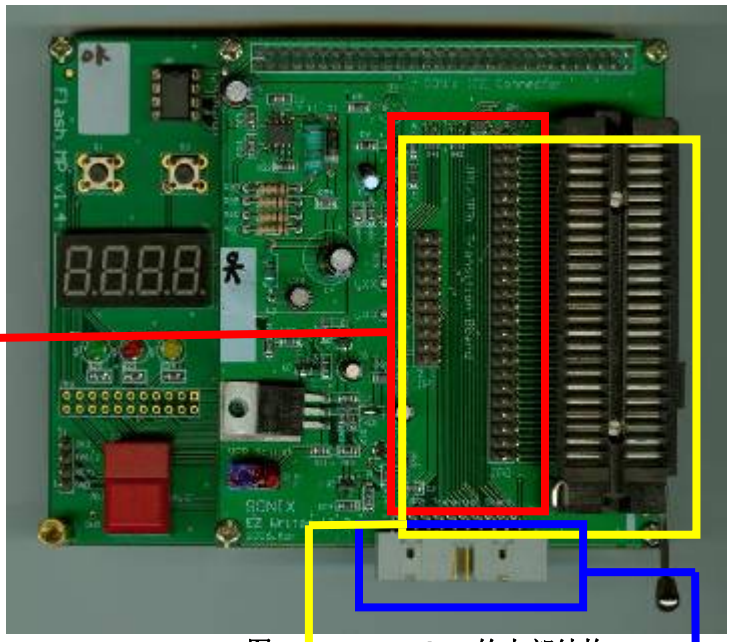
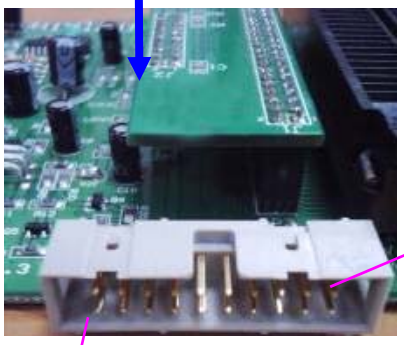


图 1　MPIII Writer 的内部结构

Writer 上板 JP1/JP3

Writer 上板 JP1/JP3



Writer 上板 JP2

注 1：JP1 连接 MP 烧录转接板，JP3 连接 OTP MCU。

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

下面两个图演示了如何焊接烧录转接板。

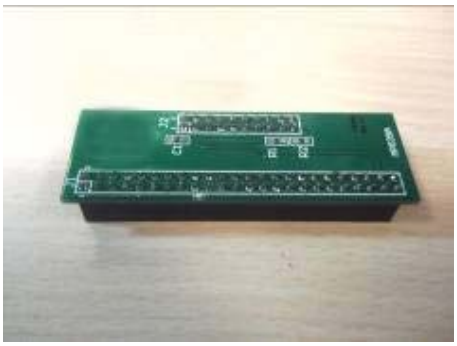


图 2

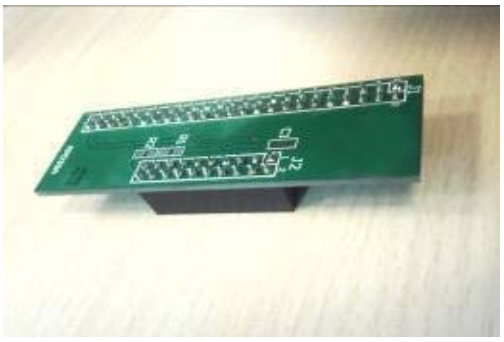


图 3

注：

1、 印有 IC 型号的这一面为转接板的正面。

2、 180 度的母座必须焊接在 MP 转接板的背面。请参考图 2 和图 3。

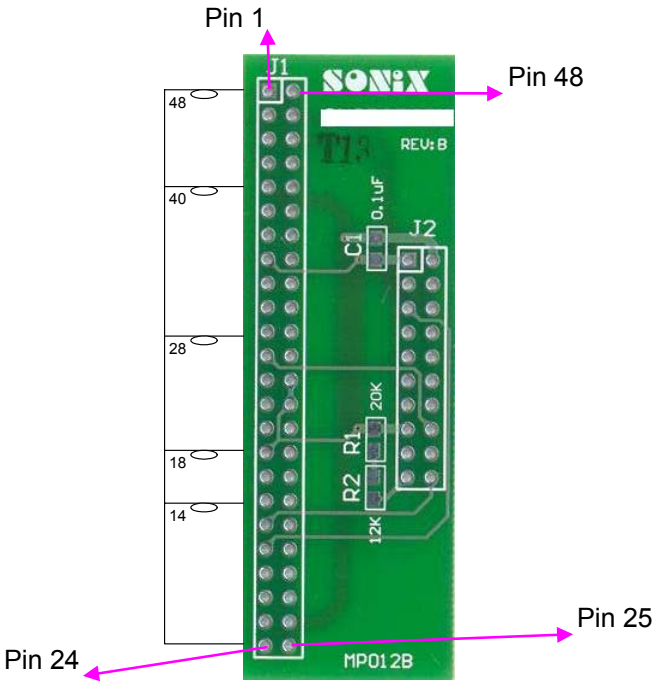


图 4　MP 转接板（连接到 JP1&JP3）

JP3（连接 48-pin text tool）

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

JP1/JP2

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

JP1 连接 MP 烧录转接板
JP2 连接外部转接板

15.2 SN8P1927 系列的烧录信息

SN8P1927 OTP 烧录信息				
单片机型号		SN8P1927		
MPIII Writer		OTP IC / JP3 引脚配置		
JP1/JP2 Pin Number	JP1/JP2 Pin Name	IC Pin Number	IC Pin Name	JP3 Pin Number
1	VDD	10,23,28	VDD	34
2	GND	17,25,40	VSS	15
3	CLK	30	P1.0	22
4	CE	-	-	-
5	PGM	31	P1.1	21
6	OE	32	P1.2	20
7	D1	-	-	-
8	D0	-	-	-
9	D3	-	-	-
10	D2	-	-	-
11	D5	-	-	-
12	D4	-	-	-
13	D7	-	-	-
14	D6	-	-	-
15	VDD	10,23,28	VDD	34
16	VPP	41	RST	18
17	HLS	-	-	-
18	RST	-	-	-
19	-	-	-	-
20	ALSB/PDB	33	P1.3	17

16 电气特性

16.1 极限参数

Supply voltage (V_{DD})..... - 0.3V ~ 6.0V
Input in voltage (V_{IN})..... V_{SS} - 0.2V ~ V_{DD} + 0.2V
Operating ambient temperature (T_{OPR})..... 0°C ~ + 70°C
Storage ambient temperature (T_{STOR})..... –40°C ~ + 125°C

16.2 电气特性

(All of voltages refer to V_{SS}, V_{DD} = 5.0V,F_{OSC} = 4MHz,Fcpu=1MHZ, ambient temperature is 25℃ unless otherwise note.)

PARAMETER	SYM.	DESCRIPTION		MIN.	TYP.	MAX.	UNIT
Operating voltage	V_{DD}	Normal mode, V_{PP} = V_{DD}		2.4	5.0	5.5	V
RAM Data Retention voltage	V_{DR}			1.5		-	V
V_{DD} rise rate	V_{POR}	V_{DD} rise rate to ensure power-on reset		0.05	-	-	V/ms
Input Low Voltage	ViL1	All input ports		Vss	-	0.3Vdd	V
	ViL2	Reset pin		Vss	-	0.2Vdd	V
Input High Voltage	ViH1	All input ports		0.7Vdd	-	Vdd	V
	ViH2	Reset pin		0.9Vdd	-	Vdd	V
Reset pin leakage current	I_{LEKG}	V_{IN} = V_{DD}		-	-	2	uA
I/O port pull-up resistor	R_{UP}	V_{IN} = V_{SS}, V_{DD} = 3V		100	200	300	KΩ
		V_{IN} = V_{SS}, V_{DD} = 5V		50	100	180	KΩ
I/O port input leakage current	I_{LEKG}	Pull-up resistor disable, V_{IN} = V_{DD}		-	-	2	uA
I/O output source current	I_{OH}	V_{OP} = V_{DD} - 0.5V		8	12	-	mA
sink current	I_{OL}	V_{OP} = V_{SS} + 0.5V		8	15	-	mA
INT_N trigger pulse width	$T_{INT}0$	INT0 ~ INT1 interrupt request pulse width		2/F_{CPU}	-	-	Cycle
Supply Current	Idd1	normal Mode (Low Power Disable, Analog Parts OFF)	Vdd= 5V 4Mhz crystal	-	1.5	3	mA
			Vdd= 3V 4Mhz crystal	-	0.5	1	mA
	Idd2	normal Mode (Low Power Enable Analog Parts OFF)	Vdd= 5V 4Mhz crystal	-	0.9	1.8	mA
			Vdd= 3V 4Mhz crystal	-	0.35	0.7	mA
	Idd3	normal Mode (Low Power Disable Analog Parts ON)	Vdd= 5V 4Mhz crystal	-	2.5	5	mA
			Vdd= 3V 4Mhz crystal	-	2.2	4.4	mA
	Idd4	normal Mode (Low Power Enable Analog Parts ON)	Vdd= 5V 4Mhz crystal	-	2	4	mA
			Vdd= 3V 4Mhz crystal	-	1.5	3	mA
	Idd5	normal Mode (Low Power Disable, Analog Parts OFF)	Vdd= 5V IHRC	-	1.7	3.4	mA
			Vdd= 3V IHRC	-	0.9	1.8	mA
	Idd6	normal Mode (Low Power Enable Analog Parts OFF)	Vdd= 5V IHRC	-	1.1	2.2	mA
			Vdd= 3V IHRC	-	0.7	1.4	mA
	Idd7	normal Mode (Low Power Disable Analog Parts ON)	Vdd= 5V IHRC	-	2.5	5	mA
			Vdd= 3V IHRC	-	2.2	4.4	mA
	Idd8	normal Mode (Low Power Enable Analog Parts ON)	Vdd= 5V IHRC	-	2.1	4.2	mA
			Vdd= 3V IHRC	-	1.9	3.8	mA
	Idd9	Slow mode (Stop High Clock, LCD OFF, CPR OFF)	Vdd= 5V ILRC 32Khz	-	10	20	uA
			Vdd= 3V ILRC 16Khz	-	3	6	uA
	Idd10	Slow mode (Stop High Clock, LCD ON, CPR OFF)	Vdd= 5V ILRC 32Khz	-	25	50	uA
			Vdd= 3V ILRC 16Khz	-	12	24	uA
	Idd11	Slow mode (Stop High Clock, LCD ON, CPR ON)	Vdd= 5V ILRC 32Khz	-	300	600	uA
			Vdd= 3V ILRC 16Khz	-	250	500	uA
	Idd12	Sleep mode	Vdd= 5V	-	1	2	uA
			Vdd= 3V	-	0.7	1.5	uA
Internal High Clock Freq.	F_{IHRC}	Internal High RC Oscillator Frequency (Fcpu= F_{IHRC}/16)		14	16	18	MHz
LVD detect level	V_{LVD}	Internal POR detect level		1.7	2.0	2.3	V

➢ **Note: Analog Parts including Charge Pump Regulator (CPR), PGIA and ADC.**

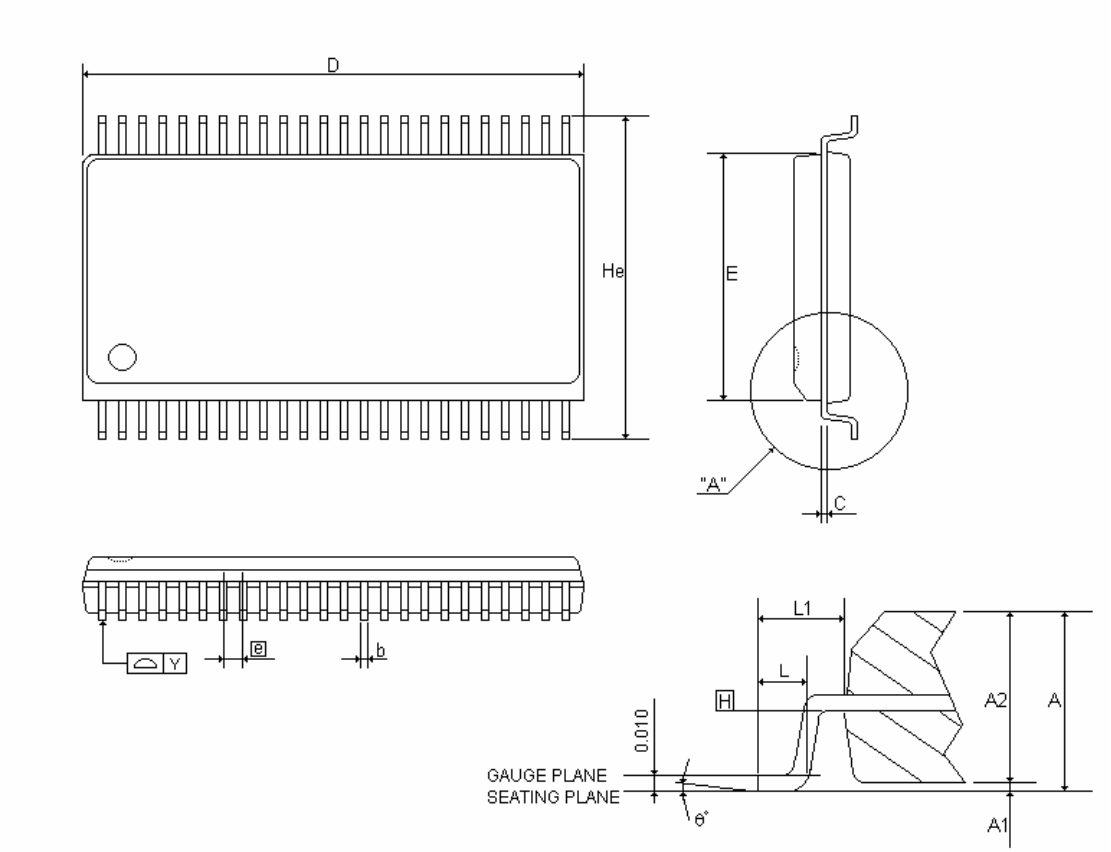
(All of voltages refer to Vdd=3.8V F_{OSC} = 4MHz, ambient temperature is 25°C unless otherwise note.)

PARAMETER	SYM.	DESCRIPTION	MIN.	TYP.	MAX.	UNIT
Analog to Digital Converter						
Operating current	I_{DD_ADC}	Run mode @ 3.8V		800	1000	uA
Power down current	I_{PDN}	Stop mode @ 3.8V		0.1	1	μA
Conversion rate	F_{SMP}	ADCKS: 200KHz			25	sps
Reference Voltage Input Voltage	Vref	R+, R- Input Range (External Ref.)	0.4		2.0	V
		R+, R- Input Range (Internal Ref.)	0.2		2.0	V
Differential non-linearity	DNL	ADC range ± 28125		±0.5	±0.5	LSB
Integral non-linearity	INL	ADC range ± 28125		±1	±4	LSB
No missing code	NMC	ADC range ± 28125	16			bit
Noise free code	NFC	ADC range ± 28125		14	16	bit
Effective number of bits	ENOB	ADC range ± 28125		14	16	bit
ADC Input range	V_{AIN}		0.4		2.0	V
Temperature Sensor inaccuracy	E_{TS}	Inaccuracy range vs. real Temp.		±8		℃
PGIA						
Current consumption	I_{DD_PGIA}	Run mode @ 3.8V		300	500	uA
Power down current	I_{PDN}	Stop mode @ 3.8V			0.1	μA
Input offset voltage	Vos			25	50	uV
Bandwidth	BW				100	Hz
PGIA Gain Range (Gain=200x)	GR	VDD = 3.8V	180	200	250	
PGIA Input Range	Vopin	VDD = 3.8V	0.4		2	V
PGIA Output Range	Vopout	VDD = 3.8V	0.4		2	V
Band gap Reference (Refer to ACM)						
Band gap Reference Voltage	V_{BG}		1.160	1.210	1.270	V
Reference Voltage Temperature Coefficient	T_{ACM}			50*		PPM/℃
Operating current	I_{BG}	Run mode @ 3.8V		50	100	uA
Charge pump regulator						
Supply voltage	V_{CPS}	Normal mode	2.4		5.5	V
Regulator output voltage AVDDR	V_{AVDDR}		3.5	3.75	4.1	V
Regulator output voltage AVE+	V_{AVE+}	AVE+ set as 2.0V	1.8	2.05	2.3	V
Analog common voltage	V_{ACM}		1.15	1.21	1.27	V
Regulator output current capacity	I_{VA+}		10			mA
Quiescent current	I_{QI}			700	1400	uA
V_{ACM} driving capacity	I_{SRC}		10	-	-	μA
V_{ACM} sinking capacity	I_{SNK}		1	-	-	mA

✸ ***Note : When Charge Pump enable, current consumption will be time 2 of ADC, PGIA, CPR and Loading from AVDDR.***

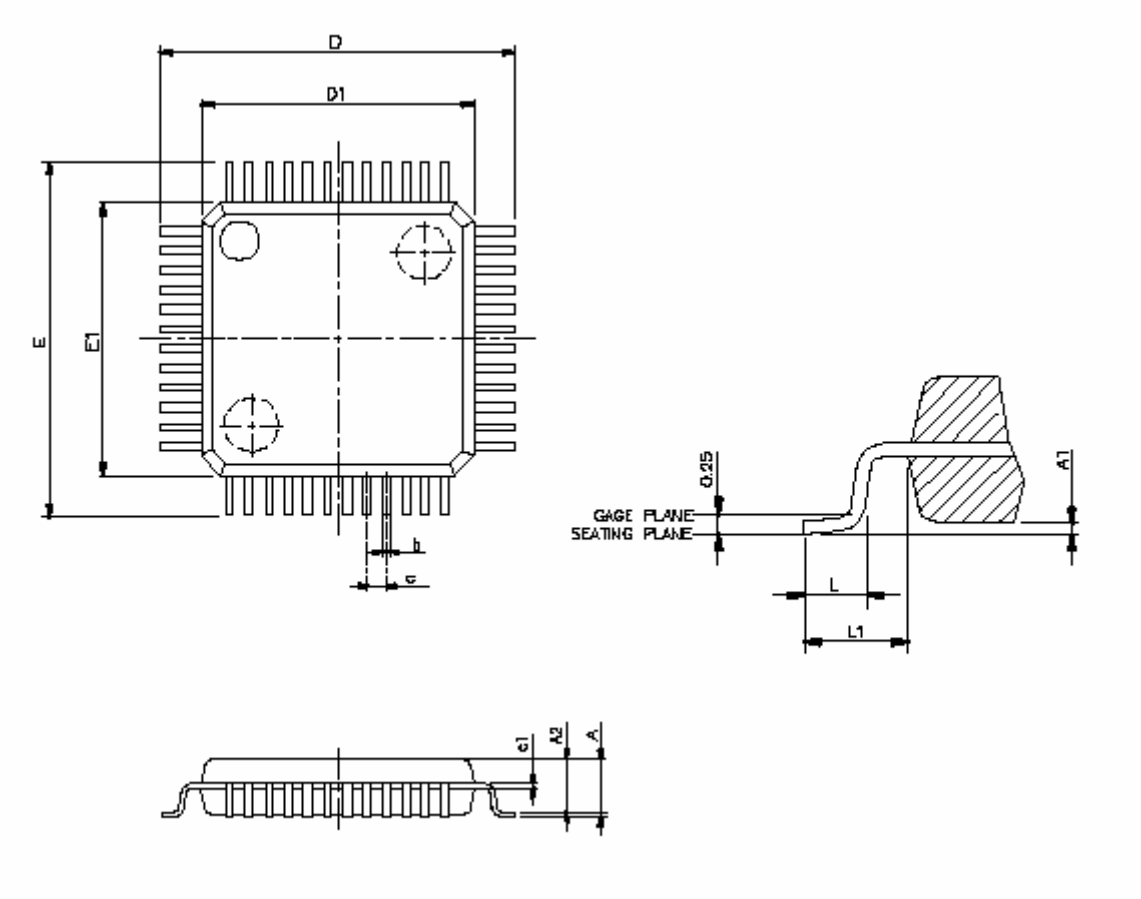
17 封装信息

17.1 SSOP 48 PIN



SYMBOLS	MIN	NOR	MAX	MIN	NOR	MAX
	(inch)			(mm)		
A	0.095	0.102	0.110	2.413	2.591	2.794
A1	0.008	0.012	0.016	0.203	0.305	0.406
A2	0.089	0.094	0.099	2.261	2.388	2.515
b	0.008	0.010	0.030	0.203	0.254	0.762
C	-	0.008	-	-	0.203	-
D	0.620	0.625	0.630	15.748	15.875	16.002
E	0.291	0.295	0.299	7.391	7.493	7.595
[e]	-	0.025	-	-	0.635	-
He	0.396	0.406	0.416	10.058	10.312	10.566
L	0.020	0.030	0.040	0.508	0.762	1.016
L1	-	0.056	-	-	1.422	-
Y	-	-	0.003	-	-	0.076
θ°	0°	-	8°	0°	-	8°

17.2 LQFP 48 PIN



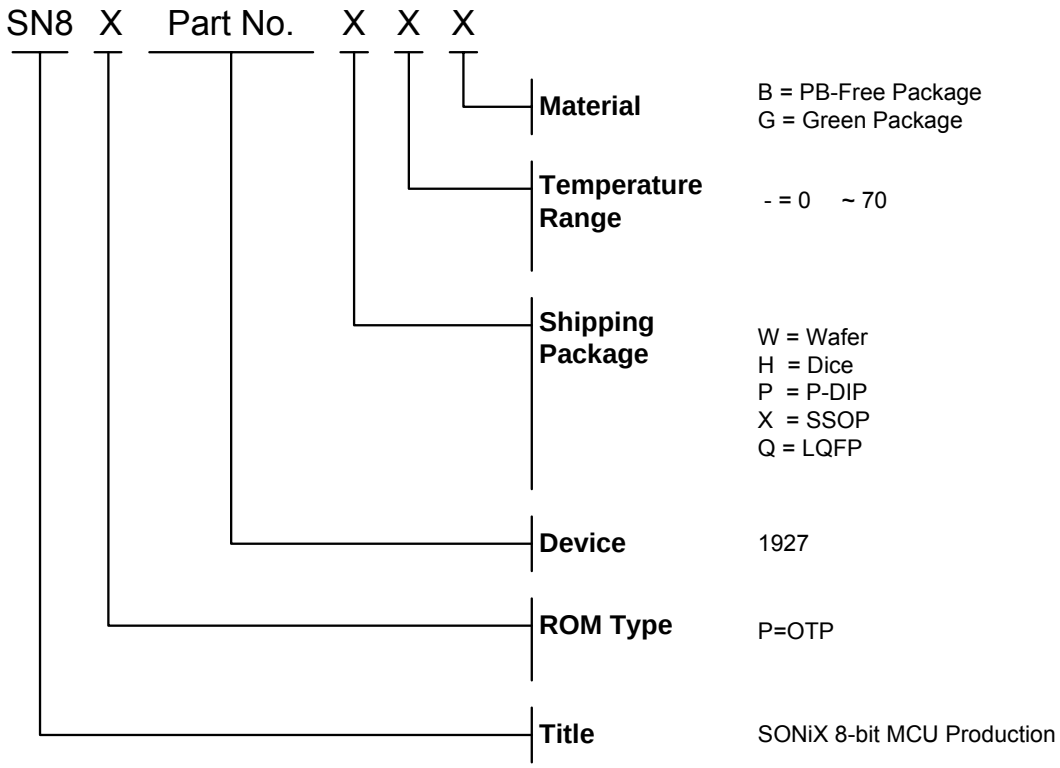
SYMBOLS	MIN	NOR	MAX
	(mm)		
A	-	-	1.6
A1	0.05	-	0.15
A2	1.35	-	1.45
c1	0.09	-	0.16
D	9.00 BSC		
D1	7.00 BSC		
E	9.00 BSC		
E1	7.00 BSC		
e	0.5 BSC		
B	0.17	-	0.27
L	0.45	-	0.75
L1	1 REF		

18 单片机正印命名规则

18.1 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。

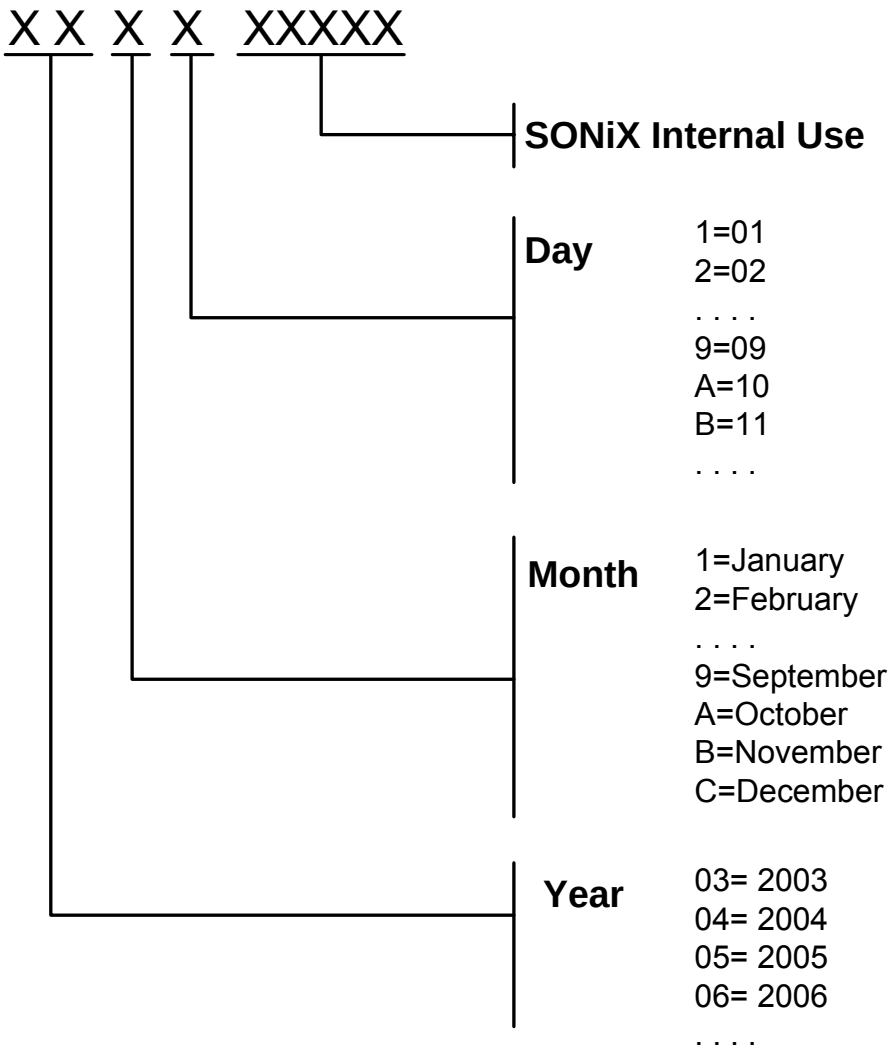
18.2 单片机型号说明



18.3 命名举例

单片机名称	ROM 类型	器件（Device）	封装形式	温度范围	封装材料
SN8P1927PB	OTP	1927	P-DIP	0℃~70℃	无铅封装（PB-Free Package）
SN8P1927XG	OTP	1927	SSOP	0℃~70℃	绿色封装（Green Package）

18.4 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