SC8P051 用户手册

增强型 OTP 8 位 CMOS 单片机

Rev. 0.1.1

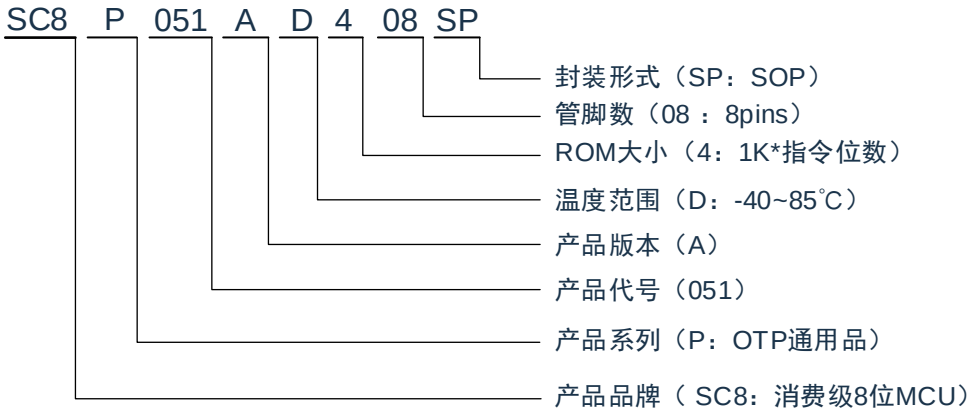
目录

1. 产品概述

1.1 功能特性

- 内存
  - OTP：1K×14Bit
  - 通用 RAM：64×8Bit
- 5 级堆栈缓存器
- 简洁实用的指令系统（66 条指令）
- 内置低压侦测电路
- 内置 WDT 定时器
- 中断源
  - 2 个定时中断
  - RB 口电平变化中断
  - 其它外设中断
- 定时器
  - 8 位定时器 TIMER0/TIMER2
- 内置比较器模块
  - 正端可选：RB1/电阻分压输出
  - 负端可选：RB1/RB2/RB4/RB5/BG/电阻分压输出
- 工作电压范围：2.5V~5.5V@16MHz
  1.8V~5.5V@8MHz
  工作温度范围：-40℃~85℃
- 内部 RC 振荡：设计频率 16MHz
- 指令周期（单指令或双指令）
- 内置 PWM 模块
  - 3 路 PWM，输出极性可选
  - PWM0/1 10Bit 共用周期，独立占空比，可设置成互补输出
  - PWM4 10Bit 独立周期，独立占空比
- 内建高精度 1.2V 基准电压
- LVR 可选 1.8V/2V/2.5V/3V

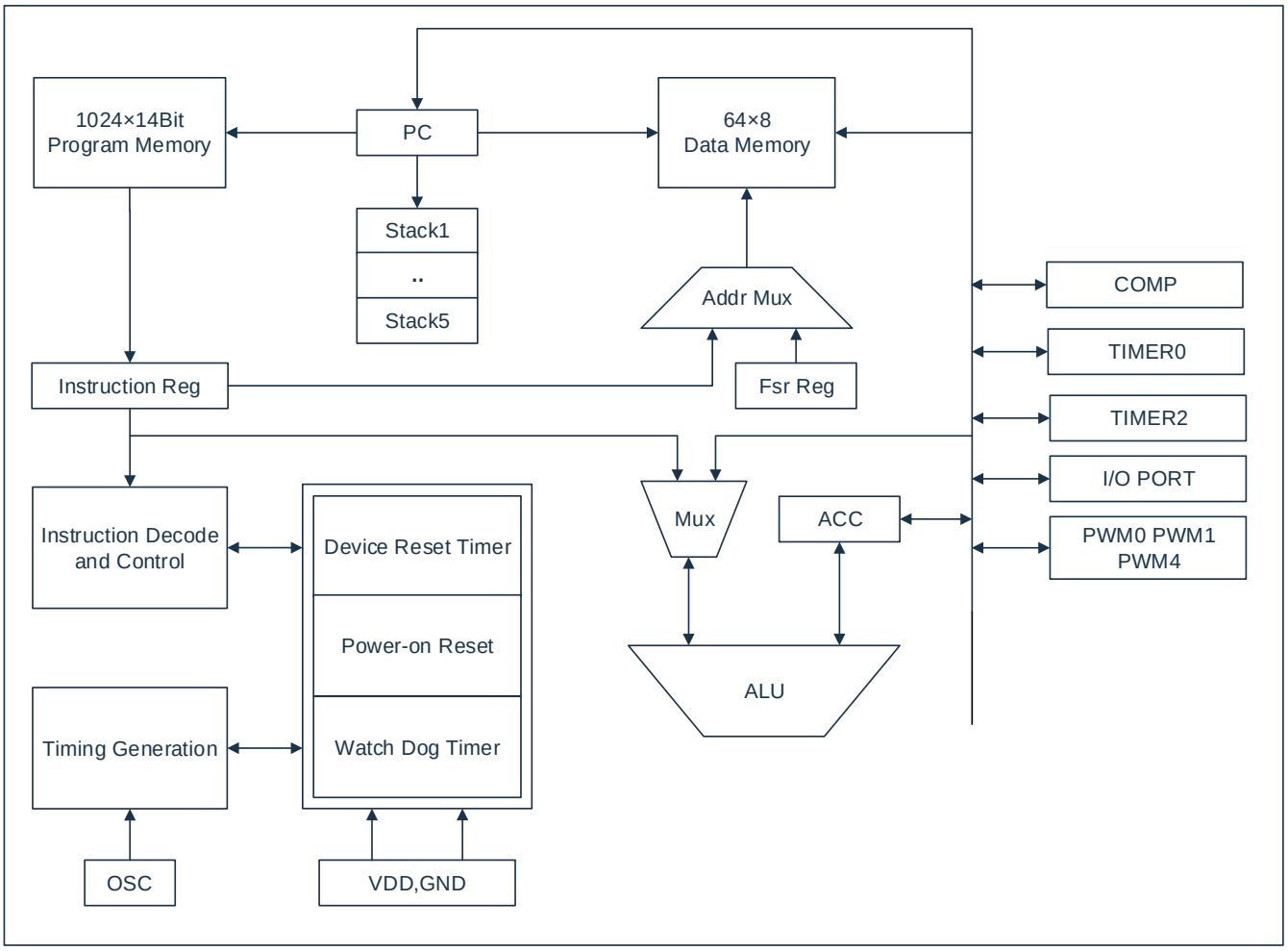
1.2 产品型号一览表



型号说明

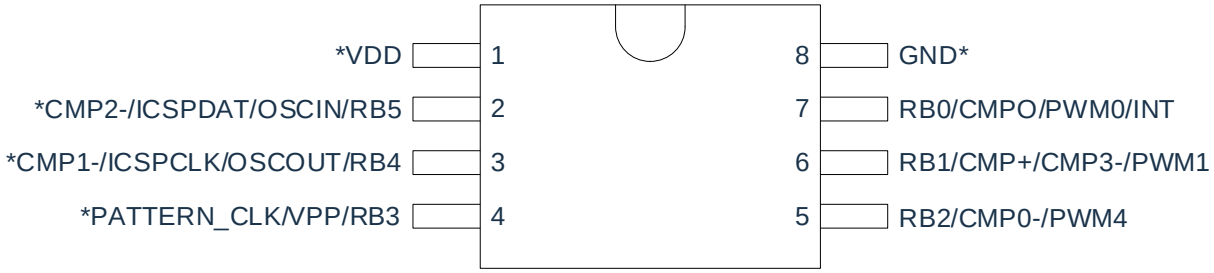
PRODUCT	ROM	RAM	PWM	ACOMP	I/O	TIMER	PACKAGE
SC8P051AD408SPNC	1K×14	64×8	3	1	6	2	SOP8

1.3 系统结构框图


1024×14Bit
Program Memory
PC
64×8
Data Memory
Stack1
..
Stack5
Addr Mux
COMP
TIMER0
TIMER2
I/O PORT
PWM0 PWM1
PWM4
Instruction Reg
Fsr Reg
Instruction Decode
and Control
Device Reset Timer
Power-on Reset
Watch Dog Timer
Mux
ACC
ALU
Timing Generation
OSC
VDD,GND

1.4 管脚分布

1.4.1 SC8P051AD408SPNC 管脚图



SC8P051AD408

SC8P051 引脚说明：

管脚名称	IO 类型	管脚说明
VDD, GND	P	电源电压输入脚，接地脚
RB0-RB2，RB4-RB5	I/O	可编程为输入脚，推挽或开漏输出脚，带上拉、下拉电阻功能、电平变化中断功能
RB3	I/O	可编程为输入脚，开漏输出脚，带上拉电阻功能、电平变化中断功能
ICSPCLK/ICSPDAT	I/O	编程时钟/数据脚
VPP	I	编程高压输入口
PWM0-PWM1，PWM4	O	PWM 输出脚
INT	I	外部中断输入脚
CMP+	I	比较器正端输入脚
CMP0-，CMP1-, CMP2-，CMP3-	I	比较器负端输入脚
CMPO	O	比较器结果输出脚
OSCIN，OSCOUT	I/O	外部低速振荡驱动口

1.5 系统配置寄存器

系统配置寄存器（CONFIG）是 MCU 初始条件的 OTP 选项。它只能被 SC 烧写器烧写，用户不能访问及操作。它包含了以下内容：

1. WDT（看门狗选择）
   - ENABLE　　打开看门狗定时器
   - DISABLE　　关闭看门狗定时器
2. PROTECT（加密）
   - DISABLE　　ROM 代码不加密
   - ENABLE　　ROM 代码加密，加密后烧写仿真器读出来的值将不确定
3. LVR_SEL（低压侦测选择）
   - 1.8V
   - 2.0V
   - 2.5V
   - 3.0V
4. 系统时钟分频
   - 4T　　4 分频，$F_{CPU}=F_{SYS}/4$
   - 2T　　2 分频，$F_{CPU}=F_{SYS}/2$
5. PORTB[2:0]拉电流选择
   - 20mA
   - 80mA
   - 100mA
   - 160mA
   - 180mA
   - 220mA
6. PORTB[2:0]灌电流选择
   - 40mA
   - 80mA
   - 120mA
   - 160mA
   - 200mA
   - 220mA
7. PORTB[5:4]拉电流选择
   - 20mA
   - 2mA
8. PORTB[5:4]灌电流选择
   - 40mA
   - 7mA
9. 比较器正端电压选择
   - RB1
   - RB2

10. 起振电容选择
    - 0pF
    - 12.5pF
11. 比较器滤波时间使能
    - 开启滤波（1us）
    - 关闭滤波

1.6 在线串行编程

可在最终应用电路中对单片机进行串行编程。编程可以简单地通过以下 5 根线完成：

- 电源线
- 接地线
- 数据线
- 时钟线
- 高压线

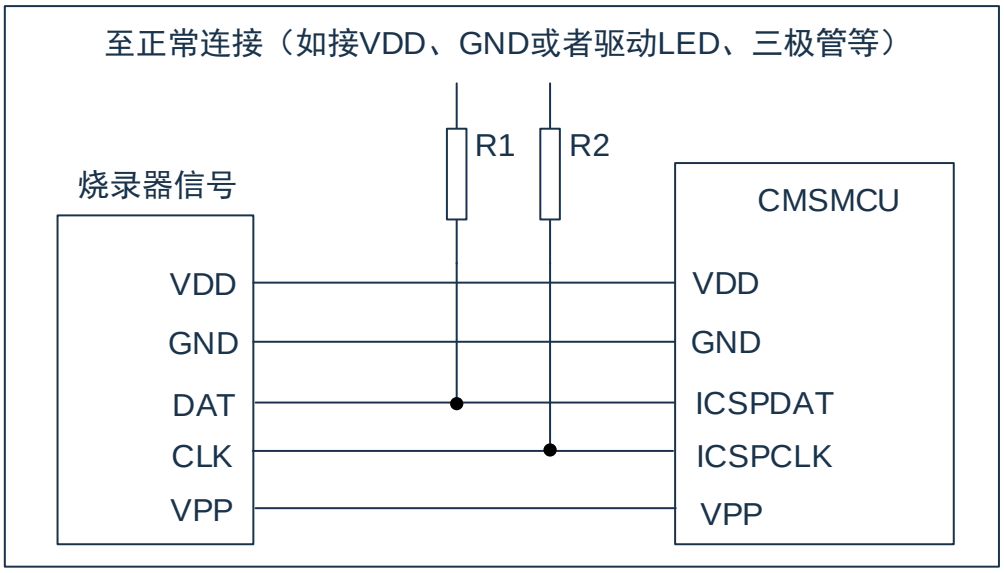


图 1-1：典型的在线串行编程连接方式

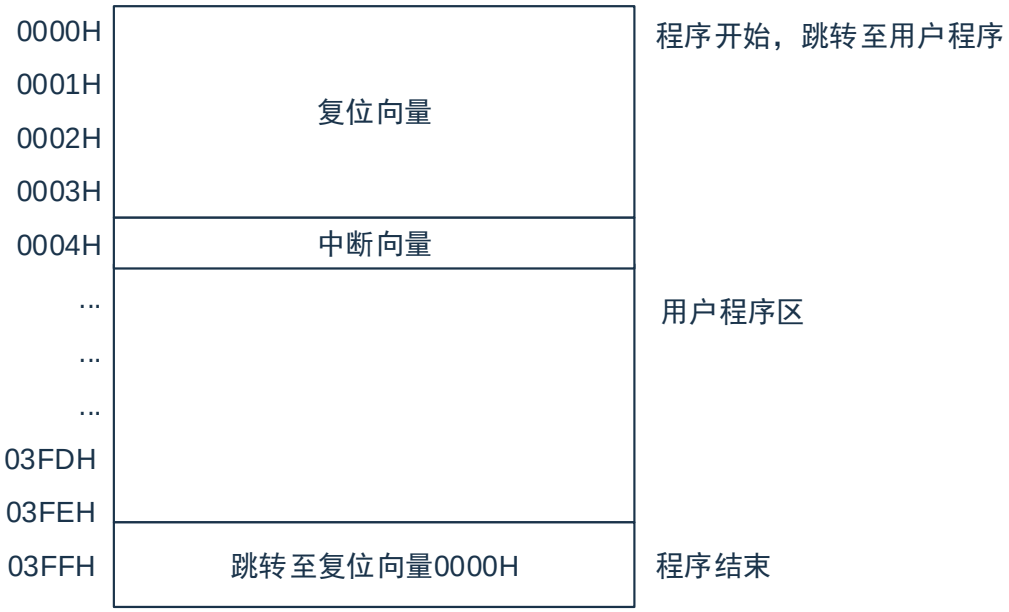
上图中，R1、R2 为电气隔离器件，常以电阻代替，其阻值如下：R1≥4.7K、R2≥4.7K。

2. 中央处理器（CPU）

2.1 内存

2.1.1 程序内存

OTP：1K



2.1.1.1 复位向量(0000H)

单片机具有一个字长的系统复位向量（0000H）。具有以下三种复位方式：

- 上电复位
- 看门狗复位
- 低压复位（LVR）

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

例：定义复位向量

```
        ORG     0000H       ;系统复位向量
        JP      START
        ORG     0010H       ;用户程序起始
START:
        ...                 ;用户程序
        ...
        END                 ;程序结束
```

2.1.1.2 中断向量

中断向量地址为 0004H。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 0004H 开始执行中断服务程序。所有中断都会进入 0004H 这个中断向量，具体执行哪个中断将由用户根据中断请求标志位寄存器的位决定。下面的示例程序说明了如何编写中断服务程序。

例：定义中断向量，中断程序放在用户程序之后

```
                ORG     0000H       ;系统复位向量
                JP      START
                ORG     0004H       ;用户程序起始
INT_START:
                CALL    PUSH        ;保存 ACC 跟 STATUS
                …                   ;用户中断程序
                …
INT_BACK:
                CALL    POP         ;返回 ACC 跟 STATUS
                RETI                ;中断返回
START:
                …                   ;用户程序
                …
                END                 ;程序结束
```

注：由于单片机并未提供专门的出栈、压栈指令，故用户需自己保护中断现场。

例：中断入口保护现场

```
PUSH:
                LD      ACC_BAK,A   ;保存 ACC 至自定义寄存器 ACC_BAK
                SWAPA   STATUS      ;状态寄存器 STATUS 高低半字节互换
                LD      STATUS_BAK,A;保存至自定义寄存器 STATUS_BAK
                RET                 ;返回
```

例：中断出口恢复现场

```
POP:
                SWAPA   STATUS_BAK  ;将保存至 STATUS_BAK 的数据高低半字节互换给 ACC
                LD      STATUS,A    ;将 ACC 的值给状态寄存器 STATUS
                SWAPR   ACC_BAK     ;将保存至 ACC_BAK 的数据高低半字节互换
                SWAPA   ACC_BAK     ;将保存至 ACC_BAK 的数据高低半字节互换给 ACC
                RET                 ;返回
```

2.1.1.3 跳转表

跳转表能够实现多地址跳转功能。由于 PCL 和 ACC 的值相加即可得到新的 PCL，因此，可以通过对 PCL 加上不同的 ACC 值来实现多地址跳转。ACC 值若为 n，PCL+ACC 即表示当前地址加 n，执行完当前指令后 PCL 值还会自加 1，可参考以下范例。如果 PCL＋ACC 后发生溢出，PC 不会自动进位，故编写程序时应注意。这样，用户就可以通过修改 ACC 的值轻松实现多地址的跳转。

PCLATH 为 PC 高位缓冲寄存器，对 PCL 操作时，必须先对 PCLATH 进行赋值。

例：正确的多地址跳转程序示例

```
OTP 地址
                LDIA        01H
                LD          PCLATH,A        ;必须对 PCLATH 进行赋值
                …
0110H:          ADDR        PCL             ;ACC+PCL
0111H:          JP          LOOP1           ;ACC=0，跳转至 LOOP1
0112H:          JP          LOOP2           ;ACC=1，跳转至 LOOP2
0113H:          JP          LOOP3           ;ACC=2，跳转至 LOOP3
0114H:          JP          LOOP4           ;ACC=3，跳转至 LOOP4
0115H:          JP          LOOP5           ;ACC=4，跳转至 LOOP5
0116H:          JP          LOOP6           ;ACC=5，跳转至 LOOP6
```

例：错误的多地址跳转程序示例

```
OTP 地址
                CLR         PCLATH
                …
00FCH:          ADDR        PCL             ;ACC+PCL
00FDH:          JP          LOOP1           ;ACC=0，跳转至 LOOP1
00FEH:          JP          LOOP2           ;ACC=1，跳转至 LOOP2
00FFH:          JP          LOOP3           ;ACC=2，跳转至 LOOP3
0100H:          JP          LOOP4           ;ACC=3，跳转至 0000H 地址
0101H:          JP          LOOP5           ;ACC=4，跳转至 0001H 地址
0102H:          JP          LOOP6           ;ACC=5，跳转至 0002H 地址
```

注：由于 PCL 溢出不会自动向高位进位，故在利用 PCL 作多地址跳转时，需要注意该段程序一定不能放在 OTP 空间的分页处。

2.1.2 数据存储器

SC8P051 数据存储器列表

	地址		地址		地址		地址
INDF	00H	INDF	80H	-----	100H	-----	180H
OPTION_REG	01H	TMR0	81H	-----	101H	-----	181H
PCL	02H	PCL	82H	-----	102H	-----	182H
STATUS	03H	STATUS	83H	-----	103H	-----	183H
FSR	04H	FSR	84H	-----	104H	-----	184H
TRISB	05H	-----	85H	-----	105H	-----	185H
PORTB	06H	-----	86H	-----	106H	-----	186H
WPDB	07H	-----	87H	-----	107H	-----	187H
WPUB	08H	-----	88H	-----	108H	-----	188H
IOCB	09H	-----	89H	-----	109H	-----	189H
PCLATH	0AH	PCLATH	8AH	-----	10AH	-----	18AH
INTCON	0BH	INTCON	8BH	-----	10BH	-----	18BH
ODCONB	0CH	-----	8CH	-----	10CH	-----	18CH
PIR1	0DH	-----	8DH	-----	10DH	-----	18DH
PIE1	0EH	-----	8EH	-----	10EH	-----	18EH
CMPCON0	0FH	-----	8FH	-----	10FH	-----	18FH
CMPCON1	10H	-----	90H	-----	110H	-----	190H
PR2	11H	-----	91H	-----	111H	-----	191H
TMR2	12H	-----	92H	-----	112H	-----	192H
T2CON	13H	-----	93H	-----	113H	-----	193H
OSCCON	14H	-----	94H	-----	114H	-----	194H
PWMCON0	15H	-----	95H	-----	115H	-----	195H
PWMCON1	16H	-----	96H	-----	116H	-----	196H
PWMTL	17H	-----	97H	-----	117H	-----	197H
PWMTH	18H	-----	98H	-----	118H	-----	198H
PWMD0L	19H	-----	99H	-----	119H		199H
PWMD1L	1AH	-----	9AH	-----	11AH		19AH
PWMD4L	1BH	-----	9BH	-----	11BH		19BH
PWMT4L	1CH	-----	9CH	-----	11CH	-----	19CH
PWMCON2	1DH	-----	9DH	-----	11DH		19DH
PWMD01H	1EH	-----	9EH	-----	11EH		19EH
PWM01DT	1FH	-----	9FH	-----	11FH		19FH
	20H		A0H		120H		1A0H
	3FH	-----		-----		-----	
	40H		EFH		16FH		1EFH
通用寄存器 64 字节			F0H		170H		1F0H
		快速存储区 70H-7FH	--	-----	--	-----	--
	7FH		FFH		17FH		1FFH
BANK0		BANK1		BANK2		BANK3	

数据存储器分为两个功能区间：特殊功能寄存器和通用数据存储器。数据存储器单元大多数是可读/写的，但有些只读的。特殊功能寄存器地址为从 00H-1FH。

SC8P051 特殊功能寄存器汇总 Bank0

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	复位值
00H	INDF	寻址该单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
01H	OPTION_REG	T0LSE_REG	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	01111011
02H	PCL	程序计数器低字节								00000000
03H	STATUS	----	----	RP0	TO	PD	Z	DC	C	--011xxx
04H	FSR	间接数据存储器地址指针								xxxxxxxx
05H	TRISB	----	----	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	--111111
06H	PORTB	----	----	RB5	RB4	RB3	RB2	RB1	RB0	--xxxxxx
07H	WPDB	----	----	WPDB5	WPDB4	----	WPDB2	WPDB1	WPDB0	--00-000
08H	WPUB	----	----	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	--000000
09H	IOCB	----	----	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	--000000
0AH	PCLATH	----	----	----	----	----	----	程序计数器高2位的写缓冲器		------00
0BH	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	00000000
0CH	ODCONB	----	----	ODCONB5	ODCONB4	----	ODCONB2	ODCONB1	ODCONB0	--00-000
0DH	PIR1	----	----	CMPIF	PWMIF	----	----	TMR2IF	----	--00--0-
0EH	PIE1	----	----	CMPIE	PWMIE	----	----	TMR2IE	----	--00--0-
0FH	CMPCON0	CMPEN	CMPPS	CMPNS2	CMPNS1	CMPNS0	CMPNV	CMPOUT	CMPOEN	00000000
10H	CMPCON1	CMPIM	ANSEL	RBIAS_H	RBIAS_L	LVDS<3:0>				00000000
11H	PR2	TIMER2周期寄存器								11111111
12H	TMR2	TIMER2模块寄存器								00000000
13H	T2CON	CLK_SEL	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	00000000
14H	OSCCON	----	IRCF2	IRCF1	IRCF0	----	----	SWDTEN	-----	-101--0-
15H	PWMCON0	CLKDIV<2:0>			PWM4EN	----	----	PWM1EN	PWM0EN	0000--00
16H	PWMCON1	----	----	----	PWM0DTEN	----	----	DT_DIV<1:0>		---0--00
17H	PWMTL	PWM0~PWM1周期低8位寄存器								00000000
18H	PWMTH	----	----	PWM4D<9:8>		PWM4T<9:8>		PWMT<9:8>		--000000
19H	PWMD0L	PWM0占空比低8位								00000000
1AH	PWMD1L	PWM1占空比低8位								00000000
1BH	PWMD4L	PWM4占空比低8位								00000000
1CH	PWMT4L	PWM4周期低8位寄存器								00000000
1DH	PWMCON2	----	----	----	PWM4DIR	----	----	PWM1DIR	PWM0DIR	---0--00
1EH	PWMD01H	----	----	PWMD1<9:8>		----	----	PWMD0<9:8>		--00--00
1FH	PWM01DT	----	----	PWM01DT<5:0>						--000000

SC8P051 特殊功能寄存器汇总 Bank1

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	复位值
80H	INDF	寻址该单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
81H	TMR0									xxxxxxxx
82H	PCL	程序计数器低字节								00000000
83H	STATUS	----	----	RP0	TO	PD	Z	DC	C	--011xxx
84H	FSR	间接数据存储器地址指针								xxxxxxxx
8AH	PCLATH	----	----	----	----	----	----	程序计数器高2位的写缓冲器		------00
8BH	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	00000000

2.2 寻址方式

2.2.1 直接寻址

通过工作寄存器（ACC）来对 RAM 进行操作。

例：ACC 的值送给 30H 寄存器

```
LD          30H,A
```

例：30H 寄存器的值送给 ACC

```
LD          A,30H
```

2.2.2 立即寻址

把立即数传给工作寄存器（ACC）

例：立即数 12H 送给 ACC

```
LDIA        12H
```

2.2.3 间接寻址

数据存储器能被直接或间接寻址。通过 INDF 寄存器可间接寻址，INDF 不是物理寄存器。当对 INDF 进行存取时，它会把 FSR 寄存器内的值作为地址，并指向该地址的寄存器，因此在设置了 FSR 寄存器后，就可把 INDF 寄存器当作目的寄存器来存取。间接读取 INDF（FSR=0）将产生 00H。间接写入 INDF 寄存器，将导致一个空操作。以下例子说明了程序中间接寻址的用法。

例：FSR 及 INDF 的应用

```
        LDIA    30H
        LD      FSR,A       ;间接寻址指针指向 30H
        CLR     INDF        ;清零 INDF 实际是清零 FSR 指向的 30H 地址 RAM
```

例：间接寻址清 RAM(20H-5FH)举例：

```
        LDIA    1FH
        LD      FSR,A       ;间接寻址指针指向 1FH
LOOP:
        INCR    FSR         ;地址加 1，初始地址为 20H
        CLR     INDF        ;清零 FSR 所指向的地址
        LDIA    5FH
        SUBA    FSR
        SNZB    STATUS,C    ;一直清零至 FSR 地址为 5FH
        JP      LOOP
```

2.3 堆栈

芯片的堆栈缓存器共 5 层，堆栈缓存器既不是数据存储器的一部分，也不是程序内存的一部分，且既不能被读出，也不能被写入。对它的操作通过堆栈指针（SP）来实现，堆栈指针（SP）也不能读出或写入，当系统复位后堆栈指针会指向堆栈顶部。当发生子程序调用及中断时的程序计数器（PC）值被压入堆栈缓存器，当从中断或子程序返回时将数值返回给程序计数器（PC），下图说明其工作原理。

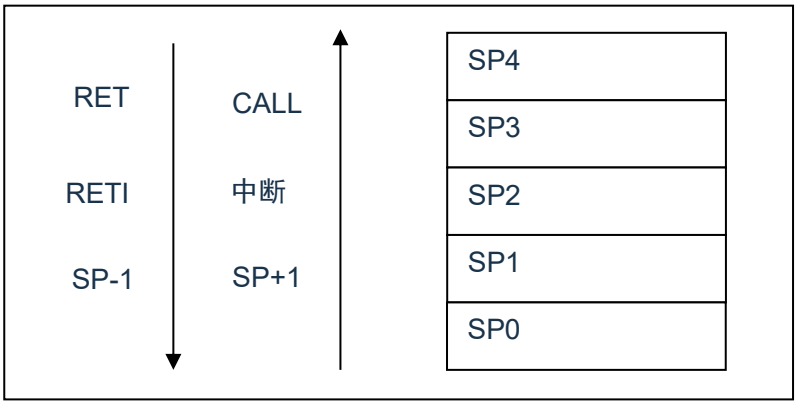


图 2-2：堆栈缓存器工作原理

堆栈缓存器的使用将遵循一个原则“先进后出”。

注：堆栈缓存器只有 5 层，如果堆栈已满，并且发生不可屏蔽的中断，那么只有中断标志位会被记录下来，而中断响应则会被抑制，直到堆栈指针发生递减，中断才会被响应，这个功能可以防止中断使堆栈溢出，同样如果堆栈已满，并且发生子程序调用，那么堆栈将会发生溢出，首先进入堆栈的内容将会丢失，只有最后 5 个返回地址被保留，故用户在写程序时应注意此点，以免发生程序走飞。

2.4 工作寄存器（ACC）

2.4.1 概述

ALU 是 8Bit 宽的算术逻辑单元，MCU 所有的数学、逻辑运算均通过它来完成。它可以对数据进行加、减、移位及逻辑运算；ALU 也控制状态位（STATUS 状态寄存器中），用来表示运算结果的状态。

ACC 寄存器是一个 8Bit 的寄存器，ALU 的运算结果可以存放在此，它并不属于数据存储器的一部分而是位于 CPU 中供 ALU 在运算中使用，因此不能被寻址，只能通过所提供的指令来使用。

2.4.2 ACC 应用

例：用 ACC 做数据传送

```
        LD      A,R01       ;将寄存器 R01 的值赋给 ACC
        LD      R02,A       ;将 ACC 的值赋给寄存器 R02
```

例：用 ACC 做立即寻址目标操作数

```
        LDIA    30H         ;给 ACC 赋值 30H
        ANDIA   30H         ;将当前 ACC 的值跟立即数 30H 进行“与”操作，
                            ;结果放入 ACC
        XORIA   30H         ;将当前 ACC 的值跟立即数 30H 进行“异或”操作，
                            ;结果放入 ACC
```

例：用 ACC 做双操作数指令的第一操作数

```
        HSUBA   R01         ;ACC-R01，结果放入 ACC
        HSUBR   R01         ;ACC-R01，结果放入 R01
```

例：用 ACC 做双操作数指令的第二操作数

```
        SUBA    R01         ;R01-ACC，结果放入 ACC
        SUBR    R01         ;R01-ACC，结果放入 R01
```

2.5 程序状态寄存器(STATUS)

STATUS 寄存器如下表所示，包含：

- ALU 的算术状态。
- 复位状态。

与其他寄存器一样，STATUS 寄存器可以是任何指令的目标寄存器。如果一条影响 Z、DC 或 C 位的指令以 STATUS 寄存器作为目标寄存器，则不能写这 3 个状态位。这些位根据器件逻辑被置 1 或清零。而且也不能写 TO 和 PD 位。因此将 STATUS 作为目标寄存器的指令可能无法得到预期的结果。

例如，CLR STATUS 会清零高 3 位，并将 Z 位置 1。这样 STATUS 的值将为 000uu1uu（其中 u=不变）。因此，建议仅使用 CLRB、SETB、SWAPA、SWAPR 指令来改变 STATUS 寄存器，因为这些指令不会影响任何状态位。

程序状态寄存器 STATUS（03H）

03H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STATUS	---	---	RP0	TO	PD	Z	DC	C
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	1	1	X	X	X

Bit7~Bit6 未用。

Bit5 RP0 存储区选择位
0= 选择Bank0
1= 选择Bank1

Bit4 TO: 超时位；
1= 上电或执行了CLRWDT指令或STOP指令；
0= 发生了WDT超时。

Bit3 PD: 掉电位；
1= 上电或执行了CLRWDT指令；
0= 执行了STOP指令。

Bit2 Z: 结果为零位；
1= 算术或逻辑运算的结果为零；
0= 算术或逻辑运算的结果不为零。

Bit1 DC: 半进位/借位位；
1= 发生了结果的低4位向高位进位或低4位没有发生借位；
0= 结果的低4位没有向高位进位或低4位向高位借位。

Bit0 C: 进位/借位位；
1= 结果的最高位发生了进位或没有发生借位；
0= 结果的最高位没有发生进位或发生了借位。

TO 和 PD 标志位可反映出芯片复位的原因，下面列出影响 TO、PD 的事件及各种复位后 TO、PD 的状态。

事件	TO	PD
电源上电	1	1
WDT 溢出	0	X
STOP 指令	1	0
CLRWDT 指令	1	1
休眠	1	0

影响 PD、TO 的事件表

TO	PD	复位原因
0	0	休眠态 WDT 溢出
0	1	非休眠态 WDT 溢出
1	1	电源上电
---	---	----

复位后 TO/PD 的状态

2.6 预分频器(OPTION_REG)

OPTION_REG 寄存器是可读写的寄存器，包括各种控制位用于配置：

- WDT 预分频器。
- 外部中断触发边沿

预分频器控制寄存器 OPTION_REG（01H）

01H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OPTION_REG	T0LSE_EN	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	1	1	1	1	0	1	1

Bit7 T0LSE_EN: TIMER0 时钟源选择 F_{LSE} 使能位
0= TIMER0 时钟源由 T0CS 决定
1= TIMER0 时钟源选择 F_{LSE}
Bit6 INTEDG: 触发中断的边沿选择位
0= INT 引脚下降沿触发中断
1= INT 引脚上升沿触发中断
Bit5 T0CS: TIMER0 时钟源选择位
0= 内部指令周期时钟（F_{CPU}）
1= T0CKI 引脚上的跳变沿
Bit4 T0SE: TIMER0 时钟源边沿选择位
0= 在 T0CKI 引脚信号从低电平跳变到高电平时递增
1= 在 T0CKI 引脚信号从高电平跳变到低电平时递增
Bit3 PSA: 预分频器分配位
0= 预分频器分配给 TIMER0 模块
1= 预分频器分配给 WDT
Bit2~Bit0 PS2~PS0: 预分频器参数配置位。

PS2	PS1	PS0	TMR0 分频比	WDT 分频比
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

预分频寄存器实际上是一个 8 位的计数器，用于监视寄存器 WDT 时，是作为一个后分频器；通常称作预分频器。

当用于 WDT 时，CLRWDT 指令将同时对预分频器和 WDT 定时器清零。

2.7 程序计数器（PC）

程序计数器（PC）控制程序内存 OTP 中的指令执行顺序，它可以寻址整个 OTP 的范围，取得指令码后，程序计数器（PC）会自动加一，指向下一个指令码的地址。但如果执行跳转、条件跳转、向 PCL 赋值、子程序调用、初始化复位、中断、中断返回、子程序返回等操作时，PC 会加载与指令相关的地址而不是下一条指令的地址。

当遇到条件跳转指令且符合跳转条件时，当前指令执行过程中读取的下一条指令将会被丢弃，且会插入一个空指令操作周期，随后才能取得正确的指令。反之，就会顺序执行下一条指令。

程序计数器（PC）是 10Bit 宽度，低 8 位通过 PCL（02H）寄存器用户可以访问，高 2 位用户不能访问。可容纳 1Kx14 位程序地址。对 PCL 赋值将会产生一个短跳转动作，跳转范围为当前页的 256 个地址。

注：当程序员在利用 PCL 作短跳转时，要先对 PC 高位缓冲寄存器 PCLATH 进行赋值。

下面给出几种特殊情况的 PC 值

复位时	PC=0000；
中断时	PC=0004（原来的 PC+1 会被自动压入堆栈）；
CALL 时	PC=程序指定地址（原来的 PC+1 会被自动压入堆栈）；
RET、RETI、RETI 时	PC=堆栈出来的值；
操作 PCL 时	PC[9:8]不变，PC[7:0]=用户指定的值；
JP 时	PC=程序指定的值；
其它指令	PC=PC+1；

2.8 看门狗计数器（WDT）

看门狗定时器（WatchDogTimer）是一个片内自振式的 RC 振荡定时器，无需任何外围组件，即使芯片的主时钟停止工作，WDT 也能保持计时。WDT 计时溢出将产生复位。

2.8.1 WDT 周期

WDT 使用 8 位预分频器。在所有复位后，WDT 溢出周期 128ms，假如你需要改变的 WDT 周期，可以设置 OPTION_REG 寄存器。WDT 的溢出周期将受到环境温度，电源电压等参数影响。

“CLRWDT”和“STOP”指令将清除 WDT 定时器以及预分频器里的计数值。WDT 一般用来防止系统失控，或者可以说是用来防止单片机程序失控。在正常情况下，WDT 应该在其溢出前被“CLRWDT”指令清零，以防止产生复位。如果程序由于某种干扰而失控，那么不能在 WDT 溢出前执行“CLRWDT”指令，就会使 WDT 溢出而产生复位。使系统重启而不至于失去控制。若是 WDT 溢出产生的复位，则状态寄存器（STATUS）的“TO”位会被清零，用户可根据此位来判断复位是否是 WDT 溢出所造成的。

注：

1. 若使用 WDT 功能，一定要在程序的某些地方放置“CLRWDT”指令，以保证在 WDT 溢出前能被清零。否者会使芯片不停的复位，造成系统无法正常工作。
2. 不能在中断程序中对 WDT 进行清零，否则无法侦测到主程序“跑飞”的情况。
3. 程序中应在主程序中有一次清 WDT 的操作，尽量不要在多个分支中清零 WDT，这种架构能最大限度发挥看门狗计数器的保护功能。
4. 看门狗计数器不同芯片的溢出时间有一定差异，所以设置清 WDT 时间时，应与 WDT 的溢出时间有较大的冗余，以避免出现不必要的 WDT 复位。

2.8.2 与看门狗控制相关的寄存器

振荡控制寄存器 OSCCON (14H)

14H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OSCCON	---	IRCF2	IRCF1	IRCF0	---	---	SWDTEN	---
R/W	---	R/W	R/W	R/W	---	---	R/W	---
复位值	---	1	0	1	---	---	0	---

Bit7 未用。

Bit6~Bit4 IRCF<2:0>: 内部振荡器频率选择位；

111= $F_{SYS} = F_{HSI}/1$；

110= $F_{SYS} = F_{HSI}/2$；

101= $F_{SYS} = F_{HSI}/4$ (默认)；

100= $F_{SYS} = F_{HSI}/8$；

011= $F_{SYS} = F_{HSI}/16$；

010= $F_{SYS} = F_{HSI}/32$；

001= $F_{SYS} = F_{HSI}/64$；

000= F_{SYS} = 32KHz(LFINTOSC)。

Bit3~Bit2 未用。

Bit1 SWDTEN： 软件使能或禁止看门狗定时器位；

1= 使能 WDT；

0= 不使能 WDT。

Bit0 未用。

注：如果 CONFIG 中 WDT 配置位=1，则 WDT 始终被使能，而与 SWDTEN 控制位的状态无关。如果 CONFIG 中 WDT 配置位=0，则可以使用 SWDTEN 控制位使能或禁止 WDT。

3. 系统时钟

3.1 概述

时钟信号由振荡器产生，在片内产生 4 个非重叠正交时钟信号，分别称作 Q1、Q2、Q3、Q4。在 IC 内部每个 Q1 使程序计数器（PC）增量加一，Q4 从程序存储单元中取出该指令，并将其锁存在指令寄存器中。在下一个 Q1 到 Q4 之间对取出的指令进行译码和执行，也就是说 4 个时钟周期才会执行一条指令。下图表示时钟与指令周期执行时序图。

一个指令周期含有 4 个 Q 周期，指令的执行和获取是采用流水线结构，取指占用一个指令周期，而译码和执行占用另一个指令周期，但是由于流水线结构，从宏观上看，每条指令的有效执行时间是一个指令周期。如果一条指令引起程序计数器地址发生改变（例如 JP）那么预取的指令操作码就无效，就需要两个指令周期来完成该条指令，这就是对 PC 操作指令都占用两个时钟周期的原因。

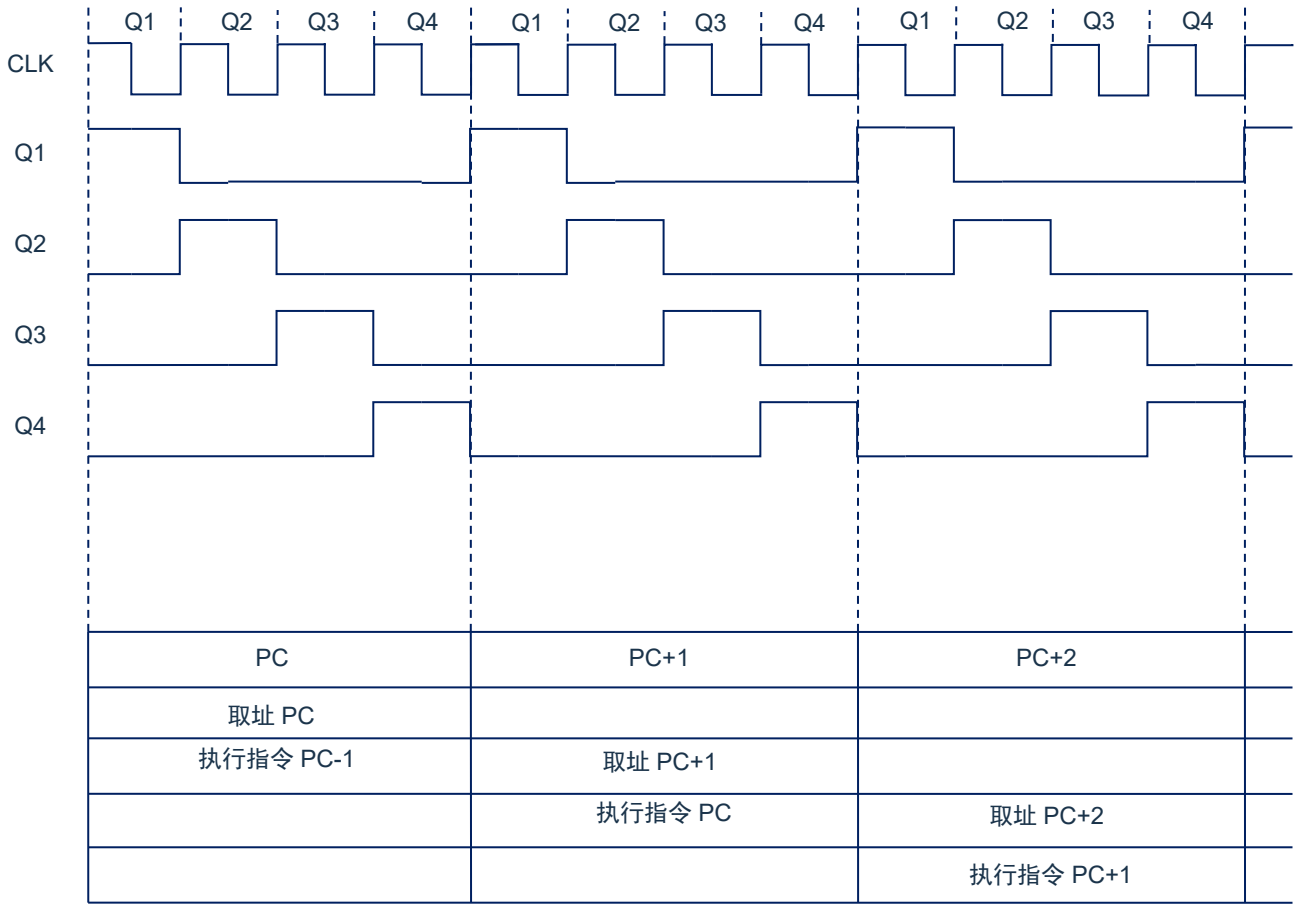


图 3-1：时钟与指令周期时序图

下面列出振荡频率与指令速度的关系：

频率	双指令周期	单指令周期
1MHz	8μs	4μs
2MHz	4μs	2μs
4MHz	2μs	1μs
8MHz	1μs	500ns

3.2 系统振荡器

芯片有 1 种振荡方式：内部 RC 振荡。

3.2.1 内部 RC 振荡

芯片默认的振荡方式为内部 RC 振荡，振荡频率固定为 16MHz，在这个基础上可通过 OSCCON 寄存器设置芯片工作频率。

3.3 起振时间

起振时间（ResetTime）是指从芯片复位到芯片振荡稳定这段时间，其设计值为 16ms。

注：无论芯片是电源上电复位，还是其它原因引起的复位，都会存在这个起振时间。

3.4 振荡器控制寄存器

振荡器控制（OSCCON）寄存器控制系统时钟和频率选择。

振荡控制寄存器 OSCCON (14H)

14H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OSCCON	---	IRCF2	IRCF1	IRCF0	---	---	SWDTEN	---
R/W	---	R/W	R/W	R/W	---	---	R/W	---
复位值	---	1	0	1	---	---	0	---

Bit7 未用。

Bit6~Bit4 IRCF<2:0>: 内部振荡器频率选择位；

111= $F_{SYS} = F_{HSI}/1$；

110= $F_{SYS} = F_{HSI}/2$；

101= $F_{SYS} = F_{HSI}/4$ (默认)；

100= $F_{SYS} = F_{HSI}/8$；

011= $F_{SYS} = F_{HSI}/16$；

010= $F_{SYS} = F_{HSI}/32$；

001= $F_{SYS} = F_{HSI}/64$；

000= F_{SYS} = 32KHz(LFINTOSC)。

Bit3~Bit2 未用。

Bit1 SWDTEN： 软件使能或禁止看门狗定时器位；

1= 使能 WDT；

0= 不使能 WDT。

Bit0 未用。

3.5 时钟框图

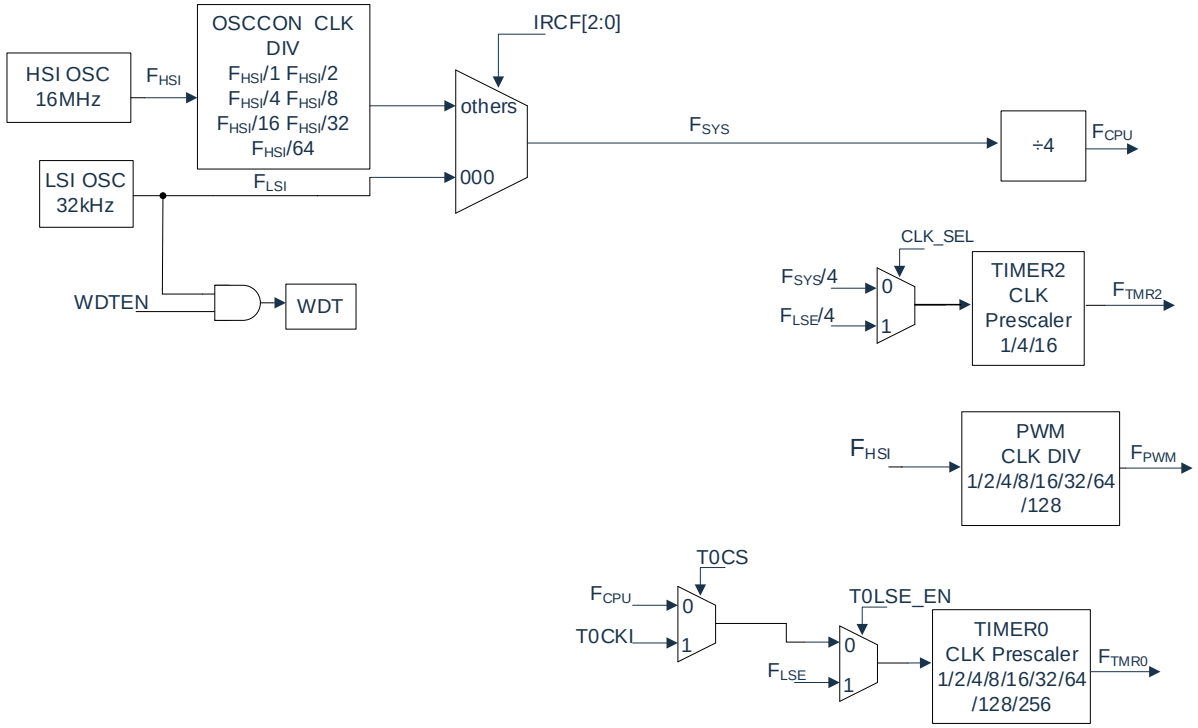


图 3-2：时钟框图

4. 复位

芯片可用如下 3 种复位方式：

- 上电复位
- LVR 复位
- 正常工作下的看门狗溢出复位

上述任意一种复位发生时，所有的系统寄存器将恢复默认状态，程序停止运行，同时程序计数器 PC 清零，复位结束后程序从复位向量 0000H 开始运行。STATUS 的 TO 和 PD 标志位能够给出系统复位状态的信息，（详见 STATUS 的说明），用户可根据 PD 和 TO 的状态，控制程序运行路径。

任何一种复位情况都需要一定的响应时间，系统提供完善的复位流程以保证复位动作的顺利进行。

4.1 上电复位

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

- 上电：系统检测到电源电压上升并等待其稳定；
- 系统初始化：所有的系统寄存器被置为初始值；
- 振荡器开始工作：振荡器开始提供系统时钟；
- 执行程序：上电结束，程序开始运行。

4.2 掉电复位

4.2.1 掉电复位概述

掉电复位针对外部因素引起的系统电压跌落情形（例如，干扰或外部负载的变化）。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。

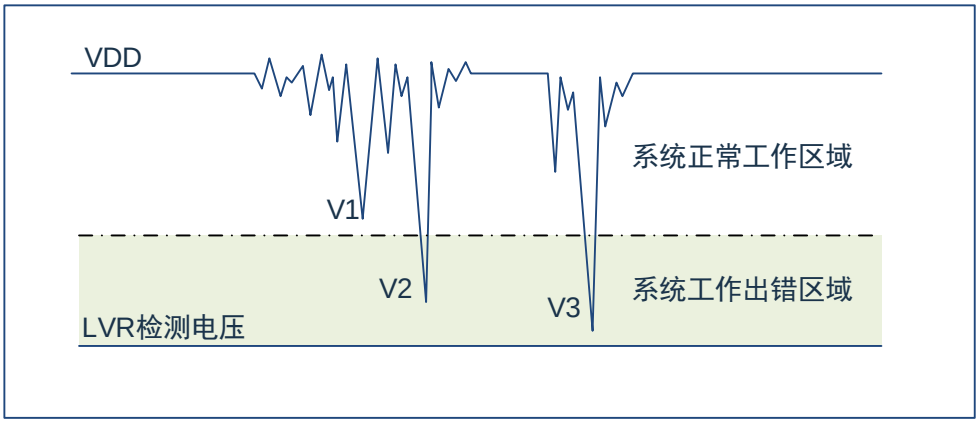


图 4-1：掉电复位示意图

上图是一个典型的掉电复位示意图。图中，VDD 受到严重的干扰，电压值降的非常低。虚线以上区域系统正常工作，在虚线以下的区域内，系统进入未知的工作状态，这个区域称作死区。当 VDD 跌至 V1 时，系统仍处于正常状态；当 VDD 跌至 V2 和 V3 时，系统进入死区，则容易导致出错。

以下情况系统可能进入死区：

- DC 运用中：
  - DC 运用中一般都采用电池供电，当电池电压过低或单片机驱动负载时，系统电压可能跌落并进入死区。这时，电源不会进一步下降到 LVD 检测电压，因此系统维持在死区。
- AC 运用中：
  - 系统采用 AC 供电时，DC 电压值受 AC 电源中的噪声影响。当外部负载过高，如驱动马达时，负载动作产生的干扰也影响到 DC 电源。VDD 若由于受到干扰而跌落至最低工作电压以下时，则系统将有可能进入不稳定工作状态。
  - 在 AC 运用中，系统上电、掉电时间都较长。其中，上电时序保护使得系统正常上电，但掉电过程却和 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。

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

4.2.2 掉电复位的改进办法

如何改进系统掉电复位性能，以下给出几点建议：

- 选择较高的 LVR 电压，有助于复位更可靠。
- 开启看门狗定时器。
- 降低系统的工作频率。
- 增大电压下降斜率。

看门狗定时器

看门狗定时器用于保证程序正常运行，当系统进入工作死区或者程序运行出错时，看门狗定时器会溢出，系统复位。

降低系统的工作速度

系统工作频率越快，系统最低工作电压越高。从而增大了工作死区的范围，降低系统工作速度就可以降低最低工作电压，从而有效的减小系统工作在死区电压的机率。

增大电压下降斜率

此方法可用于系统工作在 AC 供电的环境，一般 AC 供电系统，系统电压在掉电过程中下降很缓慢，这就会造成芯片较长时间工作在死区电压，此时若系统重新上电，芯片工作状态可能出错，建议在芯片电源与地线间加一个放电电阻，以便让 MCU 快速通过死区，进入复位区，避免芯片上电出错可能性。

4.3 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；
- 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；
- 程序：复位结束，程序开始运行。

关于看门狗定时器的应用问题请参看 2.8WDT 应用章节。

5. 休眠模式

5.1 进入休眠模式

执行 STOP 指令可进入休眠模式，如果 WDT 使能，那么：

- WDT 将被清零并继续运行。
- STATUS 寄存器中的 PD 位被清零。
- TO 位被置 1。
- 关闭振荡器驱动器。
- I/O 端口保持执行 STOP 指令之前的状态（驱动为高电平、低电平或高阻态）。

在休眠模式下，为了尽量降低电流消耗，所有 I/O 引脚都应该保持为 VDD 或 GND，没有外部电路从 I/O 引脚消耗电流。为了避免输入引脚悬空而引入开关电流，应在外部将高阻输入的 I/O 引脚拉为高电平或低电平。为了将电流消耗降至最低，还应考虑芯片内部上拉电阻的影响。

5.2 从休眠状态唤醒

可以通过下列任一事件将器件从休眠状态唤醒：

1. 看门狗定时器唤醒；
2. PORTB 电平变化中断或外设中断。

上述两种事件被认为是程序执行的延续,STATUS 寄存器中的 TO 和 PD 位用于确定器件复位的原因。PD 位在上电时被置 1，而在执行 STOP 指令时被清零。TO 位在发生 WDT 唤醒时被清零。

当执行 STOP 指令时，下一条指令(PC+1)被预先取出。如果希望通过中断事件唤醒器件，则必须将相应的中断允许位置 1（允许）。唤醒与 GIE 位的状态无关。如果 GIE 位被清零（禁止），器件将继续执行 STOP 指令之后的指令。如果 GIE 位被置 1（允许），器件执行 STOP 指令之后的指令，然后跳转到中断地址（0004h）处执行代码。如果不想执行 STOP 指令之后的指令，用户应该在 STOP 指令后面放置一条 NOP 指令。器件从休眠状态唤醒时，WDT 都将被清零，而与唤醒的原因无关。

5.3 使用中断唤醒

当禁止全局中断（GIE 被清零）时，并且有任一中断源将其中断允许位和中断标志位置 1，将会发生下列事件之一：

- 如果在执行 STOP 指令之前产生了中断，那么 STOP 指令将被作为一条 NOP 指令执行。因此，WDT 及其预分频器和后分频器（如果使能）将不会被清零，并且 TO 位将不会被置 1，同时 PD 也不会被清零。
- 如果在执行 STOP 指令期间或之后产生了中断，那么器件将被立即从休眠模式唤醒。STOP 指令将在唤醒之前执行完毕。因此，WDT 及其预分频器和后分频器（如果使能）将被清零，并且 TO 位将被置 1，同时 PD 也将被清零。即使在执行 STOP 指令之前检查到标志位为 0，它也可能在 STOP 指令执行完毕之前被置 1。要确定是否执行了 STOP 指令，可以测试 PD 位。如果 PD 位置 1，则说明 STOP 指令被作为一条 NOP 指令执行了。在执行 STOP 指令之前，必须先执行一条 CLRWDT 指令，来确保将 WDT 清零。

5.4 休眠模式应用举例

系统在进入休眠模式之前，若用户需要获得较小的休眠电流，请先确认所有 I/O 的状态，若用户方案中存在悬空的 I/O 口，把所有悬空口都设置为输出口，确保每一个输入口都有一个固定的状态，以避免 I/O 为输入状态时，口线电平处于不定态而增大休眠电流；关断 PWM 等其它外设模块；根据实际方案的功能需求可禁止 WDT 功能来减小休眠电流。

例：进入休眠的处理程序

```
SLEEP_MODE：
        CLR     INTCON      ;关断中断使能
        LDIA    B'00000000'
        LD      TRISB,A     ;所有 I/O 设置为输出口
        …                   ;关闭其它功能
        LDIA    0A5H
        LD      SP_FLAG,A   ;置休眠状态记忆寄存器(用户自定义)
        CLRWDT              ;清零 WDT
        STOP                ;执行 STOP 指令
        NOP
        NOP
```

5.5 休眠模式唤醒时间

当 MCU 从休眠态被唤醒时，需要等待一个振荡稳定时间（Reset Time），具体关系如下表所示。

系统主频时钟源	系统时钟分频选择(IRCF<2:0>)	休眠唤醒等待时间 T_{WAIT}
内部高速 RC 振荡（F_{HSI}）	$F_{SYS}=F_{HSI}$	$T_{WAIT}=128*1/F_{HSI}$
	$F_{SYS}= F_{HSI}/2$	$T_{WAIT}=128*2/F_{HSI}$
	…	…
	$F_{SYS}= F_{HSI}/64$	$T_{WAIT}=128*64/F_{HSI}$
内部低速 RC 振荡（$F_{LFINTOSC}$）	----	$T_{WAIT}=4/F_{LFINTOSC}$

6. I/O 端口

芯片有一个 I/O 端口：PORTB（最多 6 个 I/O）。可读写端口数据寄存器可直接存取这些端口。

端口	位	管脚描述	I/O
PORTB	0	施密特触发输入，可配置为推挽或开漏式输出，PWM0 输出，CMP 输出，外部中断输入	I/O
	1	施密特触发输入，可配置为推挽或开漏式输出，PWM1 输出，CMP 正端或负端输入	I/O
	2	施密特触发输入，可配置为推挽或开漏式输出，PWM4 输出，CMP 负端输入	I/O
	3	施密特触发输入，开漏式输出，编程高压输入口	I/O
	4	施密特触发输入，可配置为推挽或开漏式输出，编程时钟输入，CMP 负端输入	I/O
	5	施密特触发输入，可配置为推挽或开漏式输出，编数据输入/输出，CMP 负端输入	I/O

＜表 6-1：端口配置总概＞

6.1 I/O 口结构图

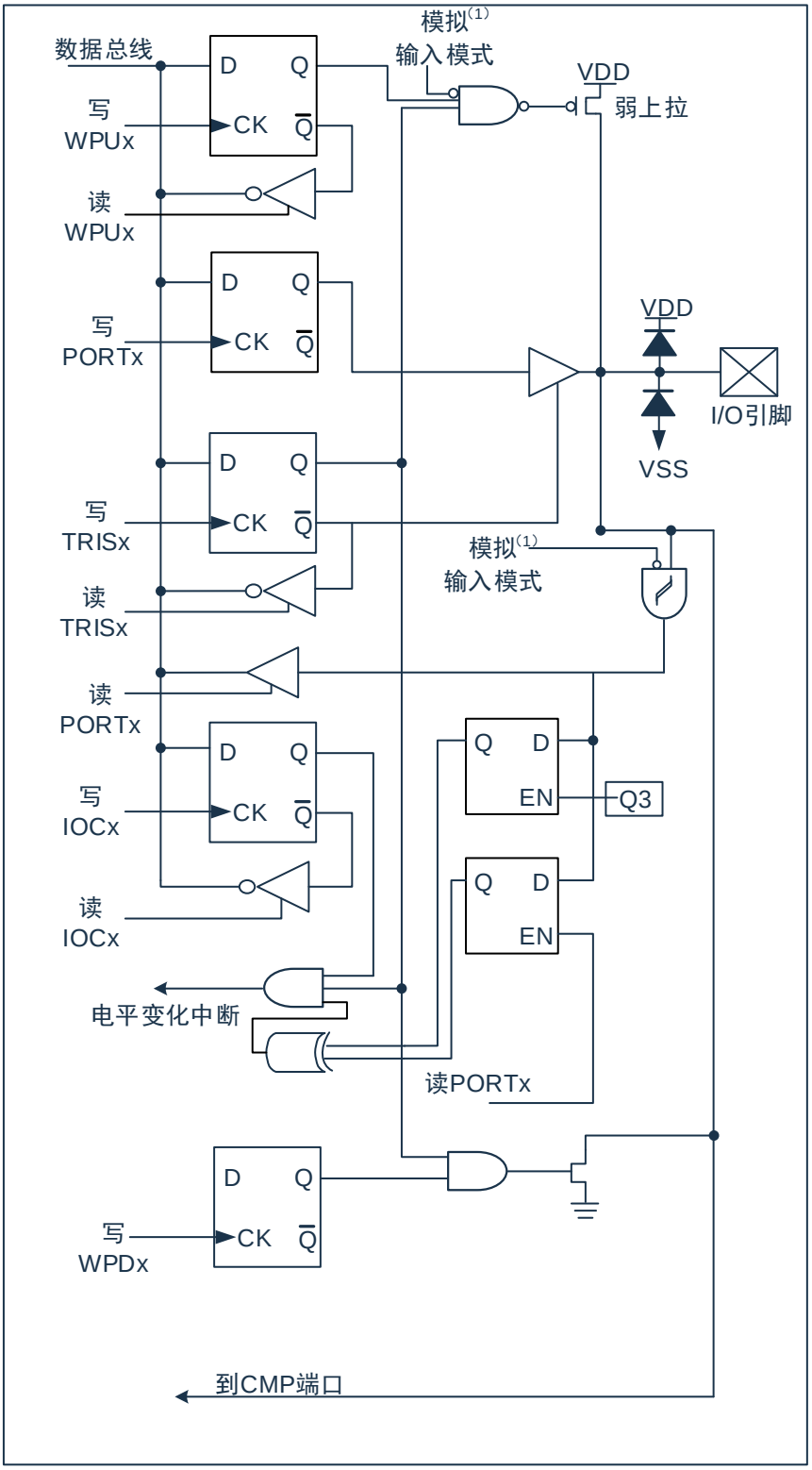


图 6-1：I/O 口结构图

注：ANSEL 决定模拟输入模式。

6.2 PORTB

6.2.1 PORTB 数据及方向

PORTB 是一个 6Bit 宽的双向端口。对应的数据方向寄存器为 TRISB。将 TRISB 中的某个位置 1（=1）可以使对应的 PORTB 引脚作为输入引脚。将 TRISB 中的某个位清零（=0）将使对应的 PORTB 引脚作为输出引脚。

读 PORTB 寄存器读的是引脚的状态而写该寄存器将会写入端口锁存器。所有写操作都是读－修改－写操作。因此，写一个端口就意味着先读该端口的引脚电平，修改读到的值，然后再将改好的值写入端口数据锁存器。即使在 PORTB 引脚用作模拟输入时，TRISB 寄存器仍然控制 PORTB 引脚的方向。当将 PORTB 引脚用作模拟输入时，用户必须确保 TRISB 寄存器中的位保持为置 1 状态。配置为模拟输入的 I/O 引脚总是读为 0。

与 PORTB 口相关寄存器有 PORTB、TRISB、WUPB、WDPB、IOCB、ODCONB 等。

PORTB 数据寄存器 PORTB(06H)

06H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PORTB	---	---	RB5	RB4	RB3	RB2	RB1	RB0
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	X	X	X	X	X	X

Bit7~Bit6　　未用。

Bit5~Bit0　　PORTB<5:0>:　PORTBI/O 引脚位；

1=　端口引脚电平>$V_{IH;}$

0=　端口引脚电平<V_{IL}。

注：RB3 口固定为开漏输出，只能输出低电平或高阻态。

PORTB 方向寄存器 TRISB (05H)

05H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TRISB	---	---	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	1	1	1	1	1	1

Bit7~Bit6　　未用。

Bit5~Bit0　　TRISB<5:0>:　PORTB 三态控制位；

1=　PORTB引脚被配置为输入（三态）；

0=　PORTB引脚被配置为输出。

例：PORTB 口处理程序

```
CLR     PORTB          ;清数据寄存器
LDIA    B'00110000'    ;设置 PORTB<5:4>为输入口，其余为输出口
LD      TRISB,A
```

6.2.2 PORTB 开漏输出控制

每个 PORTB 引脚都有可单独配置的开漏输出使能控制位（RB3 除外）。

PORTB 开漏输出使能寄存器 ODCONB(0CH)

0CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ODCONB	---	---	ODCONB5	ODCONB4	---	ODCONB2	ODCONB1	ODCONB0
R/W	---	---	R/W	R/W	---	R/W	R/W	R/W
复位值	---	---	0	0	---	0	0	0

Bit7~Bit6　　未用。

Bit5~Bit4　　ODCONB<5:4>:　PORTB 开漏输出使能；

1=　使能开漏输出；

0=　不使能开漏输出。

Bit3　　未用。

Bit2~Bit0　　ODCONB<2:0>:　PORTB 开漏输出使能；

1=　使能开漏输出；

0=　不使能开漏输出。

注：RB3 口固定为开漏输出，只能输出低电平或高阻态。

6.2.3 PORTB 上拉电阻

每个 PORTB 引脚都有可单独配置的内部弱上拉。控制位 WPUB<5:0>使能或禁止每个弱上拉。当将端口引脚配置为输出时，其弱上拉会自动切断。

PORTB 上拉电阻寄存器 WPUB(08H)

08H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPUB	---	---	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	0	0	0	0	0

Bit7　　未用。

Bit5~Bit0　　WPUB<5:0>:　PORTB 弱上拉使能位；

1=　使能上拉；

0=　禁止上拉。

注：RB0~RB2，RB4~RB5 引脚被配置为输出或者模拟输入时，将自动禁止弱上拉。
RB3 引脚被配置为输出时，其弱上拉不会被禁止。

6.2.4 PORTB 下拉电阻

每个 PORTB 引脚都有可单独配置的内部弱下拉(RB3 除外)。控制位 WPDB<5:0>使能或禁止每个弱下拉。当将端口引脚配置为输出时，其弱下拉会自动切断。

PORTB 下拉电阻寄存器 WPDB(07H)

07H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPDB	---	---	WPDB5	WPDB4	---	WPDB2	WPDB1	WPDB0
R/W	---	---	R/W	R/W	---	R/W	R/W	R/W
复位值	---	---	0	0	---	0	0	0

Bit7~Bit6　　未用。

Bit5~Bit4　　WPDB<5:4>:　PORTB 弱下拉使能位；

1=　使能下拉；

0=　禁止下拉。

Bit3　　未用。

Bit2~Bit0　　WPDB<2:0>:　PORTB 弱下拉使能位；

1=　使能下拉；

0=　禁止下拉。

注：RB0~RB2，RB4~RB5 引脚被配置为输出或者模拟输入时，将自动禁止弱下拉。
RB3 引脚无弱下拉。

6.2.5 PORTB 电平变化中断

所有的 PORTB 引脚都可以被单独配置为电平变化中断引脚。控制位 IOCB<5:0>允许或禁止每个引脚的该中断功能。上电复位时禁止引脚的电平变化中断功能。

对于已允许电平变化中断的引脚，则将该引脚上的值与上次读 PORTB 时锁存的旧值进行比较。将与上次读操作“不匹配”的输出一起进行逻辑或运算，以将 INTCON 寄存器中的 PORTB 电平变化中断标志位（RBIF）置 1。

该中断可将器件从休眠中唤醒，用户可在中断服务程序中通过以下步骤清除中断：

- 对 PORTB 进行读或写操作。这将结束引脚电平的不匹配状态。
- 将标志位 RBIF 清零。

不匹配状态会不断将 RBIF 标志位置 1。而读或写 PORTB 将结束不匹配状态，并且允许将 RBIF 标志位清零。

注：如果在执行读取操作时（Q2 周期的开始）I/O 引脚的电平发生变化，则 RBIF 中断标志位不会被置 1。此外，由于对端口的读或写影响到该端口的所有位，所以在电平变化中断模式下使用多个引脚的时候必须特别小心。在处理一个引脚电平变化的时候可能不会注意到另一个引脚上的电平变化。

PORTB 电平变化中断寄存器 IOCB(09H)

09H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOCB	---	---	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	0	0	0	0	0

Bit7~Bit6　　未用。

Bit5~Bit0　　IOCB<5:0>　PORTB 的电平变化中断控制位；

1=　允许电平变化中断；

0=　禁止电平变化中断。

6.3 I/O 使用

6.3.1 写 I/O 口

芯片的 I/O 口寄存器，和一般通用寄存器一样，可以通过数据传输指令，位操作指令等进行写操作。

例：写 I/O 口程序

```
LD      PORTB,A     ;ACC 值赋给 PORTB 口
CLRB    PORTB,1     ;PORTB.1 口清零
SET     PORTB       ;PORTB 所有输出口置 1
SETB    PORTB,1     ;PORTB.1 口置 1
```

6.3.2 读 I/O 口

例：读 I/O 口程序

```
LD      A,PORTB     ;PORTB 的值赋给 ACC
SNZB    PORTB,1     ;判断 PORTB,1 口是否为 1，为 1 跳过下一条语句
SZB     PORTB,1     ;判断 PORTB,1 口是否为 0，为 0 跳过下一条语句
```

注：当用户读一个 I/O 口状态时，若此 I/O 口为输入口，则用户读回的数据将是此口线外部电平的状态，若此 I/O 口为输出口那么读出的值将会是此口线内部输出寄存器的数据。

6.4 I/O 口使用注意事项

在操作 I/O 口时，应注意以下几个方面：

1. 当 I/O 从输出转换为输入时，要等待几个指令周期的时间，以便 I/O 口状态稳定。
2. 若使用内部上拉电阻，那么当 I/O 从输出转换为输入时，内部电平的稳定时间，与接在 I/O 口上的电容有关，用户应根据实际情况，设置等待时间，以防止 I/O 口误扫描电平。
3. 当 I/O 口为输入口时，其输入电平应在“VDD+0.3V”与“GND-0.3V”之间。若输入口电压不在此范围内可采用如下图所示方法。

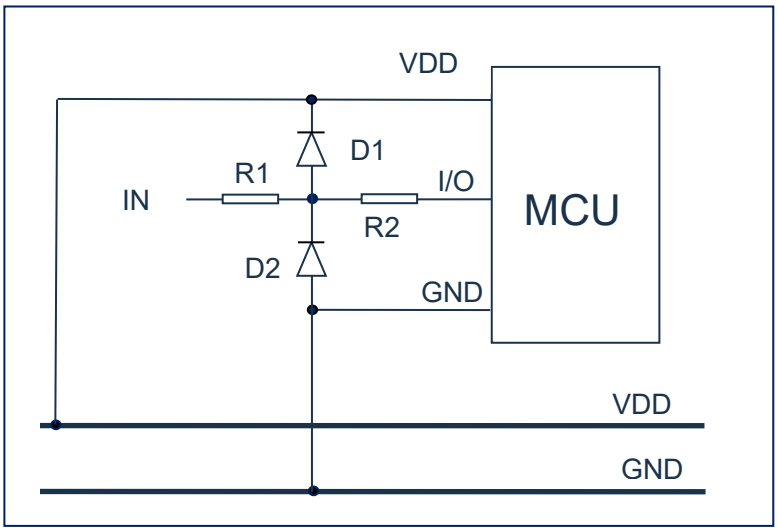


图 6-3：I/O 口注意事项连接图

4. 若在 I/O 口所在线串入较长的连接线，请在靠近芯片 I/O 的地方加上限流电阻以增强 MCU 抗 EMC 能力。

7. 中断

7.1 中断概述

芯片具有以下多种中断源：

- PORTB 电平变化中断；
- TIMER2 匹配中断；
- CMP 中断；
- PWM 中断；
- INT 中断；

中断控制寄存器（INTCON）和外设中断请求寄存器（PIR1）在各自的标志位中记录各种中断请求。INTCON 寄存器还包括各个中断允许位和全局中断允许位。

全局中断允许位 GIE（INTCON<7>）在置 1 时允许所有未屏蔽的中断，而在清零时，禁止所有中断。可以通过 INTCON、PIE1 寄存器中相应的允许位来禁止各个中断，复位时 GIE 被清零。

执行“从中断返回”指令 RETI 将退出中断服务程序并将 GIE 位置 1，从而重新允许未屏蔽的中断。

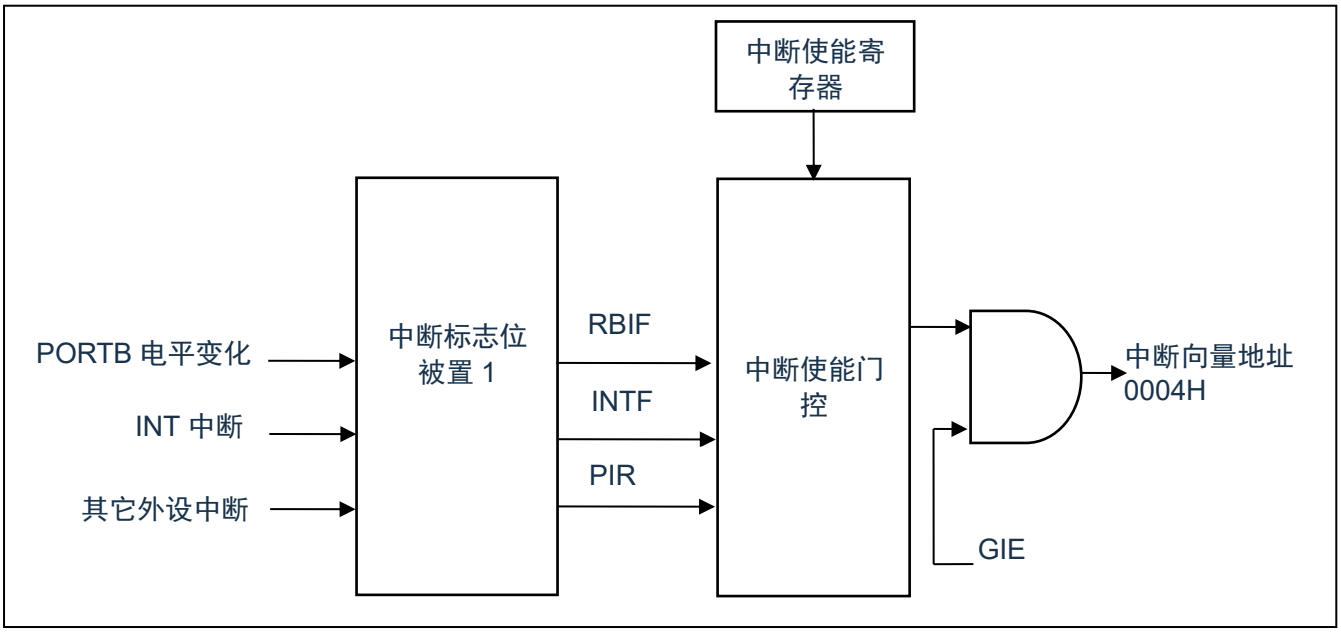


图 7-1：中断原理示意图

7.2 中断控制寄存器

7.2.1 中断控制寄存器

中断控制寄存器 INTCON 是可读写的寄存器，包含 INT 中断、PORTB 端口电平变化中断等的允许和标志位。

当有中断条件产生时，无论对应的中断允许位或（INTCON 寄存器中的）全局允许位 GIE 的状态如何，中断标志位都将置 1。用户软件应在允许一个中断之前，确保先将相应的中断标志位清零。

中断控制寄存器 INTCON（0BH）

0BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7 GIE: 全局中断允许位；
1= 允许所有未被屏蔽的中断；
0= 禁止所有中断。
Bit6 PEIE: 外设中断允许位；
1= 允许所有未被屏蔽的外设中断；
0= 禁止所有外设中断。
Bit5 T0IE: TIMER0溢出中断允许位
1= 允许TIMER0中断
0= 禁止TIMER0中断
Bit4 INTE: INT外部中断允许位；
1= 允许INT外部中断；
0= 禁止INT外部中断。
Bit3 RBIE: PORTB电平变化中断允许位；
1= 允许PORTB电平变化中断；
0= 禁止PORTB电平变化中断。
Bit2 T0IF: TIMER0溢出中断标志位（2）
1= TMR0寄存器已经溢出（必须由软件清零）
0= TMR0寄存器未发生溢出
Bit1 INTF: INT外部中断标志位；
1= 发生INT外部中断（必须由软件清零）；
0= 未发生INT外部中断。
Bit0 RBIF: PORTB电平变化中断标志位；
1= PORTB端口中至少有一个引脚的电平状态发生了改变（必须由软件清零）；
0= 没有一个PORTB通用I/O引脚的电平状态发生改变。

注：IOCB 寄存器也必须使能，相应的口线需设置为输入态。

7.2.2 外设中断允许寄存器

外设中断允许寄存器有 PIE1，在允许任何外设中断前，必须先将 INTCON 寄存器的 PEIE 位置 1。

外设中断允许寄存器 PIE1(0EH)

0EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PIE1	----	----	CMPIE	PWMIE	----	----	TMR2IE	----
R/W	----	----	R/W	R/W	----	----	R/W	----
复位值	----	----	0	0	----	----	0	----

Bit7~Bit6 未用。

Bit5 CMPIE: 比较器中断允许位
1= 允许比较器中断；
0= 禁止比较器中断。

Bit4 PWMIE: PWM中断允许位(PWM0/1)
1= 允许PWM中断；
0= 禁止PWM中断。

Bit3~Bit2 未用。

Bit1 TMR2IE: TIMER2与PR2匹配中断允许位
1= 允许TMR2与PR2匹配中断；
0= 禁止TMR2与PR2匹配中断。

Bit0 未用。

7.2.3 外设中断请求寄存器

外设中断请求寄存器为 PIR1。当有中断条件产生时，无论对应的中断允许位或全局允许位 GIE 的状态如何，中断标志位都将置 1。用户软件应在允许一个中断之前，确保先将相应的中断标志位清零。

外设中断请求寄存器 PIR1(0DH)

0DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PIR1	----	----	CMPIF	PWMIF	----	----	TMR2IF	----
R/W	----	----	R/W	R/W	----	----	R/W	----
复位值	----	----	0	0	----	----	0	----

Bit7~Bit6 未用。

Bit5 CMPIF: 比较器中断标志位(必须由软件清零)；
1= 发生比较器中断；
0= 未发生比较器中断。

Bit4 PWMIF: PWM中断标志位(PWM0/1)(必须由软件清零)；
1= 发生PWM中断；
0= 未发生PWM中断。

Bit3~Bit2 未用。

Bit1 TMR2IF: TIMER2与PR2匹配中断标志位(必须由软件清零)；
1= 发生TMR2与PR2匹配中断；
0= 未发生TMR2与PR2匹配中断。

Bit0 未用。

7.3 中断现场的保护方法

有中断请求发生并被响应后，程序转至 0004H 执行中断子程序。响应中断之前，必须保存 ACC、STATUS 的内容。芯片没有提供专用的入栈保存和出栈恢复指令，用户需自己保护 ACC 和 STATUS 的内容，以避免中断结束后可能的程序运行错误。

例：对 ACC 与 STATUS 进行入栈保护

```
                ORG     0000H
                JP      START           ;用户程序起始地址
                ORG     0004H
                JP      INT_SERVICE     ;中断服务程序
                ORG     0008H
        START：
                ...
                ...
INT_SERVICE：
        PUSH：                          ;中断服务程序入口，保存 ACC 及 STATUS
                LD      ACC_BAK,A       ;保存 ACC 的值，(ACC_BAK 需自定义)
                SWAPA   STATUS
                LD      STATUS_BAK,A    ;保存 STATUS 的值，（STATUS_BAK 需自定义）
                ...
                ...
         POP：                          ;中断服务程序出口，还原 ACC 及 STATUS
                SWAPA   STATUS_BAK
                LD      STATUS,A        ;还原 STATUS 的值
                SWAPR   ACC_BAK         ;还原 ACC 的值
                SWAPA   ACC_BAK
                RETI
```

7.4 中断的优先级，及多中断嵌套

芯片的各个中断的优先级是平等的，当一个中断正在进行的时候，不会响应另外一个中断，只有执行“RETI”指令后，才能响应下一个中断。

多个中断同时发生时，MCU 没有预置的中断优先级。首先，必须预先设定好各中断的优先权；其次，利用中断使能位和中断控制位，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。

8. 定时计数器 TIMER0

8.1 定时计数器 TIMER0 概述

TIMER0 由如下功能组成：

- 8 位定时器/计数器寄存器（TMR0）；
- 8 位预分频器（与看门狗定时器共用）；
- 可编程内部或外部时钟源；
- 可编程外部时钟边沿选择；
- 可选外部 32.768K 振荡时钟（F_{LSE}）；
- 溢出中断。

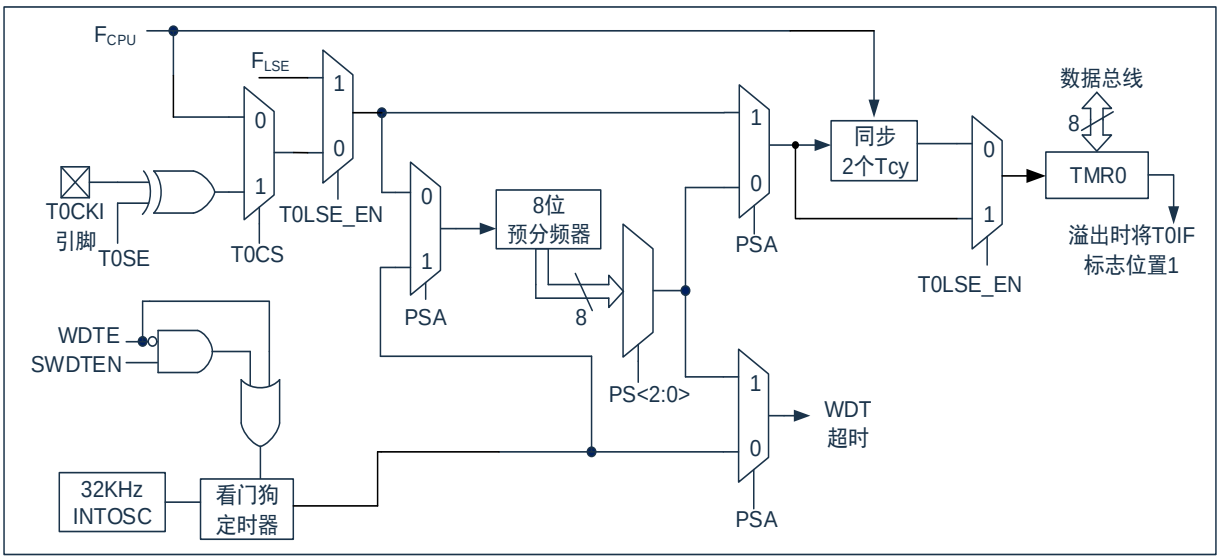


图 8-1：TIMER0/WDT 结构图

注：

1. T0SE、T0LSE_EN、T0CS、PSA、PS<2:0>为OPTION_REG寄存器中的位。
2. SWDTEN为OSCCON寄存器中的位。
3. WDTEN 位在 CONFIG 中。

8.2 TIMER0 的工作原理

TIMER0 模块既可用作 8 位定时器也可用作 8 位计数器。

8.2.1 8 位定时器模式

用作定时器时，TIMER0 模块将在每个指令周期递增（不带预分频器）。通过将 OPTION_REG 寄存器的 T0CS 位清 0 可选择定时器模式。如果对 TMR0 寄存器执行写操作，则在接下来的两个指令周期将禁止递增。可调整写入 TMR0 寄存器的值，使得在写入 TMR0 时计入两个指令周期的延时。

8.2.2 8 位计数器模式

用作计数器时，TIMER0 模块将在 T0CKI 引脚的每个上升沿或下降沿递增。递增的边沿取决于 OPTION_REG 寄存器的 T0SE 位。通过将 OPTION_REG 寄存器的 T0CS 位置 1 可选择计数器模式。

8.2.3 软件可编程预分频器

TIMER0 和看门狗定时器（WDT）共用一个软件可编程预分频器，但不能同时使用。预分频器的分配由 OPTION_REG 寄存器的 PSA 位控制。要将预分频器分配给 TIMER0，PSA 位必须清 0。

TIMER0 模块具有 8 种预分频比选择，范围为 1:2 至 1:256。可通过 OPTION_REG 寄存器的 PS<2:0>位选择预分频比。要使 TIMER0 模块具有 1:1 的预分频比，必须将预分频器分配给 WDT 模块。

预分频器不可读写。当预分频器分配给 TIMER0 模块时，所有写入 TMR0 寄存器的指令都将使预分频器清零。当预分频器分配给 WDT 时，CLRWDT 指令将同时清零预分频器和 WDT。

8.2.4 在 TIMER0 和 WDT 模块间切换预分频器

由 TIMER0 还是 WDT 使用预分频器，完全由软件控制，可以动态改变。为了避免出现不该有的芯片复位，当从 TIMER0 换为 WDT 使用时，应该执行以下指令。

```
CLR     TMR0            ;TMR0 清零
CLRWDT                  ;WDT 清零
LDIA    B'00xx1111'
LD      OPTION_REG,A
LDIA    B'00xx1xxx'     ;设置新的预分频器
LD      OPTION_REG,A
```

将预分频器从分配给 WDT 切换为分配给 TIMER0 模块，应该执行以下指令。

```
CLRWDT                  ;WDT 清零
LDIA    B'00xx0xxx'     ;设置新的预分频器
LD      OPTION_REG,A
```

8.2.5 TIMER0 中断

当 TMR0 寄存器从 FFh 溢出至 00h 时，产生 TIMER0 中断。每次 TMR0 寄存器溢出时，不论是否允许 TIMER0 中断，INTCON 寄存器的 T0IF 中断标志位都会置 1。T0IF 位必须在软件中清零。TIMER0 中断允许位是 INTCON 寄存器的 T0IE 位。

注：只有将时钟源选择 F_{LSE} 时，TIMER0 中断才能唤醒处理器。

8.3 TIMER0 相关寄存器

有两个寄存器与 TIMER0 相关，8 位定时器/计数器（TMR0）和 8 位可编程控制寄存器（OPTION_REG）。

TMR0 为一个 8 位可读写的定时/计数器，OPTION_REG 为一个 8 位可读写寄存器，用户可改变 OPTION_REG 的值，来改变 TIMER0 的工作模式等。请参看 2.6 关于预分频寄存器（OPTION_REG）的应用。

8 位定时器/计数器 TMR0（81H）

81H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR0								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	X	X	X	X	X	X	X

预分频器控制寄存器 OPTION_REG（01H）

01H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OPTION_REG	T0LSE_EN	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	1	1	1	1	0	1	1

Bit7 T0LSE_EN: TIMER0 时钟源选择 F_{LSE} 使能位
0= TIMER0 时钟源由 T0CS 决定
1= TIMER0 时钟源选择 F_{LSE}

Bit6 INTEDG: 触发中断的边沿选择位
0= INT 引脚下降沿触发中断
1= INT 引脚上升沿触发中断

Bit5 T0CS: TIMER0 时钟源选择位
0= 内部指令周期时钟（F_{CPU}）
1= T0CKI 引脚上的跳变沿

Bit4 T0SE: TIMER0 时钟源边沿选择位
0= 在 T0CKI 引脚信号从低电平跳变到高电平时递增
1= 在 T0CKI 引脚信号从高电平跳变到低电平时递增

Bit3 PSA: 预分频器分配位
0= 预分频器分配给 TIMER0 模块
1= 预分频器分配给 WDT

Bit2~Bit0 PS2~PS0: 预分配参数配置位

PS2	PS1	PS0	TMR0 分频比	WDT 分频比
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

9. 定时计数器 TIMER2

9.1 TIMER2 概述

TIMER2 模块是一个 8 位定时器/计数器，具有以下特性：

1. 8 位定时器寄存器（TMR2）
2. 8 位周期寄存器（PR2）
3. TMR2 与 PR2 匹配时中断
4. 软件可编程预分频比（1:1、1:4 和 1:16）
5. 软件可编程后分频比（1:1 至 1:16）

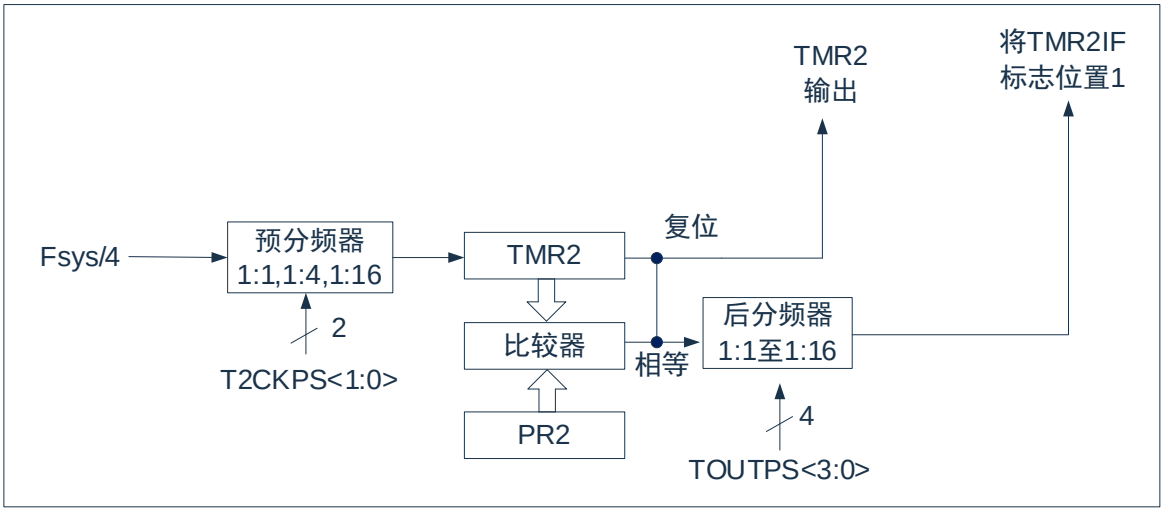


图 9-1：TIMER2 框图

9.2 TIMER2 的工作原理

TIMER2 模块的输入时钟是系统指令时钟（$F_{SYS}/4$）。时钟被输入到 TIMER2 预分频器，有如下几种分频比可供选择：1:1、1:4 或 1:16。预分频器的输出随后用于使 TMR2 寄存器递增。

持续将 TMR2 和 PR2 的值做比较以确定它们何时匹配。TMR2 将从 00h 开始递增直至与 PR2 中的值匹配。匹配发生时，会发生以下两个事件：

- TMR2 在下一递增周期被复位为 00h
- TIMER2 后分频器递增

TIMER2 与 PR2 比较器的匹配输出随后输入给 TIMER2 的后分频器，后分频器具有 1:1 至 1:16 的后分频比可供选择。TIMER2 后分频器的输出用于使 PIR1 寄存器的 TMR2IF 中断标志位置 1。

TMR2 和 PR2 寄存器均可读写。任何复位时，TMR2 寄存器均被设置为 00h 且 PR2 寄存器被设置为 FFh。

通过将 T2CON 寄存器的 TMR2ON 位置 1 使能 TIMER2；通过将 TMR2ON 位清零禁止 TIMER2。

TIMER2 预分频器由 T2CON 寄存器的 T2CKPS 位控制；TIMER2 后分频器由 T2CON 寄存器的 TOUTPS 位控制。

预分频器和后分频器计数器在以下情况下被清零：

1. 对 TMR2ON=0 时。
2. 发生任何器件复位（上电复位、看门狗定时器复位或欠压复位）。

注：写 T2CON 不会将 TMR2 清零，在 TMR2ON=0 时，TMR2 寄存器不能进行写操作。

9.3 TIMER2 相关的寄存器

有 3 个寄存器与 TIMER2 相关，分别是数据寄存器 TMR2，周期寄存器 PR2 和控制寄存器 T2CON。

TIMER2 数据寄存器 TMR2（12H）

12H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR2								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

TIMER2 周期寄存器 PR2（11H）

11H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PR2								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	1	1	1	1	1	1	1	1

TIMER2 控制寄存器 T2CON(13H)

13H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2CON	CLK_SEL	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7 CLK_SEL: 时钟源选择
1= 选择外部 FLSE/4 作为 TMR2 时钟源（休眠态可继续计数）
0= 选择内部 FSYS/4 作为 TMR2 时钟源

Bit6~Bit3 TOUTPS<3:0>: TIMER2 输出后分频比选择位。
0000= 1:1 后分频比；
0001= 1:2 后分频比；
0010= 1:3后分频比；
0011= 1:4 后分频比；
0100= 1:5 后分频比；
0101= 1:6 后分频比；
0110= 1:7 后分频比；
0111= 1:8 后分频比；
1000= 1:9 后分频比；
1001= 1:10 后分频比；
1010= 1:11 后分频比；
1011= 1:12 后分频比；
1100= 1:13 后分频比；
1101= 1:14 后分频比；
1110= 1:15 后分频比；
1111= 1:16 后分频比。

Bit2 TMR2ON: TIMER2 使能位；
1= 使能 TIMER2；
0= 禁止 TIMER2。

Bit1~Bit0 T2CKPS<1:0>: TIMER2 时钟预分频比选择位；
00= 预分频值为 1；
01= 预分频值为 4；
1x= 预分频值为 16。

10.10 位 PWM 模块（PWM0/1/4）

芯片包含一个 10 位 PWM 模块，可配置为 2 路共用周期、独立占空比的输出+1 路独立周期、独立占空比的输出，或 1 组互补输出+1 路独立输出。

10.1 引脚配置

应通过将对应的 TRIS 控制位置 0 来将相应的 PWM 引脚配置为输出。

10.2 相关寄存器说明

PWM 控制寄存器 PWMCON0(15H)

15H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCON0	CLKDIV[2:0]			PWM4EN	----	----	PWM1EN	PWM0EN
R/W	R/W	R/W	R/W	R/W	----	----	R/W	R/W
复位值	0	0	0	0	----	----	0	0

Bit7~Bit5 CLKDIV[2:0]: PWM时钟分频。
111= F_{HSI}/128
110= F_{HSI}/64
101= F_{HSI}/32
100= F_{HSI}/16
011= F_{HSI}/8
010= F_{HSI}/4
001= F_{HSI}/2
000= F_{HSI}/1

Bit4 PWM4EN: PWM4使能位；
1= 使能PWM4；
0= 禁止PWM4。

Bit3~Bit2 未用。

Bit1 PWM1EN: PWM1使能位；
1= 使能PWM1；
0= 禁止PWM1。

Bit0 PWM0EN: PWM0使能位；
1= 使能PWM0；
0= 禁止PWM0。

PWM 控制寄存器 PWMCON1(16H)

16H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCON1	---	---	---	PWM0DTEN	---	---	DT_DIV<1:0>	
R/W	---	---	---	R/W	---	---	R/W	R/W
复位值	---	---	---	0	---	---	0	0

Bit7~Bit5 未用。
Bit4 PWM0DTEN: PWM0死区使能位。
1= 使能PWM0死区功能，PWM0和PWM1组成一对互补输出；
0= 禁止PWM0死区功能。
Bit3~Bit2 未用。
Bit1~Bit0 DT_DIV<1:0> 死区时钟源分频。
11= $F_{osc}/8$
10= $F_{osc}/4$
01= $F_{osc}/2$
00= $F_{osc}/1$

PWM 控制寄存器 PWMCON2(1DH)

1DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCON2	---	---	---	PWM4DIR	---	---	PWM1DIR	PWM0DIR
R/W	---	---	---	R/W	---	---	R/W	R/W
复位值	---	---	---	0	---	---	0	0

Bit7~Bit5 未用。
Bit4 PWM4DIR PWM4输出取反控制位；
1= PWM4取反输出；
0= PWM4正常输出。
Bit3~Bit2 未用。
Bit1 PWM1DIR PWM1输出取反控制位；
1= PWM1取反输出；
0= PWM1正常输出。
Bit0 PWM0DIR PWM0输出取反控制位；
1= PWM0取反输出；
0= PWM0正常输出。

PWM0~PWM1 周期低位寄存器 PWMTL (17H)

17H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMTL	PWMT<7:0>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 PWMT[7:0]: PWM0~PWM1周期低8位。

PWM4 周期低位寄存器 PWMT4L(1CH)

1CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMT4L	PWM4T<7:0>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 PWM4T<7:0>: PWM4周期低8位。

周期高位寄存器 PWMTH (18H)

18H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMTH	---	---	PWMD4<9:8>		PWM4T<9:8>		PWMT<9:8>	
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	0	0	0	0	0

Bit7~Bit6 未用。
Bit5~Bit4 PWMD4<9:8>: PWM4占空比高2位。
Bit3~Bit2 PWM4T<9:8>: PWM4周期高2位。
Bit1~Bit0 PWMT<9:8>: PWM0~PWM1周期高2位。

注：写入 PWMD4<9:8>并不能立即生效，需有写入 PWMD4L 操作后才能生效。

PWM0 占空比低位寄存器 PWMD0L (19H)

19H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD0L	PWMD0<7:0>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 PWMD0<7:0>: PWM0占空比低8位。

PWM1 占空比低位寄存器 PWMD1L (1AH)

1AH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD1L	PWMD1<7:0>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 PWMD1<7:0>: PWM1占空比低8位。

PWM4 占空比低位寄存器 PWMD4L (1BH)

1BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD4L	PWMD4<7:0>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 PWMD4<7:0>: PWM4占空比低8位。

PWM0 和 PWM1 占空比高位寄存器 PWMD01H (1EH)

1EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD01H	---	---	PWMD1<9:8>		---	---	PWMD0<9:8>	
R/W	---	---	R/W	R/W	---	----	R/W	R/W
复位值	---	---	0	0	---	----	0	0

Bit7~Bit6 未用。
Bit5~Bit4 PWMD1<9:8>: PWM1占空比高2位。
Bit3~Bit2 未用。
Bit1~Bit0 PWMD0<9:8>: PWM0占空比高2位。

注：写入 PWMD0<9:8>并不能立即生效，需有写入 PWMD0L 操作后才能生效。写入 PWMD1<9:8>并不能立即生效，需有写入 PWMD1L 操作后才能生效。

PWM0 和 PWM1 死区时间寄存器 PWM01DT(1FH)

1FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWM01DT	---	---	PWM01DT<5:0>					
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	0	0	0	0	0

Bit7~Bit6 未用。
Bit5~Bit0 PWM01DT<5:0>: PWM0和PWM1死区时间。

10.3 10 位 PWM 寄存器写操作顺序

由于 10 位 PWM 占空比数值分配在两个寄存器中，在修改占空比时，程序总是先后修改这两个寄存器，为了保证占空比数值的正确性，芯片内部设计了缓存加载功能。操作 10 位占空比数值需严格按照以下顺序进行：

1）写高 2 位数值，此时高 2 位数值只是写入内部的缓存；

2）写低 8 位数值，此时完整的 10 位占空比数值被锁存；

3）以上操作顺序只针对 PWM0、PWM1、PWM4 占空比寄存器。

10.4 10 位 PWM 周期

PWM 周期是通过写 PWMTH 和 PWMTL 寄存器来指定的。

公式 1：PWM 周期计算公式：

$$\text{PWM 周期}=[\text{PWMT}+1]*T_{HSI}*(\text{CLKDIV 分频值})$$

注：$T_{HSI}=1/F_{HSI}$

当 PWM 周期计数器等于 PWMT 时，在下一个递增计数周期中会发生以下 5 个事件：

- PWM 周期计数器被清零；
- PWMx 引脚被置 1；
- PWM 新周期值被锁存；
- PWM 新占空比值被锁存；
- 产生 PWM 中断标志位；

10.5 10 位 PWM 占空比

可通过将一个 10 位值写入以下多个寄存器来指定 PWM 占空比：PWMDxL、PWMDxxH。

可以在任何时候写入 PWMDxL 和 PWMDxxH 寄存器，但直到 PWM 周期计数器等于 PWMT（即周期结束）时，占空比的值才被更新到内部锁存器中。

公式 2：脉冲宽度计算公式：

$$\text{脉冲宽度} = (PWMDx[9:0]+1)*T_{HSI}*(CLKDIV\ \text{分频值})$$

公式3：PWM占空比计算公式：

$$\text{占空比}=\frac{PWMDx[9:0]+1}{PWMT[9:0]+1}$$

PWM 占空比在芯片内部有双重缓冲。这种双重缓冲结构极其重要，可以避免在 PWM 操作过程中产生毛刺。

10.6 系统时钟频率的改变

PWM 频率只与芯片振荡时钟有关，系统时钟频率发生任何改变都不会影响 PWM 频率。

10.7 10 位 PWM 设置

使用 PWM 模块时应该执行以下步骤：

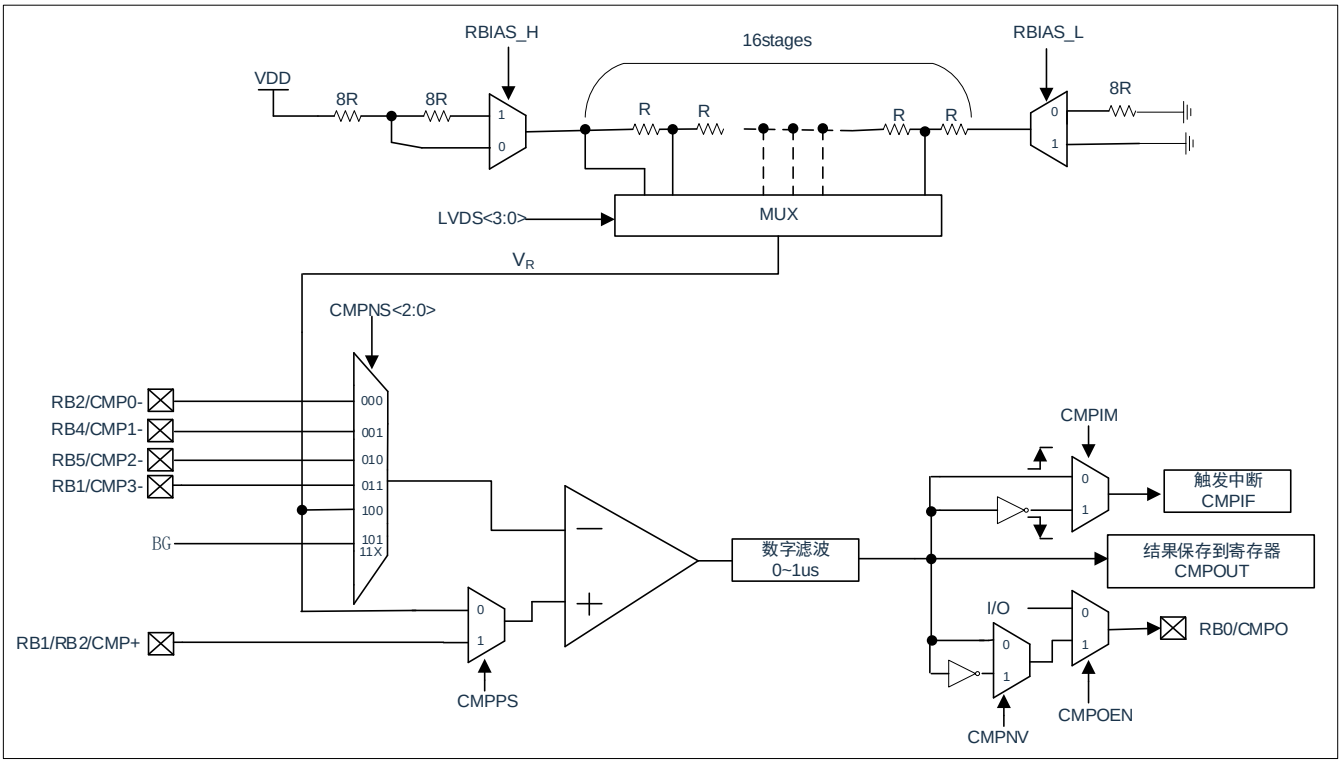
1. 通过将相应的 TRIS 位置 1，使之成为输入引脚。
2. 通过装载 PWMTH，PWMTL 寄存器设置 PWM 周期。
3. 通过装载 PWMDxL，PWMDxxH 寄存器设置 PWM 占空比。
4. 清零 PWMIF 标志位。
5. 设置 PWMENx 位以使能相应 PWM 输出。
6. 在新的 PWM 周期开始后，使能 PWM 输出：
   - 等待 PWMIF 位置 1；
   - 通过将相应的 TRIS 位清零，使能 PWM 引脚输出驱动器。

11. 比较器（COMP）

11.1 特性

- 内部集成 1 个比较器；
- 比较器失调电压≤±13mv；
- 输入共模电压范围：0V~VDD-1.3V；
- 内置 1 个电阻分压模块，参考电压为 VDD；
- 比较器结果可选上升沿或下降沿触发中断；
- 比较器结果可选择从 RB0 口输出，且支持取反输出。

11.2 框图



10-1：比较器的功能框图

11.3 比较器相关功能

11.3.1 比较器功能描述

SC8P011x 芯片内部集成了 1 个比较器模块，图 10-1 显示了它的硬件框图。比较器正端输入可通过配置 CMPCON0 寄存器的 CMPPS 位来选择 CMP+端口或者内部电阻分压输出信号 V_R；负端输入可通过配置 CMPCON0 寄存器的 CMPNS<2:0>位来选择 CMPx-端口、内部电阻分压输出信号 V_R 或者 1.2V BG 电压。当比较器正端输入电压大于负端输入电压时，比较器经过数字滤波后输出 1；反之，如果比较器正端输入电压小于负端输入电压，则比较器经过数字滤波后输出 0。

11.3.2 比较器内部电阻分压输出

比较器内部集成了一个内部电阻分压模块，参考电压为 VDD。可通过配置 CMPCON1 寄存器的 RBIAS_H、RBIAS_L、LVDS<3:0>等控制位的值来获得不同的电阻分压输出 V_R，根据 RBIAS_H、RBIAS_L 这两个位的不同值，V_R 有如下 4 种计算公式：

公式 1：RBIAS_H=0，RBIAS_L=0

$$V_R=\frac{1}{4}*VDD+\frac{n+1}{32}*VDD$$

公式 2：RBIAS_H=0，RBIAS_L=1

$$V_R=\frac{n+1}{24}*VDD$$

公式 3：RBIAS_H=1，RBIAS_L=0

$$V_R=\frac{1}{5}*VDD+\frac{n+1}{40}*VDD$$

公式 4：RBIAS_H=1，RBIAS_L=1

$$V_R=\frac{n+1}{32}*VDD$$

根据图 10-1 的比较器结构框图和以上公式可知，当比较器的负端选择 BG 1.2V，正端选择内部电阻分压输出 V_R 时，可以通过比较器来监测 VDD 电压，VDD 的计算公式如下所示：

对公式 1 而言：

$$VDD=\frac{32}{n+9}*1.2$$

对公式 2 而言：

$$VDD=\frac{24}{n+1}*1.2$$

对公式 3 而言：

$$VDD=\frac{40}{n+9}*1.2$$

对公式 4 而言：

$$VDD=\frac{32}{n+1}*1.2$$

注：n 为 LVDS<3:0>的值，即 n=0，1，2……14，15。

11.3.1 比较器监测电源电压

根据图 10-1 的比较器结构框图和 10.3.2 里的公式可知，当比较器的负端选择 BG 1.2V，正端选择内部电阻分压输出 V_R 时，可以通过比较器来监测电源电压，当电源电压低于设定值时比较器输出 0，电源电压高于设定值时比较器输出 1，通过配置 RBIAS_H、RBIAS_L、LVDS[3:0]的值可设定不同的电压电压监测点，具体如下表：

RBIAS_H	RBIAS_L	LVDS[3:0]	监测值(V)	RBIAS_H	RBIAS_L	LVDS[3:0]	监测值(V)	RBIAS_H	RBIAS_L	LVDS[3:0]	监测值(V)
0	1	0101	4.80	0	0	0100	2.95	1	0	1101	2.18
1	0	0010	4.36	0	1	1001	2.88	0	0	1001	2.13
0	0	0000	4.27	1	0	1000	2.82	1	0	1110	2.09
0	1	0110	4.11	0	0	0101	2.74	0	1	1101	2.06
1	0	0011	4.00	1	0	1001	2.67	0	0	1010	2.02
0	0	0001	3.84	0	1	1010	2.62	1	0	1111	2.00
1	0	0100	3.69	0	0	0110	2.56	-	-	-	-
0	1	0111	3.60	1	0	1010	2.53	-	-	-	-
0	0	0010	3.49	0	0	0111	2.40	-	-	-	-
1	0	0101	3.43	1	0	1100	2.29	-	-	-	-
0	0	0011	3.20	0	0	1000	2.26	-	-	-	-

11.3.2 比较器中断使用

在使用比较器的时候，若要使用中断功能则可以通过以下配置步骤来开启比较器中断：

- 配置寄存器 CMPCON0 的 CMPPS 位选择正端输入；
- 配置寄存器 CMPCON0 的 CMPNS<2:0>位选择负端输入；
- 配置寄存器 CMPCON1 的 CMPIM 位选择上升沿或者下降沿触发中断；
- 配置寄存器 CMPCON0 的 CMPEN 位使能比较器；
- 延时等待 10us；
- 清零 PIR1 寄存器的 CMPIF 位；
- 置 1 PIE1 寄存器的 CMPIE 位，使能比较器中断；
- 置 1 INTCON 寄存器的 PEIE、GIE 位，开启外设中断和全局中断。

11.3.3 比较器中断休眠唤醒

比较器中断可将芯片从休眠模式下唤醒，具体配置可参考如下程序例程：

例：比较器中断休眠唤醒程序

```
SLEEP_MODE：
        LDIA    B'00000110'
        LD      TRISB,A         ;将 RB1/CMP+,RB2/CMP0-配置为输入口
        ...                     ;关闭其它功能
        LDIA    00H
        LD      CMPCON0,A       ;CMPNS<2:0>=000，负端选择 RB2/CMP0-
        SETB    CMPCON0,6       ;CMPPS=1，正端选择 RB1/CMP+
        SETB    CMPCON1,6       ;ANSEL=1，将 CMP+、CMP0-设为模拟口，以
                                ;便降低休眠功耗
        SETB    CMPCON1,7       ;CMPIM=1，选择下降沿触发中断
        SETB    CMPCON0,7       ;使能比较器
        CALL    DELAY10US       ;使能比较器后延时等待比较器输出稳定
        CLRB    PIR1,5          ;清零 CMPIF（必须）
        SETB    PIE1,5          ;使能比较器中断
        SETB    INTCON,6        ;允许外设中断
        SETB    INTCON,7        ;开启总中断使能，唤醒后程序跳转到中断向量
                                ;地址 0004H 执行
        CLRWDT                  ;清零 WDT
        STOP                    ;执行 STOP 指令
        NOP
        NOP
```

11.3.4 比较器结果输出引脚配置

比较器的结果经过数字滤波后，通过读寄存器 CMPCON0 的 CMPOUT 位得到当前比较的结果；还可以通过以下的配置步骤输出到 RB0/CMPO 引脚：

- 将 TRISB0 控制位置 0 来将 RB0/CMPO 引脚配置为输出口；
- 配置寄存器 CMPCON0 的 CMPNV 位来选择正向输出或反向输出；
- 将寄存器 CMPCON0 的 CMPOEN 位置 1 来使能 CMPOUT 输出到 RB0/CMPO 引脚。

11.4 相关寄存器

比较器控制寄存器 CMPCON0(0FH)

0FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CMPCON0	CMPEN	CMPPS	CMPNS2	CMPNS1	CMPNS0	CMPNV	CMPOUT	CMPOEN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
复位值	0	0	0	0	0	0	0	0

Bit7 CMPEN: CMP使能位；
1= 使能CMP；
0= 禁止CMP。
Bit6 CMPPS: CMP正端输入选择位；
1= CMP+端口电压；
0= VDD经过内部电阻分压后的电压。
Bit5~Bit3 CMPNS<2:0>: CMP负端输入选择位；
000= CMP0- 端口电压；
001= CMP1- 端口电压
010= CMP2-端口电压；
011= CMP3-端口电压。
100= VDD经过内部电阻分压后的电压；
101= BG；
11x= BG。
Bit2 CMPNV: CMPO端口输出取反控制位；
1= CMPOUT在CMPO端口取反输出；
0= CMPOUT在CMPO端口正常输出。
Bit1 CMPOUT: CMP结果位。
Bit0 CMPOEN: CMPO端口输出使能位；
1= 使能CMPO端口输出CMPOUT；
0= 禁止CMPO端口输出CMPOUT。

比较器控制寄存器 CMPCON1(10H)

10H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CMPCON1	CMPIM	ANSEL	RBIAS_H	RBIAS_L	LVDS<3:0>			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7 CMPIM: CMP中断触发边沿选择；
1= CMP输出的下降沿触发中断；
0= CMP输出的上升沿触发中断。
Bit6 ANSEL: 模拟选择位，选择CMP+、CMPX-的模拟或数字功能；
1= 模拟口；
0= 数字口。
Bit5 RBIAS_H: 具体用法参考比较器框图。
Bit4 RBIAS_L: 具体用法参考比较器框图。
Bit3~Bit0 LVDS<3:0>: 内部电阻分压比选择位。

注：ANSEL 位只对被选作比较器功能的 IO 端口有效。

12. 电气参数

12.1 极限参数

电源供应电压.....GND-0.3V~GND+6.0V
存储温度.....-50℃~125℃
工作温度.....-40℃~85℃
端口输入电压.....GND-0.3V~VDD+0.3V
所有端口最大灌电流.....200mA
所有端口最大拉电流.....-150mA

注：如果器件工作条件超过上述“极限参数”，可能会对器件造成永久性损坏。上述值仅为运行条件极大值，我们不建议器件在该规范规定的范围以外运行。器件长时间工作在极限值条件下，其稳定性会受到影响。

12.2 DC 特性

（VDD=5V，T_A= 25℃，除非另有说明）

符号	参数	测试条件		最小	典型	最大	单位
		VDD	条件				
VDD	工作电压	-	F_{SYS}=16MHz	2.6	-	5.5	V
		-	F_{SYS}=8MHz	1.8	-	5.5	V
I_{DD}	工作电流	5V	F_{SYS}=16MHz, 所有模拟模块关闭	-	2	-	mA
		5V	F_{SYS}=8MHz, 所有模拟模块关闭	-	1	-	mA
		3V	F_{SYS}=16MHz, 所有模拟模块关闭	-	1.2	-	mA
		3V	F_{SYS}=8MHz, 所有模拟模块关闭	-	0.8	-	mA
I_{STB}	静态电流	5V	----	-	0.1	2	uA
		3V	----	-	0.1	2	uA
V_{IL}	低电平输入电压	-	----	-	-	0.3VDD	V
V_{IH}	高电平输入电压	-	----	0.7VDD	-	-	V
V_{OH}	高电平输出电压	-	不带负载	0.9VDD	-	-	V
V_{OL}	低电平输出电压	-	不带负载	-	-	0.1VDD	V
V_{REF}	内部固定参考电压	-	----	1.188	1.2	1.212	V
R_{PH}	上拉电阻阻值	5V	Vo=0.5VDD	-	32	-	KΩ
		3V	Vo=0.5VDD	-	52	-	KΩ
R_{PL}	下拉电阻阻值	5V	Vo=0.5VDD	-	36	-	KΩ
		3V	Vo=0.5VDD	-	50	-	KΩ
I_{OL}	输出口灌电流	5V	V_{OL}=0.3VDD	-	50	-	mA
		3V	V_{OL}=0.3VDD	-	25	-	mA
I_{OH}	输出口拉电流	5V	V_{OH}=0.7VDD	-	18	-	mA
		3V	V_{OH}=0.7VDD	-	8	-	mA

12.3 比较器特性

（TA= 25℃，除非另有说明）

符号	参数	测试条件	最小值	典型值	最大值	单位
VDD	工作电压范围	-	2.0	-	5.5	V
Temp	温度范围	-	-40	-	85	℃
Iwork	工作电流	VDD=5V COMP+=2V COMP-=2V	-	34	46	uA
		VDD=3V COMP+=1V COMP-=1V	-	20	26	uA
I_{BG}	BG工作电流	VDD=5V	-	35	46	uA
		VDD=3V	-	33	44	uA
V_{IN}	输入共模电压范围	-	0	-	VDD-1.3	V
Vos	失调电压	-	-	-	±13	mV
LSB	最小分辨率	-	-	10	20	mV
Tr	响应时间	-	-	-	6	us
-	内部电阻分压误差	VDD=5V V_R>1V	-1%	-	＋1%	-
		VDD=5V V_R<1V	-2%	-	＋2%	-

备注：V_R 为内部电阻分压输出值

12.4 上电复位特性

（T_A= 25℃，除非另有说明）

符号	参数	测试条件	最小值	典型值	最大值	单位
t_{VDD}	VDD 上升速率	-	0.05			V/ms
V_{LVR1}	LVR 设定电压=1.8V	VDD=1.6~5.5V	1.7	1.8	1.9	V
V_{LVR2}	LVR 设定电压=2.0V	VDD=1.8~5.5V	1.9	2.0	2.1	V
V_{LVR3}	LVR 设定电压=2.5V	VDD=2.4~5.5V	2.5	2.5	2.7	V
V_{LVR3}	LVR 设定电压=3.0V	VDD=2.8~5.5V	2.9	3.0	3.1	V

12.5 AC 特性

（T_A= 25℃，除非另有说明）

符号	参数	测试条件		最小值	典型值	最大值	单位
		VDD	条件				
T_{WDT}	WDT 复位时间	5V	---	-	16	-	ms
		3V	---	-	16	-	ms
F_{INTRC}	内振频率 16MHz	VDD=4.0~5.5V TA=25℃		-1.5%	16	+1.5%	MHz
		VDD=2.5~5.5V TA=25℃		-2.0%	16	+2.0%	MHz
		VDD=4.0~5.5V TA= -40~85℃		-3.5%	16	+3.5%	MHz
		VDD=2.5~5.5V TA= -40~85℃		-5%	16	+5%	MHz

13. 指令

13.1 指令一览表

助记符		操作	指令周期	标志
控制类				
NOP		空操作	1	None
STOP		进入休眠模式	1	TO,PD
CLRWDT		清零看门狗计数器	1	TO,PD
数据传送				
LD	[R],A	将 ACC 内容传送到 R	1	NONE
LD	A,[R]	将 R 内容传送到 ACC	1	Z
TESTZ	[R]	将数据存储器内容传给数据存储器	1	Z
LDIA	i	立即数 i 送给 ACC	1	NONE
逻辑运算				
CLRA		清零 ACC	1	Z
SET	[R]	置位数据存储器 R	1	NONE
CLR	[R]	清零数据存储器 R	1	Z
ORA	[R]	R 与 ACC 内容做“或”运算，结果存入 ACC	1	Z
ORR	[R]	R 与 ACC 内容做“或”运算，结果存入 R	1	Z
ANDA	[R]	R 与 ACC 内容做“与”运算，结果存入 ACC	1	Z
ANDR	[R]	R 与 ACC 内容做“与”运算，结果存入 R	1	Z
XORA	[R]	R 与 ACC 内容做“异或”运算，结果存入 ACC	1	Z
XORR	[R]	R 与 ACC 内容做“异或”运算，结果存入 R	1	Z
SWAPA	[R]	R 寄存器内容的高低半字节转换，结果存入 ACC	1	NONE
SWAPR	[R]	R 寄存器内容的高低半字节转换，结果存入 R	1	NONE
COMA	[R]	R 寄存器内容取反，结果存入 ACC	1	Z
COMR	[R]	R 寄存器内容取反，结果存入 R	1	Z
XORIA	i	ACC 与立即数 i 做“异或”运算，结果存入 ACC	1	Z
ANDIA	i	ACC 与立即数 i 做“与”运算，结果存入 ACC	1	Z
ORIA	i	ACC 与立即数 i 做“或”运算，结果存入 ACC	1	Z
移位操作				
RRCA	[R]	数据存储器带进位循环右移一位，结果存入 ACC	1	C
RRCR	[R]	数据存储器带进位循环右移一位，结果存入 R	1	C
RLCA	[R]	数据存储器带进位循环左移一位，结果存入 ACC	1	C
RLCR	[R]	数据存储器带进位循环左移一位，结果存入 R	1	C
RLA	[R]	数据存储器不带进位循环左移一位，结果存入 ACC	1	NONE
RLR	[R]	数据存储器不带进位循环左移一位，结果存入 R	1	NONE
RRA	[R]	数据存储器不带进位循环右移一位，结果存入 ACC	1	NONE
RRR	[R]	数据存储器不带进位循环右移一位，结果存入 R	1	NONE
递增递减				
INCA	[R]	递增数据存储器 R，结果放入 ACC	1	Z
INCR	[R]	递增数据存储器 R，结果放入 R	1	Z
DECA	[R]	递减数据存储器 R，结果放入 ACC	1	Z
DECR	[R]	递减数据存储器 R，结果放入 R	1	Z

助记符		操作	指令周期	标志
位操作				
CLRB	[R],b	将数据存储器 R 中某位清零	1	NONE
SETB	[R],b	将数据存储器 R 中某位置一	1	NONE
数学运算				
ADDA	[R]	ACC+[R]→ACC	1	C,DC,Z,OV
ADDR	[R]	ACC+[R]→R	1	C,DC,Z,OV
ADDCA	[R]	ACC+[R]+C→ACC	1	Z,C,DC,OV
ADDCR	[R]	ACC+[R]+C→R	1	Z,C,DC,OV
ADDIA	i	ACC+i→ACC	1	Z,C,DC,OV
SUBA	[R]	[R]-ACC→ACC	1	C,DC,Z,OV
SUBR	[R]	[R]-ACC→R	1	C,DC,Z,OV
SUBCA	[R]	[R]-ACC-C→ACC	1	Z,C,DC,OV
SUBCR	[R]	[R]-ACC-C→R	1	Z,C,DC,OV
SUBIA	i	i-ACC→ACC	1	Z,C,DC,OV
HSUBA	[R]	ACC-[R]→ACC	1	Z,C,DC,OV
HSUBR	[R]	ACC-[R]→R	1	Z,C,DC,OV
HSUBCA	[R]	ACC-[R]- $\overline{C}$ →ACC	1	Z,C,DC,OV
HSUBCR	[R]	ACC-[R]- $\overline{C}$ →R	1	Z,C,DC,OV
HSUBIA	i	ACC-i→ACC	1	Z,C,DC,OV
无条件转移				
RET		从子程序返回	2	NONE
RET	i	从子程序返回，并将立即数 I 存入 ACC	2	NONE
RETI		从中断返回	2	NONE
CALL	ADD	子程序调用	2	NONE
JP	ADD	无条件跳转	2	NONE
条件转移				
SZB	[R],b	如果数据存储器 R 的 b 位为“0”，则跳过下一条指令	1 or 2	NONE
SNZB	[R],b	如果数据存储器 R 的 b 位为“1”，则跳过下一条指令	1 or 2	NONE
SZA	[R]	数据存储器 R 送至 ACC，若内容为“0”，则跳过下一条指令	1 or 2	NONE
SZR	[R]	数据存储器 R 内容为“0”，则跳过下一条指令	1 or 2	NONE
SZINCA	[R]	数据存储器 R 加“1”，结果放入 ACC，若结果为“0”，则跳过下一条指	1 or 2	NONE
SZINCR	[R]	数据存储器 R 加“1”，结果放入 R，若结果为“0”，则跳过下一条指令	1 or 2	NONE
SZDECA	[R]	数据存储器 R 减“1”，结果放入 ACC，若结果为“0”，则跳过下一条指	1 or 2	NONE
SZDECR	[R]	数据存储器 R 减“1”，结果放入 R，若结果为“0”，则跳过下一条指令	1 or 2	NONE

13.2 指令说明

ADDA **[R]**

操作: 将 R 加 ACC，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;将 ACC 的值（09H）赋给自定义寄存器 R01
LDIA    077H    ;给 ACC 赋值 77H
ADDA    R01     ;执行结果：ACC=09H + 77H =80H
```

ADDR **[R]**

操作: 将 R 加 ACC，结果放入 R

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;将 ACC 的值（09H）赋给自定义寄存器 R01
LDIA    077H    ;给 ACC 赋值 77H
ADDR    R01     ;执行结果：R01=09H + 77H =80H
```

ADDCA **[R]**

操作: 将 R 加 ACC 加 C 位，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;将 ACC 的值（09H）赋给自定义寄存器 R01
LDIA    077H    ;给 ACC 赋值 77H
ADDCA   R01     ;执行结果：ACC= 09H + 77H + C=80H (C=0)
                          ACC= 09H + 77H + C=81H (C=1)
```

ADDCR **[R]**

操作: 将 R 加 ACC 加 C 位，结果放入 R

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;将 ACC 的值（09H）赋给自定义寄存器 R01
LDIA    077H    ;给 ACC 赋值 77H
ADDCR   R01     ;执行结果：R01 = 09H + 77H + C=80H (C=0)
                          R01 = 09H + 77H + C=81H (C=1)
```

ADDIA **i**

操作: 将立即数 i 加 ACC，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

```
LDIA    09H     ;给 ACC 赋值 09H
ADDIA   077H    ;执行结果：ACC = ACC(09H) + i(77H)=80H
```

ANDA **[R]**

操作: 寄存器 R 跟 ACC 进行逻辑与运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0FH     ;给 ACC 赋值 0FH
LD      R01,A   ;将 ACC 的值(0FH)赋给寄存器 R01
LDIA    77H     ;给 ACC 赋值 77H
ANDA    R01     ;执行结果：ACC=(0FH  and  77H)=07H
```

ANDR **[R]**

操作: 寄存器 R 跟 ACC 进行逻辑与运算，结果放入 R

周期: 1

影响标志位: Z

举例:

```
LDIA    0FH     ;给 ACC 赋值 0FH
LD      R01,A   ;将 ACC 的值(0FH)赋给寄存器 R01
LDIA    77H     ;给 ACC 赋值 77H
ANDR    R01     ;执行结果：R01=(0FH  and  77H)=07H
```

ANDIA **i**

操作: 将立即数 i 与 ACC 进行逻辑与运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0FH     ;给 ACC 赋值 0FH
ANDIA   77H     ;执行结果：ACC =(0FH  and  77H)=07H
```

CALL **add**

操作: 调用子程序

周期: 2

影响标志位: 无

举例:

```
CALL    LOOP    ;调用名称定义为”LOOP”的子程序地址
```

CLRA

操作: ACC 清零

周期: 1

影响标志位: Z

举例:

```
CLRA                    ;执行结果：ACC=0
```

CLR [R]

操作: 寄存器 R 清零

周期: 1

影响标志位: Z

举例:

```
CLR     R01             ;执行结果：R01=0
```

CLRB [R],b

操作: 寄存器 R 的第 b 位清零

周期: 1

影响标志位: 无

举例:

```
CLRB    R01,3           ;执行结果：R01 的第 3 位为零
```

CLRWDT

操作: 清零看门狗计数器

周期: 1

影响标志位: TO，PD

举例:

```
CLRWDT                  ;看门狗计数器清零
```

COMA [R]

操作: 寄存器 R 取反，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH             ;ACC 赋值 0AH
LD      R01,A           ;将 ACC 的值(0AH)赋给寄存器 R01
COMA    R01             ;执行结果：ACC=0F5H
```

COMR **[R]**

操作: 寄存器 R 取反，结果放入 R

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
COMR    R01     ;执行结果：R01=0F5H
```

DECA **[R]**

操作: 寄存器 R 自减 1，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
DECA    R01     ;执行结果：ACC=(0AH-1)=09H
```

DECR **[R]**

操作: 寄存器 R 自减 1，结果放入 R

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
DECR    R01     ;执行结果：R01=(0AH-1)=09H
```

HSUBA **[R]**

操作: ACC 减 R，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 077H
LD      R01,A   ;将 ACC 的值(077H)赋给寄存器 R01
LDIA    080H    ;ACC 赋值 080H
HSUBA   R01     ;执行结果：ACC=(80H-77H)=09H
```

HSUBR **[R]**

操作: ACC 减 R，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 077H
LD      R01,A   ;将 ACC 的值(077H)赋给寄存器 R01
LDIA    080H    ;ACC 赋值 080H
HSUBR   R01     ;执行结果：R01=(80H-77H)=09H
```

HSUBCA **[R]**

操作: ACC 减 R 减 $\overline{C}$，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 077H
LD      R01,A   ;将 ACC 的值(077H)赋给寄存器 R01
LDIA    080H    ;ACC 赋值 080H
HSUBCA  R01     ;执行结果：ACC=(80H-77H-C)=08H(C=0)
                ;          ACC=(80H-77H-C)=09H(C=1)
```

HSUBCR **[R]**

操作: ACC 减 R 减 $\overline{C}$，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H    ;ACC 赋值 077H
LD      R01,A   ;将 ACC 的值(077H)赋给寄存器 R01
LDIA    080H    ;ACC 赋值 080H
HSUBCR  R01     ;执行结果：R01=(80H-77H-C)=08H(C=0)
                ;          R01=(80H-77H-C)=09H(C=1)
```

INCA **[R]**

操作: 寄存器 R 自加 1，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
INCA    R01     ;执行结果：ACC=(0AH+1)=0BH
```

INCR **[R]**
操作: 寄存器 R 自加 1，结果放入 R
周期: 1
影响标志位: Z
举例:

```
LDIA    0AH     ;ACC 赋值 0AH
LD      R01,A   ;将 ACC 的值(0AH)赋给寄存器 R01
INCR    R01     ;执行结果：R01=(0AH+1)=0BH
```

JP **add**
操作: 跳转到 add 地址
周期: 2
影响标志位: 无
举例:

```
JP      LOOP    ;跳转至名称定义为”LOOP”的子程序地址
```

LD **A,[R]**
操作: 将 R 的值赋给 ACC
周期: 1
影响标志位: Z
举例:

```
LD      A,R01   ;将寄存器 R0 的值赋给 ACC
LD      R02,A   ;将 ACC 的值赋给寄存器 R02，实现了数据从 R01→R02 的移动
```

LD **[R],A**
操作: 将 ACC 的值赋给 R
周期: 1
影响标志位: 无
举例:

```
LDIA    09H     ;给 ACC 赋值 09H
LD      R01,A   ;执行结果：R01=09H
```

LDIA **i**
操作: 立即数 i 赋给 ACC
周期: 1
影响标志位: 无
举例:

```
LDIA    0AH     ;ACC 赋值 0AH
```

NOP

操作: 空指令
周期: 1
影响标志位: 无
举例:

```
NOP
NOP
```

ORIA i

操作: 立即数与 ACC 进行逻辑或操作，结果赋给 ACC
周期: 1
影响标志位: Z
举例:

```
LDIA    0AH     ;ACC 赋值 0AH
ORIA    030H    ;执行结果：ACC =(0AH  or  30H)=3AH
```

ORA [R]

操作: 寄存器 R 跟 ACC 进行逻辑或运算，结果放入 ACC
周期: 1
影响标志位: Z
举例:

```
LDIA    0AH     ;给 ACC 赋值 0AH
LD      R01,A   ;将 ACC(0AH)赋给寄存器 R01
LDIA    30H     ;给 ACC 赋值 30H
ORA     R01     ;执行结果：ACC=(0AH  or  30H)=3AH
```

ORR [R]

操作: 寄存器 R 跟 ACC 进行逻辑或运算，结果放入 R
周期: 1
影响标志位: Z
举例:

```
LDIA    0AH     ;给 ACC 赋值 0AH
LD      R01,A   ;将 ACC(0AH)赋给寄存器 R01
LDIA    30H     ;给 ACC 赋值 30H
ORR     R01     ;执行结果：R01=(0AH  or  30H)=3AH
```

RET

操作: 从子程序返回

周期: 2

影响标志位: 无

举例:

```
        CALL    LOOP        ;调用子程序 LOOP
        NOP                 ;RET 指令返回后将执行这条语句
        …                   ;其它程序
LOOP:
        …                   ;子程序
        RET                 ;子程序返回
```

RET i

操作: 从子程序带参数返回，参数放入 ACC

周期: 2

影响标志位: 无

举例:

```
        CALL    LOOP        ;调用子程序 LOOP
        NOP                 ;RET 指令返回后将执行这条语句
        …                   ;其它程序
LOOP:
        …                   ;子程序
        RET     35H         ;子程序返回,ACC=35H
```

RETI

操作: 中断返回

周期: 2

影响标志位: 无

举例:

```
INT_START                   ;中断程序入口
        …                   ;中断处理程序
        RETI                ;中断返回
```

RLCA [R]

操作: 寄存器 R 带 C 循环左移一位，结果放入 ACC

周期: 1

影响标志位: C

举例:

```
        LDIA    03H         ;ACC 赋值 03H
        LD      R01,A       ;ACC 值赋给 R01,R01=03H
        RLCA    R01         ;操作结果：ACC=06H(C=0);
                                    ACC=07H(C=1)
                                    C=0
```

RLCR **[R]**

操作: 寄存器 R 带 C 循环左移一位，结果放入 R

周期: 1

影响标志位: C

举例:

```
LDIA    03H         ;ACC 赋值 03H
LD      R01,A       ;ACC 值赋给 R01,R01=03H
RLCR    R01         ;操作结果：R01=06H(C=0);
                            R01=07H(C=1);
                            C=0
```

RLA **[R]**

操作: 寄存器 R 不带 C 循环左移一位，结果放入 ACC

周期: 1

影响标志位: 无

举例:

```
LDIA    03H         ;ACC 赋值 03H
LD      R01,A       ;ACC 值赋给 R01,R01=03H
RLA     R01         ;操作结果：ACC=06H
```

RLR **[R]**

操作: 寄存器 R 不带 C 循环左移一位，结果放入 R

周期: 1

影响标志位: 无

举例:

```
LDIA    03H         ;ACC 赋值 03H
LD      R01,A       ;ACC 值赋给 R01,R01=03H
RLR     R01         ;操作结果：R01=06H
```

RRCA **[R]**

操作: 寄存器 R 带 C 循环右移一位，结果放入 ACC

周期: 1

影响标志位: C

举例:

```
LDIA    03H         ;ACC 赋值 03H
LD      R01,A       ;ACC 值赋给 R01,R01=03H
RRCA    R01         ;操作结果：ACC=01H(C=0);
                            ACC=081H(C=1);
                            C=1
```

RRCR **[R]**

操作: 寄存器 R 带 C 循环右移一位，结果放入 R

周期: 1

影响标志位: C

举例:

```
LDIA    03H         ;ACC 赋值 03H
LD      R01,A       ;ACC 值赋给 R01,R01=03H
RRCR    R01         ;操作结果：R01=01H(C=0);
                    R01=81H(C=1);
                    C=1
```

RRA **[R]**

操作: 寄存器 R 不带 C 循环右移一位，结果放入 ACC

周期: 1

影响标志位: 无

举例:

```
LDIA    03H         ;ACC 赋值 03H
LD      R01,A       ;ACC 值赋给 R01,R01=03H
RRA     R01         ;操作结果：ACC=81H
```

RRR **[R]**

操作: 寄存器 R 不带 C 循环右移一位，结果放入 R

周期: 1

影响标志位: 无

举例:

```
LDIA    03H         ;ACC 赋值 03H
LD      R01,A       ;ACC 值赋给 R01,R01=03H
RRR     R01         ;操作结果：R01=81H
```

SET **[R]**

操作: 寄存器 R 所有位置 1

周期: 1

影响标志位: 无

举例:

```
SET     R01         ;操作结果：R01=0FFH
```

SETB **[R],b**

操作: 寄存器 R 的第 b 位置 1

周期: 1

影响标志位: 无

举例:

```
CLR     R01         ;R01=0
SETB    R01,3       ;操作结果：R01=08H
```

STOP

操作: 进入休眠状态

周期: 1

影响标志位: TO，PD

举例:

```
STOP                    ;芯片进入省电模式，CPU、振荡器停止工作，IO 口保持原来状态
```

SUBIA i

操作: 立即数 i 减 ACC，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    077H            ;ACC 赋值 77H
SUBIA   80H             ;操作结果：ACC=80H-77H=09H
```

SUBA [R]

操作: 寄存器 R 减 ACC，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    080H            ;ACC 赋值 80H
LD      R01,A           ;ACC 的值赋给 R01，R01=80H
LDIA    77H             ;ACC 赋值 77H
SUBA    R01             ;操作结果：ACC=80H-77H=09H
```

SUBR [R]

操作: 寄存器 R 减 ACC，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    080H            ;ACC 赋值 80H
LD      R01,A           ;ACC 的值赋给 R01，R01=80H
LDIA    77H             ;ACC 赋值 77H
SUBR    R01             ;操作结果：R01=80H-77H=09H
```

SUBCA **[R]**

操作: 寄存器 R 减 ACC 减 C，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    080H    ;ACC 赋值 80H
LD      R01,A   ;ACC 的值赋给 R01，R01=80H
LDIA    77H     ;ACC 赋值 77H
SUBCA   R01     ;操作结果：ACC=80H-77H-C=09H(C=0);
                          ACC=80H-77H-C=08H(C=1);
```

SUBCR **[R]**

操作: 寄存器 R 减 ACC 减 C，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

```
LDIA    080H    ;ACC 赋值 80H
LD      R01,A   ;ACC 的值赋给 R01，R01=80H
LDIA    77H     ;ACC 赋值 77H
SUBCR   R01     ;操作结果：R01=80H-77H-C=09H(C=0)
                          R01=80H-77H-C=08H(C=1)
```

SWAPA **[R]**

操作: 寄存器 R 高低半字节交换，结果放入 ACC

周期: 1

影响标志位: 无

举例:

```
LDIA    035H    ;ACC 赋值 35H
LD      R01,A   ;ACC 的值赋给 R01，R01=35H
SWAPA   R01     ;操作结果：ACC=53H
```

SWAPR **[R]**

操作: 寄存器 R 高低半字节交换，结果放入 R

周期: 1

影响标志位: 无

举例:

```
LDIA    035H    ;ACC 赋值 35H
LD      R01,A   ;ACC 的值赋给 R01，R01=35H
SWAPR   R01     ;操作结果：R01=53H
```

SZB **[R],b**

操作: 判断寄存器 R 的第 b 位，为 0 间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZB     R01,3       ;判断寄存器 R01 的第 3 位
JP      LOOP        ;R01 的第 3 位为 1 才执行这条语句，跳转至 LOOP
JP      LOOP1       ;R01 的第 3 位为 0 时间跳，执行这条语句，跳转至 LOOP1
```

SNZB **[R],b**

操作: 判断寄存器 R 的第 b 位，为 1 间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SNZB    R01,3       ;判断寄存器 R01 的第 3 位
JP      LOOP        ;R01 的第 3 位为 0 才执行这条语句，跳转至 LOOP
JP      LOOP1       ;R01 的第 3 位为 1 时间跳，执行这条语句，跳转至 LOOP1
```

SZA **[R]**

操作: 将寄存器 R 的值赋给 ACC，若 R 为 0 则间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZA     R01         ;R01→ACC
JP      LOOP        ;R01 不为 0 时执行这条语句，跳转至 LOOP
JP      LOOP1       ;R01 为 0 时间跳，执行这条语句，跳转至 LOOP1
```

SZR **[R]**

操作: 将寄存器 R 的值赋给 R，若 R 为 0 则间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZR     R01         ;R01→R01
JP      LOOP        ;R01 不为 0 时执行这条语句，跳转至 LOOP
JP      LOOP1       ;R01 为 0 时间跳执行这条语句，跳转至 LOOP1
```

SZINCA **[R]**

操作: 将寄存器 R 自加 1，结果放入 ACC，若结果为 0，则跳过下一条语句，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZINCA    R01      ;R01+1→ACC
JP        LOOP     ;ACC 不为 0 时执行这条语句，跳转至 LOOP
JP        LOOP1    ;ACC 为 0 时执行这条语句，跳转至 LOOP1
```

SZINCR **[R]**

操作: 将寄存器 R 自加 1，结果放入 R，若结果为 0，则跳过下一条语句，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZINCR    R01      ;R01+1→R01
JP        LOOP     ; R01 不为 0 时执行这条语句，跳转至 LOOP
JP        LOOP1    ; R01 为 0 时执行这条语句，跳转至 LOOP1
```

SZDECA **[R]**

操作: 将寄存器 R 自减 1，结果放入 ACC，若结果为 0，则跳过下一条语句，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZDECA    R01      ;R01-1→ACC
JP        LOOP     ;ACC 不为 0 时执行这条语句，跳转至 LOOP
JP        LOOP1    ;ACC 为 0 时执行这条语句，跳转至 LOOP1
```

SZDECR **[R]**

操作: 将寄存器 R 自减 1，结果放入 R，若结果为 0，则跳过下一条语句，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

```
SZDECR    R01      ;R01-1→R01
JP        LOOP     ; R01 不为 0 时执行这条语句，跳转至 LOOP
JP        LOOP1    ; R01 为 0 时执行这条语句，跳转至 LOOP1
```

TESTZ **[R]**

操作: 将 R 的值赋给 R,用以影响 Z 标志位

周期: 1

影响标志位: Z

举例:

```
TESTZ   R0          ;将寄存器 R0 的值赋给 R0，用于影响 Z 标志位
SZB     STATUS,Z    ;判断 Z 标志位，为 0 间跳
JP      Add1        ;当寄存器 R0 为 0 的时候跳转至地址 Add1
JP      Add2        ;当寄存器 R0 不为 0 的时候跳转至地址 Add1
```

XORIA **i**

操作: 立即数与 ACC 进行逻辑异或运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH         ;ACC 赋值 0AH
XORIA   0FH         ;执行结果：ACC=05H
```

XORA **[R]**

操作: 寄存器 R 与 ACC 进行逻辑异或运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

```
LDIA    0AH         ;ACC 赋值 0AH
LD      R01,A       ;ACC 值赋给 R01,R01=0AH
LDIA    0FH         ;ACC 赋值 0FH
XORA    R01         ;执行结果：ACC=05H
```

XORR **[R]**

操作: 寄存器 R 与 ACC 进行逻辑异或运算，结果放入 R

周期: 1

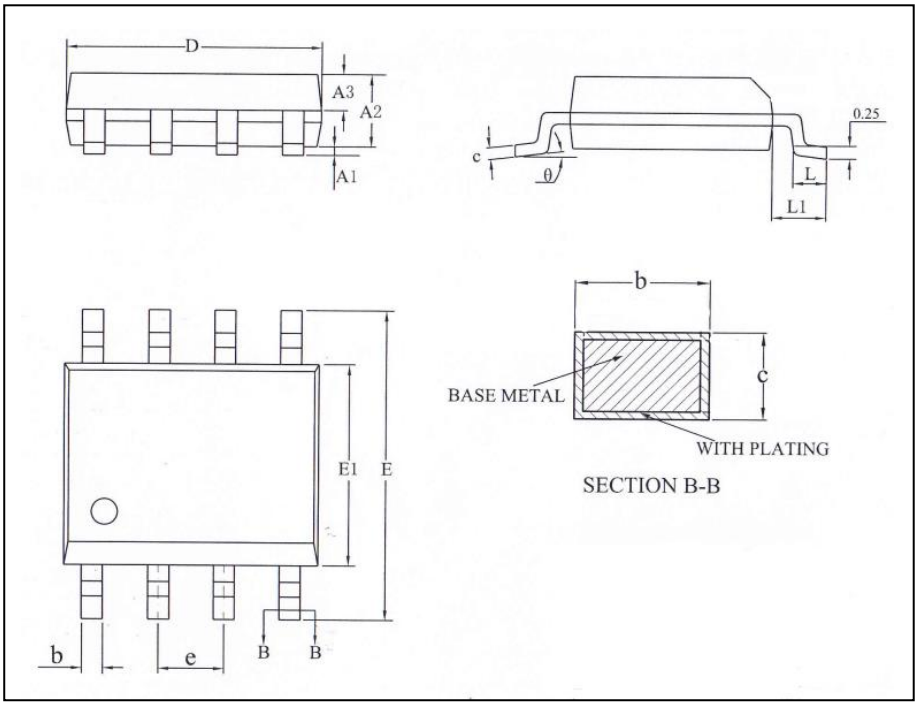
影响标志位: Z

举例:

```
LDIA    0AH         ;ACC 赋值 0AH
LD      R01,A       ;ACC 值赋给 R01,R01=0AH
LDIA    0FH         ;ACC 赋值 0FH
XORR    R01         ;执行结果：R01=05H
```

14. 封装

14.1 SOP8



Symbol	Millimeter		
	Min	Nom	Max
A1	0.05	-	0.25
A2	1.30	1.40	1.60
A3	0.55	-	0.70
b	0.33	-	0.51
c	0.17	-	0.26
D	4.70	-	5.10
E	5.80	6.00	6.20
E1	3.70	-	4.10
e	1.27BSC		
L	0.40	-	0.80
L1	1.05REF		
θ	0	-	8°

注意：封装尺寸不包括模的毛边凸起或门毛刺。

15. 版本修订说明

版本号	时间	修订内容
V0.0.1	2023年1月	初始版本
V0.1.0	2024年11月	1) 格式优化 2) 新增产品一览表
V0.1.1	2024年12月	修改config信息，删除休眠态关闭LVR相关信息，开放系统时钟分频与比较器滤波使能的相关信息