SC8P115xA 用户手册

IO 型 OTP MCU

Rev. 1.3.1

目录

使用注意事项

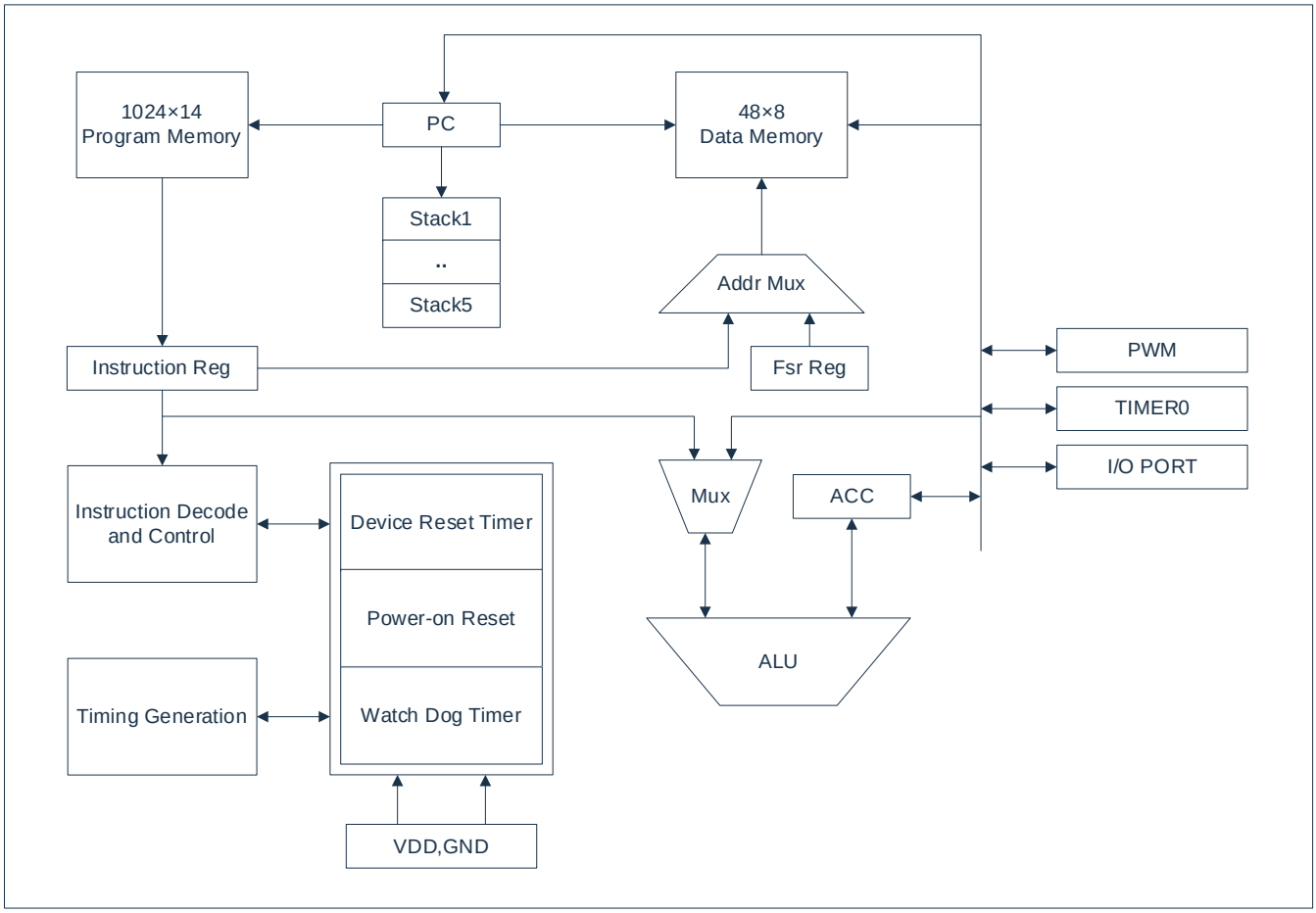
序号	注意事项
1	PB3 上拉使能，PB3 悬空，PB3 的电压为 0.7VDD 左右。
2	PB3 输入电平： T_A=25℃，VDD=5V，上拉使能：V_{IH}=1.7V，V_{IL}=0.9V。 T_A=25℃，VDD=5V，上拉禁止：V_{IH}=2.2V，V_{IL}=1.4V。
3	PB3 上拉电阻： T_A=25℃，VIH=0.5VDD：RPH=36K±7K。
4	IRCF[2:0]=000： 此时芯片内部 RC 振荡器停止，PWM 功能被禁止。

1. 产品概述

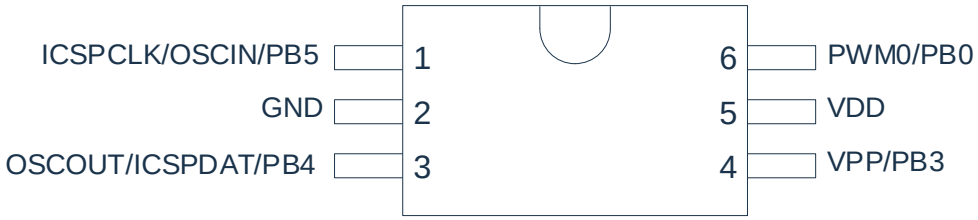
1.1 功能特性及选型表

型号	SC8P1151A	SC8P1152A
脚位	6	8
封装形式	SOT23-6	SOP8
I/O	3+1	5+1
ROM（OTP）	1K×14Bit	1K×14Bit
RAM	48 字节	48 字节
外部振荡	32768 用于定时器	32768 用于定时器
内部振荡频率(MHz)	8	8
高级 PWM	0	0
普通 PWM	1	3
TIMER(8Bit)	1	1
外部中断	2	2
内部中断	1	1
IO 内部上拉电阻	所有 IO	所有 IO
IO 内部下拉电阻	所有 IO（PB3 除外）	所有 IO（PB3 除外）
唤醒	所有中断、看门狗	所有中断、看门狗
LVR	有	有
WDT	有	有
堆栈	5 级	5 级
指令	60 条	60 条
工作电压	2.0V-5.5V	2.0V-5.5V
工作温度范围	-25℃-85℃	-25℃-85℃

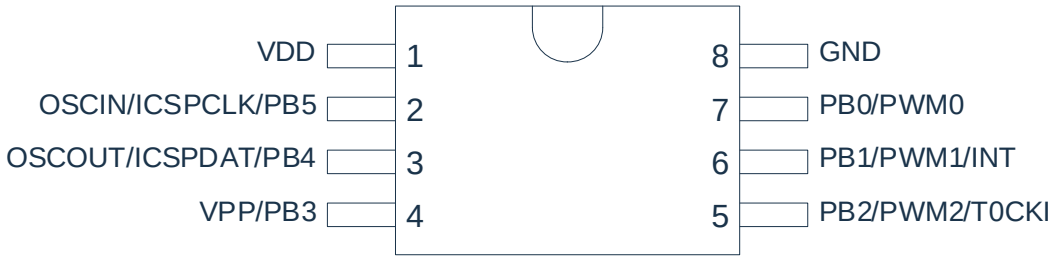
1.2 系统结构框图


1024×14
Program Memory
PC
48×8
Data Memory
Stack1
..
Stack5
Addr Mux
Instruction Reg
Fsr Reg
PWM
TIMER0
I/O PORT
Instruction Decode
and Control
Device Reset Timer
Power-on Reset
Watch Dog Timer
Mux
ACC
ALU
Timing Generation
VDD,GND

1.3 管脚分布



SC8P1151A



SC8P1152A

引脚说明：

管脚名称	IO 类型	管脚说明	共享引脚
PB0-PB2 PB4,PB5	I/O	可编程为：浮空输入口，带上下拉电阻输入口，推挽输出口，开漏输出口；也可作为 XT 振荡口，外部中断输入口，电平变化中断，定时器输入口，PWM 输出口。	OSCIN, OSCOUT, INT, T2CKI, PWM0, PWM1, PWM2
PB3	I/O	可编程为：浮空输入口，带上拉电阻输入口，开漏输出口。	
VDD, GND	P	电源电压输入脚，接地脚	
PWM0-PWM2	O	PWM 输出口	
OSCIN，OSCOUT	I/O	XT 振荡口	
INT	I	外部中断输入口	
T0CKI	I	TMR0 定时/计数器输入	

1.4 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。它只能被芯联发公司烧写器烧写，用户不能访问及操作。它包含了以下内容：

1. WAKEUP_TIME（唤醒后启动时间选择）
   - 18ms　　唤醒后启动时间为 18ms
   - 9ms　　唤醒后启动时间为 9ms
   - 4.5ms　　唤醒后启动时间为 4.5ms
   - 2.25ms　　唤醒后启动时间为 2.25ms
2. WDT（看门狗选择）
   - ENABLE　　打开看门狗定时器
   - DISABLE　　关闭看门狗定时器
3. PROTECT（加密）
   - DISABLE　　OTP 代码不加密
   - ENABLE　　OTP 代码加密
4. LVR_SEL（低压侦测选择）
   - 1.8V　　低压侦测电压选择 1.8V
   - 2.8V　　低压侦测电压选择 2.8V
   - DISABLE　　关闭低压侦测功能
5. SLEEP_LVR（休眠时 LVR 状态）
   - ENABLE　　休眠后打开 LVR
   - DISABLE　　休眠后关闭 LVR

1.5 在线串行编程

可在最终应用电路中对 SC8P115xA 单片机进行串行编程。编程可以简单地通过以下 6 根线完成：

- 电源线（VDD）
- 接地线（GND）
- 数据线（DAT）
- 数据线 2（DAT2）
- 时钟线（CLK）
- 高压线（VPP）

这使用户可使用未编程的器件制造电路板，而仅在产品交付前才对单片机进行编程。从而可以将最新版本的固件或者定制固件烧写到单片机中。

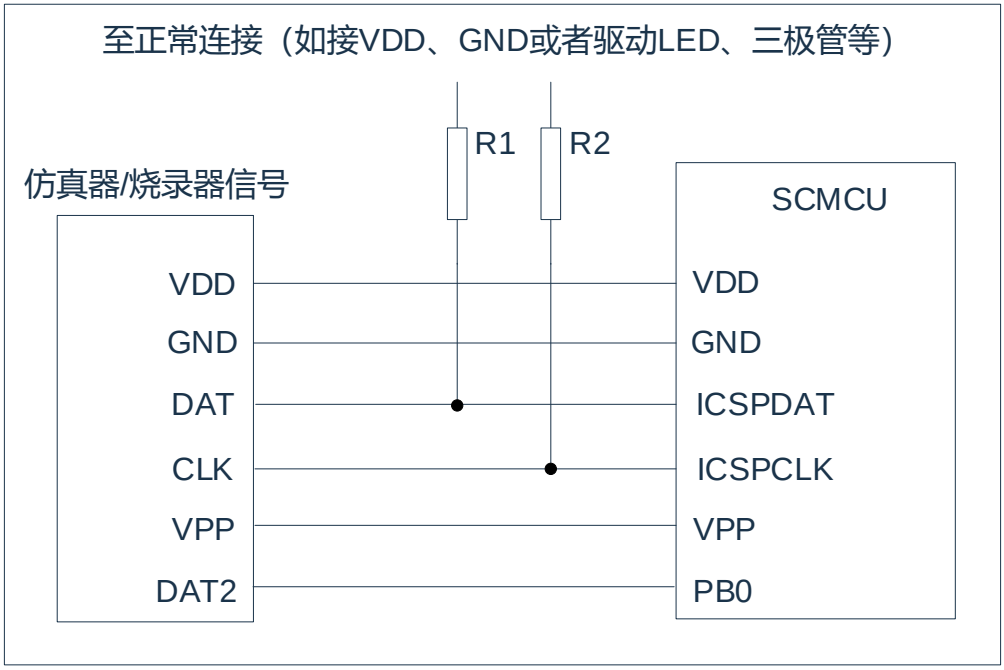


图 1-1：典型的在线串行编程连接方式

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。

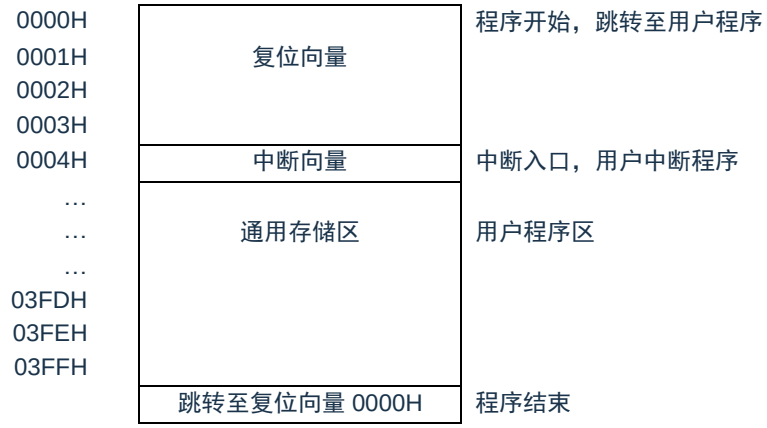
如果烧录的通信线较长，可以在 DAT、CLK 管脚接一个小电容对地，以增加通信的稳定性，其容值不能大于 100pF（101）。

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）

SC8P115xA 系列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 ROM 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       ;中断程序起始
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 SC8P115xA 系列芯片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         ;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      ;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

```
ROM 地址
                LDIA    00H
                LD      PCLATH,A        ;必须对 PCLATH 进行赋值
                …
0010H:          ADDR    PCL             ;ACC+PCL
0011H:          JP      LOOP1           ;ACC=0，跳转至 LOOP1
0012H:          JP      LOOP2           ;ACC=1，跳转至 LOOP2
0013H:          JP      LOOP3           ;ACC=2，跳转至 LOOP3
0014H:          JP      LOOP4           ;ACC=3，跳转至 LOOP4
0015H:          JP      LOOP5           ;ACC=4，跳转至 LOOP5
0016H:          JP      LOOP6           ;ACC=5，跳转至 LOOP6
```

例：错误的多地址跳转程序示例

```
ROM 地址
                LDIA    00H
                LD      PCLATH,A        ;必须对 PCLATH 进行赋值
                …
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，需要要注意该段程序一定不能放在 ROM 空间的分页处。

2.1.2 数据存储器

RAM：74 字节

地址	数据存储器
000H	系统寄存器区
001H	
…	
…	
014H	
015H	系统寄存器区（保留）
…	
019H	
020H	通用寄存器区
021H	
022H	
…	
…	
04EH	
04FH	

数据存储器由 74×8 位组成，分为两个功能区间：特殊功能寄存器（26×8）和通用数据存储器（48×8）。数据存储器单元大多数是可读/写的，但有些只读的。特殊功能寄存器包地址从 00H 到 19H，通用数据寄存器地址从 20H 到 4FH。

2.1.2.1 通用数据存储器

RAM 的 020H～04FH 地址，属于用户可自由定义的通用寄存器区，在此区域的寄存器上电为随机值。当系统上电工作后，若发生意外复位（非上电复位），此区域寄存器保持原来值不变。

2.1.2.2 系统专用数据存储器

地址	名称	说明
00H	INDF	间接寻址寄存器
01H	TMR0	TMR0 数据寄存器
02H	PCL	程序指针 PC 低 8 位
03H	STATUS	系统状态标志寄存器
04H	FSR	间接寻址指针
05H	PORTB	PB 口数据寄存器
06H	TRISB	PB 口方向寄存器
07H	OPTION_REG	预分频寄存器
08H	OSCCON	振荡器控制寄存器
09H	INTCON	中断控制寄存器
0AH	PCLATH	程序指针 PC 高 2 位的写缓冲器
0BH	PDCONB	PB 口下拉电阻寄存器
0CH	ODCONB	PB 口开漏输出寄存器
0DH	WPUB	PB 口上拉电阻寄存器
0EH	IOCB	PB 口电平变化中断寄存器
0FH	TMR0PRD	TMR0 周期寄存器
10H	PWMCTR0	PWM 控制寄存器 0
11H	PWMCTR1	PWM 控制寄存器 1
12H	PWMCTR2	PWM 控制寄存器 2
13H	PWMR	PWM 占空比间接寻址寄存器
14H	PWMPRD	PWM 周期寄存器
15H	-	-
16H	-	-
17H	-	-
18H	-	-
19H	-	-

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，它会根据 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，将导致一个空运作。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-4FH)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    4FH
        SUBA    FSR
        SNZB    STATUS,C    ;一直清零至 FSR 地址为 4FH
        JP      LOOP
```

2.3 堆栈

SC8P115xA 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。

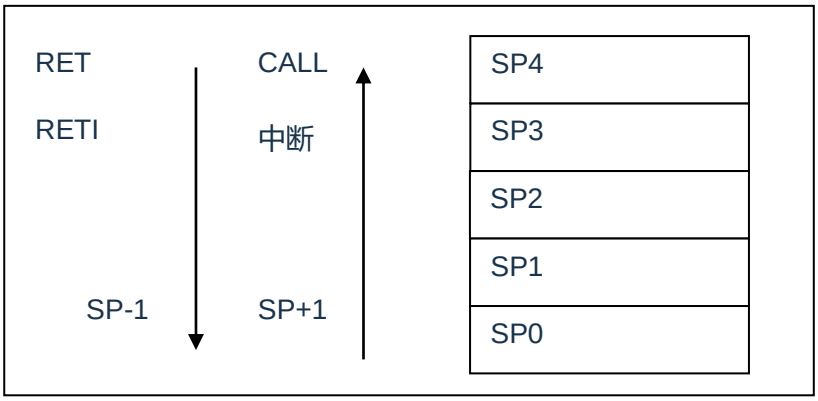


图 2-1：堆栈缓存器工作原理

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）

寄存器 STATUS 中包含 ALU 运算状态信息、系统复位状态信息。其中，位 TO 和 PD 显示系统复位状态信息，包括上电复位、外部复位和看门狗复位等；位 C、DC 和 Z 显示 ALU 的运算信息。

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

Bit7-Bit5　未使用。

Bit4　TO:　超时位；
1=　上电或是执行了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=　发生了结果的第4低位向高位进位；
0=　结果的第4低位没有向高位进位。

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

STATUS 寄存器中除了 TO 和 PD 位，其它的都可以用指令设置或者清零。例如指令：“CLRSTATUS”的结果是 STATUS=“xxx00100”，而不是想象中的全零。也就是说执行指令后，TO 和 PD 的值保持不变，而 Z 标志位因清零而置一，所以若需要改变 STATUS 的值，建议使用 “SETB”、“CLRB”、“SWAPA”、“SWAPR”这几条指令，因为这几条指令不会影响状态标志位。

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 TO/PD 的事件表

TO	PD	复位原因
0	0	WDT溢出唤醒休眠MCU
0	1	WDT溢出非休眠态
1	1	电源上电

复位后TO/PD的状态

2.6 预分频器（OPTION_REG）

预分频器（OPTION_REG）寄存器是 8Bit，可读写的寄存器，它包含各种用于配置 TMR0/WDT 预分频器和 TMR0 的控制位。

07H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OPTION_REG	XT_EN	T0SYNC	T0CS	T0SE	PSA	PS2	PS1	PS0
读写	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 XT_EN: 晶振使能位。
0: 不使能晶振。
1: 使能晶振。

Bit6 T0SYNC: 外部时钟和内部时钟异步使能位。
0: 外部时钟和内部时钟同步。
1: 外部时钟和内部时钟异步。

Bit5 T0CS: TMR0 时钟源选择位。
0: 内部时钟（$F_{OSC}/4$）。
1: 外部时钟（T0CKI 口输入波形或者外部晶体振荡器，取决于 Bit7 XT_EN）。

Bit4 T0SE: T0CKI 信号触发边沿选择位。
0: 上升沿触发。
1: 下降沿触发。

Bit3 PSA: 预分频器分配位。
0: 分给 TMR0 用。
1: 分给 WDT 用。

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

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；用于定时器/计数器时，作为一个预分频器，通常统称作预分频器。在片内只有一个物理的分频器，只能用于 WDT 或 TMR0，两者不能同时使用。也就是说，若用于 TMR0，WDT 就不能使用预分频器，反之亦然。

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

当用于 TMR0 时，有关写入 TMR0 的所有指令（如：CLR TMR0，SETB TMR0,1 等）都会对预分频器清零。

由 TMR0 还是 WDT 使用预分频器，完全由软件控制。他可以动态改变。为了避免出现不该有的芯片复位，当从 TMR0 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'     ;必要步骤，必须执行
LD      OPTION_REG,A    ;必要步骤，必须执行
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 TMR0 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
```

注：要使 TMR0 获取 1:1 的预分频比配置，可通过将选项寄存器的 PSA 位置 1 将预分频器分配给 WDT。

2.7 程序计数器（PC）

程序计数器（PC）控制程序内存中的指令执行顺序，它可以寻址整个 ROM 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。

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）；
RET i、RET、RETI 时	PC=堆栈出来的值；
操作 PCL 时	PC[9:8]=PCLATH，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（Watch Dog Timer）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。在 SC8P115xA 系列芯片中集成了 CONFIG 选项，可通过设置来使 WDT 不起作用，详见 1.5 章 CONFIG 选项说明。使用时需要注意，在禁止 WDT 工作时，必须将 CONFIG 选项和 WDTEN 同时关闭。

2.8.1 WDT 周期

WDT 有一个基本的溢出周期 18ms（无预分频器），假如你需要更长时间的 WDT 周期，可以把预分频器分配给 WDT，最大分频比为 1:128，此时 WDT 的周期约为 2.3s。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（当预分频器分配给 WDT 时）。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。否则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。

3. 系统时钟

3.1 概述

时钟信号由内部振荡产生，在片内产生 4 个非重迭正交时钟信号，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。

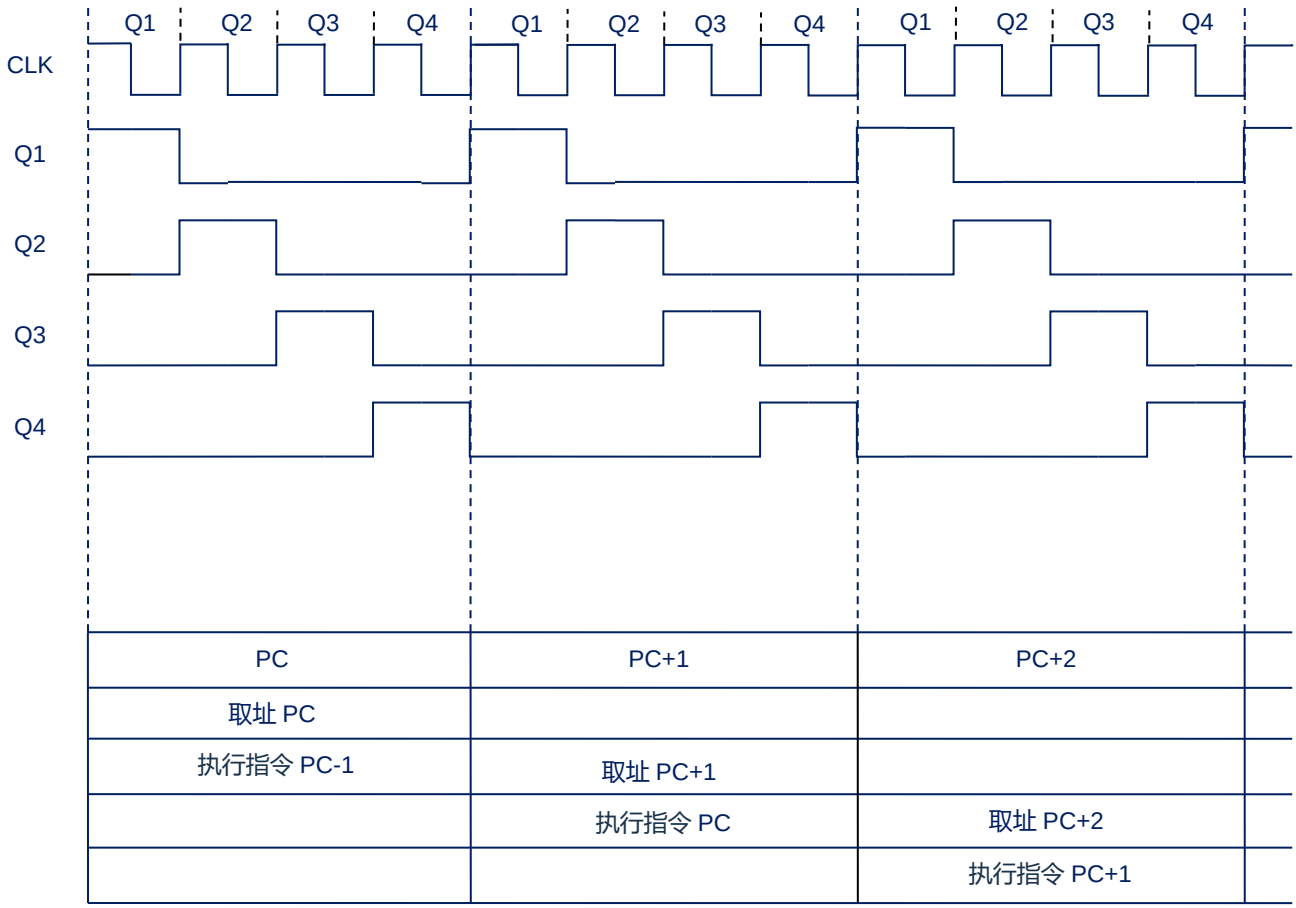


图 3-1：时钟与指令周期时序图

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	8us	4us
2MHz	4us	2us
4MHz	2us	1us
8MHz	1us	500ns

3.2 系统振荡器

SC8P115xA 只有 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，其振荡频率为 8M。

3.3 振荡器控制寄存器

振荡器控制（OSCCON）寄存器控制系统时钟、频率选择、WDT 使能和 TMR0 使能。

08H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OSCCON	SWDTEN	IRCF2	IRCF1	IRCF0	---	---	---	TMR0EN
读写	R/W	R/W	R/W	R/W	---	---	---	R/W
复位值	1	1	1	0	---	---	---	0

Bit7 SWDTEN: WDT 使能位。
0: 不使能 WDT。
1: 使能 WDT。

Bit6~Bit4 IRCF2~0: 主频选择配置位。

IRCF2	IRCF1	IRCF0	内核时钟分频比
0	0	0	F_{WDT}(8K)
0	0	1	F_{OSC}/64
0	1	0	F_{OSC}/32
0	1	1	F_{OSC}/16
1	0	0	F_{OSC}/8
1	0	1	F_{OSC}/4
1	1	0	F_{OSC}/2
1	1	1	F_{OSC}

Bit3~Bit1 未使用。

Bit0 TMR0EN: TMR0 使能位。
0: 不使能 TMR0。
1: 使能 TMR0。

3.4 起振时间

起振时间（OSC TIME）是指从芯片复位到芯片振荡稳定这段时间，固定为 18ms。无论芯片是电源上电复位还是其他原因引起的复位，都会存在这个起振时间。

4. 复位

SC8P115xA 可用如下 3 种复位方式：

- 上电复位。
- 低电压复位（LVR 使能）。
- 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 PD 和 TO 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

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）。当使用外部复位时，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，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。

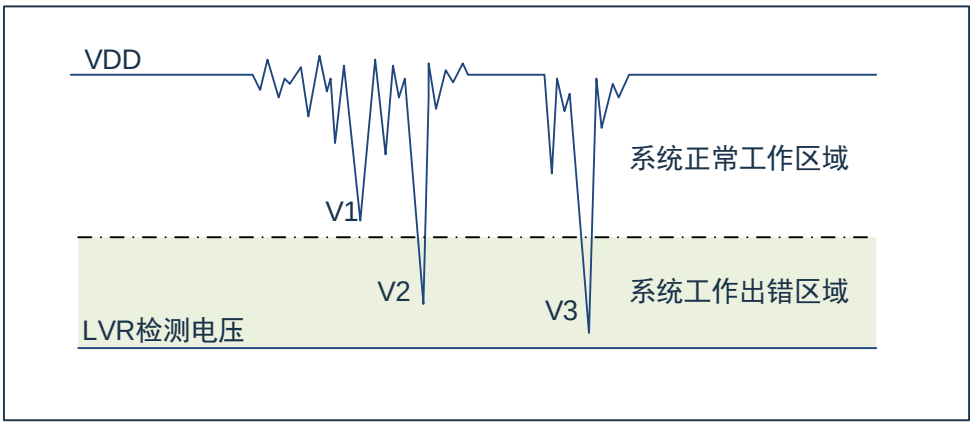


图 4-1：掉电复位示意图

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、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。

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。

4.2.1 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：

- 开启 MCU 的低压侦测功能；
- 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。

开启 MCU 的低压侦测功能

SC8P115xA 系列芯片，内部集成了低压侦测（LVR）功能，可由烧写 CONFIG 控制，详见 1.5 章关于烧写 CONFIG 选择说明。开启 LVR 功能时，当系统电压跌至低于 LVR 电压时，LVR 被触发，系统复位。由于 LVR 电压始终高于芯片的最低工作电压，因此不会存在系统工作死区。

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。

4.2.2 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第 2.8WDT 应用章节。

5. 系统工作模式

SC8P115xA 系列 MCU 存在两种工作模式，一种是正常工作模式，一种是休眠工作模式。在正常工作模式下，各个功能模块均处于工作状态，在休眠状态下，系统时钟停止，芯片保持原来的状态不变，此时 WDT 的功能若没有被烧写 CONFIG 选项禁止，则 WDT 定时器一直工作。

5.1 休眠模式

省电模式是被 STOP 指令启动的，在省电模式状态下，系统振荡停止，以减小功耗，且所有外围停止工作。省电模式可由复位、WDT 溢出或者中断而唤醒。当省电模式被唤醒时，时钟电路仍需要振荡稳定时间。当省电模式由复位唤醒时，系统从 0000H 地址开始执行程序；当省电模式由中断或者 WDT 溢出被唤醒时，PC 从 STOP 指令的下一个地址开始执行程序。

5.1.1 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；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：
        LDIA    B’00000000’
        LD      TRISB,A         ;PB 口设置为输出口
        …                       ;各个输出口设置为不带负载状态
        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
```

5.1.2 休眠模式的唤醒

当系统处于休眠状态时，有以下四种条件可以让 CPU 退出休眠状态：

- 看门狗溢出；
- 外部中断；
- TMR0 定时中断（必须打开晶振使能）；
- 系统掉电后，重新上电。

处于休眠态的 MCU，发生中断或看门狗溢出时，芯片从 STOP 指令的下一个地址开始运行程序。发生其它情况时，芯片都会从复位地址（0000H）开始运行程序，用户可根据 STATUS 的 TO 与 PD 标志位及 SP_FLAG（用户要自己定义），判断何种复位。

例：休眠唤醒用户处理程序

```
                ORG         0000H
                JP          START           ;转到复位处理程序
                ORG         0004H
                JP          INT_START       ;转到中断处理程序
                ORG         0010H
START:                                      ;复位处理程序
                SZB         STATUS,PD
                JP          START_2         ;不是从休眠态复位的处理程序
                SZB         STATUS,TO
                JP          START_3         ;非 WDT 唤醒 MCU 处理程序
                ...
                ...
                ...
SLEEP_MODE:                                 ;休眠子程序
                ...                         ;设置休眠前的状态
                STOP                        ;芯片进入 SLEEP 态处理程序。
                NOP                         ;按键唤醒，加一个空指令等时钟稳定
                JP          XXXX            ;按键唤醒应该处理的程序
```

5.1.3 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（OSC TIME），这个时间可由可由烧写 CONFIG 控制，详见 1.5 章关于烧写 CONFIG 选择说明。

6. I/O 端口

SC8P115xA 有两组 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	8 脚	6 脚	管脚描述	输入/输出
PORT B	PB0	7	/	施密特触发输入，推挽式输出，开漏式输出，PWM0	I/O
	PB1	6	/	施密特触发输入，推挽式输出，开漏式输出，PWM1，INT	I/O
	PB2	5	3	施密特触发输入，推挽式输出，开漏式输出，PWM2，T0CKI	I/O
	PB3	4	4	施密特触发输入，开漏式输出，VPP	I/O
	PB4	3	1	施密特触发输入，推挽式输出，开漏式输出，OSCIN	I/O
	PB5	2	6	施密特触发输入，推挽式输出，开漏式输出，OSCOUT	I/O

表 6-1：端口配置总概

6.1 I/O 口模式及上、下拉电阻

寄存器 PORTB、TRISB、PDCONB、ODCONB、WPUB、IOCB，用于控制 I/O 口线的工作模式。

6.1.1 PORTB 口

PORTB 口有 8Bit 的输入输出管脚。有 6 个寄存器与之相关，分别为 IO 口数据寄存器(PORTB)、IO 口方向控制寄存器(TRISB)、IO 口上拉控制寄存器(WPUB)、IO 口下拉控制寄存器(PDCONB)、IO 口开漏输出控制寄存器(ODCONB)、IO 口电平变化中断控制寄存器(IOCB)。

PORTB 口数据寄存器 PORTB(05H)

05H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PORTB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
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
定义	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O
						PWM2	PWM1	PWM0
			OSCOUT	OSCIN		T0CKI	INT	

PORTB 口方向寄存器 TRISB(06H)

06H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
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

Bit7~Bit0　PORTB 方向。

0:　输出；

1:　输入。

PORTB 口上拉寄存器 WPUB(0DH)

0DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0
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	1	0	0	0

Bit7~Bit0　PORTB 上拉电阻使能。

0:　不使能上拉电阻；

1:　使能上拉电阻（必须关闭相应位的下拉使能）。

PORTB 口下拉寄存器 PDCONB(0BH)

0BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PDCONB	PDCONB7	PDCONB6	PDCONB5	PDCONB4	---	PDCONB2	PDCONB1	PDCONB0
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

Bit7~Bit0　PORTB 下拉电阻使能。

0:　不使能下拉电阻；

1:　使能下拉电阻（必须关闭相应位的上拉使能）。

PORTB 口开漏输出寄存器 ODCONB(0CH)

0CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ODCONB	ODCONB7	ODCONB6	ODCONB5	ODCONB4	---	ODCONB2	ODCONB1	ODCONB0
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

Bit7~Bit0　PORTB 开漏输出使能。

0:　不使能开漏输出；

1:　使能开漏输出。

PORTB 口电平变化中断寄存器 IOCB(0EH)

0EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
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　PORTB 电平变化中断使能。

0:　不使能电平变化中断；

1:　使能电平变化中断。

6.2 I/O 口的使用

6.2.1 写 I/O 口

SC8P115xA 系列芯片的 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,0     ;PB0 口置零
```

6.2.2 读 I/O 口

例：读 I/O 口程序

```
        LD      A,PORTB     ; PORTB 的值赋给 ACC
        SZB     PORTB,1     ;判断 PB1 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。

6.3 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+0.7V”与“VSS-0.7V”之间。若输入口电压不在此范围内可采用如下图所示方法。

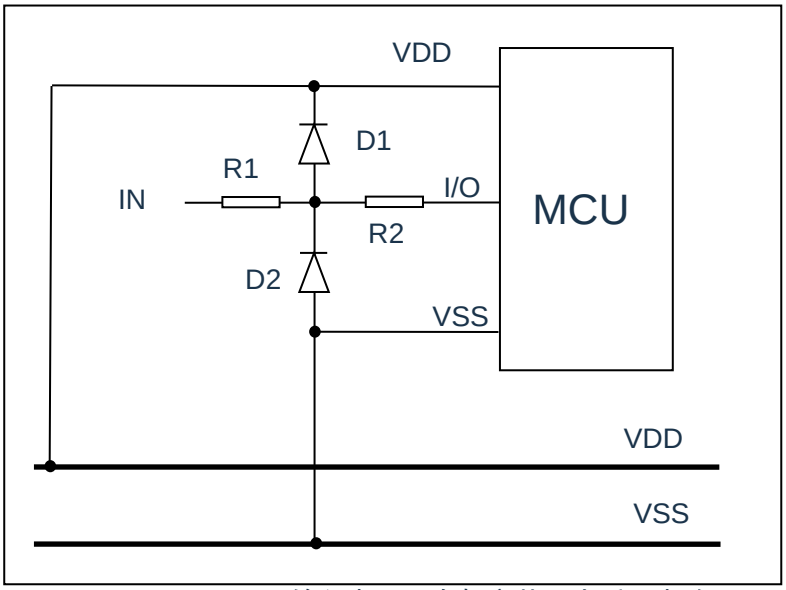


图 6-1：输入电压不在规定范围内采用电路

4. 若 I/O 口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。
5. 若使用到 PB3 口作为信号输入口，建议采用如下图做法，以增强 MCU 抗 EMC 及 ESD 能力。

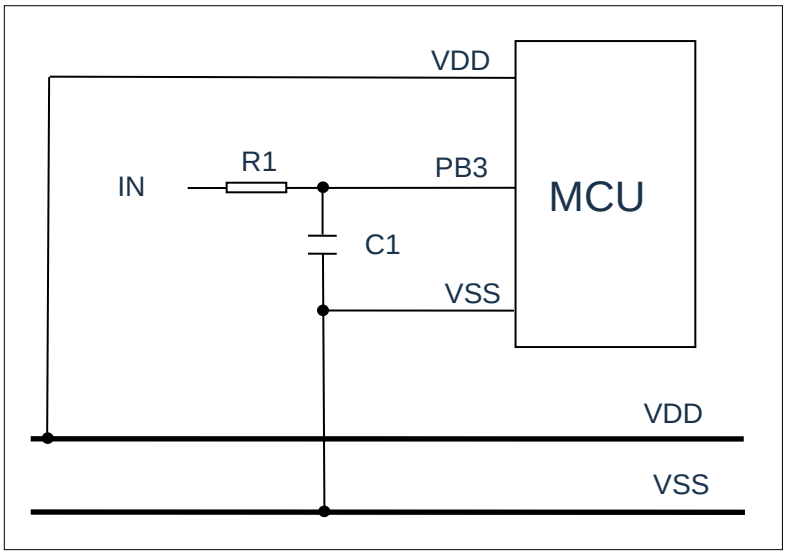


图 6-2：PB3 口作为信号输入口

7. 中断

7.1 中断概述

SC8P115xA 共有 3 个中断源：1 个内部中断（TMR0）和 2 个外部中断（INT、IOCB）。一旦程序进入中断，寄存器 INTCON 的位 GIE 位将被硬件自动清零以避免再次响应其它中断。系统退出中断，即执行完 RETI 指令后，硬件自动将 GIE 置“1”，以响应下一个中断。中断请求存放在寄存器 INTCON 寄存器中。

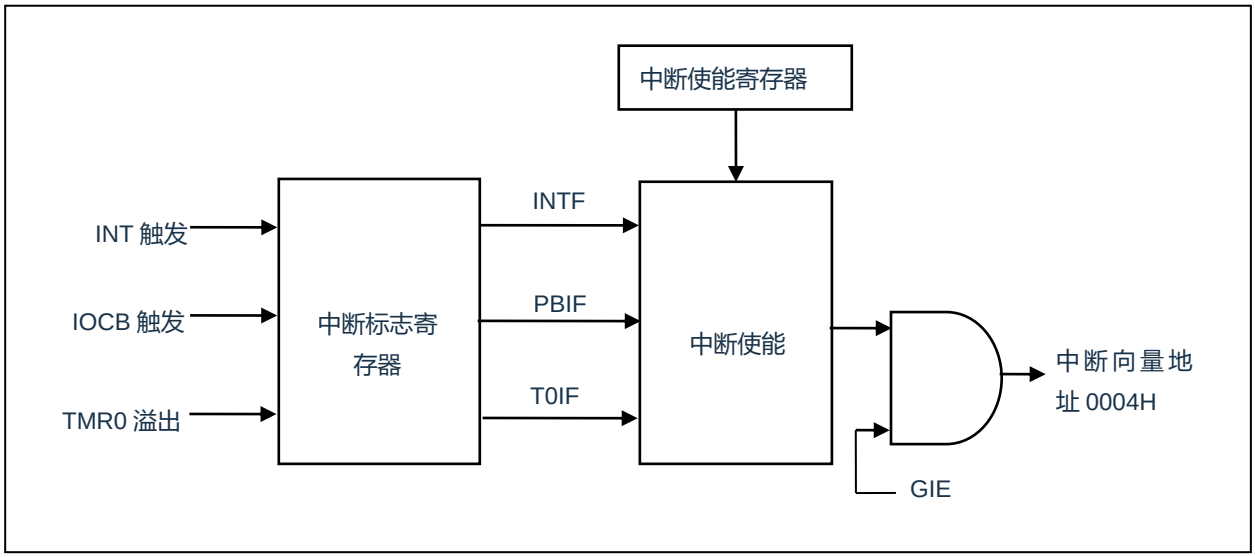


图7-1：中断系统

7.2 中断控制寄存器

中断请求控制寄存器 INTCON 包括所有中断的使能控制位和标志为。

GIE 和相应中断的有效位被置为“1”，则系统进入该中断服务程序，程序计数器入栈，程序转至 0004H 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。

一旦有中断请求发生，相应中断的标志位将被置“1”，该请求被响应后，程序应将该标志位清零，MCU 不会自动清零该中断请求标志位。

中断控制寄存器 INTCON(09H)

09H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INTCON	GIE	INTEG	T0IE	INTE	PBIE	T0IF	INTF	PBIF
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: 全局中断允许位使能位；
0: 禁止所有中断；
1: 允许所有未被屏蔽的中断。
Bit6 INTEG: 外部中断边沿选择；
0: 上升沿触发；
1: 下降沿触发。
Bit5 T0IE: TIMER0溢出中断允许位；
0: 禁止TIMER0中断；
1: 允许TIMER0中断。
Bit4 INTE: INT外部中断允许位；
0: 禁止INT外部中断；
1: 允许INT外部中断。
Bit3 PBIE: PORTB电平变化中断允许位(1)；
0: 禁止PORTB电平变化中断；
1: 允许PORTB电平变化中断。
Bit2 T0IF: TMR0溢出中断标志位(2)；
0: TMR0寄存器未发生溢出；
1: TMR0寄存器已经溢出（必须由软件清零）。
Bit1 INTF: INT外部中断标志位；
0: 未发生INT外部中断；
1: 发生INT外部中断（必须由软件清零）。
Bit0 PBIF: PORTB电平变化中断标志位；
0: 没有一个PORTB通用I/O引脚的状态发生了改变；
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）。

注：
1. IOCB寄存器也必须使能，相应的口线需设置为输入态。
2. T0IF位在TMR0计满归0时置1。复位不会使TMR0发生改变，应在将T0IF位清零前对其进行初始化。

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     0010H
        START：
                ...
                ...
INT_SERVICE：
        PUSH：                          ;中断服务程序入口，保存 ACC 及 STATUS
                LD      ACC             ;保存 ACC 的值，（ACC_BAK 需自定义）
                SWAPA   STATUS
                LD      STATUS          ;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
```

7.3.1 外部中断的应用注意事项

由于外部中断的反应时间很快，当系统外围电压波动时，或系统受到 EMC 干扰时，MCU 可能误进中断，所以需要加上 RC 滤波电路，如图所示。用户可根据外部中断所采样的信号频率选择不同的 R1 和 C1，以提高系统抗 EMC 能力。

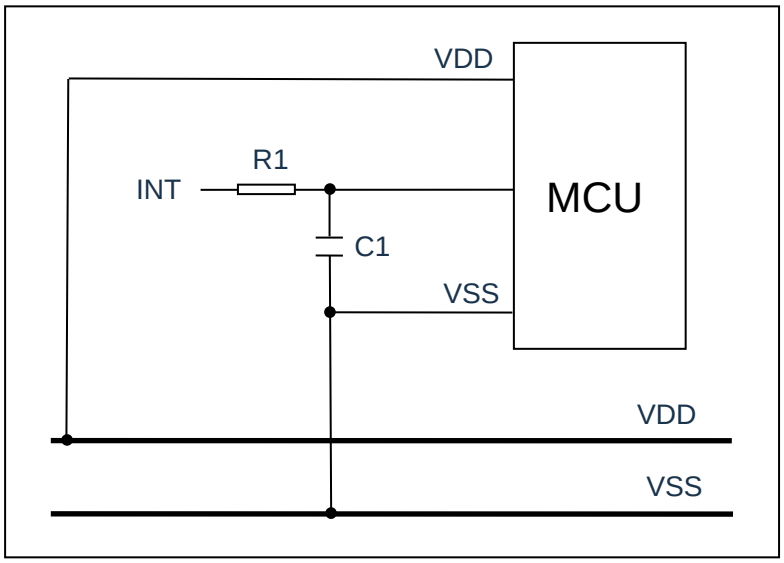


图 7-2：外部中断 RC 滤波电路

7.4 中断的优先级及中断嵌套

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。

注：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 TMR0

8.1 定时计数器 TMR0 概述

TMR0 由如下功能组成：

- 8 位定时器/计数器寄存器（TMR0）；
- 8 位周期寄存器（TMR0PRD）；
- 3 位预分频器（与看门狗定时器共用）；
- 可用程序进行读写操作；
- 可选择定时器或计数器工作方式；
- 可编程内部或外部时钟源；
- 外部时钟源可以选择 32768 晶体振荡器；
- 外部时钟边沿可选择；
- 定时器/计数器溢出中断。

TMR0 的工作模式由 TMR0 控制寄存器（OPTION_REG）的 T0CS 选择：

- 当 T0CS=0 时以定时器方式工作，在不用预分频器情况下每个指令周期加 1，若对 TMR0 进行写操作那么增量操作便禁止两个周期。
- 当 T0CS=1、XT_EN=1 时以外部晶体振荡器定时器方式工作，TMR0 模块在每个振荡周期加 1。
- 当 T0CS=1、XT_EN=0 时以外部脉冲计数器方式工作，TMR0 的计数器将对加到 T0CKI 口的脉冲进行计数。由位 T0SE 选择上升沿有效或下降沿有效，T0SE=0 时选择上升沿有效，T0SE=1 时选择下降沿有效。

8.2 与 TMR0 相关寄存器

有三个寄存器与 TMR0 相关，8 位定时器/计数器（TMR0）、8 位周期寄存器（TMR0PRD）、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，TMR0PRD 为一个 8 位周期比较寄存器，OPTION_REG 寄存器请参看 2.6 关于预分频寄存器（OPTION_REG）的应用。

8 位定时器/计数器 TMR0(01H)

01H	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

8 位周期比较寄存器 TMR0PRD(0FH)

0FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR0PRD								
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

8.3 使用外部时钟作为 TMR0 的时钟源

TMR0 用于外部时钟计数时，外部时钟输入必须满足特定条件。要求外部时钟与内部相位时钟（Tosc）同步，在同步后要经过一定延时，TMR0 才会递增。

如果不使用预分频器，那么外部时钟就是 TMR0 的输入，在内部时钟的 Q2 和 Q4 周期对预分频器输出进行采样可实现 T0CKI 与内部相位时钟同步。因此要求 T0CKI 引脚信号的高电平时间至少为 2 个 Tosc（加上一小段的 RC 延时），并且低电平时间至少为 2 个 Tosc（加上一小段的 RC 延时）。

若使用了预分频器，外部时钟输入要先经过异步脉动计数型预分频器的分频，从而使预分频器的输出对称。为了使外部时钟满足采样要求，必须考虑纹波计数器的影响。因此 T0CKI 的时钟周期至少为 4 个 Tosc（加上一小段的 RC 延时）除以预分频值。

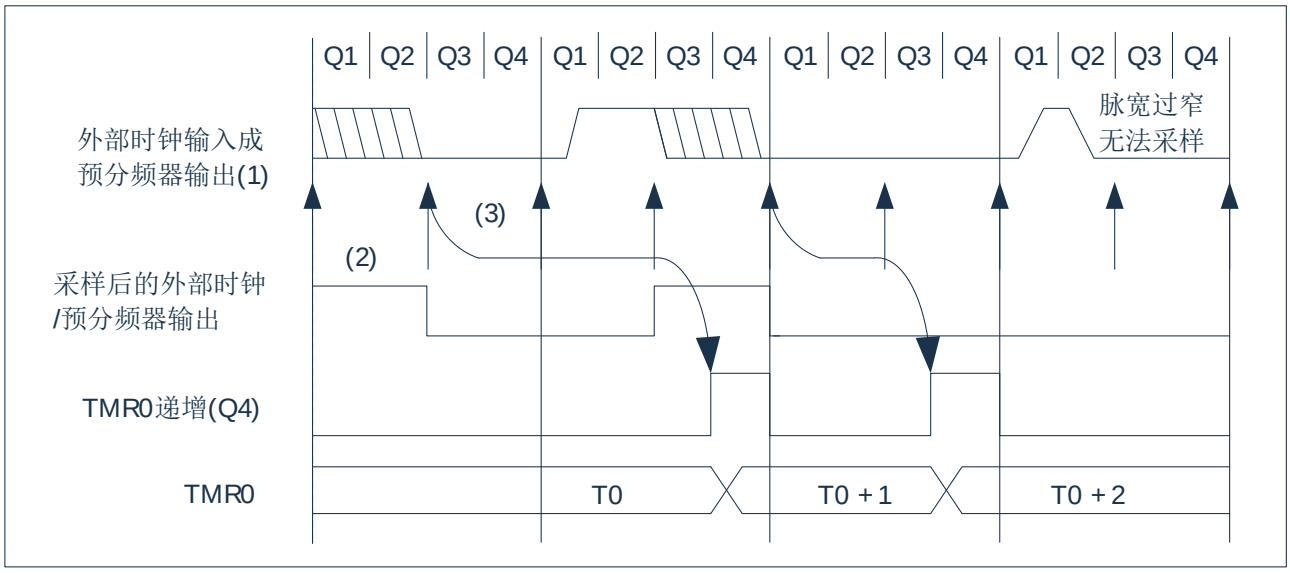


图 8-1：TMR0 与外部时钟时序

注：
1. 不选择预分频器时为外部时钟；否则为预分频器的输出。
2. 箭头所指为采样时刻。
3. 时钟输入发生变化到TMR0递增会有3个Tosc到7个Tosc（持续时间Q=Tosc）的延时，因此测量TMR0输入的相邻脉冲间隔时，其最大误差为±4Tosc。

8.4 TMR0 做定时器的应用

8.4.1 TMR0 的基本时间常数

当 OPTION_REG 的 Bit3 位被置 1 时，预分频器作为 WDT 计时的分频，此时 TMR0 的输入时钟为系统时钟的 2 分频。当 OPTION_REG 的 Bit3 位被置 0 时，预分频器作为 TMR0 计数器的分频。其基本时间常数如下表：

OPTION_REG PS2～PS0	TMR0 的输入时钟 T0CLK	Fcpu=4MHz÷4	
		最大溢出间隔时间	TMR0 递增时间
000	Fcpu/2	512μs	2μs
001	Fcpu/4	1024μs	4μs
010	Fcpu/8	2048μs	8μs
011	Fcpu/16	4096μs	16μs
100	Fcpu/32	8192μs	32μs
101	Fcpu/64	16384μs	64μs
110	Fcpu/128	32768μs	128μs
111	Fcpu/256	65536μs	256μs

8.5 TMR0 操作流程

TMR0 的操作流程为：

- 设置 TMR0 的周期，TMR0PRD 寄存器（TMR0PRD 在 TMR0EN 使能后会锁存，所以必须在 TMR0 使能前写入 TMR0PRD）；
- 设置 TMR 计数器初值（该寄存器默认从 0 自加一，若赋初值后从初值自加一，溢出后计数器的值会自动清零，所以需要再次赋初值）；
- 如果 T0CS=1，可通过 IRCF[2:0]配置系统时钟，来改变 TMR0 的周期；
- 如果 PSA=1，预分频器供 TMR0 使用，PS2：PS0 赋值，选择预分频比；
- 开 TMR0 使能，TMR0EN=1。

例：TMR0 定时设置程序

```
        LDIA    03FH
        LD      TMR0PRD,A           ；设置周期寄存器
        LDIA    000H
        LD      OPTION_REG,A        ；设置 TMR0 时钟=Fcpu/2
        CLR     TMR0                ；初始化 TMR0
        SETB    OSCCON,TMR0EN
```

注：

1. 每次 TMR0 溢出时 TMR0 的初值并不会被自动加载，故用户需在每次 TMR0 溢出时重新加载 TMR0 初值；由于对 TMR0 进行写操作，TMR0 将会有一个时钟不递增，用户要自己输入校正值来避开这个问题。
2. 持续将 TMR0 和 TMR0PRD 的值做比较以确定它们何时匹配。TMR0 将从 00h 开始递增直至与 TMR0PRD 中的值匹配。匹配发生时，会发生以下两个事件：
   1) TMR0 在下一递增周期。
   2) TMR0 后分频器递增。

9. 普通 PWM

9.1 普通 PWM 概述

普通 PWM 由如下功能组成：

- 3 路普通 PWM 共用一个 10 位周期寄存器；
- 3 路普通 PWM 共用一个前分频器；
- 3 路普通 PWM 分别拥有一个 10 位的占空比寄存器；

SC8P115xA 的普通 PWM 使能由 PWMCTR0 中 ENx（x=0-2）位控制。改变 PWM 的占空比寄存器后会在下一个周期时改变占空比。

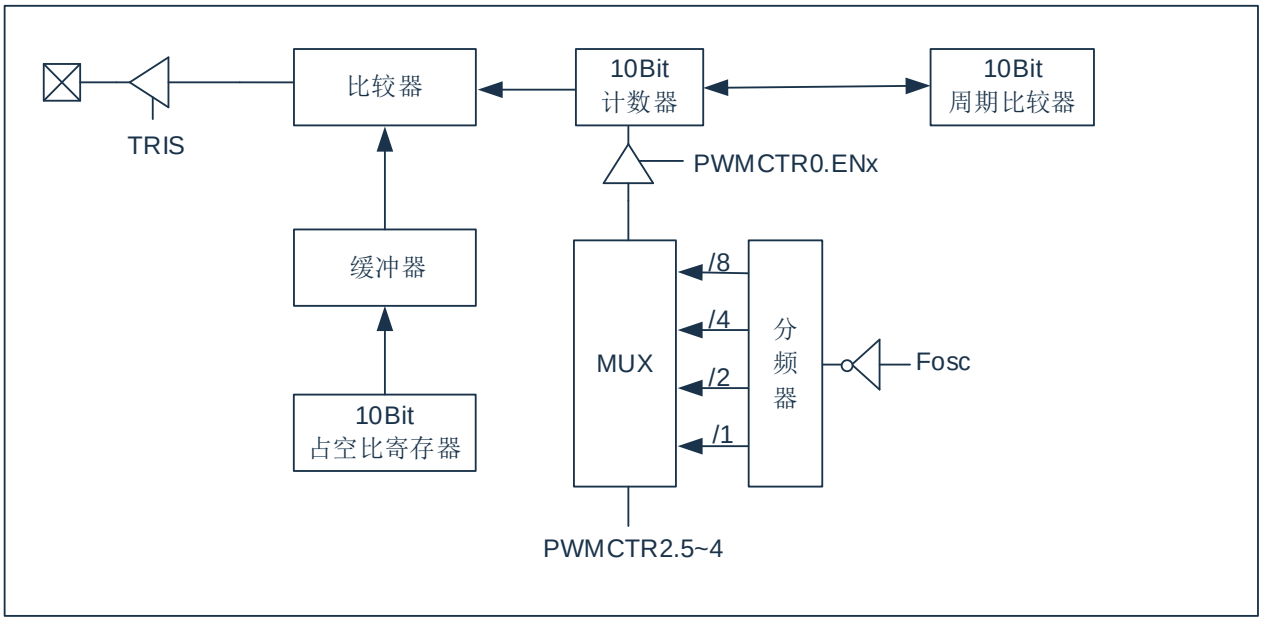


图 9-1：普通 PWM 框图

9.2 与普通 PWM 相关寄存器

有 3 个寄存器与普通 PWM 有关，PWMCTR0-2（PWM 控制寄存器）、PWMPRD（PWM 周期寄存器）和 PWMR（PWM 占空比寄存器）。其中 PWMR 为间接访问寄存器。

PWM 控制寄存器 0PWMCTR0(10H)

10H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCTR0	---	---	---	---	---	PWMEN2	PWMEN1	PWMEN0
R/W	---	---	---	---	---	R/W	R/W	R/W
复位值	---	---	---	---	---	0	0	0

Bit7-Bit5 未使用。
Bit4-Bit1 PWMENx: PWM使能位（x=0-2）；
0: 不使能PWMx功能；
1: 使能PWMx功能。

PWM 控制寄存器 1 PWMCTR1(11H)

11H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCTR1	---	---	PWMR29	PWMR28	PWMR19	PWMR18	PWMR09	PWMR08
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 未使用。
Bit5-Bit0 PWMRx9-8: PWMx占空比寄存器高2位（第9、8位，x=0-2）。

PWM 控制寄存器 2 PWMCTR2(12H)

12H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCTR2	PWMPRD9	PWMPRD8	PWMCK1	PWMCK0	---	PWMS2	PWMS1	PWMS0
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

Bit7-Bit6 PWMPRD9-8: PWM周期寄存器高2位（第9、8位，x=0-2）；
Bit5-Bit4 PWMCK1-0: PWM时钟分频选择位。
00: Fosc
01: Fosc/2
10: Fosc/4
11: Fosc/8
Bit3 未使用。
Bit2-Bit0 PWMS2-0: PWMR寄存器地址选择位。
000: 选择PWM0通道占空比寄存器低8位；
001: 选择PWM1通道占空比寄存器低8位；
010: 选择PWM2通道占空比寄存器低8位；
其它: 未使用。

PWM 占空比寄存器低 8 位 PWMR(13H)

13H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMR								
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

PWMR 占空比寄存器低 8 位（3 路共用，间接访问）

访问 PWMR 占空比寄存器之前，请先设置 PWMCTR2 中 PWMS2-0 位选择要访问的 PWM 通道。

PWM 周期寄存器低 8 位 PWMPRD(14H)

14H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMPRD								
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

9.3 普通 PWM 的周期、占空比

9.3.1 普通 PWM 输出周期

普通 PWM 输出周期由系统主频（Fosc）、PWM 时钟分频比、PWM 周期寄存器决定，计算公式如下：

PWM 调制周期 =（PWMPRD+1）×PWM 分频比÷F_{OSC}

PWMPRD 为 10Bit 寄存器，高两位是 PWMCTR2 寄存器的 Bit7-6，PWMPRD9-8。低 8 位是 PWMPRD 寄存器。

9.3.2 普通 PWM 占空比算法

普通 PWM 输出的占空比与 PWMRx 数值相关，从整体上来说，其占空比近似等于 PWMRx÷（PWMPRD+1）。具体的计算方法为当 PWM 计数器大于 PWMRx 寄存器，PWM 输出为 0；当 PWM 计数器小于等于 PWMRx 寄存器，PWM 输出为 1。PWM 计数器在等于周期寄存器时清零。

9.4 普通 PWM 应用

PWM 的应用设置需要的操作流程如下：

- 设置 PWM 预分频值，PWMCK1 和 PWMCK0；
- 设置 PWM 周期，PWM_PRD9-8，PWM_PRD[7:0]共 10Bit。
- 设置 PWMCTR2 中 PWMS2-0，选择 PWM 通道；
- 设置 PWMR 占空比：PWMRx9-8（x=2-0），PWMR[7:0]共 10Bit，因为 PWMR 是间接访问的寄存器，所以必须先设置要设置的 PWM 通道；
- 开启相应 PWM 使能位：PWMCTR0.2-0，开启任意一个使能后，PWM 周期都将被锁存，所以 PWM 周期必须在任意一路 PWM 使能前设置。

10.电气参数

10.1 DC 特性

（VDD=5V，T_A= 25℃，PB3 口除外，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	-	Fsys=8M	2.0	-	5.5	V
I_{DD}	工作电流	5V	----	-	1.5	-	mA
		3V	----	-	1	-	mA
I_{STB}	静态电流 (LVR、WDT Disable)	5V	----	0.1	1.0	2.0	uA
		3V	----	0.01	0.1	1.0	uA
	静态电流 (LVR、WDT Enable)	5V	----	5.0	6.0	10.0	uA
		3V	----	3.0	4.0	6.0	uA
V_{IL}	低电平输入电压（开上拉）	5V	----	-	1.2	-	V
	低电平输入电压（关上拉）	5V		-	1.4	-	V
V_{IH}	高电平输入电压（开上拉）	5V	----	-	2.0	-	V
	高电平输入电压（关上拉）	5V			2.2		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
R_{PH}	上拉电阻阻值	5V	0.7VDD	-	40	-	K
		3V	0.7VDD	-	60	-	K
R_{PL}	下拉电阻阻值	5V	0.3VDD	-	50	-	K
		3V	0.3VDD	-	85	-	K
I_{OL}	输出口灌电流	5V	V_{OL}=0.3VDD	-	40	-	mA
		3V	V_{OL}=0.3VDD	-	20	-	mA
I_{OH}	输出口拉电流	5V	V_{OH}=0.7VDD	-	10	-	mA
		3V	V_{OH}=0.7VDD	-	6	-	mA

10.2 LVR 电气特性

（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
V_{LVR1}	LVR 设定电压 1	VDD=1.8~5.5V	1.7	1.8	1.9	V
V_{LVR2}	LVR 设定电压 2	VDD=1.98~5.5V	2.7	2.8	2.9	V

10.3 AC 特性

（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				
Fsys	工作频率（RC）	-	2.5V~5.5V	-	8	-	MHz
T_{WDT}	WDT 复位时间	5V	-	-	18	-	ms
		3V	-	-	30	-	ms
F_{RC}	内振频率稳定性	VDD=4.0~5.5V T_A=－20~85℃		-2%	8	+2%	MHz
		VDD=2.5~5.5V		-4%	8	+4%	MHz

11.指令

11.1 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
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”，则跳过下一条指令	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”，则跳过下一条指令	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

11.2 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
```

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

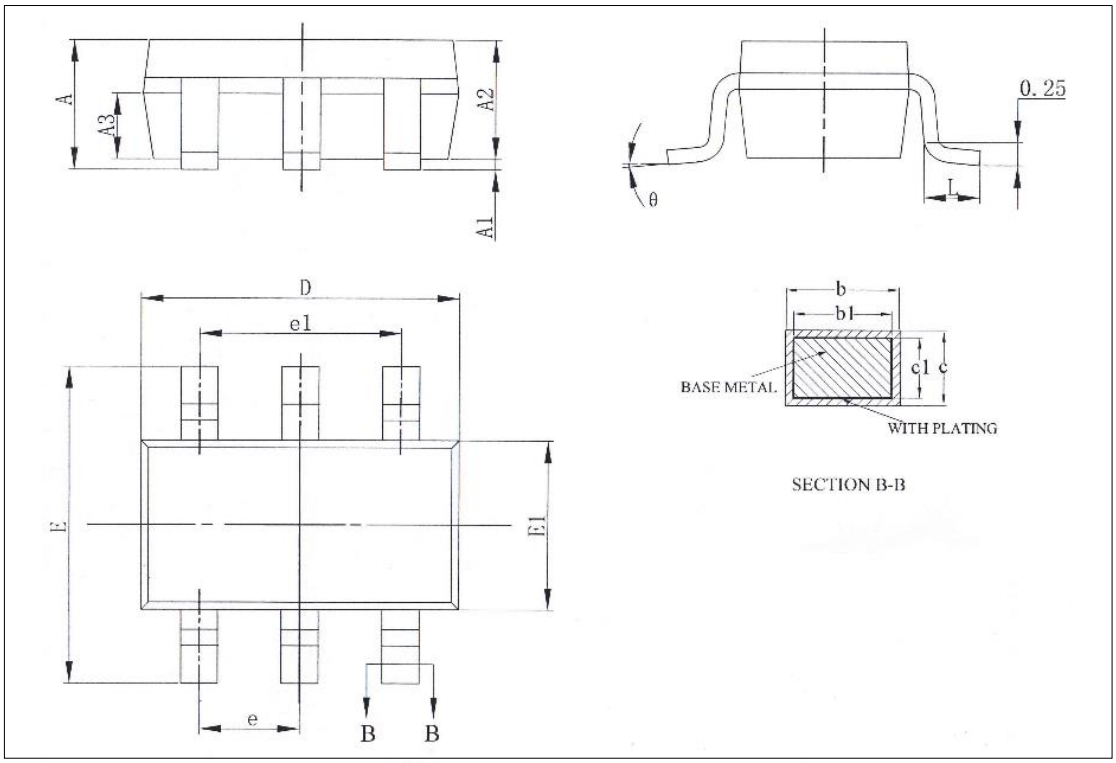
影响标志位: Z

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
```

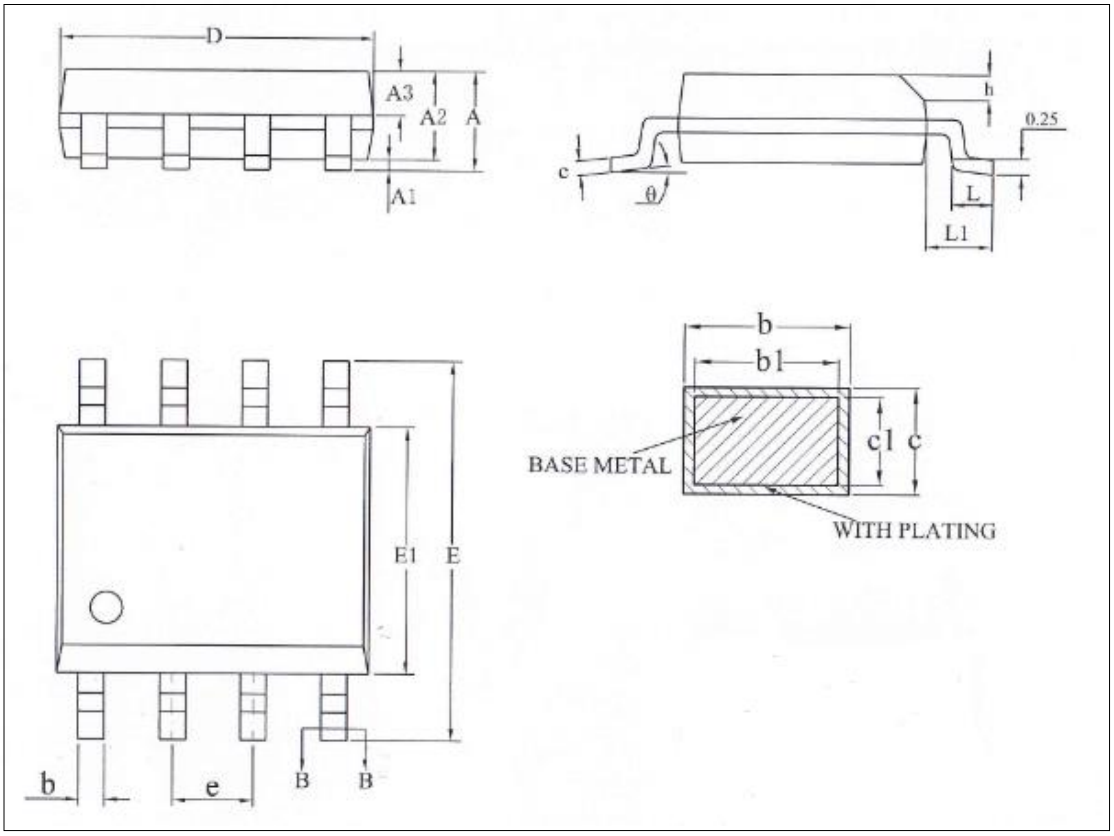
12.封装

12.1 SOT23-6



Symbol	Millimeter		
	Min	Nom	Max
A	-	-	1.25
A1	0.04	-	0.10
A2	1.00	1.10	1.20
A3	0.55	0.65	0.75
b	0.38	-	0.48
b1	0.37	0.40	0.43
c	0.11	-	0.21
c1	0.10	0.13	0.16
D	2.72	2.92	3.12
E	2.60	2.80	3.00
E1	1.40	1.60	1.80
e	0.95BSC		
e1	1.9BSC		
L	0.30	-	0.60
θ	0	-	8°

12.2 SOP8



Symbol	Millimeter		
	Min	Nom	Max
A	-	-	1.75
A1	0.10	-	0.225
A2	1.30	1.40	1.50
A3	0.60	0.65	0.70
b	0.39	-	0.47
b1	0.38	0.41	0.44
c	0.20	-	0.24
c1	0.19	0.20	0.21
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
h	0.25	-	0.50
L	0.5	-	0.80
L1	1.05REF		
θ	0	-	8°

13.版本修订说明

版本号	时间	修改内容
V1.0	2018 年 7 月	初始版本
V1.1	2019 年 7 月	功能版本升级
V1.2	2020 年 5 月	更改为新格式
V1.3	2022 年 1 月	更正部分有误内容
V1.3.1	2023 年 2 月	1) 章节 1.5 图中增加 DAT2 和 PB0 线 2) 更正 3.3 振荡器控制寄存器中 Bit7 名称