CMS89F11x

用户手册

AD型MCU

V1.3

请注意以下有关CMS知识产权政策

＊中微半导体公司已申请了专利，享有绝对的合法权益。与中微半导体公司MCU或其他产品有关的专利权并未被同意授权使用，任何经由不当手段侵害中微半导体公司专利权的公司、组织或个人，中微半导体公司将采取一切可能的法律行动，遏止侵权者不当的侵权行为，并追讨中微半导体公司因侵权行为所受的损失、或侵权者所得的不法利益。

＊中微半导体公司的名称和标识都是中微半导体公司的注册商标。

＊中微半导体公司保留对规格书中产品在可靠性、功能和设计方面的改进作进一步说明的权利。然而中微半导体公司对于规格内容的使用不负责任。文中提到的应用其目的仅仅是用来做说明，中微半导体公司不保证和不表示这些应用没有更深入的修改就能适用，也不推荐它的产品使用在会由于故障或其它原因可能会对人身造成危害的地方。中微半导体公司的产品不授权适用于救生、维生器件或系统中作为关键器件。中微半导体公司拥有不事先通知而修改产品的权利，对于最新的信息，请参考我们的网站http://www.mcu.com.cn

目录

使用注意事项

振荡方式选择外部XT振荡时，P2.2口只能作为输入口；选择内部RC振荡时P2.2口可作为普通IO口。

1. 产品概述

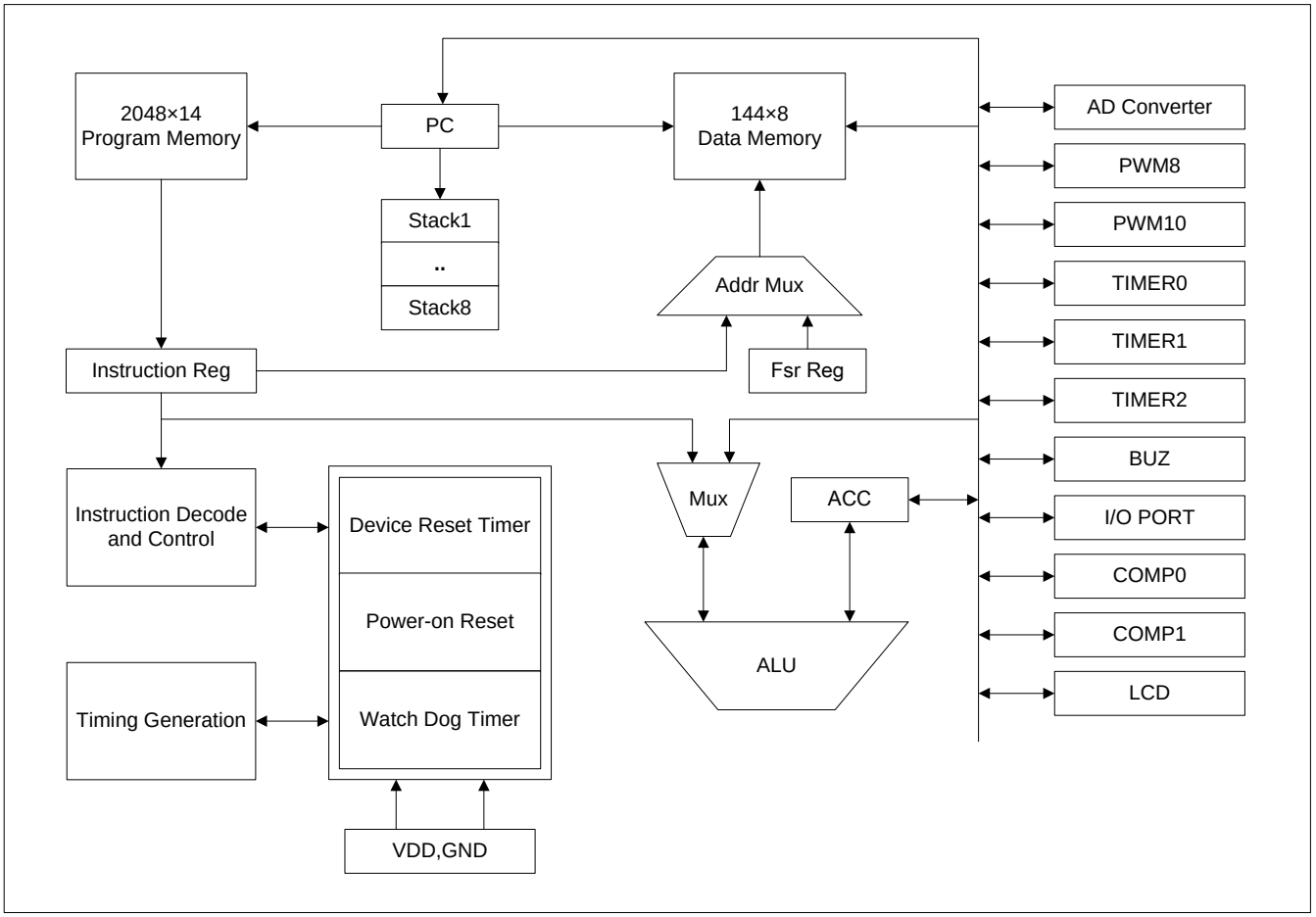
1.1 功能特性

- 内存
  FLASH：2K×14Bit
  RAM：144×8Bit
- 8 级堆栈缓存器
- 简洁实用的指令系统（68 条指令）
- 内置低压侦测电路，内部复位电压：1.8V/2.5V/3.5V
- 7 个中断源
  - 内部中断源 3 个：TMR1、TMR2、ADC
  - 外部中断源 4 个：EXT0、EXT1、EXT2、P0C
- 3 个 8 位定时器
  TMR0、TMR1、TMR2
- 高频信号输出口（CLO）
  占空比可选择：25%、50%、75%
- 14 路 12 位模数转换（ADC）
- 2 种振荡模式
  - 内部 RC 振荡：8MHz/4MHz 可选（4.5V—5.5V）&（-20℃—70℃）误差不超过±1%
  - 外部 XT 振荡：最高 8MHz
- 工作电压范围：2.5V—5.5V@8MHz
  1.8V—5.5V@4MHz
  工作温度范围：-40℃—85℃
- 指令周期（单指令或双指令周期）
- 专用蜂鸣器输出口（频率可变）
- 内置 WDT 定时器
- I/O 口配置
  - P0：具有唤醒功能、上拉电阻选项
  - P1：具有上拉电阻选项
  - P2：具有上、下拉电阻选项
- 两种工作模式
  正常模式、睡眠模式
- 2 路内置比较器
  正端可选择接内部参考电压
- 查表功能
- 2 个 PWM 输出口
  - 8 位 PWM
  - 10 位 PWM
- 内置基准源
- 4COM 1/2Bias LCD

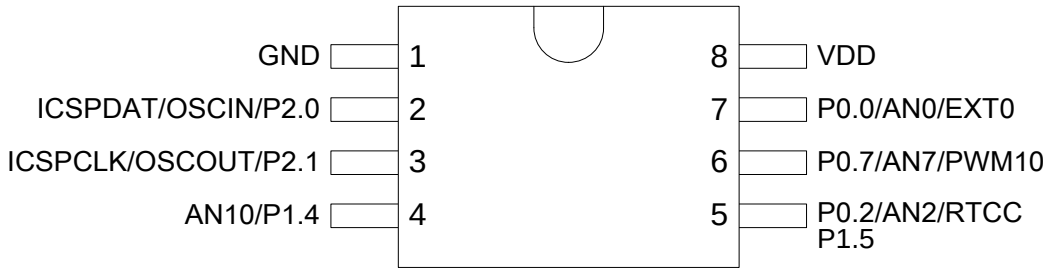
型号说明

PRODUCT	FLASH	RAM	EEPROM	I/O	ADC	LCD	PWM	PACKAGE
CMS89F111	2K×14Bit	144×8Bit	32×14Bit	6	12Bit×4	--	1	SOP8
CMS89F112	2K×14Bit	144×8Bit	32×14Bit	12	12Bit×9	1/2Bias×4	2	SOP14
CMS89F113	2K×14Bit	144×8Bit	32×14Bit	14	12Bit×11	1/2Bias×4	2	SOP16
CMS89F116	2K×14Bit	144×8Bit	32×14Bit	18	12Bit×14	1/2Bias×4	2	DIP20/SOP20

1.2 系统结构框图

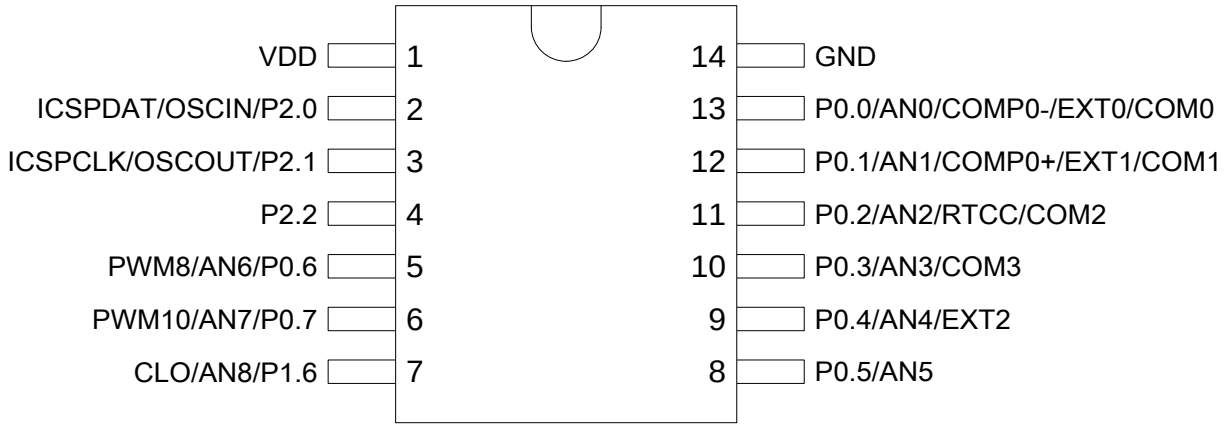

2048×14
Program Memory
PC
144×8
Data Memory
AD Converter
PWM8
PWM10
TIMER0
TIMER1
TIMER2
BUZ
I/O PORT
COMP0
COMP1
LCD
Stack1
..
Stack8
Addr Mux
Fsr Reg
Instruction Reg
Instruction Decode
and Control
Device Reset Timer
Power-on Reset
Watch Dog Timer
Timing Generation
VDD,GND
Mux
ACC
ALU

1.3 管脚分布

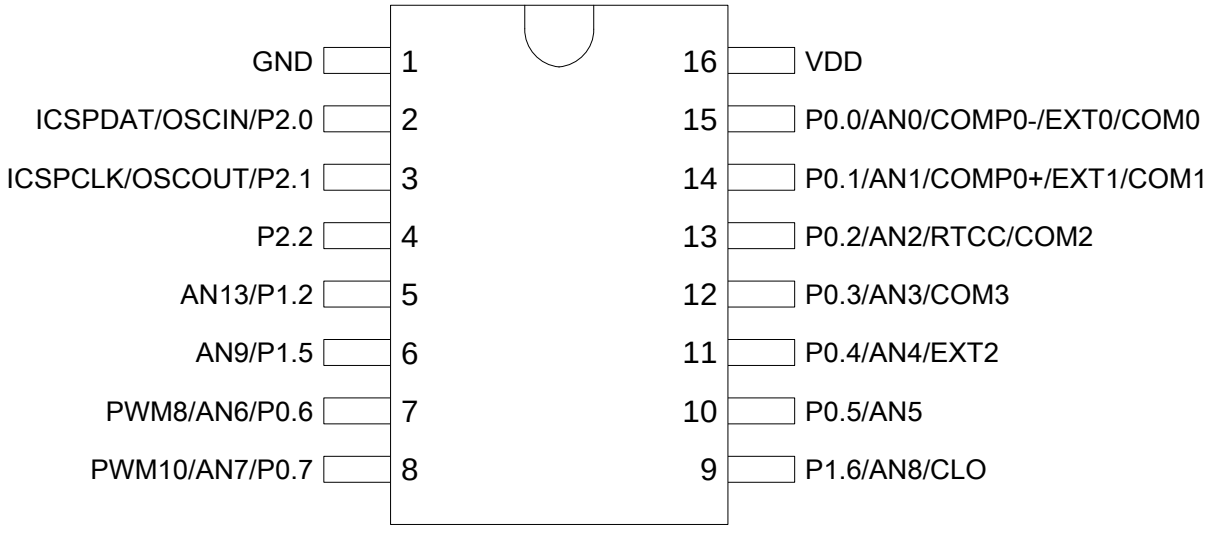


CMS89F111

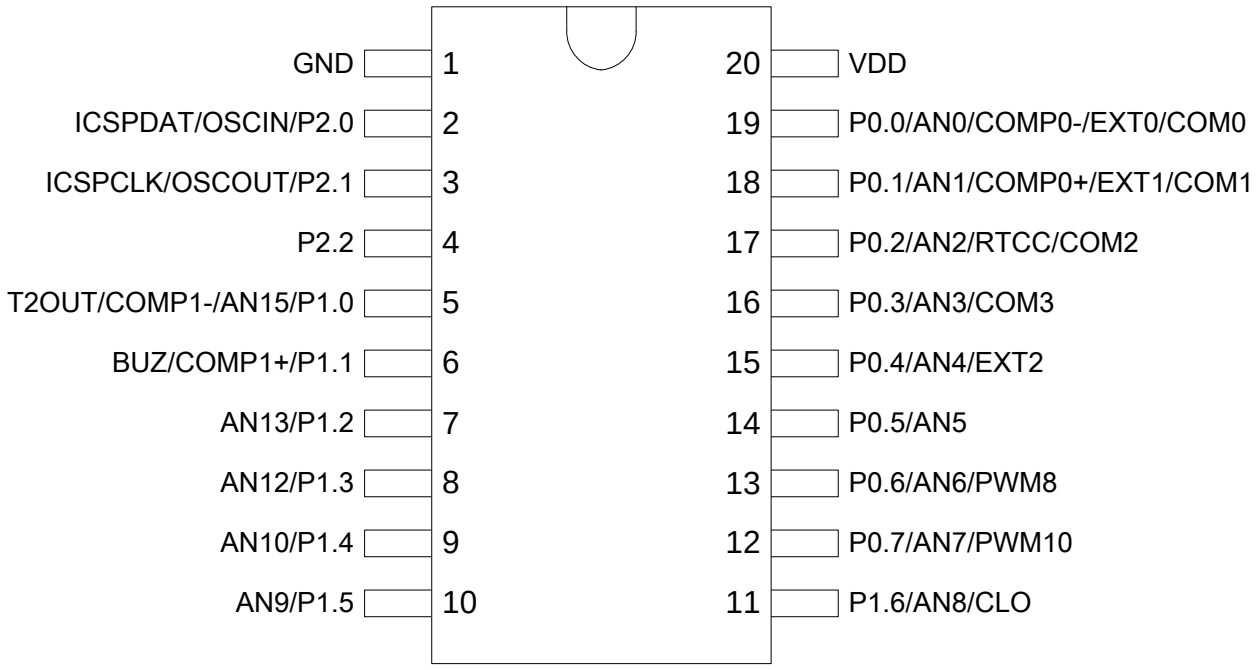
注：第 5 脚由 P0.2 和 P1.5 封装在一起，在使用时注意。



CMS89F112



CMS89F113


GND 1
ICSPDAT/OSCIN/P2.0 2
ICSPCLK/OSCOUT/P2.1 3
P2.2 4
T2OUT/COMP1-/AN15/P1.0 5
BUZ/COMP1+/P1.1 6
AN13/P1.2 7
AN12/P1.3 8
AN10/P1.4 9
AN9/P1.5 10
20 VDD
19 P0.0/AN0/COMP0-/EXT0/COM0
18 P0.1/AN1/COMP0+/EXT1/COM1
17 P0.2/AN2/RTCC/COM2
16 P0.3/AN3/COM3
15 P0.4/AN4/EXT2
14 P0.5/AN5
13 P0.6/AN6/PWM8
12 P0.7/AN7/PWM10
11 P1.6/AN8/CLO

CMS89F116

1.4 管脚描述

管脚名称	IO 类型	管脚说明	共享引脚
P0.0-P0.7	I/O	可编程为：输入口、带上拉电阻输入口、推挽输出口；也可作为 A/D 转换的模拟输入口、比较器输入口、外部中断输入口、PWM 输出口、带休眠唤醒功能	AN0-AN7,RTCC, EXT0,EXT1,EXT2,PWM8, PWM10,COMP0-,COMP0+, COM0-COM3
P1.0-P1.6	I/O	可编程为：输入口、带上拉电阻输入口、推挽输出口、开漏输出口、开漏输出口（带上拉）；也可作为比较器输入口、CLO[PWM2]输出口	AN8-AN10,AN12-AN13, AN15,BUZ,T2OUT,CLO COMP1-,COMP1+
P2.0,P2.1	I/O	可编程为：输入口、带上/下拉电阻输入口、推挽输出口、开漏输出口	OSCIN,OSCOUT ICSPDAT,ICSPCLK
P2.2	I/O	可编程为：输入口、带上拉电阻输入口、推挽输出口、开漏输出口	--
VDD,GND	P	电源电压输入脚、接地脚	--
OSCIN,OSCOUT	I/O	外部晶振管脚	P2.0,P2.1
AN0-AN15	I	A/D 转换的模拟输入口	P0.0-P0.7,P1.0-P1.6
PWM8	O	8 位 PWM 输出	P0.6
PWM10	O	10 位 PWM 输出	P0.7
CLO	O	占空比可选的系统时钟输出	P1.6
BUZ	O	蜂鸣器输出	P1.1
EXT0,EXT1,EXT2	I	外部中断输入口	P0.0,P0.1,P0.4
T2OUT	O	T2 定时/计数器输出	P1.0
COMP0-,COMP0+	I	比较器 0 输入	P0.0,P0.1
COMP1-,COMP1+	I	比较器 1 输入	P1.0,P1.1
COM0-COM3	O	LCD 模块 COM 输出端	P0.0-P0.3
ICSPDAT	I/O	编程数据输入输出	P2.0,OSCIN
ICSPCLK	I	编程时钟输入	P2.1,OSCOUT

1.5 系统配置寄存器

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

1. OSC（振荡方式选择）
   - INTRC 内部 RC 振荡
   - XT 外部晶体振荡
2. INTRC_SEL（内部 RC 振荡时钟选择,只有当 OSC 选 INTRC 时有效）
   - 8MHz
   - 4MHz
   - 2MHz
   - 1MHz
3. WDT（看门狗选择）
   - ENABLE 打开看门狗定时器
   - DISABLE 关闭看门狗定时器
4. PROTECT（加密）
   - DISABLE FLASH 代码不加密
   - ENABLE FLASH 代码加密，加密后烧写仿真器读出来的值将不确定
5. LVR_SEL（低压侦测选择）
   - 1.8V 选择此复位电压时，工作频率需设置为 4MHz 及以下
   - 2.5V
   - 3.5V
6. VREF_SEL（内部基准电源选择）
   - 0.6V
   - 1.2V

注：LVR_SEL 电压为设计电压，实际芯片的复位电压可能会有所浮动。

1.6 在线串行编程

可在最终应用电路中对 CMS89F11x 单片机进行串行编程。编程可以简单地通过以下 4 根线完成：

- 电源线
- 接地线
- 数据线
- 时钟线

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。

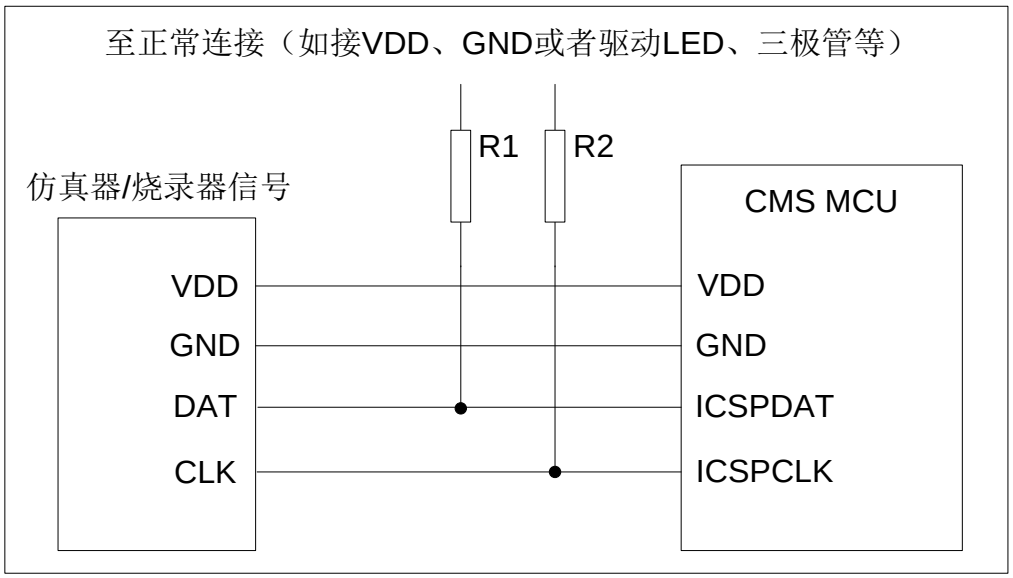


图 1-1：典型的在线串行编程连接方式

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。

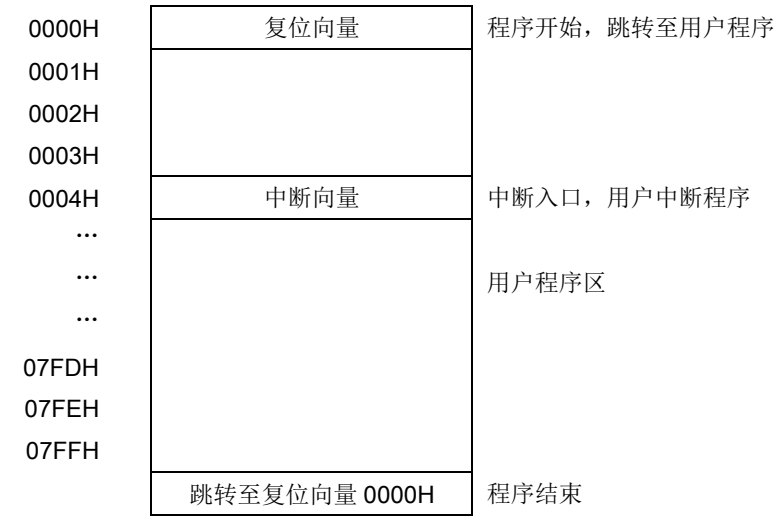
如果烧录的通信线较长，可以在 DAT、CLK 管脚接一个小电容对地，以增加通信的稳定性，其容值不能大于 100pF（101）。

2. 中央处理器（CPU）

2.1 内存

2.1.1 程序内存

FLASH:2K



2.1.1.1 复位向量（0000H）

CMS89F11x 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 FLASH 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（INT_FLAG）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 和 STATUS
                …                           ;用户中断程序
                …
INT_BACK:
                CALL        POP             ;返回 ACC 和 STATUS
                RETI                        ;中断返回
START:
                …                           ;用户程序
                …
                END                         ;程序结束
```

注：由于 CMS89F11x 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_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
```

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
```

2.1.1.3 查表

CMS89F11x 具有查表功能，FLASH 空间的任何地址都可做为查表使用。

相关指令：

- TABLE [R] 把表格内容的低字节送给寄存器 R，高字节送到寄存器 TABLE_DATAH。
- TABLEA 把表格内容的低字节送给累加器 ACC，高字节送到寄存器 TABLE_DATAH。

相关寄存器：

- TABLE_SPH 可读写寄存器，用来指明表格高 3 位地址。
- TABLE_SPL 可读写寄存器，用来指明表格低 8 位地址。
- TABLE_DATAH 只读寄存器，存放表格高字节内容。

注：在查表之前要先把表格地址写入 TABLE_SPH 和 TABLE_SPL 中。如果主程序和中断服务程序都用到查表指令，主程序中的 TABLE_SPH 的值可能会因为中断中执行的查表指令而发生变化，产生错误。也就是说要避免在主程序和中断服务程序中都使用查表指令。但如果必须这样做的话，我们可以在查表指令前先将中断禁止，在查表结束后再开放中断，以避免发生错误。

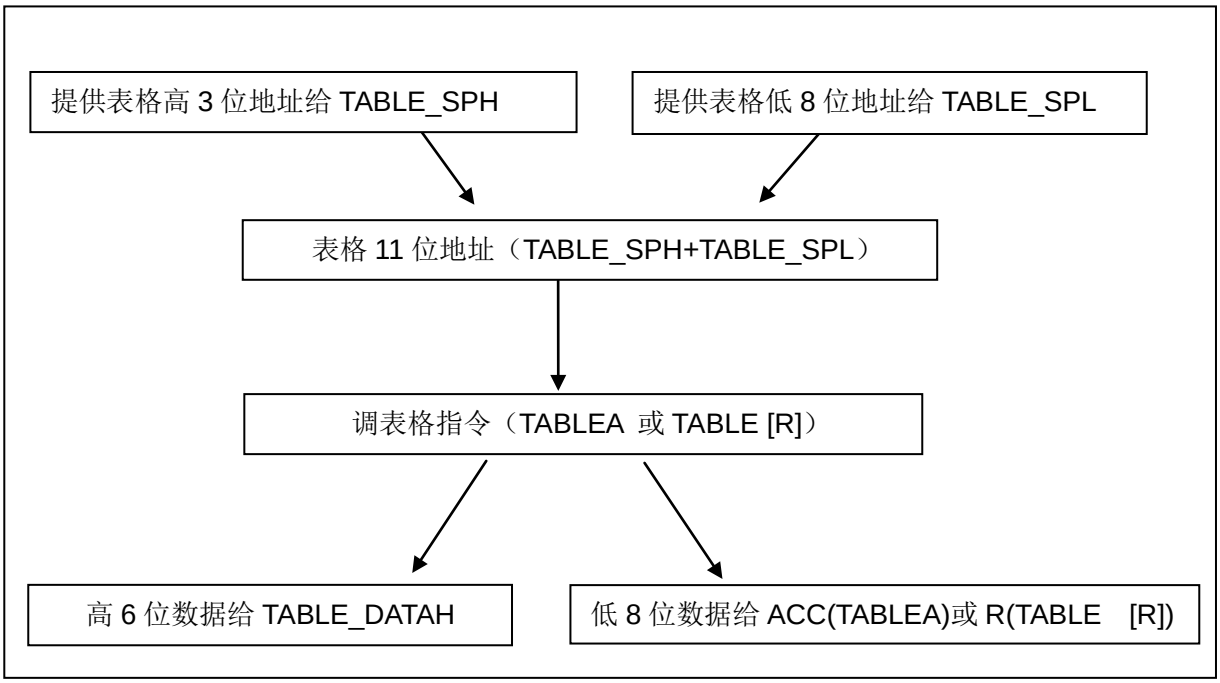


图 2-1：表格调用的流程图

下面例子给出了如何在程序中调用表格。

```
...                                 ;上接用户程序
LDIA        02H                     ;表格低位地址
LD          TABLE_SPL,A
LDIA        06H                     ;表格高位地址
LD          TABLE_SPH,A
TABLE       R01                     ;表格指令，将表格低 8 位(56H)给自定义寄存器 R01
LD          A,TABLE_DATAH           ;将查表结果的高 6 位(34H)给累加器 ACC
LD          R02,A                   ;将 ACC 值(34H)给自定义寄存器 R02
...                                 ;用户程序

ORG         0600H                   ;表格起始地址
DW          1234H                   ;0600H 地址表格内容
DW          2345H                   ;0601H 地址表格内容
DW          3456H                   ;0602H 地址表格内容
DW          0000H                   ;0603H 地址表格内容
```

2.1.1.4 跳转表

跳转表能够实现多地址跳转功能。由于 PCL 和 ACC 的值相加即可得到新的 PCL，因此，可以通过对 PCL 加上不同的 ACC 值来实现多地址跳转。ACC 值若为 n，PCL+ACC 即表示当前地址加 n，执行完当前指令后 PCL 值还会自加 1，可参考以下范例。如果 PCL＋ACC 后发生溢出，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

```
FLASH 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
```

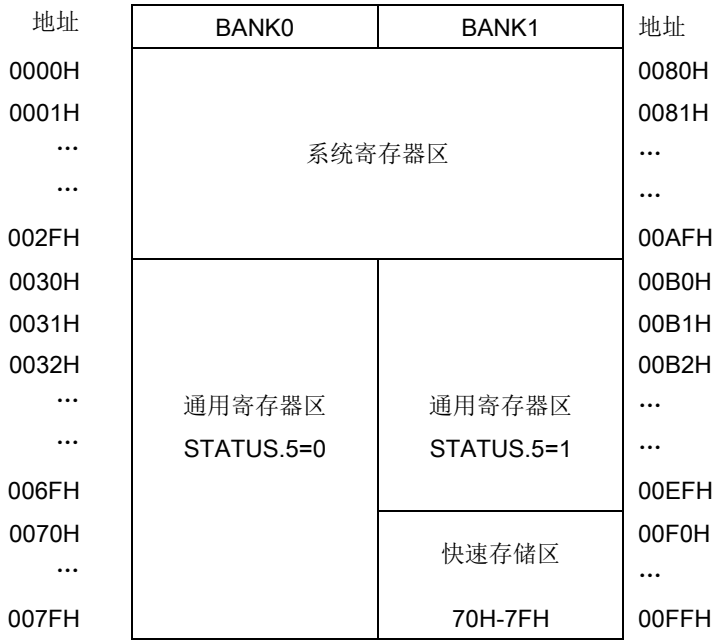
例：错误的多地址跳转程序示例

```
FLASH 地址
                CLR         PCLATH          ;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，需要注意该段程序一定不能放在 FLASH 空间的分页处。

2.1.2 数据存储器

RAM：192 字节



数据存储器由 192×8 位组成，分为两个功能区间：特殊功能寄存器（48×8，无需分页）和通用数据存储器（144×8，直接寻址时分两页，由 STATUS.5 选择页。间接寻址时不分页）。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 00H 到 2FH，通用数据寄存器地址为 30H 到 7FH。

2.1.2.1 通用数据存储器

RAM 的 0030H～007FH 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

系统专用数据存储器表

地址	名称	说明
00H	IAR	间接寻址寄存器
01H	TMR0	内部定时/计数器
02H	PCL	程序指针 PC 低 8 位
03H	STATUS	系统状态标志寄存器
04H	MP	间接寻址指针
05H	P0	P0 IO 口数据寄存器
06H	P1	P1 IO 口数据寄存器
07H	P2	P2 IO 口数据寄存器
08H	--------	未用
09H	P0CL	P0 IO 口功能控制寄存器
0AH	PCLATH	程序指针 PC 高 3 位写缓冲器
0BH	P1CL	P1 IO 口功能控制寄存器
0CH	P1CH	P1 IO 口功能控制寄存器
0DH	P2C	P2 IO 口功能控制寄存器
0EH	P0CH	P0 IO 口功能控制寄存器
0FH	OPTION_REG	预分频器
10H	SYS_GEN	中断总使能及 ADC 使能
11H	INT_EN	中断控制寄存器使能位
12H	INT_FLAG	中断控制寄存器标志位
13H	INT_EXT	外部中断边沿触发设置寄存器
14H	ADDATAH	AD 结果存放寄存器高 8 位（只读）
15H	ADCON	AD 控制寄存器
16H	TMR1	定时计数器 1
17H	TMR1C	定时计数器 1 控制寄存器
18H	T2CNT	TMR2 计数器（只读）
19H	T2CON	TMR2 控制寄存器
1AH	T2DATA	TMR2 数据寄存器
1BH	ADDATAL	AD 结果存放寄存器低 4 位（只读）
1CH	PWM8DATA	8 位 PWM 数据寄存器
1DH	PWM8CON	8 位 PWM 控制寄存器及 CLO 输出控制
1EH	PWM10CON	10 位 PWM 控制寄存器
1FH	PWM10DATA	10 位 PWM 数据寄存器
20H	COMPCON	比较器控制寄存器
21H	BUZCON	蜂鸣器控制寄存器
22H	TABLE_SPH	查表高 3 位地址
23H	TABLE_SPL	查表低 8 位地址
24H	TABLE_DATAH	查表高 6 位结果
25H	EEDAT	EE 数据区数据低位
26H	EEDATH	EE 数据区数据高位
27H	EEADR	EE 数据区地址寄存器
28H	LCDCON	LCD控制寄存器
29H	IOCA	P0口电平变化中断控制寄存器
2AH	EECON1	EE 数据区控制寄存器 1
2BH	EECON2	EE 数据区控制寄存器 2
2CH-2FH	--------	未用

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 IAR 寄存器可间接寻址，IAR 不是物理寄存器。当对 IAR 进行存取时，它会根据 MP 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 MP 寄存器后，就可把 IAR 寄存器当作目的寄存器来存取。间接读取 IAR（MP=0）将产生 00H。间接写入 IAR 寄存器，将导致一个空运作。间接寻址时不分页，寻址地址为 00H-0FFH，以下例子说明了程序中间接寻址的用法。

例：MP 及 IAR 的应用

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

例：间接寻址清通用 RAM（30H-7FH）举例

```
        LDIA    2FH
        LD      MP,A        ;间接寻址指针指向 2FH
LOOP:
        INCR    MP          ;地址加 1，初始地址为 30H
        CLR     IAR         ;清零 MP 所指向的地址
        LDIA    7FH
        SUBA    MP
        SNZB    STATUS,C    ;一直清零至 MP 地址为 7FH
        JP      LOOP
```

例：间接寻址清通用 RAM（0B0H-0EFH）举例

```
        LDIA    0AFH
        LD      MP,A            ;间接寻址指针指向 AFH
LOOP1:
        INCR    MP              ;地址加 1，初始地址为 B0H
        CLR     IAR             ;清零 MP 所指向的地址
        LDIA    0EFH
        SUBA    MP
        SNZB    STATUS,C        ;一直清零至 MP 地址为 0EFH
        JP      LOOP1
```

2.3 堆栈

CMS89F11x 的堆栈缓存器共 8 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。

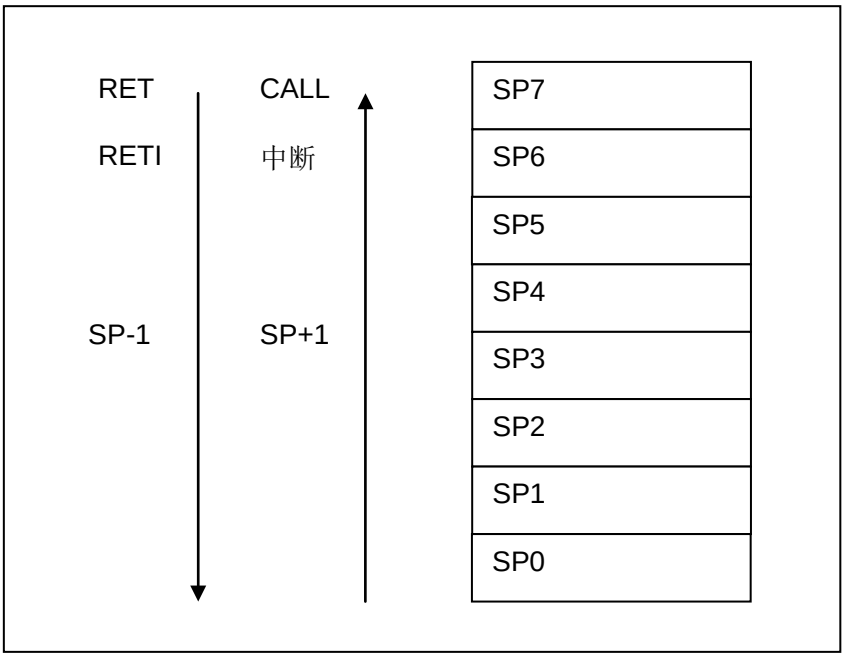


图 2-2：堆栈缓存器工作原理

堆栈缓存器的使用遵循“先进后出”的原则。

注：堆栈缓存器只有 8 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 8 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 8-Bit 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）

寄存器 STATUS 中包含 ALU 运算状态信息、系统复位状态信息、RAM 分页选择。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 RP 显示 RAM 的分页选择；位 C、DC 和 Z 显示 ALU 的运算信息。

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

Bit5 RP: 寄存器存储区选择位（直接寻址时有效，间接寻址时无效）；
1= Bank 1；
0= Bank 0。
Bit4 TO: WDT溢出标志；
1= 上电或执行了CLRWDT指令或STOP指令；
0= 发生了WDT超时。
Bit3 PD: 低功耗标志；
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；
0= 结果的低4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：“CLR STATUS”的结果是 STATUS=“XX0UU100”，“U”指未改变，而不是想象中的全零。也就是说执行指令后，PD 和 TO 的值保持不变，而 Z 标志位因清零而置 1，所以若需要改变 STATUS 的值 建议使用“SETB”、“CLRB”、“LD R,A”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 PD、TO 的事件表

TO	PD	复位原因
0	0	休眠态 WDT 溢出唤醒
0	1	非休眠态 WDT 溢出
1	0	----
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。

0FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OPTION_REG	---	---	T0CS	T0SE	PSA	PS2	PS1	PS0
读写			W	W	W	W	W	W
复位值	X	X	1	1	1	1	1	1

Bit7~ Bit6　未用

Bit5　T0CS:　TMR0 时钟源选择位。
0:　内部时钟。
1:　外部时钟（RTCC 口输入波形）。

Bit4　T0SE:　RTCC 信号触发源选择位。
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；用于定时器或计数器时，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。它可以动态改变 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
```

2.7 程序计数器（PC）

程序计数器（PC）控制程序内存 FLASH 中的指令执行顺序，它可以寻址整个 FLASH 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 11-Bit 宽度，低 8 位通过 PCL 寄存器用户可以访问，高 3 位通过 PCLATH 寄存器进行写缓冲。可容纳 2K×14Bit 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[10:8]=PCLATH[2:0]，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 CMS89F11x 系列芯片中集成了 CONFIG 选项，可将其置“0”来使 WDT 不起作用，详见 1.5 章 CONFIG 烧写的选择。

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。

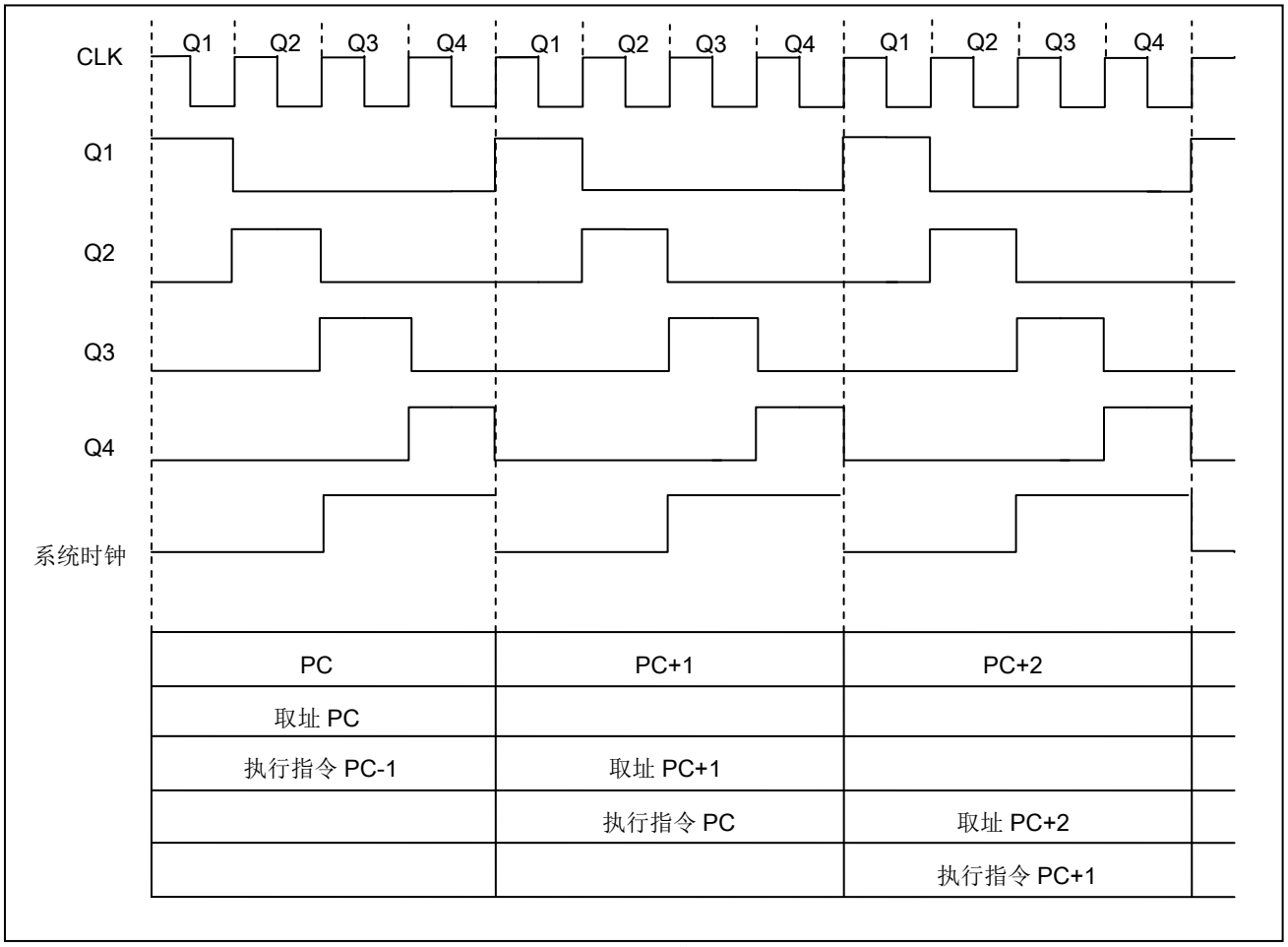


图 3-1：时钟与指令周期时序图

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	8μs	4μs
2MHz	4μs	2μs
4MHz	2μs	1μs
8MHz	1μs	500ns

3.2 系统振荡器

CMS89F11x 有 2 种振荡方式：内部 RC 振荡和外部 XT 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。振荡频率在出厂时校正，其误差范围详见电气参数章节。

当选择内部 RC 作为芯片的振荡器时，芯片的 OSCIN(P2.0)和 OSCOUT(P2.1)可以作为普通的 I/O 口。

3.2.2 外部 XT 振荡

在烧录时将 CONFIG 选项中的 OSC 选择成 XT，芯片工作在外部 XT 振荡模式下，此时内部 RC 振荡停止工作，OSCIN(P2.0)和 OSCOUT(P2.1)作为振荡口，同时 P2.2 口只能作为输入口。

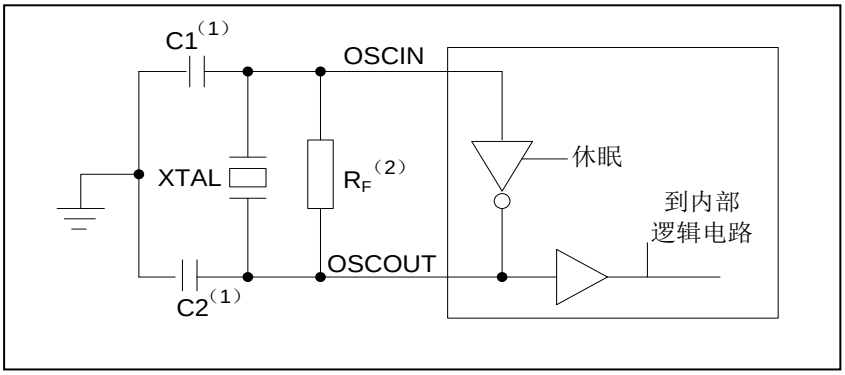


图 3-2：典型的 XT 振荡方式

建议参数：

类型	频率	建议值 R_F	建议值 C1～C2
XT	455KHz	1M	100pF～470pF
XT	2MHz	1M	10pF～47pF
XT	4MHz	1M	10pF～47pF
XT	8MHz	1M	10pF～47pF

3.3 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。

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。

4. 复位

CMS89F11x 可用如下 4 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；
- 休眠模式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。

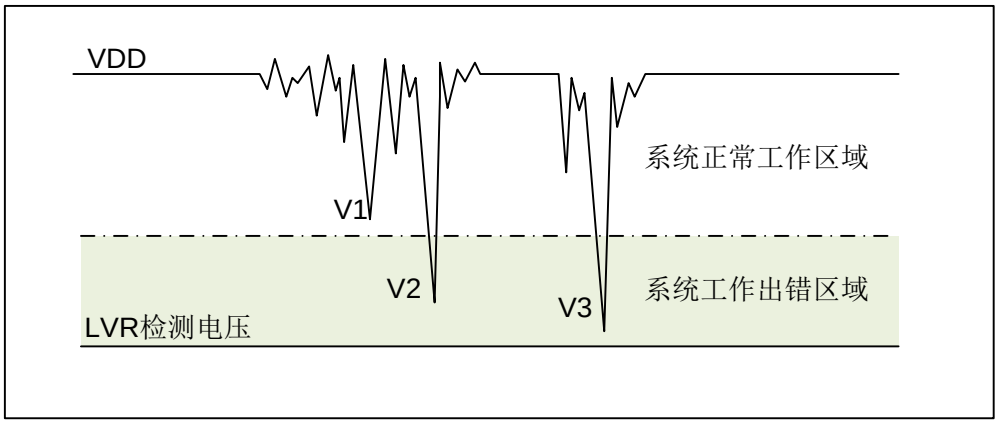


图 4-1：掉电复位示意图

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。

4.2.1 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

CMS89F11x 系列芯片，内部集成了低压侦测（LVR）功能，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。

芯片内部有 3 种复位电压选择：1.8V、2.5V、3.5V；这些电压值是设计标称电压，实际使用时会有所浮动。

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. 系统工作模式

CMS89F11x 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

休眠模式是被 STOP 指令启动的，在休眠模式下，系统振荡停止，以减小功耗，且所有外围停止工作。休眠模式可由复位、WDT 溢出或者 P0 口的下降沿而唤醒。当休眠模式被唤醒时，时钟电路仍需要振荡稳定时间。当休眠模式由复位或者 WDT 溢出被唤醒时，系统从 0000H 地址开始执行程序；当休眠模式由 P0 口的下降沿唤醒时，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；关断 AD 模块及比较器模块；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：进入休眠模式的处理程序

```
SLEEP_MODE：
        CLR     SYS_GEN     ;关断 ADC 模块及中断使能
        LDIA    B’10101010’
        LD      P0CL,A
        LD      P0CH,A      ;P0 口设置为输出口
        LD      P1CL,A
        LDIA    B’10010010’
        LD      P1CH,A      ;P1 口设置为输出口
        LDIA    B’00010010’
        LD      P2C,A       ;P2 口设置为输出口(若采用 INTRC 模式)
        CLR     BUZCON      ;禁止 BUZ 功能
        CLR     PWM8CON     ;禁止 8 位 PWM 功能
        CLR     PWM10CON    ;禁止 10 位 PWM 功能
        …                   ;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 3 种条件可以让 CPU 退出休眠状态：

- 看门狗溢出；
- P0 口下降沿；
- 系统掉电后，重新上电。

处于休眠态的 MCU，发生 P0 口下降沿唤醒时，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         ;不是从休眠模式复位的处理程序
                SZB     STATUS,TO
                JP      START_3         ;非 WDT 唤醒 MCU 处理程序
                JP      START_4         ;WDT 唤醒 MCU
                ...
                ...
SLEEP_MODE:                             ;休眠子程序
                ...                     ;设置休眠前的状态
                STOP                    ;芯片进入休眠模式处理程序
                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），这个时间为 18ms。

6. I/O 端口

CMS89F11x 有三个 I/O 端口：P0、P1 和 P2（最多 18 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	管脚描述	输入/输出
P0	0	19	施密特触发输入，推挽式输出，AN0，COMP0 -，EXT0	I/O
	1	18	施密特触发输入，推挽式输出，AN1，COMP0+，EXT1	I/O
	2	17	施密特触发输入，推挽式输出，AN2	I/O
	3	16	施密特触发输入，推挽式输出，AN3	I/O
	4	15	施密特触发输入，推挽式输出，AN4，EXT2	I/O
	5	14	施密特触发输入，推挽式输出，AN5	I/O
	6	13	施密特触发输入，推挽式输出，AN6，PWM8	I/O
	7	12	施密特触发输入，推挽式输出，AN7，PWM10	I/O
P2	0	2	施密特触发输入，推挽式输出，开漏式输出，OSCIN	I/O
	1	3	施密特触发输入，推挽式输出，开漏式输出，OSCOUT	I/O
	2	4	施密特触发输入，推挽式输出，开漏式输出	I/O
P1	0	5	施密特触发输入，推挽式输出，开漏式输出，T2OUT，COMP1-，AN15	I/O
	1	6	施密特触发输入，推挽式输出，开漏式输出，BUZ，COMP1+	I/O
	2	7	施密特触发输入，推挽式输出，开漏式输出，AN13	I/O
	3	8	施密特触发输入，推挽式输出，开漏式输出，AN12	I/O
	4	9	施密特触发输入，推挽式输出，开漏式输出，AN10	I/O
	5	10	施密特触发输入，推挽式输出，开漏式输出，AN9	I/O
	6	11	施密特触发输入，推挽式输出，开漏式输出，CLO，AN8	I/O

＜表 6-1：端口配置总概＞

6.1 I/O 口结构图

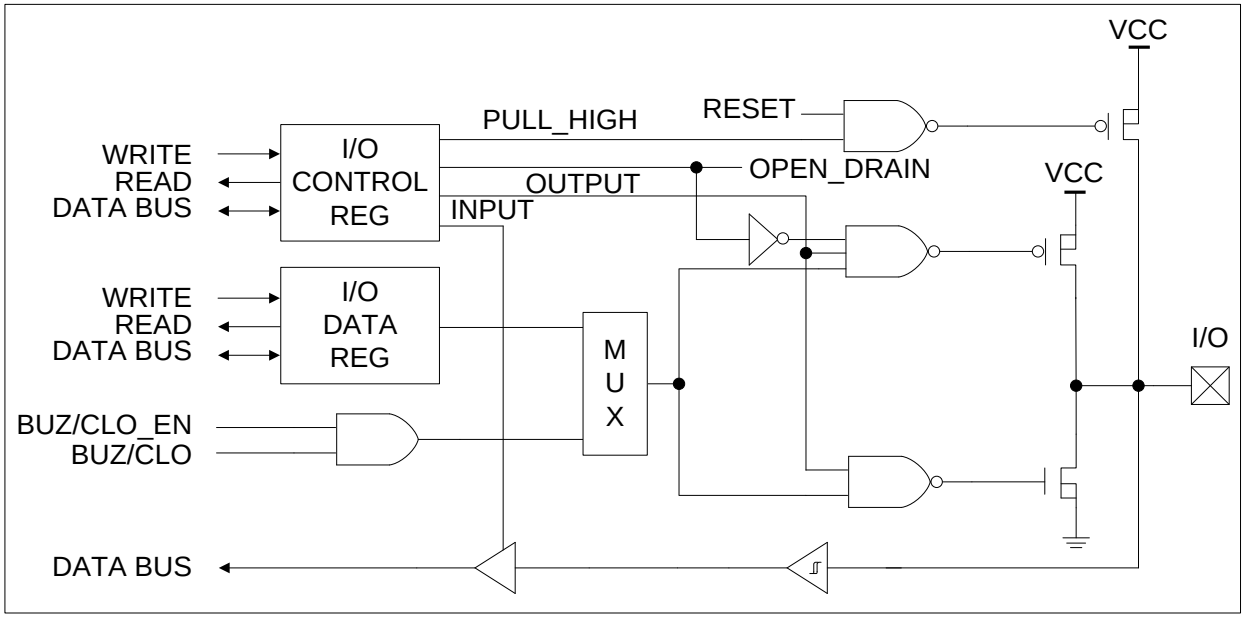


图 6-1：I/O 口结构图（1）

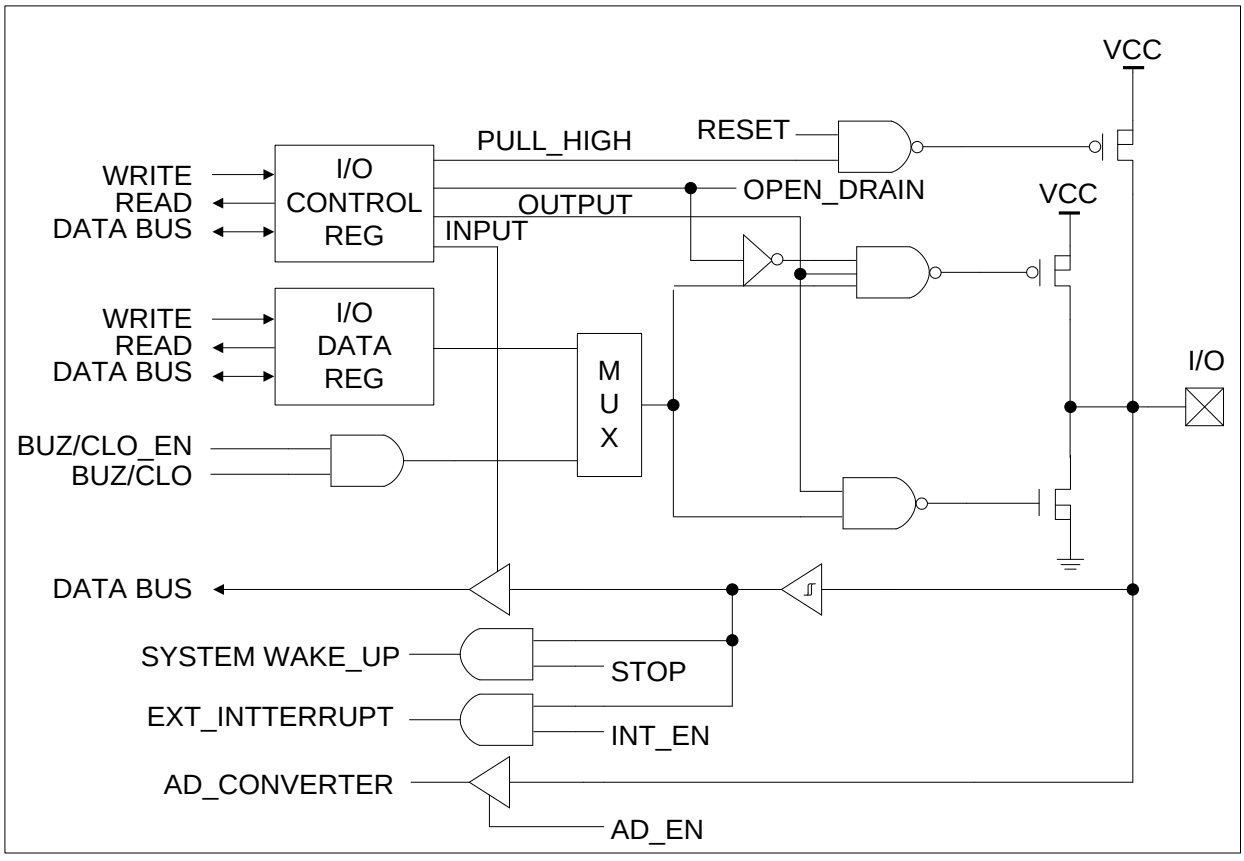


图 6-2：I/O 口结构图（2）

6.2 I/O 口模式及上、下拉电阻

寄存器 P0CL、P0CH、P1CL、P1CH、P2C 用于控制 I/O 口线的工作模式。

6.2.1 P0 口

CMS89F11x 芯片的 P0 口是一个 8Bit 的 I/O 口，有三个寄存器与之相关。分别为 IO 口数据寄存器（P0）、IO 口功能控制寄存器（P0CL、P0CH）。

P0 口数据寄存器 P0

05H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
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
	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
	PWM10	PWM8		EXT2		RTCC	EXT1	EXT0
							COMP0+	COMP0-
					COM3	COM2	COM1	COM0

P0 口功能寄存器 P0CL

09H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P0CL	P0.3 功能配置		P0.2 功能配置		P0.1 功能配置		P0.0 功能配置	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	1	0	1	0	1	0	1

Bit7~Bit6 P0.3 功能配置
- 00: 上拉输入；
- 01: 输入；
- 10: 推挽输出；
- 11: AN3。

Bit5~Bit4 P0.2 功能配置
- 00: 上拉输入；
- 01: 输入；
- 10: 推挽输出；
- 11: AN2。

Bit3~Bit2 P0.1 功能配置
- 00: 上拉输入、中断；
- 01: 输入、中断、比较器 0 的“+”端；
- 10: 推挽输出；
- 11: AN1。

Bit1~Bit0 P0.0 功能配置
- 00: 上拉输入、中断；
- 01: 输入、中断、比较器 0 的“-”端；
- 10: 推挽输出；
- 11: AN0。

P0 口功能寄存器 P0CH

0EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P0CH	P0.7 功能配置		P0.6 功能配置		P0.5 功能配置		P0.4 功能配置	
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	0	1	0	1

Bit7~Bit6 P0.7 功能配置
- 00: 上拉输入；
- 01: PWM10 输出；
- 10: 推挽输出；
- 11: AN7。

Bit5~Bit4 P0.6 功能配置
- 00: 上拉输入；
- 01: PWM8 输出；
- 10: 推挽输出；
- 11: AN6。

Bit3~Bit2 P0.5 功能配置
- 00: 上拉输入；
- 01: 输入；
- 10: 推挽输出；
- 11: AN5。

Bit1~Bit0 P0.4 功能配置
- 00: 上拉输入、中断；
- 01: 输入、中断；
- 10: 推挽输出；
- 11: AN4。

例：P0 口处理程序

```
LDIA    B'00011111'     ;P0.7 为上拉输入，P0.6 为 PWM 输出
LD      P0CH,A          ;P0.4-P0.5 为 AD 输入
LDIA    B'10101010'
LD      P0CL,A          ;P0.0-P0.3 为推挽输出
LDIA    03H             ;P0.0-P0.1 输出高，P0.2-P0.3 输出低
LD      P0,A            ;由于 P0.4、P0.5、P0.7 为输入口，P0.6 为 PWM 口，所以赋
                        0 或 1 都没影响
```

6.2.2 P1 口

P1 口有 7-Bit 的输入输出管脚。它可做正常输入输出端口（施密特触发输入、推挽式输出、开漏式输出）或者是一些选择性功能用（时钟输出、T0 时钟输出、蜂鸣器输出）。有三个寄存器与之相关。P1 口数据寄存器 P1、P1 口低位控制寄存器 P1CL、P1 口高位控制寄存器 P1CH。

P1 口数据寄存器 P1

06H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P1	---	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
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
		AN8	AN9	AN10	AN12	AN13		AN15
		CLO				RTCC	BUZ	T2OUT
							COMP1+	COMP1-

P1 口功能寄存器 P1CL

0BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P1CL	P1.3 功能配置		P1.2 功能配置		P1.1 功能配置		P1.0 功能配置	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	1	0	1	0	0	0	1

Bit7~Bit6 P1.3 功能配置
- 00: 上拉输入；
- 01: 输入，AN12；
- 10: 推挽输出；
- 11: 开漏输出，（只能输出“0”和截止态）。

Bit5~Bit4 P1.2 功能配置
- 00: 上拉输入；
- 01: 输入，AN13；
- 10: 推挽输出；
- 11: 开漏输出，（只能输出“0”和截止态）。

Bit3~Bit2 P1.1 功能配置
- 00: 上拉输入；
- 01: 蜂鸣器输出；
- 10: 推挽输出；
- 11: 开漏输出（只能输出“0”和截止态）。

Bit1~Bit0 P1.0 功能配置
- 00: 上拉输入；
- 01: 输入，AN15；
- 10: 推挽输出；
- 11: T2OUT 输出。

P1 口功能寄存器 P1CH

0CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P1CH	P1.6 功能配置			P1.5 功能配置			P1.4 功能配置	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	1	0	0	1	0	1

Bit7~Bit5 P1.6 功能配置

000: 上拉输入；
001: 输入，AN8；
01X: 未用；
100: 推挽输出；
101: 开漏输出（带上拉）；
110: 开漏输出；
111: CLO 输出。

Bit4~Bit2 P1.5 功能配置

000: 上拉输入；
001: 输入，AN9；
01X: 未用；
100: 推挽输出；
101: 开漏输出（带上拉）；
110: 开漏输出；
111: 未用。

Bit1~Bit0 P1.4 功能配置

00: 上拉输入；
01: 输入，AN10；
10: 推挽输出；
11: 开漏输出。

注：P1 口的使用方法同 P0 口。

6.2.3 P2 口

P2 口数据寄存器 P2

07H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P2	---	---	---	---	---	P2.2	P2.1	P2.0
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

P2 口功能寄存器 P2C

0DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
P2C	P2.2 功能配置		P2.1 功能配置			P2.0 功能配置		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	1	0	0	1	0	0	1

Bit7~Bit6 P2.2 功能配置
- 00: 上拉输入；
- 01: 输入；
- 10: 推挽输出；
- 11: 开漏输出。

Bit5~Bit3 P2.1 功能配置
- 000: 上拉输入；
- 001: 输入；
- 010: 推挽输出；
- 011: 下拉输入；
- 100: 开漏输出；
- 其他: 未用。

Bit2~Bit0 P2.0 功能配置
- 000: 上拉输入；
- 001: 输入；
- 010: 推挽输出；
- 011: 下拉输入；
- 100: 开漏输出；
- 其他: 未用。

6.2.4 写 I/O 口

CMS89F11x 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      P0,A        ;ACC 值赋给 P0 口
        CLRB    P1,0        ;P1.0 口置零
        CLR     P2          ;P2 口清零
        SET     P1          ;P1 所有输出口置 1
        SETB    P1.0        ;P1.0 口置 1
```

6.2.5 读 I/O 口

例：读 I/O 口程序

```
        LD      A,P0        ;P0 的值赋给 ACC
        SNZB    P0,1        ;判断 P0.1 口是否为 1，为 1 跳过下一条语句
        SZB     P0,1        ;判断 P0.1 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。

6.3 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。

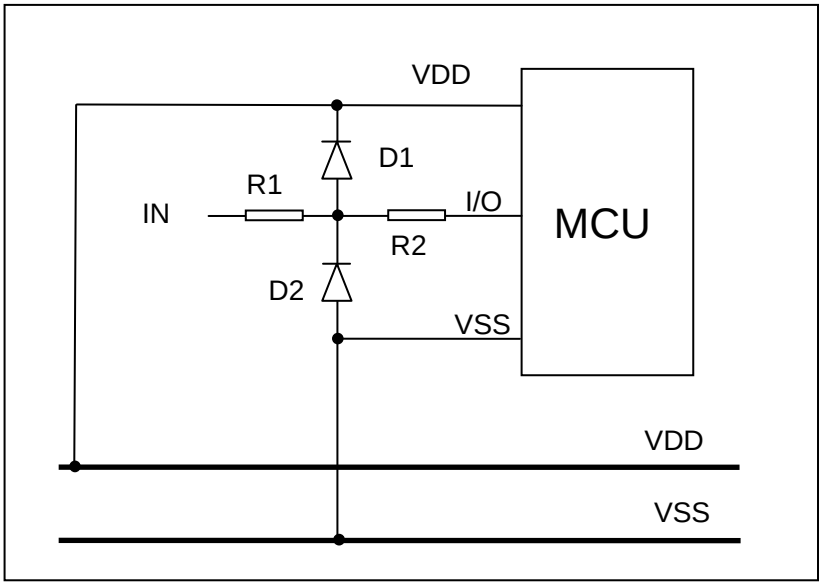


图 6-3：I/O 口注意事项连接图

4. 若在 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。

7. 中断

7.1 中断概述

CMS89F11x 共有 7 个中断源：3 个内部中断（TMR1、TMR2、ADC）和 4 个外部中断（EXT0、EXT1、EXT2、P0C）。一旦程序进入中断，寄存器 SYS_GEN 的位 INT_GEN 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 INT_GEN 置“1”，以响应下一个中断。中断请求存放在寄存器 INT_FLAG 寄存器中。

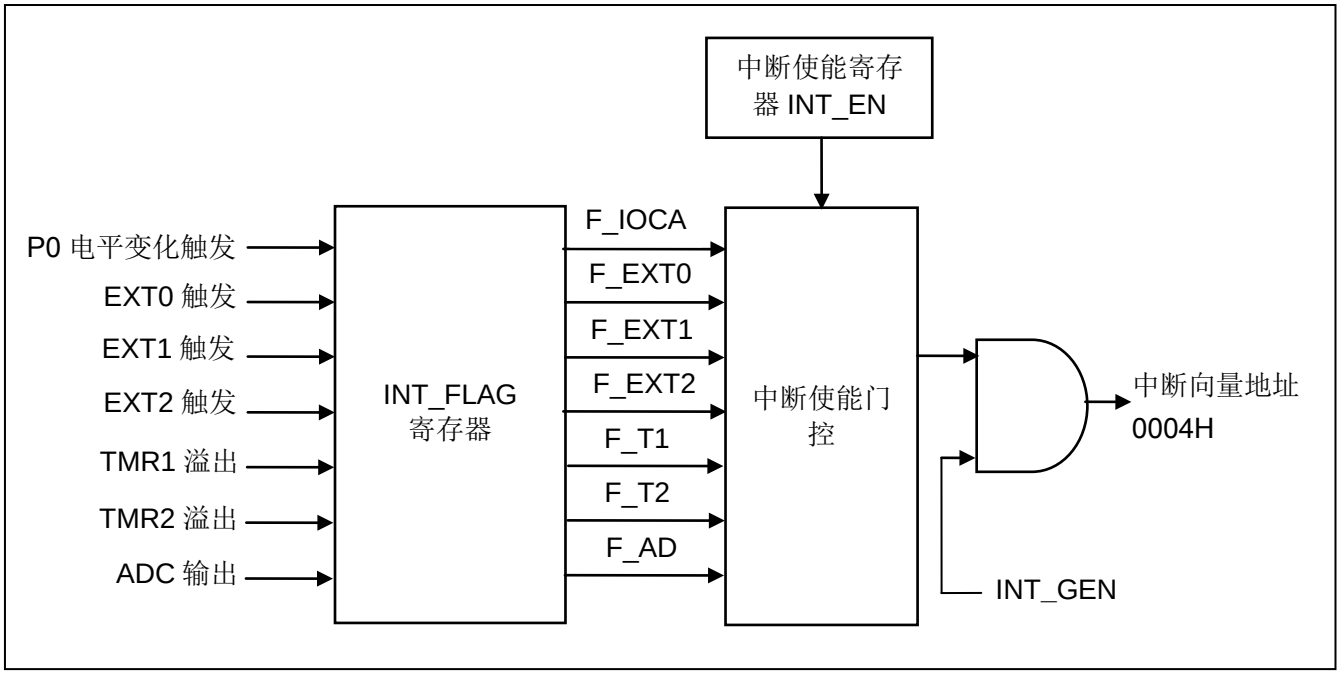


图 7-1：中断原理示意图

7.2 中断控制寄存器

中断请求控制寄存器 INT_EN 包括所有中断的使能控制位。INT_EN 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。

11H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT_EN	EN_IOCA	EN_EXT2	EN_AD	---	EN_T1	EN_T2	EN_EXT1	EN_EXT0
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 EN_IOCA: P0口电平变化中断使能位；
0: 禁止P0口电平变化中断；
1: 使能P0口电平变化中断。
Bit6 EN_EXT2: 外部中断2中断使能位；
0: 禁止EXT2中断；
1: 使能EXT2中断。
Bit5 EN_AD: AD中断使能位；
0: 禁止ADC中断；
1: 使能ADC中断。
Bit4 未用
Bit3 EN_T1: TMR1中断使能位；
0: 禁止TMR1中断；
1: 使能TMR1中断。
Bit2 EN_T2: TMR2中断使能位；
0: 禁止TMR2中断；
1: 使能TMR2中断。
Bit1 EN_EXT1: 外部中断1中断使能位；
0: 禁止EXT1中断；
1: 使能EXT1中断。
Bit0 EN_EXT0: 外部中断0中断使能位；
0: 禁止EXT0中断；
1: 使能EXT0中断。

7.3 中断请求寄存器

中断请求寄存器 INT_FLAG 中存放各中断请求标志。一旦有中断请求发生，INT_FLAG 中的相应位将被置“1”，该请求被响应后，程序应将该标志位清零，MCU 不会自动清零该中断请求标志位。根据 INT_FLAG 的状态，程序判断是否有中断发生，并执行相应的中断服务。

12H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT_FLAG	F_IOCA	F_EXT2	F_AD	---	F_T1	F_T2	F_EXT1	F_EXT0
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

Bit7 F_IOCA P0口电平变化中断请求标志位；
0: 无P0口电平变化中断请求；
1: 有P0口电平变化中断请求。
Bit6 F_EXT2 外部中断2中断请求标志位；
0: 无EXT2中断请求；
1: 有EXT2中断请求。
Bit5 F_AD AD中断请求标志位；
0: 无AD中断请求；
1: 有AD中断请求。
Bit4 未用
Bit3 F_T1 TMR1中断请求标志位；
0: 无TMR1中断请求；
1: 有TMR1中断请求。
Bit2 F_T2 TMR2中断请求标志位；
0: 无TMR2中断请求；
1: 有TMR2中断请求。
Bit1 F_EXT1 外部中断1中断请求标志位；
0: 无EXT1中断请求；
1: 有EXT1中断请求。
Bit0 F_EXT0 外部中断0中断请求标志位；
0: 无EXT0中断请求；
1: 有EXT0中断请求。

7.4 总中断使能控制寄存器

只有当全局中断控制位 INT_GEN 置“1”的时候程序才能响应中断请求。一旦有中断发生，程序计数器（PC）指向中断向量地址（0004H），堆栈层数加 1。

10H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SYS_GEN	---	---	---	---	---	---	ADC_EN	INT_GEN
R/W	---	---	---	---	---	---	R/W	R/W
复位值	X	X	X	X	X	X	1	1

Bit1 ADC_EN: AD功能使能位；
0: 禁止ADC功能；
1: 使能ADC功能。

Bit0 INT_GEN: 中断总使能位；
0: 禁止所有中断；
1: 使能中断功能。

7.5 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_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
```

7.6 外部中断

CMS89F11x 系列芯片有三个外部中断源（EXT0、EXT1、EXT2），只有当外部中断被触发，且 EN_EXT0、EN_EXT1、EN_EXT2 使能时，F_EXT0、F_EXT1、F_EXT2 才会被置 1。当任何一个中断使能位与中断请求标志位同时为“1”，且总中断使能位为使能状态，系统就会响应中断。

7.6.1 外部中断控制寄存器

CMS89F11x 系列芯片的三个外部中断源（EXT0、EXT1、EXT2）的触发方式可由 INT_EXT 寄存器控制，用户可通过写 INT_EXT 控制外部中断源的触发方式。

13H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
INT_EXT	---	---	EXT2		EXT1		EXT0	
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	X	0	0	0	0	0	0

Bit5~Bit4 EXT2: EXT2信号边缘选择；
00: 下降沿；
01: 上升沿；
1X: 两种边沿。

Bit3~Bit2 EXT1: EXT1信号边缘选择；
00: 下降沿；
01: 上升沿；
1X: 两种边沿。

Bit1~Bit0 EXT0: EXT0信号边缘选择；
00: 下降沿；
01: 上升沿；
1X: 两种边沿。

7.6.2 外部中断 0

外部中断 EXT0 与 P0.0 口共用一个 I/O 口，若要启用外部中断功能，要将 P0.0 口设置为中断输入模式，当 EN_EXT0=1 且 F_EXT0=1 时，系统会响应外部中断 0，当 EN_EXT0=0 时无论 F_EXT0 为任何状态，都不会响应中断。

例：外部中断 0 应用程序

```
                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：
                …
                LDIA    B’10101001’
                LD      P0CL,A                  ;P0.0 口设置为中断输入口
                CLR     INT_EXT                 ;EXT0 为下降沿触发
                CLRB    INT_FLAG,F_EXT0         ;清零 EXT0 中断请求标志位
                SETB    INT_EN,EN_EXT0          ;使能 EXT0 中断
                SETB    SYS_GEN,INT_GEN         ;使能 INT_GEN
                …
                JP      START
INT_SERVICE：
PUSH：                                          ;中断服务程序入口，保存 ACC 及 STATUS
                …
                SNZB    INT_EN,EN_EXT0          ;判断是否使能 EXT0 中断
                JP      INT_EXIT
                SNZB    INT_FLAG,F_EXT0         ;检查有无 EXT0 中断请求标志位
                JP      INT_EXIT
                CLRB    INT_FLAG,F_EXT0         ;清零 EXT0 中断请求标志位
                …                               ;EXT0 中断处理程序
INT_EXIT：
POP：                                           ;中断服务程序出口，还原 ACC 及 STATUS
                …
                RETI
```

7.6.3 外部中断 1

外部中断 EXT1 与 P0.1 口共用一个 I/O 口，若要启用外部中断功能，要将 P0.1 口设置为中断输入模式，当 EN_EXT1=1 且 F_EXT1=1 时，系统会响应外部中断 1，当 EN_EXT1=0 时无论 F_EXT1 为任何状态，都不会响应中断。应用程序请参考外部中断 0。

7.6.4 外部中断 2

外部中断 EXT2 与 P0.4 口共用一个 I/O 口，若要启用外部中断功能，要将 P0.4 口设置为中断输入模式，当 EN_EXT2=1 且 F_EXT2=1 时，系统会响应外部中断 1，当 EN_EXT2=0 时无论 F_EXT2 为任何状态，都不会响应中断。应用程序请参考外部中断 0。

7.6.5 外部中断的响应时间

外部中断的响应时间为 2 个指令周期。

7.6.6 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。

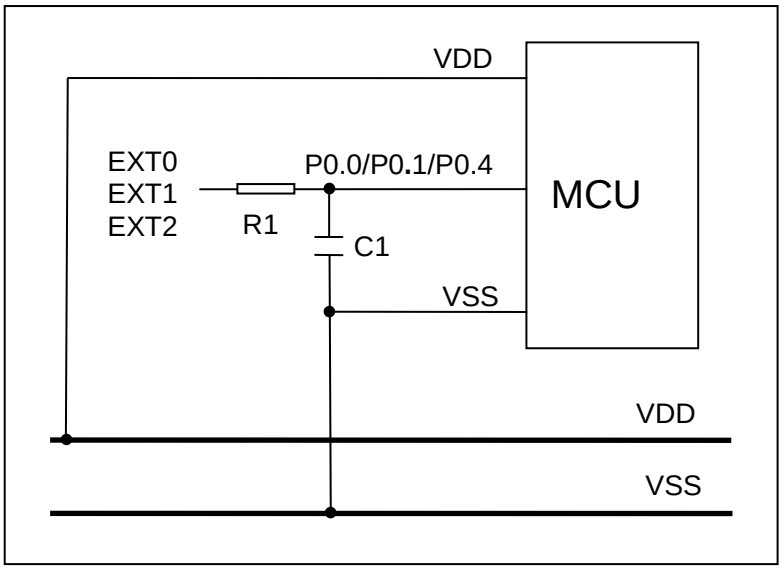


图 7-2：外部中断 I/O 连接图

7.7 P0 电平变化中断

所有的 P0 引脚都可以被单独配置为电平变化中断引脚。控制位 IOCA<7:0>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 P0 时锁存的旧值进行比较，两次的数据不相同则将 INT_FLAG 寄存器中的 P0 电平变化中断标志位（F_IOCA）置 1。

用户可在中断服务程序中通过以下方式清除中断：

1) 对 P0 进行读或写操作。这将结束引脚电平的不匹配状态。
2) 将标志位 F_IOCA 清零。

不匹配状态会不断将 F_IOCA 标志位置 1。而读或写 P0 将结束不匹配状态，并且允许将 F_IOCA 标志位清零。锁存器将保持最后一次读取的值不受欠压复位的影响。在复位之后，如果不匹配仍然存在，F_IOCA 标志位将继续置 1。

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 F_IOCA 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。

P0 电平变化中断寄存器 IOCA

29H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOCA	IOCA7	IOCA6	IOCA5	IOCA4	IOCA3	IOCA2	IOCA1	IOCA0
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 IOCA<7:0>: P0 的电平变化中断控制位
0= 允许电平变化中断
1= 禁止电平变化中断

7.8 内部定时中断

CMS89F11x 系列芯片内部有三个定时器，TMR0、TMR1、TMR2，只有 TMR1 和 TMR2 可以产生中断。

7.8.1 TMR1 中断

TMR1 溢出时，如果 EN_T1 置“1”，则 F_T1 会被置“1”，系统就会响应 TMR1 的中断；若 EN_T1= 0，则 F_T1 不会被置“1”，系统不会响应 TMR1 中断。尤其需要注意多种中断下的情形。

例：TMR1 中断应用程序

```
                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：
                …
                LDIA    06H
                LD      TMR1,A              ;初始化 TMR1
                LDIA    80H
                LD      TMR1C,A             ;设置 TMR1 时钟=Fcpu、TMR1 定时器模式
                CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位
                SETB    INT_EN,EN_T1        ;使能 TMR1 中断
                SETB    TMR1C,TON           ;TMR1 计时器开始工作
                SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN
                …
MAIN：
                …
                JP      MAIN
INT_SERVICE：
PUSH：                                      ;中断服务程序入口，保存 ACC 及 STATUS
                …
                SNZB    INT_EN,EN_T1        ;判断是否使能 TMR1 中断
                JP      INT_EXIT
                SNZB    INT_FLAG,F_T1       ;检查有无 TMR1 中断请求标志位
                JP      INT_EXIT
                CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位
                …                           ;TMR1 中断处理程序
INT_EXIT：
POP：                                       ;中断服务程序出口，还原 ACC 及 STATUS
                …
                RETI
```

7.8.2 TMR2 中断

TMR2 溢出时，如果 EN_T2 置“1”，则 F_T2 会被 置“1”，系统就会响应 TMR2 的中断；若 EN_T2=0，则 F_T2 不会被置“1”，系统不会响应 TMR2 中断。尤其需要注意多种中断下的情形。

例：TMR2 中断应用程序

```
                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：
                …
                LDIA    032H
                LD      T2DATA,A            ;设置 T2 溢出时间
                LDIA    B’00110000’
                LD      T2CON,A             ;设置 TMR2 时钟
                CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
                SETB    INT_EN,EN_T2        ;使能 TMR2 中断
                SETB    T2CON,0             ;TMR2 计时器开始工作
                SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN
                …
MAIN：
                …
                …
                JP      MAIN
INT_SERVICE：
PUSH：                                      ;中断服务程序入口，保存 ACC 及 STATUS
                …
                …
                SNZB    INT_EN,EN_T2        ;判断是否使能 TMR2 中断
                JP      INT_EXIT
                SNZB    INT_FLAG,F_T2       ;检查有无 TMR2 中断请求标志位
                JP      INT_EXIT
                CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
                …
                …                           ;TMR2 中断处理程序
INT_EXIT：
POP：                                       ;中断服务程序出口，还原 ACC 及 STATUS
                …
                RETI
```

7.9 ADC 中断

当 ADC 转换完成后，如果 EN_AD 使能，那么系统就会响应 ADC 中断，F_AD 被置 1。若 EN_AD=0，则 ADC 转换完成后，F_AD 不会被置“1”，系统不会进入 ADC 中断。用户应注意多种中断下的处理。

例：AD 中断设置程序

```
        CLRB    INT_EN,EN_AD
        LDIA    B'10101011'
        LD      P0CL,A              ;P0.0 口设置为 AD 输入口
        LDIA    03H
        LD      SYS_GEN,A           ;开启 AD 模块，及使能中断
        LDIA    B'00000110'
        LD      ADCON,A             ;选择 AN0 通道
        NOP
        CLRB    INT_FLAG,F_AD       ;清零 AD 中断请求标志
        SETB    INT_EN,EN_AD        ;使能 AD 中断
        NOP
        SETB    ADCON,CONV
        NOP
        NOP
        NOP
        CLRB    ADCON,CONV          ;AD 中断开始信号
```

例：AD 中断处理程序

```
                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：
                ...
MAIN：
                ...
                JP      MAIN
INT_SERVICE：
PUSH：                                  ;中断服务程序入口，保存 ACC 及 STATUS
                ...
                ...
                SNZB    INT_EN,EN_AD    ;判断是否使能 ADC 中断
                JP      INT_EXIT
                SNZB    INT_FLAG,F_AD   ;检查有无 ADC 中断请求标志位
                JP      INT_EXIT
                CLRB    INT_FLAG,F_AD   ;清零 ADC 中断请求标志位
                ...
                ...                     ;ADC 中断处理程序
INT_EXIT：
POP：                                   ;中断服务程序出口，还原 ACC 及 STATUS
                ...
                RETI
```

7.10 中断的优先级，及多中断嵌套

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。中断请求标志 INT_FLAG 由中断事件触发，当 F_XX 处于有效值“1”时，系统并不一定会响应该中断。各中断触发事件如下表所示：

中断	有效触发
F_IOCA	由IOCA决定
F_EXT0	由INT_EXT0决定
F_EXT1	由INT_EXT1决定
F_EXT2	由INT_EXT2决定
F_T2	TMR2溢出
F_T1	TMR1溢出
F_AD	AD转换结束

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。

例：多个中断处理程序

```
                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：                                 ;用户程序起始地址
                …
MAIN：
                …                       ;用户主程序
                JP      MAIN
INT_SERVICE：
PUSH：                                  ;中断服务程序入口，保存 ACC 及 STATUS
                …
EXT0CH：
                SNZB    INT_EN,EN_EXT0  ;判断是否使能 EXT0 中断
                JP      EXT1CH
                SZB     INT_FLAG,F_EXT0 ;检查有无 EXT0 中断请求标志位
                JP      INT_EXT0        ;EXT0 中断处理程序
EXT1CH：
                SNZB    INT_EN,EN_EXT1  ;判断是否使能 EXT1 中断
                JP      TMR1CH
                SZB     INT_FLAG,F_EXT1 ;检查有无 EXT1 中断请求标志位
                JP      INT_EXT1        ;EXT1 中断处理程序
```

例：多个中断处理程序(续)

```
TMR1CH：
        SNZB    INT_EN,EN_T1        ;判断是否使能 TMR1 中断
        JP      TMR2CH
        SZB     INT_FLAG,F_T1       ;检查有无 TMR1 中断请求标志位
        JP      INT_TMR1            ;TMR1 中断处理程序
TMR2CH：
        SNZB    INT_EN,EN_T2        ;判断是否使能 TMR2 中断
        JP      ADCCH
        SZB     INT_FLAG,F_T2       ;检查有无 TMR2 中断请求标志位
        JP      INT_TMR2            ;TMR2 中断处理程序
ADCCH：
        SNZB    INT_EN,EN_AD        ;判断是否使能 ADC 中断
        JP      INT_EXIT
        SNZB    INT_FLAG,F_AD       ;检查有无 ADC 中断请求标志位
        JP      INT_EXIT
INT_ADC：
        CLRB    INT_FLAG,F_AD
        …                           ;AD 中断处理程序
        JP      INT_EXIT
INT_TMR2：
        CLRB    INT_FLAG,F_TMR2
        …                           ;TMR2 中断处理程序
        JP      INT_EXIT
INT_TMR1：
        CLRB    INT_FLAG,F_TMR1
        …                           ;TMR1 中断处理程序
        JP      INT_EXIT
INT_EXT1：
        CLRB    INT_FLAG,F_EXT1
        …                           ;EXT1 中断处理程序
        JP      INT_EXIT
INT_EXT0：
        CLRB    INT_FLAG,F_EXT0
        …                           ;EXT0 中断处理程序
        JP      INT_EXIT
INT_EXIT：
    POP：                           ;中断服务程序出口，还原 ACC 及 STATUS
        …
        RETI
```

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；
- 可用程序进行读写操作；
- 可选择定时器或计数器工作方式；
- 8 位可编程控制寄存器（OPTION_REG）；
- 外部时钟边沿可选择。

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，TMR0 没有中断。
- 当 T0CS=1 时以计数器方式工作，TIMER0 模块的计数器将对加到 RTCC 口的脉冲进行计数。是上升沿还是下降沿有效则由位 T0SE 选择，T0SE=0 时选择上升沿有效，T0SE=1 时选择下降沿有效。

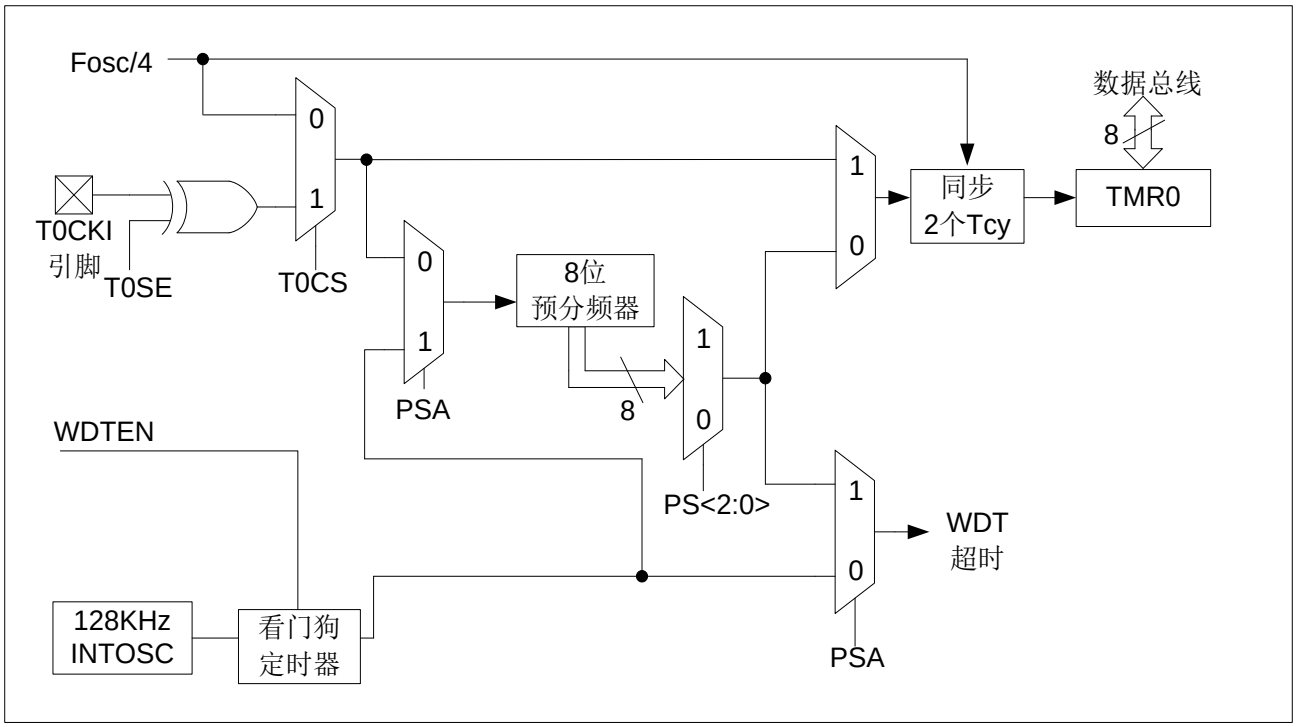


图 8-1：TMR0/WDT 结构图

注：T0SE、T0CS、PSA、PS<2:0>为 OPTION_REG 寄存器中的位。

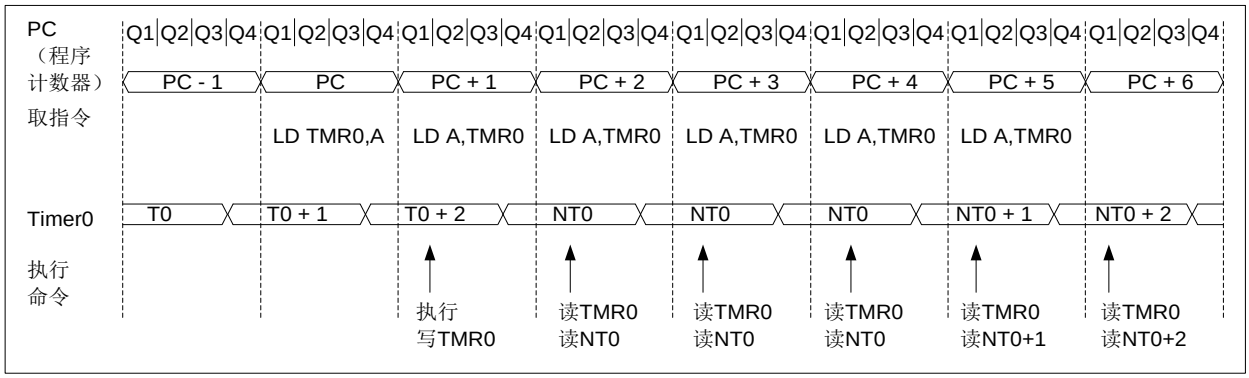


图 8-2：TMR0 时序图（内部时钟/无预分频器）

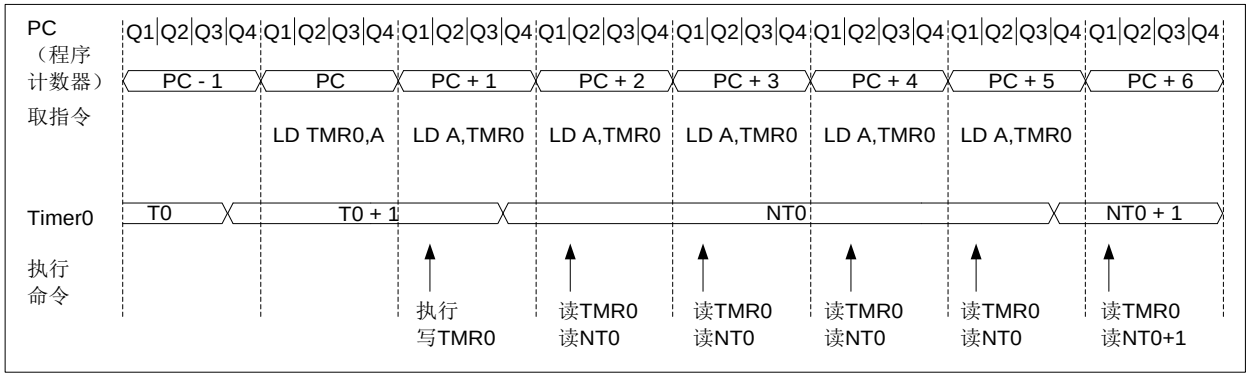


图 8-3：TMR0 时序图（内部时钟/预分频器 1:2）

8.2 与 TMR0 相关寄存器

有两个寄存器与 TMR0 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 6 位可读写寄存器，用户可改变 OPTION_REG 的值，来改变 TMR0 的工作模式等。请参照 2.6 关于预分频寄存器（OPTION_REG）的应用。

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

0AH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OPTION_REG	---	---	T0CS	T0SE	PSA	PS2	PS1	PS0
读写			W	W	W	W	W	W
复位值	X	X	1	1	1	1	1	1

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 RTCC 与内部相位时钟同步。因此要求 RTCC 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 RTCC 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。

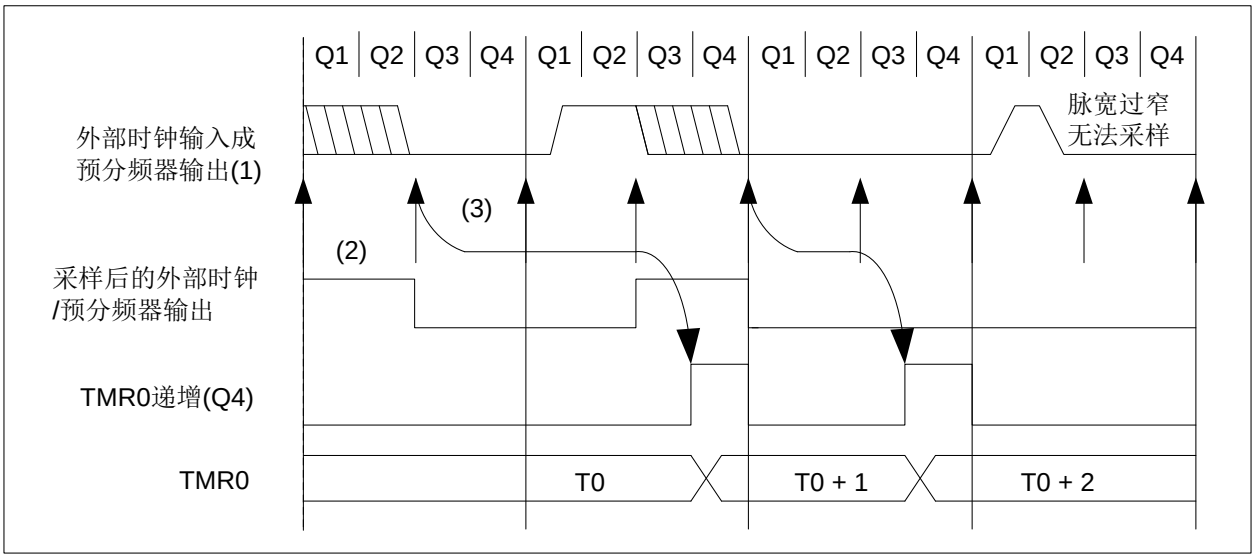


图 8-4：TMR0 与外部时钟时序

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 的第 3 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 1 分频。当 OPTION_REG 的第 3 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 PS3～PS0	TMR0 的输入时钟 T0CLK	Fcpu=4MHz÷4	
		最大溢出间隔时间	TMR0 递增时间
1xxx	Fcpu/1	256μs	1μs
0000	Fcpu/2	512μs	2μs
0001	Fcpu/4	1024μs	4μs
0010	Fcpu/8	2048μs	8μs
0011	Fcpu/16	4096μs	16μs
0100	Fcpu/32	8192μs	32μs
0101	Fcpu/64	16384μs	64μs
0110	Fcpu/128	32768μs	128μs
0111	Fcpu/256	65536μs	256μs

8.4.2 TMR0 操作流程

TMR0 的操作流程为：

- 设置 TMR0 工作模式及分频比；
- 设置 TMR0 初值。

例：TMR0 定时设置程序

```
LDIA    00H
LD      OPTION_REG,A    ;设置 TMR0 时钟=Fcpu/2
CLR     TMR0            ;初始化 TMR0
```

注：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 T0CLKS 时钟不递增，用户要自己输入校正值来避开这个问题。

9. 定时计数器 TMR1

9.1 TMR1 概述

TMR1 由如下功能组成：

- 可选择时钟频率；
- TMR1 控制寄存器（TMR1C）；
- 外部时钟为边沿可选择；
- 8 位定时器/计数器；
- 中断在 0FFH 到 00H 时溢出；
- 两种不同工作模式。

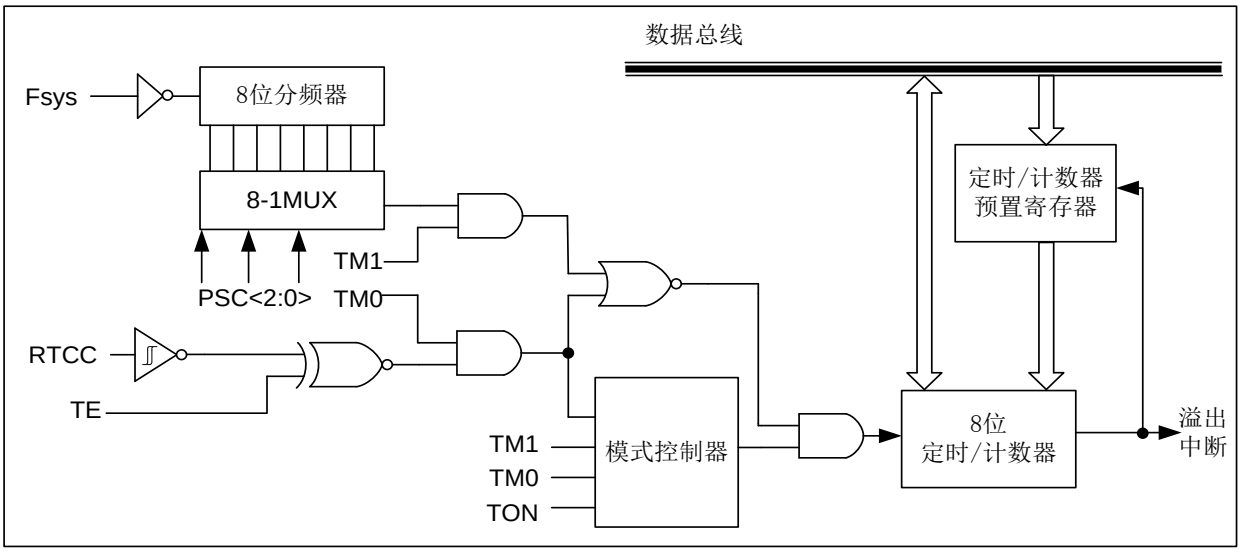


图 9-1：TMR1 结构图

TMR1 有两个与定时/计数器有关的寄存器，TMR1 和 TMR1C。TMR1 寄存器有两个物理空间；写入 TMR1 会将初始值装入到定时/计数器的预置寄存器中，而读 TMR1 则会取得定时/计数器的内容。TMR1C 是定时/计数器控制寄存器，它可以定义定时/计数器的工作模式。

TM0、TM1 用来定义定时/计数器的工作模式。外部事件计数模式是用来记录外部事件的，其时钟来源为外部 RTCC 引脚输入。

定时器模式是一个常用模式，其时钟来源为内部时钟。无论是定时模式还是外部事件计数模式，一旦开始计数，定时/计数器会从寄存器当前值向上计到 0FFH。一旦发生溢出，定时/计数器会从预置寄存器中重新加载初值，并开始计数；同时置位中断请求标志 F_T1，位 F_T1 须用软件清零。

当计数器溢出时，会从定时/计数器的预置寄存器中重新加载初值，而中断的处理方式与其它两种模式一样。要启动计数器，只要置位 TON，TON 只能由指令来清除。定时/计数器溢出可以做为唤醒信号。不管是什么模式，只要写 0 到 EN_T1 位即可禁止定时/计数器中断服务。

在定时/计数器停止计数时，写数据到定时/计数器的预置寄存器中，同时会将该数据写入到定时/计数器。但如果在定时/计数器工作时数据只能写入到预置寄存器中，直到发生溢出时才会将数据从预置寄存器加载到定时/计数器寄存器。读取定时/计数器时，计数会被停止，以避免发生错误；计数停止会导致计数错误，程序员必须注意到这一点。

9.2 TMR1 相关寄存器

TMR1 数据寄存器 TMR1

16H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR1								
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

TMR1 控制寄存器 TMR1C

17H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR1C	TM1	TM0	---	TON	TE	PSC2	PSC1	PSC0
读写	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~Bit6 TM1~TM0: TMR1 工作模式选择位；
00: 未用；
01: 外部事件计数器；
10: 内部定时器；
11: 未用。
Bit4 TON: 工作使能位；
0: 禁止；
1: 使能。
Bit3 TE: 计数模式边缘选择；
0: 上升沿计数；
1: 下降沿计数。
Bit1~Bit0 PSC2～PSC0: 分频比选择；
000: 1:1；
001: 1:2；
010: 1:4；
011: 1:8；
100: 1:16；
101: 1:32；
110: 1:64；
111: 1:128。

9.3 TMR1 的时间常数

9.3.1 TMR1 基本时间参数

TMR1 在设置不同分频时候的基本事件参数如下表：

TMR1C PSC2～PSC0	TMR1 的输入时钟 T1CLK	Fcpu=4MHz÷4	
		最大溢出间隔时间	TMR1 递增时间
000	Fcpu	256μs	1μs
001	Fcpu/2	512μs	2μs
010	Fcpu/4	1024μs	4μs
011	Fcpu/8	2048μs	8μs
100	Fcpu/16	4096μs	16μs
101	Fcpu/32	8192μs	32μs
110	Fcpu/64	16384μs	64μs
111	Fcpu/128	32768μs	128μs

9.4 TMR1 的应用

9.4.1 TMR1 作定时器使用

TMR1 作内部定时器时，可以产生一个定时中断，设置 T1 定时器的操作流程如下：

- 禁止 TMR1 定时器；
- 禁止 TMR1 中断并清除 TMR1 中断标志位；
- 设置 TMR1 为定时器模式及分频比；
- 设置 TMR1 初值；
- 开启 TMR1 中断。

例：TMR1 定时设置程序

```
        CLRB    TMR1C,TON           ;禁止 TMR1 定时器工作
        CLRB    INT_EN,EN_T1        ;禁止 TMR1 中断
        CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位
        LDIA    B'10000000'
        LD      TMR1C,A             ;设置 TMR1 为定时器模式，分频比 1:1
        LDIA    06H
        LD      TMR1,A              ;设置 TMR1 初值
        CLRB    INT_FLAG,F_T1       ;清零 TMR1 中断请求标志位
        SETB    INT_EN,EN_T1        ;使能 TMR1 中断
        SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN
        SETB    TMR1C,TON           ;使能 TMR1 定时器
```

9.4.2 TMR1 作计数器使用

通过设置 TMR1C 的 TM1、TM0 位可以使 TMR1 工作在外部事件计数模式，当 TMR1 被设置为外部事件计数模式时，在外部计数器模式下 TMR1 会根据 RTCC 口的上升沿或则下降沿递增，在此种模式下分频比选项是不起作用的。

设置 T1 计数器的操作流程如下：

- 禁止 TMR1 计数器；
- 禁止 TMR1 中断并清除 TMR1 中断标志位；
- 设置 TMR1 为计数器模式；
- 设置 TMR1 初值；
- 开启 TMR1 中断。

例：TMR1 计数器设置程序（上升沿递增模式）

```
CLRB    TMR1C,TON       ;禁止 TMR1 工作
CLRB    INT_EN,EN_T1    ;禁止 TMR1 中断
CLRB    INT_FLAG,F_T1   ;清零 TMR1 中断请求标志位
LDIA    B'01000000'
LD      TMR1C,A         ;设置 TMR1 为计数器模式
LDIA    06H
LD      TMR1,A          ;设置 TMR1 初值
CLRB    INT_FLAG,F_T1   ;清零 TMR1 中断请求标志位
SETB    INT_EN,EN_T1    ;使能 TMR1 中断
SETB    SYS_GEN,INT_GEN ;使能 INT_GEN
SETB    TMR1C,TON       ;使能 TMR1 定时器
```

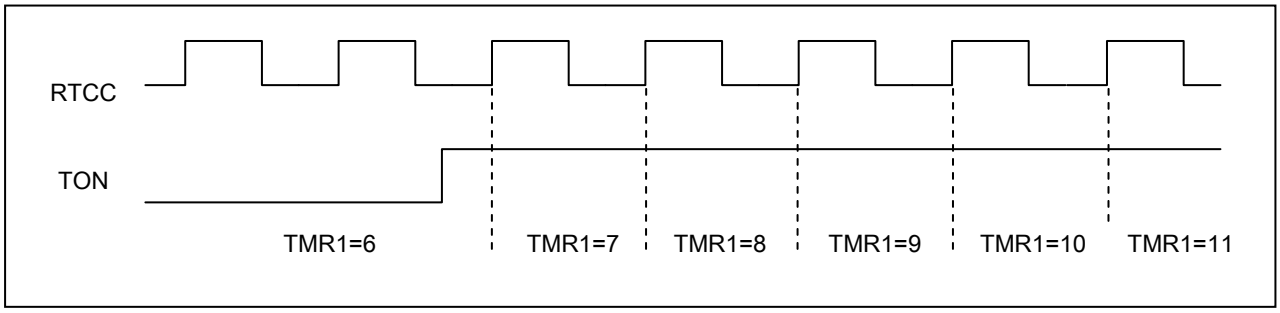


图 9-2：TMR1 作外部计数器时工作时序

TMR1 在作外部计数器时工作流程如下：

- RTCC 口有方波信号输入；
- TMR1 在方波信号的上升沿或者下降沿递增；
- 当 TMR1 从 FF 加到 00 时产生中断请求信号 F_T1；
- 若中断使能，则回应 TMR1 中断。

关于 TMR1 作计数器的下降沿递增模式，与上升沿一样，只是一个在下降沿递增、一个在上升沿递增。这里不再赘述。

10. 定时计数器 TMR2

10.1 TMR2 概述

TMR2 由如下功能组成：

- 选择时钟频率；
- 8Bit 计数器（T2CNT）、8Bit 比较器、8Bit 数据寄存器（T2DATA）和 T2DATA 缓冲器；
- TMR2 控制寄存器（T2CON）。

T2CON（Bit4 和 Bit5）用来选择 TMR2 的输入时钟频率。定时器 2 中断的使能位和标志位由 INT_EN.2 和 INT_FLAG.2 控制。在定时模式下，当 TMR2 计数器的值和 T2DATA 值相等时，将产生一个 TMR2 的匹配信号来清除 T2 计数器的值和重载 T2DATA 的比较值，假如 T2 中断是使能的，那么也会同时产生中断请求信号。如果 TMR2 中断禁止（INT_EN.2=0），匹配的信号不产生匹配的中断请求。时钟分配器不是 TMR2 的组成部分，且分配的时钟和定时器中断使能信号是同步的。所以，在第一个匹配时间间隔里是矛盾的。

注：中断请求标志位 F_T2 必须由软件清除。

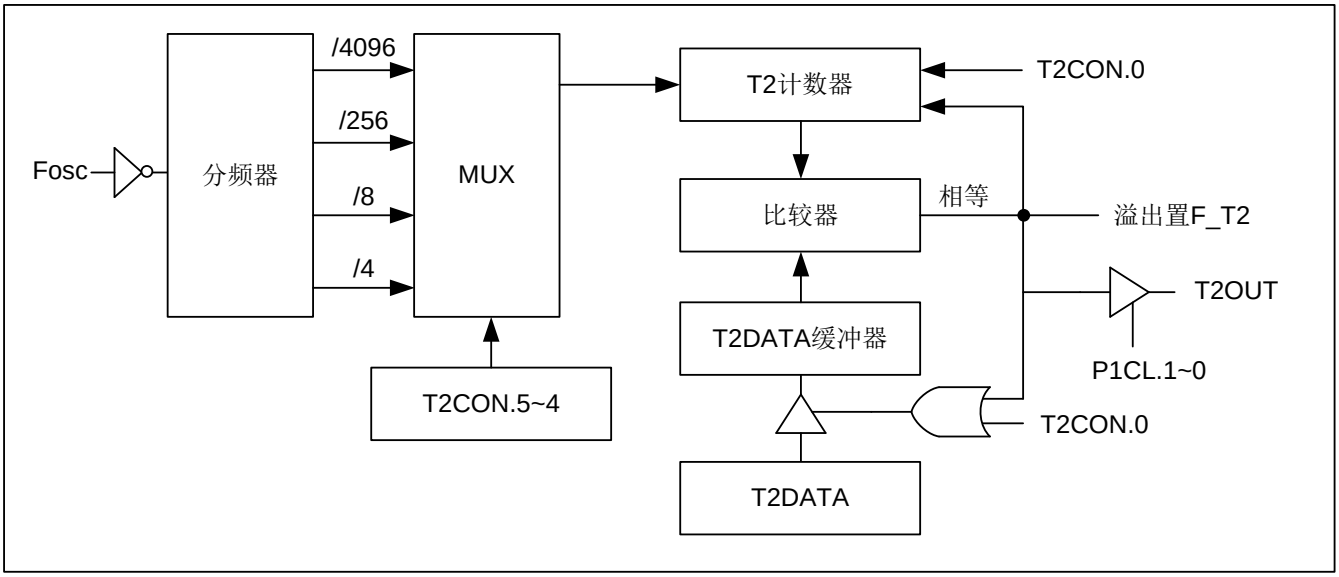


图 10-1：TMR2 结构框图

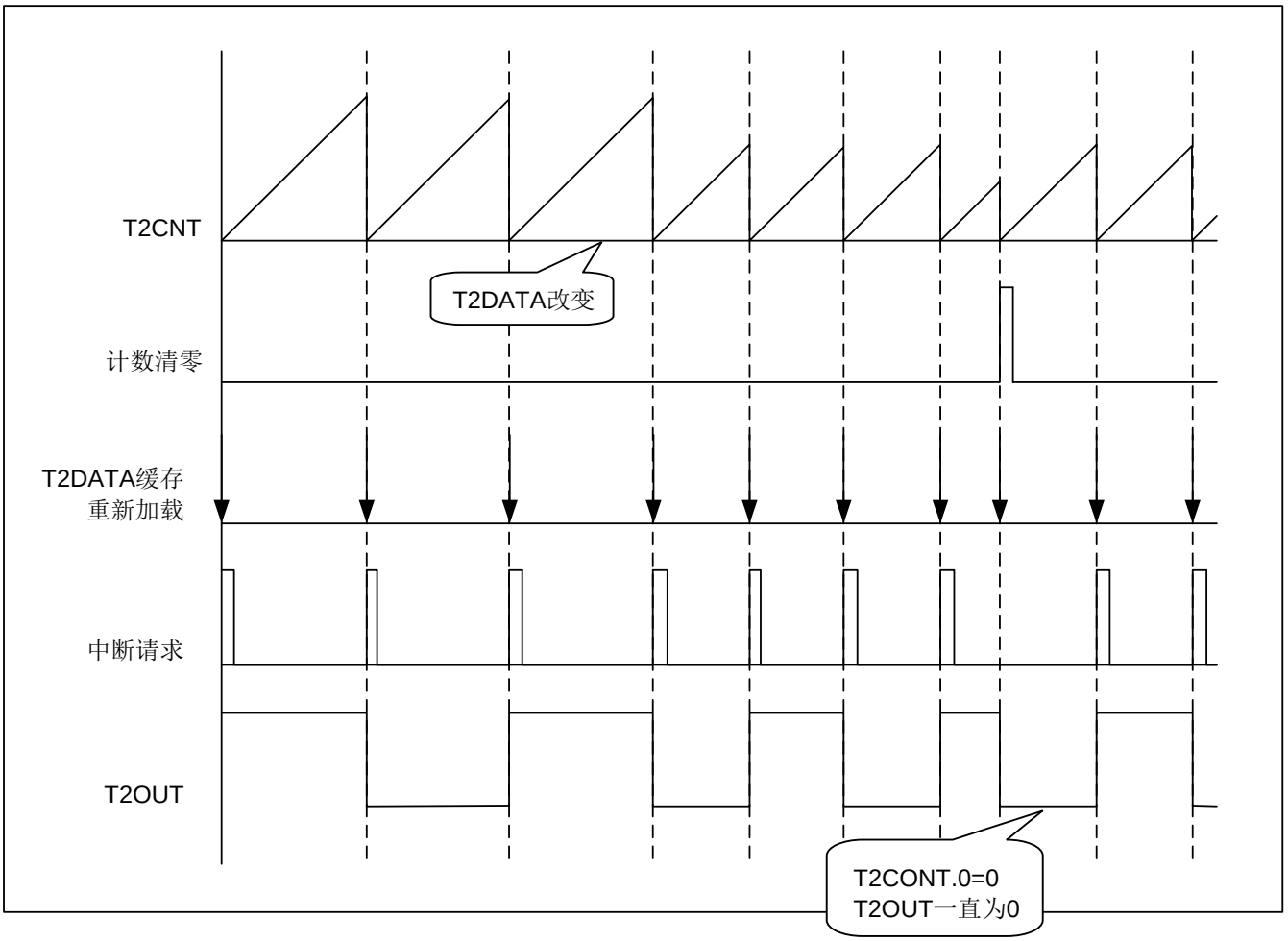


图 10-2：TMR2 时序图

10.2 TMR2 相关的寄存器

有三个寄存器与 TMR2 相关，分别是数据存储器 T2DATA、计数器 T2CNT 和控制寄存器 T2CON。

TMR2 计数器 T2CNT

18H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2CNT								
R/W	R	R	R	R	R	R	R	R
复位值	0	0	0	0	0	0	0	0

TMR2 控制寄存器 T2CON

19H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2CON	---	---	T2C1	T2C0	---	---	---	T2_CLR
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~Bit6 未用

Bit5~Bit4 T2C1、T2C0: 时钟选择位；
00: Fosc/4096；
01: Fosc/256；
10: Fosc/8；
11: Fosc/4。

Bit3~Bit1 未用

Bit0 T2_CLR: 工作使能位；
0: 禁止 TMR2 时钟，T2CNT 清零；
1: 使能 TMR2 时钟，T2CNT 从“0”开始计数。

TMR2 数据存储器 T2DATA

1AH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2DATA								
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

10.3 TMR2 的时间常数

10.3.1 TMR2 基本时间参数

在 8M 的时钟下，TMR2 基本事件参数如下表：

T2CON、T2C1、T2C0	T2CNT 计数时钟	Fosc=8MHz	
		最大溢出间隔时间	最小溢出间隔时间
00	Fosc /4096	131072μs	512μs
01	Fosc /256	8192μs	32μs
10	Fosc /8	256μs	1μs
11	Fosc /4	128μs	0.5μs

10.3.2 T2DATA 初值计算方法

T2DATA 初值计算方法如下：

T2DATA 初值 =T2 溢出时间×时钟频率÷分频比－1

例：Fosc=8MHz、分频比 1:4、T2 溢出时间 100μs 时 T2DATA 的值。

T2DATA 初值 =T2 溢出时间×时钟频率÷分频比－1
=100μs×8MHz÷4－1
=199

10.4 TMR2 应用

TMR2 作计数器时，可以产生一个定时中断，设置 T2 计数器的操作流程如下：

- 禁止 TMR2 定时器；
- 禁止 TMR2 中断并清除 TMR2 中断标志位；
- 设置 TMR2 分频比；
- 开启 TMR2 中断；
- 开始 TMR2 计数。

例：TMR2 定时设置程序

```
        CLRB    T2CON,T2_CLR        ;清零 T2CNT
        CLRB    INT_EN,EN_T2        ;禁止 TMR2 中断
        CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
        LDIA    063H
        LD      T2DATA,A            ;设置 TMR2 目标值
        LDIA    B'00110000'
        LD      T2CON,A             ;设置 TMR2 分频比 1:4
        CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
        SETB    INT_EN,EN_T2        ;使能 TMR2 中断
        SETB    SYS_GEN,INT_GEN     ;使能 INT_GEN
        SETB    T2CON,T2_CLR        ;T2CNT 开始计数
```

10.5 T2OUT 输出

当 T2 溢出时，I/O 口（P1.0）可以与其匹配输出，由 I/O 口控制寄存器 P1CL 控制，当 P1.0 设置为 T2OUT 口时，无论此时往 P1.0 I/O 寄存器写“1”，还是“0”；P1.0 都会匹配 T2 溢出输出。

10.5.1 T2OUT 的周期

T2OUT 的输出周期为 T2 溢出中断的两倍。计算公式如下：

T2OUT 周期 =T2 溢出时间×2

10.5.2 T2OUT 基本时间参数

在 8M 时钟下，T2OUT 基本时间参数如下表：

T2CON T2C1、T2C0	T2CNT 计数时钟	Fosc=8MHz	
		T2OUT 最大输出周期	T2OUT 最小输出周期
00	Fosc /4096	262144μs	1024μs
01	Fosc /256	16384μs	64μs
10	Fosc /8	512μs	2μs
11	Fosc /4	256μs	1μs

10.5.3 T2OUT 应用

在 P1.0 口输出 T2 溢出匹配信号的操作流程如下：

- 禁止 TMR2 定时器；
- 禁止 TMR2 中断并清除 TMR2 中断标志位；
- 设置 TMR2 分频比；
- 设置 P1.0 口为 T2OUT 口；
- 开始 TMR2 计数。

例：T2OUT 设置程序

```
        CLRB    T2CON,T2_CLR        ;清零 T2CNT
        CLRB    INT_EN,EN_T2        ;禁止 TMR2 中断
        CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
        LDIA    063H
        LD      T2DATA,A            ;设置 TMR2 目标值
        LDIA    B'00110000'
        LD      T2CON,A             ;设置 TMR2 分频比 1:4
        CLRB    INT_FLAG,F_T2       ;清零 TMR2 中断请求标志位
        LDIA    B'10101011'
        LD      P1CL,A              ;P1.0 设置为 T2OUT 口
        SETB    T2CON,T2_CLR        ;T2CNT 开始计数
```

11. 模数转换（ADC）

11.1 ADC 概述

模数转换器（Analog-to-digital Converter，ADC）可以将模拟输入信号转换为 12 位二进制代码表示。该模块使用模拟信道通过多路开关连接到一个采样保持电路，采样保持电路的输出与转换器的输入相连接，转换器通过逐次逼近法产生 12 位的二进制结果，并将转换结果存入 ADDATAH 和 ADDATAL 寄存器中，并产生中断请求信号 F_AD。

ADC 由如下功能组成：

- 15 通道输入多任务器；
- 12-Bit 连续近似值寄存器；
- 输出寄存器组成（ADDATAH、ADDATAL）；
- 控制寄存器（ADCON）；
- 转换结束产生中断。

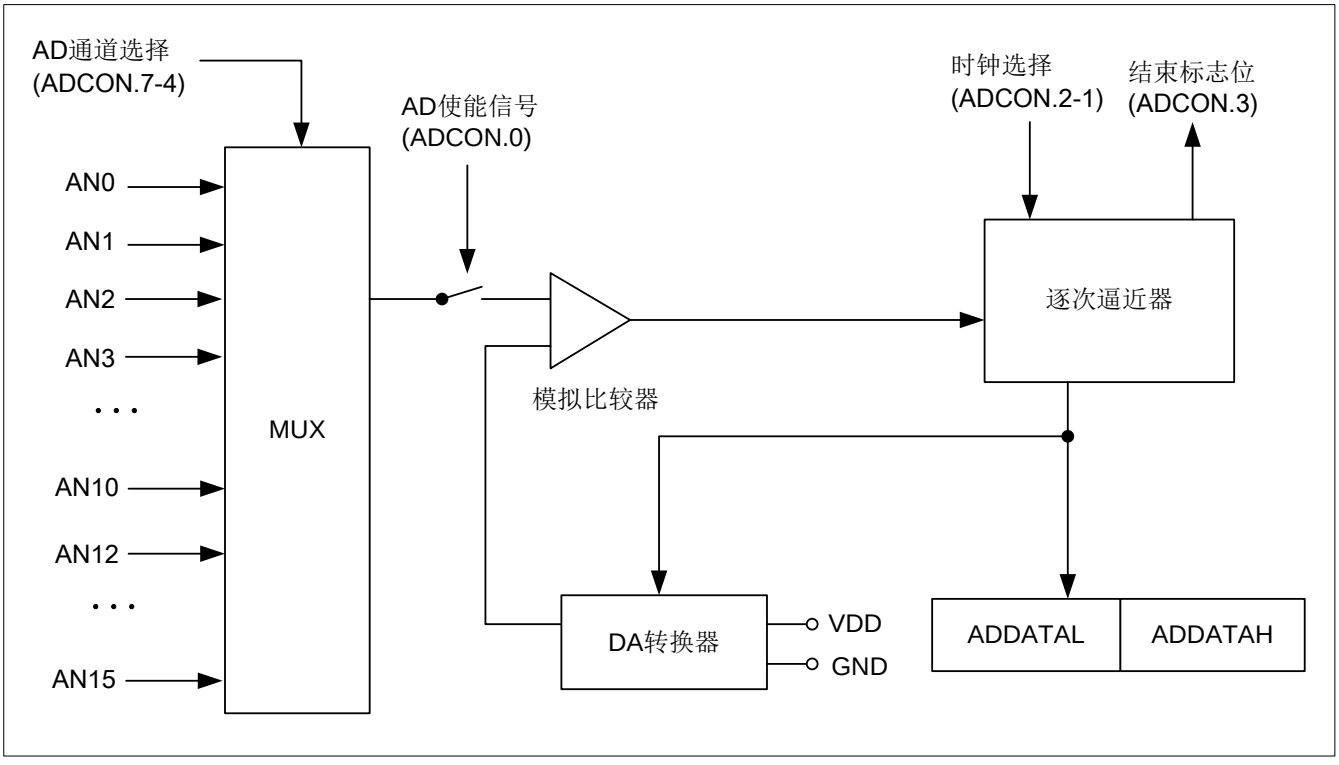


图 11-1：ADC 结构框图

11.2 与 ADC 相关寄存器

有三个寄存器与 ADC 相关，分别是数据存储器高 8 位 ADDATAH、数据存储器低 4 位 ADDATAL 和 AD 控制寄存器 ADCON。

AD 控制寄存器 ADCON

15H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADCON	通道选择位				EOC	时钟选择		CONV
R/W	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W
复位值	0	0	0	0	1	0	0	0

Bit7~Bit4 AD 通道选择位。

0000: 选择 AN0 通道；
0001: 选择 AN1 通道；
0010: 选择 AN2 通道；
0011: 选择 AN3 通道；
0100: 选择 AN4 通道；
0101: 选择 AN5 通道；
0110: 选择 AN6 通道；
0111: 选择 AN7 通道；
1000: 选择 AN8 通道；
1001: 选择 AN9 通道；
1010: 选择 AN10 通道；
1011: 选择 AN11 通道（内部基准源：0.6V/1.2V）；
1100: 选择 AN12 通道；
1101: 选择 AN13 通道；
1110: 未用；
1111: 选择 AN15 通道。

Bit3 EOC: 结束标志；
0: 转换未完成；
1: 转换结束。

Bit2~Bit1 AD 时钟选择位（选择较慢的时钟频率有利于提高 AD 精度）。

00: Fosc/16；
01: Fosc/8；
10: Fosc/4；
11: Fosc/2（在 8M 或者更高的振荡频率下不建议使用）。

Bit0 CONV: AD 转换开始信号；
“0”→“1”→“0”: AD 转换开始。

AD 数据寄存器高 8 位 ADDATAH

14H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADDATAH								
R/W	R							
复位值	X							

AD 数据寄存器低 4 位 ADDATAL

1BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADDATAL								
R/W	R				----	----	----	----
复位值	X				----	----	----	----

ADDATAH 和 ADDATAL 是 A/D 转换结果寄存器，是只读寄存器，当完成 A/D 转换后可从 ADDATAH 和 ADDATAL 读取 A/D 转换结果。

ADCON 是 A/D 转换控制寄存器，用来定义 A/D 转换时钟，模拟输入通道选择，A/D 转换开始控制和完成标志。如果要进行 A/D 转换要先定义好 I/O 口的设置选择，转换的仿真通道,然后给 CONV 控制位一个上升沿信号和一个下降沿信号：0→1→0，完成 A/D 转换后 EOC 位会被置 1，若 A/D 中断被允许，则会产生 A/D 转换中断。当 CONV 标志由 0 置为 1 时 EOC 也会清零。

注：A/D 中断请求标志 F_AD 须由软件清除，为了确保 A/D 转换顺利完成，CONV 位应保持为 0 直到 EOC 位变为 1（A/D 转换完成信号）。

11.3 内部电压基准

CMS89F11x 内部集成了一个 0.6V/1.2V 左右的电压基准，其个体间误差范围大约为 5%。使用方法为：开启 AD 转换并将 AD 通道选择内部电压基准通道（第 11 通道），测试其 AD 值。该值表示的是当前电源电压下，AD 输入口为 0.6V/1.2V 左右时，芯片所测试得到的 AD 值。

内部基准 AD 使用流程：

1. A/D 转换使能，即 ADC_EN(SYS_GEN.1)=1；
2. 设置 ADCON 为 B’10110000’，等待几十条指令时间；
3. 置 CONV(ADCON[0])=1，触发 AD 转换；
4. 等待至少 1 个 CLK 的延时后，清 CONV(ADCON[0])=0；
5. 等待 AD 转换结束，判断 EOC(ADCON[3])是否为“1”，等于“1”代表转换结束；
6. 读取 AD 数据 ADDATAH[14H]和 ADDATAL[1BH]；
7. 根据测试到的 AD 值，通过比例关系换算成当前电压。

例：内部基准 AD 测试程序

```
AD_START：
        SETB    SYS_GEN,ADC_EN      ;开启ADC使能
        LDIA    B‘10110000’
        LD      ADCON,A             ;选择电压基准通道，AD时钟为Fosc/16
        NOP
        SETB    ADCON,CONV
        NOP
        NOP
        NOP
        CLRB    ADCON,CONV          ;开始AD转换
WAIT：
        SNZB    ADCON,EOC           ;等待AD转换结束
        JP      WAIT
        LD      A,ADDATAH           ;保存AD转换结果
        LD      R01,A
        LD      A,ADDATAL
        LD      R02,A
        CLRB    SYS_GEN,ADC_EN      ;关断ADC模块
        JP      XXXX                ;AD转换结束转到其他程序
```

检测完成后，可通过比例换算，得到当前电源电压，转换公式为：

$$1.2/VDD=ADDATAH/256$$

例：当前测试得到的 AD 值为：ADDATAH=83，ADDATAL=0，则该电源电压 VDD 为：

$$1.2/VDD=83/256$$

$$VDD\approx 3.7V$$

11.4 ADC 应用

11.4.1 用查询模式做 AD 转换流程

1. A/D 转换使能，即 ADC_EN(SYS_GEN.1)=1；
2. 设置对应 I/O 口为 AD 口；
3. 等待几十条指令时间；
4. 设置 AD 转换时钟 ADCON[2:1]，采样通道 ADCON[7:4]；
5. 置 CONV(ADCON[0])=1，触发 AD 转换；
6. 等待至少 1 个 CLK 的延时后，清 CONV(ADCON[0])=0；
7. 等待 AD 转换结束，判断 EOC(ADCON[3])是否为“1”，等于“1”代表转换结束；
8. 读取 AD 数据 ADDATAH、ADDATAL；
9. 如果需要采样另外通道则转到流程 1 或 2；否则结束 AD 转换，转到流程 10；
10. AD 转换结束，如需省电可以关闭 AD 模块，设置 ADC_EN(SYS_GEN.1)=0。

例：查询模式的 AD(AN0)转换程序

```
AD_SET_MODE：
                SETB    SYS_GEN,ADC_EN      ;开启 ADC 使能
                LDIA    B’XXXXXX11’
                LD      P0CL,A              ;设置 P0.0 口为 AD 口
                CALL    DELY_TIME           ;延时几十个指令周期
                LDIA    B’00000000’
                LD      ADCON,A             ;选择 AN0 通道，AD 时钟为 Fosc/16
                NOP
                SETB    ADCON,CONV          ;CONV 下降沿开始 AD 转换
                NOP
                NOP
                NOP
                CLRB    ADCON,CONV          ;开始 AD 转换
        AD_WAIT：
                SNZB    ADCON,EOC           ;等待 AD 转换结束
                JP      AD_WAIT
                LD      A,ADDATAH
                LD      R01,A               ;保存高 8 位结果用户自定义 RAM 里面
                LD      A,ADDATAL
                LD      R02,A               ;保存低 4 位结果用户自定义 RAM 里面
                CLRB    SYS_GEN,ADC_EN      ;关断 ADC 模块
                JP      XXXX                ;AD 转换结束转到其它程序
```

11.4.2 AD 中断模式流程

1. A/D 转换使能，即 ADC_EN(SYS_GEN.1)=1；
2. 设置对应 I/O 口为 AD 口；
3. 等待几十条指令时间；
4. 设置 AD 转换时钟 ADCON[2:1]，采样通道 ADCON[7:4]；
5. 置 CONV(ADCON[0])=1，触发 AD 转换；
6. 开启 AD 中断 EN_AD(INT_EN.5)=1；
7. 等待至少 1 个 CLK 的延时后，清 CONV(ADCON[0])=0=0；
8. 等待 AD 中断产生；
9. 读取 AD 数据 ADDATAH、ADDATAL；
10. 如果需要采样另外通道则转到流程 1 或 2；否则结束 AD 转换，转到下一步；
11. AD 转换结束，如需省电可以关闭 AD 模块，设置 ADC_EN(SYS_GEN.1)=0。

例：用 AD 中断做 AD 转换

```
                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：
                …                       ;用户初始化程序
MAIN：
                …
                CALL    ADC_SUB         ;调用 AD 设置程序
                …
                JP      MAIN
INT_SERVICE：
                CALL    PUSH            ;中断服务程序入口，保存 ACC 及 STATUS
ADCCH：
                SNZB    INT_EN,EN_AD    ;判断是否使能 ADC 中断
                JP      INT_EXIT
                SNZB    INT_FLAG,F_AD   ;检查有无 ADC 中断请求标志位
                JP      INT_EXIT
INT_ADC：                               ;AD 中断处理程序
                CLRB    INT_FLAG,F_AD   ;清零 AD 中断请求标志
                CLRB    FLAG,AD_EN      ;清零设置 AD 标志
                LD      A,ADDATAH
                LD      R01,A           ;保存 AD 转换高 8 位值
                LD      A,ADDATAL
```

例：用 AD 中断做 AD 转换（续）

```
                LD      R02,A                   ;保存 AD 转换低 2 位值

INT_EXIT：
                CALL    POP                     ;中断服务程序出口，还原 ACC 及 STATUS
                RETI

ADC_SUB：       SZB     FLAG,AD_EN              ;AD 检测子程序入口
                RET
                SETB    SYS_GEN,ADC_EN          ;开启 ADC 使能
                LDIA    B'XXXX11XX'
                LD      P0CL,A                  ;设置 P0.1 口为 AD 口
                CALL    DELY_TIME               ;延时几十个指令周期
                LDIA    B'00010110'
                LD      ADCON,A                 ;选择 AN1 通道，AD 时钟为 Fosc/2
                NOP
                SETB    ADCON,CONV
                NOP
                CLRB    INT_FLAG,F_AD
                SETB    INT_EN,EN_AD
                CLRB    ADCON,CONV              ;开始 AD 转换
                SETB    FLAG,AD_EN
                RET
```

12. LCD 驱动模块

芯片内置 LCD 驱动模块。CMS89F11x 只能驱动 1/2Bias 的 LCD，并且必须由软件来实现驱动。

12.1 LCD 功能使能

将 LCDCON 的第 0 位 LCDEN 置 1，允许 LCD 驱动功能。

12.2 LCD 相关设置

CMS89F11x 由 4 个 I/O 口内置上下拉电阻，可用来做 1/2Bias 的 LCD 驱动，其控制流程为：

- 将 LCDCON 第 0 位 LCDEN 置 1；
- 设置 LCD_ISLE[1:0]位选择 COM 口输出电流；
- 设置相应的 COMEN 位为“1”，允许该管脚为 1/2Bias 的 COM 口；
- 当其中一个 COM 口需要点亮，需要把该管脚做输出，并将相应的 COMEN 位置“0”；其它未点亮 COM 口需要把管脚做输入，并将相应的 COMEN 位置“1”。

相关寄存器如下：

LCD 控制寄存器 LCDCON

28H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
LCDCON	COM3EN	COM2EN	COM1EN	COM0EN	---	LCD_ISLE[1:0]		LCDEN
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~Bit4	COMxEN:	COM 口功能设置；
	0:	对应 COMx 口为普通 I/O 口；
	1:	对应 COMx 口为 LCD 功能的 COM 口。
Bit3	未用	
Bit2~Bit1	LCD_ISLE[1:0]:	LCD 输出电流选择位
	00=	25uA@5V；
	01=	50uA@5V；
	10=	100uA@5V；
	11=	200uA@5V。
Bit0	LCDEN:	LCD 模块使能；
	0:	禁止 LCD 模块；
	1:	使能 LCD 模块。

13. 内置比较器

13.1 内置比较器概述

CMS89F11x 系列芯片内置了两路高精度比较器，COMP0 及 COMP1，芯片比较器的正端输入可以设置为接外部输入，也可设置为 GND、1/10 VDD、2/10 VDD、3/10 VDD、4/10 VDD、5/10 VDD 等参考电压。用户可通过设置 COMPCON 寄存器来开启、关断比较器功能及设置比较器正端标准电压。

比较器工作原理：当“＋”＞“－”时输出高，当“＋”＜“－”时输出低。用户可以读取 P0.0 及 P1.0 的状态可以确定比较器的输出状态，此时程序读取的不是 P0.0 或者 P1.0 的外部电平状态，而是读取其内部寄存器的值。

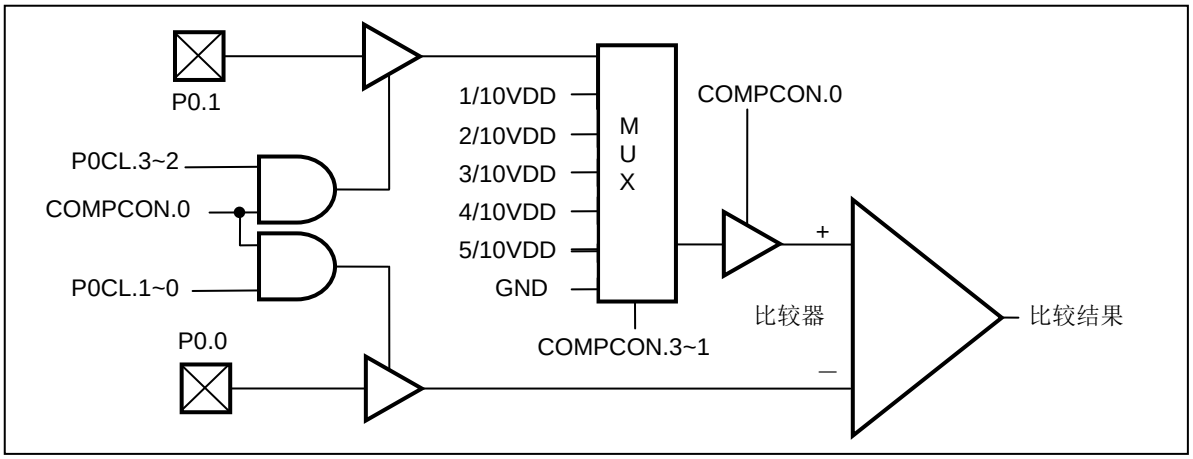


图 13-1：COMP0 结构框图

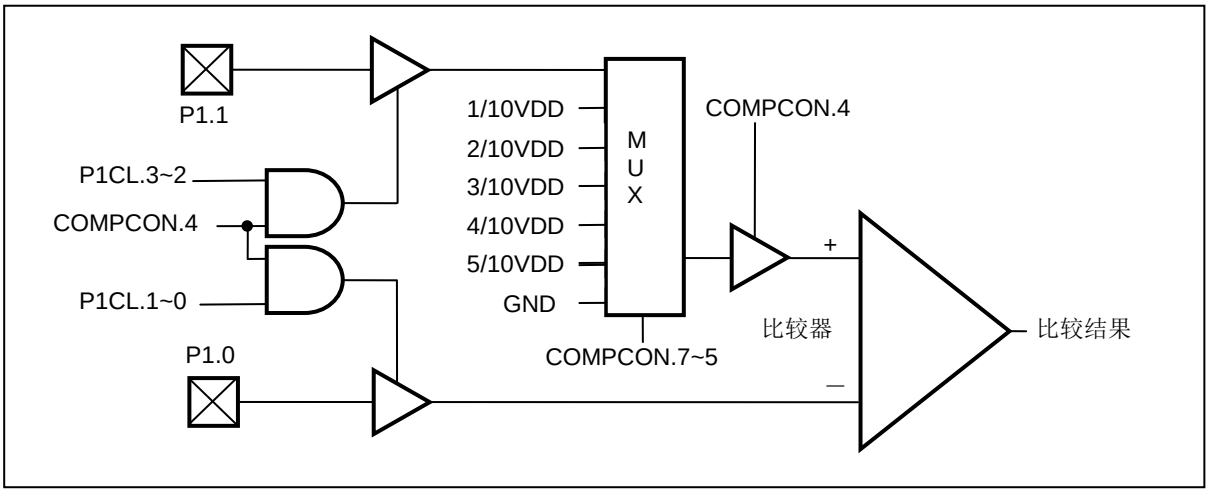


图 13-2：COMP1 结构框图

13.2 与比较器相关的寄存器

比较器控制寄存器 COMPCON

20H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
COMPCON	BPLUS2	BPLUS1	BPLUS0	COMP1_EN	APLUS2	APLUS1	APLUS0	COMP0_EN
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~Bit5 BPLUS2~BPLUS0: 比较器1正端电压选择。
000: 比较器1正端电压为GND、P1.1为普通I/O口；
001: 比较器1正端电压为1/10VDD、P1.1为普通I/O口；
010: 比较器1正端电压为2/10VDD、P1.1为普通I/O口；
011: 比较器1正端电压为3/10VDD、P1.1为普通I/O口；
100: 比较器1正端电压为4/10VDD、P1.1为普通I/O口；
101: 比较器1正端电压为5/10VDD、P1.1为普通I/O口；
110: 比较器1正端接P1.1口；
111: 比较器1正端接P1.1口。

Bit4 COMP1_EN: 比较器1使能控制。
0: 比较器1关闭，P1.0为普通I/O口；
1: 比较器1开启，P1.0为比较器1“－”端；

Bit3~Bit1 APLUS2~APLUS0: 比较器0正端电压选择。
000: 比较器0正端电压为GND、P0.1为普通I/O口；
001: 比较器0正端电压为1/10VDD、P0.1为普通I/O口；
010: 比较器0正端电压为2/10VDD、P0.1为普通I/O口；
011: 比较器0正端电压为3/10VDD、P0.1为普通I/O口；
100: 比较器0正端电压为4/10VDD、P0.1为普通I/O口；
101: 比较器0正端电压为5/10VDD、P0.1为普通I/O口；
110: 比较器0正端接P0.1口；
111: 比较器0正端接P0.1口。

Bit0 COMP0_EN: 比较器0使能控制。
0: 比较器0关闭，P0.0为普通I/O口；
1: 比较器0开启，P0.0为比较器0“－”端。

13.3 比较器 0 应用

输入输出端口说明：

- “－”端：对应于 IO 口 P0.0；
- “＋”端：对应于 IO 口 P0.1 或内部选择比较电压；
- “输出”端：对应于[05H].0 即 P0.0。

芯片由 COMPCON 控制打开或关闭比较器 0，COMPCON.0=1 打开比较器，COMPCON.0=0 关闭比较器。当打开比较器时，只要 COMPCON.0=1，P0.0 自动设为比较器输入口“－”端，否则为普通 IO 口；当 COMPCON.[3:2:1]=110 或 111，P0.1 自动设为比较器“＋”端输入口，COMPCON.[3:2:1]为其余数，比较器“＋”端连接到内部选择比较电压。需要判断比较器输出时，直接读取 P0.0 的状态即为输出结果。

比较器使用程序流程：

- 设置 P0.0、P0.1 口状态；
- 设置 COMPCON.0=1，（如果需要正端输入则设置 COMPCON.[3:2:1]=110 或 111）；
- 等待比较稳定；
- 读取比较结果 P0.0；
- 设置 COMPCON.0=0，关闭比较器 0。

例：COMP0 的应用程序

```
            LDIA    B'XXXX0101'
            LD      P0CL,A          ;设置P0.0、P0.1为比较器输入
            LDIA    B'XXXX1111'
            LD      COMPCON,A       ;设置并开启比较器
            CALL    DELY_TIME       ;延时等待比较结果稳定
            SZB     P0,0            ;判断比较结果
            JP      P_LT_N
N_LT_P:
            ...
            JP      EXIT
P_LT_N:
            ...
            JP      EXIT
EXIT:
            CLRB    COMPCON,0       ;关闭比较器
```

13.4 比较器 1 应用

输入输出端口说明：

- “－”端：对应于 IO 口 P1.0；
- “＋”端：对应于 IO 口 P1.1 或内部选择比较电压；
- “输出”端：对应于[06H].0 即 P1.0。

芯片由 COMPCON 控制打开或关闭比较器 1，COMPCON.4=1 打开比较器，COMPCON.4=0 关闭比较器。当打开比较器时，只要 COMPCON.4=1，P1.0 自动设为比较器输入口“－”端，否则为普通 IO 口；当 COMPCON.[7:6:5]=110 或 111，P1.1 自动设为比较器“＋”端输入口，COMPCON.[7:6:5]为其余数，比较器“＋”端连接到内部选择比较电压。需要判断比较器输出时，直接读取 P1.0 的状态即为输出结果。

比较器 1 程序流程：

- 设置 COMPCON.4=1，（如果需要正端输入则设置 COMPCON.[7:6:5]=110 或 111）；
- 等待比较器稳定；
- 读取比较器结果 P1.0；
- 设置 COMPCON.4=0 关闭比较器 1。

例：COMP1 的应用程序

```
        LDIA    B'1111XXXX'
        LD      COMPCON,A       ;设置并开启比较器
        CALL    DELY_TIME       ;延时等待比较结果稳定
        SZB     P1,0            ;判断比较结果
        JP      P_LT_N
N_LT_P:
        ...
        JP      EXIT
P_LT_N:
        ...
        JP      EXIT
EXIT:
        CLRB    COMPCON,4       ;关闭比较器
```

注：当用户使用 CMS89F11x 系列芯片的内部比较器时需要注意以下几点：

1. P0.0、P0.1、P1.0、P1.1 口电压不能超过（VDD－1）V；若有可能请把电压限制在 1/2VDD 以下，以提高比较器精度。
2. 当比较器从开启到输出稳定的比较结果需要一定的等待时间，具体时间与外围元件有关，用户可根据不同的应用环境设置不同的等待时间。

14. 数据 EEPROM 控制

14.1 数据 EEPROM 概述

数据 EEPROM 在正常工作状态下是可读写的。这些存储器并不直接映射到寄存器文件空间，而是通过特殊功能寄存器对其进行间接寻址。共有 5 个寄存器用于访问这些存储器：

- EECON1
- EECON2
- EEDAT
- EEDATH
- EEADR

当与数据存储器模块接口时，EEDAT 和 EEDATH 寄存器形成一个双字节字用于保存要读/写的 14 位数据，而 EEADR 寄存器存放被访问的 EEDAT 单元的地址。该系列中的器件具有 32 字的数据 EEPROM，地址范围为 00H 到 1FH。

EEPROM 数据存储器允许字节读写。字节写操作可自动擦除目标单元并写入新数据（在写入前擦除）。

写入时间由片上定时器控制。写入和擦除电压是由片上电荷泵产生的，此电荷泵额定工作在器件的电压范围内，用于进行字节或字操作。

当器件受代码保护时，CPU 仍可继续读写数据 EEPROM。代码保护时，器件编程器将不再能访问数据。

14.2 相关寄存器

14.2.1 EEADR 寄存器

EEADR 寄存器能寻址最大 32 字数据 EEPROM。

14.2.2 EECON1 和 EECON2 寄存器

EECON1 是访问 EE 存储器的控制寄存器。

控制位 EEPGD 需要置 1 后才能操作数据存储器。该位被清零时，和复位时一样，任何后续操作都是无效的。

控制位 RD 和 WR 分别启动读和写。用软件只能将这些位置 1 而无法清零。在读或写操作完成后，由硬件将它们清零。由于无法用软件将 WR 位清零，从而可避免意外地过早终止写操作。

- 当 WREN 置 1 时，允许对数据 EEPROM 执行写操作。上电时，WREN 位被清零。当正常的写入操作被 LVR 复位或 WDT 超时复位中断时，WRERR 位会置 1。在这些情况下，复位后用户可以检查 WRERR 位并重写相应的单元。
- 当写操作完成时 PIR2 寄存器中的中断标志位 EEIF 被置 1。此标志位必须用软件清零。

EECON2 不是物理寄存器。读 EECON2 得到的是全 0。

EECON2 寄存器仅在执行数据 EEPROM 写序列时使用。

EEPROM 数据寄存器 EEDAT

25H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0
读写	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 EEDAT<7:0>: 对数据EEPROM进行读取或写入的数据的低8位。

EEPROM 数据寄存器 EEDATH

26H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEDATH	---	---	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0
读写	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	0	0	0	0	0

Bit5~Bit0 EEDATH<5:0>: 对数据EEPROM进行读取或写入数据的高6位。

EEPROM 地址寄存器 EEADR

27H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEADR	---	---	---	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0
读写	---	---	---	R/W	R/W	R/W	R/W	R/W
复位值	---	---	---	0	0	0	0	0

Bit7~Bit5 未用

Bit4~Bit0 EEADR<4:0>: 指定EEPROM读取/写入操作的地址。

EEPROM 控制寄存器 EECON1

2AH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EECON1	EEPGD	---	---	---	WRERR	WREN	WR	RD
读写	R/W	---	---	---	R/W	R/W	R/W	R/W
复位值	0	---	---	---	X	0	0	0

Bit7 EEPGD: 数据EEPROM使能位；

1= 允许操作数据EEPROM；

0= 禁止操作数据EEPROM。

Bit6~Bit4 未用，读为0

Bit3 WRERR: EEPROM错误标志位；

1= 写操作过早终止（正常工作期间的任何WDT复位或欠压复位）；

0= 写操作完成。

Bit2 WREN: EEPROM写使能位；

1= 允许写周期；

0= 禁止写入数据EEPROM。

Bit1 WR: 写控制位；

1= 启动写周期（写操作一旦完成由硬件清零该位，用软件只能将WR位置1，但不能清零）；

0= 数据EEPROM写周期完成。

Bit0 RD: 读控制位；

1= 启动数据EEPROM读操作（由硬件清零RD，用软件只能将RD位置1，但不能清零）；

0= 不启动数据EEPROM读操作。

EEPROM 控制寄存器 EECON2

2BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EECON2	---							
读写	W							

EECON2 不是物理寄存器。读 EECON2 得到的是全 0。

EECON2 寄存器仅在执行数据 EEPROM 写序列时使用。

14.3 读数据 EEPROM 存储器

要读取数据存储器单元，用户必须将地址写入 EEADR 寄存器，将 EECON1 寄存器的 EEPGD 控制位置 1，然后将控制位 RD 置 1。一旦设置好读控制位，数据存储器控制器将使用第二个指令周期来读数据。这会导致紧随“SETB EECON1,RD”指令的第二条指令被忽略。在紧接着的下一个周期 EEDAT 寄存器中就有数据了。EEDAT 将保存此值直至下一次用户向该单元读取或写入数据时为止。

注：程序存储器读操作后的两条指令必须为 NOP。这可阻止用户在 RD 位置 1 后的下一条指令执行双周期指令。

读数据 EEPROM

```
LD      A,EE_ADD        ;将要读取的地址放入 EEADR 寄存器
LD      EEADR,A
SETB    EECON1,EEPGD    ;使能 EEPROM
SETB    EECON1,RD       ;使能读信号
NOP                     ;这里读取数据，必须加 NOP 指令
NOP
LD      A,EEDAT         ;读取数据到 ACC
```

14.4 写数据 EEPROM 存储器

要写 EEPROM 数据存储单元，用户应首先将该单元的地址写入 EEADR 寄存器并将数据写入 EEDATA 寄存器。然后用户必须按特定顺序开始写入每个字节。

如果没有完全按照下面的指令顺序（即首先将 55h 写入 EECON2，随后将 AAh 写入 EECON2，最后将 WR 位置 1）写每个字节，将不会启动写操作。在该代码段中应禁止中断。

此外，必须将 EECON1 中的 WREN 位置 1 以使能写操作。这种机制可防止由于代码执行错误（异常）（即程序跑飞）导致误写 EEPROM。在不更新 EEPROM 时，用户应该始终保持 WREN 位清零。WREN 位不能被硬件清零。

一个写过程启动后，将 WREN 位清零将不会影响此写周期。除非 WREN 位置 1，否则 WR 位将无法置 1。写周期完成时，WR 位由硬件清零。

执行了 SETB EECON1,WR 指令之后，处理器需要 2 个指令周期以设置擦除/写操作。用户必须在将 WR 位置 1 的指令后放置两条 NOP 指令。执行完写操作指令后，处理器会使内部操作暂停 2.5ms 时间。因为时钟和外设仍继续工作，所以这并不是休眠模式。写周期结束后，处理器将从 EECON1 写指令后的第三条指令恢复工作。

写数据 EEPROM 存储器

```
        LD      A,ADDRL         ;写地址
        LD      EEADR,A
        LD      A,DATAL         ;写数据
        LD      EEDAT,A
        LD      A,DATAH
        LD      EEDATH,A
        SETB    EECON1,EEPGD    ;允许操作 EEPROM
        SETB    EECON1,WREN     ;使能写信号
        CLRB    SYS_GEN,INT_GEN ;关闭中断
        SZB     SYS_GEN,INT_GEN ;确认中断关闭
        JP      $-2

        LDIA    055H            ;给 EECON2 寄存器写 55H 和 0AAH
        LD      EECON2,A
        LDIA    0AAH
        LD      EECON2,A
        SETB    EECON1,WR       ;开始写 EEPROM
        NOP                     ;写缓冲需要延时
        NOP

        CLRB    EECON1,WREN
        SETB    SYS_GEN,INT_GEN ;打开中断
        SZB     EECON1,WR       ;判断写操作是否完成
        JP      $-1
        CLRB    EECON1,WREN     ;写结束，关闭写使能位
```

14.5 数据 EEPROM 操作注意事项

14.5.1 写校验

根据具体的应用，好的编程习惯一般要求将写入数据 EEPROM 的值对照期望值进行校验。

14.5.2 避免误写的保护

有些情况下，用户可能不希望向数据 EEPROM 存储器写入数据。为防止误写 EEPROM，芯片内嵌了各种保护机制。上电时清零 WREN 位。而且，上电延时定时器（延迟时间为 18ms）会防止对 EEPROM 执行写操作。

写操作的启动序列以及 WREN 位将共同防止在以下情况下发生误写操作：

- 欠压；
- 电源毛刺；
- 软件故障。

15. 8 位 PWM（PWM0）

15.1 8 位 PWM 概述

PWM8 由如下功能组成：

- 选择时钟频率。
- 8-Bit 计数器（PWM8CON）、6-Bit 比较器、8-Bit 数据存储器（PWM6DATA）和 6-Bit 数据缓冲器。
- 2-Bit 扩展逻辑、2-Bit 扩展寄存器和数据缓冲器。
- 两种模式选择（6+2）/（7+1）。

CMS89F11x 的 8 位脉冲宽度调制器有两种工作模式，由 PWM8CON.3 位控制，PWM8CON.3=1 选择“7+1”模式，PWM8CON.3=0 选择“6+2”模式。PWM8CON.2=1 为 6 位溢出时加载，即改变 PWM8 的数据存储器后会在下一个波形输出时改变占空比，PWM8CON.2=0 时为 8 位溢出时加载，即改变 PWM8 的数据存储器后会在下一个周期时改变占空比（也就是说当选择“6+2”模式时，PWM8 将为 4 个波形一个周期，此时不论你在哪一个波形输出时改变 PWM8 的数据存储器都将在下一个周期才生效）。

所谓“6+2”模式就是指 PWM8DATA 的高 6 位（PWMDATA.7～2）用于控制 PWM8 的调制周期及在调制周期内占空比，低 2 位（PWMDATA.1～0）用于控制扩展周期。

所谓“7+1”模式就是指 PWM8DATA 的高 7 位（PWMDATA.7～1）用于控制 PWM8 的调制周期及在调制周期内占空比，低 1（PWMDATA.0）位用于控制扩展周期。

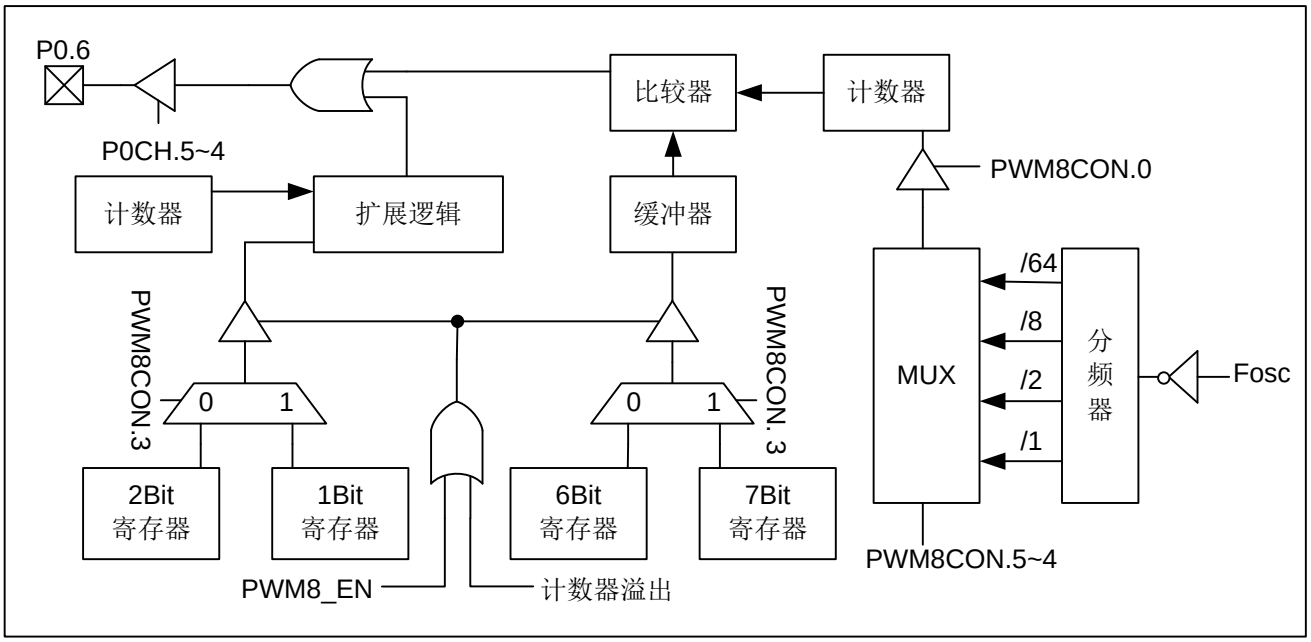


图 15-1：8 位 PWM 框图

15.2 与 8 位 PWM 相关寄存器

有两个寄存器与 PWM8 有关，PWM8DATA(数据存储器)、PWM8CON(控制寄存器)。

1CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWM8DATA								
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

1DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWM8CON	CLO占空比选择		PWM8时钟选择		模式选择	加载选择	---	PWM8_EN
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-Bit6 CLO 占空比选择。

00: CLO 占空比 50%；

01: CLO 占空比 25%；

10: CLO 占空比 50%；

11: CLO 占空比 75%。

Bit5~Bit4 PWM8 时钟选择。

00: PWM8 时钟为 Fosc/64；

01: PWM8 时钟为 Fosc/8；

10: PWM8 时钟为 Fosc/2；

11: PWM8 时钟为 Fosc/1。

Bit3 模式选择位。

0: “6+2”模式；

1: “7+1”模式。

Bit2 加载选择位。

0: 数据缓冲在 8 位溢出时加载；

1: 数据缓冲在 6 位溢出时加载。

Bit0 PWM8_EN: PWM8 使能控制位；

0: PWM8 停止工作；

1: PWM8 允许工作。

15.3 8 位 PWM 的周期

15.3.1 8 位 PWM 调制周期

8 位 PWM 调制周期由系统主频（Fosc）、PWM8 分频比、PWM8 模式决定，计算公式如下：

PWM8 调制周期 $=2^N \times$ PWM8 分频比 $\div F_{OSC}$

注：N=6 或者 7 由 PWM8 模式决定。

例：Fosc=8MHz、分频比 1：2、“6+2”模式，时 PWM 调制周期

PWM8 调制周期 $=2^N \times$ PWM8 分频比 $\div F_{OSC}$

$=2^6 \times 2 \div (8 \times 10^6)$s

=16μs

Fosc=8MHz 时 PWM8 的调制周期表

PWM8CON Bit5、Bit4	PWM8 时钟	Fosc=8MHz	
		“6+2”模式	“7+1”模式
00	Fosc /64	512μs	1024μs
01	Fosc /8	64μs	64μs
10	Fosc /2	16μs	32μs
11	Fosc /1	8μs	16μs

15.3.2 8 位 PWM 输出周期

8 位 PWM 输出周期由 PWM8 模式确定，当选择“6+2”模式时，4 个调制周期为一个输出周期，当选择“7+1”模式时 2 个调制周期为一个输出周期。

15.4 8 位 PWM 占空比算法

8 位 PWM 输出的占空比与 PWM8DATA 的数值相关，从整体上来说，其占空比近似等于 PWMDATA÷256。不同的模式 PWM 占空比算法不同，我们不妨把 PWM8DATA 的值分为两个部分：基本输出周期控制部分（DC）和额外输出周期控制部分（AC）。

15.4.1 6+2 模式 PWM 占空比

当选择 6+2 模式的时候，PWMDATA 的高 6 位为基本输出周期，低 2 位为额外输出周期。根据额外输出周期的不同，在基本输出周期的基础上分 4 个周期的补偿。

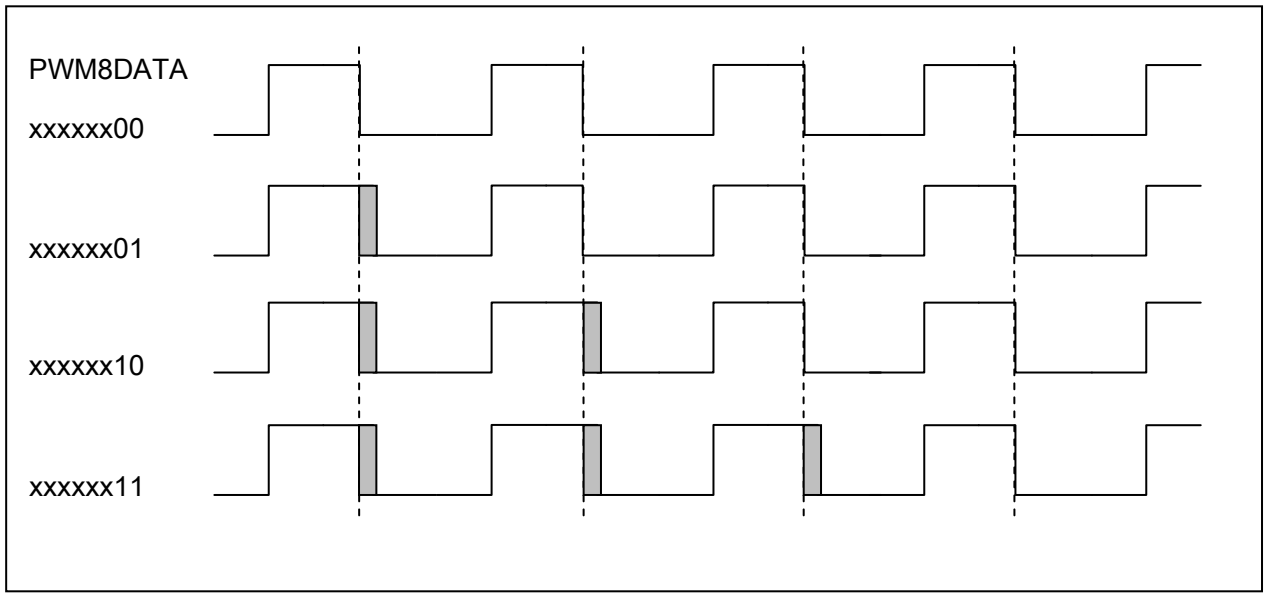


图 15-2：6+2 模式 8 位 PWM 输出示意图

如上图所示，当基本输出周期为 DC，那么不同的额外输出周期时，PWM8 的连续 4 个实际周期分别为：

额外输出周期	周期 1	周期 2	周期 3	周期 4
00	DC/64	DC/64	DC/64	DC/64
01	DC/64+1	DC/64	DC/64	DC/64
10	DC/64+1	DC/64+1	DC/64	DC/64
11	DC/64+1	DC/64+1	DC/64+1	DC/64

15.4.2 7+1 模式 PWM 占空比

当选择 7+1 模式的时候，PWMDATA 的高 7 位为基本输出周期，最低位为额外输出周期。根据额外输出周期的不同，在基本输出周期的基础上分 2 个周期的补偿。

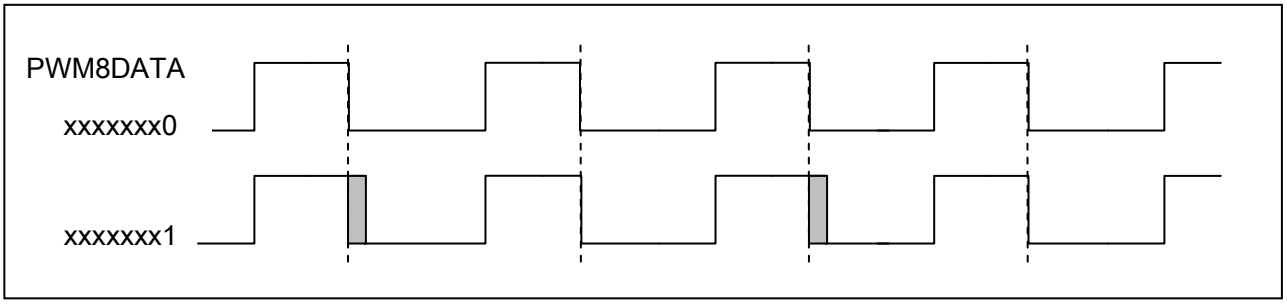


图 15-3：7+1 模式 8 位 PWM 输出示意图

如上图所示，当基本输出周期为 DC，那么不同的额外输出周期时，PWM8 的连续 2 个实际周期分别为：

额外输出周期	周期 1	周期 2
0	DC/64	DC/64
1	DC/64+1	DC/64

15.5 8 位 PWM 应用

PWM8 的应用设置需的操作流程如下：

- 设置 PWM8 工作模式及时钟；
- 设置 PWM8DATA；
- P0.6 设置为 PWM8 输出口；
- PWM8 开始工作。

例：PWM8 的设置程序

```
        LDIA    B'XX110000'
        LD      PWM8CON,A       ;FPWM=FOSC、"6+2"模式
        LDIA    B'10000001'
        LD      PWM8DATA,A      ;设置 PWM8 占空比
        LDIA    B'XX01XXXX'
        LD      P0CH,A          ;P0.6 设置为 PWM 输出口
        SETB    PWM8CON,0       ;开启 PWM8
```

16. 10 位 PWM（PWM1）

16.1 10 位 PWM 概述

PWM10 由如下功能组成：

- 选择时钟频率；
- 10-Bit 计数器（PWM10CON）、8-Bit 比较器、8-Bit 数据存储器（PWM10DATA）和 8-Bit 数据缓冲器；
- 2-Bit 扩展逻辑，2-Bit 扩展寄存器和数据缓冲器；
- 控制寄存器（PWM10CON）。

计数器的高 8 位和 PWM10 数据存储器相比较（PWM10DATA）来确定 PWM10 的工作频率。为了更高的精度，计数器的低 2 位可被用来作为扩展周期。

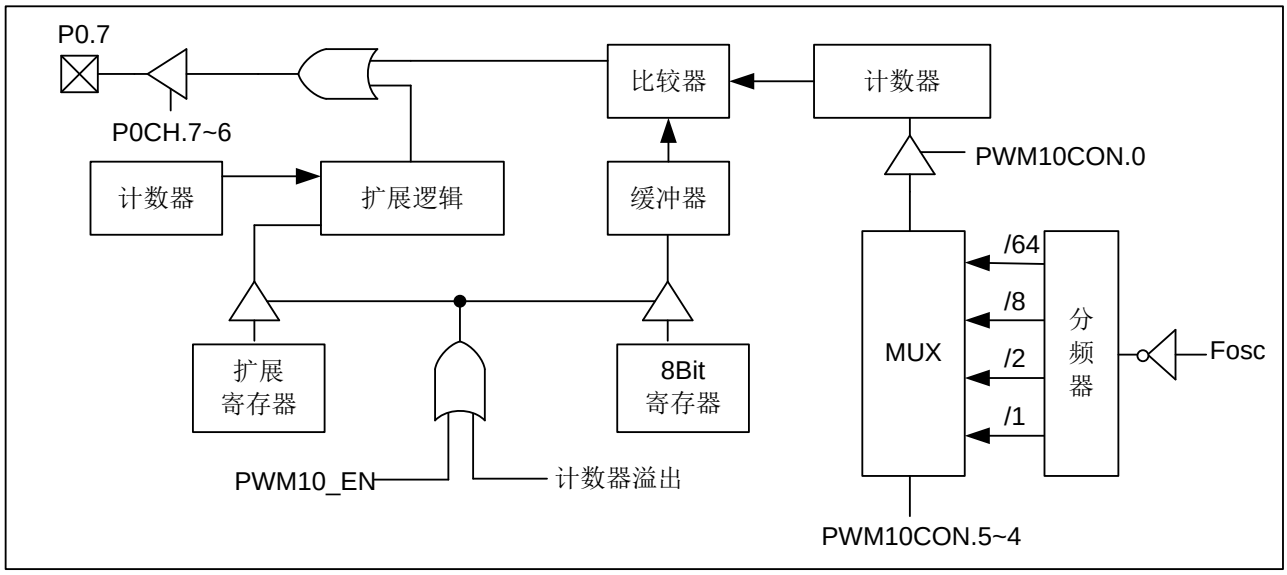


图 16-1：PWM10 结构框图

当计数器的高 8 位和相关的数据寄存器（PWM10DATA）匹配时，PWM 输出低电平。如果 PWM10DATA 寄存器的值不是 0，计数器的高 8 位溢出将使 PWM 输出高电平。这样的话，写进相关寄存器的值就决定了模块的基本工作周期。

计数器的低 2 位值和 2 位扩展数据寄存器（PWM10CON.7-6）的扩展设置比较。计数器值的低 2 位被用来扩展 PWM 的输出工作周期。扩展值是一个在特殊周期的额外时钟周期（如下表）。

PWM10CON.7-6	扩展周期
00	None
01	1
10	1,2
11	1,2,3

16.2 与 10 位 PWM 相关寄存器

有两个寄存器与 PWM10 有关，PWM10 数据寄存器 PWM10DATA、PWM10 控制寄存器 PWM10CON。

1FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWM10DATA								
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

1EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWM10CON	扩展周期选择		PWM10时钟选择		---	加载选择	---	PWM10_EN
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-Bit6 扩展周期选择。
- 00: 无扩展周期；
- 01: 扩展周期为 1；
- 10: 扩展周期为 1、2；
- 11: 扩展周期为 1、2、3。

Bit5~Bit4 PWM10 时钟选择。
- 00: PWM10 时钟为 Fosc/64；
- 01: PWM10 时钟为 Fosc/8；
- 10: PWM10 时钟为 Fosc/2；
- 11: PWM10 时钟为 Fosc/1。

Bit3 未用

Bit2 加载选择位。
- 0: 数据缓冲在 10 位溢出时加载；
- 1: 数据缓冲在 8 位溢出时加载。

Bit1 未用

Bit0 PWM10_EN: PWM10 使能控制位；
- 0: PWM10 停止工作；
- 1: PWM10 允许工作。

16.3 10 位 PWM 调制周期

16.3.1 10 位 PWM 调制周期

10 位 PWM 调制周期由系统主频（Fosc）、PWM10 分频比，计算公式如下：

PWM10 调制周期 $=2^8\times$PWM10 分频比$\div F_{OSC}$

例：Fosc=8MHz，分频比 1:1，时 PWM 调制周期。

PWM10 调制周期 $=2^8\times$PWM10 分频比$\div F_{OSC}$

$=2^8\times1\div(8\times10^6)$s

=32μs

例：Fosc=8MHz 时 PWM10 的调制周期表

PWM10CON、Bit5、Bit4	PWM10 时钟	Fosc=8MHz
00	Fosc /64	2048μs
01	Fosc /8	256μs
10	Fosc /2	64μs
11	Fosc /1	32μs

16.3.2 10 位 PWM 输出周期

4 个调制周期为一个输出周期。

16.4 10 位 PWM 占空比算法

10 位 PWM 输出的占空比与 PWM10DATA（DC）及 PWM10CON.7～6(AC)的数值相关。从整体上来看，PWM10 的占空比可用下面公式计算：

PWM10 的占空比 =（PWM10 输出周期×4+PWM10 额外周期）÷1024

实际输出时，PWM10 的占空比根据扩展位的不同，分 4 个周期不同的输出。

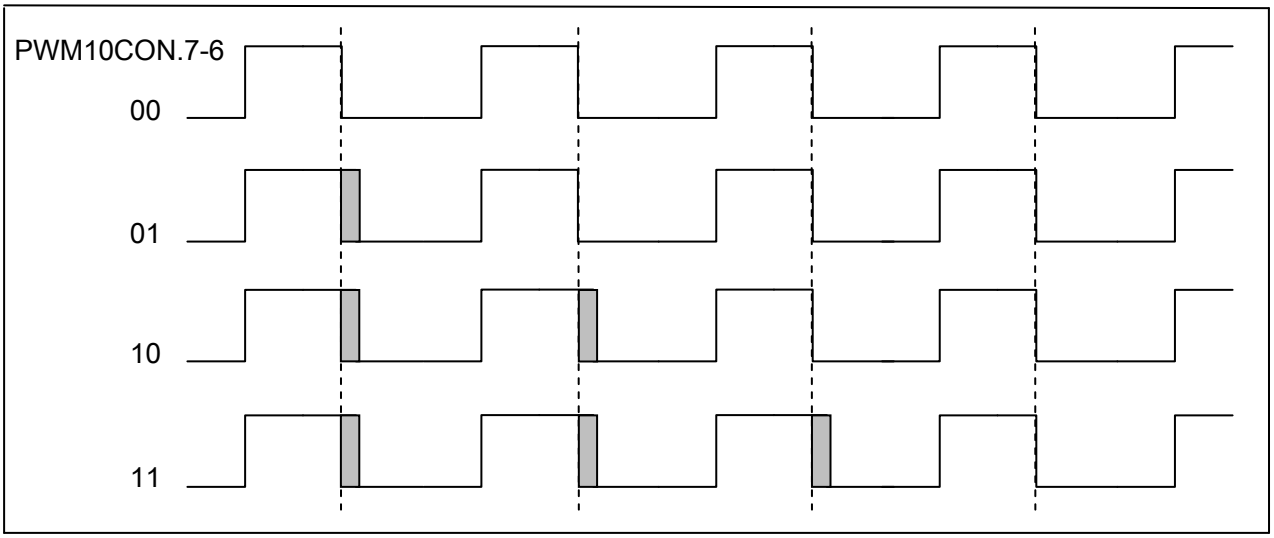


图 16-2：PWM10 占空比输出示意图

如上图所示，当基本输出周期为 PWM10DATA，那么不同的扩展周期时，PWM10 的连续 4 个实际周期分别为：

额外输出周期	周期 1	周期 2	周期 3	周期 4
00	PWM10DATA/256	PWM10DATA/256	PWM10DATA/256	PWM10DATA/256
01	PWM10DATA/256+1	PWM10DATA/256	PWM10DATA/256	PWM10DATA/256
10	PWM10DATA/256+1	PWM10DATA/256+1	PWM10DATA/256	PWM10DATA/256
11	PWM10DATA/256+1	PWM10DATA/256+1	PWM10DATA/256+1	PWM10DATA/256

16.5 10 位 PWM 应用

PWM10 的应用设置需的操作流程如下：

- 设置 PWM10 工作模式及时钟。
- 设置 PWM10DATA。
- P0.7 设置为 PWM10 输出口。
- PWM10 开始工作。

例：PWM10 的设置程序

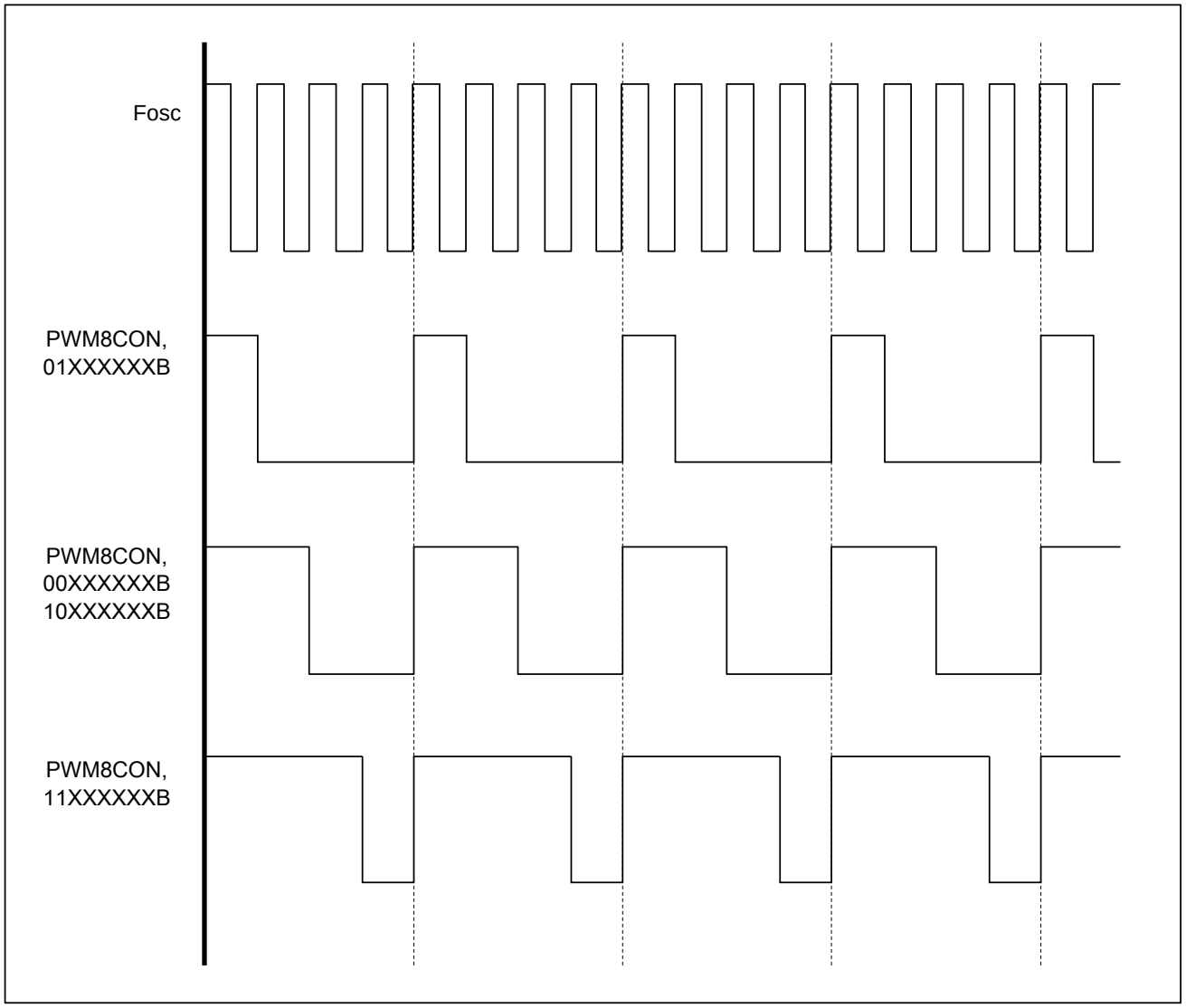
```
        LDIA    B'00110010'
        LD      PWM10CON,A      ;FPWM=FOSC，8 位溢出加载，无扩展周期
        LDIA    B'10000001'
        LD      PWM10DATA,A     ;设置 PWM10 占空比
        LDIA    B'01XXXXXX'
        LD      P0CH,A          ;P0.7 设置为 PWM 输出口
        SETB    PWM10CON,0      ;开启 PWM10
```

17. 高频时钟（CLO）输出

17.1 高频时钟（CLO）输出概述

CMS89F11x 有一个高频脉冲宽度调制器 PWM2，其输出频率为系统时钟的 4 分频，占空比 为 25%、50%、75%可调，由 PWM8CON.7～6 控制，具体请参照关于 PWM8CON 的说明表格。

17.2 高频时钟（CLO）输出波形



17.3 高频时钟（CLO）应用

CLO 的应用设置需的操作流程如下：

- 设置 CLO 占空比。
- P1.6 设置为 CLO 输出口。

例：CLO 的设置程序

```
        LDIA    B'00XXXXXX'
        LD      PWM8CON,A       ; CLO 占空比 50%
        LDIA    B'111XXXXX'
        LD      P1CH,A          ; P1.6 设置为 CLO 输出口
```

18. 蜂鸣器输出（BUZZER）

18.1 BUZZER 概述

CMS89F11x 的蜂鸣驱动器由 6-Bit 计数器、时钟驱动器、控制寄存器组成。它产生 50%的占空方波，其频率覆盖一个较宽的范围。BUZZER 的输出频率由 BUZCON 寄存器的值控制。

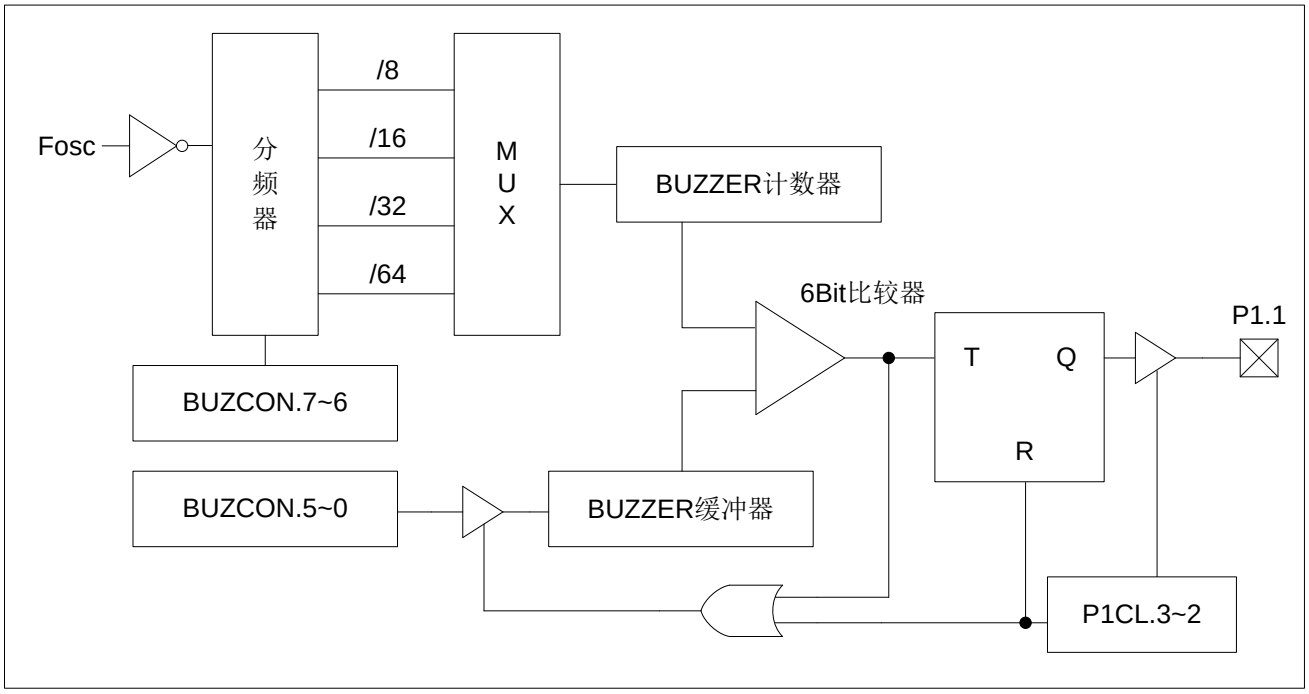


图 18-1：BUZZER 结构框图

设置 P1.1 的控制寄存器，即将 P1CL 的 B3，B2 设为 01，可使蜂鸣输出功能处于使能状态，当蜂鸣输出使能时，6-Bit 计数器被清零，P1.1 输出状态为 0，开始往上计数。如果计数器值和周期数据（BUZCON.5-0）相符，则 P1.1 输出状态被固定，计数器被清零。另外，6-Bit 计数器溢出也可使计数器清零，BUZCON.5-0 决定输出频率。

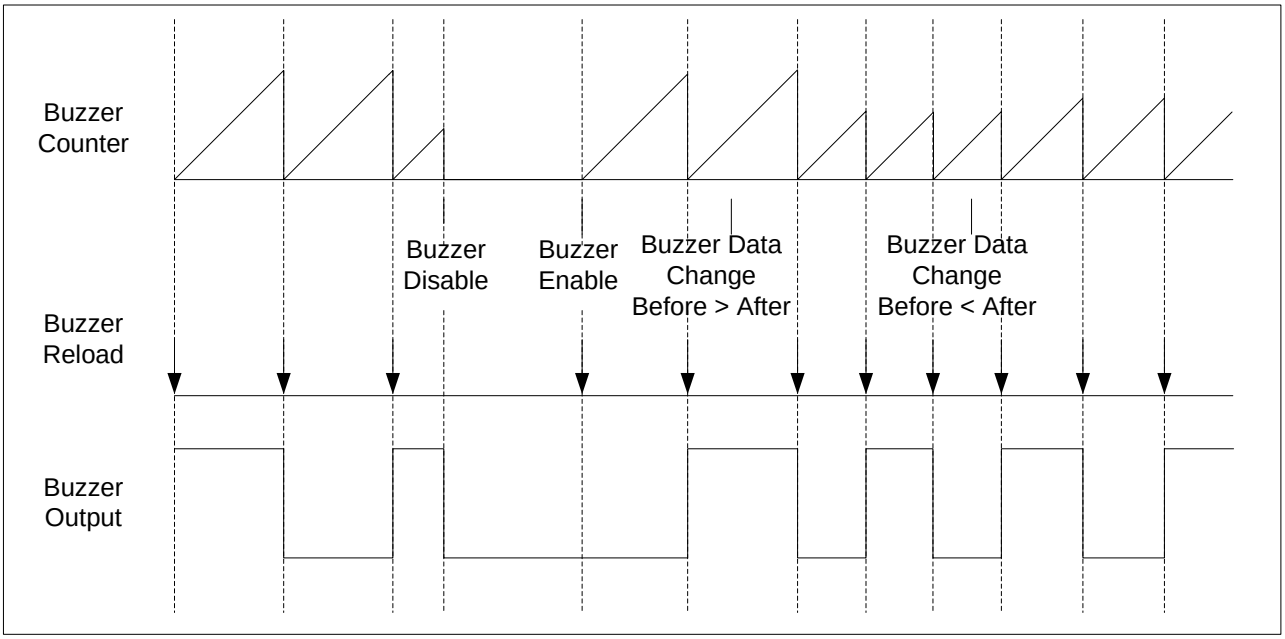


图 18-2：BUZZER 输出时序图

18.2 与 BUZZER 相关的寄存器

21H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
BUZCON	时钟选择		BUZDATA					
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~Bit6　时钟选择。

00: BUZZER 时钟为 Fosc/8；

01: BUZZER 时钟为 Fosc/16；

10: BUZZER 时钟为 Fosc/32；

11: BUZZER 时钟为 Fosc/64。

Bit5~Bit0　BUZDATA: BUZ 输出周期数据。

18.3 BUZZER 输出频率

18.3.1 BUZZER 输出频率计算方法

计算 BUZZER 输出频率的方法如下所示：

BUZZER 输出频率 =Fosc÷[2×分频比×(BUZDATA+1)]

例：Fosc=4MHz，BUZDATA=4，BUZZER 时钟为 Fosc/8 时 BUZZER 的输出频率。

BUZZER 输出频率 =Fosc÷[2×分频比×(BUZDATA+1)]
=4×10^6÷[2×8×(4+1)]
=50KHz

18.3.2 BUZZER 输出频率表

BUZCON、Bit7、Bit6	BUZZER 计数时钟	Fosc=8MHz	
		BUZZER 最小输出频率	BUZZER 最大输出频率
00	Fosc /8	7.81KHz	500KHz
01	Fosc /16	3.91KHz	250KHz
10	Fosc /32	1.95KHz	125KHz
11	Fosc /64	0.98KHz	62.5KHz

18.4 BUZZER 应用

BUZZER 应用设置的操作流程如下：

- 设置 BUZZER 频率。
- P1.1 设置为 BUZZER 输出口。

例：BUZZER 的设置程序

```
        LDIA    B'00000001'
        LD      BUZCON,A
        LDIA    B'XXXX01XX'
        LD      P1CL,A
        CALL    DELY_TIME
        LDIA    B'XXXX10XX'
        LD      P1CL,A
```

19. 电气参数

19.1 DC 特性

符号	参数	测试条件		最小	典型	最大	单位
		VDD	条件				
VDD	工作电压	-	频率=8MHz	2.5	-	5.5	V
			频率=4MHz	1.8	-	5.5	V
I_{DD}	工作电流	5V	ADC 使能	-	2	-	mA
		3V	ADC 使能	-	1.5	-	mA
I_{STB}	静态电流	5V	----	0.1	1	10	μA
		3V	----	0.01	0.1	1	μA
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_{ADI}	AD口输入电压	-	----	0	-	VDD	V
V_{AD}	AD模块工作电压	-	$F_{ADC}=F_{CPU}/16$	2.3	-	5.5	V
E_{AD}	AD转换误差	-	$F_{ADC}=F_{CPU}/16$	-	±4	-	LSB
R_{PH}	上拉电阻阻值	5V	----	-	35	-	K
		3V	----	-	65	-	K
R_{PL}	下拉电阻阻值	5V	----	-	45	-	K
		3V	----	-	100	-	K
	比较器精度	-	----	-	50	-	mV
I_{OL}	输出口灌电流	5V	$V_{OL}=0.3VDD$	-	60	-	mA
		3V	$V_{OL}=0.3VDD$	-	25	-	mA
I_{OH}	输出口拉电流	5V	$V_{OH}=0.7VDD$	-	15	-	mA
		3V	$V_{OH}=0.7VDD$	-	10	-	mA
V_{REF}	内部基准电压	5V	T_A=-20~70℃	-2%	-	2%	-

19.2 AC 特性

符号	参数	测试条件		最小	典型	最大	单位
		VDD	条件				
T_{WDT}	WDT 复位时间	5V	---	-	18	-	ms
		3V	---	-	34	-	ms
T_{AD}	AD 转换时间	5V	---	-	49	-	CLK
		3V	---	-	49	-	CLK
F_{INTRC}	内振频率 8 MHz	VDD=4.5 to 5.5V，-20 to 70℃		-1%	-	1%	-
		VDD=4.5 to 5.5V，-40 to 85℃		-1.5%	-	1.5%	-
		VDD=3.0 to 5.5V，-40 to 85℃		-3%	-	3%	-
		VDD=2.5 to 5.5V，-40 to 85℃		-5%	-	5%	-

20. 指令

20.1 指令一览表

助记符		操作	指令周期	标志
控制类-3				
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-4				
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-16				
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-8				
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-4				
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

助记符		操作	指令周期	标志
DECR	[R]	递减数据存储器 R，结果放入 R	1	Z
位操作-2				
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
查表-2				
TABLE	[R]	读取 FLASH 内容结果放入 TABLE_DATAH 与 R	2	NONE
TABLEA		读取 FLASH 内容结果放入 TABLE_DATAH 与 ACC	2	NONE
数学运算-16				
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-5				
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-8				
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

20.2 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 和 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 和 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 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)=09H(C=0)
                           ACC=(80H-77H-C)=08H(C=1)
```

HSUBCR **[R]**

操作: ACC 减 R 减 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)=09H(C=0)
                           R01=(80H-77H-C)=08H(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
```

TABLE **[R]**

操作: 查表，查表结果低 8 位放入 R，高位放入专用寄存器 TABLE_SPH

周期: 2

影响标志位: 无

举例:

```
LDIA    01H             ;ACC 赋值 01H
LD      TABLE_SPH,A     ;ACC 值赋给表格高位地址，TABLE_SPH=1
LDIA    015H            ;ACC 赋值 15H
LD      TABLE_SPL,A     ;ACC 值赋给表格地位地址，TABLE_SPL=15H
TABLE   R01             ;查表 0115H 地址，操作结果：TABLE_DATAH=12H，R01=34H
…
ORG     0115H
DW      1234H
```

TABLEA

操作: 查表，查表结果低 8 位放入 ACC，高位放入专用寄存器 TABLE_SPH

周期: 2

影响标志位: 无

举例:

```
LDIA    01H             ;ACC 赋值 01H
LD      TABLE_SPH,A     ;ACC 值赋给表格高位地址，TABLE_SPH=1
LDIA    015H            ;ACC 赋值 15H
LD      TABLE_SPL,A     ;ACC 值赋给表格地位地址，TABLE_SPL=15H
TABLEA                  ;查表 0115H 地址，操作结果：TABLE_DATAH=12H，ACC=34H
…
ORG     0115H
DW      1234H
```

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 Add2
```

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

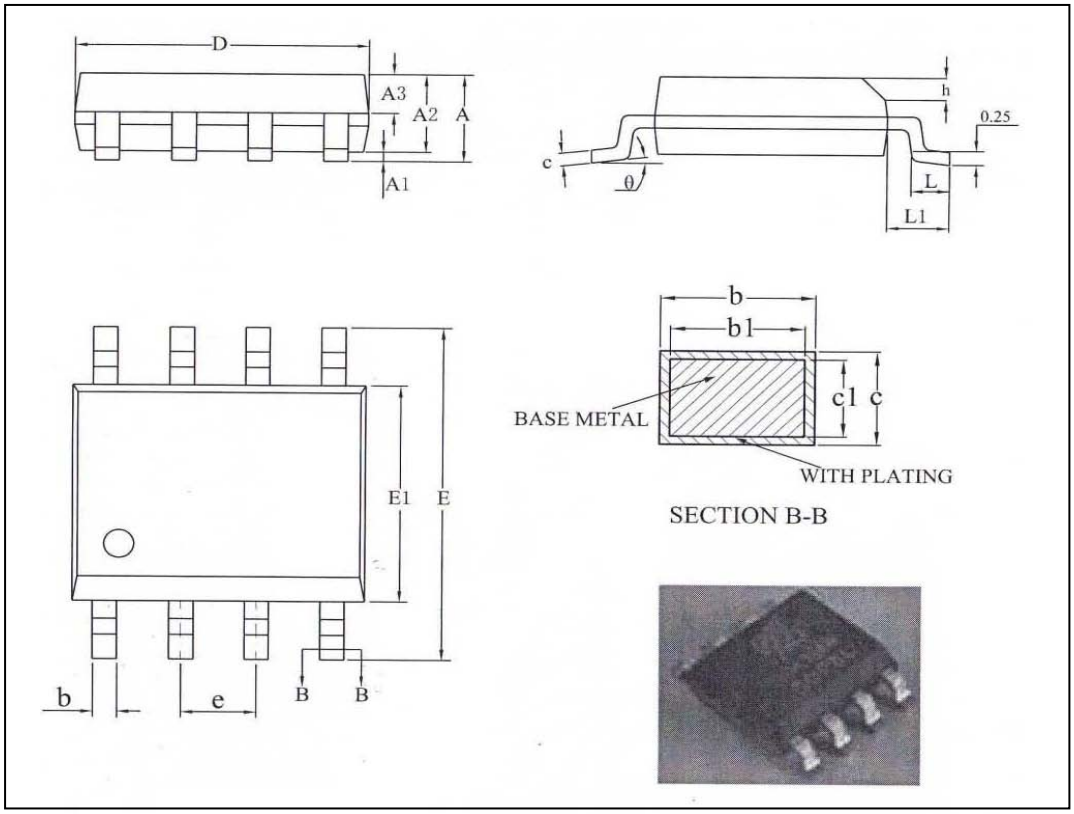
影响标志位: Z

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
```

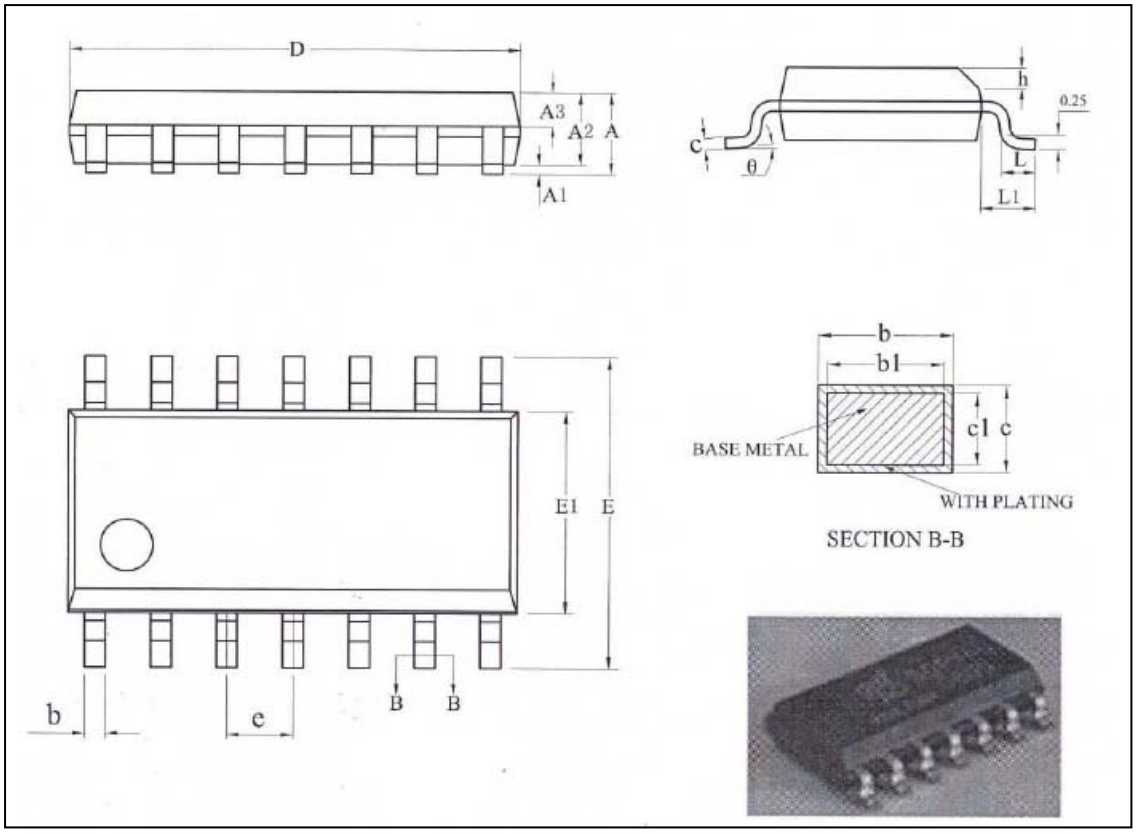
21. 封装

21.1 SOP 8



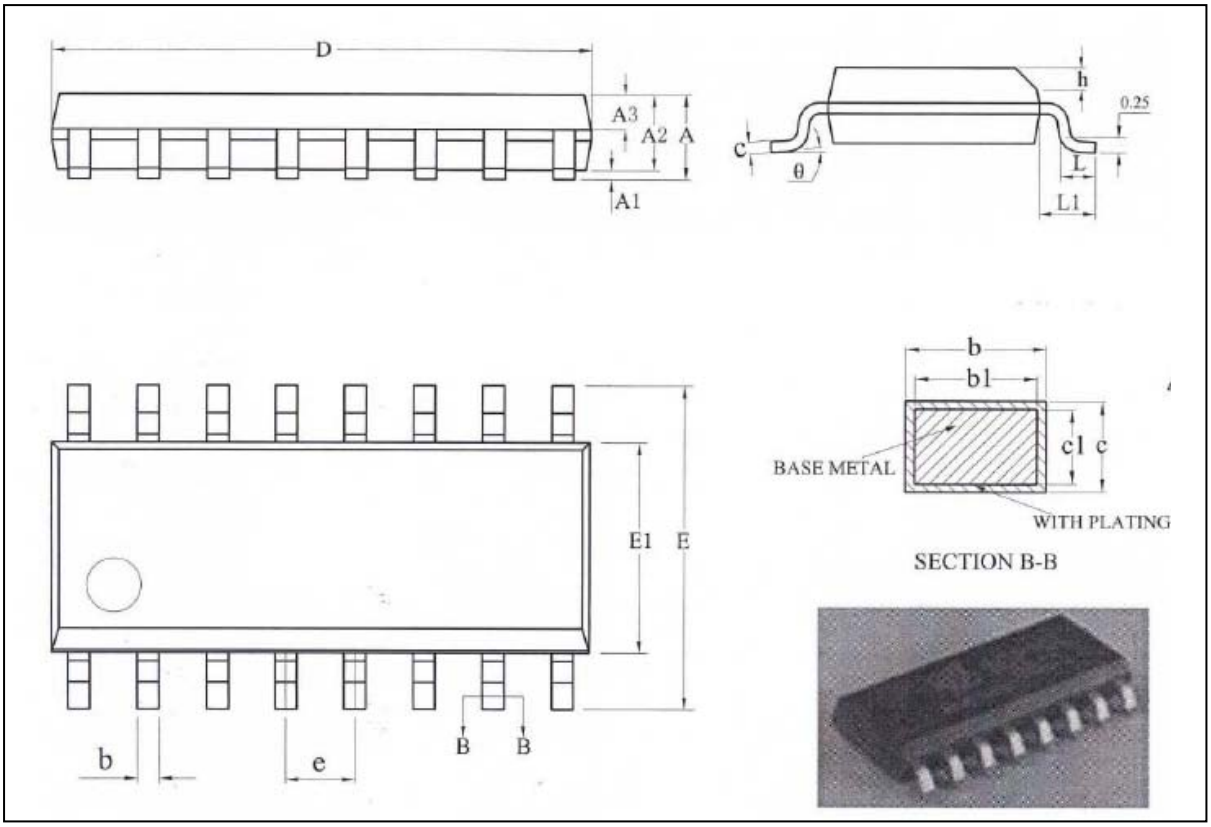
Symbol	Millimeter		
	Min	Nom	Max
A	-	-	1.75
A1	0.10	-	0.225
A2	1.30	1.40	1.50
A3	0.60	0.65	0.70
b	0.39	-	0.47
b1	0.38	0.41	0.44
c	0.20	-	0.24
c1	0.19	0.20	0.21
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
h	0.25	-	0.50
L	0.5	-	0.80
L1	1.05REF		
θ	0	-	8°

21.2 SOP14



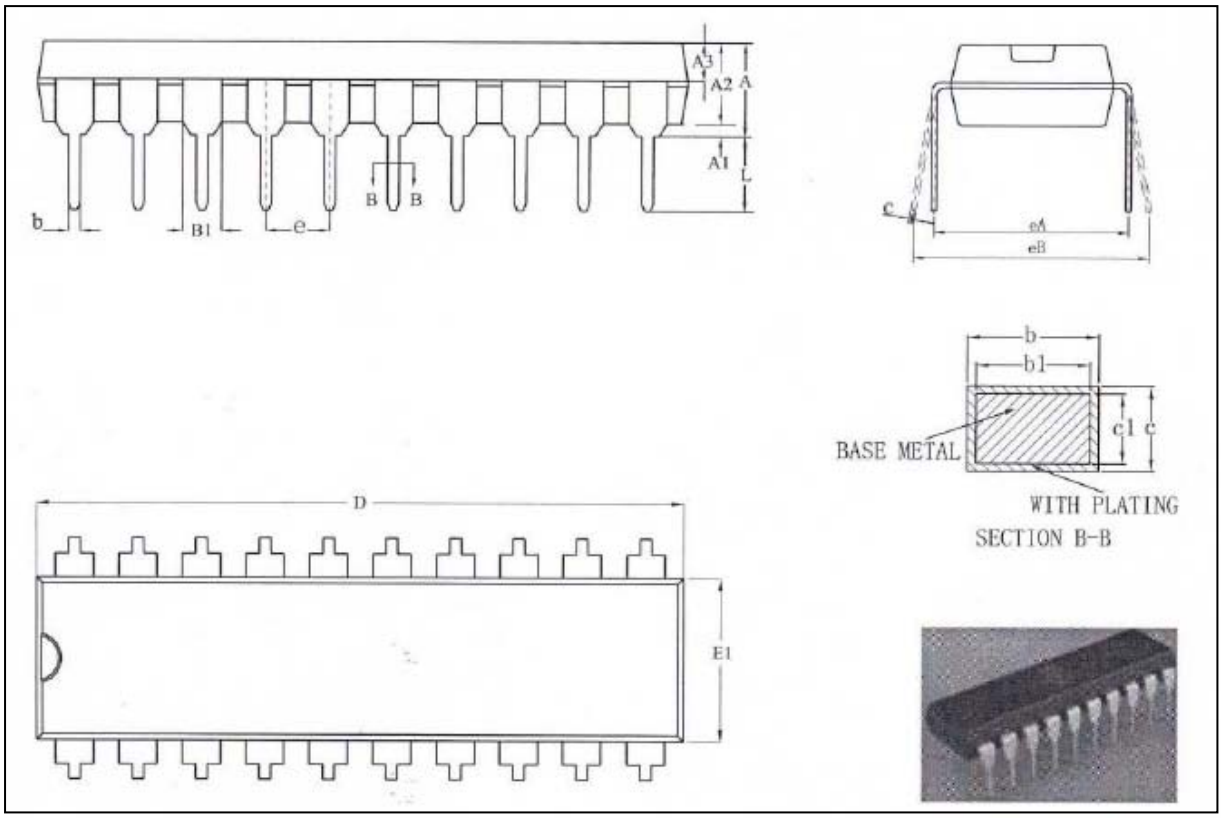
Symbol	Millimeter		
	Min	Nom	Max
A	-	-	1.75
A1	0.05	-	0.225
A2	1.30	1.40	1.50
A3	0.60	0.65	0.70
b	0.39	-	0.47
b1	0.38	0.41	0.44
c	0.20	-	0.24
c1	0.19	0.20	0.21
D	8.55	8.65	8.75
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
h	0.25	-	0.50
L	0.5	-	0.80
L1	1.05REF		
θ	0	-	8°

21.3 SOP 16



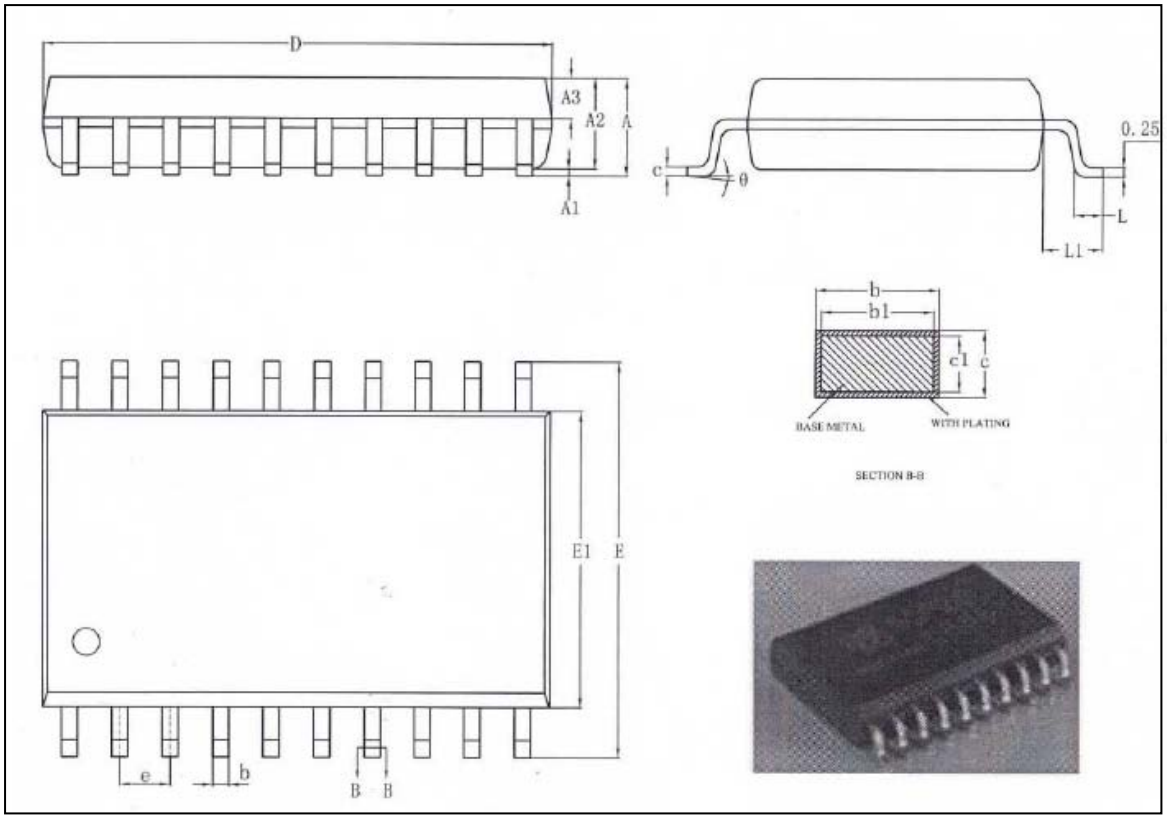
Symbol	Millimeter		
	Min	Nom	Max
A	-	-	1.75
A1	1.10	-	0.225
A2	1.30	1.40	1.50
A3	0.60	0.65	0.70
b	0.39	-	0.47
b1	0.38	0.41	0.44
c	0.20	-	0.24
c1	0.19	0.20	0.21
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
h	0.25	-	0.50
L	0.5	-	0.80
L1	1.05REF		
θ	0	-	8°

21.4 DIP 20



Symbol	Millimeter		
	Min	Nom	Max
A	3.60	3.80	4.00
A1	0.51	-	-
A2	3.20	3.30	3.40
A3	1.47	1.52	1.57
b	0.44	-	0.52
b1	0.43	0.46	0.49
B1	1.52REF		
c	0.25	-	0.29
c1	0.24	0.25	0.26
D	25.80	25.90	26.00
E1	6.45	6.55	6.65
e	2.54BSC		
eA	7.62REF		
eB	7.62	-	9.30
eC	0	-	0.84
L	3.00	-	-

21.5 SOP 20



Symbol	Millimeter		
	Min	Nom	Max
A	-	-	2.65
A1	1.10	-	0.30
A2	2.25	2.30	2.35
A3	0.97	1.02	1.07
b	0.35	-	0.43
b1	0.34	0.37	0.40
c	0.25	-	0.29
c1	0.24	0.25	0.26
D	12.70	12.80	12.90
E	10.10	10.30	10.50
E1	7.40	7.50	7.60
e	1.27BSC		
L	0.70	-	1.00
L1	1.40REF		
θ	0	-	8°

22. 版本修订说明

版本号	时间	修改内容
V1.0	2017 年 8 月	初始版本
V1.1	2017 年 9 月	修改 EE 相关寄存器定义
V1.2	2017 年 9 月	增加使用注意事项
V1.3	2018 年 6 月	修改间接寻址使用说明及注意事项