CMS89F12x用户手册

AD型 MCU

Rev. 1.1

目录

1. 产品概述

1.1 功能特性

- 内存
  - ROM：4K×16Bit
  - 通用 RAM：256×8Bit
- 8 级堆栈缓存器
- 简洁实用的指令系统（66 条指令）
- 内置低压侦测电路
- 内置 WDT 定时器
- 中断源
  - 2 个定时中断
  - RB 口电平变化中断
  - 其它外设中断
- 定时器
  - 8 位定时器 TIMER0，TIMER2
  - TIMER2 可选择外部 32.768KHz 振荡时钟源
- 内置 LVD 模块
  - 支持多种电压 2.2V/2.4V/2.7V/3.0V/3.3V/3.7V/4.0V/4.3V
- 内置 32 字节程序 EEPROM
- 工作电压范围：2.5V~5.5V@16MHz
  1.8V~5.5V@8MHz
  工作温度范围：-40℃~85℃
- 内部 RC 振荡：设计频率 8MHz/16MHz
- 指令周期（单指令或双指令）
- 带互补输出的 PWM 模块
  - 5 路 PWM，可设置成 2 路互补输出
  - RB0/RB1，RB3/RB4 可使能 70mA 大电流驱动
  - 5 路 PWM 共用周期，独立占空比
- 高精度 12 位 ADC
  - 内建高精度 0.6V 基准电压
  - ±1.5% @VDD=2.5V~5.5V T_A=25℃
  - ±2% @VDD=2.5V~5.5V T_A=-40℃~85℃
  - 可选择内部参考源：1.2V/2V/2.4V/3.0V
- 内置 1 路 USART 通信模块

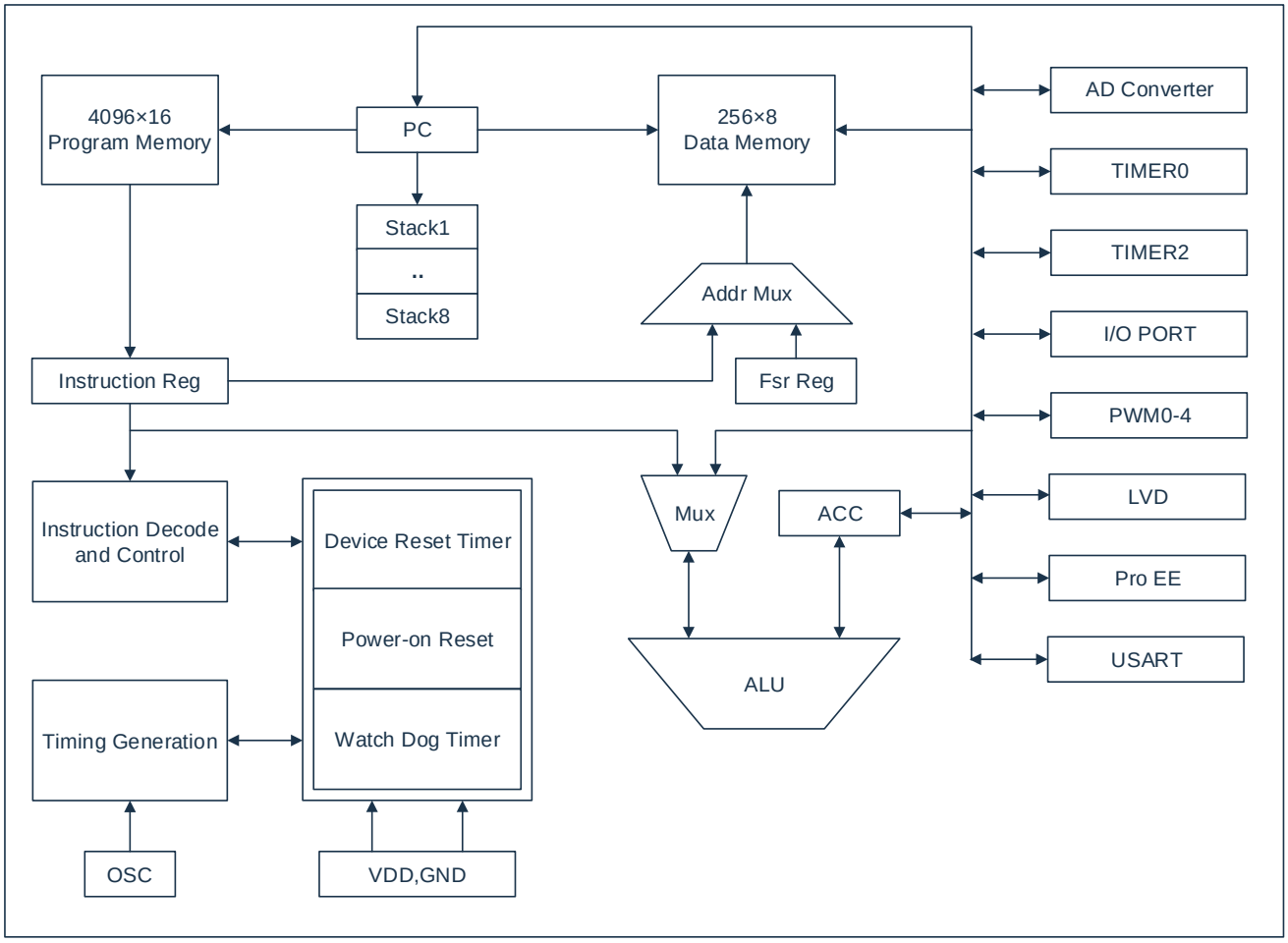
  支持同步主从模式和异步全双工模式

型号说明

PRODUCT	ROM	Pro EE	RAM	USART	I/O	PWM	ADC	PACKAGE
CMS89F121	4Kx16	32x16	256x8	1	6	5	6	SOP8
CMS89F123	4Kx16	32x16	256x8	1	14	5	14	SOP16
CMS89F1231	4Kx16	32x16	256x8	1	14	5	14	SOP16
CMS89F126	4Kx16	32x16	256x8	1	18	5	18	SOP20

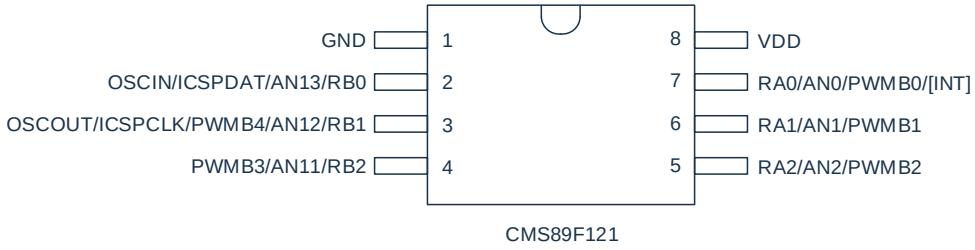
注：ROM----程序存储器　　Pro EE----程序EEPROM

1.2 系统结构框图


4096×16
Program Memory
PC
256×8
Data Memory
AD Converter
TIMER0
TIMER2
I/O PORT
PWM0-4
LVD
Pro EE
USART
Stack1
..
Stack8
Addr Mux
Instruction Reg
Fsr Reg
Instruction Decode
and Control
Device Reset Timer
Power-on Reset
Watch Dog Timer
Mux
ACC
ALU
Timing Generation
OSC
VDD,GND

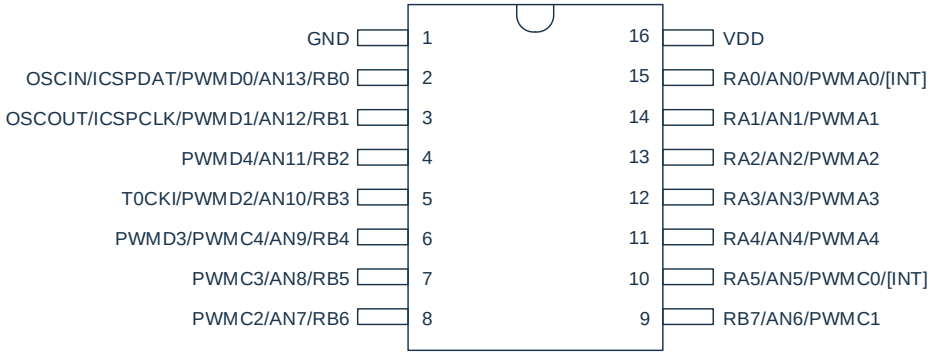
1.3 管脚分布

1.3.1 CMS89F121 引脚图



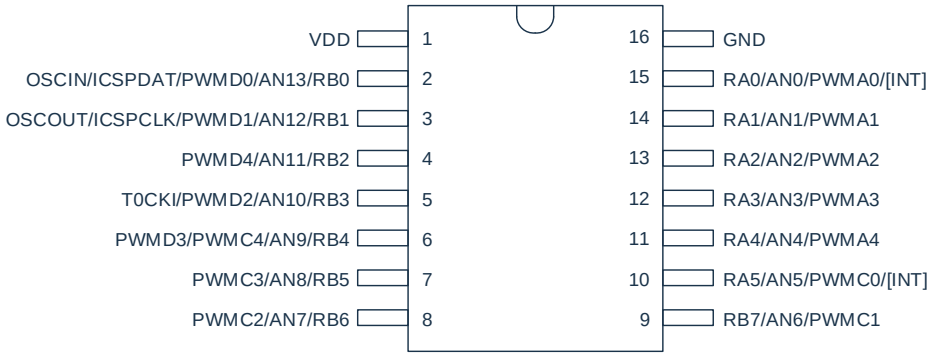
CMS89F121

1.3.2 CMS89F123 引脚图



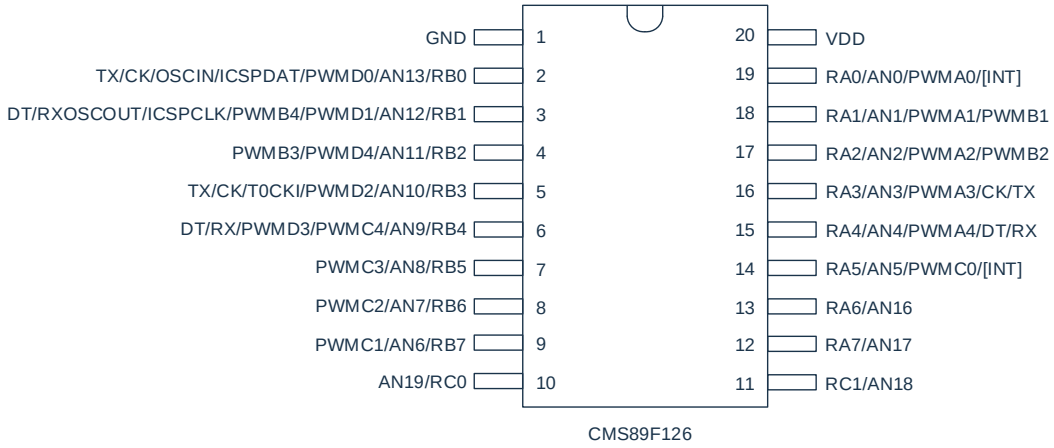
CMS89F123

1.3.3 CMS89F1231 引脚图



CMS89F1231

1.3.4 CMS89F126 引脚图



CMS89F12x 引脚说明：

管脚名称	IO 类型	管脚说明
VDD,GND	P	电源电压输入脚，接地脚
OSCIN/OSCOUT	I/O	晶振输入/输出引脚
RA0-RA7	I/O	可编程为输入脚，推挽输出脚，带上拉电阻功能
RB0-RB7	I/O	可编程为输入脚，推挽输出脚，带上拉电阻功能、带下拉电阻功能、电平变化中断功能
RC0-RC1	I/O	可编程为输入脚，推挽输出脚，带上拉电阻功能
ICSPCLK	I	编程时钟输入脚
ICSPDAT	I/O	编程数据输入/输出脚
AN0-AN13 AN16-AN19	I	12 位 ADC 输入脚
PWMA0-PWMA4	O	A 组 PWM0-4 输出功能
PWMB0-PWMB4	O	B 组 PWM0-4 输出功能
PWMC0-PWMC4	O	C 组 PWM0-4 输出功能
PWMD0-PWMD4	O	D 组 PWM0-4 输出功能
TX/CK	O	异步串口发送引脚/同步串口时钟输入输出脚（可配置在不同 IO 口）
RX/DT	I	串口接收引脚/同步串口数据输入输出脚（可配置在不同 IO 口）
INT	I	外部中断输入脚
T0CKI	I	TIMER0 外部时钟输入脚

1.4 系统配置寄存器

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

1. OSC（振荡方式选择）
   - INTRC8M　　FOSC 选择内部 8MHz RC 振荡
   - INTRC16M　　FOSC 选择内部 16MHz RC 振荡
2. WDT（看门狗选择）
   - ENABLE　　打开看门狗定时器
   - DISABLE　　关闭看门狗定时器
3. PROTECT（加密）
   - DISABLE　　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
4. LVR_SEL（低压侦测电压选择）
   - 1.8V
   - 2.1V
   - 2.5V
   - 3.0V
5. SLEEP_LVREN（休眠态 LVR 使能位）
   - DISABLE　　休眠态下 LVR 功能关闭
   - ENABLE　　休眠态下 LVR 功能打开
6. EXTINT_SEL（外部中断选择）
   - RA0　　选择 RA0 为外部中断口
   - RA5　　选择 RA5 为外部中断口
7. USART_SEL（USART 引脚选择）
   - RA3/RA4
   - RB3/RB4
   - RB0/RB1
8. PWM3_ISEL（PWM3 电流选择）
   - 70mA　　PWMD3 推挽输出电流均为 70mA
   - Normal　　PWMD3 推挽输出为普通 IO 口电流
9. PWM2_ISEL（PWM2 电流选择）
   - 70mA　　PWMD2 推挽输出电流均为 70mA
   - Normal　　PWMD2 推挽输出为普通 IO 口电流
10. PWM1_ISEL（PWM1 电流选择）
    - 70mA　　PWMD1 推挽输出电流均为 70mA
    - Normal　　PWMD1 推挽输出为普通 IO 口电流
11. PWM0_ISEL（PWM0 电流选择）
    - 70mA　　PWMD0 推挽输出电流均为 70mA
    - Normal　　PWMD0 推挽输出为普通 IO 口电流

12. ICSPPORT_SEL（仿真口功能选择）

- ICSP　　ICSPCLK、DAT 口一直保持为仿真口，所有功能均不能使用
- NORMAL　　ICSPCLK、DAT 口为普通功能口

1.5 在线串行编程

可在最终应用电路中对单片机进行串行编程。编程可以简单地通过以下 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。

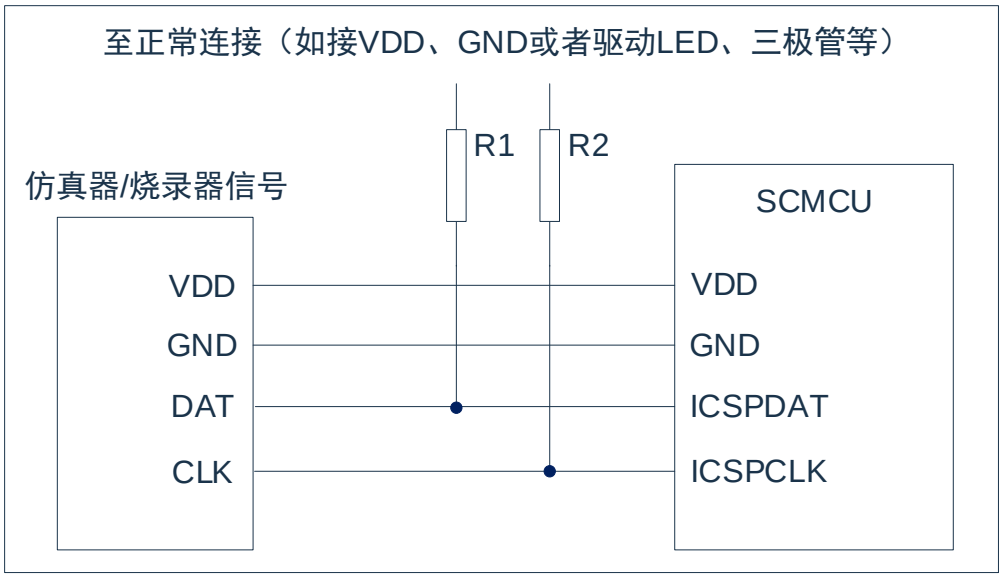


图 1-1：典型的在线串行编程连接方法

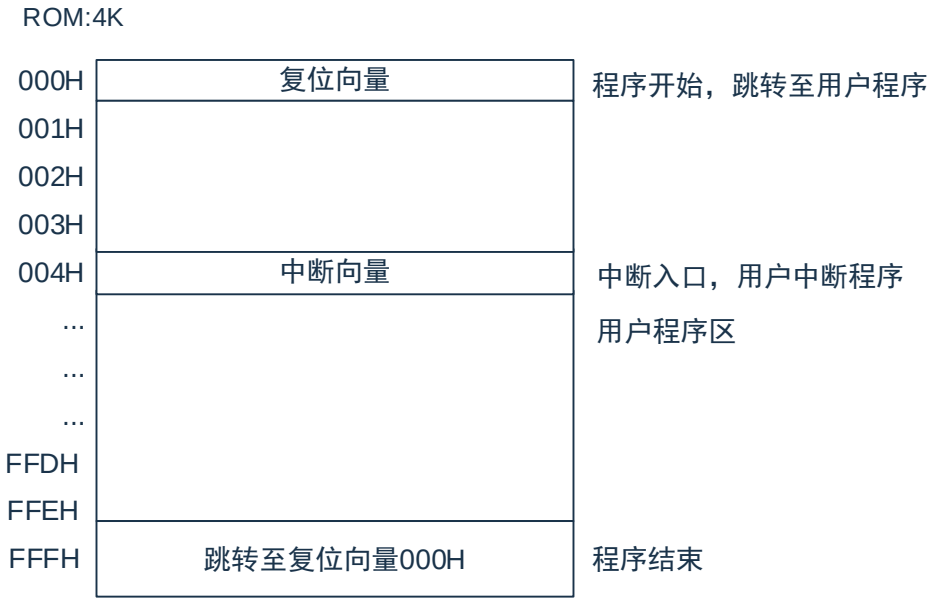
上图中，R1、R2 为电气隔离器件，常以电阻代替，其阻值如下：R1≥4.7K、R2≥4.7K。

2. 中央处理器（CPU）

2.1 内存

2.1.1 程序内存

CMS89F12x 程序存储器空间



2.1.1.1 复位向量（0000H）

单片机具有一个字长的系统复位向量（0000H）。具有以下 3 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的位决定。下面的示例程序说明了如何编写中断服务程序。

例：定义中断向量，中断程序放在用户程序之后

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

注：由于单片机并未提供专门的出栈、压栈指令，故用户需自己保护中断现场。

例：中断入口保护现场

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

例：中断出口恢复现场

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

2.1.1.3 跳转表

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

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

例：正确的多地址跳转程序示例

```
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
                …
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

CMS89F12x 数据存储器列表

BANK0	地址	BANK1	地址	BANK2	地址	BANK3	地址
INDF	00H	INDF	80H	INDF	100H	INDF	180H
TMR0	01H	OPTION_REG	81H		101H		181H
PCL	02H	PCL	82H	PCL	102H	PCL	182H
STATUS	03H	STATUS	83H	STATUS	103H	STATUS	183H
FSR	04H	FSR	84H	FSR	104H	FSR	184H
PORTA	05H	TRISA	85H		105H	PORTC	185H
PORTB	06H	TRISB	86H		106H	TRISC	186H
WPUA	07H	WPDB	87H	PIR2	107H		187H
WPUB	08H	OSCCON	88H	PIE2	108H		188H
IOCB	09H	----	89H		109H		189H
PCLATH	0AH	PCLATH	8AH	PCLATH	10AH	PCLATH	18AH
INTCON	0BH	INTCON	8BH	INTCON	10BH	INTCON	18BH
PIR1	0CH	EECON1	8CH		10CH		18CH
PIE1	0DH	EECON2	8DH		10DH		18DH
PWMD23H	0EH	EEDAT	8EH		10EH		18EH
PWM01DT	0FH	EEDATH	8FH		10FH		18FH
PWM23DT	10H	EEADR	90H		110H		190H
TMR2	11H	PR2	91H		111H		191H
T2CON	12H		92H		112H		192H
PWMCON0	13H		93H		113H		193H
PWMCON1	14H		94H		114H		194H
PWMTL	15H		95H	WPUC	115H		195H
PWMTH	16H	EEADRH	96H		116H		196H
PWMD0L	17H		97H	TXSTA	117H		197H
PWMD1L	18H		98H	RCSTA	118H		198H
PWMD2L	19H		99H	SPBRG	119H		199H
PWMD3L	1AH		9AH	TXREG	11AH		19AH
PWMD4L	1BH		9BH	RCREG	11BH		19BH
PWMD01H	1CH	ADCON1	9CH		11CH	----	19CH
PWMCON2	1DH	ADCON0	9DH		11DH	----	19DH
	1EH	ADRESL	9EH		11EH	----	19EH
	1FH	ADRESH	9FH	LVDCON	11FH	----	19FH
通用寄存器 96 字节	20H	通用寄存器 80 字节	A0H	通用寄存器 80 字节	120H		1A0H
	6FH		EFH		16FH		1EFH
	70H	快速存储区 70H-7FH	F0H	快速存储区 70H-7FH	170H	快速存储区 70H-7FH	1F0H
	--		--		--		--
	7FH		FFH		17FH		1FFH

数据存储器由 512×8 位组成，分为两个功能区间：特殊功能寄存器和通用数据存储器。数据存储器单元大多数是可读/写的，但有些只读的。特殊功能寄存器地址为从 00H-1FH, 80H-9FH, 100H-11FH, 180H-19FH。

CMS89F12x 特殊功能寄存器汇总 Bank0

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	复位值
00H	INDF	寻址该单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
01H	TMR0	TIMER0数据寄存器								xxxxxxxx
02H	PCL	程序计数器低字节								00000000
03H	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	00011xxx
04H	FSR	间接数据存储器地址指针								xxxxxxxx
05H	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxxxxxx
06H	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxxxxxx
07H	WPUA	WPUA7	WPUA6	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0	00000000
08H	WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	00000000
09H	IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	00000000
0AH	PCLATH	----	----	----	----	程序计数器高4位的写缓冲器				-----0000
0BH	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	00000000
0CH	PIR1	----	EEIF	----	----	----	PWMIF	TMR2IF	ADIF	-0---000
0DH	PIE1	----	EEIE	----	----	----	PWMIE	TMR2IE	ADIE	-0---000
0EH	PWMD23H	----	----	PWMD3[9:8]		----	----	PWMD2[9:8]		--00—00
0FH	PWM01DT	----	----	PWM01死区延时时间						--00000
10H	PWM23DT	----	----	PWM23死区延时时间						--00000
11H	TMR2	TIMER2模块寄存器								00000000
12H	T2CON	CLK_SEL	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	00000000
13H	PWMCON0	CLKDIV[2:0]			PWM4EN	PWM3EN	PWM2EN	PWM1EN	PWM0EN	00000000
14H	PWMCON1	PWMIO_SEL[1:0]		PWM2DTEN	PWM0DTEN	----	----	DT_DIV[1:0]		0000—00
15H	PWMTL	PWM周期低位寄存器								00000000
16H	PWMTH	----	----	PWMD4[9:8]		----	----	PWMT9	PWMT8	--00--00
17H	PWMD0L	PWM0占空比低位寄存器								00000000
18H	PWMD1L	PWM1占空比低位寄存器								00000000
19H	PWMD2L	PWM2占空比低位寄存器								00000000
1AH	PWMD3L	PWM3占空比低位寄存器								00000000
1BH	PWMD4L	PWM4占空比低位寄存器								00000000
1CH	PWMD01H	----	----	PWMD1[9:8]		----	----	PWMD0[9:8]		--00—00
1DH	PWMCON2	----	----	----	PWM4DIR	PWM3DIR	PWM2DIR	PWM1DIR	PWM0DIR	----00000

CMS89F12x 特殊功能寄存器汇总 Bank1

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	复位值
80H	INDF	寻址该地址单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
81H	OPTION_REG	----	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	-1111011
82H	PCL	程序计数器（PC）的低字节								00000000
83H	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	00011xxx
84H	FSR	间接数据存储器地址指针								xxxxxxxx
85H	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	11111111
86H	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	11111111
87H	WPDB	WPDB7	WPDB6	WPDB5	WPDB4	WPDB3	WPDB2	WPDB1	WPDB0	00000000
88H	OSCCON	----	IRCF2	IRCF1	IRCF0	----	----	SWDTEN	----	-101—0-
8AH	PCLATH	----	----	----	----	程序计数器高4位的写缓冲器				-----0000
8BH	INTCON	GIE	PEIE	T01E	INTE	RBIE	T0IF	INTF	RBIF	00000000
8CH	EECON1	EEPGD	----	----	----	WRERR	WREN	WR	RD	0—0x000
8DH	EECON2	EEPROM控制寄存器2（不是物理寄存器）								--------
8EH	EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0	xxxxxxxx
8FH	EEDATH	EEDATH7	EEDATH6	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0	xxxxxxxx
90H	EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0	00000000
91H	PR2	TIMER2周期寄存器								00000000
92H	----	KDONE	----	KCAPK[2:0]			KOUT	KCAP	KEN	0-000000
93H	----	KVREF[1:0]		KCLK[1:0]		KCHS[3:0]				00000000
94H	----	----								xxxxxxxx
95H	----	----								xxxxxxxx
96H	EEADRH	----	----	----	----	EEADRH3	EEADRH2	EEADRH1	EEADRH0	----0000
97H	----	CAP_LVBO[2:0]			----	----	----	KCHS4	TKEN	000—-00
98H	----	----	----	----	----	----	----	----	----	00-0---0
99H	----	----	----	----	----					000xxxxx
9AH	----	----	----	----	----	----	----	----	----	00-0---0
9BH	----	----	----	----	----					000xxxxx
9CH	ADCON1	ADFM	CHS4	----	----	----	LDO_EN	LDO_SEL[1:0]		00---000
9DH	ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/ $\overline{\text{DONE}}$	ADON	00000000
9EH	ADRESL	ADC结果寄存器的低字节								xxxxxxxx
9FH	ADRESH	ADC结果寄存器的高字节								xxxxxxxx

CMS89F12x 特殊功能寄存器汇总 Bank2

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	复位值
100H	INDF	寻址该地址单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
102H	PCL	程序计数器（PC）的低字节								00000000
103H	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	00011xxx
104H	FSR	间接数据存储器地址指针								xxxxxxxx
107H	PIR2	----	----	----	----	----	----	----	LVDIF	-------0
108H	PIE2	----	----	----	----	----	----	----	LVDIE	-------0
10AH	PCLATH	----	----	----	----	程序计数器高4位的写缓冲器				-----0000
10BH	INTCON	GIE	PEIE	T01E	INTE	RBIE	T0IF	INTF	RBIF	00000000
115H	WPUC	----	----	----	----	----	----	WPUC1	WPUC0	------00
117H	TXSTA	CSRC	TX9EN	TXEN	SYNC	SCKP	STOPBIT	TRMT	TX9D	00000010
118H	RCSTA	SPEN	RX9EN	SREN	CREN	RCIDL	FERR	OERR	RX9D	00001000
119H	SPBRG	USART波特率8位寄存器								00000000
11AH	TXREG	USART发送数据寄存器								00000000
11BH	RCREG	USART接收数据寄存器								xxxxxxxx
11FH	LVDCON	LVD_RES	----	----	----	LVD_SEL[2:0]			LVDEN	00---0000

CMS89F12x 特殊功能寄存器汇总 Bank3

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	复位值
180H	INDF	寻址该地址单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
182H	PCL	程序计数器（PC）的低字节								00000000
183H	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	00011xxx
184H	FSR	间接数据存储器地址指针								xxxxxxxx
185H	PORTC	----	----	----	----	----	----	RC1	RC0	------xx
186H	TRISC	----	----	----	----	----	----	TRISC1	TRISC0	------11
18AH	PCLATH	----	----	----	----	程序计数器高4位的写缓冲器				-----0000
18BH	INTCON	GIE	PEIE	T01E	INTE	RBIE	T0IF	INTF	RBIF	00000000

2.2 寻址方式

2.2.1 直接寻址

通过工作寄存器（ACC）来对 RAM 进行操作。

例：ACC 的值送给 30H 寄存器

```
LD      30H,A
```

例：30H 寄存器的值送给 ACC

```
LD      A,30H
```

2.2.2 立即寻址

把立即数传给工作寄存器（ACC）。

例：立即数 12H 送给 ACC

```
LDIA    12H
```

2.2.3 间接寻址

数据存储器能被直接或间接寻址。通过 INDF 寄存器可间接寻址，INDF 不是物理寄存器。当对 INDF 进行存取时，它会根据 FSR 寄存器内的值（低 8 位）和 STATUS 寄存器的 IRP 位（第 9 位）作为地址，并指向该地址的寄存器，因此在设置了 FSR 寄存器和 STATUS 寄存器的 IRP 位后，就可把 INDF 寄存器当作目的寄存器来存取。间接读取 INDF（FSR=0）将产生 00H。间接写入 INDF 寄存器，将导致一个空操作。以下例子说明了程序中间接寻址的用法。

例：FSR 及 INDF 的应用

```
        LDIA    30H
        LD      FSR,A           ;间接寻址指针指向 30H
        CLRB    STATUS,IRP      ;指针第 9 位清零
        CLR     INDF            ;清零 INDF 实际是清零 FSR 指向的 30H 地址 RAM
```

例：间接寻址清 RAM(20H-7FH)举例：

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

2.3 堆栈

芯片的堆栈缓存器共 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。

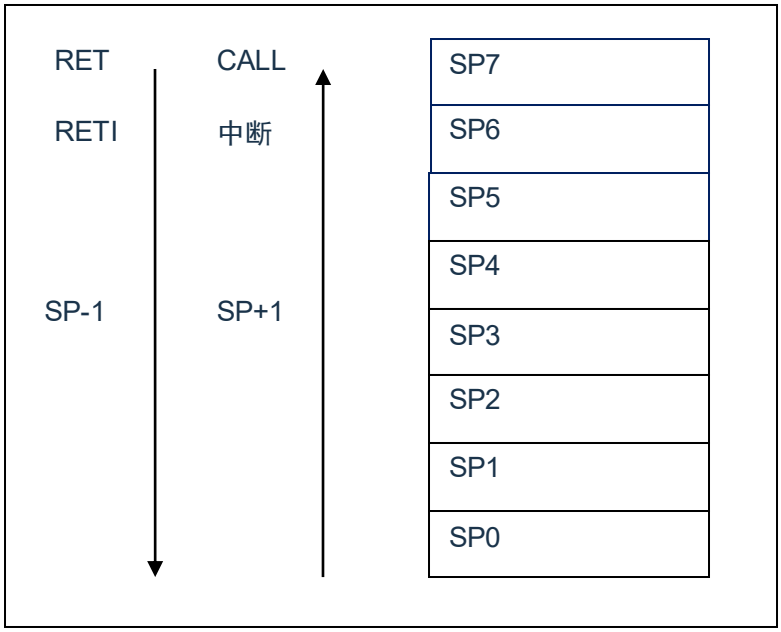


图 2-1：堆栈缓存器工作原理

堆栈缓存器的使用将遵循一个原则“先进后出”。

注：堆栈缓存器只有 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）

STATUS 寄存器如下表所示，包含：

- ALU 的算术状态。
- 复位状态。
- 数据存储器（GPR 和 SFR）的存储区选择位。

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

例如，CLRSTATUS 会清零高 3 位，并将 Z 位置 1。这样 STATUS 的值将为 000u u1uu（其中 u=不变）。因此，建议仅使用 CLRB、SETB、SWAPA、SWAPR 指令来改变 STATUS 寄存器，因为这些指令不会影响任何状态位。

程序状态寄存器 STATUS(03H)

03H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C
读写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	1	1	X	X	X

Bit7 IRP: 寄存器存储器选择位（用于间接寻址）；
1= Bank2和Bank3（100h-1FFh）；
0= Bank0和Bank1（00h-FFh）。

Bit6~Bit5 RP[1:0]: 存储区选择位；
00: 选择Bank 0；
01: 选择Bank 1；
10: 选择Bank 2；
11: 选择Bank 3。

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

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

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

Bit1 DC: 半进位/借位位；
1= 发生了结果的第4低位向高位进位；
0= 结果的第4低位没有向高位进位。

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

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

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

影响 TO/PD 的事件表

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

复位后 TO/PD 的状态

2.6 预分频器（OPTION_REG）

OPTION_REG 寄存器是可读写的寄存器，包括各种控制位用于配置：

- TIMER0/WDT 预分频器。
- TIMER0。

预分频器 OPTION_REG(81H)

81H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OPTION_REG	---	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
读写	---	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	1	1	1	1	0	1	1

Bit7 未用

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

T0CS: TIMER0 时钟源选择位。

Bit5 0= 内部指令周期时钟（F_{CPU}）。
1= T0CKI 引脚上的跳变沿。

T0SE: TIMER0 时钟源边沿选择位。

Bit4 0= 在 T0CKI 引脚信号从低电平跳变到高电平时递增。
1= 在 T0CKI 引脚信号从高电平跳变到低电平时递增。

PSA: 预分频器分配位。

Bit3 0= 预分频器分配给 TIMER0 模块。
1= 预分频器分配给 WDT。

Bit2~Bit0 PS2~PS0: 预分频参数配置位。

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

预分频寄存器实际上是一个 8 位的计数器，用于监视寄存器 WDT 时，是作为一个后分频器；用于定时器/计数器时，作为一个预分频器，通常统称作预分频器。在片内只有一个物理的分频器，只能用于 WDT 或 TIMER0，两者不能同时使用。也就是说，若用于 TIMER0，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 TIMER0 时，有关写入 TIMER0 的所有指令（如：CLR TMR0,SETB TMR0,1 等）都会对预分频器清零。

由 TIMER0 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 TIMER0 换为 WDT 使用时，应该执行以下指令。

```
        CLRB    INTCON,GIE          ;关中断总使能位,避免在执行以下特定时序时进入中断程序
        LDIA    B'00000111'
        ORR     OPTION_REG,A        ;预分频器设置为最大值
        CLR     TMR0                ;TMR0 清零
        SETB    OPTION_REG,PSA      ;设置预分频器分配给 WDT
        CLRWDT                      ;WDT 清零
        LDIA    B'xxxx1xxx'         ;设置新的预分频器
        LD      OPTION_REG,A
        CLRWDT                      ;WDT 清零
        SETB    INTCON,GIE          ;若程序需要用到中断,此处重新打开总使能位
```

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

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

注：要使 TIMER0 获取 1:1 的预分频比配置，可通过将选项寄存器的 PSA 位置 1 将预分频器分配给 WDT。

2.7 程序计数器（PC）

程序计数器（PC）控制程序内存 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 12Bit 宽度，低 8 位通过 PCL（02H）寄存器用户可以访问，高 4 位用户不能访问。可容纳 4K×16Bit 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、RET i 时	PC=堆栈出来的值；
操作 PCL 时	PC[11:8]不变，PC[7:0]=用户指定的值；
JP 时	PC=程序指定的值；
其它指令	PC=PC+1；

2.8 看门狗计数器（WDT）

看门狗定时器（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。

2.8.1 WDT 周期

WDT 与 TIMER0 共用 8 位预分频器。在所有复位后，WDT 默认溢出周期为 128ms，WDT 溢出周期计算方式为 16ms*分频系数，假如你需要改变的 WDT 周期，可以设置 OPTION_REG 寄存器。WDT 的溢出周期将受到环境温度、电源电压等参数影响。

“CLRWDT”和“STOP”指令将清除 WDT 定时器以及预分频器里的计数值（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。

2.8.2 看门狗定时器控制

SWDTEN: 软件使能或禁止看门狗定时器位。
1= 使能 WDT。
0= 禁止 WDT（复位值）。

注：
1) SWDTEN 位于 OSCCON 寄存器 Bit1。
2) 如果 CONFIG 中 WDT 配置位=1，则 WDT 始终被使能，而与 SWDTEN 控制位的状态无关。如果 CONFIG 中 WDT 配置位=0，则可以使用 SWDTEN 控制位使能或禁止 WDT。

3. 系统时钟

3.1 概述

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

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

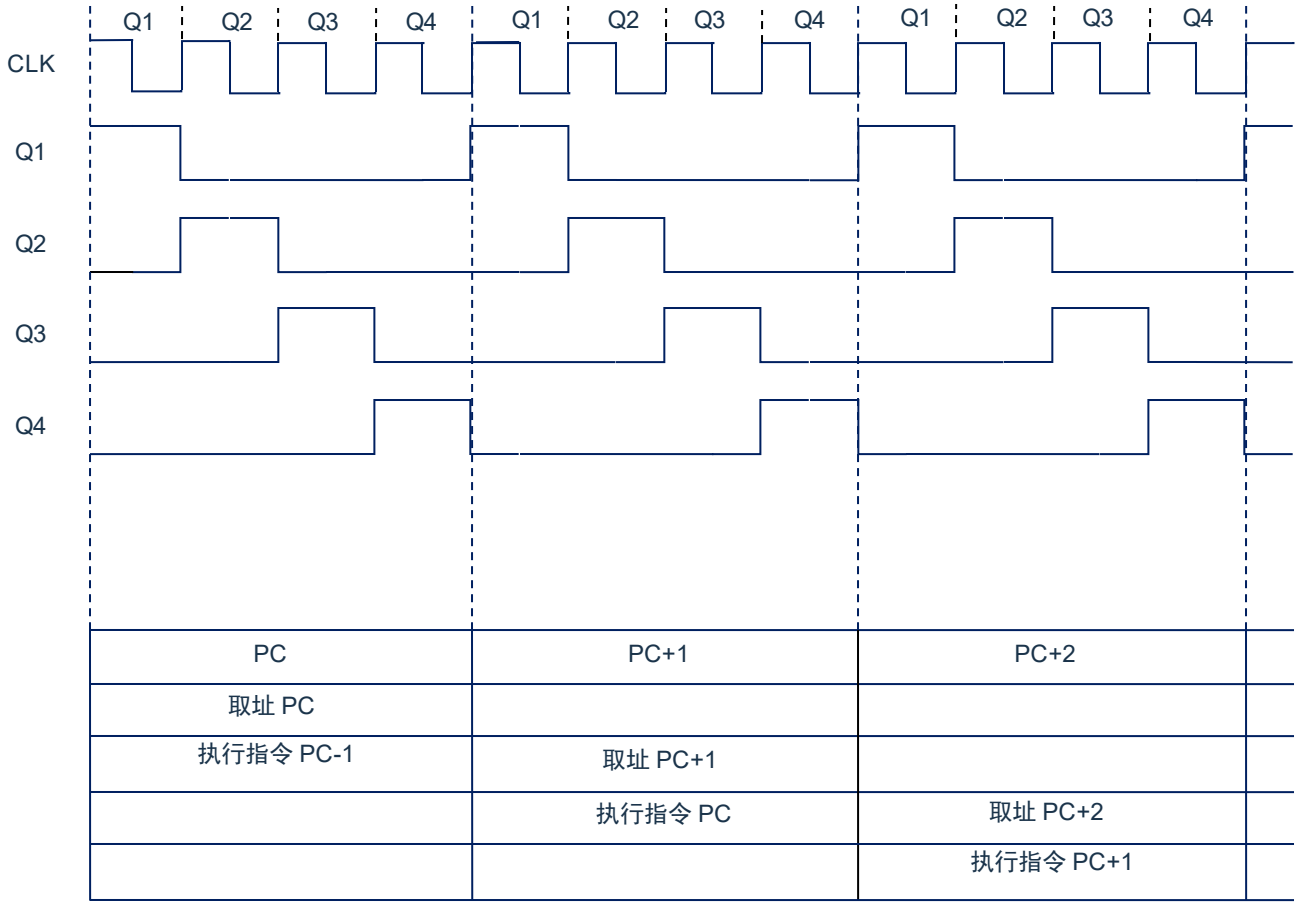


图 3-1：时钟与指令周期时序图

下面列出系统工作频率与指令速度的关系：

系统工作频率(Fsys)	双指令周期	单指令周期
1MHz	8μs	4μs
2MHz	4μs	2μs
4MHz	2μs	1μs
8MHz	1μs	500ns

3.2 系统振荡器

芯片内部集成 8M/16M RC 振荡。

3.2.1 内部 RC 振荡

芯片默认的振荡方式为内部 RC 振荡，其振荡频率为 8MHz 或 16MHz，可通过 OSCCON 寄存器设置芯片工作频率。

3.3 起振时间

起振时间（Reset Time）是指从芯片复位到芯片振荡稳定这段时间，其设计值约为 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。

3.4 振荡器控制寄存器

振荡器控制（OSCCON）寄存器控制系统时钟和频率选择。

振荡器控制寄存器 OSCCON(88H)

88H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OSCCON	---	IRCF2	IRCF1	IRCF0	---	---	SWDTEN	---
R/W	---	R/W	R/W	R/W	---	---	R/W	---
复位值	---	1	0	1	---	---	0	---

Bit7　　未用，读为 0。

Bit6~Bit4　　IRCF<2:0>:　内部振荡器分频选择位。

111= F_{SYS} = F_{OSC}/1

110= F_{SYS} = F_{OSC} /2

101= F_{SYS} = F_{OSC} /4（默认）

100= F_{SYS} = F_{OSC} /8

011= F_{SYS} = F_{OSC} /16

010= F_{SYS} = F_{OSC} /32

001= F_{SYS} = F_{OSC} /64

000= F_{SYS} = 32kHz（LFINTOSC）。

Bit3~Bit2　　未用。

Bit1　　SWDTEN:　软件使能或禁止看门狗定时器位。

1=　使能WDT。

0=　禁止WDT（复位值）。

Bit0　　未用。

注：F_{OSC} 为内部振荡器频率，可选择 8MHz 或 16MHz；F_{SYS} 为系统工作频率。

4. 复位

芯片可用如下 3 种复位方式：

- 上电复位；
- 低电压复位；
- 正常工作下的看门狗溢出复位；

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

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

4.1 上电复位

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

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

4.2 掉电复位

4.2.1 掉电复位概述

掉电复位针对外部因素引起的系统电压跌落情形（例如，干扰或外部负载的变化）。电压跌落可能会进入系统死区，系统死区意味着电源不能满足系统的最小工作电压要求。

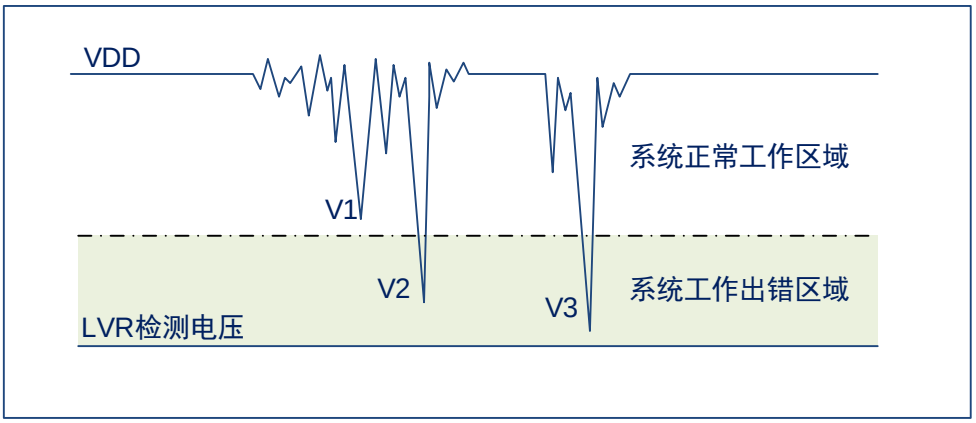


图4-1：掉电复位示意图

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

以下情况系统可能进入死区：

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

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

4.2.2 掉电复位的改进办法

如何改进系统掉电复位性能，以下给出几点建议：

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

看门狗定时器

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

降低系统的工作速度

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

增大电压下降斜率

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

4.3 看门狗复位

看门狗复位是系统的一种保护设置。在正常状态下，由程序将看门狗定时器清零。若出错，系统处于未知状态，看门狗定时器溢出，此时系统复位。看门狗复位后，系统重启进入正常状态。

看门狗复位的时序如下：

- 看门狗定时器状态：系统检测看门狗定时器是否溢出，若溢出，则系统复位；
- 初始化：所有的系统寄存器被置为默认状态；
- 振荡器开始工作：振荡器开始提供系统时钟；
- 程序：复位结束，程序开始运行。

关于看门狗定时器的应用问题请参看 2.8WDT 应用章节。

5. 休眠模式

5.1 进入休眠模式

执行 STOP 指令可进入休眠模式。如果 WDT 使能，那么：

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

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

5.2 从休眠状态唤醒

可以通过下列任一事件将器件从休眠状态唤醒：

1. 看门狗定时器唤醒（WDT 强制使能）
2. PORTB 电平变化中断
3. 其他外设中断

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

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

5.3 使用中断唤醒

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

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

5.4 休眠模式应用举例

系统在进入休眠模式之前，若用户需要获得较小的休眠电流，请先确认所有 I/O 的状态，若用户方案中存在悬空的 I/O 口，把所有悬空口都设置为输出口，确保每一个 I/O 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 等其它外设模块；根据实际方案的功能需求可禁止 WDT 功能来减小休眠电流。

例：进入休眠的处理程序

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

5.5 休眠模式唤醒时间

当 MCU 从休眠态被唤醒时，需要等待一个振荡稳定时间（Reset Time），这个时间在内部高速振荡模式下为 1032 个 T_{SYS} 时钟周期，在内部低速振荡模式下为 15 个 T_{SYS} 时钟周期。具体关系如下表所示

系统主频时钟源	系统时钟分频选择(IRCF<2:0>)	休眠唤醒等待时间 T_{WAIT}
内部高速 RC 振荡（F_{OSC}）	$F_{SYS}=F_{OSC}$	$T_{WAIT}=1032*1/F_{OSC}$
	$F_{SYS}= F_{OSC}/2$	$T_{WAIT}=1032*2/F_{OSC}$
	...	...
	$F_{SYS}= F_{OSC}/64$	$T_{WAIT}=1032*64/F_{OSC}$
内部低速 RC 振荡（$F_{LFINTOSC}$）	----	$T_{WAIT}=15/F_{LFINTOSC}$

6. I/O 端口

6.1 I/O 概述

芯片有 3 个 I/O 端口：PORTA、PORTB、PORTC（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	管脚描述	I/O
PORTA	0	施密特触发输入，推挽式输出，AN0，PWM，INT	I/O
	1	施密特触发输入，推挽式输出，AN1，PWM	I/O
	2	施密特触发输入，推挽式输出，AN2，PWM	I/O
	3	施密特触发输入，推挽式输出，AN3，PWM	I/O
	4	施密特触发输入，推挽式输出，AN4，PWM	I/O
	5	施密特触发输入，推挽式输出，AN5，PWM，INT	I/O
	6	施密特触发输入，推挽式输出，AN16	I/O
	7	施密特触发输入，推挽式输出，AN17	I/O
PORTB	0	施密特触发输入，推挽式输出，AN13，PWM，ICSPDAT，OSCI	I/O
	1	施密特触发输入，推挽式输出，AN12，PWM，ICSPCLK，OSCO	I/O
	2	施密特触发输入，推挽式输出，AN11，PWM	I/O
	3	施密特触发输入，推挽式输出，AN10，PWM	I/O
	4	施密特触发输入，推挽式输出，AN9，PWM	I/O
	5	施密特触发输入，推挽式输出，AN8，PWM	I/O
	6	施密特触发输入，推挽式输出，AN7，PWM	I/O
	7	施密特触发输入，推挽式输出，AN6，PWM	I/O
PORTC	0	施密特触发输入，推挽式输出，AN19	I/O
	1	施密特触发输入，推挽式输出，AN18	I/O

＜表 6-1：端口配置总概＞

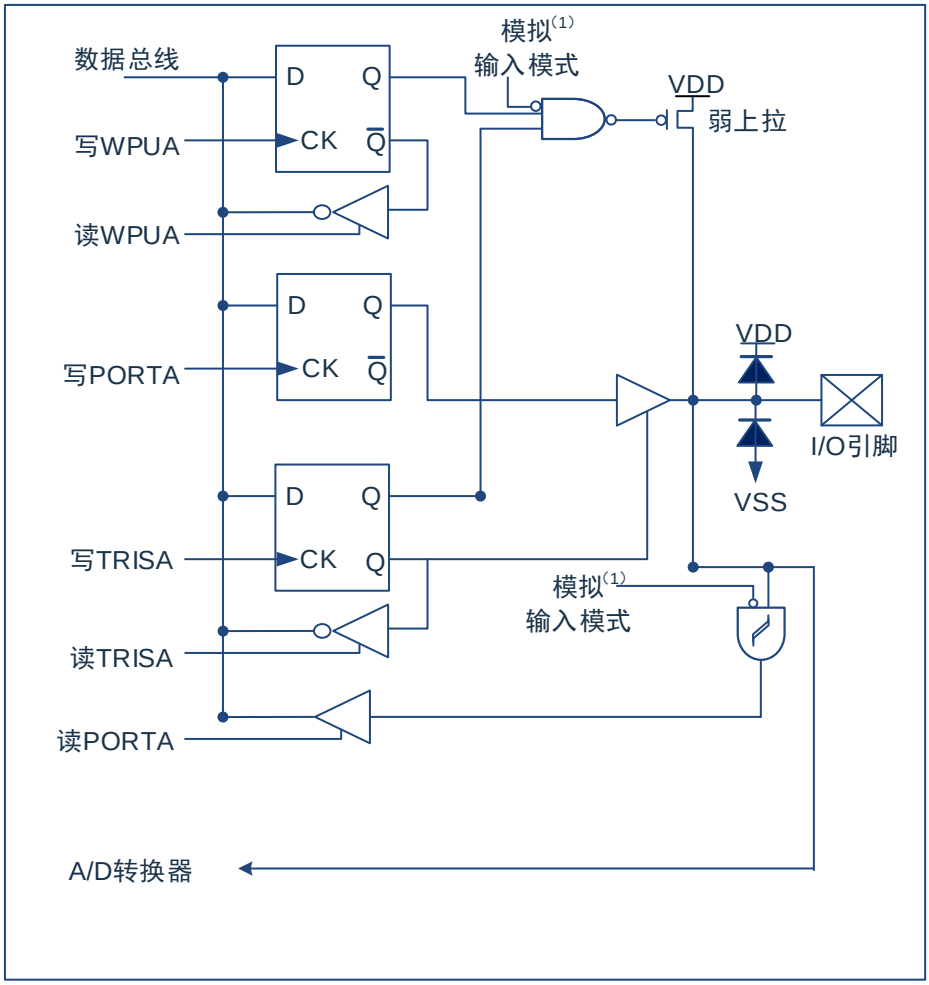

数据总线
写WPUA
读WPUA
写PORTA
写TRISA
读TRISA
读PORTA
D
Q
CK
Q
模拟(1)
输入模式
VDD
弱上拉
VDD
VSS
I/O引脚
模拟(1)
输入模式
A/D转换器

图 6-1：I/O 口结构图

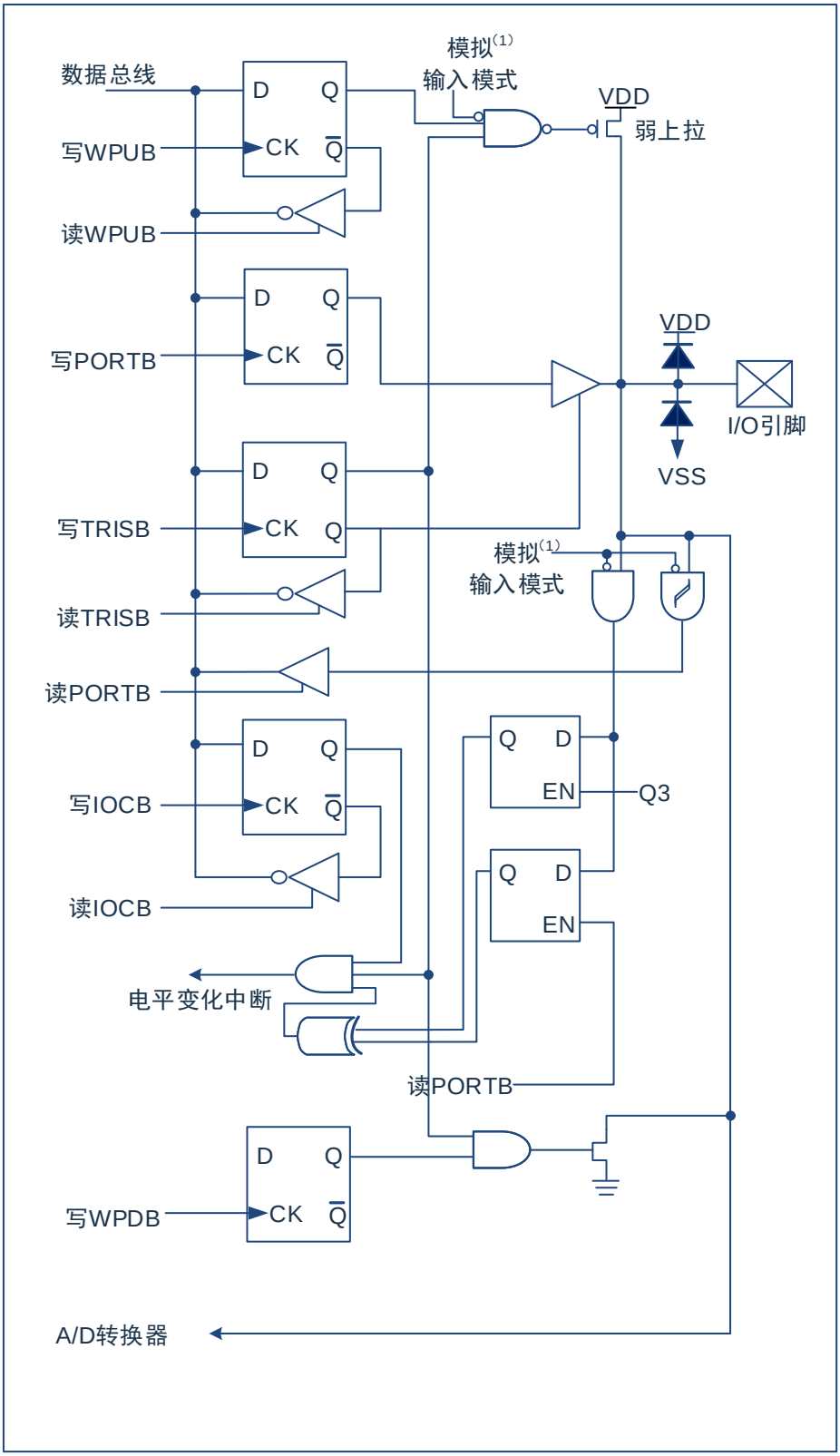

数据总线
写WPUB
读WPUB
写PORTB
写TRISB
读TRISB
读PORTB
写IOCB
读IOCB
写WPDB
D
Q
CK
Q
模拟(1)
输入模式
VDD
弱上拉
VDD
VSS
I/O引脚
模拟(1)
输入模式
Q
D
EN
Q3
电平变化中断
读PORTB
A/D转换器

图 6-2：I/O 口结构图

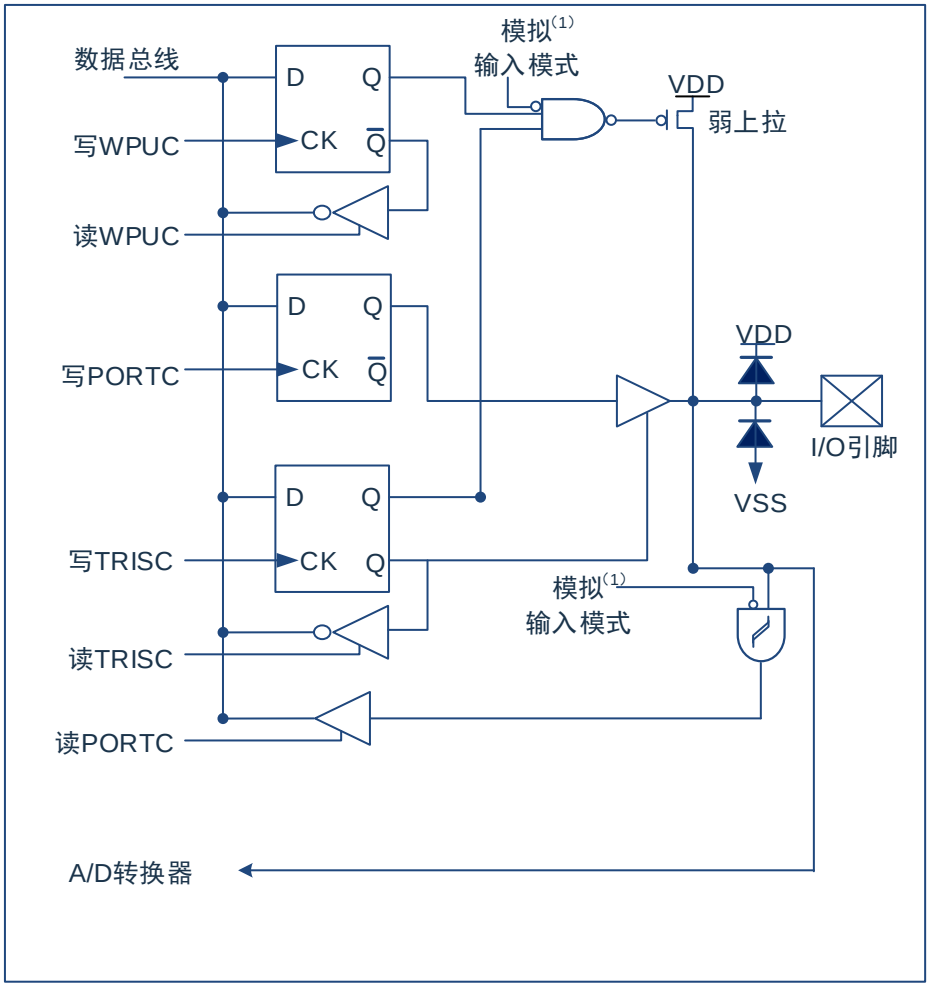

数据总线
写WPUC
读WPUC
写PORTC
写TRISC
读TRISC
读PORTC
D
Q
CK
Q
模拟(1)
输入模式
VDD
弱上拉
VDD
VSS
I/O引脚
模拟(1)
输入模式
A/D转换器

图 6-3：I/O 口结构图

6.2 PORTA

6.2.1 PORTA 数据及方向控制

PORTA 是 8Bit 宽的双向端口。它所对应的数据方向寄存器是 TRISA。将 TRISA 的一个位置 1（=1）可以将相应的引脚配置为输入。清零 TRISA 的一个位（= 0）可将相应的 PORTA 引脚配置为输出。

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

与 PORTA 口相关寄存器有 PORTA、TRISA、WPUA 等。

PORTA 数据寄存器 PORTA(05H)

05H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
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

Bit7~Bit0 PORTA<7:0>: PORTAI/O 引脚位；
1= 端口引脚电平>V_{IH}；
0= 端口引脚电平<V_{IL}。

PORTA 方向寄存器 TRISA(85H)

85H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	1	1	1	1	1	1	1	1

Bit7~Bit0 TRISA<7:0>: PORTA 三态控制位；
1= PORTA 引脚被配置为输入（三态）；
0= PORTA 引脚被配置为输出。

例：PORTA 口处理程序

```
LDIA    B'11110000'     ;设置PORTA<3:0>为输出口，PORTA<7:4>为输入口
LD      TRISA,A
LDIA    03H             ;PORTA<1:0>输出高电平，PORTA<3:2>输出低电平
LD      PORTA,A         ;由于PORTA<7:4>为输入口，所以赋0或1都没影响
```

6.2.2 PORTA 上拉电阻

每个 PORTA 引脚都有可单独配置的内部弱上拉。控制位 WPUA<7:0>使能或禁止每个弱上拉。

PORTA 上拉电阻寄存器 WPUA(07H)

07H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPUA	WPUA7	WPUA6	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0
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　　WPUA<7:0>:　弱上拉寄存器位。

1=　使能上拉。

0=　禁止上拉。

注：如果引脚被配置为输出，将自动禁止弱上拉。

6.3 PORTB

6.3.1 PORTB 数据及方向

PORTB 是一个 8Bit 宽的双向端口。对应的数据方向寄存器为 TRISB。将 TRISB 中的某个位置 1（=1）可以使对应的 PORTB 引脚作为输入引脚。将 TRISB 中的某个位清零（=0）将使对应的 PORTB 引脚作为输出引脚。

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

与 PORTB 口相关寄存器有 PORTB、TRISB、WPUB、WPDB、IOCB 等。

PORTB 数据寄存器 PORTB(06H)

06H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
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

Bit7~Bit0　　PORTB<7:0>:　PORTBI/O 引脚位。

1=　端口引脚电平>V_{IH}。

0=　端口引脚电平<V_{IL}。

PORTB 方向寄存器 TRISB (86H)

86H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	1	1	1	1	1	1	1	1

Bit7~Bit0　　TRISB<7:0>:　PORTB 三态控制位。

1=　PORTB引脚被配置为输入（三态）。

0=　PORTB引脚被配置为输出。

例：PORTB 口处理程序

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

6.3.2 PORTB 下拉电阻

每个 PORTB 引脚都有可单独配置的内部弱下拉。控制位 WPDB<7:0>使能或禁止每个弱下拉。

PORTB 下拉电阻寄存器 WPDB(87H)

87H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPDB	WPDB7	WPDB6	WPDB5	WPDB4	WPDB3	WPDB2	WPDB1	WPDB0
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　　WPDB<7:0>:　弱下拉寄存器位。

1=　使能下拉。

0=　禁止下拉。

注：如果引脚被配置为输出，将自动禁止弱下拉。

6.3.3 PORTB 上拉电阻

每个 PORTB 引脚都有可单独配置的内部弱上拉。控制位 WPUB<7:0>使能或禁止每个弱上拉。

PORTB 上拉电阻寄存器 WPUB(08H)

08H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0　　WPUB<7:0>:　弱上拉寄存器位。
1=　使能上拉。
0=　禁止上拉。

注：如果引脚被配置为输出，将自动禁止弱上拉。

6.3.4 PORTB 电平变化中断

所有的 PORTB 引脚都可以被单独配置为电平变化中断引脚。控制位 IOCB<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 PORTB 时锁存的旧值进行比较。将与上次读操作“不匹配”的输出一起进行逻辑或运算，以将 INTCON 寄存器中的 PORTB 电平变化中断标志位（RBIF）置 1。

该中断可将器件从休眠中唤醒，用户可在中断服务程序中通过以下方式清除中断：

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

不匹配状态会不断将 RBIF 标志位置 1。而读或写 PORTB 将结束不匹配状态，并且允许将 RBIF 标志位清零。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，RBIF 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 RBIF 中断标志位不会被置 1。此外，由于对端口的读或写影响到该端口的所有位，所以在电平变化中断模式下使用多个引脚的时候必须特别小心。在处理一个引脚电平变化的时候可能不会注意到另一个引脚上的电平变化。

PORTB 电平变化中断寄存器 IOCB(09H)

09H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0　　IOCB<7:0>　PORTB 的电平变化中断控制位。
1=　允许电平变化中断。
0=　禁止电平变化中断。

6.4 PORTC

6.4.1 PORTC 数据及方向

PORTC 是一个 2Bit 宽的双向端口。对应的数据方向寄存器为 TRISC。将 TRISB 中的某个位置 1（=1）可以使对应的 PORTC 引脚作为输入引脚。将 TRISC 中的某个位清零（=0）将使对应的 PORTC 引脚作为输出引脚。

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

与 PORTC 口相关寄存器有 PORTC、TRISC、WPUC 等。

PORTC 数据寄存器 PORTC(185H)

185H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PORTC	----	----	----	----	----	----	RC1	RC0
R/W	----	----	----	----	----	----	R/W	R/W
复位值	----	----	----	----	----	----	X	X

Bit7~Bit2　　未用

Bit1~Bit0　　PORTC<1:0>:　PORTC I/O 引脚位。

1=　端口引脚电平>V_{IH}。

0=　端口引脚电平<V_{IL}。

PORTC 方向寄存器 TRISC (186H)

186H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TRISC	----	----	----	----	----	----	TRISC1	TRISC0
R/W	----	----	----	----	----	----	R/W	R/W
复位值	----	----	----	----	----	----	1	1

Bit7~Bit2　　未用

Bit1~Bit0　　TRISC<1:0>:　PORTC 三态控制位。

1=　PORTC引脚被配置为输入（三态）。

0=　PORTC引脚被配置为输出。

例：PORTC 口处理程序

```
CLR     PORTC       ;清数据寄存器
LDIA    B'00000001' ;设置 PORTC<0>为输入口，PORTC<1>为输出口
LD      TRISC,A
```

6.4.2 PORTC 上拉电阻

每个 PORTC 引脚都有可单独配置的内部弱上拉。控制位 WPUC<1:0>使能或禁止每个弱上拉。

PORTC 上拉电阻寄存器 WPUC(115H)

115H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPUC	----	----	----	----	----	----	WPUC1	WPUC0
R/W	----	----	----	----	----	----	R/W	R/W
复位值	----	----	----	----	----	----	0	0

Bit7~Bit2　　未用

Bit1~Bit0　　WPUC<1:0>:　弱上拉寄存器位。

1=　使能上拉。

0=　禁止上拉。

注：如果引脚被配置为输出，将自动禁止弱上拉。

6.5 I/O 使用

6.5.1 写 I/O 口

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

例：写 I/O 口程序

```
LD      PORTA,A     ;ACC 值赋给 PORTA 口
CLRB    PORTB,1     ;PORTB.1 口置零
SET     PORTA       ;PORTA 所有输出口置 1
SETB    PORTB,1     ;PORTB.1 口置 1
```

6.5.2 读 I/O 口

例：读 I/O 口程序

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

6.6 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“GND-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。

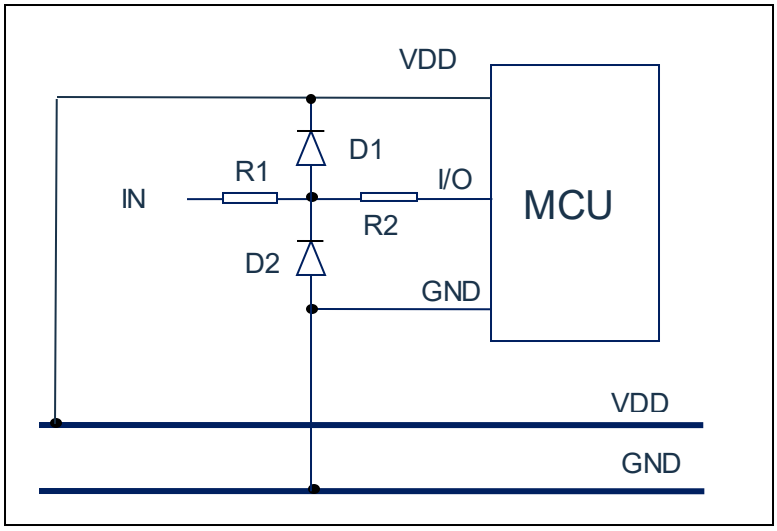


图 6-3：输入电压不在规定范围内采用电路

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

7. 中断

7.1 中断概述

芯片具有以下多种中断源：

- TIMER0 溢出中断
- TIMER2 匹配中断
- PORTB 电平变化中断
- LVD 中断
- USART 接收/发送中断
- AD 中断
- PWM 中断
- INT 中断
- 程序 EEPROM 写操作中断

中断控制寄存器（INTCON）和外设中断请求寄存器（PIR1、PIR2）在各自的标志位中记录各种中断请求。INTCON 寄存器还包括各个中断允许位和全局中断允许位。

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

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

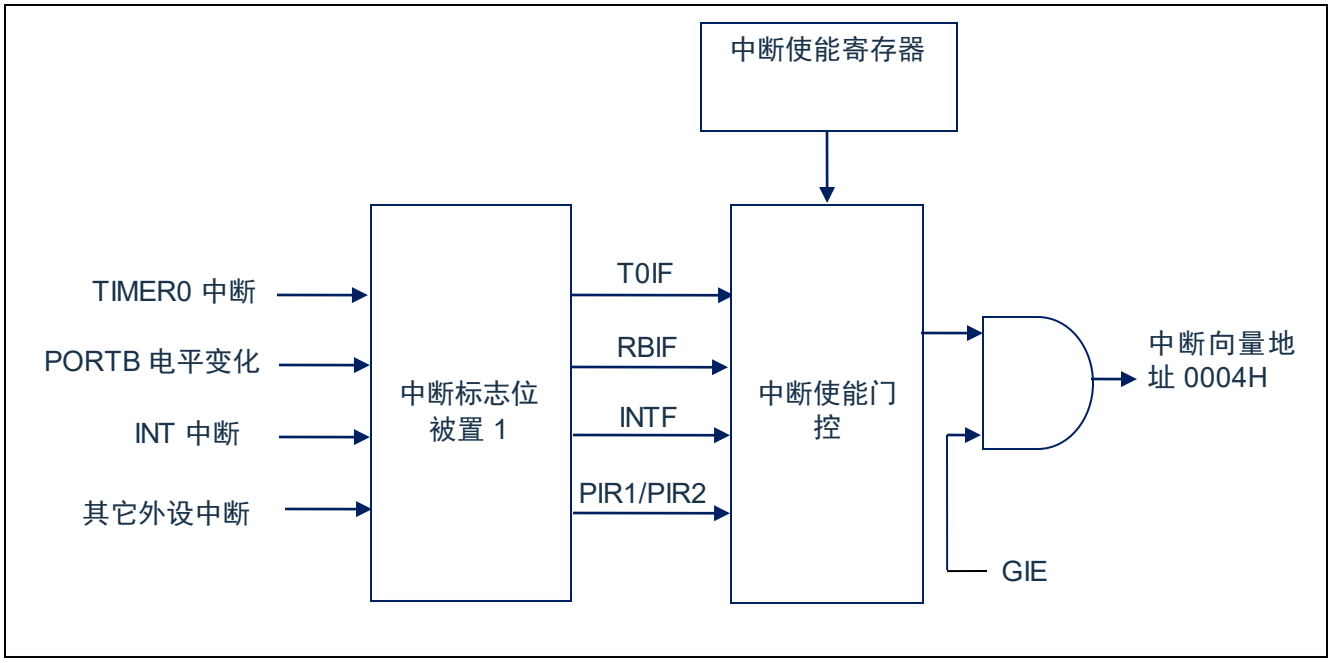


图 7-1：中断原理示意图

7.2 中断控制寄存器

7.2.1 中断控制寄存器

中断控制寄存器 INTCON 是可读写的寄存器，包含 TMR0 寄存器溢出、PORTB 端口电平变化中断等的允许和标志位。

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

中断控制寄存器 INTCON (0BH)

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

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

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

7.2.2 外设中断允许寄存器

外设中断允许寄存器有 PIE1 和 PIE2，在允许任何外设中断前，必须先将 INTCON 寄存器的 PEIE 位置1。

外设中断允许寄存器 PIE1(0DH)

0DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PIE1	----	EEIE	RCIE	TXIE	----	PWMIE	TMR2IE	ADIE
R/W	----	R/W	R/W	R/W	----	R/W	R/W	R/W
复位值	----	0	0	0	----	0	0	0

Bit7~Bit3 未用

Bit6 EEIE: EEDATA中断允许位
1= 允许EEDATA写中断；
0= 禁止EEDATA写中断。

Bit5 RCIE: USART接收中断允许位
1= 允许USART接收中断；
0= 禁止USART接收中断。

Bit4 TXIE: USART发送中断允许位
1= 允许USART发送中断；
0= 禁止USART发送中断。

Bit3 未用

Bit2 PWMIE: PWM中断允许位
1= 允许PWM中断；
0= 禁止PWM中断。

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

Bit0 ADIE: AD转换器（ADC）中断允许位
1= 允许ADC中断；
0= 禁止ADC中断。

外设中断允许寄存器 PIE2(108H)

108H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PIE2	---	---	---	---	---	---	---	LVDIE
R/W	---	---	---	---	---	---	---	R/W
复位值	---	---	---	---	---	---	---	0

Bit7~Bit1 未用

Bit0 LVDIE: LVD中断允许位
1= 允许LVD中断；
0= 禁止LVD中断。

7.2.3 外设中断请求寄存器

外设中断请求寄存器为 PIR1 和 PIR2。当有中断条件产生时，无论对应的中断允许位或全局允许位 GIE 的状态如何，中断标志位都将置 1。用户软件应在允许一个中断之前，确保先将相应的中断标志位清零。

外设中断请求寄存器 PIR1(0CH)

0CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PIR1	----	EEIF	RCIF	TXIF	----	PWMIF	TMR2IF	ADIF
R/W	----	R/W	R/W	R/W	----	R/W	R/W	R/W
复位值	----	0	0	0	----	0	0	0

Bit7 未用

Bit6 EEIF: 程序EEPROM写操作中断标志位；
1= 写操作完成（必须由软件清零）；
0= 写操作未完成或尚未启动。

Bit5 RCIF: USART接收中断标志位；
1= USART接收缓冲器满（通过读RCREG清零）；
0= USART接收缓冲器空。

Bit4 TXIF: USART发送中断标志位；
1= USART发送缓冲器满（通过写TXREG清零）；
0= USART发送缓冲器空。

Bit3 未用

Bit2 PWMIF: PWM中断标志位；
1= 发生了PWM中断（必须由软件清零）
0= 未发生PWM中断

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

Bit0 ADIF: AD转换器中断标志位；
1= AD转换完成（必须由软件清零）；
0= AD转换未完成或尚未启动。

外设中断请求寄存器 PIR2(107H)

107H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PIR2	---	---	---	---	---	---	---	LVDIF
R/W	---	---	---	---	---	---	---	R/W
复位值	---	---	---	---	---	---	---	0

Bit7~Bit1 未用

Bit0 LVDIF: LVD中断标志位
1= 电源电压低于LVD设定的电压点（必须由软件清零）；
0= 电源电压高于LVD设定的电压点。

7.3 中断现场的保护方法

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

例：对 ACC 与 STATUS 进行入栈保护

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

7.4 中断的优先级，及多中断嵌套

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

多个中断同时发生时，MCU 没有预置的中断优先级。首先，必须预先设定好各中断的优先权；其次，利用中断使能位和中断控制位，控制系统是否响应该中断。在程序中，必须对中断控制位和中断请求标志进行检测。

8. 定时计数器 TIMER0

8.1 定时计数器 TIMER0 概述

TIMER0 由如下功能组成：

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

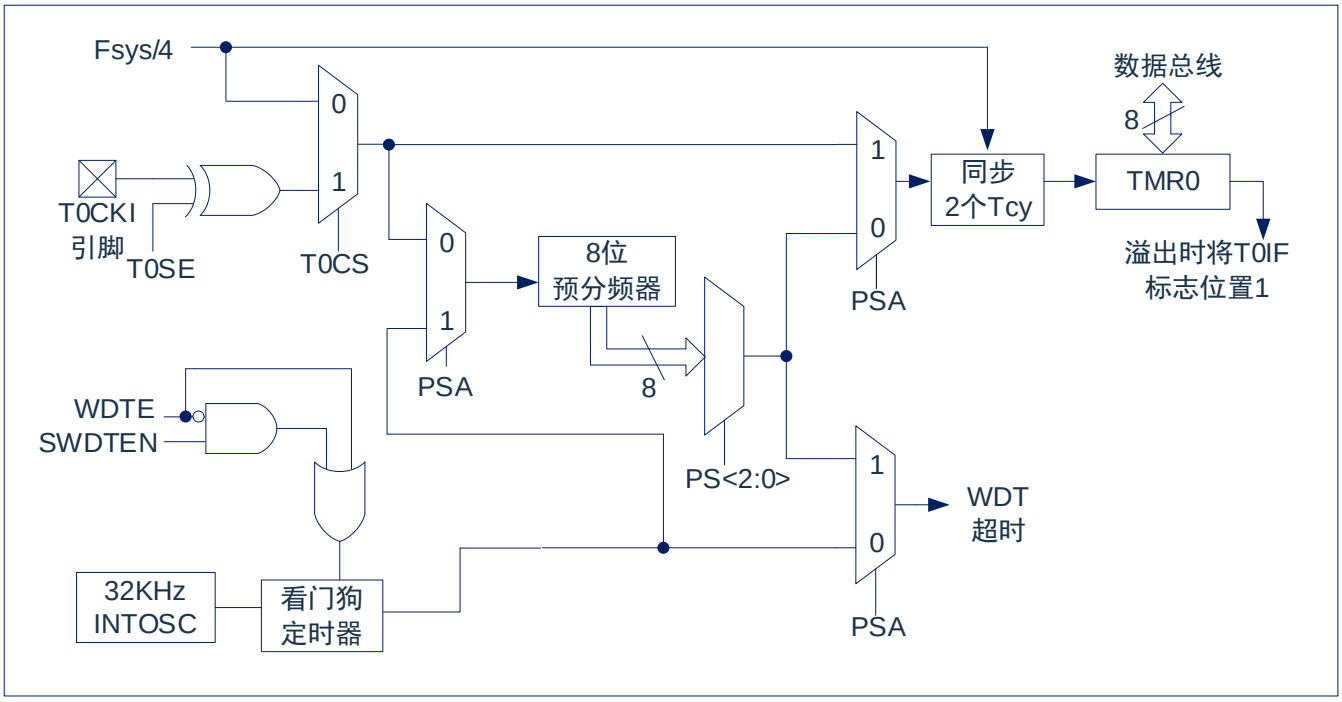


图 8-1：TIMER0/WDT 模块结构图

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

8.2 TIMER0 的工作原理

TIMER0 模块既可用作 8 位定时器也可用作 8 位计数器。

8.2.1 8 位定时器模式

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

8.2.2 8 位计数器模式

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

8.2.3 软件可编程预分频器

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

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

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

8.2.4 在 TIMER0 和 WDT 模块间切换预分频器

将预分频器分配给 TIMER0 或 WDT 后，在切换预分频比时可能会产生无意的器件复位。要将预分频器从分配给 TIMER0 改为分配给 WDT 模块时，必须执行如下所示的指令序列。

更改预分频器（TMR0-WDT）

```
CLRB    INTCON,GIE      ;关中断总使能位,避免在执行以下特定时序时进入中断程序
LDIA    B'00000111'
ORR     OPTION_REG,A    ;预分频器设置为最大值
CLR     TMR0            ;TMR0 清零
SETB    OPTION_REG,PSA  ;设置预分频器分配给 WDT
CLRWDT                  ;WDT 清零
LDIA    B'xxxx1xxx'     ;设置新的预分频器
LD      OPTION_REG,A
CLRWDT                  ;WDT 清零
SETB    INTCON,GIE      ;若程序需要用到中断,此处重新打开总使能位
```

要将预分频器从分配给 WDT 改为分配给 TIMER0 模块，必须执行以下指令序列。

更改预分频器（WDT-TMR0）

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

8.2.5 TIMER0 中断

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

注：由于在休眠状态下定时器是关闭的，所以 TIMER0 中断无法唤醒处理器。

8.3 与 TIMER0 相关寄存器

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

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

8 位定时器/计数器 TMR0(01H)

01H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR0								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	X	X	X	X	X	X	X

OPTION_REG 寄存器(81H)

81H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
OPTION_REG	----	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
读写	----	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	----	1	1	1	1	0	1	1

Bit7 未用

Bit6 INTEDG: 中断边沿选择位。
1= INT 引脚的上升沿触发中断。
0= INT 引脚的下降沿触发中断。

Bit5 T0CS: TMR0 时钟源选择位。
1= T0CKI 引脚上的跳变沿。
0= 内部指令周期时钟（F_{CPU}）。

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

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

Bit2~Bit0 PS2~PS0: 预分频参数配置位。

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

9. 定时计数器 TIMER2

9.1 TIMER2 概述

TIMER2 模块是一个 8 位定时器/ 计数器，具有以下特性：

- 8 位定时器寄存器（TMR2）；
- 8 位周期寄存器（PR2）；
- TMR2 与 PR2 匹配时中断；
- 软件可编程预分频比（1:1，1:4 和 1:16）；
- 软件可编程后分频比（1:1 至 1:16）。
- 可选 LP 振荡(外部 32.768KHz)

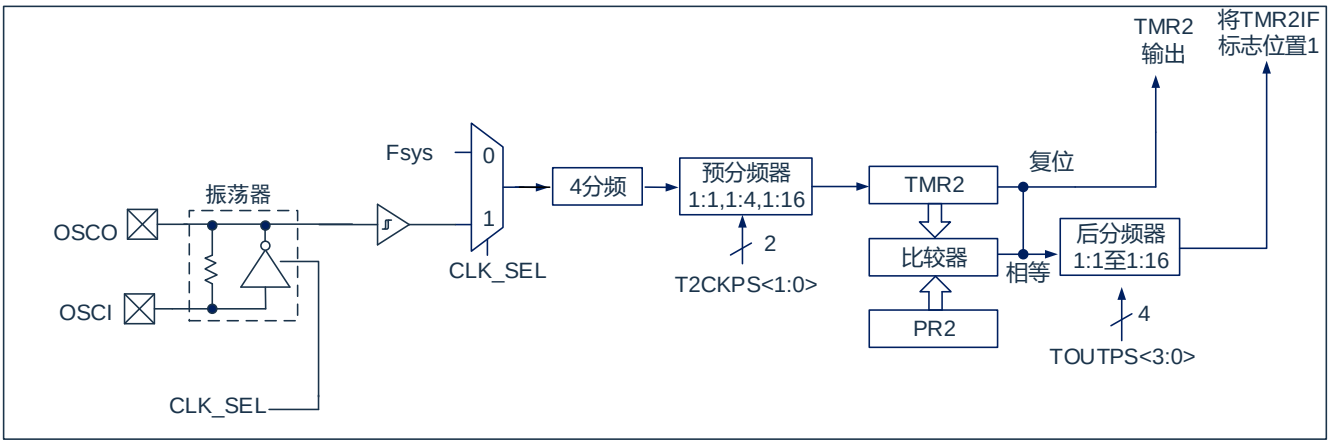


图 9-1：TIMER2 框图

9.2 TIMER2 的工作原理

TIMER2 模块的输入时钟是系统时钟（F_{sys}）或外部 32.768kHz 振荡。时钟经过 4 分频后被输入到 TIMER2 预分频器，有如下几种分频比可供选择：1:1、1:4 或 1:16。预分频器的输出随后用于使 TMR2 寄存器递增。

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

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

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

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

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

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

预分步器和后分步器计数器在以下情况下被清零：

- 对 TMR2 寄存器执行写操作
- 对 T2CON 寄存器执行写操作
- 发生任何器件复位（上电复位、看门狗定时器复位或欠压复位）。

注：
1. TMR2ON 为 0 时强制清零 TMR2 寄存器
2. PR2 寄存器的复位值为 00H，在使用 TIMER2 时，注意先设置 PR2 寄存器以避免误触发匹配

9.3 TIMER2 相关的寄存器

有 3 个寄存器与 TIMER2 相关，分别是数据存储器 TMR2、PR2 和控制寄存器 T2CON。

TIMER2 数据寄存器 TMR2(11H)

11H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR2								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	X	X	X	X	X	X	X

TIMER2 控制寄存器 T2CON(12H)

12H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T2CON	CLK_SEL	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
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 CLK_SEL: 时钟源选择；
1= 选择外部 32.768kHz 振荡/4 作为 TMR2 时钟源（休眠态可继续计数）；
0= 选择内部 F_{sys}/4 作为 TMR2 时钟源。

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

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

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

注：使能 LP 振荡，需要起振时间理论值为 32ms，实际可能更大

10. 模数转换（ADC）

10.1 ADC 概述

模数转换器（ADC）可以将模拟输入信号转换为表示该信号的一个 12 位二进制数。器件使用的模拟输入通道共用一个采样保持电路。采样保持电路的输出与模数转换器的输入相连。模数转换器采用逐次逼近法产生一个 12 位二进制结果，并将该结果保存在 ADC 结果寄存器（ADRESL 和 ADRESH）中。

ADC 参考电压可以选择内部 LDO 或 VDD。ADC 在转换完成之后可以产生一个中断。

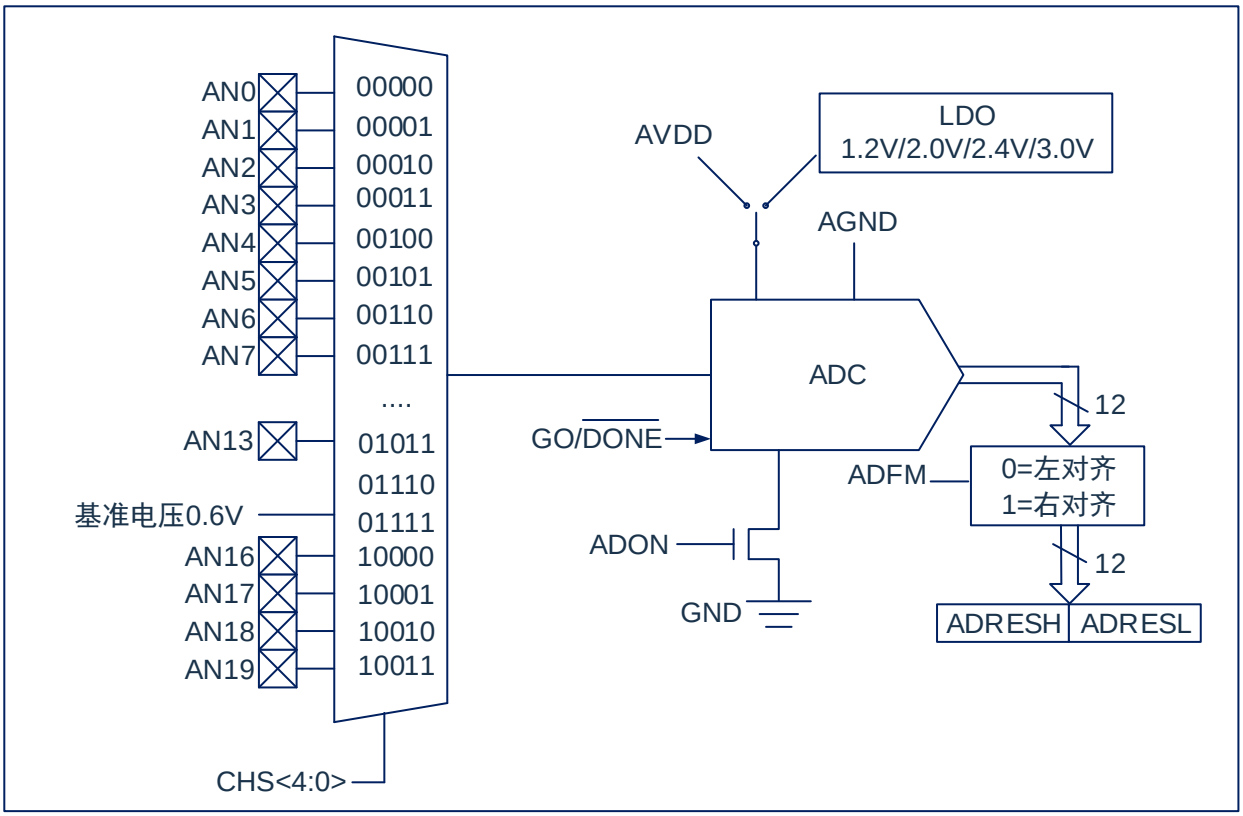


图 10-1：ADC 框图

10.2 ADC 配置

配置和使用 ADC 时，必须考虑如下因素：

- 端口配置；
- 参考电压选择；
- 通道选择；
- AD 转换时钟源；
- 中断控制；
- 结果的存储格式。

10.2.1 端口配置

ADC 既可以转换模拟信号，又可以转换数字信号。当转换模拟信号时，应该通过将相应的 TRIS 位置 1，将 I/O 引脚配置为模拟输入引脚。更多信息请参见相应的端口章节。

注：对定义为数字输入的引脚施加模拟电压可能导致输入缓冲器出现过电流。

10.2.2 通道选择

由 ADCON0 和 ADCON1 寄存器的 CHS 位决定将哪个通道连接到采样保持电路。

如果更改了通道，在下一次转换开始前需要一定的延迟。更多信息请参见“ADC 工作原理”章节。

注：
1. 打开 ADC 通道需要配置 CHS[3:0]，且将 ADCON0.ADON 置 1；
2. 如果引脚的 ADC 通道被打开(AN12/AN13 除外)，其 SMIT 将被关闭，读一直为 0；

10.2.3 ADC 内部基准电压

芯片内置基准电压 0.6V，需要检测该基准电压时，需把设置 CHS[4:0]位为 01111。

10.2.4 ADC 参考电压

ADC 的参考电压可选择内部 LDO 输出或芯片的 VDD 和 GND 提供。内部参考电压可选 1.2V/2.0V/2.4V3.0V。

当选择内部参考电压时，需要选择较慢的转换时钟，参考转换时钟章节。

注：当选择内部 LDO 作为参考电压时，ADC 有效精度会下降。检测电压越低，得到的 ADC 精度越高，建议输入电压设置为<1V。

10.2.5 转换时钟

可以通过软件设置 ADCON0 寄存器的 ADCS 位来选择转换的时钟源。有以下 4 种可能的时钟频率可供选择：

- F_{SYS}/8
- F_{SYS}/16
- F_{SYS}/32
- F_{RC}（专用内部振荡器）

完成一位转换的时间定义为 T_{ADCCLK}。一个完整的 12 位转换需要 49 个 T_{ADCCLK} 周期。
必须符合相应的 TAD 规范，才能获得正确的转换结果，下表为正确选择 ADC 时钟的示例。

注：除非使用 F_{RC}，否则系统时钟频率的任何改变都会改变 ADC 时钟的频率，从而对 ADC 转换结果产生负面影响。

ADC 时钟周期（TAD）与器件工作频率的关系（VDD=5.0V）

ADC 时钟周期		器件工作频率		
ADC 时钟源	ADCS<1:0>	16MHz	8MHz	4MHz
F_{SYS}/8	00	24.5μs	49.0μs	98.0μs
F_{SYS}/16	01	49.0μs	98.0μs	196.0μs
F_{SYS}/32	10	98.0μs	196.0μs	392.0μs
F_{RC}	11	1-3ms	1-3ms	1-3ms

10.2.6 ADC 中断

ADC 模块允许在完成模数转换后产生一个中断。ADC 中断标志位是 PIR1 寄存器中的 ADIF 位。ADC 中断允许位是 PIE1 寄存器中的 ADIE 位。ADIF 位必须用软件清零。每次转换结束后 ADIF 位都会被置 1，与是否允许 ADC 中断无关。

10.2.7 结果格式化

12 位 AD 转换的结果可采用两种格式：左对齐或右对齐。由 ADCON1 寄存器的 ADFM 位控制输出格式。

当 ADFM=0 时，AD 转换结果左对齐，AD 转换结果为 12Bit；当 ADFM=1 时，AD 转换结果右对齐，AD 转换结果为 10Bit。

10.3 ADC 工作原理

10.3.1 启动转换

要使能 ADC 模块，必须将 ADCON0 寄存器的 ADON 位置 1，将 ADCON0 寄存器的 GO/$\overline{DONE}$位置 1 开始模数转换。

注：不能用开启 AD 模块的同一指令将 GO/$\overline{DONE}$位置 1。

10.3.2 完成转换

当转换完成时，ADC 模块将：

- 清零 GO/$\overline{DONE}$位；
- 将 ADIF 标志位置 1；
- 用转换的新结果更新 ADRESH:ADRESL 寄存器。

10.3.3 终止转换

如果必须要在转换完成前终止转换，则可用软件清零 GO/$\overline{DONE}$位。不会用尚未完成的模数转换结果更新 ADRESH:ADRESL 寄存器。因此，ADRESH:ADRESL 寄存器将保持上次转换所得到的值。此外，在 AD 转换终止以后，必须经过 2 个 TAD 的延时才能开始下一次采集。延时过后，将自动开始对选定通道的输入信号进行采集。

注：器件复位将强制所有寄存器进入复位状态。复位会关闭ADC模块并且终止任何待处理的转换。

10.3.4 ADC 在休眠模式下的工作原理

注：ADC模块在休眠模式下无法唤醒芯片。

10.3.5 AD 转换步骤

如下步骤给出了使用 ADC 进行模数转换的示例：

1. 端口配置：
   - 禁止引脚输出驱动器（见 TRIS 寄存器）；
   - 将引脚配置为模拟输入引脚。
2. 配置 ADC 模块：
   - 选择 ADC 参考电压（当参考电压从 VDD 切换到内部 LDO 时，需延时 100us 以上，才能进行 AD 转换）；
   - 选择 AD 转换时钟；
   - 选择 ADC 输入通道；
   - 选择结果的格式；
   - 启动 ADC 模块。
3. 配置 ADC 中断（可选）：
   - 清零 ADC 中断标志位；
   - 允许 ADC 中断；
   - 允许外设中断；
   - 允许全局中断。
4. 等待所需的采集时间。
5. 将 GO/$\overline{DONE}$置 1 启动转换。
6. 由如下方法之一等待 AD 转换结束：
   - 查询 GO/$\overline{DONE}$位；
   - 等待 ADC 中断（允许中断）。
7. 读 ADC 结果。
8. 将 ADC 中断标志位清零（如果允许中断的话，需要进行此操作）。
9. 当 GO/$\overline{DONE}$位从 1 变 0 或 ADIF 从 0 变 1 时，需至少等待两个 TAD 时间，才能再次启动 AD 转换。

注：如果用户尝试在使器件从休眠模式唤醒后恢复顺序代码执行，则必须禁止全局中断。

例：AD 转换

```
        LDIA    B'10000000'
        LD      ADCON1,A
        SETB    TRISA,0         ;设置 PORTA.0 为输入口
        LDIA    B'11000001'
        LD      ADCON0,A
        CALL    DELAY           ;延时一段时间
        SETB    ADCON0,GO
        SZB     ADCON0,GO       ;等待 AD 转换结束
        JP      $-1
        LD      A,ADRESH        ;保存 AD 转换结果高位
        LD      RESULTH,A
        LD      A,ADRESL        ;保存 AD 转换结果低位
        LD      RESULTL,A
```

10.4 ADC 相关寄存器

主要有 4 个寄存器与 AD 转换相关，分别是控制寄存器 ADCON0，ADCON1，数据寄存器 ADRESH 和 ADRESL。

AD 控制寄存器 ADCON0(9DH)

9DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	ADON
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~Bit6 ADCS<1:0>: AD转换时钟选择位。
00= FSYS/8
01= FSYS/16
10= FSYS/32
11= F_{RC}（内部振荡器32KHz的时钟）

Bit5~Bit2 CHS<3:0>: 模拟通道选择位。与ADCON1寄存器CHS4组合CHS<4:0>
00000= AN0
00001= AN1
00010= AN2
00011= AN3
....
01101= AN13
01110= 保留
01111= 0.6V固定参考电压
10000= AN16
10001= AN17
10010= AN18
10011= AN19
其他= 保留

Bit1 GO/$\overline{DONE}$: AD转换状态位。
1= AD转换正在进行。将该位置1启动AD转换。当AD转换完成以后，该位由硬件自动清零。
当GO/$\overline{DONE}$位从1变0或ADIF从0变1时，需至少等待两个TAD时间，才能再次启动AD转换。
0= AD转换完成/或不在进行中。

Bit0 ADON: ADC使能位。
1= 使能ADC；
0= 禁止ADC，不消耗工作电流。

AD 控制寄存器 ADCON1(9CH)

9CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADCON1	ADFM	CHS4	---	---	---	LDO_EN	LDO_SEL1	LDO_SEL0
读写	R/W	R/W	---	---	---	R/W	R/W	R/W
复位值	0	0	---	---	---	0	0	0

Bit7 ADFM: AD转换结果格式选择位；
1= 右对齐；
0= 左对齐。
Bit6 CHS4 与ADCON0的CHS3~0组合使能
Bit5~Bit3 未用
Bit2 LDO_EN: 内部参考电压使能位。
1= 使能ADC内部LDO参考电压；
当选择内部LDO作参考电压时，ADC最大有效精度为8位。
0= VDD作为ADC参考电压。
Bit1~Bit0 LDO_SEL<1:0>: 参考电压选择位
00= 1.2V (注：选择此此参考电压时，转换时钟需选择F_{RC})
01= 2.0V
10= 2.4V
11= 3.0V

AD 数据寄存器高位 ADRESH(9FH)，ADFM=0

9FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRESH	ADRES11	ADRES10	ADRES9	ADRES8	ADRES7	ADRES6	ADRES5	ADRES4
读写	R	R	R	R	R	R	R	R
复位值	X	X	X	X	X	X	X	X

Bit7~Bit0 ADRES<11:4>: ADC结果寄存器位。
12位转换结果的高8位。

AD 数据寄存器低位 ADRESL(9EH)，ADFM=0

9EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRESL	ADRES3	ADRES2	ADRES1	ADRES0	---	---	---	---
读写	R	R	R	R	---	---	---	---
复位值	X	X	X	X	---	---	---	---

Bit7~Bit4 ADRES<3:0>: ADC结果寄存器位。
12位转换结果的低4位。
Bit3~Bit0 未用。

AD 数据寄存器高位 ADRESH(9FH)，ADFM=1

9FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRESH	----	----	----	----	----	----	ADRES11	ADRES10
读写	----	----	----	----	----	----	R	R
复位值	----	----	----	----	----	----	X	X

Bit7~Bit2　　未用。

Bit1~Bit0　　ADRES<11:10>:　ADC结果寄存器位。

12位转换结果的高2位。

AD 数据寄存器低位 ADRESL(9EH)，ADFM=1

9EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRESL	ADRES9	ADRES8	ADRES7	ADRES6	ADRES5	ADRES4	ADRES3	ADRES2
读写	R	R	R	R	R	R	R	R
复位值	X	X	X	X	X	X	X	X

Bit7~Bit0　　ADRES<9:2>:　ADC结果寄存器位。

12位转换结果的第9-2位。

注：在 ADFM=1 的情况下，AD 转换结果只保存 12 位结果的高 10 位，其中 ADRESH 保存高 2 位，ADRESL 保存第 9 位至第 2 位。

11. PWM 模块

11.1 PWM 概述

芯片内置一个可编程 10 位 PWM 模块，可配置为 5 路共周期、独立占空比的输出 PWM0~4；其中，PWM0/PWM1，PWM2/PWM3 可配置成带互补的正反向输出。

11.2 相关寄存器说明

PWM 控制寄存器 PWMCON0（13H)

13H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCON0	CLKDIV[2:0]			PWM4EN	PWM3EN	PWM2EN	PWM1EN	PWM0EN
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~Bit5 CLKDIV[2:0]: PWM时钟分频。
111= F_{OSC}/128
110= F_{OSC}/64
101= F_{OSC}/32
100= F_{OSC}/16
011= F_{OSC}/8
010= F_{OSC}/4
001= F_{OSC}/2
000= F_{OSC}/1

Bit4~Bit0 PWMxEN: PWMx使能位。
1= 使能PWMx。
0= 禁止PWMx。

PWM 控制寄存器 PWMCON1 (14H)

14H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCON1	PWMIO_SEL[1:0]		PWM2DTEN	PWM0DTEN	---	---	DT_DIV[1:0]	
读写	R/W	R/W	R/W	R/W	---	---	R/W	R/W
复位值	0	0	0	0	---	---	0	0

Bit7~6 PWMIO_SEL[1:0]: PWM IO选择。
11= PWM分配在A组，PWM0-RA0,PWM1-RA1,PWM2-RA2,PWM3-RA3,PWM4-RA4
10= PWM分配在B组，PWM0-RA0,PWM1-RA1,PWM2-RA2,PWM3-RB2,PWM4-RB1
01= PWM分配在C组，PWM0-RA5,PWM1-RB7,PWM2-RB6,PWM3-RB5,PWM4-RB4
00= PWM分配在D组，PWM0-RB0,PWM1-RB1,PWM2-RB3,PWM3-RB4,PWM4-RB2

Bit5 PWM2DTEN: PWM2死区使能位。
1= 使能PWM2死区功能，PWM2和PWM3组成一对互补输出。
0= 禁止PWM2死区功能。

Bit4 PWM0DTEN: PWM0死区使能位。
1= 使能PWM0死区功能，PWM0和PWM1组成一对互补输出。
0= 禁止PWM0死区功能。

Bit3~Bit2 未用。

Bit1~Bit0 DT_DIV[1:0] 死区时钟源分频。
11= $F_{OSC}/8$
10= $F_{OSC}/4$
01= $F_{OSC}/2$
00= $F_{OSC}/1$

PWM 控制寄存器 PWMCON2 (1DH)

1DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCON2	---	---	---	PWM4DIR	PWM3DIR	PWM2DIR	PWM1DIR	PWM0DIR
R/W	---	---	---	R/W	R/W	R/W	R/W	R/W
复位值	---	---	---	0	0	0	0	0

Bit7~Bit5 未用
Bit4~Bit0 PWMxDIR PWM输出取反控制位。
1= PWMx取反输出。
0= PWMx正常输出。

PWM0~PWM4 周期低位寄存器 PWMTL (15H)

15H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMTL	PWMT[7:0]							
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 PWMT[7:0]: PWM0~PWM4周期低8位。

PWM 周期高位寄存器 PWMTH (16H)

16H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMTH	---	---	PWMD4[9:8]		---	---	PWMT[9:8]	
读写	---	---	R/W	R/W	---	---	R/W	R/W
复位值	---	---	0	0	---	---	0	0

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

PWM0 占空比低位寄存器 PWMD0L (17H)

17H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD0L	PWMD0[7:0]							
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 PWMD0[7:0]: PWM0占空比低8位。

PWM1 占空比低位寄存器 PWMD1L (18H)

18H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD1L	PWMD1[7:0]							
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 PWMD1[7:0]: PWM1占空比低8位。

PWM2 占空比低位寄存器 PWMD2L (19H)。

19H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD2L	PWMD2[7:0]							
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 PWMD2[7:0]: PWM2占空比低8位。

PWM3 占空比低位寄存器 PWMD3L (1AH)

1AH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD3L	PWMD3[7:0]							
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 PWMD3[7:0]: PWM3占空比低8位。

PWM4 占空比低位寄存器 PWMD4L (1BH)

1BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD4L	PWMD4[7:0]							
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 PWMD4[7:0]: PWM4占空比低8位。

PWM0 和 PWM1 占空比高位寄存器 PWMD01H (1CH)

1CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD01H	---	---	PWMD1[9:8]		---	---	PWMD0[9:8]	
读写	---	---	R/W	---	---	----	R/W	R/W
复位值	---	---	0	---	---	----	0	0

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

PWM2 和 PWM3 占空比高位寄存器 PWMD23H (0EH)

0EH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD23H	---	---	PWMD3[9:8]		---	---	PWMD2[9:8]	
读写	---	---	R/W	---	---	----	R/W	R/W
复位值	---	---	0	---	---	----	0	0

Bit7~Bit6 未用。
Bit5~Bit4 PWMD3[9:8]: PWM3占空比高2位。
Bit3~Bit2 未用。
Bit1~Bit0 PWMD2[9:8]: PWM2占空比高2位。

PWM0 和 PWM1 死区时间寄存器 PWM01DT (0FH)

0FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWM01DT	---	---	PWM01DT[5:0]					
读写	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	0	0	0	0	0

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

PWM2 和 PWM3 死区时间寄存器 PWM23DT (10H)

10H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWM23DT	---	---	PWM23DT[5:0]					
读写	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	0	0	0	0	0

Bit7~Bit6 未用。
Bit5~Bit0 PWM23DT[5:0]: PWM2和PWM3死区时间。

11.3 PWM 周期

PWM 周期是通过写 PWMTH 和 PWMTL 寄存器来指定的。

公式 1：PWM 周期计算公式：

$$PWM\ 周期=[PWMT+1]*Tosc*(CLKDIV\ 分频值)$$

注：$T_{OSC}=1/F_{OSC}$

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

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

11.4 PWM 占空比

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

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

公式 2：脉冲宽度计算公式：

$$脉冲宽度 = (PWMDx[9:0]+1)*Tosc*(CLKDIV\ 分频值)$$

公式3：PWM占空比计算公式：

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

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

11.5 系统时钟频率的改变

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

11.6 可编程的死区延时模式

可以通过设置 PWMxDT_EN 使能互补输出模式，使能互补输出后自动使能死区延时功能。

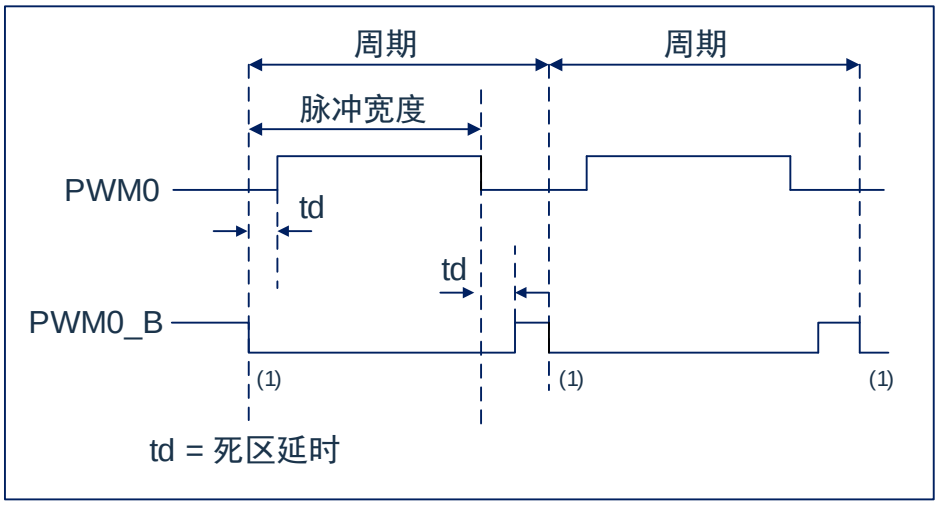


图11-1：PWM死区延时输出示例

死区时间计算公式为：

$$td=(PWMxxDT[5:0]+1)*T_{OSC}*(DT\_DIV\ 分频值)$$

11.7 PWM 设置

使用 PWM 模块时应该执行以下步骤：

1. 设置 IO_SEL 控制位，选择 PWM 输出 IO 口。
2. 通过将相应的 TRIS 位置 1，使之成为输入引脚。
3. 通过装载 PWMTH，PWMTL 寄存器设置 PWM 周期。
4. 通过装载 PWMDxL，PWMDxxH 寄存器设置 PWM 占空比。
5. 若需要使用互补输出模式，需设置 PWMCON1[5:4]位，并装载 PWMxxDT 寄存器设置死区时间。
6. 清零 PWMIF 标志位
7. 设置 PWMCN0[4:0]位以使能相应 PWM 输出。
8. 在新的 PWM 周期开始后，使能 PWM 输出：
   - 等待 PWMIF 位置 1；
   - 通过将相应的 TRIS 位清零，使能 PWM 引脚输出驱动器。

12. 程序 EEPROM 和程序存储器控制

12.1 概述

该系列中器件具有 4K 字节的程序存储器，地址范围从 0000h 到 0FFFh，在所有地址范围内都是只读的；器件具有 32 字节的程序 EEPROM，地址范围为 0000h 到 001Fh，在所有地址范围内都是可读写的。

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（SFR）对其进行间接寻址。共有 6 个 SFR 寄存器用于访问这些存储器：

- EECON1
- EECON2
- EEDAT
- EEDATH
- EEADR
- EEADRH

当访问程序 EEPROM 时，EEDAT 寄存器存放 8 位读写的数据，而 EEADR 寄存器存放被访问的程序 EEPROM 单元的地址。

当访问器件的程序存储器时，EEDAT 和 EEDATH 寄存器形成一个双字节字用于保存要读的 16 位数据，EEADR 和 EEADRH 寄存器组成一个双字节字用于保存待读取的 12 位 EEPROM 单元地址。

程序存储器允许以字为单位读取。程序 EEPROM 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 仍可继续读写程序 EEPROM 和程序存储器。代码保护时，器件编程器将不再能访问程序 EEPROM 或程序存储器。

12.2 相关寄存器

12.2.1 EEADR 和 EEADRH 寄存器

EEADR 和 EEADRH 寄存器能寻址最大 32 字节的程序 EEPROM 或最大 4K 字的程序存储器。

当选择程序存储器地址值时，地址的高字节被写入 EEADRH 寄存器而低字节被写入 EEADR 寄存器。当选择程序 EEPROM 地址值时，只将地址的低字节写入 EEADR 寄存器。

12.2.2 EECON1 和 EECON2 寄存器

EECON1 是访问程序 EEPROM 的控制寄存器。

控制位 EEPGD 决定访问的是程序存储器还是程序 EEPROM。该位被清零时，和复位时一样，任何后续操作都将针对程序 EEPROM 进行。该位置 1 时，任何后续操作都将针对程序存储器进行。程序存储器是只读的。

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，允许对程序 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 PIR1 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 寄存器仅在执行程序 EEPROM 写序列时使用。

EEPROM 数据寄存器 EEDAT(8EH)

8EH	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	X	X	X	X	X	X	X	X

Bit7~Bit0　　EEDAT<7:0>:　要从程序EEPROM中读取或向程序EEPROM写入数据的低8位，或者要从程序存储器中读取数据的低8位。

EEPROM 地址寄存器 EEADR(90H)

90H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0
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　　EEADR<7:0>:　指定程序EEPROM读/写操作的地址的低8位，或程序存储器读操作的地址的低8位。

EEPROM 数据寄存器 EEDATH(8FH)

8FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEDATH	EEDATH7	EEDATH6	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0
读写	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

Bit7~Bit0　　EEDATH<7:0>:　从程序EEPROM/程序存储器读出的数据的高8位。

EEPROM 地址寄存器 EEADRH(96H)

96H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEADRH	---	---	---	---	EEADRH3	EEADRH2	EEADRH1	EEADRH0
读写	---	---	---	---	R/W	R/W	R/W	R/W
复位值	---	---	---	---	0	0	0	0

Bit7~Bit4　　未用，读为0。
Bit3~Bit0　　EEADRH<3:0>:　指定程序存储器读操作的高4位地址。

EEPROM 控制寄存器 EECON1(8CH)

8CH	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:　程序/程序EEPROM选择位；
　　1=　操作程序存储器；
　　0=　操作程序EEPROM。
Bit6~Bit4　　未用。
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=　禁止写入存储器。
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
　　0=　写周期完成。
Bit0　　RD:　读控制位；
　　1=　启动存储器读操作（由硬件清零RD，用软件只能将RD位置1，但不能清零）；
　　0=　不启动存储器读操作。

12.3 读程序 EEPROM

要读取程序 EEPROM 单元，用户必须将地址写入 EEADR 寄存器，清零 EECON1 寄存器的 EEPGD 控制位，然后将控制位 RD 置 1。一旦设置好读控制位，程序 EEPROM 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(1)。在紧接下来的一个时钟周期，程序 EEPROM 相应地址的值会被锁存到 EEDAT 寄存器中，用户可在随后的指令中读取这两个寄存器。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。

例：读程序 EEPROM

```
        LD      A,EE_ADD        ;将要读取的地址放入 EEADR 寄存器
        LD      EEADR,A
        CLRB    EECON1,EEPGD    ;选择程序 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
```

12.4 写程序 EEPROM

要写程序 EEPROM 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 EEDAT 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 位由硬件清零并且 EE 写完成中断标志位（EEIF）置 1。用户可以允许此中断或查询此位。EEIF 必须用软件清零。

> 注：在写程序 EEPROM 期间，CPU 会停止工作，需在写操作开始前执行 CLRWDT 命令，以避免在此期间 WDT 溢出复位芯片。

例：写程序 EEPROM

```
        LD      A,EE_ADDL       ;将要写入的地址放入 EEADR 寄存器
        LD      EEADR,A
        LD      A,EE_DATAL      ;将要写入的低 8 位数据放入 EEDAT 寄存器
        LD      EEDAT,A
        LD      A,EE_DATAH      ;将要写入的高 8 位数据放入 EEDATH 寄存器
        LD      EEDATH,A
        CLRWDT
        CLRB    EECON1,EEPGD
        SETB    EECON1,WREN     ;允许写操作
        CLRB    INTCON,GIE      ;关闭中断
        SZB     INTCON,GIE      ;确保中断已关闭
        JP      $-2
        LDIA    055H            ;给 EECON2 写 55H
        LD      EECON2,A
        LDIA    0AAH            ;给 EECON2 写 0AAH
        LD      EECON2,A
        SETB    EECON1,WR       ;使能写信号
        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
        SETB    INTCON,GIE
```

12.5 读程序存储器

要读取程序存储器单元，用户必须将地址的高位和低位分别写入 EEADR 和 EEADRH 寄存器，将 EECON1 寄存器的 EEPGD 位置 1，然后将控制位 RD 置 1。一旦设置好读控制位，程序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。在紧接下来的一个时钟周期，程序存储器相应地址的值会被锁存到 EEDAT 和 EEDATH 寄存器中，用户可在随后的指令中读取这两个寄存器。EEDAT 和 EEDATH 寄存器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：

1) 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。
2) 当 EEPGD=1 时如果 WR 位置 1，它会立即复位为 0，而不执行任何操作。

例：读闪存程序存储器

```
        LD      A,EE_ADDL       ;将要读取的地址放入 EEADR 寄存器
        LD      EEADR,A
        LD      A,EE_ADDH       ;将要读取的地址高位放入 EEADRH 寄存器
        LD      EEADRH,A
        SETB    EECON1,EEPGD    ;选择操作程序存储器
        SETB    EECON1,RD       ;允许读操作
        NOP
        NOP
        LD      A,EEDAT         ;保存读取的数据
        LD      EE_DATL,A
        LD      A,EEDATH
        LD      EE_DATH,A
```

12.6 写程序存储器

程序存储器是只读的，不可写。

12.7 程序 EEPROM 操作注意事项

12.7.1 关于程序 EEPROM 的烧写时间

程序 EEPROM 烧写时间是固定的，约为 4.6ms，并且在烧写期间 CPU 停止工作，程序需要做好相关处理。

12.7.2 写校验

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

12.7.3 避免误写的保护

有些情况下，用户可能不希望向程序 EEPROM 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

13. LVD 低电压检测

13.1 LVD 模块概述

CMS89F12x 系列单片机具有低电压检测功能，可以用于监测电源电压，如果电源电压低于设定的值，可以产生一个中断信号；程序可实时读取 LVD 输出标志位。

13.2 LVD 相关的寄存器

LVD 控制寄存器 LVDCON(11FH)

11FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
LVDCON	LVD_RES	—	—	—	LVD_SEL[2:0]			LVDEN
R/W	R	—	—	—	R/W	R/W	R/W	R/W
复位值	X	—	—	—	0	0	0	0

Bit7	LVD_RES:	LVD 输出结果
	0=	VDD>设定的 LVD 电压；
	1=	VDD<设定的 LVD 电压；
Bit6~Bit4	未用	
Bit3~Bit1	LVD_SEL[2:0]:	LVD 电压选择
	000=	2.2V；
	001=	2.4V；
	010=	2.7V；
	011=	3.0V；
	100=	3.3V；
	101=	3.7V；
	110=	4.0V；
	111=	4.3V；
Bit0	LVDEN:	LVD 使能位
	0=	禁止；
	1=	使能；

13.3 LVD 操作

通过设定 LVDCON 寄存器中的 LVD 电压值，使能 LVDEN 之后，当电源电压低于设定的电压值，LVDCON 寄存器中的 LVD_RES 位被置高。当 LVD 模块使能后，需要延时 1ms 的时间才能够读取 LVD_RES 位，因为内部做了滤波处理，以减少在 VLVD 电压值附近时，LVD 输出结果的频繁波动。

LVD 模块有自己的中断标志位，当设定好相关的中断使能位，电源电压低于设定的电压值时，会产生 LVD 中断，中断标志位 LVDIF 将被置 1，中断产生。LVD 不能用于中断唤醒模式。

14. 通用同步/异步收发器（USART）

通用同步/异步收发器（USART）模块是一个串行 I/O 通信外设。该模块包括所有执行与器件程序执行无关的输入或输出串行数据传输所必需的时钟发生器、移位寄存器和数据缓冲器。USART 也可称为串行通信接口（SerialCommunicationsInterface，SCI），它可被配置为能与 CRT 终端和个人计算机等外设通信的全双工异步系统；也可以被配置为能与 A/D 或 D/A 集成电路、串行 EEPROM 等外设或其他单片机通信的半双工同步系统。与之通信的单片机通常不具有产生波特率的内部时钟，它需要主控同步器件提供外部时钟信号。

USART 模块包含如下功能：

- 全双工异步发送和接收
- 单字符输出缓冲器
- 双字符输入缓冲器
- 接收到字符的帧错误检测
- 半双工同步从动模式
- 可将字符长度编程为 8 位或 9 位
- 输入缓冲溢出错误检测
- 半双工同步主控模式
- 同步模式下，可编程时钟极性

以下图 16-1 和图 16-2 为 USART 收发器的框图。

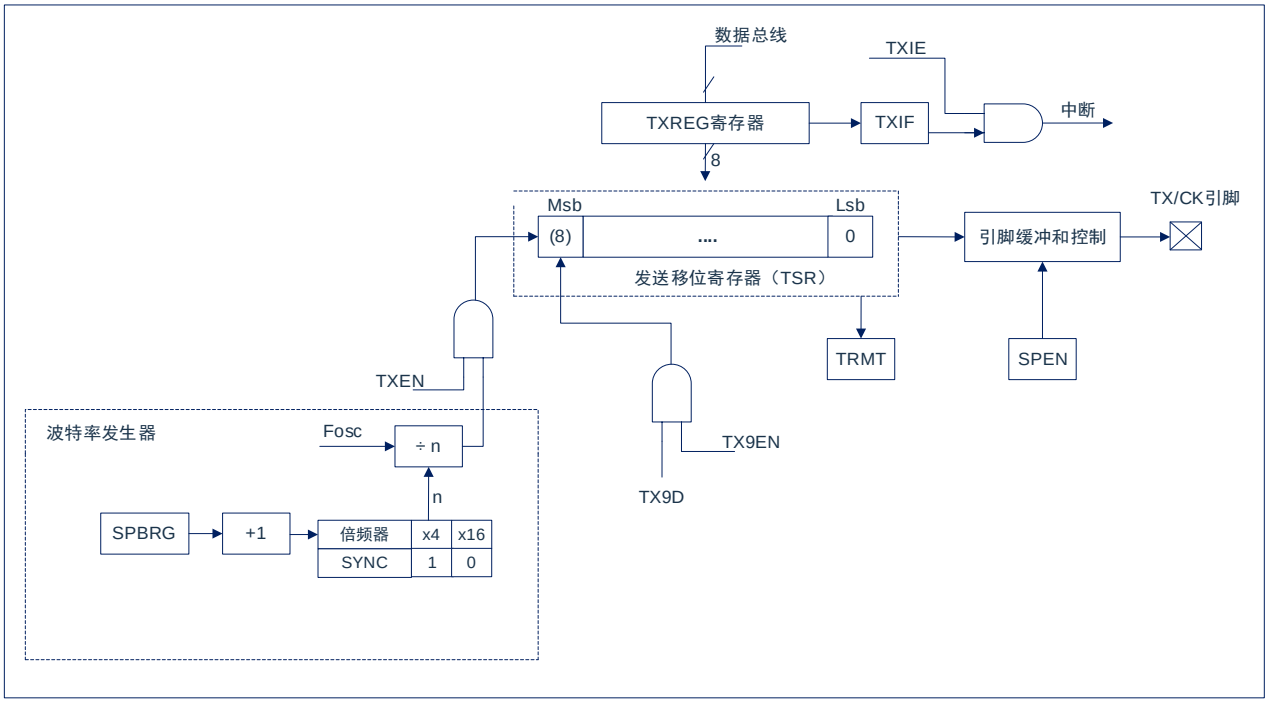


图16-1：USART发送框图

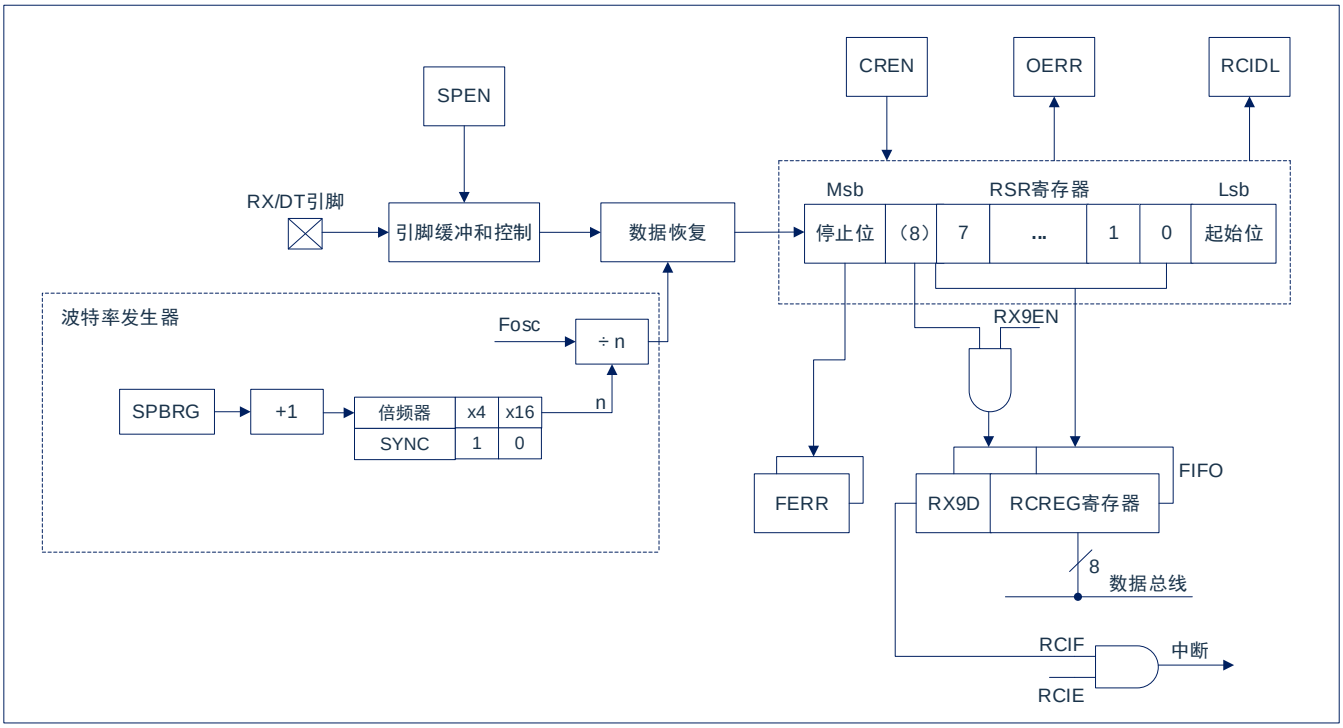


图16-2：USART接收框图

USART 模块的操作是通过 3 个寄存器控制的：

- 发送状态和控制寄存器（TXSTA）
- 接收状态和控制寄存器（RCSTA）

14.1 USART 异步模式

USART 使用标准不归零码（non-return-to-zero，NRZ）格式发送和接收数据。使用 2 种电平实现 NRZ：

代表 1 数据位的 VOH 标号状态（markstate），和代表 0 数据位的 VOL 空格状态（spacestate）。采用 NRZ 格式连续发送相同值的数据位时，输出电平将保持该位的电平，而不会在发送完每个位后返回中间电平值。NRZ 发送端口在标号状态空闲。每个发送的字符都包括一个起始位，后面跟有 8 个或 9 个数据位和一个或多个终止字符发送的停止位。起始位总是处于空格状态，停止位总是处于标号状态。最常用的数据格式为 8 位。每个发送位的持续时间为 1/（波特率）。片上专用 8 位/16 位波特率发生器可用于通过系统振荡器产生标准波特率频率。

USART 首先发送和接收 LSb。USART 的发送器和接收器在功能上是相互独立的，但采用相同的数据格式和波特率。硬件不支持奇偶校验，但可以用软件实现（奇偶校验位是第 9 个数据位）。

14.1.1 USART 异步发生器

图 16-1 所示为 USART 发送器的框图。发送器的核心是串行发送移位寄存器（TSR），该寄存器不能由软件直接访问。TSR 从 TXREG 发送缓冲寄存器获取数据。

14.1.1.1 使能发送器

通过配置如下三个控制位使能 USART 发送器，以用于异步操作：

- TXEN=1
- SYNC=0
- SPEN=1

假设所有其他 USART 控制位处于其默认状态。

将 TXSTA 寄存器的 TXEN 位置 1，使能 USART 发送器电路。将 TXSTA 寄存器的 SYNC 位清零，将 USART 配置用于异步操作。

注：
1. 当将 SPEN 位和 TXEN 位置 1，SYNC 位清零，TX/CKI/O 引脚被自动配置为输出引脚，无需考虑相应 TRIS 位的状态。
2. 当将 SPEN 位和 CREN 位置 1，SYNC 位清零，RX/DTI/O 引脚被自动配置为输入引脚，无需考虑相应 TRIS 位的状态。

14.1.1.2 发送数据

向 TXREG 寄存器写入一个字符，以启动发送。如果这是第一个字符，或者前一个字符已经完全从 TSR 中移出，TXREG 中的数据会立即发送给 TSR 寄存器。如果 TSR 中仍保存全部或部分前一字符，新的字符数据将保存在 TXREG 中，直到发送完前一字符的停止位为止。然后，在停止位发送完毕后经过一个 TCY，TXREG 中待处理的数据将被传输到 TSR。当数据从 TXREG 传输至 TSR 后，立即开始进行起始位、数据位和停止位序列的发送。

14.1.1.3 发送中断标志

只要使能 USART 发送器且 TXREG 中没有待发送数据，就将 PIR1 寄存器的 TXIF 中断标志位置 1。换句话说，只有当 TSR 忙于处理字符和 TXREG 中有排队等待发送的新字符时，TXIF 位才处于清零状态。写 TXREG 时，不立即清零 TXIF 标志位。TXIF 在写指令后的第 2 个指令周期清零。在写 TXREG 后立即查询 TXIF 会返回无效结果。TXIF 为只读位，不能由软件置 1 或清零。

可通过将 PIE1 寄存器的 TXIE 中断允许位置 1 允许 TXIF 中断。然而，只要 TXREG 为空，不管 TXIE 允许位的状态如何都会将 TXIF 标志位置 1。

如果要在发送数据时使用中断，只在有待发送数据时，才将 TXIE 位置 1。当将待发送的最后一个字符写入 TXREG 后，将 TXIE 中断允许位清零。

14.1.1.4 TSR 状态

TXSTA 寄存器的 TRMT 位指示 TSR 寄存器的状态。TRMT 位为只读位。当 TSR 寄存器为空时，TRMT 位被置 1，当有字符从 TXREG 传输到 TSR 寄存器时，TRMT 被清零。TRMT 位保持清零状态，直到所有位从 TSR 寄存器移出为止。没有任何中断逻辑与该位有关，所以用户必须查询该位来确定 TSR 位的状态。

注：TSR 寄存器并未映射到数据存储器中，因此用户不能直接访问它。

14.1.1.5 发送 9 位字符

USART 支持 9 位字符发送。当 TXSTA 寄存器的 TX9EN 位置 1 时，USART 将移出每个待发送字符的 9 位。TXSTA 寄存器的 TX9D 位为第 9 位，即最高数据位。当发送 9 位数据时，必须在将 8 个最低位写入 TXREG 之前，写 TX9D 数据位。在写入 TXREG 寄存器后会立即将 9 个数据位传输到 TSR 移位寄存器。

14.1.1.6 设置异步发送

1. 初始化 SPBRG 寄存器，以获得所需的波特率
   （请参见“USART 波特率发生器（BRG）”章节。
2. 通过将 SYNC 位清零并将 SPEN 位置 1 使能异步串口。
3. 如果需要 9 位发送，将 TX9EN 控制位置 1。当接收器被设置为进行地址检测时，将数据位的第 9 位置 1，指示 8 个最低数据位为地址。
4. 将 TXEN 控制位置 1，使能发送；这将导致 TXIF 中断标志位置 1。
5. 如果需要中断，将 PIE1 寄存器中的 TXIE 中断允许位置 1；如果 INTCON 寄存器的 GIE 和 PEIE 位也置 1 将立即产生中断。
6. 若选择发送 9 位数据，第 9 位应该被装入 TX9D 数据位。
7. 将 8 位数据装入 TXREG 寄存器开始发送数据。

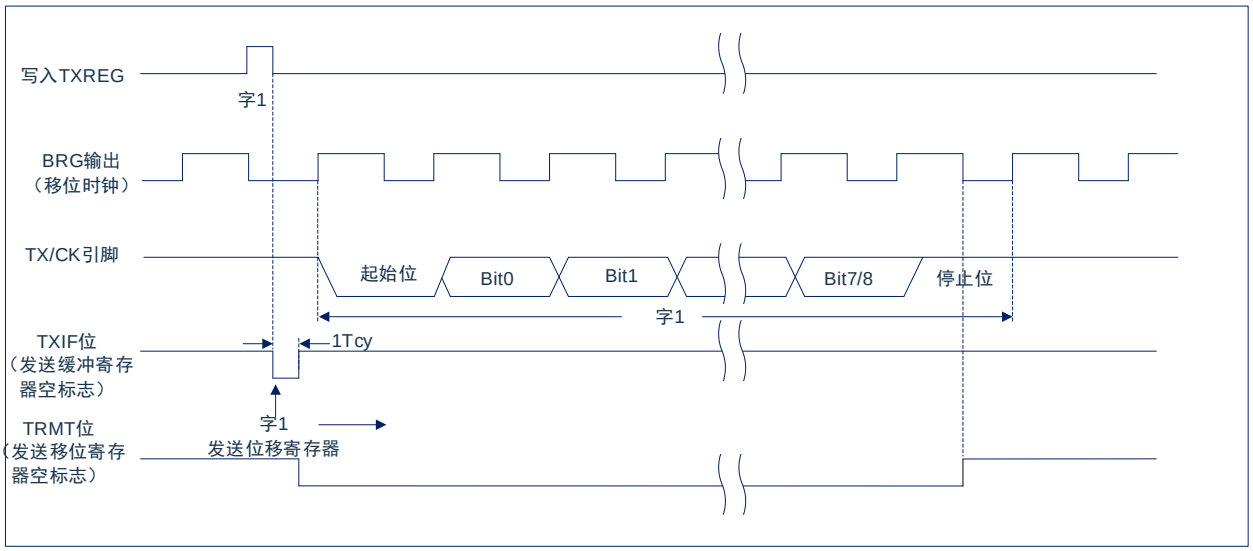


图 16-3：异步发送

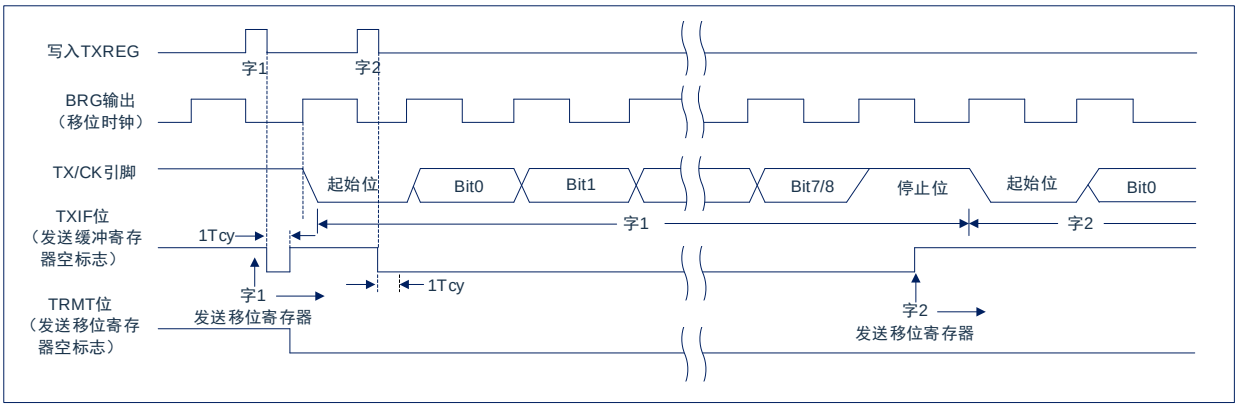


图 16-4：异步发送（背靠背）

注：本时序图显示了两次连续的发送。

14.1.2 USART 异步接收器

异步模式通常用于 RS-232 系统。图 16-2 给出了接收器的框图。在 RX/DT 引脚上接收数据和驱动数据恢复电路。数据恢复电路实际上是一个以 16 倍波特率为工作频率的高速移位器，而串行接收移位寄存器（ReceiveShiftRegister，RSR）则以比特率工作。当字符的全部 8 位或 9 位数据位被移入后，立即将它们传输到一个 2 字符的先入先出（FIFO）缓冲器。FIFO 缓冲器允许接收 2 个完整的字符和第 3 个字符的起始位，然后必须由软件将接收到的数据提供给 USART 接收器。FIFO 和 RSR 寄存器不能直接由软件访问。通过 RCREG 寄存器访问接收到的数据。

14.1.2.1 使能接收器

通过配置如下三个控制位使能 USART 接收器，以用于异步操作。

- CREN=1
- SYNC=0
- SPEN=1

假设所有其他 USART 控制位都处于默认状态。将 RCSTA 寄存器的 CREN 位置 1，使能 USART 接收器电路。将 TXSTA 寄存器的 SYNC 位清零，配置 USART 以用于异步操作。

注：
1. 当将 SPEN 位和 TXEN 位置 1，SYNC 位清零，TX/CKI/O 引脚被自动配置为输出引脚，无需考虑相应 TRIS 位的状态。
2. 当将 SPEN 位和 CREN 位置 1，SYNC 位清零，RX/DTI/O 引脚被自动配置为输入引脚，无需考虑相应 TRIS 位的状态。

14.1.2.2 接收数据

接收器数据恢复电路在第一个位的下降沿开始接收字符。第一个位，通常称为起始位，始终为 0。由数据恢复电路计数半个位时间，到起始位的中心位置，校验该位是否仍为零。如果该位不为零，数据恢复电路放弃接收该字符，而不会产生错误，并且继续查找起始位的下降沿。如果起始位零校验通过，则数据恢复电路计数一个完整的位时间，到达下一位的中心位置。由择多检测电路对该位进行采样，将相应的采样结果 0 或 1 移入 RSR。重复该过程，直到完成所有数据位的采样并将其全部移入 RSR 寄存器。测量最后一个位的时间并采样其电平。此位为停止位，总是为 1。如果数据恢复电路在停止位的位置采样到 0，则该字符的帧错误标志将置 1，反之，该字符的帧错误标志会清零。

当接收到所有数据位和停止位后，RSR 中的字符会被立即传输到 USART 的接收 FIFO 并将 PIR1 寄存器的 RCIF 中断标志位置 1。通过读 RCREG 寄存器将 FIFO 最顶端的字符移出 FIFO。

注：如果接收 FIFO 溢出，则不能再继续接收其他字符，直到溢出条件被清除。

14.1.2.3 接收中断

只要使能 USART 接收器且在接收 FIFO 中没有未读数据，PIR1 寄存器中的 RCIF 中断标志位就会置 1。RCIF 中断标志位为只读，不能由软件置 1 或清零。

通过将下列所有位均置 1 来允许 RCIF 中断：

- PIE1 寄存器的 RCIE 中断允许位；
- INTCON 寄存器的 PEIE 外设中断允许位；
- INTCON 寄存器的 GIE 全局中断允许位。

如果 FIFO 中有未读数据，无论中断允许位的状态如何，都会将 RCIF 中断标志位置 1。

14.1.2.4 接收帧错误

接收 FIFO 缓冲器中的每个字符都有一个相应的帧错误状态位。帧错误指示未在预期的时间内接收到停止位。

由 RCSTA 寄存器的 FERR 位获取帧错误状态。必须在读 RCREG 寄存器之后读 FERR 位。

帧错误（FERR=1）并不会阻止接收更多的字符。无需清零 FERR 位。

清零 RCSTA 寄存器的 SPEN 位会复位 USART，并强制清零 FERR 位。帧错误本身不会产生中断。

注：如果接收 FIFO 缓冲器中所有接收到的字符都有帧错误，重复读 RCREG 不会清零 FERR 位。

14.1.2.5 接收溢出错误

接收 FIFO 缓冲器可以保存 2 个字符。但如果在访问 FIFO 之前，接收到完整的第 3 个字符，则会产生溢出错误。此时，RCSTA 寄存器的 OERR 位会置 1。可以读取 FIFO 缓冲器内的字符，但是在错误清除之前，不能再接收其他字符。可以通过清零 RCSTA 寄存器的 CREN 位或通过清零 RCSTA 寄存器的 SPEN 位使 USART 复位来清除错误。

14.1.2.6 接收 9 位字符

USART 支持 9 位数据接收。将 RCSTA 寄存器的 RX9EN 位置 1 时，USART 将接收到的每个字符的 9 位移入 RSR。必须在读 RCREG 中的低 8 位之后，读取 RX9D 数据位。

14.1.2.7 异步接收设置

1. 初始化 SPBRG 寄存器，以获得所需的波特率。
   （请参见“USART 波特率发生器（BRG）”章节。
2. 将 SPEN 位置 1，使能串行端口。必须清零 SYNC 位以执行异步操作。
3. 如果需要中断，将 PIE1 寄存器中的 RCIE 位和 INTCON 寄存器的 GIE 和 PEIE 位置 1。
4. 如果需要接收 9 位数据，将 RX9EN 位置 1。
5. 将 CREN 位置 1 使能接收。
6. 当一个字符从 RSR 传输到接收缓冲器时，将 RCIF 中断标志位置 1。如果 RCIE 中断允许位也置 1 还将产生中断。
7. 读 RCREG 寄存器，从接收缓冲器获取接收到的 8 个低数据位。
8. 读 RCSTA 寄存器获取错误标志位和第 9 位数据位（如果使能 9 位数据接收）。
9. 如果发生溢出，通过清零 CREN 接收器使能位清零 OERR 标志。

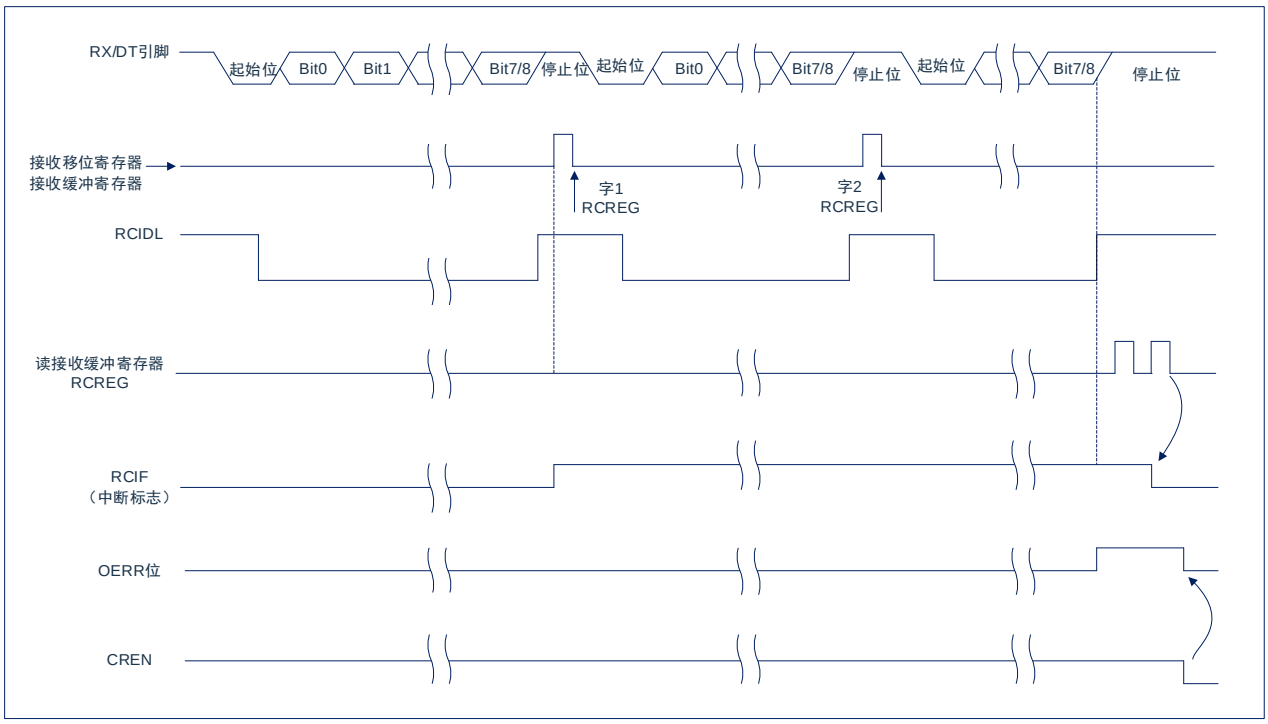


图 16-5：异步接收

注：本时序图显示出了在 RX 输入引脚接收三个字的情况，在第 3 个字后读取 RCREG（接收缓冲器），导致 OERR（溢出）位置 1。

14.2 异步操作时的时钟准确度

由厂家校准内部振荡电路（INTOSC）的输出。但在 VDD 或温度变化时，INTOSC 会发生频率漂移，从而会直接影响异步波特率。可通过以下方法调整波特率时钟，但需要某种类型的参考时钟源。

14.3 USART 相关寄存器

TXSTA：发送状态和控制寄存器(117H)

117H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TXSTA	CSRC	TX9EN	TXEN(1)	SYNC	SCKP	STOPBIT	TRMT	TX9D
读写	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
复位值	0	0	0	0	0	0	1	0

Bit7 CSRC: 时钟源选择位；
异步模式: 任意值；
同步模式:
1=主控模式（由内部BRG产生时钟信号）；
0=从动模式（由外部时钟源产生时钟）。

Bit6 TX9EN: 9位发送使能位；
1= 选择9位发送；
0= 选择8位发送。

Bit5 TXEN: 发送使能位(1)；
1= 使能发送；
0= 禁止发送。

Bit4 SYNC: USART模式选择位；
1= 同步模式；
0= 异步模式。

Bit3 SCKP: 同步时钟极性选择位。
异步模式:
1=将数据字符的电平取反后发送到TX/CK引脚；
0=直接将数据字符发送到TX/CK引脚。
同步模式:
0=在时钟上升沿传输数据；
1=在时钟下降沿传输数据。

Bit2 STOPBIT: 停止位选择（仅对异步发送有效），当通过判断TRMT=1来送数据时，此位需写0。
1= 1位停止位；
0= 2位停止位。

Bit1 TRMT: 发送移位寄存器状态位；
1= TSR为空；
0= TSR为满。

Bit0 TX9D: 发送数据的第9位。
可以是地址/数据位或奇偶校验位。

注：
1) 同步模式下，SREN/CREN 会颠覆 TXEN 的值。
2) 当通过判断 TRMT=1 来送数据时，STOPBIT 需写 0。

RCSTA：接收状态和控制寄存器(118H)

118H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RCSTA	SPEN	RX9EN	SREN	CREN	RCIDL	FERR	OERR	RX9D
读写	R/W	R/W	R/W	R/W	R	R/W	R	R
复位值	0	0	0	0	1	0	0	0

Bit7 SPEN: 串行端口使能位；
1= 使能串行端口（将RX/DT和TX/CK引脚配置为串行端口引脚）；
0= 禁止串行端口（保持在复位状态）。

Bit6 RX9EN: 9位接收使能位；
1= 选择9位接收；
0= 选择8位接收。

Bit5 SREN: 单字节接收使能位。
异步模式: 任意值。
同步主控模式:
1=使能单字节接收；
0=禁止单字节接收。
接收完成后清零该位。
同步从动模式: 任意值。

Bit4 CREN: 连续接收使能位。
异步模式:
1=使能接收；
0=禁止接收。
同步模式:
1=使能连续接收直到清零CREN使能位（CREN覆盖SREN）；
0=禁止连续接收。

Bit3 RCIDL: 接收空闲标志位。
异步模式: 1=接收器空闲；
0=已接收到起始位，接收器正在接收数据。
同步模式: 任意值。

Bit2 FERR: 帧错误位。
1= 帧错误（可通过读RCREG寄存器更新并接收下一个有效字节）；
0= 没有帧错误。

Bit1 OERR: 溢出错误位。
1= 溢出错误（可通过清零CREN位清零）；
0= 没有溢出错误。

Bit0 RX9D: 接收到数据的第9位。
此位可以是地址/数据位或奇偶校验位，必须由用户固件计算得到。

14.4 USART 波特率发生器（BRG）

波特率发生器（BRG）是一个 8 位，专用于支持 USART 的异步和同步工作模式。

SPBRG 寄存器对决定自由运行的波特率定时器的周期。

表 16-1 包含了计算波特率的公式。公式 1 为一个计算波特率和波特率误差的示例。

表 16-1 中给出了已经计算好的各种异步模式下的典型波特率和波特率误差值，可便于您使用。

向 SPBRG 寄存器对写入新值会导致 BRG 定时器复位（或清零）。这可以确保 BRG 无需等待定时器溢出就可以输出新的波特率。

如果系统时钟在有效的接收过程中发生了变化，可能会产生接收错误或导致数据丢失。为了避免此问题，应该检查 RCIDL 位的状态以确保改变系统时钟之前，接收操作处于空闲状态。

公式 1：计算波特率误差

对于 F_{SYS} 为 8MHz，目标波特率为 9600bps，异步模式采用 8 位 BRG 的器件：

$$目标波特率=\frac{F_{OSC}}{16([SPBRG]+1)}$$

求解 SPBRG：

$$X=\frac{\frac{F_{OSC}}{目标波特率}}{16}-1=\frac{\frac{8000000}{9600}}{16}-1=[51.08]=51$$

$$计算波特率=\frac{8000000}{16\ （51+1）}=9615$$

$$误差=\frac{计算波特率-目标波特率}{目标波特率}=\frac{（9615-9600）}{9600}=0.16\%$$

表 16-1：波特率公式

配置位	BRG/USART 模式	波特率公式
SYNC		
0	8 位/异步	$F_{OSC}/[16(n+1)]$
1	8 位/同步	$F_{OSC}/[4(n+1)]$

说明：n= SPBRG 寄存器的值。

表 16-2：异步模式下的波特率

目标波特率	SYNC=0					
	F_{OSC}=8.00MHz			F_{OSC}=16.00MHz		
	实际波特率	误差（%）	SPBRG 值	实际波特率	误差（%）	SPBRG 值
2400	2404	0.16	207	----	----	----
9600	9615	0.16	51	9615	0.16	103
10417	10417	0	47	10417	0	95
19200	19230	0.16	25	19230	0.16	51

14.5 USART 同步模式

同步串行通信通常用在具有一个主控器件和一个或多个从动器件的系统中。主控器件包含产生波特率时钟所必需的电路，并为系统中的所有器件提供时钟。从动器件可以使用主控时钟，因此无需内部时钟发生电路。

在同步模式下，有 2 条信号线：双向数据线和时钟线。从动器件使用主控器件提供的外部时钟，将数据串行移入或移出相应的接收和发送移位寄存器。因为使用双向数据线，所以同步操作只能采用半双工方式。半双工是指：主控器件和从动器件都可以接收和发送数据，但是不能同时进行接收或发送。USART 既可以作为主控器件，也可以作为从动器件。

同步发送无需使用起始位和停止位。

14.5.1 同步主控模式

下列位用来将 USART 配置为同步主控操作：

- SYNC=1
- CSRC=1
- SREN=0（用于发送）；SREN=1（用于接收）
- CREN=0（用于发送）；CREN=1（用于接收）
- SPEN=1

将 TXSTA 寄存器的 SYNC 位置 1，可将 USART 配置用于同步操作。将 TXSTA 寄存器的 CSRC 位置 1，将器件配置为主控器件。将 RCSTA 寄存器的 SREN 和 CREN 位清零，以确保器件处于发送模式，否则器件配置为接收模式。将 RCSTA 寄存器的 SPEN 位置 1，使能 USART。

14.5.1.1 主控时钟

同步数据传输使用独立的时钟线同步传输数据。配置为主控器件的器件在 TX/CK 引脚发送时钟信号。当 USART 被配置为同步发送或接收操作时，TX/CK 输出驱动器自动使能。串行数据位在每个时钟的上升沿发生改变，以确保它们在下降沿有效。每个数据位的时间为一个时钟周期，有多少数据位就只能产生多少个时钟周期。

14.5.1.2 时钟极性

器件提供时钟极性选项以与 Microwire 兼容。由 TXSTA 寄存器的 SCKP 位选择时钟极性。将 SCKP 位置 1 将时钟空闲状态设置为高电平。当 SCKP 位置 1 时，数据在每个时钟的下降沿发生改变。清零 SCKP 位，将时钟空闲状态设置为低电平。当清零 SCKP 位时，数据在每个时钟的上升沿发生改变。

14.5.1.3 同步主控发送

由器件的 RX/DT 引脚输出数据。当 USART 配置为同步主控发送操作时，器件的 RX/DT 和 TX/CK 输出引脚自动使能。

向 TXREG 寄存器写入一个字符开始发送。如果 TSR 中仍保存全部或部分前一字符，新的字符数据保存在 TXREG 中，直到发送完前一字符的停止位为止。如果这是第一个字符，或者前一个字符已经完全从 TSR 中移出，则 TXREG 中的数据会被立即传输到 TSR 寄存器。当字符从 TXREG 传输到 TSR 后会立即开始发送数据。每个数据位在主控时钟的上升沿发生改变，并保持有效，直至下一个时钟的上升沿为止。

注：TSR 寄存器并未映射到数据存储器中，因此用户不能直接访问它。

14.5.1.4 同步主控发送设置

1. 初始化 SPBRG 寄存器，以获得所需的波特率。
   （请参见“USART 波特率发生器（BRG）”章节）
2. 将 SYNC、SPEN 和 CSRC 位置 1，使能同步主控串行端口。
3. 将 SREN 和 CREN 位清零，禁止接收模式。
4. 将 TXEN 位置 1 使能发送模式。
5. 如果需要发送 9 位字符，将 TX9EN 置 1。
6. 若需要中断，将 PIE1 寄存器中的 TXIE 位，以及 INTCON 寄存器中的 GIE 和 PEIE 位置 1。
7. 如果选择发送 9 位字符，应该将第 9 位数据装入 TX9D 位。
8. 通过将数据装入 TXREG 寄存器启动发送。

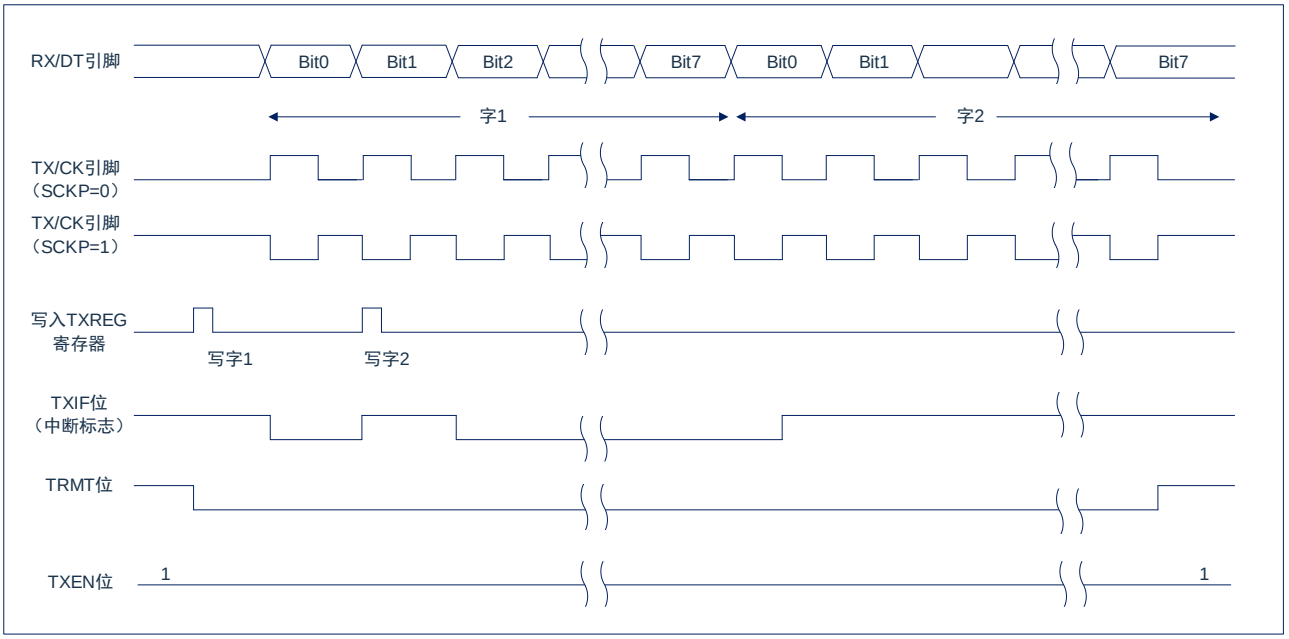


图 16-6：同步发送

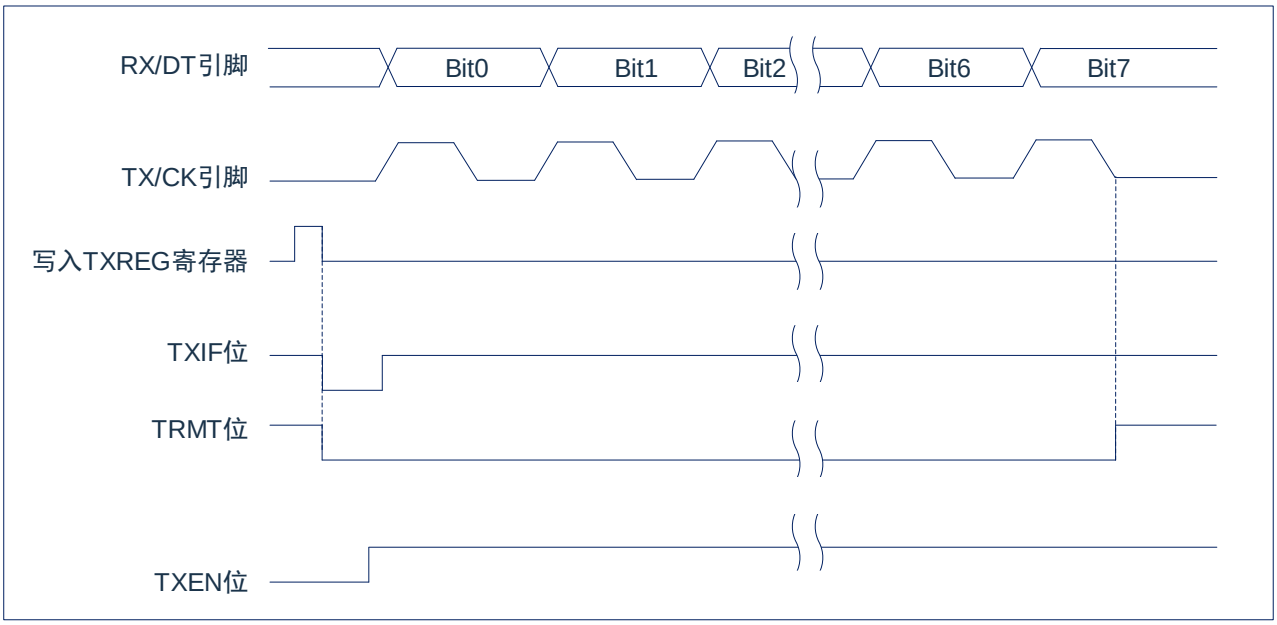


图 16-7：同步发送（通过 TXEN）

14.5.1.5 同步主控接收

在 RX/DT 引脚接收数据。当 USART 配置为同步主控接收时，自动禁止器件的 RX/DT 引脚的输出驱动器。

在同步模式下，将单字接收使能位（RCSTA 寄存器的 SREN 位）或连续接收使能位（RCSTA 寄存器的 CREN 位）置 1 使能接收。当将 SREN 置 1，CREN 位清零时，一个单字符中有多少数据位就只能产生多少时钟周期。一个字符传输结束后，自动清零 SREN 位。当 CREN 置 1 时，将产生连续时钟，直到清零 CREN 为止。如果 CREN 在一个字符的传输过程中清零，则 CK 时钟立即停止，并丢弃该不完整的字符。如果 SREN 和 CREN 都置 1，则当第一个字符传输完成时，SREN 位被清零，CREN 优先。

将 SREN 或 CREN 位置 1，启动接收。在 TX/CK 时钟引脚信号的下降沿采样 RX/DT 引脚上的数据，并将采样到的数据移入接收移位寄存器（RSR）。当 RSR 接收到一个完整字符时，将 RCIF 位置 1，字符自动移入 2 字节接收 FIFO。接收 FIFO 中最顶端字符的低 8 位可通过 RCREG 读取。只要接收 FIFO 中仍有未读字符，则 RCIF 位就保持置 1 状态。

14.5.1.6 从时钟

同步数据传输使用与数据线同步的独立时钟线。配置为从器件的器件接收 TX/CK 线上的时钟信号。当器件被配置为同步从发送或接收操作时，TX/CK 引脚的输出驱动器自动被禁止。串行数据位在时钟信号的前沿改变，以确保其在每个时钟的后沿有效。每个时钟周期只能传输一位数据，因此有多少数据位要传输就必须接收多少个时钟。

14.5.1.7 接收溢出错误

接收 FIFO 缓冲器可以保存 2 个字符。在读 RCREG 以访问 FIFO 之前，若完整地接收到第 3 个字符，则产生溢出错误。此时，RCSTA 寄存器的 OERR 位会置 1。FIFO 中先前的数据不会被改写。可以读取 FIFO 缓冲器内的 2 个字符，但是在错误被清除前，不能再接收其他字符。只能通过清除溢出条件，将 OERR 位清零。如果发生溢出时，SREN 位为置 1 状态，CREN 位为清零状态，则通过读 RCREG 寄存器清除错误。如果溢出时，CREN 为置 1 状态，则可以清零 RCSTA 寄存器的 CREN 位或清零 SPEN 位以复位 USART，从而清除错误。

14.5.1.8 接收 9 位字符

USART 支持接收 9 位字符。当 RCSTA 寄存器的 RX9EN 位置 1 时，USART 将接收到的每个字符的 9 位数据移入 RSR。当从接收 FIFO 缓冲器读取 9 位数据时，必须在读 RCREG 的 8 个低位之后，读取 RX9D 数据位。

14.5.1.9 同步主控接收设置

1. 初始化 SPBRG 寄存器，以获得所需的波特率。（注：必须满足 SPBRG>05H）
2. 将 SYNC、SPEN 和 CSRC 位置 1 使能同步主控串行端口。
3. 确保将 CREN 和 SREN 位清零。
4. 如果使用中断，将 INTCON 寄存器的 GIE 和 PEIE 位置 1，并将 PIE1 寄存器的 RCIE 位也置 1。
5. 如果需要接收 9 位字符，将 RX9EN 位置 1。
6. 将 SREN 位置 1，启动接收，或将 CREN 位置 1 使能连续接收。
7. 当字符接收完毕后，将 RCIF 中断标志位置 1。如果允许位 RCIE 置 1，还会产生一个中断。
8. 读 RCREG 寄存器获取接收到的 8 位数据。
9. 读 RCSTA 寄存器以获取第 9 个数据位（使能 9 位接收时），并判断接收过程中是否产生错误。
10. 如果产生溢出错误，清零 RCSTA 寄存器的 CREN 位或清零 SPEN 以复位 USART 来清除错误。

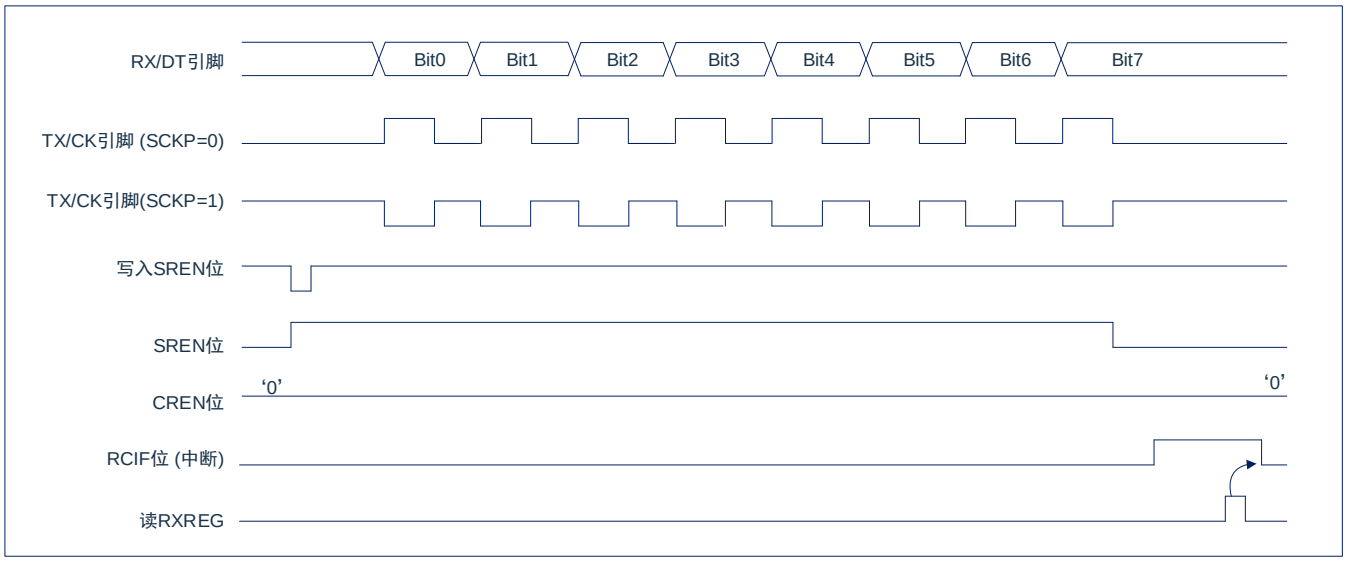


图 16-8：同步接收（主控模式，SREN）

注：时序图说明了 SREN=1 时的同步主控模式。

14.5.2 同步从动模式

下列位用来将 USART 配置为同步从动操作：

- SYNC=1
- CSRC=0
- SREN=0（用于发送）；SREN=1（用于接收）
- CREN=0（用于发送）；CREN=1（用于接收）
- SPEN=1

将 TXSTA 寄存器的 SYNC 位置 1，可将器件配置用于同步操作。将 TXSTA 寄存器的 CSRC 位置 1，将器件配置为从动器件。将 RCSTA 寄存器的 SREN 和 CREN 位清零，以确保器件处于发送模式，否则器件将被配置为接收模式。将 RCSTA 寄存器的 SPEN 位置 1，使能 USART。

14.5.2.1 USART 同步从动发送

同步主控和从动模式的工作原理是相同的（见章节“同步主控发送”）

14.5.2.2 同步从动发送设置

1. 将 SYNC 和 SPEN 位置 1 并将 CSRC 位清零。
2. 将 CREN 和 SREN 位清零。
3. 如果使用中断，将 INTCON 寄存器的 GIE 和 PEIE 位置 1，并将 PIE1 寄存器的 TXIE 位也置 1。
4. 如果需要发送 9 位数据，将 TX9EN 位置 1。
5. 将 TXEN 位置 1 使能发送。
6. 若选择发送 9 位数据，将最高位写入 TX9D 位。
7. 将低 8 位数据写入 TXREG 寄存器开始传输。

14.5.2.3 USART 同步从动接收

除了以下不同外，同步主控和从动模式的工作原理相同。

1. CREN 位总是置 1，因此接收器不能进入空闲状态。
2. SREN 位，在从动模式可为“任意值”。

14.5.2.4 同步从动接收设置

1. 将 SYNC 和 SPEN 位置 1 并将 CSRC 位清零。
2. 如果使用中断，将 INTCON 寄存器的 GIE 和 PEIE 位置 1，并将 PIE1 寄存器的 RCIE 位也置 1。
3. 如果需要接收 9 位字符，将 RX9EN 位置 1。
4. 将 CREN 位置 1，使能接收。
5. 当接收完成后，将 RCIF 位置 1。如果 RCIE 已置 1，还会产生一个中断。
6. 读 RCREG 寄存器，从接收 FIFO 缓冲器获取接收到的 8 个低数据位。
7. 如果使能 9 位模式，从 RCSTA 寄存器的 RX9D 位获取最高位。

如果产生溢出错误，清零 RCSTA 寄存器的 CREN 位或清零 SPEN 位以复位 USART 来清除错误。

15. 电气参数

15.1 极限参数

电源供应电压.....GND-0.3V~GND+6V
存储温度.....-50℃~125℃
工作温度.....-40℃~85℃
端口输入电压.....GND-0.3V~VDD+0.3V
所有端口最大灌电流.....200mA
所有端口最大拉电流.....-150mA

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

15.2 直流电气特性

符号	参数	测试条件		最小值	典型值	最大值	单位
		VDD	条件				
VDD	工作电压	-	F_{SYS}=16MHz	V_{LVR3}		5.5	V
		-	F_{SYS}=8MHz	V_{LVR1}		5.5	V
I_{DD}	工作电流	5V	F_{SYS}=16MHz		3		mA
		3V	F_{SYS}=16MHz		2		mA
		5V	F_{SYS}=8MHz		2		mA
		3V	F_{SYS}=8MHz		1		mA
		5V	烧写程序 EEPROM		6		mA
I_{STB}	静态电流	5V	LVR=DIS WDT=DIS	-	0.5	5	μA
		3V	LVR=DIS WDT=DIS	-	0.4	3	μA
		5V	LVR=DIS WDT=EN		4.5		μA
		3V	LVR=DIS WDT=EN		3.5		μA
		5V	LVR=EN WDT=DIS		9.5		μA
		3V	LVR=EN WDT=DIS		8.5		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_{EEPROM}	EEPROM 模块工作电压	-	----	2.1	-	5.5	V
R_{PH}	上拉电阻阻值	5V	V_O=0.5VDD	-	32	-	KΩ
		3V	V_O=0.5VDD	-	56	-	KΩ
R_{PD}	下拉电阻阻值	5V	V_O=0.5VDD	-	35	-	KΩ
		3V	V_O=0.5VDD	-	60	-	KΩ
I_{OL1}	输出口灌电流 普通 I/O 口	5V	V_{OL}=0.3VDD	-	47	-	mA
		3V	V_{OL}=0.3VDD	-	23	-	mA
I_{OL2}	输出口灌电流 大电流 PWM 口	5V	V_{OL}=0.3VDD	-	68	-	mA
		3V	V_{OL}=0.3VDD	-	29	-	mA
I_{OH1}	输出口拉电流 普通 I/O 口	5V	V_{OH}=0.7VDD	-	-20	-	mA
		3V	V_{OH}=0.7VDD	-	-8	-	mA
I_{OH2}	输出口拉电流 大电流 PWM 口	5V	V_{OH}=0.7VDD	-	-70	-	mA
		3V	V_{OH}=0.7VDD	-	-35	-	mA
V_{BG}	内部基准电压 0.6V	VDD=2.5~5.5V TA=25℃		-1.5%	0.6	1.5%	V
		VDD=2.5~5.5V TA=－40~85℃		-2.0%	0.6	2.0%	V

15.3 ADC 电气特性

（T_A= 25℃，除非另有说明）

符号	参数	测试条件	最小值	典型值	最大值	单位
V_{ADC}	ADC 工作电压	V_{ADREF}= VDD,F_{ADCCLK}=2MHz	3.0		5.5	V
		V_{ADREF}= VDD,F_{ADCCLK}=1MHz	2.5		5.5	V
		V_{ADREF}=1.2V,F_{ADCCLK}=62.5kHz	2.5		5.5	V
		V_{ADREF}=2.0V,F_{ADCCLK}=250kHz	2.5		5.5	V
		V_{ADREF}=2.4V,F_{ADCCLK}=250kHz	2.6		5.5	V
		V_{ADREF}=3.0V,F_{ADCCLK}=500kHz	3.3		5.5	V
I_{ADC}	ADC 转换电流	V_{ADC}=5V,V_{ADREF}= VDD, F_{ADCCLK}=500kHz			500	uA
		V_{ADC} =3V,V_{ADREF}= VDD, F_{ADCCLK}=500kHz			200	uA
V_{AIN}	ADC 输入电压	V_{ADC}=5V, V_{ADREF} = VDD, F_{ADCCLK}=500kHz	0		V_{ADC}	V
DNL1	微分非线性误差	V_{ADC}=5V, V_{ADREF} = VDD, F_{ADCCLK}=1MHz			±2	LSB
INL1	积分非线性误差	V_{ADC}=5V, V_{ADREF} = VDD, F_{ADCCLK}=1MHz			±2	LSB
DNL2	微分非线性误差	V_{ADC}=5V, V_{ADREF} = 3.0V, F_{ADCCLK}=500kHz,V_{AIN}<1V			±3	LSB
INL2	积分非线性误差	V_{ADC}=5V, V_{ADREF} = 3.0V, F_{ADCCLK}=500kHz,V_{AIN}<1V			±3	LSB
DNL3	微分非线性误差	V_{ADC}=5V, V_{ADREF} = 2.4V, F_{ADCCLK}=250kHz,V_{AIN}<0.8V			±3	LSB
INL3	积分非线性误差	V_{ADC}=5V, V_{ADREF} = 2.4V, F_{ADCCLK}=250kHz,V_{AIN}<0.8V			±3	LSB
DNL4	微分非线性误差	V_{ADC}=5V, V_{ADREF} = 2.0V, F_{ADCCLK}=250kHz,V_{AIN}<0.7V			±3	LSB
INL4	积分非线性误差	V_{ADC}=5V, V_{ADREF} = 2.0V, F_{ADCCLK}=250kHz,V_{AIN}<0.7V			±3	LSB
DNL5	微分非线性误差	V_{ADC}=5V, V_{ADREF} = 1.2V, F_{ADCCLK}=62.5kHz,V_{AIN}<0.4V			±3	LSB
INL5	积分非线性误差	V_{ADC}=5V, V_{ADREF} = 1.2V, F_{ADCCLK}=62.5kHz,V_{AIN}<0.4V			±3	LSB
T_{ADC}	ADC 转换时间			49		T_{ADCCLK}

15.4 ADC 内部 LDO 参考电压特性

（T_A= 25℃，除非另有说明）

符号	参数	测试条件	最小值	典型值	最大值	单位
V_{ADREF1}	LDO_OUT=1.2V	VDD=2.5~5.5V	-0.6%	1.2	+0.6%	V
V_{ADREF2}	LDO_OUT=2.0V	VDD=2.5~5.5V	-0.6%	2.0	+0.6%	V
V_{ADREF3}	LDO_OUT=2.4V	VDD=2.6~5.5V	-0.6%	2.4	+0.6%	V
V_{ADREF4}	LDO_OUT=3.0V	VDD=3.3~5.5V	-0.6%	3.0	+0.6%	V

15.5 LVR 电气特性

（T_A= 25℃，除非另有说明）

符号	参数	测试条件	最小值	典型值	最大值	单位
V_{LVR1}	LVR 设定电压=1.8V	VDD=1.5~5.5V		1.8		V
V_{LVR2}	LVR 设定电压=2.1V	VDD=1.8~5.5V		2.1		V
V_{LVR3}	LVR 设定电压=2.5V	VDD=2.2~5.5V		2.5		V
V_{LVR4}	LVR 设定电压=3.0V	VDD=2.8~5.5V		3.0		V

15.6 LVD 电气特性

（TA= 25℃，除非另有说明）

符号	参数	测试条件	最小值	典型值	最大值	单位
V_{LVD}	工作电压	-	1.8		5.5	V
	精度	VDD=1.8~5.5V, T_A=－40~85℃	-5%	V_{SET}	+5%	V

15.7 交流电气特性

（T_A= 25℃，除非另有说明）

符号	参数	测试条件		最小值	典型值	最大值	单位
		VDD	条件				
T_{WDT}	WDT 复位时间	5V	-		16		ms
		3V	-		16		ms
T_{EEPROM}	EEPROM 编程时间	5V	F_{OSC}=8MHz		4.6		ms
		3V	F_{OSC}=8MHz		4.6		ms
		5V	F_{OSC}=16MHz		4.6		ms
		3V	F_{OSC}=16MHz		4.6		ms
F_{RC}	内振频率稳定性	VDD=2.5~5.5V	T_A=25℃	-1.5%	8	+1.5%	MHz
		VDD=2.5~5.5V	T_A=-40~85℃	-2.5%	8	+2.5%	MHz
		VDD=2.5~5.5V	T_A=25℃	-1.5%	16	+1.5%	MHz
		VDD=2.5~5.5V	T_A=-40~85℃	-2.5%	16	+2.5%	MHz

16. 指令

16.1 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
DECR	[R]	递减数据存储器 R，结果放入 R	1	Z

助记符		操作	指令周期	标志
位操作-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
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

16.2 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
```

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

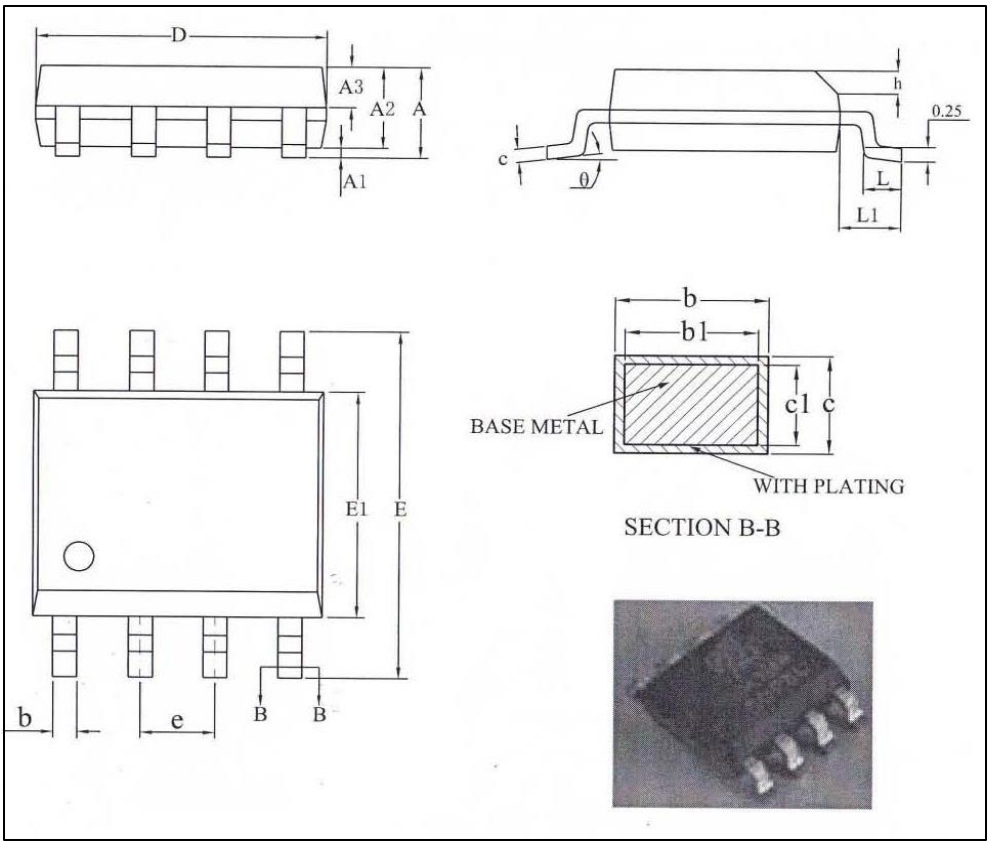
影响标志位: Z

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
```

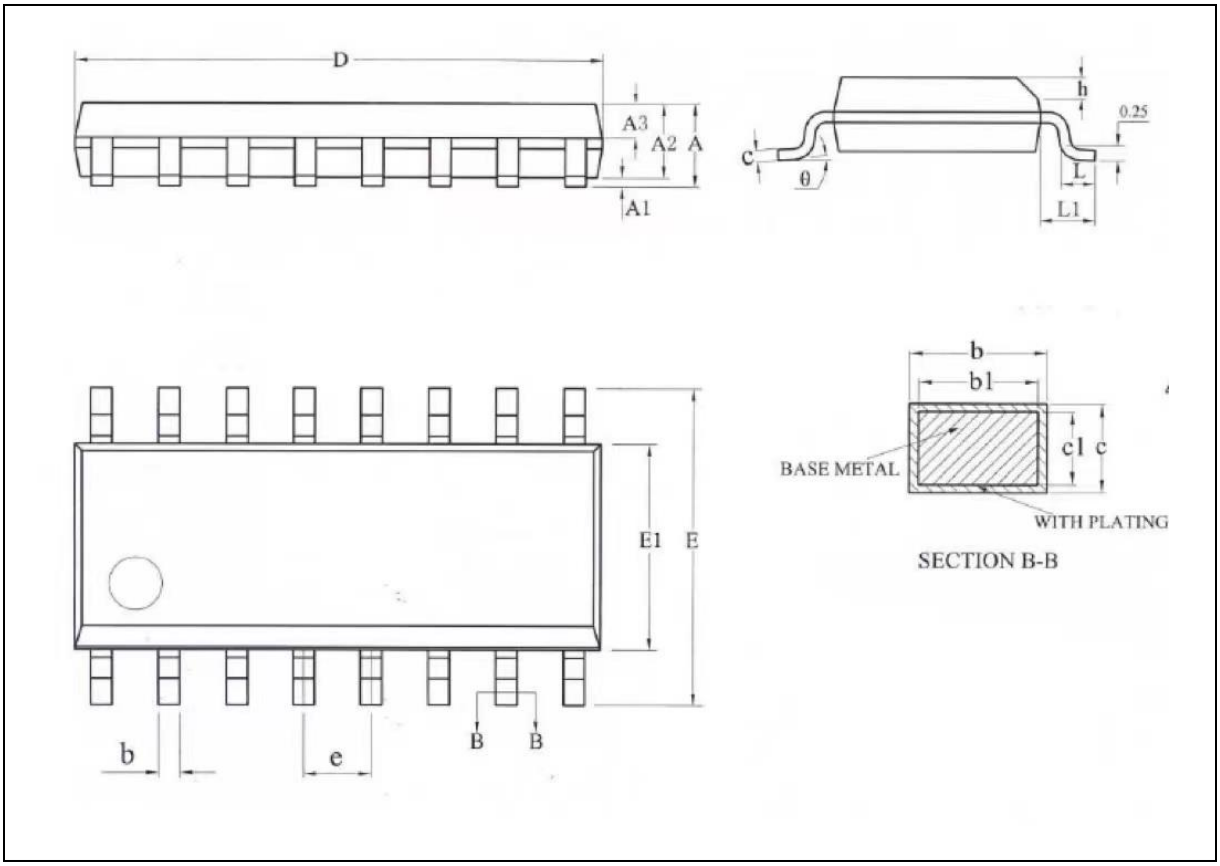
17. 封装

17.1 SOP8



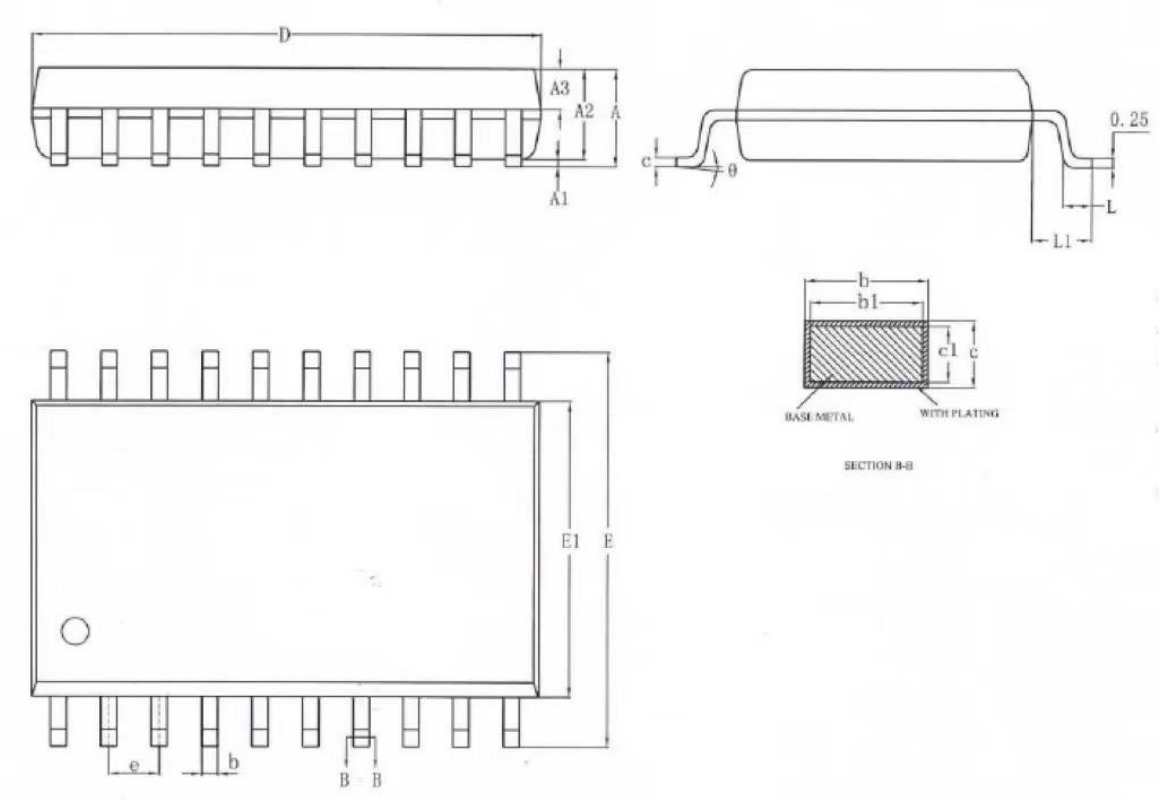
Symbol	Millimeter		
	Min	Nom	Max
A	-	-	1.75
A1	0.10	-	0.225
A2	1.30	1.40	1.50
A3	0.60	0.65	0.70
b	0.39	-	0.47
b1	0.38	0.41	0.44
c	0.20	-	0.24
c1	0.19	0.20	0.21
D	4.80	4.90	5.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
h	0.25	-	0.50
L	0.50	-	0.80
L1	1.05REF		
θ	0	-	8°

17.2 SOP16



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	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°

17.3 SOP20



Symbol	Millimeter		
	Min	Nom	Max
A	-	-	2.65
A1	0.10	-	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°

18. 版本修订说明

版本号	时间	修改内容
V1.0	2021 年 12 月	初始版本
V1.1	2022 年 9 月	增加 SOP8 封装产品