SC8F073 用户手册

增强型闪存 8 位 CMOS 单片机

Rev. 1.0.1

请注意以下有关CMS知识产权政策

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

＊中微半导体（深圳）股份有限公司的名称和标识都是本公司的注册商标。

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

目录

1. 产品概述 6
1.1 功能特性 6
1.2 产品型号一览表 7
1.3 系统结构框图 8
1.4 管脚分布 9
1.4.1 SC8F073AD716SP 管脚图 9
1.4.2 SC8F073AD716NPR 管脚图 10
1.4.3 SC8F073AD720NPR 管脚图 11
1.5 系统配置寄存器 13
1.6 在线串行编程 14
1.7 集成开发环境 15
2. 中央处理器（CPU） 16
2.1 内存 16
2.1.1 程序内存 16
2.1.2 数据存储器 19
2.2 寻址方式 23
2.2.1 直接寻址 23
2.2.2 立即寻址 23
2.2.3 间接寻址 23
2.3 堆栈 24
2.4 工作寄存器（ACC） 25
2.4.1 概述 25
2.4.2 ACC 应用 25
2.5 程序状态寄存器(STATUS) 26
2.6 预分频器(OPTION_REG) 28
2.7 程序计数器（PC） 29
2.8 看门狗计数器（WDT） 30
2.8.1 WDT 周期 30
2.8.2 与看门狗控制相关的寄存器 31
3. 系统时钟 32
3.1 概述 32
3.2 系统振荡器 34
3.2.1 内部 RC 振荡 34
3.3 起振时间 34
3.4 振荡器控制寄存器 34
3.5 时钟框图 35
4. 复位 36
4.1 上电复位 36
4.2 外部复位 36
4.3 掉电复位 37
4.3.1 掉电复位概述 37
4.3.2 掉电复位的改进办法 38
4.4 看门狗复位 39
5. 休眠模式 40
5.1 进入休眠模式 40
5.2 从休眠状态唤醒 40

5.3 使用中断唤醒 40
5.4 休眠模式应用举例 41
5.5 休眠模式唤醒时间 41
6. I/O 端口 42
6.1 I/O 口结构图 43
6.1.1 PORTA I/O 口结构图 43
6.1.2 PORTB I/O 口结构图 44
6.1.3 PORTC I/O 口结构图 45
6.2 PORTA 46
6.2.1 PORTA 数据及方向 46
6.2.2 PORTA 开漏输出控制 47
6.2.3 PORTA 模拟选择控制 47
6.2.4 PORTA 上拉电阻 47
6.2.5 PORTA 下拉电阻 48
6.2.6 PORTA 电平变化中断 48
6.3 PORTB 49
6.3.1 PORTB 数据及方向 49
6.3.2 PORTB 开漏输出控制 50
6.3.3 PORTB 模拟选择控制 50
6.3.4 PORTB 上拉电阻 50
6.3.5 PORTB 下拉电阻 51
6.3.6 PORTB 电平变化中断 51
6.4 PORTC 52
6.4.1 PORTC 数据及方向 52
6.4.2 PORTC 模拟选择控制 53
6.4.3 PORTC 上拉电阻 53
6.5 I/O 使用 54
6.5.1 写 I/O 口 54
6.5.2 读 I/O 口 54
6.6 I/O 口使用注意事项 55
7. 中断 56
7.1 中断概述 56
7.2 中断控制寄存器 57
7.2.1 中断控制寄存器 57
7.2.2 外设中断允许寄存器 58
7.2.3 外设中断请求寄存器 59
7.3 中断现场的保护方法 60
7.4 中断的优先级，及多中断嵌套 60
8. 定时计数器 TIMER0 61
8.1 定时计数器 TIMER0 概述 61
8.2 TIMER0 的工作原理 62
8.2.1 8 位定时器模式 62
8.2.2 8 位计数器模式 62
8.2.3 软件可编程预分频器 62
8.2.4 在 TIMER0 和 WDT 模块间切换预分频器 62
8.2.5 TIMER0 中断 62
8.3 TIMER0 相关寄存器 63
9. 定时计数器 TIMER1 64

9.1 TIMER1 概述 64
9.2 TIMER1 的工作原理 65
9.3 时钟源选择 65
9.3.1 内部时钟源 65
9.3.2 外部时钟源 65
9.4 TIMER1 预分频器 66
9.5 TIMER1 振荡器 66
9.6 在异步计数器模式下的 TIMER1 工作原理 66
9.6.1 异步计数器模式下对 TIMER1 的读写操作 66
9.7 TIMER1 门控 66
9.8 TIMER1 中断 67
9.9 休眠期间的 TIMER1 工作原理 67
9.10 TIMER1 相关寄存器 68

10. 定时计数器 TIMER2 69

10.1 TIMER2 概述 69
10.2 TIMER2 的工作原理 70
10.3 TIMER2 相关的寄存器 71

11. 10 位 PWM 模块 72

11.1 引脚配置 72
11.2 相关寄存器说明 72
11.3 10 位 PWM 寄存器写操作顺序 77
11.4 10 位 PWM 周期 77
11.5 10 位 PWM 占空比 77
11.6 系统时钟频率的改变 77
11.7 可编程的死区延时模式 78
11.8 10 位 PWM 设置 78

12. 通用同步/异步收发器(USART) 79

12.1 USART 异步模式 81
12.1.1 USART 异步发送器 81
12.1.2 USART 异步接收器 84
12.2 异步操作时的时钟准确度 87
12.3 USART 相关寄存器 87
12.4 USART 波特率发生器（BRG） 90
12.5 USART 同步模式 91
12.5.1 同步主控模式 91
12.5.2 同步从动模式 95

13. 比较器（CMP1 和 CMP2） 96

13.1 CMPx 的功能框图 96
13.2 CMPx 功能特性 97
13.3 CMPx 相关功能 97
13.3.1 CMPx 功能描述 97
13.3.2 CMPx 内部电阻分压输出 97
13.3.3 CMPx 监测电源电压 98
13.3.4 CMPx 的中断使用 98
13.3.5 CMPx 中断休眠唤醒 99
13.3.6 CMPx 结果输出引脚配置 99
13.4 CMPx 相关寄存器 100

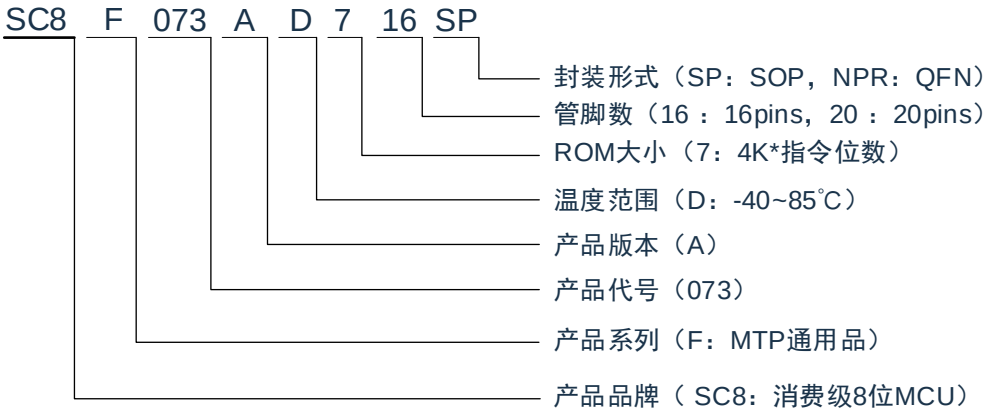
14. 模数转换（ADC） 101
14.1 ADC 概述 101
14.2 ADC 配置 102
14.2.1 端口配置 102
14.2.2 通道选择 102
14.2.3 ADC 内部基准电压 102
14.2.4 ADC 参考电压 102
14.2.5 转换时钟 103
14.2.6 ADC 中断 103
14.2.7 结果格式化 103
14.3 ADC 工作原理 104
14.3.1 启动转换 104
14.3.2 完成转换 104
14.3.3 终止转换 104
14.3.4 ADC 在休眠模式下的工作原理 104
14.3.5 A/D 转换步骤 105
14.4 ADC 相关寄存器 106
15. 程序 EEPROM 和程序存储器控制 109
15.1 概述 109
15.2 相关寄存器 110
15.2.1 EEADR 和 EEADRH 寄存器 110
15.2.2 EECON1 和 EECON2 寄存器 110
15.3 读程序 EEPROM 112
15.4 写程序 EEPROM 113
15.5 读程序存储器 115
15.6 写程序存储器 115
15.7 程序 EEPROM 操作注意事项 116
15.7.1 关于程序 EEPROM 的烧写时间 116
15.7.2 写校验 116
15.7.3 避免误写的保护 116
16. 电气参数 117
16.1 极限参数 117
16.2 直流电气特性 118
16.3 比较器特性 119
16.4 ADC 电气特性 119
16.5 上电复位特性 120
16.6 交流电气特性 120
16.7 LSE 特性 121
17. 指令 122
17.1 指令一览表 122
17.2 指令说明 124
18. 封装 140
18.1 SOP16 140
18.2 QFN16(3*3*0.75-0.50) 141
18.3 QFN20 142
19. 版本修订说明 143

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
  - 3 个定时中断
  - RA、RB 口电平变化中断
  - 其它外设中断
- 内置 128 字程序 EEPROM
  - 可重复擦写 1 万次
- 定时器
  - 8 位定时器 TIMER0，TIMER2
  - 16 位定时器 TIMER1
  - TIMER0、TIMER1、TIMER2 可选择外部 32.768Khz 振荡时钟源
- 内置 2 路比较器模块
  - COMP1 正端可选:RB1/电阻分压输出
  - COMP1 负端可选：RB1/RB2/RB4/RB5/BG/电阻分压输出
  - COMP2 正端可选:RA1/电阻分压出
  - COMP2 负端可选：RA1/RA2/RB0/RB1/BG/电阻分压输出
- 工作电压范围：3.0V~5.5V@16MHz/2T
  1.8V~5.5V@16MHz/4T
  工作温度范围：-40℃~85℃
- 内部 RC 振荡：设计频率 16MHz
- 指令周期（单指令或双指令）
- 内置 PWM 模块
  - 5 路 PWM，可设置 2 路互补输出，极性可选
  - 4 路 PWM 共用周期，独立占空比
  - 1 路 PWM 独立周期，独立占空比
  - 10 位 PWM 精度
- LVR 可选 1.8V/2V/2.5V/3V
- 高精度 12 位 ADC
  - 内建高精度 1.2V 基准电压
  - 可选择内部参考源：2.0V/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
  - 可将通信口配置在 RA3/RA4、RB0/RB1

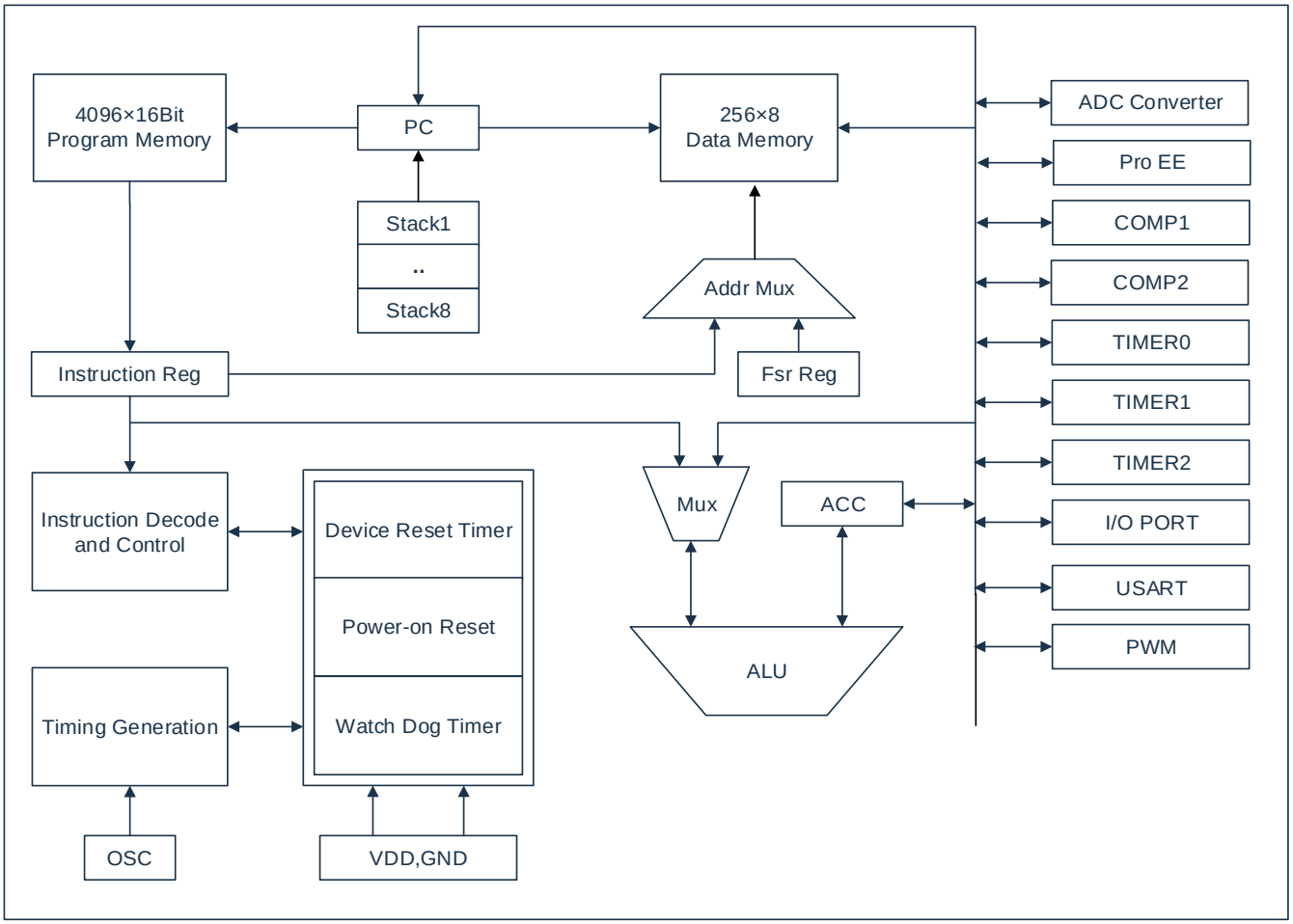
1.2 产品型号一览表



型号说明

PRODUCT	ROM	RAM	Pro EE	PWM	ACOMP	I/O	ADC	TIMER	USART	PACKAGE
SC8F073AD716SP	4K×16	256×8	128×16	5	2	14	12Bit×14	3	1	SOP16
SC8F073AD716NPR	4K×16	256×8	128×16	5	2	14	12Bit×14	3	1	QFN16
SC8F073AD720NPR	4K×16	256×8	128×16	5	2	18	12Bit×18	3	1	QFN20

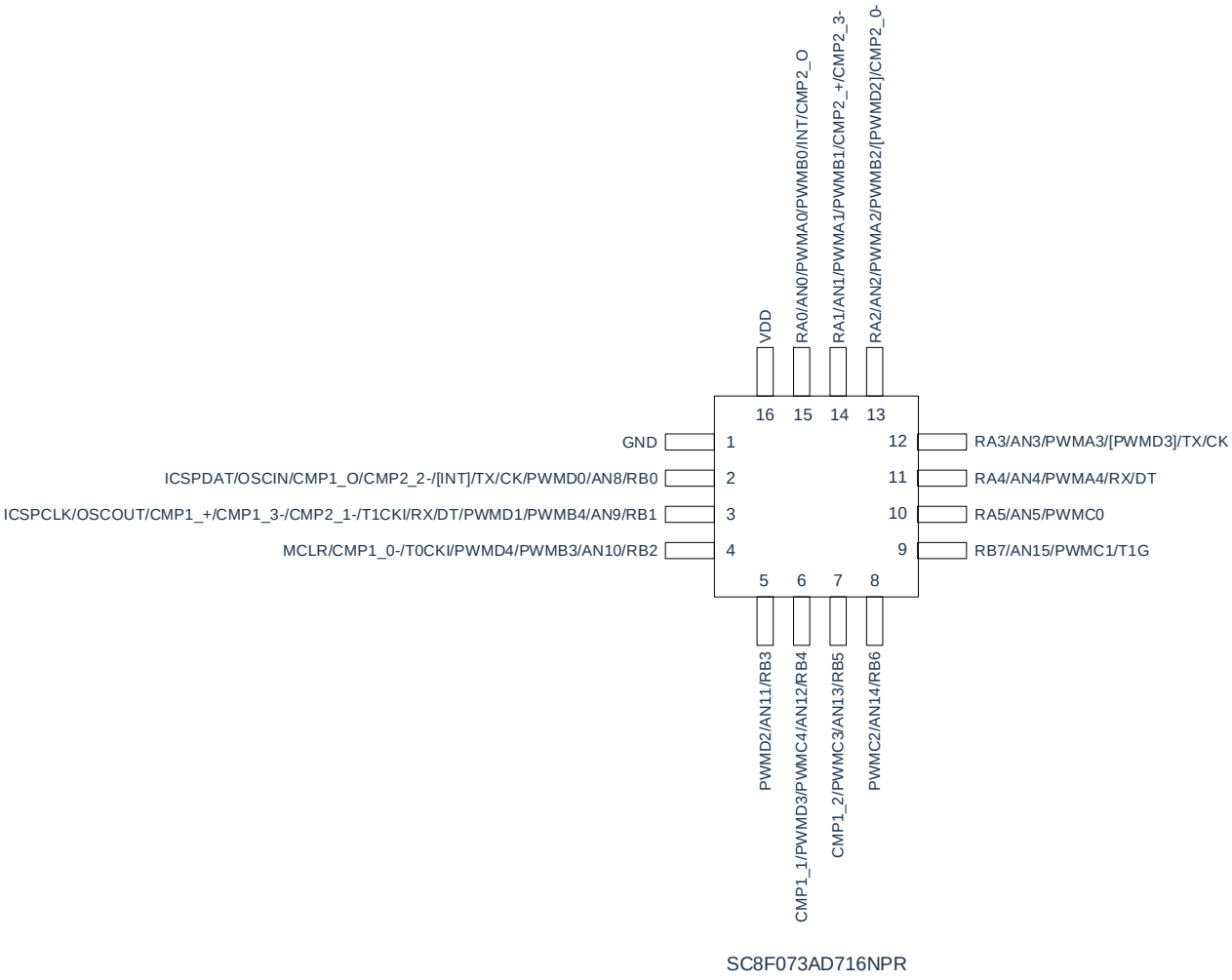
1.3 系统结构框图


4096×16Bit
Program Memory
PC
256×8
Data Memory
ADC Converter
Pro EE
COMP1
COMP2
TIMER0
TIMER1
TIMER2
I/O PORT
USART
PWM
Stack1
..
Stack8
Addr Mux
Fsr Reg
Instruction Reg
Instruction Decode
and Control
Device Reset Timer
Power-on Reset
Watch Dog Timer
Timing Generation
OSC
VDD,GND
Mux
ACC
ALU

1.4 管脚分布

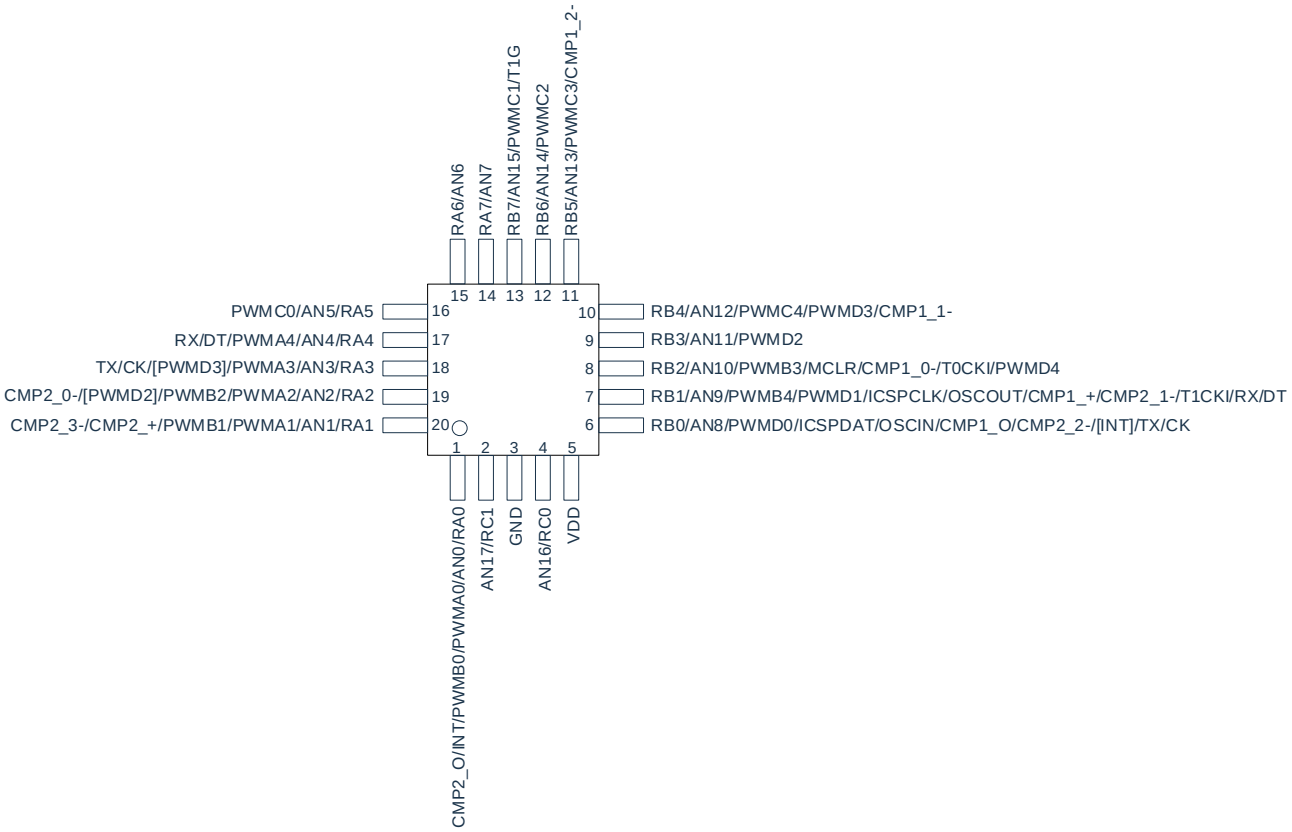
1.4.1 SC8F073AD716SP 管脚图

1.4.2 SC8F073AD716NPR 管脚图



SC8F073AD716NPR

1.4.3 SC8F073AD720NPR 管脚图



SC8F073AD720NPR

注：

1) RA3/RA4 与 RB0/RB1 的串口功能由 CONFIG 设置；
2) RA0 与 RB0 的 INT 功能由 CONFIG 设置；
3) RA2/RA3 与 RB3/RB4 的 PWMD 组功能由 CONFIG 配置。

SC8F073 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
RA0-RA7	I/O	可编程为输入脚，推挽或开漏输出脚，带上拉、下拉电阻功能、电平变化中断功能
RB0-RB7	I/O	可编程为输入脚，推挽或开漏输出脚，带上拉、下拉电阻功能、电平变化中断功能
RC0-RC1	I/O	可编程为输入脚，推挽输出脚，带上拉电阻功能
ICSPCLK/ICSPDAT	I/O	编程时钟/数据脚
PWMA0-PWMA4	O	PWMA 组输出脚
PWMB0-PWMB4	O	PWMB 组输出脚
PWMC0-PWMC4	O	PWMC 组输出脚
PWMD0-PWMD4	O	PWMD 组输出脚
AN0-AN17	I	12 位 ADC 输入脚
INT	I	外部中断输入脚
CMP1_+	I	比较器 1 正端输入脚
CMP2_+	I	比较器 2 正端输入脚
CMP1_0-， CMP1_1-，CMP1_2-，CMP1_3-	I	比较器 1 负端输入脚
CMP2_0-， CMP2_1-，CMP2_2-，CMP2_3-	I	比较器 2 负端输入脚
CMP1_O	O	比较器 1 结果输出脚
CMP2_O	O	比较器 2 结果输出脚
T0CKI	I	TIMER0 外部时钟输入脚
T1CKI	I	TIMER1 外部时钟输入脚
T1G	I	TIMER1 门控输入脚
TX/CK	I/O	USART 异步发送输出/同步时钟输入/输出脚
RX/DT	I/O	USART 异步接收输入脚/同步数据输入/输出脚
OSCIN/OSCOUT	I/O	32.768K 晶振输入/输出脚
MCLR	I	外部复位输入脚

1.5 系统配置寄存器

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

1. WDT（看门狗选择）
   - ENABLE　　打开看门狗定时器
   - DISABLE　　关闭看门狗定时器
2. PROTECT（加密）
   - DISABLE　　ROM 代码不加密
   - ENABLE　　ROM 代码加密，加密后烧写仿真器读出来的值将不确定
3. LVR_SEL（低压侦测选择）
   - 1.8V
   - 2.0V
   - 2.5V
   - 3.0V
4. F_{CPU}_DIV（指令时钟分频）
   - 4T　　4 分频，$F_{CPU}=F_{SYS}/4$
   - 2T　　2 分频，$F_{CPU}=F_{SYS}/2$
5. WDT_DIV（WDT 预分频系数控制）
   - DISABLE　　通过 OPTION_REG 寄存器可选择 WDT 预分频从 1:128
   - ENABLE　　通过 OPTION_REG 寄存器可选择 WDT 预分频从 3:384
6. ICSP（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
7. USART_PORT_SEL（USART 端口选择）
   - RA3/RA4　　选择 RA3 为 TX 口，RA4 为 RX 口
   - RB0/RB1　　选择 RB0 为 TX 口，RB1 为 RX 口
8. INT_PORT_SEL（INT 端口选择）
   - RA0　　选择 RA0 为 INT 口
   - RB0　　选择 RB0 为 INT 口
9. PWMD_PORT_SEL（PWMD 组端口选择）
   - PWMD0~PWMD4 分配在 RB0/RB1/RB3/RB4/RB2
   - PWMD0~PWMD4 分配在 RB0/RB1/RA2/RA3/RB2
10. EXT_RESET（外部复位口功能选择）
    - DISABLE　　禁止外部复位功能，RB2 为普通 IO 口
    - ENABLE　　使能外部复位功能，RB2 为外部复位口

1.6 在线串行编程

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

- 电源线
- 接地线
- 数据线
- 时钟线

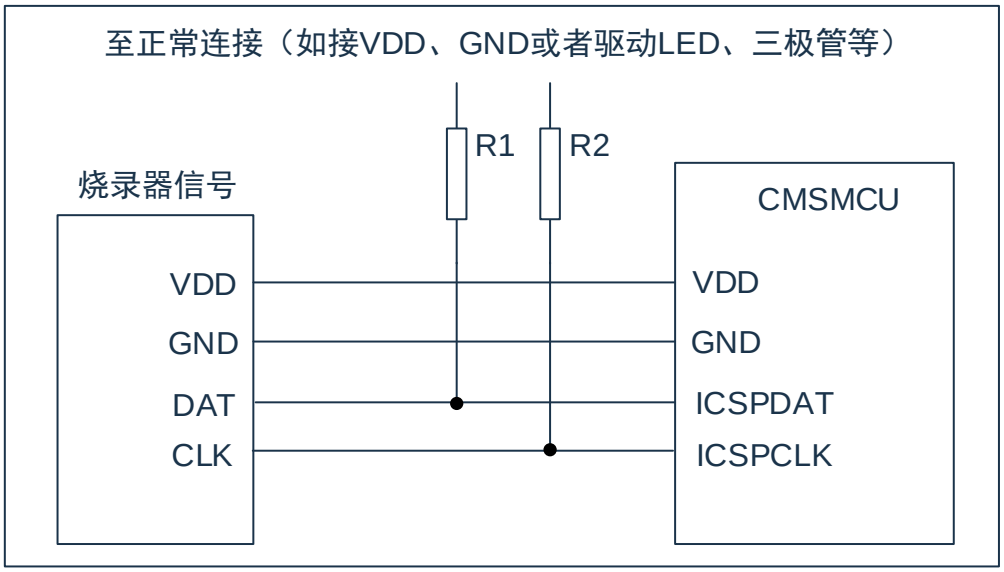


图 1-1：典型的在线串行编程连接方式

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

1.7 集成开发环境

- 片上调试（OCD），ISP
- 4 个硬件断点
- 软件复位，暂停，单步，运行等

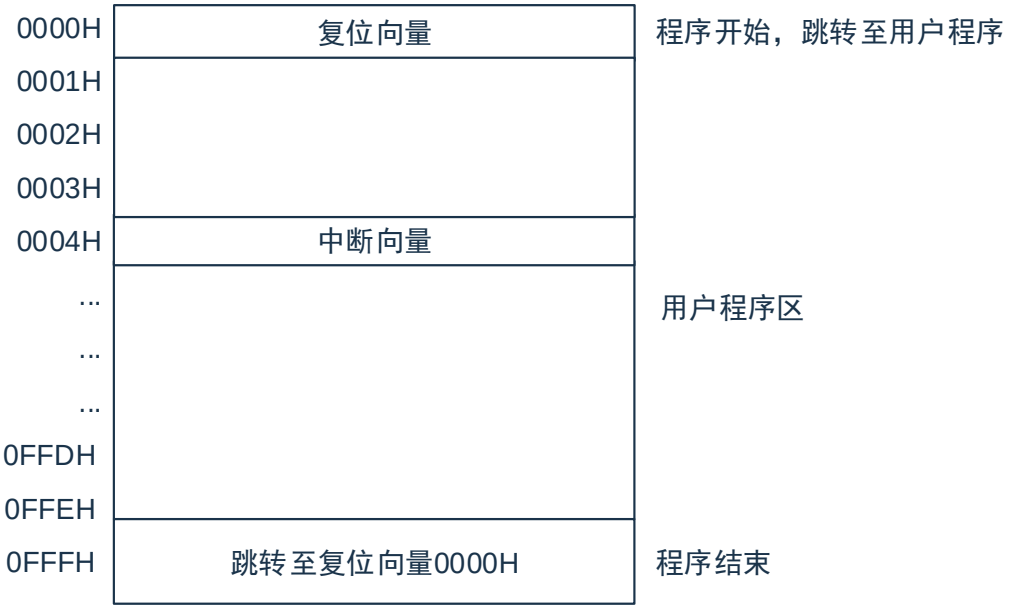
2. 中央处理器（CPU）

2.1 内存

2.1.1 程序内存

SC8F073 程序存储器空间

MTP：4K



2.1.1.1复位向量(0000H)

单片机具有一个字长的系统复位向量（0000H）。具有以下三种复位方式：

- 上电复位
- 看门狗复位
- 低压复位（LVR）

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

```
MTP 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

```
MTP 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 MTP 空间的分页处。

2.1.2 数据存储器

SC8F073 数据存储器列表

	地址		地址		地址		地址
INDF	00H	INDF	80H	INDF	100H	INDF	180H
OPTION_REG	01H	TMR0	81H	OPTION_REG	101H	TMR0	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
TRISB	05H	TRISA	85H	TRISC	105H	-----	185H
PORTB	06H	PORTA	86H	PORTC	106H	-----	186H
WPDB	07H	WPDA	87H	-----	107H	-----	187H
WPUB	08H	WPUA	88H	WPUC	108H	-----	188H
IOCB	09H	IOCA	89H	ANSEL2	109H	-----	189H
PCLATH	0AH	PCLATH	8AH	PCLATH	10AH	PCLATH	18AH
INTCON	0BH	INTCON	8BH	INTCON	10BH	INTCON	18BH
ODCONB	0CH	ODCONA	8CH	TMR1L	10CH	-----	18CH
PIR1	0DH	EECON1	8DH	TMR1H	10DH	-----	18DH
PIE1	0EH	EECON2	8EH	T1CON	10EH	-----	18EH
CMP1CON0	0FH	EEDAT	8FH	PIR2	10FH	-----	18FH
CMP1CON1	10H	EEDATH	90H	PIE2	110H	-----	190H
PR2	11H	EEADR	91H	-----	111H	-----	191H
TMR2	12H	EEADRH	92H	-----	112H	-----	192H
T2CON	13H	ANSEL0	93H	-----	113H	-----	193H
OSCCON	14H	ANSEL1	94H	-----	114H	-----	194H
PWMCON0	15H	ADCON0	95H	-----	115H	-----	195H
PWMCON1	16H	ADCON1	96H	-----	116H	-----	196H
PWMTL	17H	-----	97H	-----	117H	-----	197H
PWMTH	18H	ADRESL	98H	TXSTA	118H		198H
PWMD0L	19H	ADRESH	99H	RCSTA	119H		199H
PWMD1L	1AH	CMP2CON0	9AH	SPBRG	11AH		19AH
PWMD4L	1BH	PWMD2L	9BH	TXREG	11BH	-----	19BH
PWMT4L	1CH	PWMD3L	9CH	RCREG	11CH		19CH
PWMCON2	1DH	PWM23DT	9DH	-----	11DH		19DH
PWMD01H	1EH	PWMD23H	9EH	-----	11EH		19EH
PWM01DT	1FH	CMP2CON1	9FH	-----	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
			EFH		16FH		1EFH
		快速存储区 70H-7FH	F0H -- FFH	快速存储区 70H-7FH	170H -- 17FH	快速存储区 70H-7FH	1F0H -- 1FFH
	7FH						
BANK0		BANK1		BANK2		BANK3	

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

SC8F073 特殊功能寄存器汇总 Bank0

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	复位值
00H	INDF	寻址该单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
01H	OPTION_REG	T0LSE_EN	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	01111011
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	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	11111111
06H	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxxxxxx
07H	WPDB	WPDB7	WPDB6	WPDB5	WPDB4	WPDB3	WPDB2	WPDB1	WPDB0	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3位的写缓冲器			-----000
0BH	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	00000000
0CH	ODCONB	ODCONB7	ODCONB6	ODCONB5	ODCONB4	ODCONB3	ODCONB2	ODCONB1	ODCONB0	00000000
0DH	PIR1	RCIF	TXIF	CMP1IF	PWMIF	RAIF	TMR1IF	TMR2IF	ADIF	--000-00
0EH	PIE1	RCIE	TXIE	CMP1IE	PWMIE	RAIE	TMR1IE	TMR2IE	ADIE	--000-00
0FH	CMP1CON0	CMP1EN	CMP1PS	CMP1NS2	CMP1NS1	CMP1NS0	CMP1NV	CMP1OUT	CMP1OEN	00000000
10H	CMP1CON1	CMP1IM	AN1_EN	RBIAS1_H	RBIAS1_L	LVDS1<3:0>				00000000
11H	PR2	TIMER2周期寄存器								11111111
12H	TMR2	TIMER2模块寄存器								00000000
13H	T2CON	CLK_SEL	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	00000000
14H	OSCCON	----	IRCF2	IRCF1	IRCF0	----	----	SWDTEN	-----	-101--1-
15H	PWMCON0	CLKDIV<2:0>			PWM4EN	PWM3EN	PWM2EN	PWM1EN	PWM0EN	00000000
16H	PWMCON1	PWMIO_SEL<1:0>		PWM2DTEN	PWM0DTEN	----	----	DT_DIV<1:0>		0000--00
17H	PWMTL	PWM0~PWM1周期低8位寄存器								00000000
18H	PWMTH	----	----	PWM4D<9:8>		PWM4T<9:8>		PWMT<9:8>		--000000
19H	PWMD0L	PWM0占空比低8位								00000000
1AH	PWMD1L	PWM1占空比低8位								00000000
1BH	PWMD4L	PWM4占空比低8位								00000000
1CH	PWMT4L	PWM4周期低8位寄存器								00000000
1DH	PWMCON2	----	----	----	PWM4DIR	PWM3DIR	PWM2DIR	PWM1DIR	PWM0DIR	---00000
1EH	PWMD01H	----	----	PWMD1<9:8>		----	----	PWMD0<9:8>		--00--00
1FH	PWM01DT	----	----	PWM01DT<5:0>						--000000

SC8F073 特殊功能寄存器汇总 Bank1

地址	名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	复位值
80H	INDF	寻址该单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
81H	TMR0	TIMER0数据寄存器								xxxxxxxx
82H	PCL	程序计数器低字节								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	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxxxxxx
87H	WPDA	WPDA7	WPDA6	WPDA5	WPDA4	WPDA3	WPDA2	WPDA1	WPDA0	00000000
88H	WPUA	WPUA7	WPUA6	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0	00000000
89H	IOCA	IOCA7	IOCA6	IOCA5	IOCA4	IOCA3	IOCA2	IOCA1	IOCA0	00000000
8AH	PCLATH	----	----	----	----	程序计数器高4位的写缓冲器				----0000
8BH	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	00000000
8CH	ODCONA	ODCONA7	ODCONA6	ODCONA5	ODCONA4	ODCONA3	ODCONA2	ODCONA1	ODCONA0	00000000
8DH	EECON1	EEPGD	----	----	----	WRERR	WREN	WR	RD	0---0000
8EH	EECON2	EEPROM控制寄存器2（不是物理寄存器）								--------
8FH	EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0	xxxxxxxx
90H	EEDATH	EEDATH7	EEDATH6	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0	xxxxxxxx
91H	EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0	00000000
92H	EEADRH	----	----	----	----	EEADRH3	EEADRH2	EEADRH1	EEADRH0	----0000
93H	ANSEL0	ANS7	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0	00000000
94H	ANSEL1	ANS15	ANS14	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	00000000
95H	ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/ $\overline{\text{DONE}}$	ADON	00000000
96H	ADCON1	ADFM	CHS4	----	----	----	LDO_EN	LDO_SEL1	LDO_SEL0	00---000
98H	ADRESL	A/D结果寄存器的低字节								xxxxxxxx
99H	ADRESH	A/D结果寄存器的高字节								xxxxxxxx
9AH	CMP2CON0	CMP2EN	CMP2PS	CMP2NS2	CMP2NS1	CMP2NS0	CMP2NV	CMP2OUT	CMP2OEN	00000000
9BH	PWMD2L	PWM2占空比低8位								00000000
9CH	PWMD3L	PWM3占空比低8位								00000000
9DH	PWM23DT	----	----	PWM23死区延时时间						--000000
9EH	PWMD23H	----	----	PWMD3<9:8>		----	----	PWMD2<9:8>		--00--00
9FH	CMP2CON1	CMP2IM	AN2_EN	RBIAS2_H	RBIAS2_L	LVDS2<3:0>				00000000

SC8F073 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	寻址该单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
101H	OPTION_REG	T0LSE_EN	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	01111011
102H	PCL	程序计数器低字节								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
105H	TRISC	----	----	----	----	----	----	TRISC1	TRISC0	------11
106H	PORTC	----	----	----	----	----	----	RC1	RC0	------xx
108H	WPUC	----	----	----	----	----	----	WPUC1	WPUC0	------00
109H	ANSEL2	----	----	----	----	----	----	ANS17	ANS16	------00
10AH	PCLATH	----	----	----	----	程序计数器高4位的写缓冲器				----0000
10BH	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	00000000
10CH	TMR1L	16位TIMER1寄存器低字节的数据寄存器								xxxxxxxx
10DH	TMR1H	16位TIMER1寄存器高字节的数据寄存器								xxxxxxxx
10EH	T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	00000000
10FH	PIR2	----	----	CMP2IF	----	----	----	----	----	--0-----
110H	PIE2	----	----	CMP2IE	----	----	----	----	----	--0-----
118H	TXSTA	CSRC	TX9EN	TXEN	SYNC	SCKP	----	TRMT	TX9D	00000-10
119H	RCSTA	SPEN	RX9EN	SREN	CREN	RCIDL	FERR	OERR	RX9D	00001000
11AH	SPBRG	USART波特率数据寄存器								00000000
11BH	TXREG	USART发送数据寄存器								00000000
11CH	RCREG	USART接收数据寄存器								00000000

SC8F073 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	寻址该单元会使用FSR的内容寻址数据存储器（不是物理寄存器）								xxxxxxxx
181H	TMR0	TIMER0数据寄存器								xxxxxxxx
182H	PCL	程序计数器低字节								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
18AH	PCLATH	----	----	----	----	程序计数器高4位的写缓冲器				----0000
18BH	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	00000000

2.2 寻址方式

2.2.1 直接寻址

通过工作寄存器（ACC）来对 RAM 进行操作。

例：ACC 的值送给 30H 寄存器

```
LD      30H,A
```

例：30H 寄存器的值送给 ACC

```
LD      A,30H
```

2.2.2 立即寻址

把立即数传给工作寄存器（ACC）

例：立即数 12H 送给 ACC

```
LDIA    12H
```

2.2.3 间接寻址

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

例：FSR 及 INDF 的应用

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

例：间接寻址清 RAM(20H-7FH)举例：

```
        LDIA    1FH
        LD      FSR,A       ;间接寻址指针指向 1FH
LOOP:
        INCR    FSR         ;地址加 1，初始地址为 20H
        CLR     INDF        ;清零 FSR 所指向的地址
        LDIA    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。

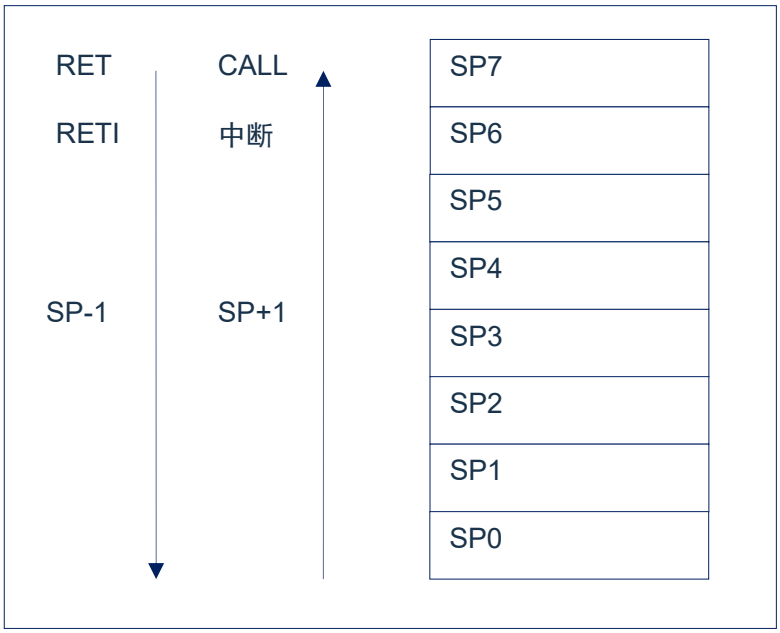


图 2-2：堆栈缓存器工作原理

堆栈缓存器的使用将遵循一个原则“先进后出”。

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

2.4.2 ACC 应用

例：用 ACC 做数据传送

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

例：用 ACC 做立即寻址目标操作数

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

例：用 ACC 做双操作数指令的第一操作数

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

例：用 ACC 做双操作数指令的第二操作数

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

2.5 程序状态寄存器(STATUS)

STATUS 寄存器如下表所示，包含：

- ALU 的算术状态。
- 复位状态。

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

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

程序状态寄存器 STATUS（03H）

03H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C
R/W	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(用与间接寻址)
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Bank0
01= 选择Bank1
10= 选择Bank2
11= 选择Bank3
Bit4 TO: 超时位
1= 上电或执行了CLRWDT指令或STOP指令
0= 发生了WDT超时
Bit3 PD: 掉电位
1= 上电或执行了CLRWDT指令
0= 执行了STOP指令
Bit2 Z: 结果为零位
1= 算术或逻辑运算的结果为零
0= 算术或逻辑运算的结果不为零
Bit1 DC: 半进位/借位位
1= 发生了结果的低4位向高位进位或低4位没有发生借位
0= 结果的低4位没有向高位进位或低4位向高位借位
Bit0 C: 进位/借位位
1= 结果的最高位发生了进位或没有发生借位
0= 结果的最高位没有发生进位或发生了借位

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

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

影响 PD、TO 的事件表

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

复位后 TO/PD 的状态

2.6 预分频器(OPTION_REG)

OPTION_REG 寄存器是可读写的寄存器，包括各种控制位用于配置：

- WDT 预分频器。
- 外部中断触发边沿

预分频器控制寄存器 OPTION_REG（01H）

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

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

PS2	PS1	PS0	TMR0 分频比	WDT 分频比 (WDT_DIV=DISABLE)	WDT 分频比 (WDT_DIV=ENABLE)
0	0	0	1:2	1:1	1:3
0	0	1	1:4	1:2	1:6
0	1	0	1:8	1:4	1:12
0	1	1	1:16	1:8	1:24
1	0	0	1:32	1:16	1:48
1	0	1	1:64	1:32	1:96
1	1	0	1:128	1:64	1:192
1	1	1	1:256	1:128	1:384

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。

2.7 程序计数器（PC）

程序计数器（PC）控制程序内存 MTP 中的指令执行顺序，它可以寻址整个 MTP 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 4Kx16 位程序地址。对 PCL 赋值将会产生一个短跳转动作，跳转范围为当前页的 256 个地址。

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

下面给出几种特殊情况的 PC 值

复位时	PC=0000；
中断时	PC=0004（原来的 PC+1 会被自动压入堆栈）；
CALL 时	PC=程序指定地址（原来的 PC+1 会被自动压入堆栈）；
RET、RETI、RETI 时	PC=堆栈出来的值；
操作 PCL 时	PC[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（WatchDogTimer）是一个片内自振式的 RC 振荡定时器，无需任何外围组件，即使芯片的主时钟停止工作，WDT 也能保持计时。WDT 计时溢出将产生复位。

2.8.1 WDT 周期

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

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

注：

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

2.8.2 与看门狗控制相关的寄存器

振荡控制寄存器 OSCCON (14H)

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

Bit7 未用。

Bit6~Bit4 IRCF<2:0>: 内部振荡器频率选择位

111= $F_{SYS} = F_{HSI}/1$

110= $F_{SYS} = F_{HSI}/2$

101= $F_{SYS} = F_{HSI}/4$ (默认)

100= $F_{SYS} = F_{HSI}/8$

011= $F_{SYS} = F_{HSI}/16$

010= $F_{SYS} = F_{HSI}/32$

001= $F_{SYS} = F_{HSI}/64$

000= F_{SYS} = 32KHz(F_{LSI})

Bit3~Bit2 未用

Bit1 SWDTEN： 软件使能或禁止看门狗定时器位

1= 使能 WDT

0= 不使能 WDT

Bit0 未用

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

3. 系统时钟

3.1 概述

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

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

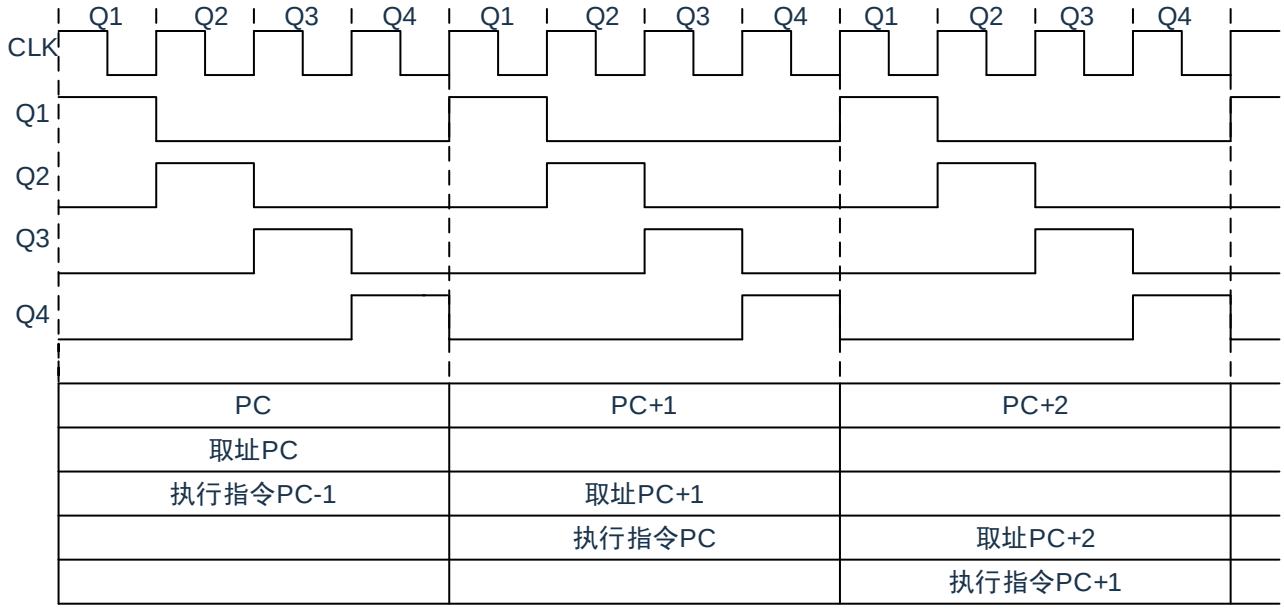


图 3-1：时钟与指令周期时序图(F_{CPU}_DIV=4T)

下面列出 F_{CPU}_DIV=4T 时系统工作频率与指令速度的关系：

系统工作频率（F_{SYS}）	双指令周期	单指令周期
1MHz	8μs	4μs
2MHz	4μs	2μs
4MHz	2μs	1μs
8MHz	1μs	500ns
16MHz	500ns	250ns

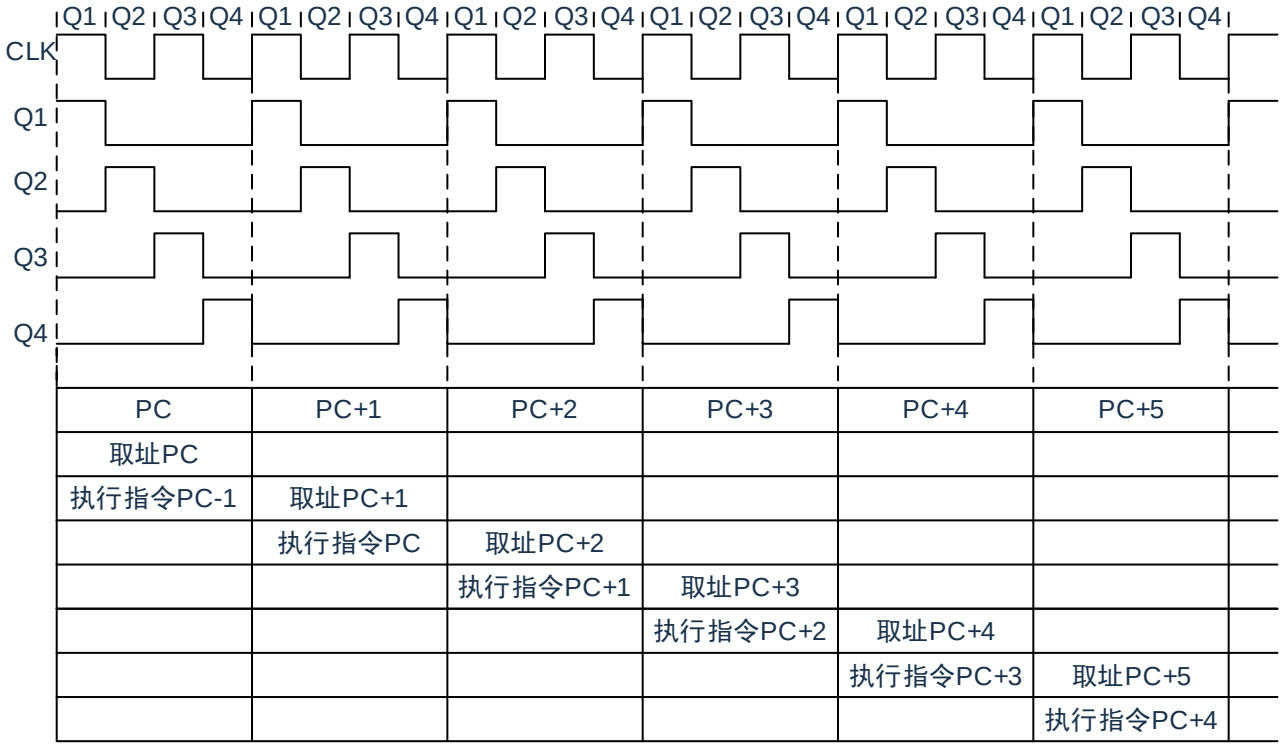


图 3-2：时钟与指令周期时序图(F_{CPU}_DIV=2T)

下面列出 F_{CPU}_DIV=2T 时系统工作频率与指令速度的关系：

系统工作频率（F_{SYS}）	双指令周期	单指令周期
1MHz	4μs	2μs
2MHz	2μs	1μs
4MHz	1μs	500ns
8MHz	500ns	250ns
16MHz	250ns	125ns

3.2 系统振荡器

芯片有 1 种振荡方式：内部 RC 振荡。

3.2.1 内部 RC 振荡

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

3.3 起振时间

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

注：无论芯片是电源上电复位，还是其它原因引起的复位，都会存在这个起振时间。

3.4 振荡器控制寄存器

振荡器控制（OSCCON）寄存器控制系统时钟和频率选择。

振荡控制寄存器 OSCCON (14H)

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

Bit7 未用

Bit6~Bit4 IRCF<2:0>: 内部振荡器频率选择位

111= $F_{SYS} = F_{HSI}/1$

110= $F_{SYS} = F_{HSI}/2$

101= $F_{SYS} = F_{HSI}/4$ (默认)

100= $F_{SYS} = F_{HSI}/8$

011= $F_{SYS} = F_{HSI}/16$

010= $F_{SYS} = F_{HSI}/32$

001= $F_{SYS} = F_{HSI}/64$

000= F_{SYS} = 32KHz(LFINTOSC)

Bit3~Bit2 未用。

Bit1 SWDTEN： 软件使能或禁止看门狗定时器位

1= 使能 WDT

0= 不使能 WD

Bit0 未用

3.5 时钟框图

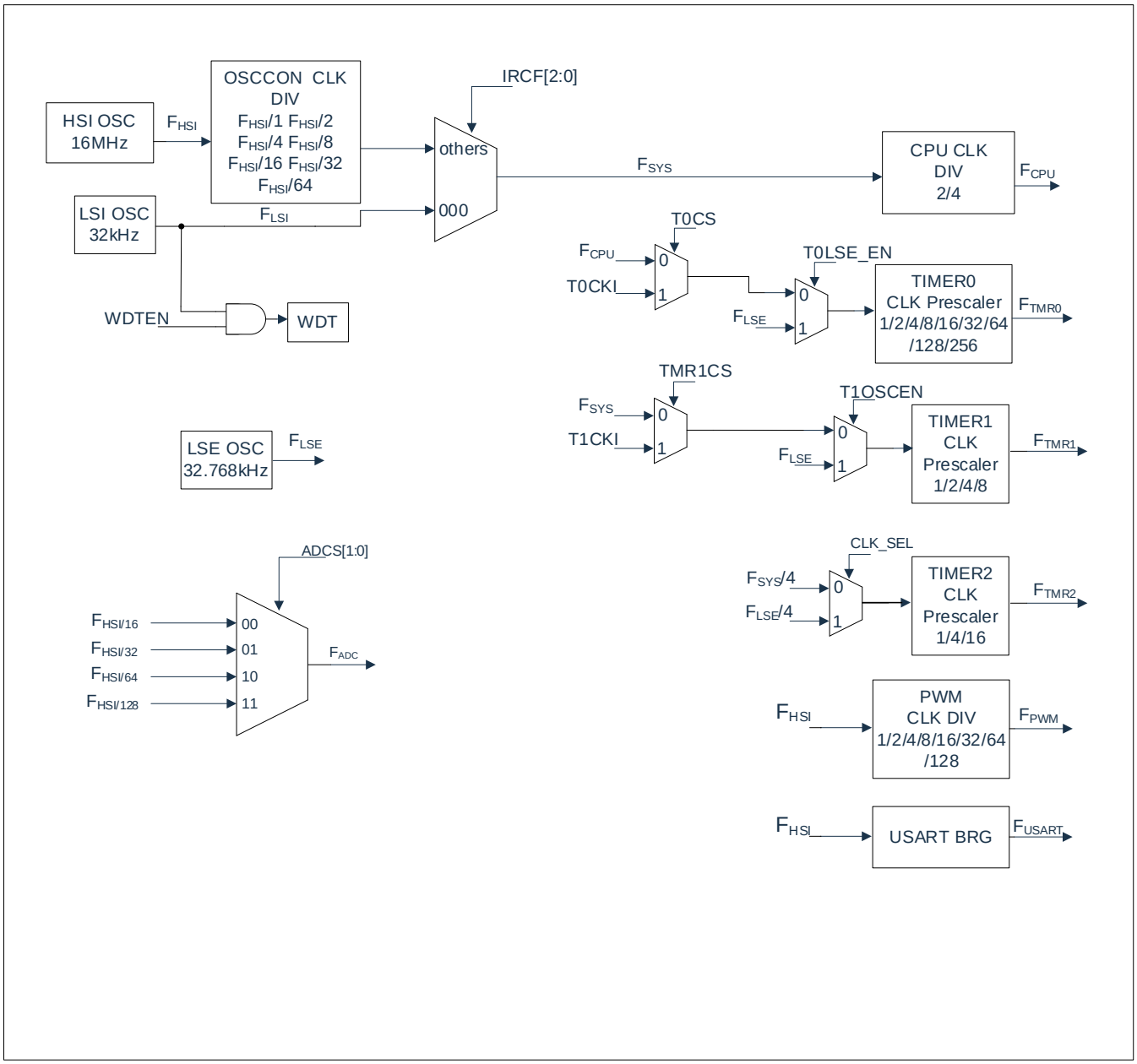


图 3-3：时钟框图

4. 复位

芯片可用如下 4 种复位方式：

- 上电复位
- 外部复位
- LVR 复位
- 正常工作下的看门狗溢出复位

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

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

4.1 上电复位

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

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

4.2 外部复位

SC8F073 支持外部复位功能，可以通过 CONFIG 配置 RB2 作为复位口，此时 RB2 自动使能内部弱上拉。若 RB2 被拉低，芯片将会发生复位。

4.3 掉电复位

4.3.1 掉电复位概述

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

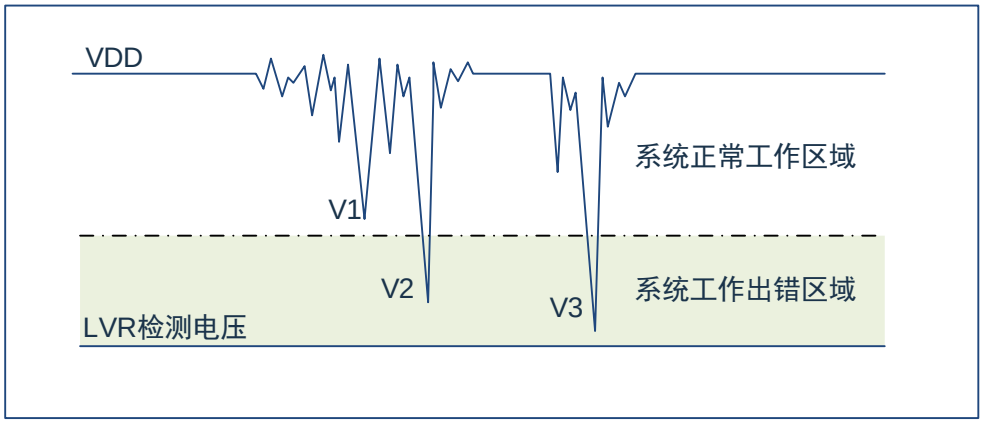


图 4-1：掉电复位示意图

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

以下情况系统可能进入死区：

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

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

4.3.2 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。

4.4 看门狗复位

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

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

关于看门狗定时器的应用问题请参看 2.8WDT 应用章节。

5. 休眠模式

5.1 进入休眠模式

执行 STOP 指令可进入休眠模式，如果 WDT 使能，那么：

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

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

5.2 从休眠状态唤醒

可以通过下列任一事件将器件从休眠状态唤醒：

1. 看门狗定时器唤醒；
2. INT 中断；
3. PORTB 电平变化中断；
4. PORTA 电平变化中断或外设中断。

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

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

5.3 使用中断唤醒

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

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

5.4 休眠模式应用举例

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

例：进入休眠的处理程序

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

5.5 休眠模式唤醒时间

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

系统主频时钟源	系统时钟分频选择(IRCF<2:0>)	休眠唤醒等待时间 T_{WAIT}
内部高速 RC 振荡（F_{HSI}）	$F_{SYS}=F_{HSI}$	$T_{WAIT}=264*1/F_{HSI}+16*1/F_{HSI}$
	$F_{SYS}= F_{HSI} /2$	$T_{WAIT}=264*2/F_{HSI}+16*1/F_{HSI}$
	…	…
	$F_{SYS}= F_{HSI} /64$	$T_{WAIT}=264*64/F_{HSI}+16*1/F_{HSI}$
内部低速 RC 振荡（$F_{LFINTOSC}$）	----	$T_{WAIT}=11/F_{LSI}$

6. I/O 端口

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，推挽或开漏输出，AN0，PWMA0，PWMB0，INT，CMP2_O	I/O
	1	施密特触发输入，推挽或开漏输出，AN1，PWMA1，PWMB1，CMP2_+，CMP2_3-	I/O
	2	施密特触发输入，推挽或开漏输出，AN2，PWMA2，PWMB2，PWMD2，CMP2_0-	I/O
	3	施密特触发输入，推挽或开漏输出，AN3，PWMA3，PWMD3，TX/CK	I/O
	4	施密特触发输入，推挽或开漏输出，AN4，PWMA4，RX/DT	I/O
	5	施密特触发输入，推挽或开漏输出，AN5，PWMC0	I/O
	6	施密特触发输入，推挽或开漏输出，AN6	I/O
	7	施密特触发输入，推挽或开漏输出，AN7	I/O
PORTB	0	施密特触发输入，推挽或开漏输出，编程数据输入/输出，AN8，PWMD0，TX/CK，INT，CMP2_2-，CMP1_O，OSCIN	I/O
	1	施密特触发输入，推挽或开漏输出，编程时钟输入，AN9， PWMB4，PWMD1，RX/DT，T1CKI，CMP2_1-，CMP1_3-，CMP1_+，OSCOUT	I/O
	2	施密特触发输入，推挽或开漏输出，AN10，PWMB3，PWMD4，T0CKI，CMP1_0-，MCLR	I/O
	3	施密特触发输入，推挽或开漏输出，AN11，PWMD2	I/O
	4	施密特触发输入，推挽或开漏输出，AN12，PWMC4，PWMD3，CMP1_1-	I/O
	5	施密特触发输入，推挽或开漏输出，AN13，PWMC3，PWMC3，CMP1_2-	I/O
	6	施密特触发输入，推挽或开漏输出，AN14，PWMC2	I/O
	7	施密特触发输入，推挽或开漏输出，AN15，PWMC1，T1G	I/O
PORTC	0	施密特触发输入，推挽输出，编程时钟输入，AN16	I/O
	1	施密特触发输入，推挽输出，编程数据输入/输出，AN17	I/O

＜表 6-1：端口配置总概＞

6.1 I/O 口结构图

6.1.1 PORTA I/O 口结构图

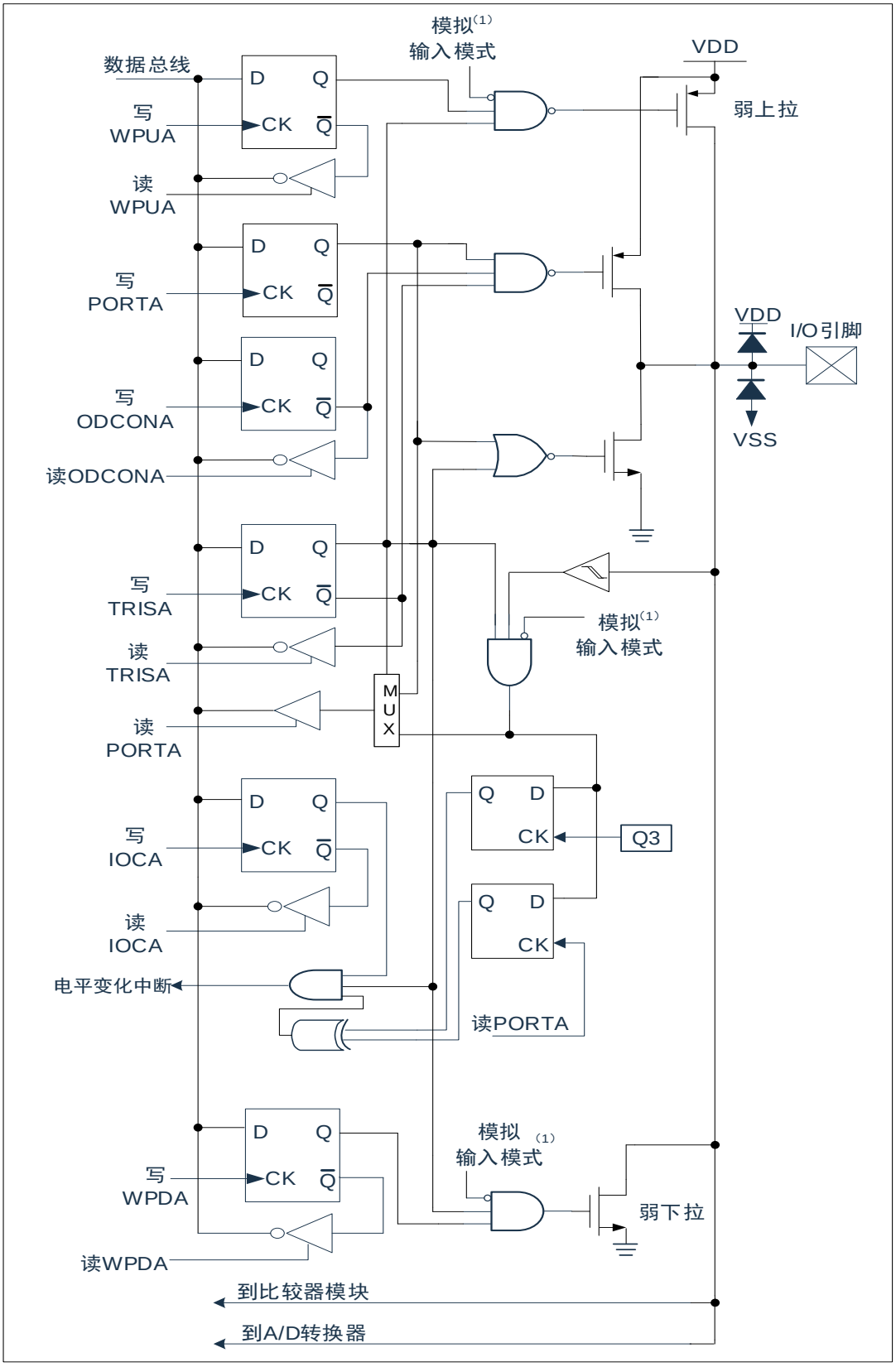


图 6-1：PORTA I/O 口结构图

6.1.2 PORTB I/O 口结构图

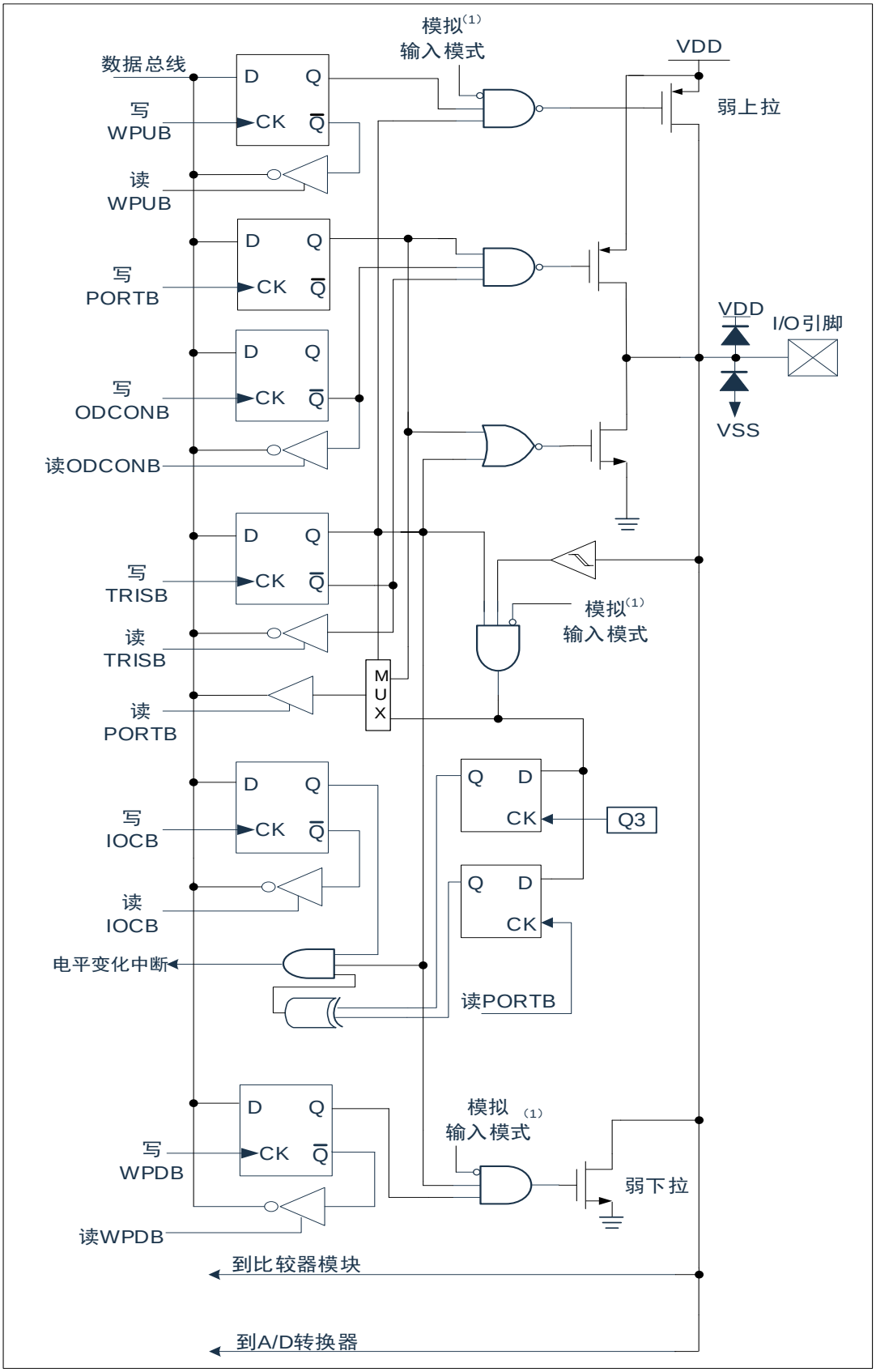


图 6-2：PORTB I/O 口结构图

6.1.3 PORTC I/O 口结构图

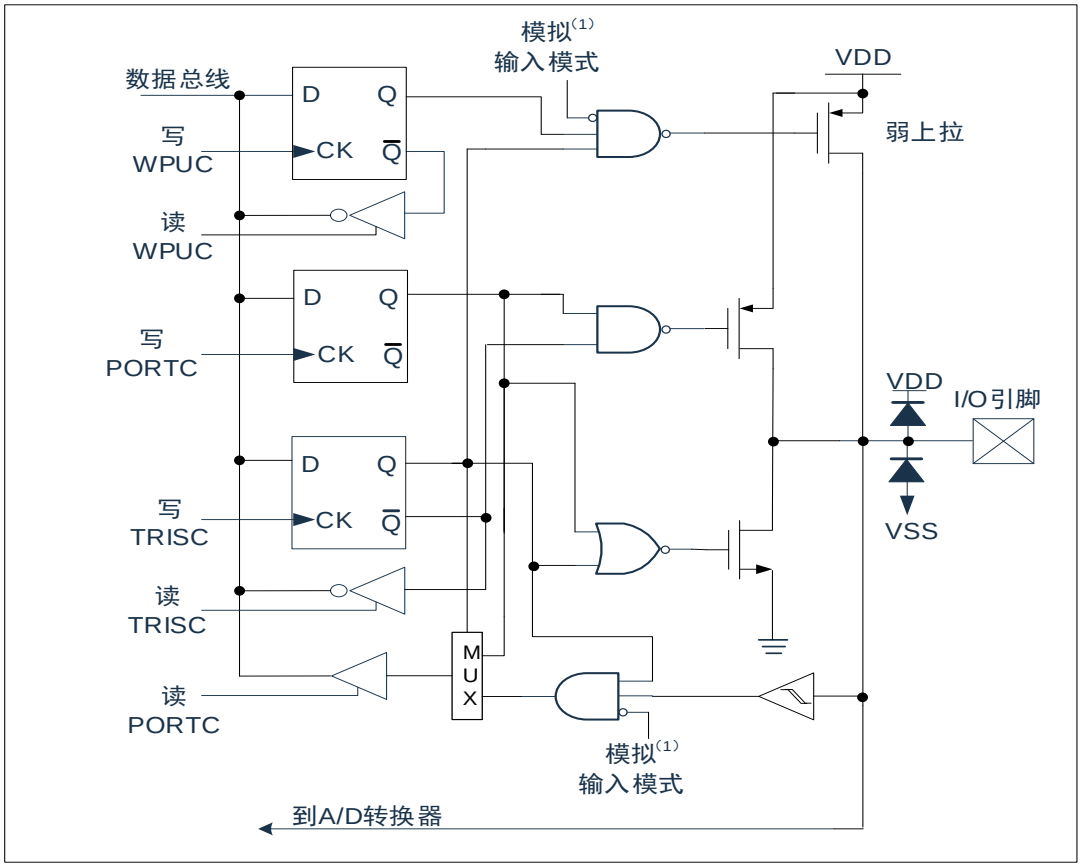


图 6-3：PORTC I/O 口结构图

注：ANx_EN 或 ANSx 决定模拟输入模式。

6.2 PORTA

6.2.1 PORTA 数据及方向

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、WPDA、ODCONA、IOCA、ANSEL0 等。

PORTA 数据寄存器 PORTA(86H)

86H	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'00110000'    ;设置PORTA<3:0>为输出口，PORTA<5:4>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<5:4>为输入口，所以赋0或1都没影响
```

6.2.2 PORTA 开漏输出控制

每个 PORTA 引脚都有可单独配置的开漏输出使能控制位。

PORTA 开漏输出使能寄存器 ODCONA(8CH)

8CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ODCONA	ODCONA7	ODCONA6	ODCONA5	ODCONA4	ODCONA3	ODCONA2	ODCONA1	ODCONA0
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 ODCONA<7:0>: PORTA 开漏输出使能
1= 使能开漏输出
0= 不使能开漏输出

6.2.3 PORTA 模拟选择控制

ANSEL0 寄存器用于将 I/O 引脚的输入模式配置为模拟模式。将 ANSEL0 中适当的位置 1 将导致对相应引脚的所有数字读操作返回 0，并使引脚的模拟功能正常工作。ANSEL0 位的状态对数字输出功能没有影响。TRIS 清零且 ANSEL0 置 1 的引脚仍作为数字输出，但输入模式将成为模拟模式。这会导致在受影响的端口上执行读—修改—写操作时产生不可预计的结果。

PORTA 模拟选择寄存器 ANSEL0(93H)

93H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ANSEL0	ANS7	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0
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 ANS<7:0>: 模拟选择位,分别选择引脚 AN<7:0>的模拟或数字功能
1= 模拟输入，引脚被分配为模拟输入
0= 数字 I/O，引脚被分配给端口或特殊功能

6.2.4 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。当将端口引脚配置为输出或模拟输入时，其弱上拉会自动切断。

PORTA 上拉电阻寄存器 WPUA(88H)

88H	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或模拟输入，将自动禁止弱上拉。

6.2.5 PORTA 下拉电阻

每个 PORTA 引脚都有可单独配置的内部弱下拉。控制位 WPDA<7:0>使能或禁止每个弱下拉。当将端口引脚配置为输出或模拟输入时，其弱下拉会自动切断。

PORTA 下拉电阻寄存器 WPDA(87H)

87H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
WPDA	WPDA7	WPDA6	WPDA5	WPDA4	WPDA3	WPDA2	WPDA1	WPDA0
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 WPDA<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或模拟输入，将自动禁止弱下拉。

6.2.6 PORTA 电平变化中断

所有的 PORTA 引脚都可以被单独配置为电平变化中断引脚。控制位 IOCA<7:0>允许或禁止每个引脚的该中断功能。上电复位时禁止引脚的电平变化中断功能。

对于已允许电平变化中断的引脚，则将该引脚上的值与上次读 PORTA 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 PIR1 寄存器中的 PORTA 电平变化中断标志位（RAIF）置 1。

该中断可将器件从休眠中唤醒，用户可在中断服务程序中通过以下步骤清除中断：

- 对 PORTA 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 RAIF 清零。

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

注：如果在执行读取操作时（Q2 周期的开始）I/O 引脚的电平发生变化，则 RAIF 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

PORTA 电平变化中断寄存器 IOCA(89H)

89H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IOCA	IOCA7	IOCA6	IOCA5	IOCA4	IOCA3	IOCA2	IOCA1	IOCA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 IOCA<7:0> PORTA 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

8.3 TIMER0 相关寄存器

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

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

8 位定时器/计数器 TMR0（81H）

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

预分频器控制寄存器 OPTION_REG（01H）

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

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

PS2	PS1	PS0	TMR0 分频比	WDT 分频比 (WDT_DIV=DISABLE)	WDT 分频比 (WDT_DIV=ENABLE)
0	0	0	1:2	1:1	1:3
0	0	1	1:4	1:2	1:6
0	1	0	1:8	1:4	1:12
0	1	1	1:16	1:8	1:24
1	0	0	1:32	1:16	1:48
1	0	1	1:64	1:32	1:96
1	1	0	1:128	1:64	1:192
1	1	1	1:256	1:128	1:384

9. 定时计数器 TIMER1

9.1 TIMER1 概述

TIMER1 模块是一个 16 位定时器/计数器，具有以下特性：

- 16 位定时器/计数器寄存器（TMR1H:TMR1L）
- 3 位预分频器
- 同步或异步操作
- 溢出时唤醒（仅外部时钟异步模式）
- 特殊事件触发功能（带有 ECCP）
- 可编程内部或外部时钟源
- 可选 LP 振荡器(F_{LSE})
- 通过 T1G 引脚门控 TIMER1（使能计数）
- 溢出中断
- 捕捉/比较功能的时基

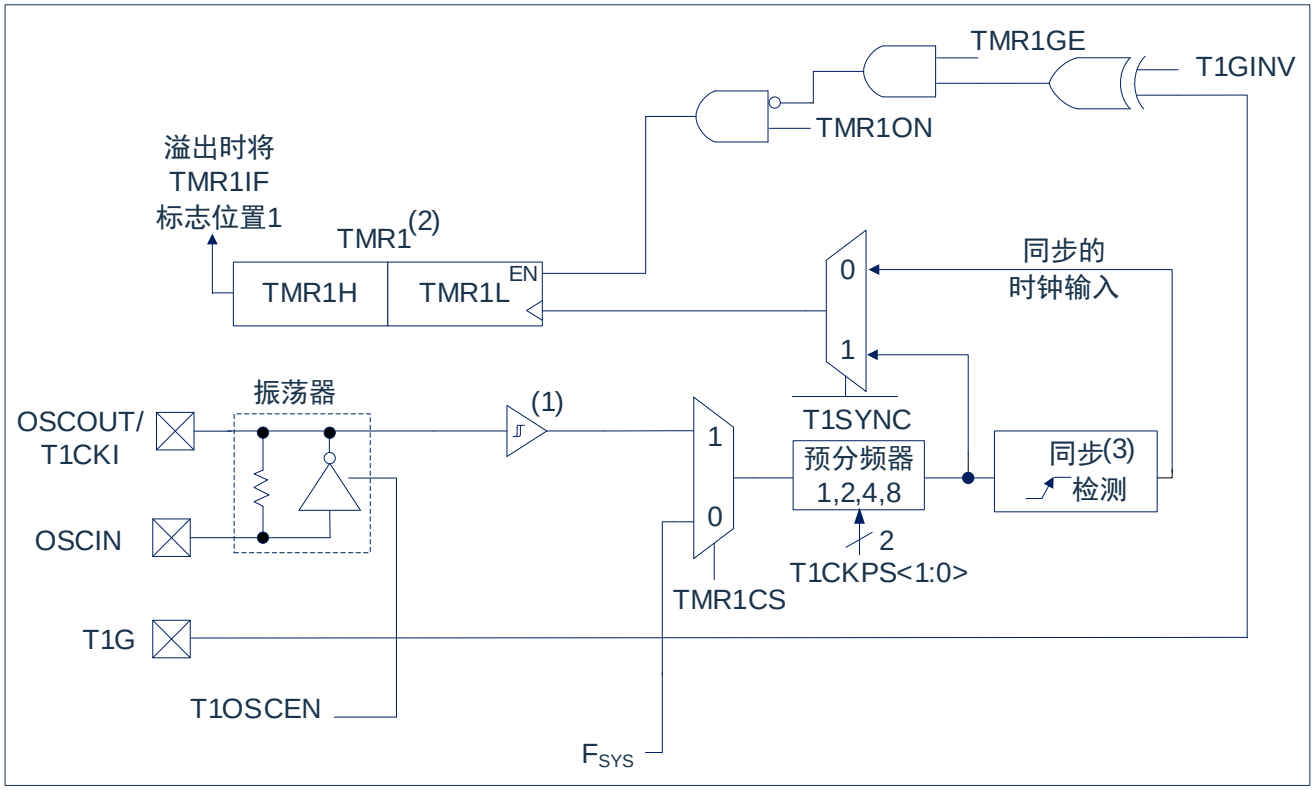


图9-1：TIMER1结构图

注：

1. ST 缓冲器在使用 LP 振荡器时处于低功耗模式，而在使用 T1CKI 时处于高速模式。
2. Timer1 寄存器在上升沿递增。
3. 休眠时不进行同步。

9.2 TIMER1 的工作原理

TIMER1 模块是一个通过一对寄存器 TMR1H：TMR1L 访问的 16 位递增计数器。写入 TMR1H 或 TMR1L 可直接更新该计数器。

当与内部时钟源一同使用时，此模块用作定时器。当与外部时钟源一同使用时，此模块可用作定时器或计数器。

9.3 时钟源选择

T1CON 寄存器的 TMR1CS 位用于选择时钟源。当 TMR1CS=0 时，时钟源的频率为 F_{SYS}。当 TMR1CS=1 时，时钟源由外部提供。

时钟源	TMR1CS
F_{SYS}	0
T1CKI 引脚	1

9.3.1 内部时钟源

选择内部时钟源后，TMR1H:TMR1L 寄存器将以 F_{SYS} 的倍数为频率递增，具体倍数由 TIMER1 预分频器决定。

9.3.2 外部时钟源

选择外部时钟源后，TIMER1 模块可作为定时器或计数器。

计数时，TIMER1 在外部时钟输入 T1CKI 的上升沿递增。此外，计数器模式下的时钟可与单片机系统时钟同步或异步。

如需一个外部时钟振荡器，TIMER1 可使用 LP 振荡器作为时钟源。

在计数器模式下，在出现以下一个或多个条件时，必须先经过一个下降沿，计数器才可以在随后的上升沿进行第一次递增计数（见图 9-2）：

- 使能 TIMER1。
- 对 TMR1H 或 TMR1L 执行了写操作。
- 禁止 TIMER1 时，T1CKI 为高电平；当重新使能 TIMER1 时，T1CKI 为低电平。

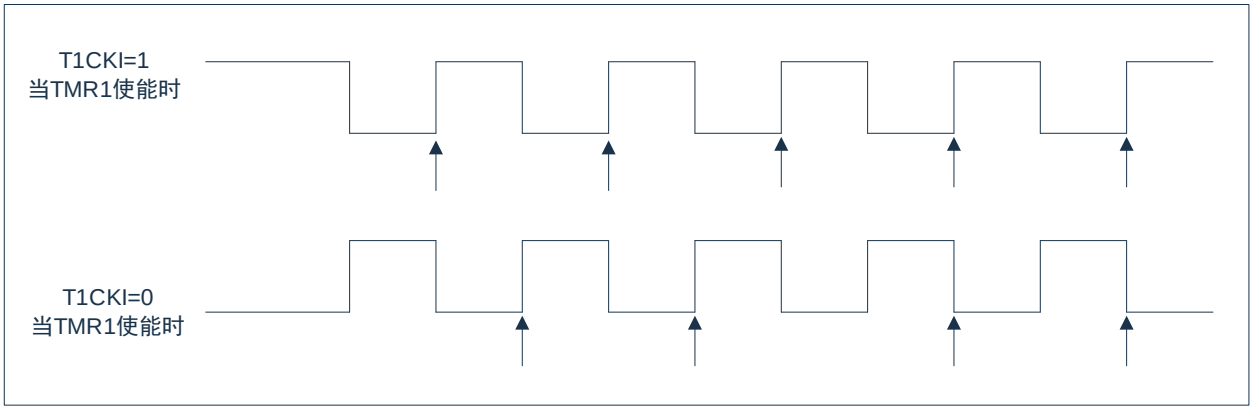


图 9-2：TIMER1 的递增边沿

注：
1) 箭头表示计数器递增。
2) 在计数器模式下，必须先经过一个下降沿，计数器才可以在随后的上升沿进行第一次递增计数。

9.4 TIMER1 预分频器

TIMER1 具有四种预分频比选择，允许对输入时钟进行 1、2、4 或 8 分频。T1CON 寄存器的 T1CKPS 位控制预分频计数器。不能直接对预分频计数器进行读或写操作；但是，通过写入 TMR1H 或 TMR1L 可清零预分频计数器。

9.5 TIMER1 振荡器

在 T1OSI（输入）引脚和 T1OSO（放大器输出）引脚之间连接有一个内置的低功耗 32.768KHz 振荡器。将 T1CON 寄存器的 T1OSCEN 控制位置 1 可使能该振荡器。此振荡器将在休眠模式下继续运行，但是必须使 TIMER1 选择为异步计数模式。

TIMER1 振荡器与 LP 振荡器完全相同。用户必须提供软件延时，以保证振荡器正常振荡。

使能 TIMER1 振荡器时 PORTB0 和 PORTB1 被置为模拟输入。

> 注：振荡器需要经过一段起振和稳定时间后才能使用。因此，在使能 TIMER1 前应将 T1OSCEN 置 1 并经过适当的延时。

9.6 在异步计数器模式下的 TIMER1 工作原理

如果 T1CON 寄存器中的控制位 T1SYNC 被置 1，外部时钟输入就不同步。定时器继续进行与内部相位时钟异步的递增计数。在休眠状态下定时器仍将继续运行，并在溢出时产生中断，从而唤醒处理器。但是，在用软件对定时器进行读/写操作时应该特别小心（请参见第 9.6.1 节“异步计数器模式下对 TIMER1 的读写”）。

> 注：
> 1) 当从同步操作切换到异步操作时，有可能漏过一个递增。
> 2) 当从异步操作切换到同步操作时，有可能产生一个误递增。

9.6.1 异步计数器模式下对 TIMER1 的读写操作

当定时器采用外部异步时钟工作时，对 TMR1H 或 TMR1L 的读操作将确保有效（由硬件负责）。但用户应牢记，用读两个 8 位值来读一个 16 位定时器本身就存在问题，这是因为在两次读操作之间定时器可能会溢出。

对于写操作，建议用户停止定时器后再写入所需数值。当寄存器正在递增计数时，向定时器的寄存器写入数据可能会产生写争用。从而会在 TMR1H:TMR1L 这对寄存器中产生不可预测的值。

9.7 TIMER1 门控

可用软件将 TIMER1 门控信号源配置为 T1G 引脚，这让器件可以直接使用 T1G 为外部事件定时。

> 注：必须将 T1CON 寄存器的 TMR1GE 位置 1 以使用 TIMER1 的门控信号。

使用 T1CON 寄存器的 T1GINV 位来设置 TIMER1 门控信号的极性，门控信号可以来自 T1G 引脚。该位可将 TIMER1 配置为对事件之间的高电平时间或低电平时间进行计时。

9.8 TIMER1 中断

一对 TIMER1 寄存器（TMR1H:TMR1L）递增计数到 FFFFH 后，将溢出返回 0000H。当 TIMER1 溢出时，PIR1 寄存器的 TIMER1 中断标志位被置 1。要允许该溢出中断，用户应将以下位置 1：

- PIE1 寄存器中的 TIMER1 中断允许位；
- INTCON 寄存器中的 PEIE 位；
- INTCON 寄存器中的 GIE 位。

在中断服务程序中将 TMR1IF 位清零可以清除该中断。

注：再次允许该中断前，应将 TMR1H:TMR1L 这对寄存器以及 TMR1IF 位清零。

9.9 休眠期间的 TIMER1 工作原理

只有设置为异步计数器模式时，TIMER1 才可在休眠模式下工作。在该模式下，可使用外部晶振或时钟源使计数器进行递增计数。通过如下设置使定时器能够唤醒器件：

- T1CON 寄存器中的 TMR1ON 位必须置 1；
- PIE1 寄存器中的 TMR1IE 位必须置 1；
- INTCON 寄存器中的 PEIE 位必须置 1。

器件将在溢出时被唤醒并执行下一条指令。如果 INTCON 寄存器中的 GIE 位置 1，器件将调用中断服务程序（0004h）。

9.10 TIMER1 相关寄存器

有 3 个寄存器与 TIMER1 相关，分别是数据寄存器 TMR1L、TMR1H，控制寄存器 T1CON。

TIMER1 数据低位寄存器 TMR1L(10CH)

10CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR1L	TMR1<7:0>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	X	X	X	X	X	X	X	X

Bit7~Bit0　　TMR1<7:0>:　TIMER1 数据低 8 位

TIMER1 数据高位寄存器 TMR1H(10DH)

10DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TMR1H	TMR1<15:8>							
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　　TMR1<15:8>:　TIMER1 数据高 8 位

TIMER1 控制寄存器 T1CON(101H)

101H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
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　　T1GINV:　TIMER1 门控信号极性位；
1=　TIMER1 门控信号高电平有效（当门控信号为高电平时 TIMER1 计数）；
0=　TIMER1 门控信号低电平有效（当门控信号为低电平时 TIMER1 计数）。

Bit6　　TMR1GE:　TIMER1 门控使能位。
如果TMR1ON=0，此位被忽略。
如果 TMR1ON=1:　1=TIMER1 计数由 TIMER1 门控功能控制；
0=TIMER1 始终计数。

Bit5~Bit4　　T1CKPS<1:0>:　TIMER1 输入时钟预分频比选择位；
11=　1:8 预分频比；
10=　1:4 预分频比；
01=　1:2 预分频比；
00=　1:1 预分频比。

Bit3　　T1OSCEN:　LP 振荡器使能控制位；
1=　使能 LP 振荡器作为 TIMER1 的时钟源；
0=　LP 振荡器关闭。

Bit2　　T1SYNC:　TIMER1 外部时钟输入同步控制位。
TMR1CS=1:　1= 不与外部时钟输入同步；
0= 与外部时钟输入同步。
TMR1CS=0：忽略此位，TIMER1 使用内部时钟。

Bit1　　TMR1CS:　TIMER1 时钟源选择位；
1=　来自 LP 振荡器时钟源或来自 T1CKI 引脚的时钟源（上升沿触发）；
0=　内部时钟源 F_{SYS}。

Bit0　　TMR1ON:　TIMER1 使能位；
1=　使能 TIMER1；
0=　禁止 TIMER1。

10. 定时计数器 TIMER2

10.1 TIMER2 概述

TIMER2 模块是一个 8 位定时器/计数器，具有以下特性：

1. 8 位定时器寄存器（TMR2）
2. 8 位周期寄存器（PR2）
3. TMR2 与 PR2 匹配时中断
4. 软件可编程预分频比（1:1、1:4 和 1:16）
5. 软件可编程后分频比（1:1 至 1:16）
6. 可选外部 32.768KHz 振荡时钟（F_{LSE}）

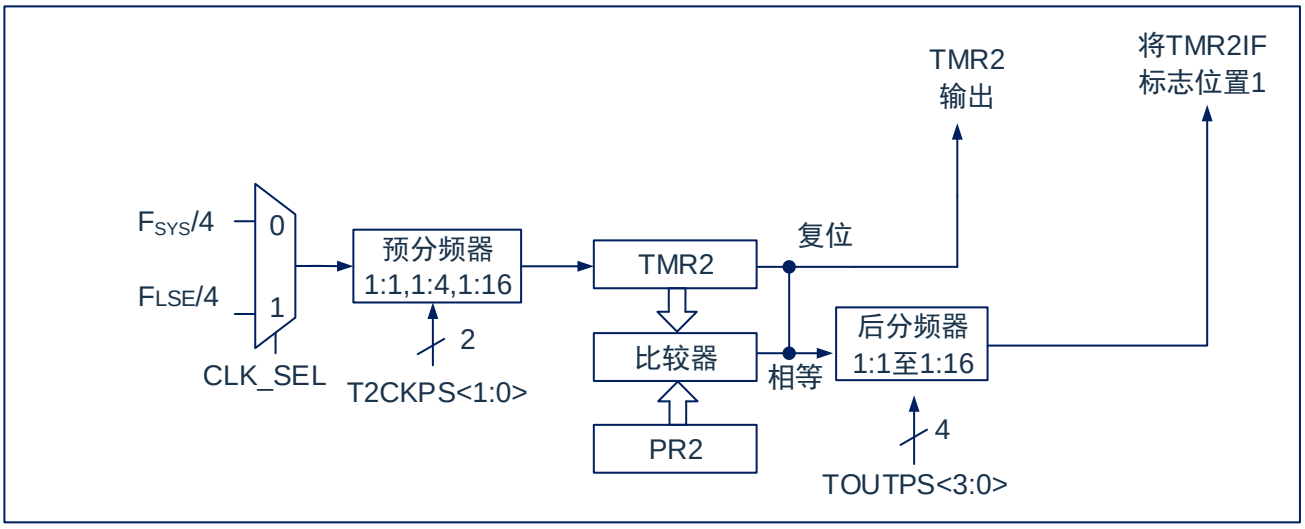


图 10-1：TIMER2 框图

10.2 TIMER2 的工作原理

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

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

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

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

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

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

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

预分频器和后分频器计数器在以下情况下被清零：

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

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

10.3 TIMER2 相关的寄存器

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

TIMER2 数据寄存器 TMR2（12H）

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

TIMER2 周期寄存器 PR2（11H）

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

TIMER2 控制寄存器 T2CON(13H)

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

Bit7 CLK_SEL: 时钟源选择
1= 选择外部 F_{LSE}/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

11. 10 位 PWM 模块

芯片包含一个 10 位 PWM 模块，可配置为 4 路共用周期、独立占空比的输出，和 1 路独立周期、独立占空比的输出，或 2 组互补输出。

11.1 引脚配置

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

11.2 相关寄存器说明

PWM 控制寄存器 PWMCON0(15H)

15H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCON0	CLKDIV<2:0>			PWM4EN	PWM3EN	PWM2EN	PWM1EN	PWM0EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

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

Bit4~Bit0 PWM0/1/2/3/4EN: PWM0/1/2/3/4使能位
1= 使能PWM0/1/2/3/4
0= 禁止PWM0/1/2/3/4

PWM 控制寄存器 PWMCON1(16H)

16H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMCON1	PWMIO_SEL<1:0>		PWM2DTEN	PWM0DTEN	---	---	DT_DIV<1:0>	
R/W	R/W	R/W	R/W	R/W	---	---	R/W	R/W
复位值	0	0	0	0	---	---	0	0

Bit7~Bit6 PWMIO_SEL： PWM IO分组选择
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= $F_{HSI}/8$
10= $F_{HSI}/4$
01= $F_{HSI}/2$
00= $F_{HSI}/1$

注： 选择 PWMD 组时，PWM2、PWM3 可通过 config 配置在 RB3、RB4 或者 RA2、RA3。

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 PWM0/1/2/3/4DIR PWM输出取反控制位
1= PWM0/1/2/3/4取反输出
0= PWM0/1/2/3/4正常输出

PWM0~PWM3 周期低位寄存器 PWMTL (17H)

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

Bit7~Bit0 PWMT<7:0>: PWM0~PWM3周期低8位

PWM4 周期低位寄存器 PWMT4L(1CH)

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

Bit7~Bit0 PWM4T<7:0>: PWM4周期低8位

周期高位寄存器 PWMTH (18H)

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

Bit7~Bit6 未用。
Bit5~Bit4 PWMD4<9:8>: PWM4占空比高2位
Bit3~Bit2 PWM4T<9:8>: PWM4周期高2位
Bit1~Bit0 PWMT<9:8>: PWM0~PWM3周期高2位

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

PWM0 占空比低位寄存器 PWMD0L (19H)

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

Bit7~Bit0 PWMD0<7:0>: PWM0占空比低8位

PWM1 占空比低位寄存器 PWMD1L (1AH)

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

Bit7~Bit0 PWMD1<7:0>: PWM1占空比低8位

PWM2 占空比低位寄存器 PWMD2L (9BH)

9BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD2L	PWMD2<7:0>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 PWMD2<7:0>: PWM0占空比低8位

PWM3 占空比低位寄存器 PWMD3L (9CH)

9CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWMD3L	PWMD3<7:0>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 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>							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit0 PWMD4<7:0>: PWM4占空比低8位

PWM0 和 PWM1 占空比高位寄存器 PWMD01H (1EH)

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

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

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

PWM2 和 PWM3 占空比高位寄存器 PWMD23H (9EH)

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

Bit7~Bit6 未用。
Bit5~Bit4 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

注：写入 PWMD2<9:8>并不能立即生效，需有写入 PWMD2L 操作后才能生效。写入 PWMD3<9:8>并不能立即生效，需有写入 PWMD3L 操作后才能生效。

PWM0 和 PWM1 死区时间寄存器 PWM01DT(1FH)

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

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

PWM2 和 PWM3 死区时间寄存器 PWM23DT(9DH)

9DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PWM23DT	---	---	PWM23DT<5:0>					
R/W	---	---	R/W	R/W	R/W	R/W	R/W	R/W
复位值	---	---	0	0	0	0	0	0

Bit7~Bit6 未用
Bit5~Bit0 PWM23DT<5:0>: PWM2和PWM3死区时间

11.3 10 位 PWM 寄存器写操作顺序

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

1）写高 2 位数值，此时高 2 位数值只是写入内部的缓存；

2）写低 8 位数值，此时完整的 10 位占空比数值被锁存；

3）以上操作顺序只针对 PWM0、PWM1、PWM2、PWM3、PWM4 占空比寄存器。

11.4 10 位 PWM 周期

PWM 周期是通过写 PWMTH 和 PWMTL 寄存器来指定的。

公式 1：PWM 周期计算公式：

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

注：$T_{HSI}=1/F_{HSI}$

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

- 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；（PWM4 无中断）

11.5 10 位 PWM 占空比

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

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

公式 2：脉冲宽度计算公式：

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

公式3：PWM占空比计算公式：

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

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

11.6 系统时钟频率的改变

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

11.7 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。

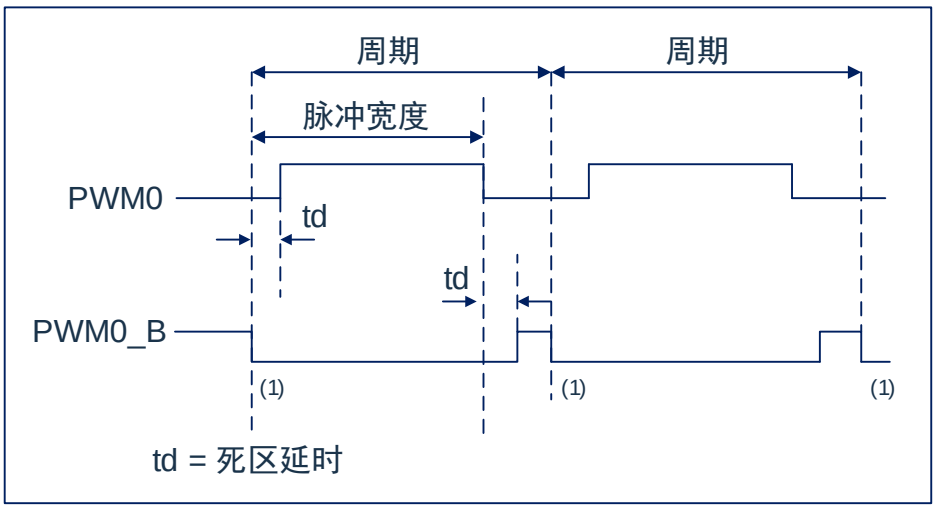


图11-1：PWM死区延时输出示例

死区时间计算公式为：

$$td=(PWMxxDT<5:0>+1)*T_{HSI}*(DT_DIV\ 分频值)$$

11.8 10 位 PWM 设置

使用 PWM 模块时应该执行以下步骤：

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

12. 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 12-1 和图 12-2 为 USART 收发器的框图。

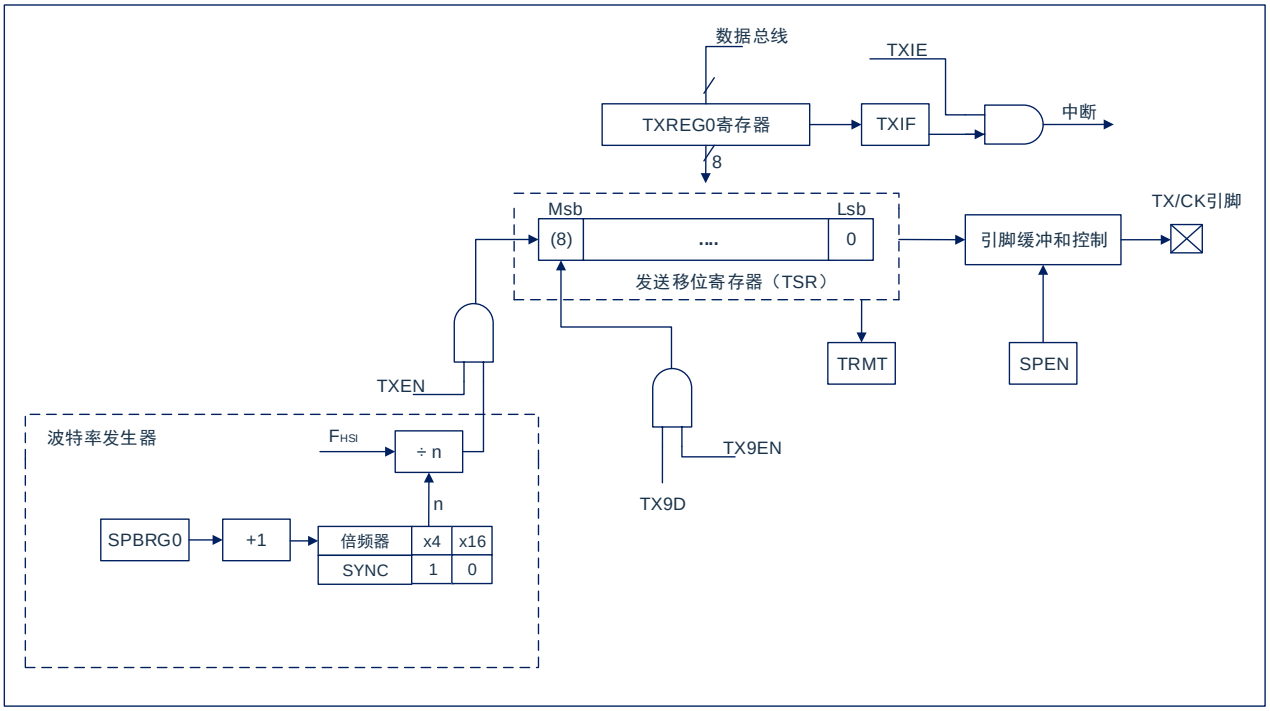


图12-1：USART发送框图

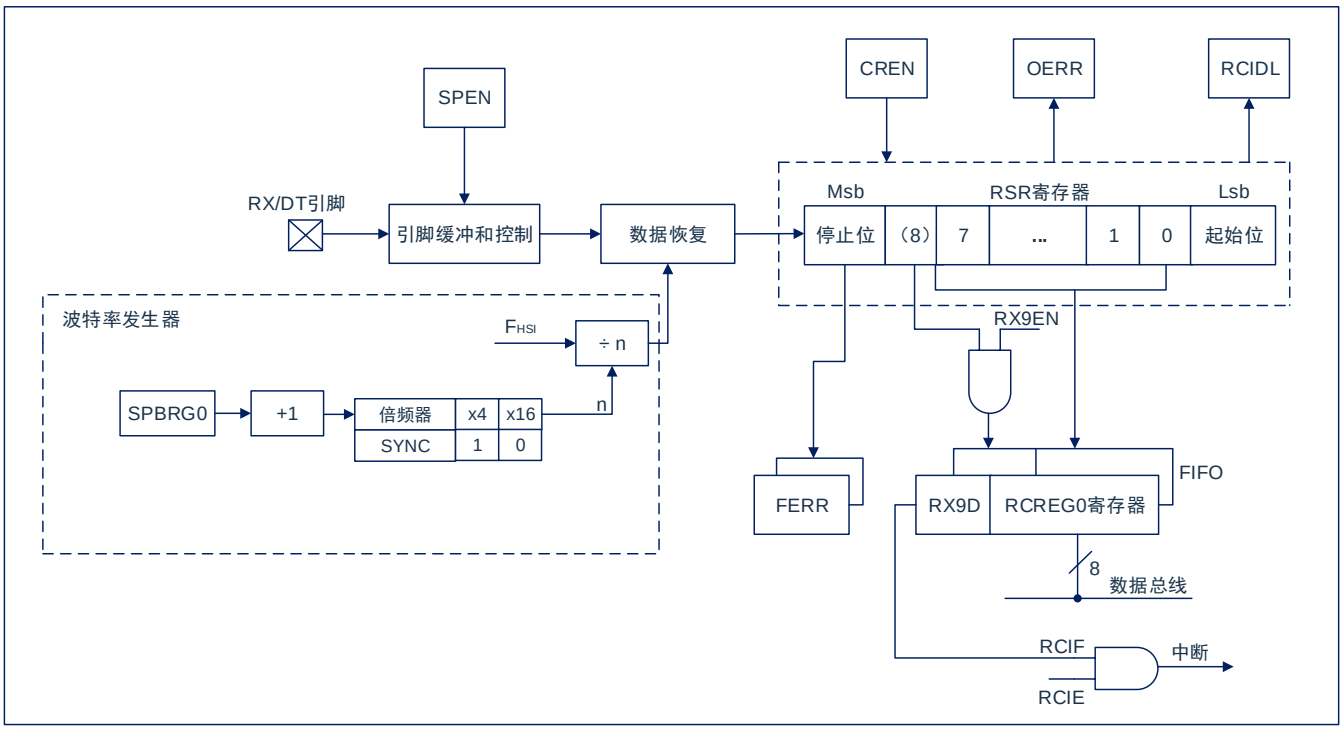


图12-2：USART接收框图

USART 模块的操作是通过 2 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）

12.1 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）。

12.1.1 USART 异步发送器

图 12-1 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。

12.1.1.1 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/CK 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/DT 引脚被自动配置为输入引脚，无需考虑相应 TRIS 位的状态。

12.1.1.2 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。

12.1.1.3 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。

12.1.1.4 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 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。

12.1.1.5 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。

12.1.1.6 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。

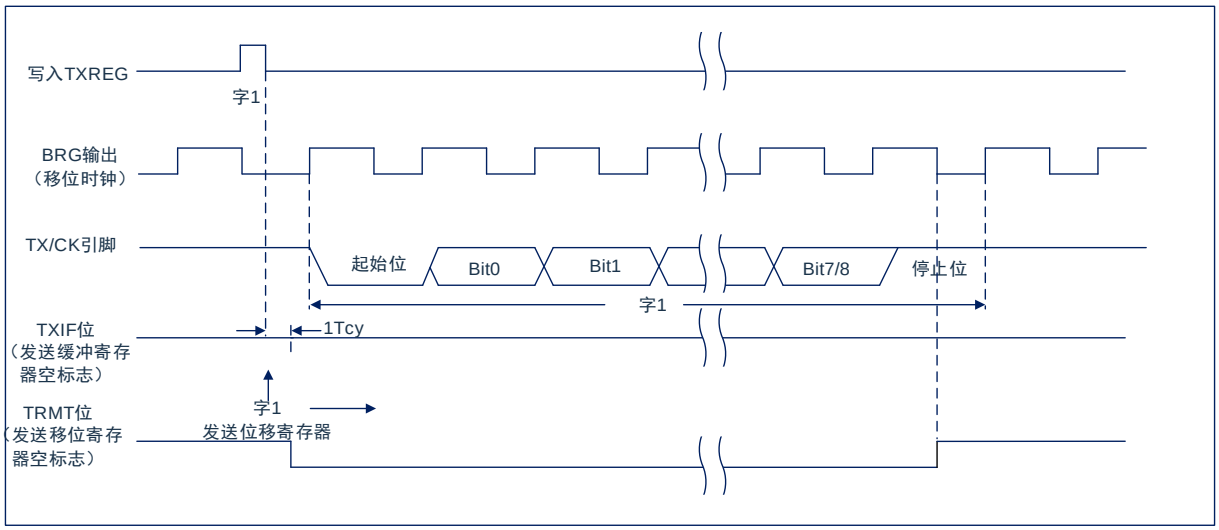


图 12-3：异步发送

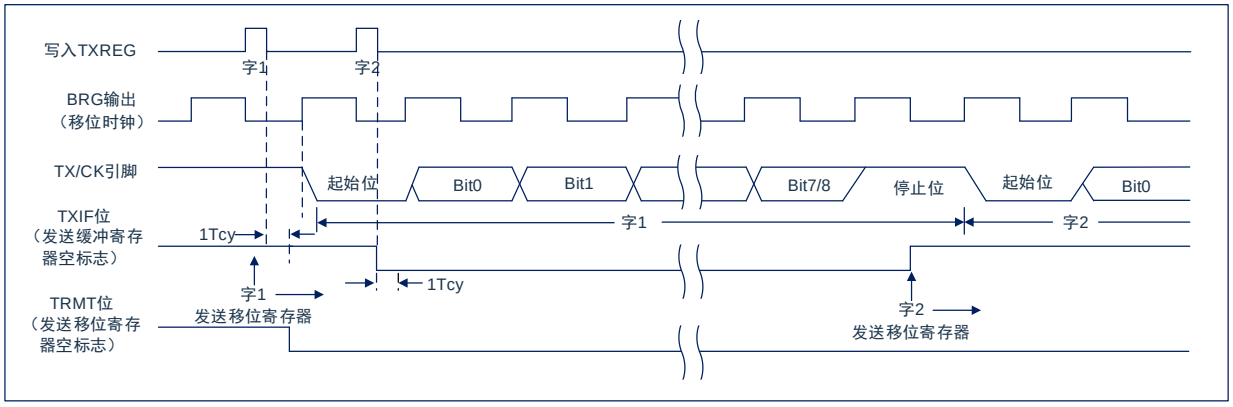


图 12-4：异步发送（背靠背）

注：本时序图显示了两次连续的发送。

12.1.2 USART 异步接收器

异步模式通常用于 RS-232 系统。图 12-2 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。

12.1.2.1 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/CK 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/DT 引脚被自动配置为输入引脚，无需考虑相应 TRIS 位的状态。

12.1.2.2 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。

12.1.2.3 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 0。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。

12.1.2.4 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。清零 RCSTA 寄存器的 CREN 位会强制清零 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。

12.1.2.5 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。

12.1.2.6 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。

12.1.2.7 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。

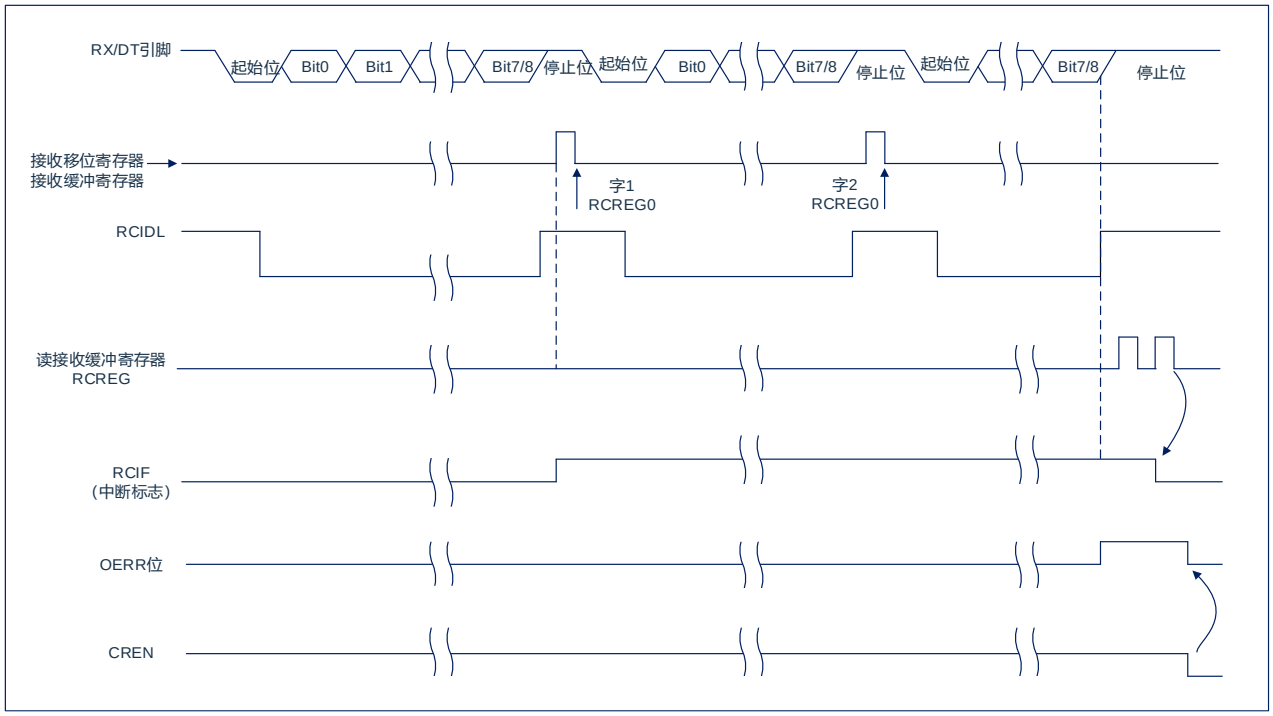


图 12-5：异步接收

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。

12.2 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。

12.3 USART 相关寄存器

发送状态和控制寄存器 TXSTA(118H)

118H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TXSTA	CSRC	TX9EN	TXEN	SYNC	SCKP	----	TRMT	TX9D
R/W	R/W	R/W	R/W	R/W	R/W	----	R	R/W
复位值	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 未用

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：同步模式下，SREN/CREN 会覆盖 TXEN 的值。

接收状态和控制寄存器 RCSTA(119H)

119H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RCSTA	SPEN	RX9EN	SREN	CREN	RCIDL	FERR	OERR	RX9D
R/W	R/W	R/W	R/W	R/W	R	R	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

发送数据寄存器 TXREG(11BH)

11BH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
TXREG								
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

接收数据寄存器 RCREG(11CH)

11CH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RCREG								
R/W	R	R	R	R	R	R	R	R
复位值	0	0	0	0	0	0	0	0

波特率数据寄存器 SPBRG(11AH)

11AH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPBRG								
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

12.4 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 12-1 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 12-2 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_{HSI} 为 16MHz，目标波特率为 9600bps，异步模式采用 8 位 BRG 的器件：

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

求解 SPBRG：

$$X=\frac{\frac{F_{HSI}}{目标波特率}}{16}-1=\frac{\frac{160000000}{9600}}{16}-1=[103.16]=103$$

$$计算波特率=\frac{16000000}{16（103+1）}=9615$$

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

表 12-1：波特率公式

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

说明：n= SPBRG 寄存器的值。

表 12-2：异步模式下的波特率

目标波特率	SYNC=0		
	F_{HSI}=16.00MHz		
	实际波特率	误差（%）	SPBRG 值
2400	----	----	----
9600	9615	0.16	103
10417	10417	0	95
14400	14286	-0.8%	68
19200	19230	0.16	51

12.5 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。

12.5.1 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。

12.5.1.1 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。

12.5.1.2 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。

12.5.1.3 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。

12.5.1.4 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。

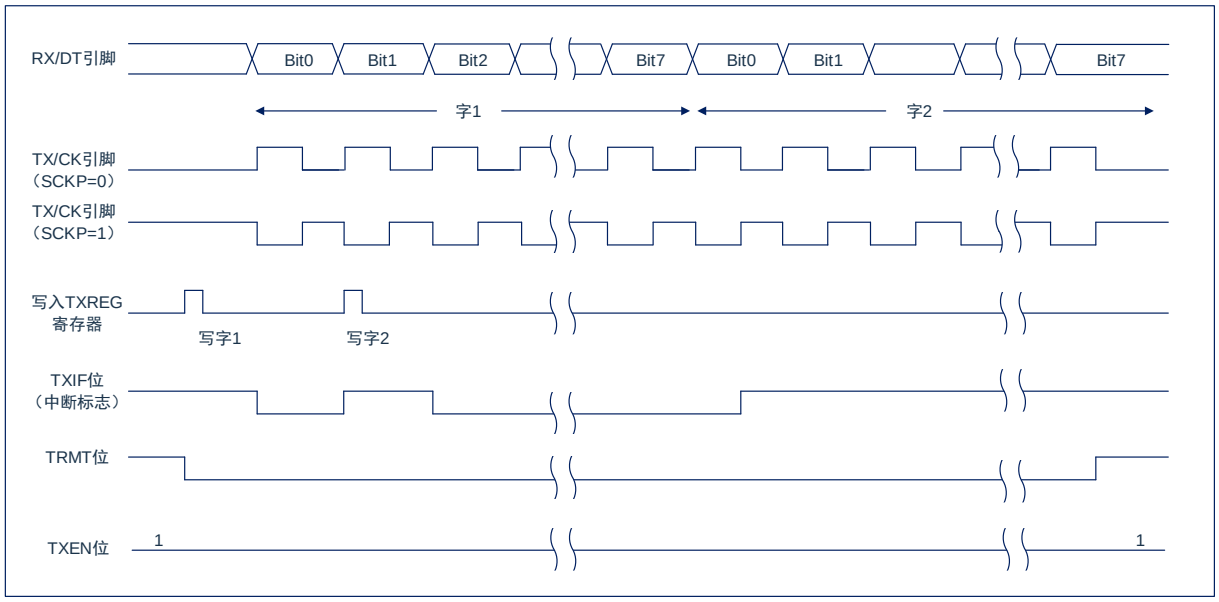


图 12-6：同步发送

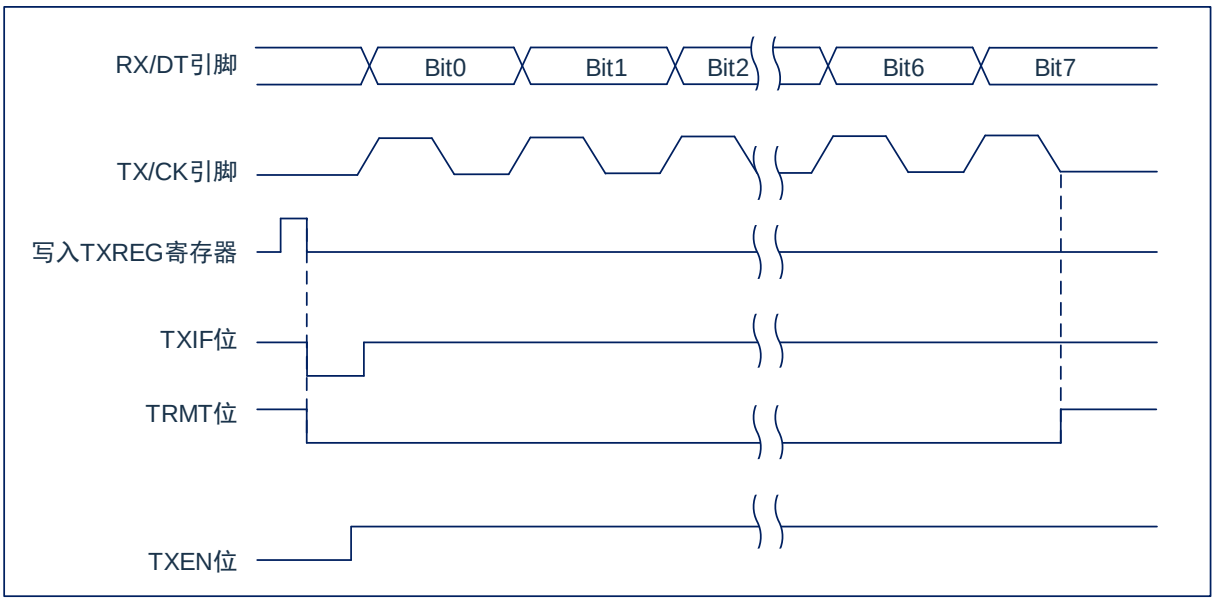


图 12-7：同步发送（通过 TXEN）

12.5.1.5 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。

12.5.1.6 从时钟

同步数据传输使用与数据线通读的独立时钟线。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。

12.5.1.7 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 RCSTA0 寄存器的 CREN 位或清零 SPEN 位以复位 USART，从而清除错误。

12.5.1.8 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。

12.5.1.9 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。

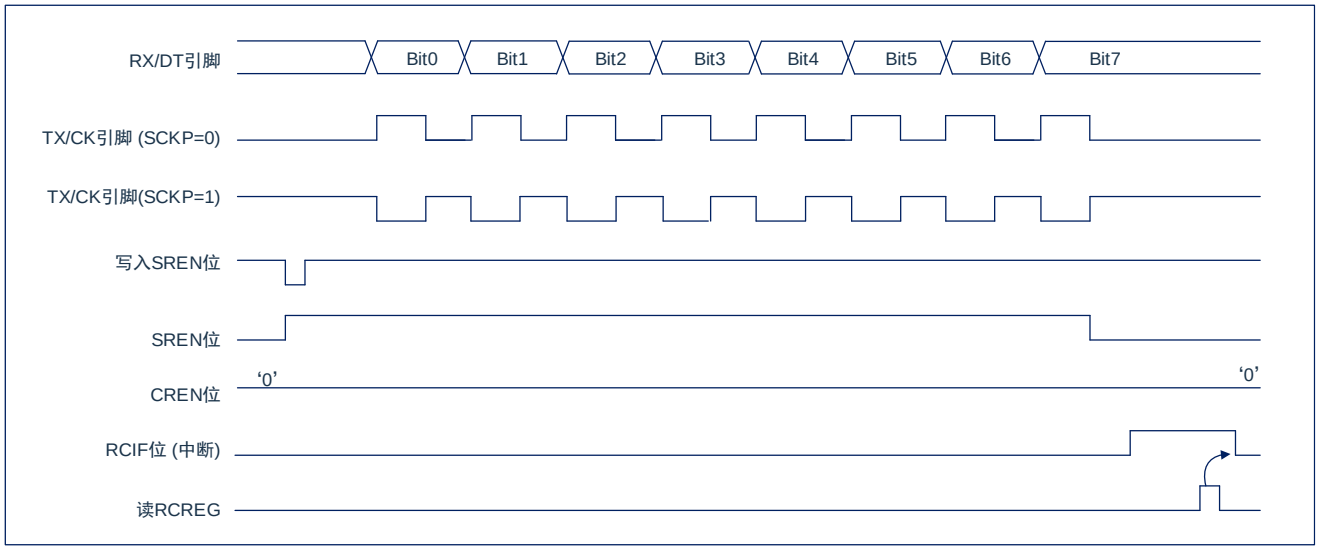


图 12-8：同步接收（主控模式，SREN）

注：时序图说明了 SREN=1 时的同步主控模式。

12.5.2 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 0，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。

12.5.2.1 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”章节。）

12.5.2.2 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。

12.5.2.3 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”。

12.5.2.4 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。
8. 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。

13. 比较器（CMP1 和 CMP2）

芯片内置 2 个比较器 CMP1 和 CMP2，两个比较器的功能和性能一样，以下描述 x 的值为 1，2。

13.1 CMPx 的功能框图

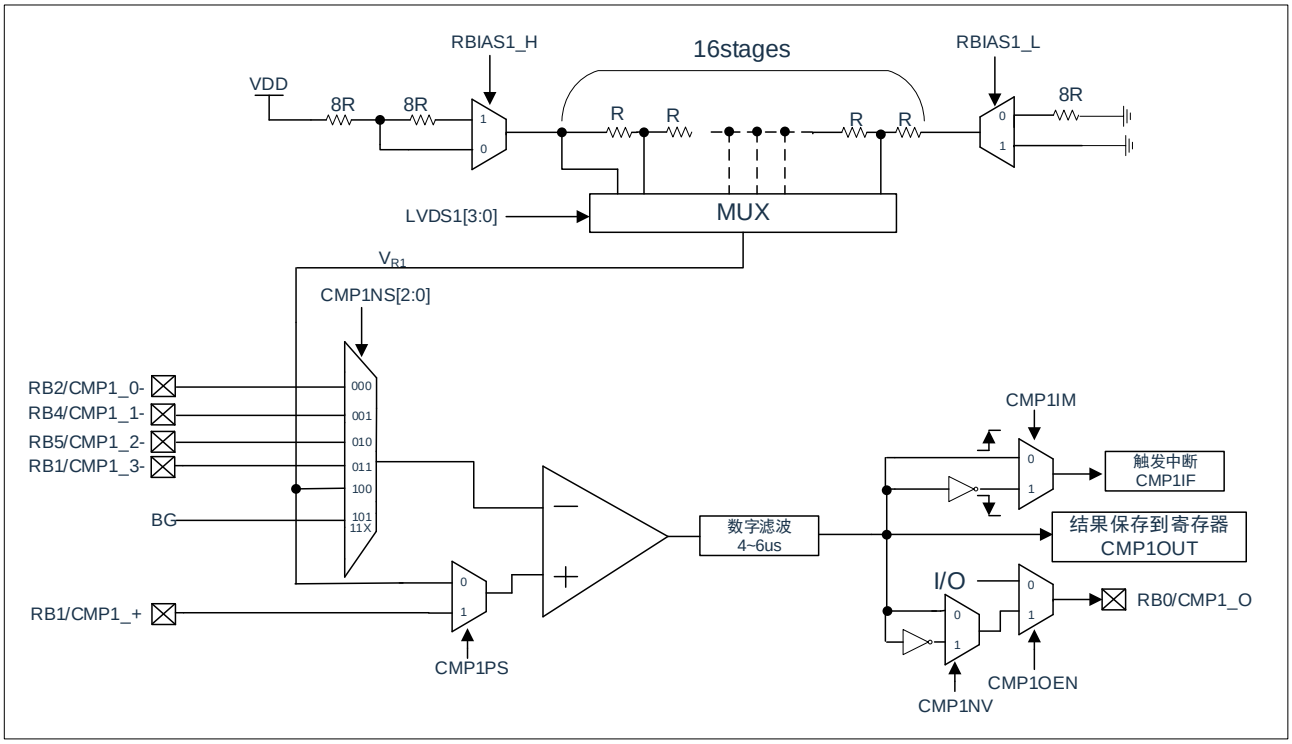


图 13-1：CMP1 的功能框图

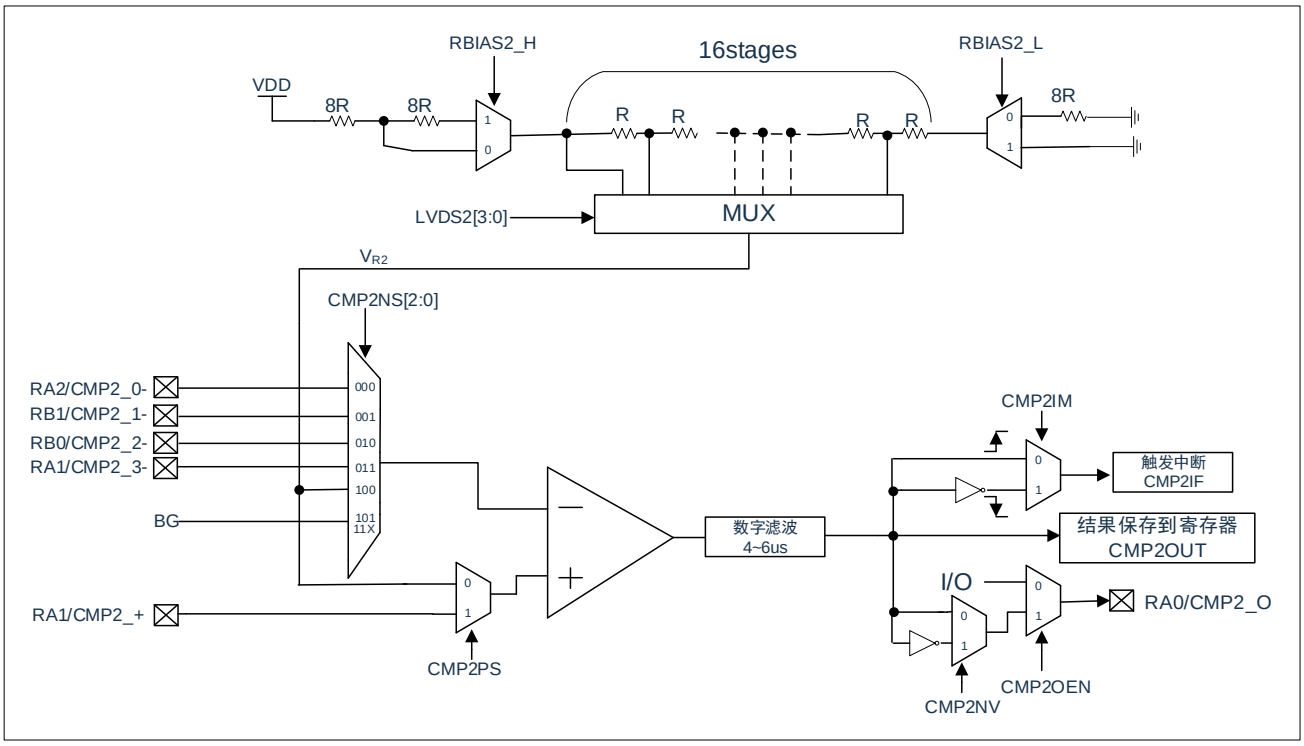


图 13-2：CMP2 的功能框图

13.2 CMPx 功能特性

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

13.3 CMPx 相关功能

13.3.1 CMPx 功能描述

CMPx 正端输入可通过配置 CMPxCON0 寄存器的 CMPxPS 位来选择 CMPx_+端口或者内部电阻分压输出信号 V_{Rx}；负端输入可通过配置 CMPxCON0 寄存器的 CMPxNS<2:0>位来选择 CMPx_m-端口、内部电阻分压输出信号 V_{Rx} 或者 1.2V BG 电压。当 CMPx 正端输入电压大于负端输入电压时，CMPx 经过数字滤波后输出 1；反之，如果 CMPx 正端输入电压小于负端输入电压，则 CMPx 经过数字滤波后输出 0。

注： CMPx_m-表示 CMPx_0-、CMPx_1-、CMPx_2-、CMPx_3-。

13.3.2 CMPx 内部电阻分压输出

CMPx 内部集成了一个电阻分压模块，参考电压为 VDD。可通过配置 CMPxCON1 寄存器的 RBIASx_H、RBIASx_L、LVDSx<3:0>等控制位的值来获得不同的电阻分压输出 V_{Rx}，根据 RBIASx_H、RBIASx_L 这两个位的不同值，V_{Rx} 有如下 4 种计算公式：

RBIASx_H	RBIASx_L	V_{Rx} 计算公式
0	0	$V_{Rx}=\frac{1}{4}*VDD+\frac{n+1}{32}*VDD$
0	1	$V_{Rx}=\frac{n+1}{24}*VDD$
1	0	$V_{Rx}=\frac{1}{5}*VDD+\frac{n+1}{40}*VDD$
1	1	$V_{Rx}=\frac{n+1}{32}*VDD$

注：n 为 LVDSx<3:0>的值，即 n=0，1，2……14，15。

13.3.3 CMPx 监测电源电压

根据 CMPx 功能框图和 13.3.2 里的公式可知，当 CMPx 的负端选择 BG 1.2V，正端选择内部电阻分压输出 V_{Rx} 时，可以通过 CMPx 来检测电源电压，当电源电压低于设定值时 CMPx 输出 0，电源电压高于设定值时 CMPx 输出 1，通过配置 RBIASx_H、RBIASx_L、LVDSx<3:0>的值可设定不同的电压监测点，具体如下表：

RBIASx_H	RBIASx_L	LVDSx[3:0]	监测值(V)	RBIASx_H	RBIASx_L	LVDSx[3:0]	监测值(V)	RBIASx_H	RBIASx_L	LVDSx[3:0]	监测值(V)
0	1	0101	4.80	0	0	0100	2.95	1	0	1101	2.18
1	0	0010	4.36	0	1	1001	2.88	0	0	1001	2.13
0	0	0000	4.27	1	0	1000	2.82	1	0	1110	2.09
0	1	0110	4.11	0	0	0101	2.74	0	1	1101	2.06
1	0	0011	4.00	1	0	1001	2.67	0	0	1010	2.02
0	0	0001	3.84	0	1	1010	2.62	1	0	1111	2.00
1	0	0100	3.69	0	0	0110	2.56	-	-	-	-
0	1	0111	3.60	1	0	1010	2.53	-	-	-	-
0	0	0010	3.49	0	0	0111	2.40	-	-	-	-
1	0	0101	3.43	1	0	1100	2.29	-	-	-	-
0	0	0011	3.20	0	0	1000	2.26	-	-	-	-
1	0	0111	3.00	0	1	1100	2.22	-	-	-	-

13.3.4 CMPx 的中断使用

若要使用 CMPx 的中断功能则可以通过以下配置步骤来开启：

- 配置寄存器 CMPxCON0 的 CMPxPS 位选择正端输入；
- 配置寄存器 CMPxCON0 的 CMPxNS<2:0>位选择负端输入；
- 配置寄存器 CMPxCON1 的 CMPxIM 位选择上升沿或者下降沿触发中断；
- 配置寄存器 CMPxCON0 的 CMPxEN 位使能比较器；
- 延时等待 10us；
- 清零 PIR1 或 PIR2 寄存器中的 CMPxIF 位；
- 置 1 PIE1 或 PIE2 寄存器的 CMPxIE 位，使能比较器中断；
- 置 1 INTCON 寄存器的 PEIE、GIE 位，开启外设中断和全局中断。

13.3.5 CMPx 中断休眠唤醒

CMPx 中断可将芯片从休眠模式下唤醒，具体程序配置可以参考以下 CMP1 的例程：

例：CMP1 中断休眠唤醒程序

```
SLEEP_MODE：
        LDIA    B'00000110'
        LD      TRISB,A         ;将 RB1/CMP1_+,RB2/CMP1_0-配置为输入口
        ...                     ;关闭其它功能
        LDIA    00H
        LD      CMP1CON0,A      ;CMP1NS<2:0>=000，负端选择 RB2/CMP1_0-
        SETB    CMP1CON0,6      ;CMP1PS=1，正端选择 RB1/CMP1_+
        SETB    CMP1CON1,6      ;AN1_EN=1，将 CMP1_+、CMP1_0-设为模拟口,以便降低休眠功耗
        SETB    CMP1CON1,7      ;CMP1IM=1，选择下降沿触发中断
        SETB    CMP1CON0,7      ;使能 CMP1
        CALL    DELAY10US       ;使能后延时等待比较器输出稳定
        CLRB    PIR1,5          ;清零 CMP1IF（必须）
        SETB    PIE1,5          ;使能 CMP1 中断
        SETB    INTCON,6        ;允许外设中断
        SETB    INTCON,7        ;开启总中断使能，唤醒后程序跳转到中断向量地址 0004H 执行
        CLRWDT                  ;清零 WDT
        STOP                    ;执行 STOP 指令
        NOP
        NOP
```

13.3.6 CMPx 结果输出引脚配置

CMPx 的输出经过数字滤波后，通过读寄存器 CMPxCON0 的 CMPxOUT 位得到当前的比较结果；还可以通过以下的配置步骤将比较结果输出到 CMPx_O 引脚：

- 将 CMPx_O 引脚配置为输出口；
- 配置寄存器 CMPxCON0 的 CMPxNV 位来选择正向输出或反向输出；
- 将寄存器 CMPxCON0 的 CMPxOEN 位置 1 来使能 CMPxOUT 输出到 CMPx_O 引脚。

13.4 CMPx 相关寄存器

CMPx 控制寄存器 CMPxCON0

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CMPxCON0	CMPxEN	CMPxPS	CMPxNS2	CMPxNS1	CMPxNS0	CMPxNV	CMPxOUT	CMPxOEN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
复位值	0	0	0	0	0	0	0	0

Bit7 CMPxEN: CMPx使能位
1= 使能CMPx
0= 禁止CMPx
Bit6 CMPxPS: CMPx正端输入选择位
1= CMPx_+端口电压
0= VDD经过内部电阻分压后的电压
Bit5~Bit3 CMPxNS<2:0>: CMPx负端输入选择位
000= CMPx_0-端口电压
001= CMPx_1-端口电压
010= CMPx_2-端口电压
011= CMPx_3-端口电压
100= VDD经过内部电阻分压后的电压
101= BG
11x= BG
Bit2 CMPxNV: CMPx_O端口输出取反控制位
1= CMPxOUT在CMPx_O端口取反输出
0= CMPxOUT在CMPx_O端口正常输出
Bit1 CMPxOUT: CMPx结果位
Bit0 CMPxOEN: CMPx_O端口输出使能位
1= 使能CMPx_O端口输出CMPxOUT
0= 禁止CMPx_O端口输出CMPxOUT

CMPx 控制寄存器 CMPxCON1

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
CMPxCON1	CMPxIM	ANx_EN	RBIASx_H	RBIASx_L	LVDSx<3:0>			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7 CMPxIM: CMPx中断触发边沿选择
1= CMPx输出的下降沿触发中断
0= CMPx输出的上升沿触发中断
Bit6 ANx_EN: 模拟口使能位，使能CMPx_+、CMPx_m-的模拟口功能
1= 模拟口
0= 数字口
Bit5 RBIASx_H: 具体用法参考CMPx的功能框图
Bit4 RBIASx_L: 具体用法参考CMPx的功能框图
Bit3~Bit0 LVDSx<3:0>: 内部电阻分压比选择位

注：ANx_EN 位只对被选作比较器功能的 IO 端口有效，另外通过 ANSEL0、ANSEL1 寄存器也可以使能 CMPx_+、CMPx_m-的模拟口功能。

14. 模数转换（ADC）

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

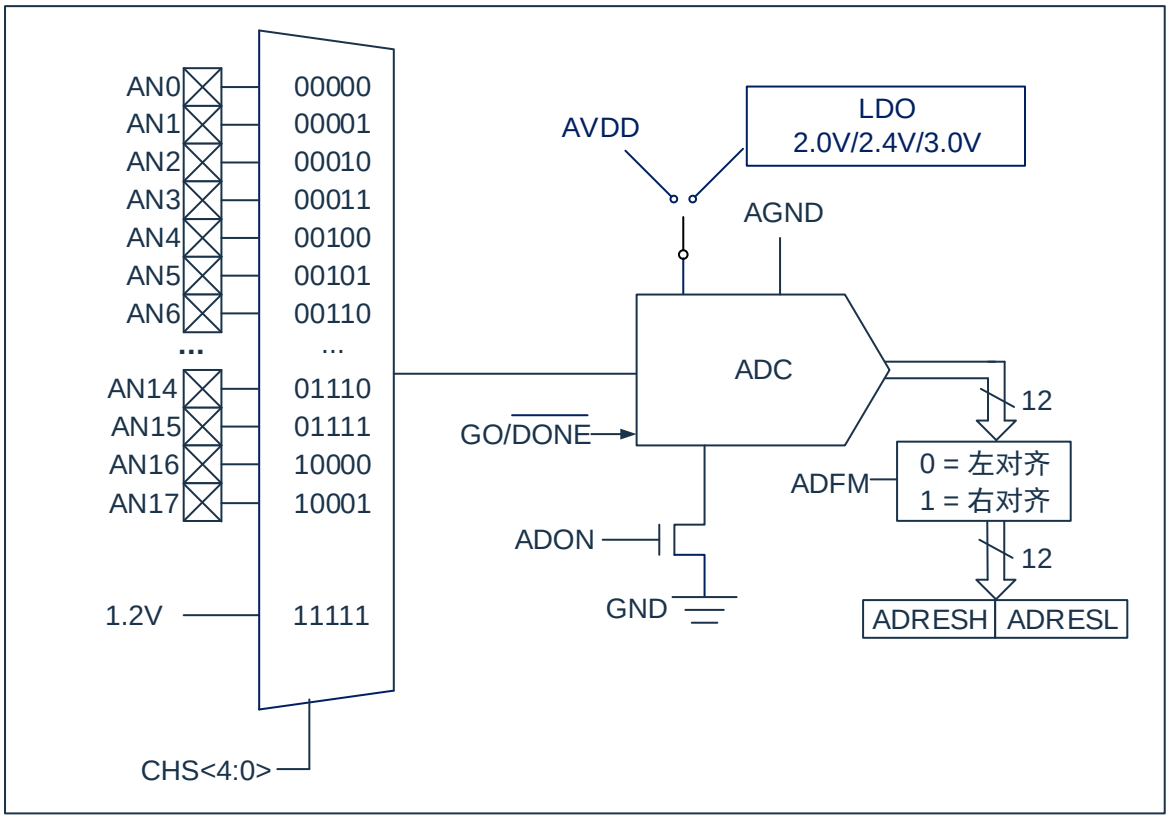


图 14-1：ADC 框图

14.2 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；
- ADC 转换时钟源；
- 中断控制；
- 结果的存储格式。

14.2.1 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。

14.2.2 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。

14.2.3 ADC 内部基准电压

芯片内置基准电压，需要检测该基准电压时，需把设置 CHS<4:0>位为 11111。

14.2.4 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 2.0V/2.4/3.0V。

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。

14.2.5 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：

- $F_{HSI}/16$
- $F_{HSI}/32$
- $F_{HSI}/64$
- $F_{HSI}/128$

完成一位转换的时间定义为 TAD。一个完整的 12 位转换需要 16 个 TAD 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。

不同参考电压和不同 VDD 时，需要参考以下表格设置合理的分频。

参考电压（V）	工作电压（V）	最快分频设置	转换时间(us)
		F_{HSI}=16M	
VDD	4.0~5.5	$F_{HSI}/16$	16
VDD	2.7~4.0	$F_{HSI}/32$	32
3.0	4.0~5.5	$F_{HSI}/32$	32
3.0	3.3~4.0	$F_{HSI}/64$	64
2.4	4.0~5.5	$F_{HSI}/32$	32
2.4	2.7~4.0	$F_{HSI}/64$	64
2.0	4.0~5.5	$F_{HSI}/64$	64
2.0	2.7~4.0	$F_{HSI}/128$	128

14.2.6 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。

14.2.7 结果格式化

12 位 A/D 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。

14.3 ADC 工作原理

14.3.1 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/$\overline{\text{DONE}}$位置 1 开始模数转换。

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

14.3.2 完成转换

当转换完成时，ADC 模块将：

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

14.3.3 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/$\overline{\text{DONE}}$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 A/D 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。因此，复位会关闭ADC模块并且终止任何待处理的转换。

14.3.4 ADC 在休眠模式下的工作原理

ADC 模块不能在休眠模式下工作。

14.3.5 A/D 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：
   - 将引脚配置为输入引脚（见 TRIS 寄存器）。
2. 配置 ADC 模块：
   - 选择 ADC 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/$\overline{\text{DONE}}$置 1 启动转换。
6. 由如下方法之一等待 ADC 转换结束：
   - 查询 GO/$\overline{\text{DONE}}$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）。

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
```

14.4 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 和 ADCON1，数据寄存器 ADRESH 和 ADRESL。

AD 控制寄存器 ADCON0(95H)

95H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	ADON
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
复位值	0	0	0	0	0	0	0	0

Bit7~Bit6 ADCS<1:0>: A/D转换时钟选择位
00= F_{HSI}/16
01= F_{HSI}/32
10= F_{HSI}/64
11= F_{HSI}/128

Bit5~Bit2 CHS<3:0>: 模拟通道选择位低四位与CHS4组成五位通道选择
CHS<4:0>:
00000= AN0
00001= AN1
00010= AN2
00011= AN3
00100= AN4
00101= AN5
…
01111= AN15
10000= AN16
10001= AN17
11111= 1.2V（固定参考电压）
其他= 保留

Bit1 GO/$\overline{DONE}$: A/D转换状态位
1= A/D转换正在进行，将该位置1启动A/D转换。当A/D转换完成以后，该位由硬件自动清零
0= A/D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 数据寄存器高位 ADCON1(96H)

96H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADCON1	ADFM	CHS4	---	---	---	LDO_EN	LDO_SEL1	LDO_SEL0
R/W	R/W	R/W	---	---	---	R/W	R/W	R/W
复位值	0	0	---	---	---	0	0	0

Bit7 ADFM: A/D转换结果格式选择位
1= 右对齐
0= 左对齐
Bit6 CHS4: 通道选择位
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
11= 3.0V
10= 2.4V
0x= 2.0V

AD 数据寄存器高位 ADRESH(99H)，ADFM=0

99H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRESH	ADRES11	ADRES10	ADRES9	ADRES8	ADRES7	ADRES6	ADRES5	ADRES4
R/W	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(98H)，ADFM=0

98H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRESL	ADRES3	ADRES2	ADRES1	ADRES0	----	----	----	----
R/W	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(99H)，ADFM=1

99H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRESH	----	----	----	----	----	----	ADRES11	ADRES10
R/W	----	----	----	----	----	----	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(98H)，ADFM=1

98H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRESL	ADRES9	ADRES8	ADRES7	ADRES6	ADRES5	ADRES4	ADRES3	ADRES2
R/W	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2-9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 2 位至第 9 位。

15. 程序 EEPROM 和程序存储器控制

15.1 概述

该系列中器件具有 4K 字的程序存储器，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 128 字的程序 EEPROM，地址范围为 00h 到 7Fh，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 和 EEDATH 寄存器形成一个双字节字用于存放 16 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 11 位程序存储器单元地址。

程序存储器允许以字为单位读取。程序 EEPROM 允许字读写。字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。

注：
1) 程序存储器指 ROM 空间，即指令代码存储的空间，只可读；
   程序 EEPROM 是可存储用户数据的空间，可读写。
2) 程序 EEPROM 正常写电压范围为 2.5V~5.5V，写电流为 5mA@VDD=5V.

15.2 相关寄存器

15.2.1 EEADR 和 EEADRH 寄存器

EEADR 和 EEADRH 寄存器能寻址最大 128 字的程序 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。

15.2.2 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。

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(8FH)

8FH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0
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　　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(91H)

91H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0
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　　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(90H)

90H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEDATH	EEDATH7	EEDATH6	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0
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　　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(92H)

92H	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEADRH	---	---	---	---	EEADRH3	EEADRH2	EEADRH1	EEADRH0
R/W	---	---	---	---	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(8DH)

8DH	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EECON1	EEPGD	---	---	---	WRERR	WREN	WR	RD
R/W	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=　写操作出错（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，但不能清零）
　　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

15.3 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 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 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

```
EEPDATA_READ:
        LD      A,RADDR         ;将要读取的地址放入 EEADR 寄存器
        LD      EEADR,A
        CLRB    EECON1,EEPGD    ;访问数据存储器
        SETB    EECON1,RD       ;启动读操作
        NOP
        NOP
        LD      A,EEDAT         ;读取数据到 ACC
        LD      RDATAL,A
        LD      A,EEDATH
        LD      RDATAH,A
EEPDATA_READ_BACK:
        RET
```

15.4 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 和 EEDATH 寄存器。然后用户必须按特定顺序开始写入每个字。

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，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。

注：在写程序 EEPROM 期间，CPU 会停止工作，停止时间为 T_{EEPROM}。

例：写程序 EEPROM

```
EEPDATA_WRITE:
        LD      A,WADDR         ;将要写入的地址放入 EEADR 寄存器
        LD      EEADR,A
        LD      A,WDATAL        ;将要写入的数据低 8 位给 EEDAT 寄存器
        LD      EEDAT,A
        LD      A,WDATAH        ;将要写入的数据高 8 位给 EEDATH 寄存器
        LD      EEDATH,A
        CLR     EECON1
        CLRB    EECON1,EEPGD    ;访问数据存储器
        SETB    EECON1,WREN     ;允许写周期
        CLRB    F_GIE_ON        ;保存中断开启状态
        SZB     INTCON,GIE
        SETB    F_GIE_ON
        CLRB    INTCON,GIE      ;关闭中断
        SZB     INTCON,GIE      ;确保中断已关闭
        JP      $-2

        LDIA    055H
        LD      EECON2,A
        LDIA    0AAH
        LD      EECON2,A
        SETB    EECON1,WR       ;启动写操作
        NOP
        NOP
        CLRWDT
        CLRB    EECON1,WREN     ;写结束，关闭写使能位

        SZB     F_GIE_ON        ;恢复中断开启状态
        SETB    INTCON,GIE

        SNZB    EECON1,WRERR    ;判断 EEPROM 写操作是否出错
        JP      EEPDATA_WRITE_BACK
```

```
        SZDECR      WERR_C              ;计数超时则退出，用户可自定义
        JP          EEPDATA_WRITE       ;EEPROM 写操作出错则重写
EEPDATA_WRITE_BACK:
        RET
```

15.5 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,RADDRL        ;将要读取的地址放入 EEADR 寄存器
        LD      EEADR,A
        LD      A,RADDRH        ;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      RDATL,A
        LD      A,EEDATH
        LD      RDATH,A
```

15.6 写程序存储器

程序存储器是只读的，不可写。

15.7 程序 EEPROM 操作注意事项

15.7.1 关于程序 EEPROM 的烧写时间

程序 EEPROM 烧写时间是大致固定的，烧写不同的数据需要的时间约为 4.6ms，并且在烧写期间 CPU 停止工作，程序需要做好相关处理。

15.7.2 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。

15.7.3 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 16ms）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

16. 电气参数

16.1 极限参数

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

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

16.2 直流电气特性

（VDD=5V，T_A= 25℃，除非另有说明）

符号	参数	测试条件		最小	典型	最大	单位
		VDD	条件				
VDD	工作电压	-	F_{SYS}=16MHz/2T	3.0	-	5.5	V
		-	F_{SYS}=16MHz/4T	1.8	-	5.5	V
I_{DD}	工作电流	5V	F_{SYS}=16MHz, 所有模拟模块关闭	-	2.5	-	mA
		5V	F_{SYS}=8MHz, 所有模拟模块关闭	-	1	-	mA
		3V	F_{SYS}=16MHz, 所有模拟模块关闭	-	1.5	-	mA
		3V	F_{SYS}=8MHz, 所有模拟模块关闭	-	0.5	-	mA
		5V	烧写程序 EEPROM	-	6	-	mA
I_{STB}	静态电流	5V	LVR=DIS WDT=DIS	-	1.5	5	uA
		3V	LVR=DIS WDT=DIS	-	0.8	3	uA
		5V	LVR=DIS WDT=EN	-	4.8	12	uA
		3V	LVR=DIS WDT=EN	-	2.1	5.5	uA
V_{IL}	低电平输入电压	-	----	-	-	0.3VDD	V
V_{IH}	高电平输入电压	-	----	0.7VDD	-	-	V
V_{OH}	高电平输出电压	-	不带负载	0.9VDD	-	-	V
V_{OL}	低电平输出电压	-	不带负载	-	-	0.1VDD	V
V_{EEPROM}	EEPROM 模块工作电压	-	----	2.5	-	5.5	V
R_{PH}	上拉电阻阻值	5V	Vo=0.5VDD	-	32	-	KΩ
		3V	Vo=0.5VDD	-	52	-	KΩ
R_{PL}	下拉电阻阻值	5V	Vo=0.5VDD	-	34	-	KΩ
		3V	Vo=0.5VDD	-	56	-	KΩ
I_{OL}	输出口灌电流	5V	V_{OL}=0.3VDD	-	37	-	mA
		3V	V_{OL}=0.3VDD	-	17	-	mA
I_{OH}	输出口拉电流	5V	V_{OH}=0.7VDD	-	-16	-	mA
		3V	V_{OH}=0.7VDD	-	-7	-	mA
V_{BG}	内部基准电压 1.2V	VDD=2.5~5.5V T_A=25℃		-1.5%	1.2	+1.5%	V
		VDD=2.0~5.5V T_A=25℃		-2.5%	1.2	+2.5%	V
		VDD=2.5~5.5V T_A=-40~85℃		-2.0%	1.2	+2.0%	V
		VDD=2.0~5.5V T_A=-40~85℃		-3.0%	1.2	+3.0%	V

16.3 比较器特性

（TA= 25℃，除非另有说明）

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

备注：V_R 为内部电阻分压输出值

16.4 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}=1MHz	3.0		5.5	V
		V_{ADREF}= VDD, F_{ADCCLK}=500kHz	2.7		5.5	V
		V_{ADREF}=2.0V, F_{ADCCLK}=250kHz	2.7		5.5	V
		V_{ADREF}=2.4V, F_{ADCCLK}=250kHz	2.7		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		±4		LSB
INL1	积分非线性误差	V_{ADC}=5V, V_{ADREF} = VDD, F_{ADCCLK}=1MHz		±8		LSB
DNL2	微分非线性误差	V_{ADC}=5V, V_{ADREF} = 3.0V, F_{ADCCLK}=500kHz, V_{AIN}<1V		±4		LSB
INL2	积分非线性误差	V_{ADC}=5V, V_{ADREF} = 3.0V, F_{ADCCLK}=500kHz, V_{AIN}<1V		±16		LSB
DNL3	微分非线性误差	V_{ADC}=5V, V_{ADREF} = 2.4V, F_{ADCCLK}=250kHz, V_{AIN}<0.8V		±4		LSB
INL3	积分非线性误差	V_{ADC}=5V, V_{ADREF} = 2.4V, F_{ADCCLK}=250kHz, V_{AIN}<0.8V		±16		LSB
DNL4	微分非线性误差	V_{ADC}=5V, V_{ADREF} = 2.0V, F_{ADCCLK}=250kHz, V_{AIN}<0.7V		±4		LSB
INL4	积分非线性误差	V_{ADC}=5V, V_{ADREF} = 2.0V, F_{ADCCLK}=250kHz, V_{AIN}<0.7V		±16		LSB
T_{ADC}	ADC 转换时间			16		T_{ADCCLK}

备注：低温规格值由设计保证，量产不测低温条件。

16.5 上电复位特性

（T_A= 25℃，除非另有说明）

符号	参数	测试条件	最小值	典型值	最大值	单位
t_{VDD}	VDD 上升速率	-	0.05	-	-	V/ms
V_{LVR1}	LVR 设定电压=1.8V	VDD=1.6~5.5V	1.7	1.8	1.9	V
V_{LVR2}	LVR 设定电压=2.0V	VDD=1.8~5.5V	1.9	2.0	2.1	V
V_{LVR3}	LVR 设定电压=2.5V	VDD=2.3~5.5V	2.4	2.5	2.6	V
V_{LVR4}	LVR 设定电压=3.0V	VDD=2.8~5.5V	2.9	3.0	3.1	V

16.6 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	典型	最大	单位
		VDD	条件				
F_{WDT}	WDT 时钟源	VDD=2.5~5.5V	T_A=25℃	-20%	32	+20%	KHz
		VDD=1.8~5.5V	T_A=25℃	-30%	32	+30%	KHz
		VDD=2.5~5.5V	T_A=－40~85℃	-30%	32	+30%	KHz
		VDD=1.8~5.5V	T_A=－40~85℃	-50%	32	+50%	KHz
T_{EEPROM}	EEPROM 编程时间	5V	F_{HSI}=16MHz	-	4.6	-	ms
		3V	F_{HSI}=16MHz	-	4.6	-	ms
F_{INTRC}	内振频率 16MHz	VDD=4.0~5.5V	TA=25℃	-2.0%	16	+2.0%	MHz
		VDD=2.5~5.5V	TA=25℃	-2.5%	16	+2.5%	MHz
		VDD=1.8~5.5V	TA=25℃	-3.0%	16	+3.0%	MHz
		VDD=4.0~5.5V	TA=－40~85℃	-3.0%	16	+3.0%	MHz
		VDD=2.5~5.5V	TA=－40~85℃	-4.0%	16	+4.0%	MHz
		VDD=1.8~5.5V	TA=－40~85℃	-5.0%	16	+5.0%	MHz

16.7 LSE 特性

（TA= 25℃，除非另有说明）

符号	参数	测试条件	最小值	典型值	最大值	单位
VDD	工作电压范围	-	1.8	-	5.5	V
F_{LSE}	LSE振荡频率	-	-	32.768	-	KHz
C1	OSCIN引脚匹配电容	-	-	22	-	pF
C2	OSCOUT引脚匹配电容	-	-	22	-	pF
I_{LSE}	LSE工作电流	VDD=5V C_1=22pF C_2=22pF	-	20	-	uA
		VDD=3V C_1=22pF C_2=22pF	-	8	-	uA
T_{LSE}	LSE稳定时间	VDD=5V C_1=22pF C_2=22pF	-	260	700	ms
		VDD=3V C_1=22pF C_2=22pF		300	1000	ms

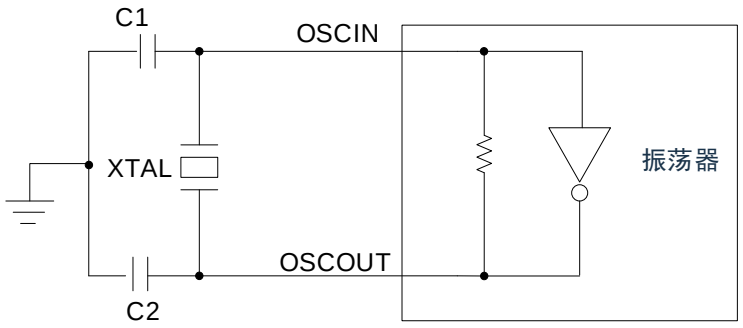


图 16-1：典型应用电路

17. 指令

17.1 指令一览表

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

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

17.2 指令说明

ADDA **[R]**

操作: 将 R 加 ACC，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

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

ADDR **[R]**

操作: 将 R 加 ACC，结果放入 R

周期: 1

影响标志位: C，DC，Z，OV

举例:

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

ADDCA **[R]**

操作: 将 R 加 ACC 加 C 位，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

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

ADDCR **[R]**

操作: 将 R 加 ACC 加 C 位，结果放入 R

周期: 1

影响标志位: C，DC，Z，OV

举例:

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

ADDIA **i**

操作: 将立即数 i 加 ACC，结果放入 ACC

周期: 1

影响标志位: C，DC，Z，OV

举例:

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

ANDA **[R]**

操作: 寄存器 R 跟 ACC 进行逻辑与运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

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

ANDR **[R]**

操作: 寄存器 R 跟 ACC 进行逻辑与运算，结果放入 R

周期: 1

影响标志位: Z

举例:

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

ANDIA **i**

操作: 将立即数 i 与 ACC 进行逻辑与运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

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

CALL **add**

操作: 调用子程序

周期: 2

影响标志位: 无

举例:

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

CLRA

操作: ACC 清零

周期: 1

影响标志位: Z

举例:

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

CLR [R]

操作: 寄存器 R 清零

周期: 1

影响标志位: Z

举例:

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

CLRB [R],b

操作: 寄存器 R 的第 b 位清零

周期: 1

影响标志位: 无

举例:

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

CLRWDT

操作: 清零看门狗计数器

周期: 1

影响标志位: TO，PD

举例:

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

COMA [R]

操作: 寄存器 R 取反，结果放入 ACC

周期: 1

影响标志位: Z

举例:

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

COMR **[R]**

操作: 寄存器 R 取反，结果放入 R

周期: 1

影响标志位: Z

举例:

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

DECA **[R]**

操作: 寄存器 R 自减 1，结果放入 ACC

周期: 1

影响标志位: Z

举例:

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

DECR **[R]**

操作: 寄存器 R 自减 1，结果放入 R

周期: 1

影响标志位: Z

举例:

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

HSUBA **[R]**

操作: ACC 减 R，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

HSUBR **[R]**

操作: ACC 减 R，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

HSUBCA **[R]**

操作: ACC 减 R 减 $\overline{C}$，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

HSUBCR **[R]**

操作: ACC 减 R 减 $\overline{C}$，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

INCA **[R]**

操作: 寄存器 R 自加 1，结果放入 ACC

周期: 1

影响标志位: Z

举例:

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

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

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

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

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

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

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

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

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

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

```
LDIA    0AH     ;ACC 赋值 0AH
```

NOP

操作: 空指令
周期: 1
影响标志位: 无
举例:

```
NOP
NOP
```

ORIA i

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

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

ORA [R]

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

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

ORR [R]

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

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

RET

操作: 从子程序返回

周期: 2

影响标志位: 无

举例:

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

RET i

操作: 从子程序带参数返回，参数放入 ACC

周期: 2

影响标志位: 无

举例:

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

RETI

操作: 中断返回

周期: 2

影响标志位: 无

举例:

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

RLCA [R]

操作: 寄存器 R 带 C 循环左移一位，结果放入 ACC

周期: 1

影响标志位: C

举例:

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

RLCR **[R]**

操作: 寄存器 R 带 C 循环左移一位，结果放入 R

周期: 1

影响标志位: C

举例:

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

RLA **[R]**

操作: 寄存器 R 不带 C 循环左移一位，结果放入 ACC

周期: 1

影响标志位: 无

举例:

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

RLR **[R]**

操作: 寄存器 R 不带 C 循环左移一位，结果放入 R

周期: 1

影响标志位: 无

举例:

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

RRCA **[R]**

操作: 寄存器 R 带 C 循环右移一位，结果放入 ACC

周期: 1

影响标志位: C

举例:

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

RRCR **[R]**

操作: 寄存器 R 带 C 循环右移一位，结果放入 R

周期: 1

影响标志位: C

举例:

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

RRA **[R]**

操作: 寄存器 R 不带 C 循环右移一位，结果放入 ACC

周期: 1

影响标志位: 无

举例:

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

RRR **[R]**

操作: 寄存器 R 不带 C 循环右移一位，结果放入 R

周期: 1

影响标志位: 无

举例:

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

SET **[R]**

操作: 寄存器 R 所有位置 1

周期: 1

影响标志位: 无

举例:

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

SETB **[R],b**

操作: 寄存器 R 的第 b 位置 1

周期: 1

影响标志位: 无

举例:

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

STOP

操作: 进入休眠状态

周期: 1

影响标志位: TO，PD

举例:

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

SUBIA i

操作: 立即数 i 减 ACC，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

SUBA [R]

操作: 寄存器 R 减 ACC，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

SUBR [R]

操作: 寄存器 R 减 ACC，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

SUBCA **[R]**

操作: 寄存器 R 减 ACC 减 C，结果放入 ACC

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

SUBCR **[R]**

操作: 寄存器 R 减 ACC 减 C，结果放入 R

周期: 1

影响标志位: C,DC,Z,OV

举例:

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

SWAPA **[R]**

操作: 寄存器 R 高低半字节交换，结果放入 ACC

周期: 1

影响标志位: 无

举例:

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

SWAPR **[R]**

操作: 寄存器 R 高低半字节交换，结果放入 R

周期: 1

影响标志位: 无

举例:

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

SZB **[R],b**

操作: 判断寄存器 R 的第 b 位，为 0 间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

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

SNZB **[R],b**

操作: 判断寄存器 R 的第 b 位，为 1 间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

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

SZA **[R]**

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

周期: 1 or 2

影响标志位: 无

举例:

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

SZR **[R]**

操作: 将寄存器 R 的值赋给 R，若 R 为 0 则间跳，否则顺序执行

周期: 1 or 2

影响标志位: 无

举例:

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

SZINCA **[R]**

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

周期: 1 or 2

影响标志位: 无

举例:

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

SZINCR **[R]**

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

周期: 1 or 2

影响标志位: 无

举例:

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

SZDECA **[R]**

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

周期: 1 or 2

影响标志位: 无

举例:

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

SZDECR **[R]**

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

周期: 1 or 2

影响标志位: 无

举例:

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

TESTZ **[R]**

操作: 将 R 的值赋给 R,用以影响 Z 标志位

周期: 1

影响标志位: Z

举例:

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

XORIA **i**

操作: 立即数与 ACC 进行逻辑异或运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

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

XORA **[R]**

操作: 寄存器 R 与 ACC 进行逻辑异或运算，结果放入 ACC

周期: 1

影响标志位: Z

举例:

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

XORR **[R]**

操作: 寄存器 R 与 ACC 进行逻辑异或运算，结果放入 R

周期: 1

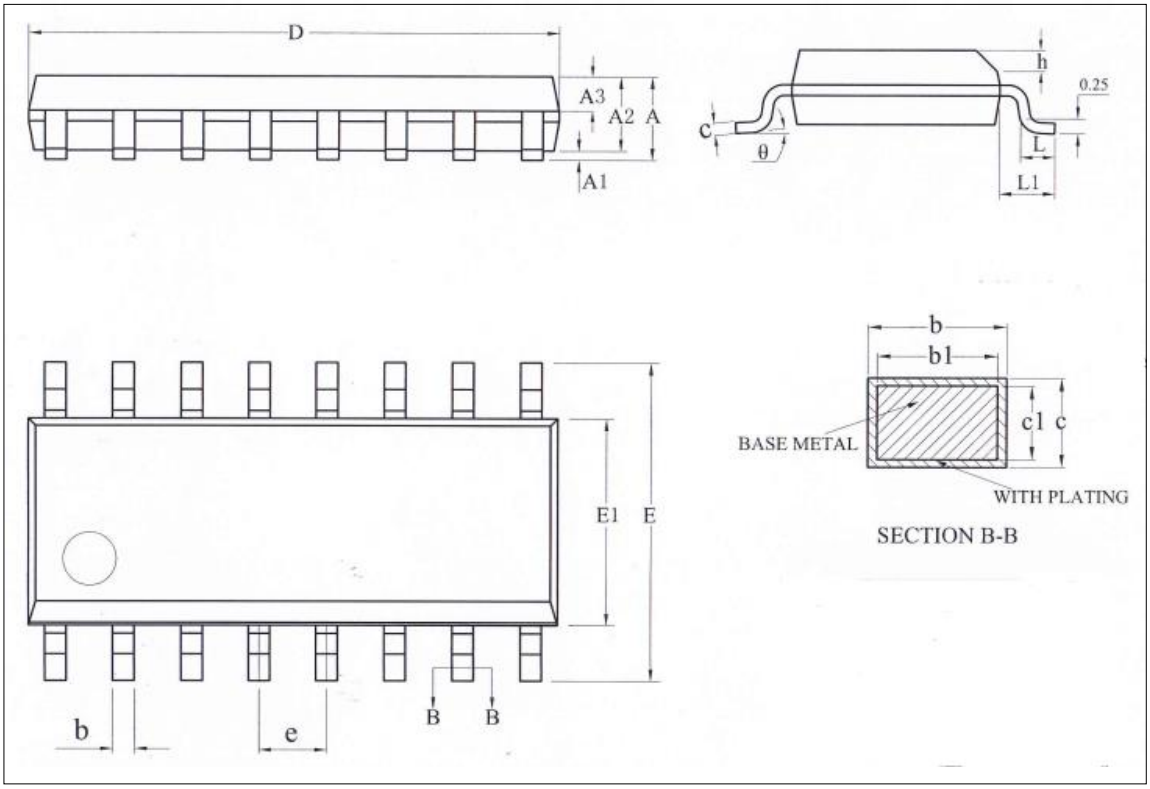
影响标志位: Z

举例:

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

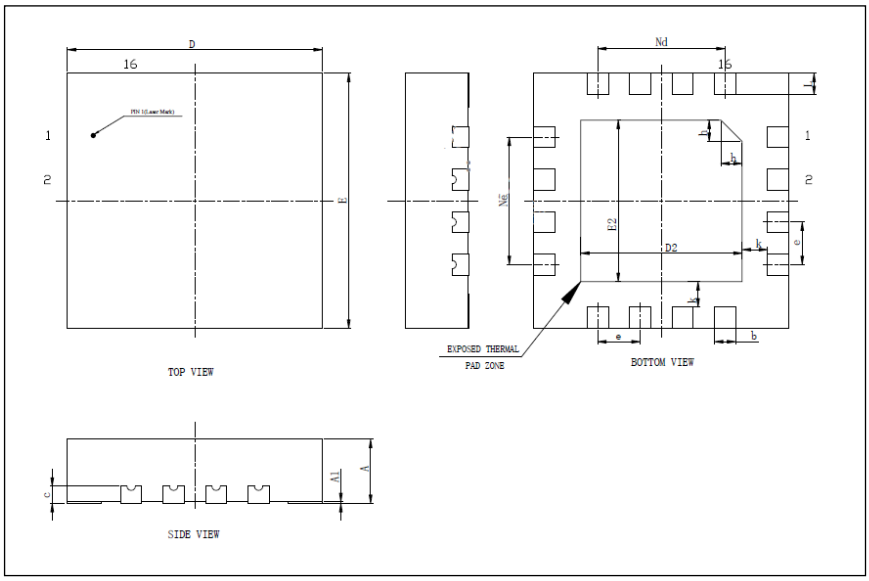
18. 封装

18.1 SOP16



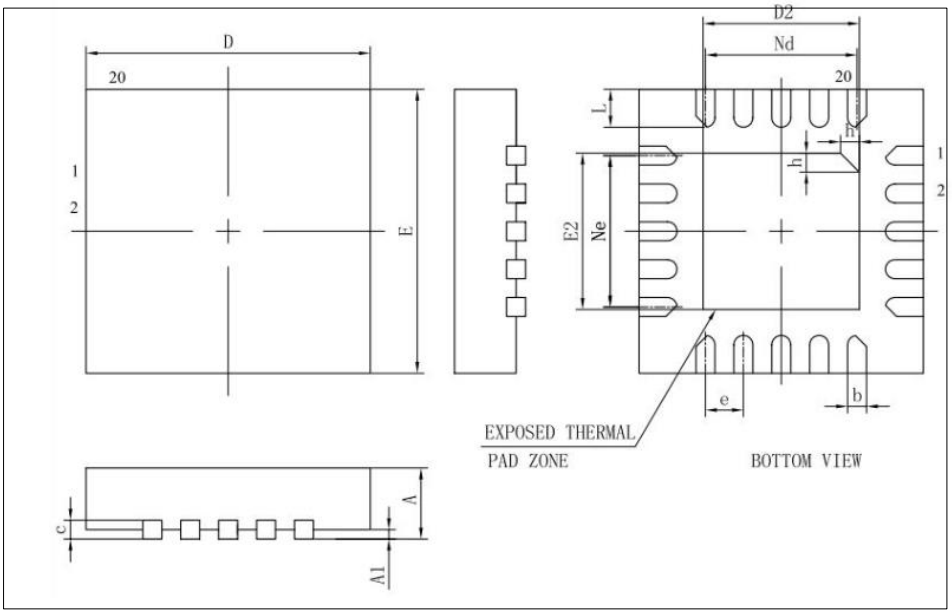
Symbol	Millimeter		
	Min	Nom	mAx
A	-	-	1.75
A1	0.10	-	0.225
A2	1.30	1.40	1.50
A3	0.60	0.65	0.70
b	0.39	-	0.47
b1	0.38	0.41	0.44
c	0.20	-	0.24
c1	0.19	0.20	0.21
D	9.80	9.90	10.00
E	5.80	6.00	6.20
E1	3.80	3.90	4.00
e	1.27BSC		
h	0.25	-	0.50
L	0.50	-	0.80
L1	1.05REF		
θ	0	-	8°

18.2 QFN16(3*3*0.75-0.50)



Symbol	Millimeter		
	Min	Nom	Max
A	0.70	0.75	0.80
A1	0	0.02	0.05
b	0.20	0.25	0.30
c		0.203REF	
D	2.90	3.00	3.10
D2	1.80	1.90	2.00
e		0.50BSC	
Nd		1.50BSC	
Ne		1.50BSC	
E	2.90	3.00	3.10
E2	1.80	1.90	2.00
L	0.20	0.25	0.30
h	0.20	0.25	0.30

18.3 QFN20



Symbol	Millimeter		
	Min	Nom	Max
A	0.70	0.75	0.80
A1	-	0.02	0.05
b	0.15	0.20	0.25
c	0.18	0.20	0.25
D	2.90	3.00	3.10
D2	1.55	1.65	1.75
e	0.40BSC		
Ne	1.60BSC		
Nd	1.60BSC		
E	2.90	3.00	3.10
E2	1.55	1.65	1.75
L	0.35	0.40	0.45
h	0.20	0.25	0.30

19. 版本修订说明

版本号	时间	修订内容
V0.0.1	2023年8月	初始版本
V0.5.0	2023年11月	1) 修改电气参数 2) 更正特殊功能寄存器列表 3) 删除SC8F073AD720SS型号及相关信息
V1.0.0	2024年1月	1) 修改“16.4 ADC电气特性”ADC精度 2) 删除“ADC内部LDO参考电压特性”章节内容 3) 修改“16.6 交流电气特性”内振精度
V1.0.1	2024年3月	删除SC8F073AD720SA型号及相关信息