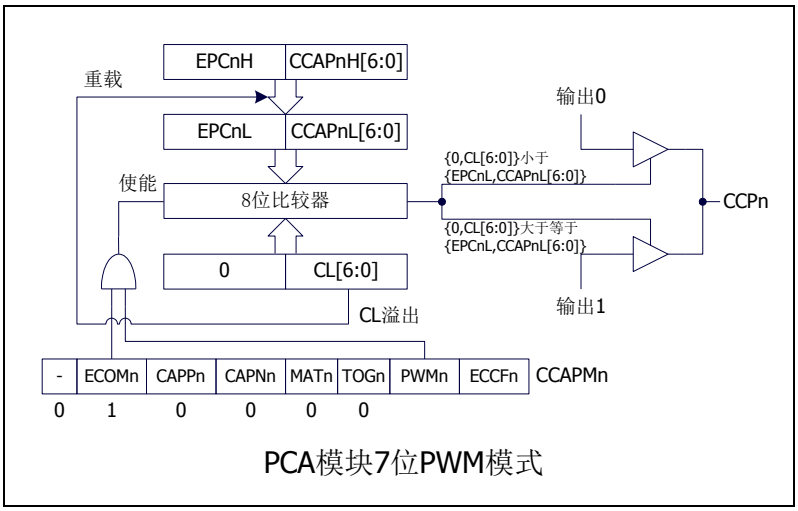


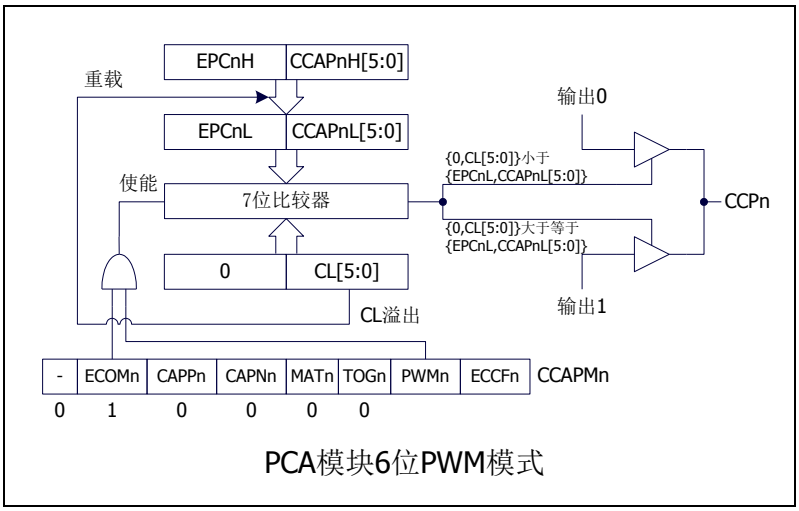
18.3.4.3 6 位 PWM 模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 10 时，PCA 模块 n 工作于 6 位 PWM 模式，此时将{0,CL[5:0]}与捕获寄存器{EPCnL,CCAPnL[5:0]}进行比较。当 PCA 模块工作于 6 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[5:0]}进行设置。当{0,CL[5:0]}的值小于{EPCnL,CCAPnL[5:0]}时，输出为低电平；当{0,CL[5:0]}的值等于或大于{EPCnL,CCAPnL[5:0]}时，输出为高电平。当 CL[5:0]的值由 3F 变为 00 溢出时，{EPCnH,CCAPnH[5:0]}的内容重新装载到{EPCnL,CCAPnL[5:0]}中。这样就可实现无干扰地更新 PWM。

$$8\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{64}$$

当EPCnH=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 6 位 PWM 模式的结构图如下图所示：



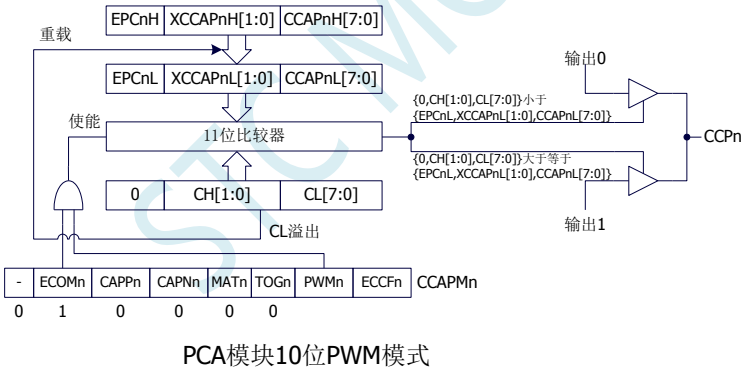
18.3.4.4 10 位 PWM 模式

PCA_PWMn 寄存器中的 EBSn[1:0]设置为 11 时，PCA 模块 n 工作于 10 位 PWM 模式，此时将{CH[1:0],CL[7:0]}与捕获寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行比较。当 PCA 模块工作于 10 位 PWM 模式时，由于所有模块共用一个 PCA 计数器，所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}进行设置。当{CH[1:0],CL[7:0]}的值小于{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}时，输出为低电平；当{CH[1:0],CL[7:0]}的值等于或大于{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}时，输出为高电平。当{CH[1:0],CL[7:0]}的值由 3FF 变为 00 溢出时，{EPCnH,XCCAPnH[1:0],CCAPnH[7:0]}的内容重新装载到{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}中。这样就可实现无干扰地更新 PWM。

$$10\text{位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{1024}$$

当EPCnH=0, XCCAP0H=0及CCAPnH=00H时，PWM固定输出高
当EPCnH=1, XCCAP0H=3及CCAPnH=FFH时，PWM固定输出低

PCA 模块工作于 10 位 PWM 模式的结构图如下图所示：

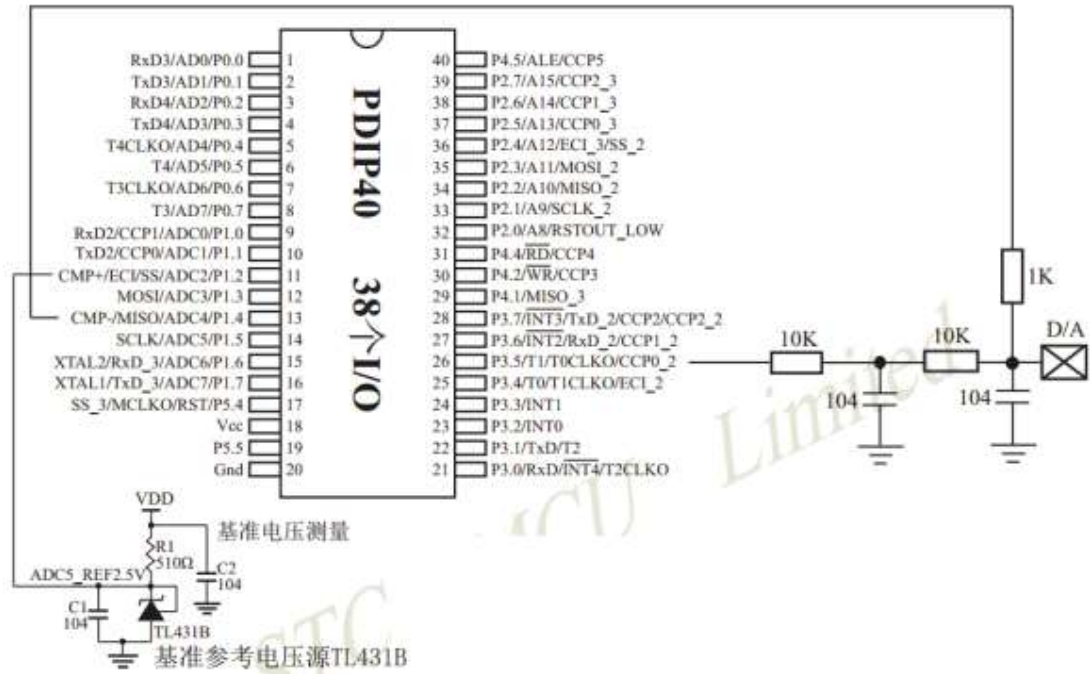


PCA模块10位PWM模式

18.3.4.5 如何控制 PWM 固定输出高电平/低电平

当 PCA_PWMn &= 0xC0，CCAPnH = 0x00 时，PWM 固定输出高电平
当 PCA_PWMn |= 0x3F，CCAPnH = 0xFF 时，PWM 固定输出低电平

18.4 利用 CCP/PCA/PWM 模块实现 8~16 位 DAC 的参考线路图



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

利用CCP/PCA模块的高速脉冲输出功能实现9~16位PWM来实现9~16位DAC，或用本身的硬件8位PWM来实现8位DAC，单片机本身也有10位ADC。

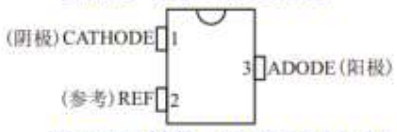
提示：

（1）PWM频率越高，输出波形越平滑。

（2）如果工作电压为5V，需输出1V电压，则设置高电平为1/5，低电平为4/5，则PWM输出电压就为1V。

（3）如果要输出高精准电压，建议用A/D检测输出的电压值，然后根据A/D检测的电压值逐步调整到所需要的电压。

基准参考电压源TL431B



SOT23-3封装，RMB￥0.15~0.3

基准参考电压源TL431B的符号

如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

18.5 范例程序

18.5.1 PCA 输出 PWM（6/7/8/10 位）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
//——6 位 PWM——
    CCAPM0 = 0x42;                  //PCA 模块 0 为 PWM 工作模式
    PCA_PWM0 = 0x80;                //PCA 模块 0 输出 6 位 PWM
    CCAP0L = 0x20;                  //PWM 占空比为 50%[(40H-20H)/40H]
    CCAP0H = 0x20;
//——7 位 PWM——
    CCAPM1 = 0x42;                  //PCA 模块 1 为 PWM 工作模式
    PCA_PWM1 = 0x40;                //PCA 模块 1 输出 7 位 PWM
    CCAP1L = 0x20;                  //PWM 占空比为 75%[(80H-20H)/80H]
    CCAP1H = 0x20;
//——8 位 PWM——
//  CCAPM2 = 0x42;                  //PCA 模块 2 为 PWM 工作模式
//  PCA_PWM2 = 0x00;                //PCA 模块 2 输出 8 位 PWM
//  CCAP2L = 0x20;                  //PWM 占空比为 87.5%[(100H-20H)/100H]
//  CCAP2H = 0x20;
//——10 位 PWM——
    CCAPM2 = 0x42;                  //PCA 模块 2 为 PWM 工作模式
    PCA_PWM2 = 0xc0;                //PCA 模块 2 输出 10 位 PWM
    CCAP2L = 0x20;                  //PWM 占空比为 96.875%[(400H-20H)/400H]
    CCAP2H = 0x20;
    CR = 1;                         //启动 PCA 计时器

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H
```

```
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#08H           ;PCA 时钟为系统时钟
            MOV     CL,#00H
            MOV     CH,#0H
;——6 位 PWM——
            MOV     CCAPM0,#42H         ;PCA 模块 0 为 PWM 工作模式
            MOV     PCA_PWM0,#80H       ;PCA 模块 0 输出 6 位 PWM
            MOV     CCAP0L,#20H         ;PWM 占空比为 50%[(40H-20H)/40H]
            MOV     CCAP0H,#20H
;——7 位 PWM——
            MOV     CCAPM1,#42H         ;PCA 模块 1 为 PWM 工作模式
            MOV     PCA_PWM1,#40H       ;PCA 模块 1 输出 7 位 PWM
            MOV     CCAP1L,#20H         ;PWM 占空比为 75%[(80H-20H)/80H]
            MOV     CCAP1H,#20H
;——8 位 PWM——
;           MOV     CCAPM2,#42H         ;PCA 模块 2 为 PWM 工作模式
;           MOV     PCA_PWM2,#00H       ;PCA 模块 2 输出 8 位 PWM
;           MOV     CCAP2L,#20H         ;PWM 占空比为 87.5%[(100H-20H)/100H]
;           MOV     CCAP2H,#20H
;——10 位 PWM——
            MOV     CCAPM2,#42H         ;PCA 模块 2 为 PWM 工作模式
            MOV     PCA_PWM2,#0C0H      ;PCA 模块 2 输出 10 位 PWM
            MOV     CCAP2L,#20H         ;PWM 占空比为 96.875%[(400H-20H)/400H]
            MOV     CCAP2H,#20H
            SETB    CR                  ;启动 PCA 计时器
```

```
        JMP     $

        END
```

18.5.2 PCA 捕获测量脉冲宽度

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

unsigned char       cnt;                //存储 PCA 计时溢出次数
unsigned long       count0;             //记录上一次的捕获值
unsigned long       count1;             //记录本次的捕获值
unsigned long       length;             //存储信号的时间长度

void PCA_Isr() interrupt 7
{
    if (CF)
```

```
    {
        CF = 0;
        cnt++;                                  //PCA 计时溢出次数+1
    }
    if (CCF0)
    {
        CCF0 = 0;
        count0 = count1;                        //备份上一次的捕获值
        ((unsigned char *)&count1)[3] = CCAP0L;
        ((unsigned char *)&count1)[2] = CCAP0H;
        ((unsigned char *)&count1)[1] = cnt;
        ((unsigned char *)&count1)[0] = 0;
        length = count1 - count0;               //length 保存的即为捕获的脉冲宽度
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    cnt = 0;                                    //用户变量初始化
    count0 = 0;
    count1 = 0;
    length = 0;
    CCON = 0x00;
    CMOD = 0x09;                                //PCA 时钟为系统时钟,使能 PCA 计时中断
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                              //PCA 模块 0 为 16 位捕获模式（下降沿捕获）
//  CCAPM0 = 0x21;                              //PCA 模块 0 为 16 位捕获模式（上升沿捕获）
//  CCAPM0 = 0x31;                              //PCA 模块 0 为 16 位捕获模式（边沿捕获）
    CCAP0L = 0x00;
    CCAP0H = 0x00;
    CR = 1;                                     //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
```

```
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

CNT         DATA    20H
COUNT0      DATA    21H         ;3 bytes
COUNT1      DATA    24H         ;3 bytes
LENGTH      DATA    27H         ;3 bytes, (COUNT1-COUNT0)

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            JNB     CF,CHECKCCF0
            CLR     CF          ;清中断标志
            INC     CNT         ; PCA 计时溢出次数+1
CHECKCCF0:
            JNB     CCF0,ISREXIT
            CLR     CCF0
            MOV     COUNT0,COUNT1       ;备份上一次的捕获值
            MOV     COUNT0+1,COUNT1+1
            MOV     COUNT0+2,COUNT1+2
            MOV     COUNT1,CNT          ;保存本次的捕获值
            MOV     COUNT1+1,CCAP0H
            MOV     COUNT1+2,CCAP0L
            CLR     C                   ;计算两次的捕获差值
            MOV     A,COUNT1+2
            SUBB    A,COUNT0+2
```

```
        MOV     LENGTH+2,A
        MOV     A,COUNT1+1
        SUBB    A,COUNT0+1
        MOV     LENGTH+1,A
        MOV     A,COUNT1
        SUBB    A,COUNT0
        MOV     LENGTH,A            ;LENGTH 保存的即为捕获的脉冲宽度
ISREXIT:
        POP     PSW
        POP     ACC
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        CLR     A
        MOV     CNT,A               ;用户变量初始化
        MOV     COUNT0,A
        MOV     COUNT0+1,A
        MOV     COUNT0+2,A
        MOV     COUNT1,A
        MOV     COUNT1+1,A
        MOV     COUNT1+2,A
        MOV     LENGTH,A
        MOV     LENGTH+1,A
        MOV     LENGTH+2,A

        MOV     CCON,#00H
        MOV     CMOD,#09H           ;PCA 时钟为系统时钟,使能 PCA 计时中断
        MOV     CL,#00H
        MOV     CH,#0H
        MOV     CCAPM0,#11H         ;PCA 模块 0 为 16 位捕获模式（下降沿捕获）
;       MOV     CCAPM0,#21H         ;PCA 模块 0 为 16 位捕获模式（上升沿捕获）
;       MOV     CCAPM0,#31H         ;PCA 模块 0 为 16 位捕获模式（边沿捕获）
        MOV     CCAP0L,#00H
        MOV     CCAP0H,#00H
        SETB    CR                  ;启动 PCA 计时器
        SETB    EA

        JMP     $

        END
```

18.5.3 PCA 实现 16 位软件定时

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   T50HZ           (11059200L / 12 / 2 / 50)

sfr       CCON      =     0xd8;
sbit      CF        =     CCON^7;
sbit      CR        =     CCON^6;
sbit      CCF2      =     CCON^2;
sbit      CCF1      =     CCON^1;
sbit      CCF0      =     CCON^0;
sfr       CMOD      =     0xd9;
sfr       CL        =     0xe9;
sfr       CH        =     0xf9;
sfr       CCAPM0    =     0xda;
sfr       CCAP0L    =     0xea;
sfr       CCAP0H    =     0xfa;
sfr       PCA_PWM0  =     0xf2;
sfr       CCAPM1    =     0xdb;
sfr       CCAP1L    =     0xeb;
sfr       CCAP1H    =     0xfb;
sfr       PCA_PWM1  =     0xf3;
sfr       CCAPM2    =     0xdc;
sfr       CCAP2L    =     0xec;
sfr       CCAP2H    =     0xfc;
sfr       PCA_PWM2  =     0xf4;

sfr       P0M1      =     0x93;
sfr       P0M0      =     0x94;
sfr       P1M1      =     0x91;
sfr       P1M0      =     0x92;
sfr       P2M1      =     0x95;
sfr       P2M0      =     0x96;
sfr       P3M1      =     0xb1;
sfr       P3M0      =     0xb2;
sfr       P4M1      =     0xb3;
sfr       P4M0      =     0xb4;
sfr       P5M1      =     0xc9;
sfr       P5M0      =     0xca;

sbit      P10       =     P1^0;

unsigned int              value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;

    P10 = !P10;                                   //测试端口
```

```
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x00;                    //PCA 时钟为系统时钟/12
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;                  //PCA 模块 0 为 16 位定时器模式
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = 1;                         //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

T50HZ       EQU     2400H           ;11059200/12/2/50
```

```
P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     003BH
        LJMP    PCAISR

        ORG     0100H
PCAISR:
        PUSH    ACC
        PUSH    PSW
        CLR     CCF0
        MOV     A,CCAP0L
        ADD     A,#LOW T50HZ
        MOV     CCAP0L,A
        MOV     A,CCAP0H
        ADDC    A,#HIGH T50HZ
        MOV     CCAP0H,A
        CPL     P1.0            ;测试端口,闪烁频率为50Hz
        POP     PSW
        POP     ACC
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CCON,#00H
        MOV     CMOD,#00H       ;PCA 时钟为系统时钟/12
        MOV     CL,#00H
        MOV     CH,#0H
        MOV     CCAPM0,#49H     ;PCA 模块0 为16 位定时器模式
        MOV     CCAP0L,#LOW T50HZ
        MOV     CCAP0H,#HIGH T50HZ
        SETB    CR              ;启动PCA 计时器
        SETB    EA
```

```
        JMP     $

        END
```

18.5.4 PCA 输出高速脉冲

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   T38K4HZ       (11059200L / 2 / 38400)

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

unsigned int            value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
```

```
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x4d;                  //PCA 模块 0 为 16 位定时器模式并使能脉冲输出
    value = T38K4HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T38K4HZ;
    CR = 1;                         //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CCON        DATA    0D8H
CF          BIT     CCON.7
CR          BIT     CCON.6
CCF2        BIT     CCON.2
CCF1        BIT     CCON.1
CCF0        BIT     CCON.0
CMOD        DATA    0D9H
CL          DATA    0E9H
CH          DATA    0F9H
CCAPM0      DATA    0DAH
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
```

```
PCA_PWM2    DATA    0F4H

T38K4HZ     EQU     90H             ;11059200/2/38400

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            CLR     CCF0
            MOV     A,CCAP0L
            ADD     A,#LOW T38K4HZ
            MOV     CCAP0L,A
            MOV     A,CCAP0H
            ADDC    A,#HIGH T38K4HZ
            MOV     CCAP0H,A
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#08H       ;PCA 时钟为系统时钟
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#4DH     ;PCA 模块 0 为 16 位定时器模式并使能脉冲输出
            MOV     CCAP0L,#LOW T38K4HZ
            MOV     CCAP0H,#HIGH T38K4HZ
```

```
        SETB    CR                  ;启动 PCA 计时器
        SETB    EA

        JMP     $

        END
```

18.5.5 PCA 扩展外部中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CCON    =   0xd8;
sbit    CF      =   CCON^7;
sbit    CR      =   CCON^6;
sbit    CCF2    =   CCON^2;
sbit    CCF1    =   CCON^1;
sbit    CCF0    =   CCON^0;
sfr     CMOD    =   0xd9;
sfr     CL      =   0xe9;
sfr     CH      =   0xf9;
sfr     CCAPM0  =   0xda;
sfr     CCAP0L  =   0xea;
sfr     CCAP0H  =   0xfa;
sfr     PCA_PWM0=   0xf2;
sfr     CCAPM1  =   0xdb;
sfr     CCAP1L  =   0xeb;
sfr     CCAP1H  =   0xfb;
sfr     PCA_PWM1=   0xf3;
sfr     CCAPM2  =   0xdc;
sfr     CCAP2L  =   0xec;
sfr     CCAP2H  =   0xfc;
sfr     PCA_PWM2=   0xf4;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
```

```
    P10 = !P10;
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA 时钟为系统时钟
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                  //扩展外部端口 CCP0 为下降沿中断口
//  CCAPM0 = 0x21;                  //扩展外部端口 CCP0 为上升沿中断口
//  CCAPM0 = 0x31;                  //扩展外部端口 CCP0 为边沿中断口
    CCAP0L = 0;
    CCAP0H = 0;
    CR = 1;                         //启动 PCA 计时器
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
CCAPM0  DATA    0DAH
CCAP0L  DATA    0EAH
CCAP0H  DATA    0FAH
PCA_PWM0 DATA   0F2H
CCAPM1  DATA    0DBH
CCAP1L  DATA    0EBH
CCAP1H  DATA    0FBH
PCA_PWM1 DATA   0F3H
CCAPM2  DATA    0DCH
CCAP2L  DATA    0ECH
CCAP2H  DATA    0FCH
PCA_PWM2 DATA   0F4H
```

```
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            CLR     CCF0
            CPL     P1.0
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     CCON,#00H
            MOV     CMOD,#08H       ;PCA 时钟为系统时钟
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#11H     ;扩展外部端口 CCP0 为下降沿中断口
;           MOV     CCAPM0,#21H     ;扩展外部端口 CCP0 为上升沿中断口
;           MOV     CCAPM0,#31H     ;扩展外部端口 CCP0 为边沿中断口
            MOV     CCAP0L,#0
            MOV     CCAP0H,#0
            SETB    CR              ;启动 PCA 计时器
            SETB    EA

            JMP     $

            END
```

19 精度可达 15 位的增强型 PWM

STC8A8K64D4-64Pin/48Pin 系列单片机集成了 1 组增强型的 PWM 波形发生器，可产生各自独立的 8 路 PWM。PWM 的时钟源可以选择。PWM 波形发生器内部有一个 15 位的 PWM 计数器供 8 路 PWM 使用，用户可以设置每路 PWM 的初始电平。另外，PWM 波形发生器为每路 PWM 又设计了两个用于控制波形翻转的计数器 T1/T2，可以非常灵活的控制每路 PWM 的高低电平宽度，从而达到对 PWM 的占空比以及 PWM 的输出延迟进行控制的目的。由于 8 路 PWM 是各自独立的，且每路 PWM 的初始状态可以进行设定，所以用户可以将其中的任意两路配合起来使用，即可实现互补对称输出以及死区控制等特殊应用。（注：增强型 PWM 只有输出功能，如果需要测量脉冲宽度，请使用本系列的 PCA/CCP/PWM 功能）增强型的 PWM 波形发生器还设计了对外部异常事件（包括外部端口 P3.5 电平异常、比较器比较结果异常）进行监控的功能，可用于紧急关闭 PWM 输出。PWM 波形发生器还可与 ADC 相关联，设置 PWM 周期的任一时间点触发 ADC 转换事件。

STC 三种硬件 PWM 比较：

兼容传统 8051 的 PCA/CCP/PWM：可输出 PWM 波形、捕获外部输入信号以及输出高速脉冲。可对外输出 6 位/7 位/8 位/10 位的 PWM 波形，6 位 PWM 波形的频率为 PCA 模块时钟源频率/64；7 位 PWM 波形的频率为 PCA 模块时钟源频率/128；8 位 PWM 波形的频率为 PCA 模块时钟源频率/256；10 位 PWM 波形的频率为 PCA 模块时钟源频率/1024。捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿。

15 位增强型 PWM：只能对外输出 PWM 波形，无输入捕获功能。对外输出 PWM 的频率以及占空比均可任意设置。通过软件干预，可实现多路互补/对称/带死区的 PWM 波形。有外部异常检测功能以及实时触发 ADC 转换功能。

STC8H 系列的 16 位高级 PWM 定时器：是目前 STC 功能最强的 PWM，可对外输出任意频率以及任意占空比的 PWM 波形。无需软件干预即可输出互补/对称/带死区的 PWM 波形。能捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿，测量外部波形时，可同时测量波形的周期值和占空比值。有正交编码功能、外部异常检测功能以及实时触发 ADC 转换功能。

19.1 增强型 PWM 输出功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF14H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF1CH	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF2CH	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF34H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI
PWM5CR	FF3CH	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF44H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF4CH	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI

C0_S[1:0]：增强型 PWM 通道 0 输出脚选择位

C0_S[1:0]	PWM0
00	P2.0
01	P1.0
10	P6.0

11	保留

C1_S[1:0]：增强型 PWM 通道 1 输出脚选择位

C1_S[1:0]	PWM1
00	P2.1
01	P1.1
10	P6.1
11	保留

C2_S[1:0]：增强型 PWM 通道 2 输出脚选择位

C2_S[1:0]	PWM2
00	P2.2
01	P1.2
10	P6.2
11	保留

C3_S[1:0]：增强型 PWM 通道 3 输出脚选择位

C3_S[1:0]	PWM3
00	P2.3
01	P1.3
10	P6.3
11	保留

C4_S[1:0]：增强型 PWM 通道 4 输出脚选择位

C4_S[1:0]	PWM4
00	P2.4
01	P1.4
10	P6.4
11	保留

C5_S[1:0]：增强型 PWM 通道 5 输出脚选择位

C5_S[1:0]	PWM5
00	P2.5
01	P1.5
10	P6.5
11	保留

C6_S[1:0]：增强型 PWM 通道 6 输出脚选择位

C6_S[1:0]	PWM6
00	P2.6
01	P1.6
10	P6.6
11	保留

C7_S[1:0]：增强型 PWM 通道 7 输出脚选择位

C7_S[1:0]	PWM7
00	P2.7
01	P1.7
10	P6.7
11	保留

19.2 PWM 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMSET	增强型 PWM 全局配置寄存器	F1H	-	PWMRST	-	-	-	-	-	ENPWM	x0xx,xxx0
PWMCFG	增强型 PWM 配置寄存器	F6H	-	-	-	-	PWMCBIF	EPWMCBI	ENPWMTA	PWMCEN	xxxx,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCH	PWM 计数器高字节	FF00H	-								x000,0000
PWMCL	PWM 计数器低字节	FF01H									0000,0000
PWMCKS	PWM 时钟选择	FF02H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWMTADCH	PWM 触发 ADC 计数高字节	FF03H	-								x000,0000
PWMTADCL	PWM 触发 ADC 计数低字节	FF04H									0000,0000
PWMIF	PWM 中断标志寄存器	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDCR	PWM 异常检测控制寄存器	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM0T1H	PWM0T1 计数值高字节	FF10H	-								x000,0000
PWM0T1L	PWM0T1 计数值低字节	FF11H									0000,0000
PWM0T2H	PWM0T2 计数值高字节	FF12H	-								x000,0000
PWM0T2L	PWM0T2 计数值低字节	FF13H									0000,0000
PWM0CR	PWM0 控制寄存器	FF14H	ENO	INI	-	C0_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM0HLD	PWM0 电平保持控制寄存器	FF15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM1T1H	PWM1T1 计数值高字节	FF18H	-								x000,0000
PWM1T1L	PWM1T1 计数值低字节	FF19H									0000,0000
PWM1T2H	PWM1T2 计数值高字节	FF1AH	-								x000,0000
PWM1T2L	PWM1T2 计数值低字节	FF1BH									0000,0000
PWM1CR	PWM1 控制寄存器	FF1CH	ENO	INI	-	C1_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM1HLD	PWM1 电平保持控制寄存器	FF1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM2T1H	PWM2T1 计数值高字节	FF20H	-								x000,0000
PWM2T1L	PWM2T1 计数值低字节	FF21H									0000,0000
PWM2T2H	PWM2T2 计数值高字节	FF22H	-								x000,0000
PWM2T2L	PWM2T2 计数值低字节	FF23H									0000,0000
PWM2CR	PWM2 控制寄存器	FF24H	ENO	INI	-	C2_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM2HLD	PWM2 电平保持控制寄存器	FF25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM3T1H	PWM3T1 计数值高字节	FF28H	-								x000,0000
PWM3T1L	PWM3T1 计数值低字节	FF29H									0000,0000
PWM3T2H	PWM3T2 计数值高字节	FF2AH	-								x000,0000
PWM3T2L	PWM3T2 计数值低字节	FF2BH									0000,0000
PWM3CR	PWM3 控制寄存器	FF2CH	ENO	INI	-	C3_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM3HLD	PWM3 电平保持控制寄存器	FF2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM4T1H	PWM4T1 计数值高字节	FF30H	-								x000,0000
PWM4T1L	PWM4T1 计数值低字节	FF31H									0000,0000
PWM4T2H	PWM4T2 计数值高字节	FF32H	-								x000,0000
PWM4T2L	PWM4T2 计数值低字节	FF33H									0000,0000

PWM4CR	PWM4 控制寄存器	FF34H	ENO	INI	-	C4_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM4HLD	PWM4 电平保持控制寄存器	FF35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM5T1H	PWM5T1 计数值高字节	FF38H	-								x000,0000
PWM5T1L	PWM5T1 计数值低字节	FF39H									0000,0000
PWM5T2H	PWM5T2 计数值高字节	FF3AH	-								x000,0000
PWM5T2L	PWM5T2 计数值低字节	FF3BH									0000,0000
PWM5CR	PWM5 控制寄存器	FF3CH	ENO	INI	-	C5_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM5HLD	PWM5 电平保持控制寄存器	FF3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM6T1H	PWM6T1 计数值高字节	FF40H	-								x000,0000
PWM6T1L	PWM6T1 计数值低字节	FF41H									0000,0000
PWM6T2H	PWM6T2 计数值高字节	FF42H	-								x000,0000
PWM6T2L	PWM6T2 计数值低字节	FF43H									0000,0000
PWM6CR	PWM6 控制寄存器	FF44H	ENO	INI	-	C6_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM6HLD	PWM6 电平保持控制寄存器	FF45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM7T1H	PWM7T1 计数值高字节	FF48H	-								x000,0000
PWM7T1L	PWM7T1 计数值低字节	FF49H									0000,0000
PWM7T2H	PWM7T2 计数值高字节	FF4AH	-								x000,0000
PWM7T2L	PWM7T2 计数值低字节	FF4BH									0000,0000
PWM7CR	PWM7 控制寄存器	FF4CH	ENO	INI	-	C7_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM7HLD	PWM7 电平保持控制寄存器	FF4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00

19.2.1 增强型 PWM 全局配置寄存器（PWMSET）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMSET	F1H	-	PWMRST	-	-	-	-	-	ENPWM

PWMRST：软件复位 PWM。

0：无效

1：复位所有 PWM 的 XFR 寄存器，但不复位 SFR。（需要软件清零）

ENPWM0：PWM 使能位（包括 PWM0~PWM7）。

0：关闭 PWM

1：使能 PWM

19.2.2 增强型 PWM 配置寄存器（PWMCFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG	F6H	-	-	-	-	PWMCBIF	EPWMCBI	ENPWMTA	PWMCEN

PWMCBIF：PWM 计数器归零中断标志位。

当 15 位的 PWM 计数器记满溢出归零时，硬件自动将此位置 1，并向 CPU 提出中断请求，此标志位需要软件清零。

EPWMCBI：PWM 计数器归零中断使能位。

0：关闭 PWM 计数器归零中断（PWMCBIF 依然会被硬件置位）

1：使能 PWM 计数器归零中断

EPWMTA：PWM 是否与 ADC 关联。

0：PWM 与 ADC 不关联

1：PWM 与 ADC 相关联。

允许在 PWM 周期中某个时间点触发 A/D 转换，使用 PWMTADCH 和 PWMTADCL 进行设置。（**注意：需要同时使能 ADC_CONTR 寄存器中的 ADC_POWER 位和 ADC_EPWMT 位，PWM 只是会自动将 ADC_START 置 1**）

PWMCEN： PWM 波形发生器开始计数。

0：PWM 停止计数

1：PWM 计数器开始计数

关于 PWMCEN 控制位的重要说明:

- PWMCEN 一旦被使能后，内部的 PWM 计数器会立即开始计数，并与 T1/T2 的值进行比较。所以 PWMCEN 必须在其他所有的 PWM 设置（包括 T1/T2 的设置、初始电平的设置、PWM 异常检测的设置以及 PWM 中断设置）都完成后，最后才能使能 PWMCEN 位。
- 在 PWM 计数器计数的过程中，PWMCEN 控制位被关闭时，PWM 计数会立即停止，当再次使能 PWMCEN 控制位时，PWM 的计数会从 0 开始重新计数，而不会记忆 PWM 停止计数前的计数值
- **特别注意：当 PWMCEN 由 0 变为 1 时，内部的 PWM 计数器是从之前的不确定值归零后重新开始计数，所以此时会产生立即产生一个归零中断，当用户需要使用 PWM 的归零中断时，需特别注意这一点，即第一个归零中断并不是真正的 PWM 周期记满后归零所产生的。**

19.2.3 PWM 中断标志寄存器（PWMIF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMIF	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

CiIF：PWM 的第 i 通道中断标志位。（i＝0～7）

可设置在各路 PWM 的 T1 和 T2。当所设置的点发生匹配事件时，硬件自动将此位置 1，并向 CPU 提出中断请求，此标志位需要软件清零。

19.2.4 PWM 异常检测控制寄存器（PWMnFDCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

INVCMP：比较器器结果异常信号处理

0：比较器器结果由低变高为异常信号

1：比较器器结果由高变低为异常信号

INVIO：外部 PWMFLT 端口异常信号处理

0：外部 PWMFLT 端口信号由低变高为异常信号

1：外部 PWMFLT 端口信号由高变低为异常信号

ENFD：PWM 外部异常检测控制位。

0：关闭 PWM 外部异常检测功能

1：使能 PWM 外部异常检测功能

FLTFLIO：发生 PWM 外部异常时对 PWM 输出口控制位。

0：发生 PWM 外部异常时，PWM 的输出口不作任何改变

1：发生 PWM 外部异常时，PWM 的输出口立即被设置为高阻输入模式。

（注：只有 ENO=1 所对应的端口才会被强制悬空）

EFDI：PWM 异常检测中断使能位。

0：关闭 PWM 异常检测中断（FDIF 依然会被硬件置位）

1：使能 PWM 异常检测中断

FDCMP：比较器输出异常检测使能位。

0：比较器与 PWM 无关

1：设定 PWM 异常检测源为比较器输出（异常类型由 INVCMP 设定）

FDIO：PWMFLT 端口电平异常检测使能位。

0：PWMFLT 端口电平与 PWM 无关

1：设定 PWM 异常检测源为 PWMFLT 端口（异常类型由 INVIO 设定）

FDIF：PWM 异常检测中断标志位。

当发生 PWM 异常时，硬件自动将此位置 1。当 EFDI==1 时，程序会跳转到相应中断入口执行中断服务程序。需要软件清零。

19.2.5 PWM 计数器寄存器（PWMCH，PWMCL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCH	FF00H	-							
PWMCL	FF01H								

PWMCH：PWM 计数器周期值的高 7 位。

PWMCL：PWM 计数器周期值的低 8 位。

PWM 计数器为一个 15 位的寄存器，可设定 1~32767 之间的任意值作为 PWM 的周期。PWM 波形发生器内部的计数器从 0 开始计数，每个 PWM 时钟周期递增 1，当内部计数器的计数值达到[PWMCH，PWMCL]所设定的 PWM 周期时，PWM 波形发生器内部的计数器将会从 0 重新开始开始计数，硬件会自动将 PWM 归零中断中断标志位 PWMCBIF 置 1，若 EPWMCBI=1，程序将跳转到相应中断入口执行中断服务程序。

19.2.6 PWM 时钟选择寄存器（PWMCKS），输出频率计算公式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMCKS	FF02H	-	-	-	SELT2	PWM_PS[3:0]			

SELT2：PWM 时钟源选择。

0：PWM 时钟源为系统时钟经分频器分频之后的时钟

1：PWM 时钟源为定时器 2 的溢出脉冲

PWM_PS[3:0]：系统时钟预分频参数

SELT2	PWM_PS[3:0]	PWM 输入时钟源频率
1	xxxx	定时器 2 的溢出脉冲
0	0000	SYSclk/1
0	0001	SYSclk/2
0	0010	SYSclk/3
…	…	…
0	x	SYSclk/(x+1)
…	…	…
0	1111	SYSclk/16

PWM 输出频率计算公式

时钟源选择（SELT2）	PWM输出频率计算公式
SELT2＝0 （系统时钟）	$\text{PWM输出频率} = \frac{\text{系统工作频率SYSclk}}{(\text{PWM\_PS} + 1) \times (\text{[PWMCH, PWMCL]} + 1)}$
SELT2＝1 （定时器2的溢出脉冲）	$\text{PWM输出频率} = \frac{\text{定时器2的溢出脉冲频率}}{(\text{[PWMCH, PWMCL]} + 1)}$

19.2.7 PWM 触发 ADC 计数器寄存器（PWMTADC）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMTADCH	FF03H	-							
PWMTADCL	FF04H								

PWMTADCH：PWM 触发 ADC 时间点的高 7 位。

PWMTADCL：PWM 触发 ADC 时间点的低 8 位。

若 EPWMTA =1 且 ADC_POWER=1,ADC_EPWMT=1 时，在 PWM 的计数周期中，当 PWM 的内部计数值与{PWMTADCH, PWMTADCL}所组成一个 15 位的寄存器的值相等时，硬件自动触发 A/D 转换。

19.2.8 PWM 电平输出设置计数值寄存器（PWMnT1，PWMnT2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0T1H	FF10H	-							
PWM0T1L	FF11H								
PWM0T2H	FF12H	-							
PWM0T2L	FF13H								
PWM1T1H	FF18H	-							
PWM1T1L	FF19H								
PWM1T2H	FF1AH	-							
PWM1T2L	FF1BH								
PWM2T1H	FF20H	-							
PWM2T1L	FF21H								
PWM2T2H	FF22H	-							
PWM2T2L	FF23H								
PWM3T1H	FF28H	-							
PWM3T1L	FF29H								
PWM3T2H	FF2AH	-							
PWM3T2L	FF2BH								
PWM4T1H	FF30H	-							
PWM4T1L	FF31H								
PWM4T2H	FF32H	-							

PWM4T2L	FF33H		
PWM5T1H	FF38H	-	
PWM5T1L	FF39H		
PWM5T2H	FF3AH	-	
PWM5T2L	FF3BH		
PWM6T1H	FF40H	-	
PWM6T1L	FF41H		
PWM6T2H	FF42H	-	
PWM6T2L	FF43H		
PWM7T1H	FF48H	-	
PWM7T1L	FF49H		
PWM7T2H	FF4AH	-	
PWM7T2L	FF4BH		

PWMiT1H：PWM 的通道 i 的 T1 计数器值的高 7 位。（i＝0～7）

PWMiT1L：PWM 的通道 i 的 T1 计数器值的低 8 位。（i＝0～7）

PWMiT2H：PWM 的通道 i 的 T2 计数器值的高 7 位。（i＝0～7）

PWMiT2L：PWM 的通道 i 的 T2 计数器值的低 8 位。（i＝0～7）

每组 PWM 的每个通道的{PWMiT1H, PWMiT1L}和{PWMiT2H, PWMiT2L}分别组合成两个 15 位的寄存器，用于控制各路 PWM 每个周期中输出 PWM 波形的两个触发点。在 PWM 的计数周期中，当 PWM 的内部计数值与所设置的 T1 的值{PWMiT1H, PWMiT1L}相等时，PWM 的输出低电平；当 PWM 的内部计数值与 T2 的值{PWMiT2H, PWMiT2L}相等时，PWM 的输出高电平。

注意：当{PWMiT1H, PWMiT1L}与{PWMiT2H, PWMiT2L}的值设置相等时，若 PWM 的内部计数值与所设置的 T1/T2 的值相等，则会固定输出低电平。

19.2.9 PWM 通道控制寄存器（PWMnCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF14H	ENO	INI	-	C0_S[1:0]		ENI	ENT2I	ENT1I
PWM1CR	FF1CH	ENO	INI	-	C1_S[1:0]		ENI	ENT2I	ENT1I
PWM2CR	FF24H	ENO	INI	-	C2_S[1:0]		ENI	ENT2I	ENT1I
PWM3CR	FF2CH	ENO	INI	-	C3_S[1:0]		ENI	ENT2I	ENT1I
PWM4CR	FF34H	ENO	INI	-	C4_S[1:0]		ENI	ENT2I	ENT1I
PWM5CR	FF3CH	ENO	INI	-	C5_S[1:0]		ENI	ENT2I	ENT1I
PWM6CR	FF44H	ENO	INI	-	C6_S[1:0]		ENI	ENT2I	ENT1I
PWM7CR	FF4CH	ENO	INI	-	C7_S[1:0]		ENI	ENT2I	ENT1I

ENO：PWMi 输出使能位。（i＝0～7）

0：PWM 的 i 通道相应 PWMi 端口为 GPIO

1：PWM 的 i 通道相应 PWMi 端口为 PWM 输出口，受 PWM 波形发生器控制

INI：设置 PWMi 输出端口的初始电平。（i＝0～7）

0：PWM 的 i 通道初始电平为低电平

1：PWM 的 i 通道初始电平为高电平

Ci_S[1:0]：切换 PWMi 输出端口（i＝0～7）

详情见“功能脚切换”章节

ENI： PWM 的 i 通道中断使能控制位。（i＝0～7）

0：关闭 PWM 的 i 通道的 PWM 中断

1：使能 PWM 的 i 通道的 PWM 中断

ENT2I：PWM 的 i 通道在第 2 个触发点中断使能控制位。（i＝0～7）

0：关闭 PWM 的 i 通道在第 2 个触发点中断

1：使能 PWM 的 i 通道在第 2 个触发点中断

ENT1I：PWM 的 i 通道在第 1 个触发点中断使能控制位。（i＝0～7）

0：关闭 PWM 的 i 通道在第 1 个触发点中断

1：使能 PWM 的 i 通道在第 1 个触发点中断

19.2.10 PWM 通道电平保持控制寄存器（PWMnHLD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM0HLD	FF15H	-	-	-	-	-	-	HLDH	HLDL
PWM1HLD	FF1DH	-	-	-	-	-	-	HLDH	HLDL
PWM2HLD	FF25H	-	-	-	-	-	-	HLDH	HLDL
PWM3HLD	FF2DH	-	-	-	-	-	-	HLDH	HLDL
PWM4HLD	FF35H	-	-	-	-	-	-	HLDH	HLDL
PWM5HLD	FF3DH	-	-	-	-	-	-	HLDH	HLDL
PWM6HLD	FF45H	-	-	-	-	-	-	HLDH	HLDL
PWM7HLD	FF4DH	-	-	-	-	-	-	HLDH	HLDL

HLDH：PWM 的 i 通道强制输出高电平控制位。（i＝0～7）

0：PWM 的 i 通道正常输出

1：PWM 的 i 通道强制输出高电平

HLDL：PWM 的 i 通道强制输出低电平控制位。（i＝0～7）

0：PWM 的 i 通道正常输出

1：PWM 的 i 通道强制输出低电平

19.3 范例程序

19.3.1 输出任意周期和任意占空比的波形

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

sfr         PWMSET          =   0xF1;
sfr         PWMCFG          =   0xF6;

#define     PWMC                (*(unsigned int   volatile xdata *)0xFF00)
#define     PWMCH               (*(unsigned char volatile xdata *)0xFF00)
#define     PWMCL               (*(unsigned char volatile xdata *)0xFF01)
#define     PWMCKS              (*(unsigned char volatile xdata *)0xFF02)
#define     PWMTADC             (*(unsigned int   volatile xdata *)0xFF03)
#define     PWMTADCH            (*(unsigned char volatile xdata *)0xFF03)
#define     PWMTADCL            (*(unsigned char volatile xdata *)0xFF04)
#define     PWMIF               (*(unsigned char volatile xdata *)0xFF05)
#define     PWMFDCR             (*(unsigned char volatile xdata *)0xFF06)
#define     PWM0T1              (*(unsigned int   volatile xdata *)0xFF10)
#define     PWM0T1H             (*(unsigned char volatile xdata *)0xFF10)
#define     PWM0T1L             (*(unsigned char volatile xdata *)0xFF11)
#define     PWM0T2H             (*(unsigned char volatile xdata *)0xFF12)
#define     PWM0T2              (*(unsigned int   volatile xdata *)0xFF12)
#define     PWM0T2L             (*(unsigned char volatile xdata *)0xFF13)
#define     PWM0CR              (*(unsigned char volatile xdata *)0xFF14)
#define     PWM0HLD             (*(unsigned char volatile xdata *)0xFF15)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMSET = 0x01;                  //使能 PWM 模块（必须先使能模块后面的设置才有效）

    P_SW2 = 0x80;
    PWMCKS = 0x00;                  //PWM 时钟为系统时钟
    PWMC = 0x1000;                  //设置 PWM 周期为 1000H 个 PWM 时钟
    PWM0T1= 0x0100;                 //在计数值为 100H 地方 PWM0 通道输出低电平
    PWM0T2= 0x0500;                 //在计数值为 500H 地方 PWM0 通道输出高电平
    PWM0CR= 0x80;                   //使能 PWM0 输出
    P_SW2 = 0x00;

    PWMCFG = 0x01;                  //启动 PWM 模块

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG      DATA    0F6H

PWMCH       EQU     0FF00H
PWMCL       EQU     0FF01H
PWMCKS      EQU     0FF02H
PWMTADCH    EQU     0FF03H
PWMTADCL    EQU     0FF04H
PWMIF       EQU     0FF05H
PWMFDCR     EQU     0FF06H
PWM0T1H     EQU     0FF10H
PWM0T1L     EQU     0FF11H
PWM0T2H     EQU     0FF12H
PWM0T2L     EQU     0FF13H
PWM0CR      EQU     0FF14H
PWM0HLD     EQU     0FF15H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
```

```
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     PWMSET,#01H             ;使能 PWM 模块（必须先使能模块后面的设置才有效）

        MOV     P_SW2,#80H
        CLR     A
        MOV     DPTR,#PWMCKS
        MOVX    @DPTR,A                 ;PWM 时钟为系统时钟
        MOV     A,#10H
        MOV     DPTR,#PWMCH             ;设置 PWM 周期为 1000H 个 PWM 时钟
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWMCL
        MOVX    @DPTR,A
        MOV     A,#01H
        MOV     DPTR,#PWM0T1H           ;在计数值为 100H 地方 PWM0 通道输出低电平
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T1L
        MOVX    @DPTR,A
        MOV     A,#05H
        MOV     DPTR,#PWM0T2H           ;在计数值为 500H 地方 PWM0 通道输出高电平
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T2L
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM0CR            ;使能 PWM0 输出
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG,#01H             ;启动 PWM 模块

        JMP     $

        END
```

19.3.2 两路 PWM 实现互补对称带死区控制的波形

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

sfr         P_SW2       =   0xba;

sfr         PWMSET      =   0xF1;
sfr         PWMCFG      =   0xF6;

#define     PWMC            (*(unsigned int   volatile xdata *)0xFF00)
#define     PWMCH           (*(unsigned char volatile xdata *)0xFF00)
#define     PWMCL           (*(unsigned char volatile xdata *)0xFF01)
#define     PWMCKS          (*(unsigned char volatile xdata *)0xFF02)
#define     PWMTADC         (*(unsigned int   volatile xdata *)0xFF03)
#define     PWMTADCH        (*(unsigned char volatile xdata *)0xFF03)
#define     PWMTADCL        (*(unsigned char volatile xdata *)0xFF04)
#define     PWMIF           (*(unsigned char volatile xdata *)0xFF05)
#define     PWMFDCR         (*(unsigned char volatile xdata *)0xFF06)
#define     PWM0T1          (*(unsigned int   volatile xdata *)0xFF10)
#define     PWM0T1H         (*(unsigned char volatile xdata *)0xFF10)
#define     PWM0T1L         (*(unsigned char volatile xdata *)0xFF11)
#define     PWM0T2          (*(unsigned int   volatile xdata *)0xFF12)
#define     PWM0T2H         (*(unsigned char volatile xdata *)0xFF12)
#define     PWM0T2L         (*(unsigned char volatile xdata *)0xFF13)
#define     PWM0CR          (*(unsigned char volatile xdata *)0xFF14)
#define     PWM0HLD         (*(unsigned char volatile xdata *)0xFF15)
#define     PWM1T1          (*(unsigned int   volatile xdata *)0xFF18)
#define     PWM1T1H         (*(unsigned char volatile xdata *)0xFF18)
#define     PWM1T1L         (*(unsigned char volatile xdata *)0xFF19)
#define     PWM1T2          (*(unsigned int   volatile xdata *)0xFF1A)
#define     PWM1T2H         (*(unsigned char volatile xdata *)0xFF1A)
#define     PWM1T2L         (*(unsigned char volatile xdata *)0xFF1B)
#define     PWM1CR          (*(unsigned char volatile xdata *)0xFF1C)
#define     PWM1HLD         (*(unsigned char volatile xdata *)0xFF1D)

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
```

```
    P5M1 = 0x00;

    PWMSET = 0x01;                          //使能 PWM 模块（必须先使能模块后面的设置才有效）

    P_SW2 = 0x80;
    PWMCKS = 0x00;                          //PWM 时钟为系统时钟
    PWMC = 0x0800;                          //设置 PWM 周期为 0800H 个 PWM 时钟
    PWM0T1= 0x0100;                         //PWM0 在计数值为 100H 地方输出低电平
    PWM0T2= 0x0700;                         //PWM0 在计数值为 700H 地方输出高电平
    PWM1T2= 0x0080;                         //PWM1 在计数值为 0080H 地方输出高电平
    PWM1T1= 0x0780;                         //PWM1 在计数值为 0780H 地方输出低电平
    PWM0CR= 0x80;                           //使能 PWM0 输出
    PWM1CR= 0x80;                           //使能 PWM1 输出
    P_SW2 = 0x00;

    PWMCFG = 0x01;                          //启动 PWM 模块

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG      DATA    0F6H

PWMCH       EQU     0FF00H
PWMCL       EQU     0FF01H
PWMCKS      EQU     0FF02H
PWMTADCH    EQU     0FF03H
PWMTADCL    EQU     0FF04H
PWMIF       EQU     0FF05H
PWMFDCR     EQU     0FF06H
PWM0T1H     EQU     0FF10H
PWM0T1L     EQU     0FF11H
PWM0T2H     EQU     0FF12H
PWM0T2L     EQU     0FF13H
PWM0CR      EQU     0FF14H
PWM0HLD     EQU     0FF15H
PWM1T1H     EQU     0FF18H
PWM1T1L     EQU     0FF19H
PWM1T2H     EQU     0FF1AH
PWM1T2L     EQU     0FF1BH
PWM1CR      EQU     0FF1CH
PWM1HLD     EQU     0FF1DH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
```

```
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     PWMSET,#01H             ;使能 PWM 模块（必须先使能模块后面的设置才有效）

        MOV     P_SW2,#80H
        CLR     A
        MOV     DPTR,#PWMCKS
        MOVX    @DPTR,A                 ;PWM 时钟为系统时钟
        MOV     A,#08H
        MOV     DPTR,#PWMCH             ;设置 PWM 周期为 0800H 个 PWM 时钟
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWMCL
        MOVX    @DPTR,A
        MOV     A,#01H
        MOV     DPTR,#PWM0T1H           ;PWM0 在计数值为 0100H 地方输出低电平
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T1L
        MOVX    @DPTR,A
        MOV     A,#07H
        MOV     DPTR,#PWM0T2H           ;PWM0 在计数值为 0700H 地方输出高电平
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T2L
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM1T2H           ;PWM1 在计数值为 0080H 地方输出高电平
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM1T2L
        MOVX    @DPTR,A
        MOV     A,#07H
        MOV     DPTR,#PWM1T1H           ;PWM1 在计数值为 0780H 地方输出低电平
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM1T1L
        MOVX    @DPTR,A
        MOV     A,#080H
```

```
        MOV     DPTR,#PWM0CR        ;使能 PWM0 输出
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM1CR        ;使能 PWM1 输出
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG,#01H         ;启动 PWM 模块

        JMP     $

        END
```

19.3.3 PWM 实现渐变灯（呼吸灯）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   CYCLE             0x1000

sfr       P_SW2       =     0xba;

sfr       PWMSET      =     0xF1;
sfr       PWMCFG      =     0xF6;

#define   PWMC              (*(unsigned int  volatile xdata *)0xFF00)
#define   PWMCH             (*(unsigned char volatile xdata *)0xFF00)
#define   PWMCL             (*(unsigned char volatile xdata *)0xFF01)
#define   PWMCKS            (*(unsigned char volatile xdata *)0xFF02)
#define   PWMTADC           (*(unsigned int  volatile xdata *)0xFF03)
#define   PWMTADCH          (*(unsigned char volatile xdata *)0xFF03)
#define   PWMTADCL          (*(unsigned char volatile xdata *)0xFF04)
#define   PWMIF             (*(unsigned char volatile xdata *)0xFF05)
#define   PWMFDCR           (*(unsigned char volatile xdata *)0xFF06)
#define   PWM0T1            (*(unsigned int  volatile xdata *)0xFF10)
#define   PWM0T1H           (*(unsigned char volatile xdata *)0xFF10)
#define   PWM0T1L           (*(unsigned char volatile xdata *)0xFF11)
#define   PWM0T2H           (*(unsigned char volatile xdata *)0xFF12)
#define   PWM0T2            (*(unsigned int  volatile xdata *)0xFF12)
#define   PWM0T2L           (*(unsigned char volatile xdata *)0xFF13)
#define   PWM0CR            (*(unsigned char volatile xdata *)0xFF14)
#define   PWM0HLD           (*(unsigned char volatile xdata *)0xFF15)

sfr       P0M1        =     0x93;
sfr       P0M0        =     0x94;
sfr       P1M1        =     0x91;
sfr       P1M0        =     0x92;
sfr       P2M1        =     0x95;
sfr       P2M0        =     0x96;
sfr       P3M1        =     0xb1;
sfr       P3M0        =     0xb2;
sfr       P4M1        =     0xb3;
```

```
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void PWM0_Isr() interrupt 22
{
    static bit dir = 1;
    static int val = 0;

    if (PWMCFG & 0x08)
    {
        PWMCFG &= ~0x08;                    //清中断标志
        if (dir)
        {
            val++;
            if (val >= CYCLE) dir = 0;
        }
        else
        {
            val--;
            if (val <= 1) dir = 1;
        }
        _push_(P_SW2);
        P_SW2 |= 0x80;
        PWM0T2 = val;
        _pop_(P_SW2);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMSET = 0x01;                          //使能 PWM 模块（必须先使能模块后面的设置才有效）

    P_SW2 = 0x80;
    PWMCKS = 0x00;                          //PWM 时钟为系统时钟
    PWMC = CYCLE;                           //设置 PWM 周期
    PWM0T1= 0x0000;
    PWM0T2= 0x0001;
    PWM0CR= 0x80;                           //使能 PWM 输出
    P_SW2 = 0x00;

    PWMCFG = 0x05;                          //启动 PWM 模块并使能 PWM 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CYCLE       EQU     1000H

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG      DATA    0F6H

PWMCH       EQU     0FF00H
PWMCL       EQU     0FF01H
PWMCKS      EQU     0FF02H
PWMTADCH    EQU     0FF03H
PWMTADCL    EQU     0FF04H
PWMIF       EQU     0FF05H
PWMFDCR     EQU     0FF06H
PWM0T1H     EQU     0FF10H
PWM0T1L     EQU     0FF11H
PWM0T2H     EQU     0FF12H
PWM0T2L     EQU     0FF13H
PWM0CR      EQU     0FF14H
PWM0HLD     EQU     0FF15H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

DIR         BIT     20H.0
VALL        DATA    21H
VALH        DATA    22H

            ORG     0000H
            LJMP    MAIN
            ORG     00B3H
            LJMP    PWM0ISR

            ORG     0100H
PWMISR:
            PUSH    ACC
            PUSH    PSW
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2

            MOV     P_SW2,#80H
            MOV     A,PWMCFG
            JNB     ACC.3,ISREXIT
```

```
            ANL     PWMCFG,#NOT 08H         ;清中断标志
            JNB     DIR,PWMDN
PWMUP:
            MOV     A,VALL
            ADD     A,#1
            MOV     VALL,A
            MOV     A,VALH
            ADDC    A,#0
            MOV     VALH,A
            CJNE    A,#HIGH CYCLE,SETPWM
            MOV     A,VALL
            CJNE    A,#LOW CYCLE,SETPWM
            CLR     DIR
            JMP     SETPWM
PWMDN:
            MOV     A,VALL
            ADD     A,#0FFH
            MOV     VALL,A
            MOV     A,VALH
            ADDC    A,#0FFH
            MOV     VALH,A
            JNZ     SETPWM
            MOV     A,VALL
            CJNE    A,#1,SETPWM
            SETB    DIR
SETPWM:
            MOV     A,VALH
            MOV     DPTR,#PWM0T2H
            MOVX    @DPTR,A
            MOV     A,VALL
            MOV     DPTR,#PWM0T2L
            MOVX    @DPTR,A
ISREXIT:
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    DIR
            MOV     VALH,#00H
            MOV     VALL,#01H
```

```
        MOV     PWMSET,#01H             ;使能 PWM 模块（必须先使能模块后面的设置才有效）

        MOV     P_SW2,#80H
        CLR     A
        MOV     DPTR,#PWMCKS
        MOVX    @DPTR,A                 ;PWM 时钟为系统时钟
        MOV     A,#HIGH CYCLE
        MOV     DPTR,#PWMCH             ;设置 PWM 周期
        MOVX    @DPTR,A
        MOV     A,#LOW CYCLE
        MOV     DPTR,#PWMCL
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T1H
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T1L
        MOVX    @DPTR,A
        MOV     A,VALH
        MOV     DPTR,#PWM0T2H
        MOVX    @DPTR,A
        MOV     A,VALL
        MOV     DPTR,#PWM0T2L
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM0CR            ;使能 PWM0 输出
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG,#05H             ;启动 PWM 模块并使能 PWM 中断
        SETB    EA
        JMP     $

        END
```

19.3.4 使用 PWM 触发 ADC 转换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

sfr     PWMSET      =   0xF1;
sfr     PWMCFG      =   0xF6;

#define PWMC        (*(unsigned int   volatile xdata *)0xFF00)
#define PWMCH       (*(unsigned char volatile xdata *)0xFF00)
#define PWMCL       (*(unsigned char volatile xdata *)0xFF01)
#define PWMCKS      (*(unsigned char volatile xdata *)0xFF02)
#define PWMTADC     (*(unsigned int   volatile xdata *)0xFF03)
#define PWMTADCH    (*(unsigned char volatile xdata *)0xFF03)
#define PWMTADCL    (*(unsigned char volatile xdata *)0xFF04)
```

```
#define   PWMIF          (*(unsigned char volatile xdata *)0xFF05)
#define   PWMFDCR        (*(unsigned char volatile xdata *)0xFF06)
#define   PWM0T1         (*(unsigned int  volatile xdata *)0xFF10)
#define   PWM0T1H        (*(unsigned char volatile xdata *)0xFF10)
#define   PWM0T1L        (*(unsigned char volatile xdata *)0xFF11)
#define   PWM0T2H        (*(unsigned char volatile xdata *)0xFF12)
#define   PWM0T2         (*(unsigned int  volatile xdata *)0xFF12)
#define   PWM0T2L        (*(unsigned char volatile xdata *)0xFF13)
#define   PWM0CR         (*(unsigned char volatile xdata *)0xFF14)
#define   PWM0HLD        (*(unsigned char volatile xdata *)0xFF15)

sfr       ADC_CONTR  =   0xbc;
#define   ADC_POWER      0x80
#define   ADC_START      0x40
#define   ADC_FLAG       0x20
#define   ADC_EPWMT      0x10
sfr       ADC_RES    =   0xbd;
sfr       ADC_RESL   =   0xbe;

sbit      EADC       =   IE^5;

sfr       P0M1       =   0x93;
sfr       P0M0       =   0x94;
sfr       P1M1       =   0x91;
sfr       P1M0       =   0x92;
sfr       P2M1       =   0x95;
sfr       P2M0       =   0x96;
sfr       P3M1       =   0xb1;
sfr       P3M0       =   0xb2;
sfr       P4M1       =   0xb3;
sfr       P4M0       =   0xb4;
sfr       P5M1       =   0xc9;
sfr       P5M0       =   0xca;

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x01;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    ADC_CONTR = ADC_POWER | ADC_EPWMT | 0;      //选择 P1.0 为 ADC 输入通道
    delay();                //等待 ADC 电源稳定
    EADC = 1;
```

```
    PWMSET = 0x01;                          //使能 PWM 模块（必须先使能模块后面的设置才有效）

    P_SW2 = 0x80;
    PWMCKS = 0x00;                          //PWM 时钟为系统时钟
    PWMC = 0x1000;                          //设置 PWM 周期为 1000H 个 PWM 时钟
    PWM0T1= 0x0100;                         //在计数值为 100H 地方 PWM0 通道输出低电平
    PWM0T2= 0x0500;                         //在计数值为 500H 地方 PWM0 通道输出高电平
    PWMTADC = 0x0200;                       //设置 ADC 触发点
    PWM0CR= 0x80;                           //使能 PWM0 输出
    P_SW2 = 0x00;

    PWMCFG = 0x07;                          //启动 PWM 模块并使能 PWM 中断以及 ADC 触发
    EA = 1;

    while (1);
}

void pwm0_isr() interrupt 22
{
    if (PWMCFG & 0x08)
    {
        PWMCFG &= ~0x08;
    }
}

void ADC_ISR() interrupt 5
{
    ADC_CONTR &= ~ADC_FLAG;
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG      DATA    0F6H

PWMCH       EQU     0FF00H
PWMCL       EQU     0FF01H
PWMCKS      EQU     0FF02H
PWMTADCH    EQU     0FF03H
PWMTADCL    EQU     0FF04H
PWMIF       EQU     0FF05H
PWMFDCR     EQU     0FF06H
PWM0T1H     EQU     0FF10H
PWM0T1L     EQU     0FF11H
PWM0T2H     EQU     0FF12H
PWM0T2L     EQU     0FF13H
PWM0CR      EQU     0FF14H
PWM0HLD     EQU     0FF15H

ADC_CONTR   DATA    0BCH
ADC_POWER   EQU     080H
ADC_START   EQU     040H
ADC_FLAG    EQU     020H
ADC_EPWMT   EQU     010H
ADC_RES     DATA    0BDH
```

```
ADC_RESL    DATA    0BEH

EADC        BIT     IE.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
            LJMP    ADCISR
            ORG     00B3H
            LJMP    PWMISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT ADC_FLAG
            RETI

PWMISR:
            PUSH    ACC
            MOV     A,PWMCFG
            JNB     ACC.3,ISREXIT
            ANL     PWMCFG,#NOT 08H
ISREXIT:
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     ADC_CONTR,# ADC_POWER | ADC_EPWMT
            SETB    EADC

            MOV     PWMSET,#01H             ;使能 PWM 模块（必须先使能模块后面的设置才有效）

            MOV     P_SW2,#80H
```

```
CLR     A
MOV     DPTR,#PWMCKS
MOVX    @DPTR,A         ;PWM 时钟为系统时钟
MOV     A,#10H
MOV     DPTR,#PWMCH     ;设置 PWM 周期为 1000H 个 PWM 时钟
MOVX    @DPTR,A
MOV     A,#00H
MOV     DPTR,#PWMCL
MOVX    @DPTR,A
MOV     A,#01H
MOV     DPTR,#PWM0T1H   ;在计数值为 100H 地方 PWM0 通道输出低电平
MOVX    @DPTR,A
MOV     A,#00H
MOV     DPTR,#PWM0T1L
MOVX    @DPTR,A
MOV     A,#05H
MOV     DPTR,#PWM0T2H   ;在计数值为 500H 地方 PWM0 通道输出高电平
MOVX    @DPTR,A
MOV     A,#00H
MOV     DPTR,#PWM0T2L
MOVX    @DPTR,A
MOV     A,#02H
MOV     DPTR,#PWMTADCH  ;置 ADC 触发点
MOVX    @DPTR,A
MOV     A,#00H
MOV     DPTR,#PWM0TADCL
MOVX    @DPTR,A
MOV     A,#80H
MOV     DPTR,#PWM0CR    ;使能 PWM0 输出
MOVX    @DPTR,A
MOV     P_SW2,#00H

MOV     PWMCFG,#07H     ;启动 PWM 模块并使能 PWM 中断以及 ADC 触发
SETB    EA

JMP     $

END
```

20 同步串行外设接口 SPI

STC8A8K64D4 系列单片机内部集成了一种高速串行通信接口——SPI 接口。SPI 是一种全双工的高速同步通信总线。STC8A8K64D4 系列集成的 SPI 接口提供了两种操作模式：主模式和从模式。

20.1 SPI 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-

SPI_S[1:0]：SPI 功能脚选择位

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P7.4	P7.5	P7.6	P7.7
11	P3.5	P3.4	P3.3	P3.2

20.2 SPI 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH									0000,0000

20.2.1 SPI 状态寄存器（SPSTAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL：SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

20.2.2 SPI 控制寄存器（SPCTL），SPI 速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG：SS 引脚功能控制位

0：SS 引脚确定器件是主机还是从机

1：忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN：SPI 使能控制位

0：关闭 SPI 功能

1：使能 SPI 功能

DORD：SPI 数据位发送/接收的顺序

0：先发送/接收数据的高位（MSB）

1：先发送/接收数据的低位（LSB）

MSTR：器件主/从模式选择位

设置主机模式：

若 SSIG＝0，则 SS 管脚必须为高电平且设置 MSTR 为 1

若 SSIG＝1，则只需要设置 MSTR 为 1（忽略 SS 管脚的电平）

设置从机模式：

若 SSIG＝0，则 SS 管脚必须为低电平（与 MSTR 位无关）

若 SSIG＝1，则只需要设置 MSTR 为 0（忽略 SS 管脚的电平）

CPOL：SPI 时钟极性控制

0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿

1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿

CPHA：SPI 时钟相位控制

0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据（必须 SSIG＝0）

1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样

SPR[1:0]：SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	**SYSclk/2**

20.2.3 SPI 数据寄存器（SPDAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI 发送/接收数据缓冲器。

20.3 SPI 通信方式

SPI 的通信方式通常有 3 种：单主单从（一个主机设备连接一个从机设备）、互为主从（两个设备连接，设备和互为主机和从机）、单主多从（一个主机设备连接多个从机设备）

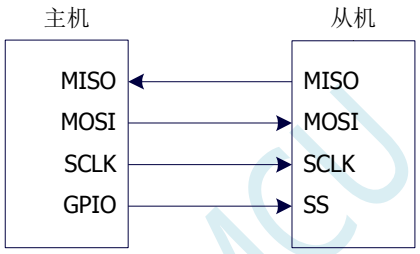
20.3.1 单主单从

两个设备相连，其中一个设备固定作为主机，另外一个固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口连接从机的 SS 管脚，拉低从机的 SS 脚即可使能从机

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主单从连接配置图如下所示：



单主单从配置

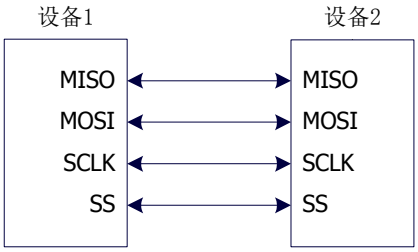
20.3.2 互为主从

两个设备相连，主机和从机不固定。

设置方法 1：两个设备初始化时都设置为 SSIG 设置为 0，MSTR 设置为 1，且将 SS 脚设置为双向口模式输出高电平。此时两个设备都是不忽略 SS 的主机模式。当其中一个设备需要启动传输时，可将自己的 SS 脚设置为输出模式并输出低电平，拉低对方的 SS 脚，这样另一个设备就被强行设置为从机模式了。

设置方法 2：两个设备初始化时都将自己设置成忽略 SS 的从机模式，即将 SSIG 设置为 1，MSTR 设置为 0。当其中一个设备需要启动传输时，先检测 SS 管脚的电平，如果时候高电平，就将自己设置成忽略 SS 的主模式，即可进行数据传输了。

互为主从连接配置图如下所示：



互为主从配置

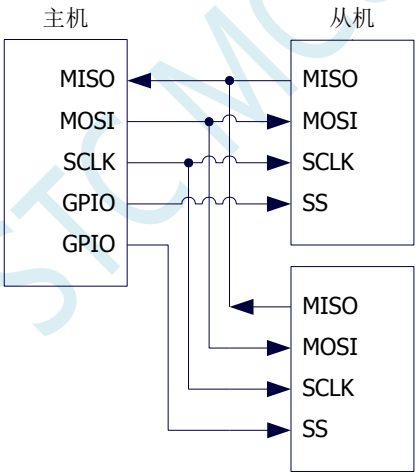
20.3.3 单主多从

多个设备相连，其中一个设备固定作为主机，其他设备固定作为从机。

主机设置：SSIG 设置为 1，MSTR 设置为 1，固定为主机模式。主机可以使用任意端口分别连接各个从机的 SS 管脚，拉低其中一个从机的 SS 脚即可使能相应的从机设备

从机设置：SSIG 设置为 0，SS 管脚作为从机的片选信号。

单主多从连接配置图如下所示：



单主多从配置

20.4 配置 SPI

控制位			通信端口				说明
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	输入	输入	输入	关闭 SPI 功能，SS/MOSI/MISO/SCLK 均为普通 I/O
1	0	0	0	输出	输入	输入	**从机模式**，且被选中
1	0	0	1	高阻	输入	输入	**从机模式**，但未被选中
1	0	1→0	0	输出	输入	输入	**从机模式**，不忽略 SS 且 MSTR 为 1 的主机模式，当 SS 管脚被拉低时，MSTR 将被硬件自动清零，工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	**主机模式**，空闲状态
					输出	输出	**主机模式**，激活状态
1	1	0	x	输出	输入	输入	**从机模式**
1	1	1	x	输入	输出	输出	**主机模式**

从机模式的注意事项：

当 CPHA＝0 时，SSIG 必须为 0（即不能忽略 SS 脚）。在每次串行字节开始还发送前 SS 脚必须拉低，并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作，否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA＝1 时，SSIG 可以置 1（即可以忽略脚）。如果 SSIG＝0，SS 脚可在连续传输之间保持低有效（即一直固定为低电平）。这种方式适用于固定单主单从的系统。

主机模式的注意事项：

在 SPI 中，传输总是由主机启动的。如果 SPI 使能（SPEN=1）并选择作为主机时，主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后，数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后，SPI 时钟发生器停止，传输完成标志（SPIF）置位，如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1，SSIG=0 且 MSTR=1，SPI 使能为主机模式，并将 SS 脚可配置为输入模式化或准双向口模式。这种情况下，另外一个主机可将该脚驱动为低电平，从而将该器件选择为 SPI 从机并向其发送数据。为了避免争夺总线，SPI 系统将该从机的 MSTR 清零，MOSI 和 SCLK 强制变为输入模式，而 MISO 则变为输出模式，同时 SPSTAT 的 SPIF 标志位置 1。

用户软件必须一直对 MSTR 位进行检测，如果该位被一个从机选择动作而被动清零，而用户想继续将 SPI 作为主机，则必须重新设置 MSTR 位，否则将一直处于从机模式。

写冲突

SPI 在发送时为单缓冲，在接收时为双缓冲。这样在前一次发送尚未完成之前，不能将新的数据写入移位寄存器。当发送过程中对数据寄存器 SPDAT 进行写操作时，WCOL 位将被置 1 以指示发生数据写冲突错误。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

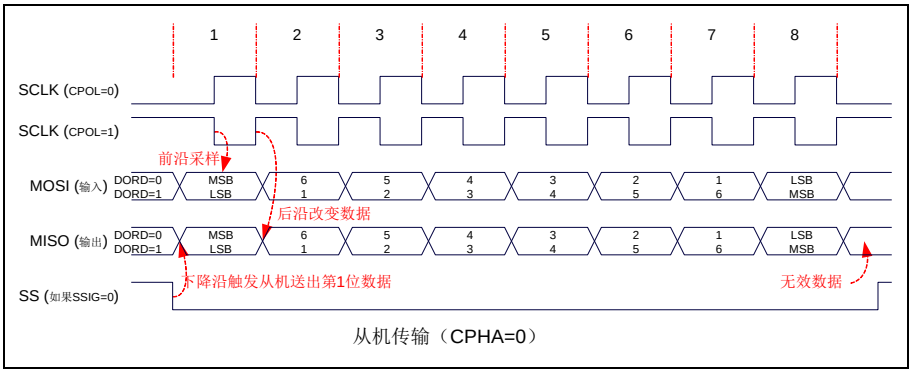
当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

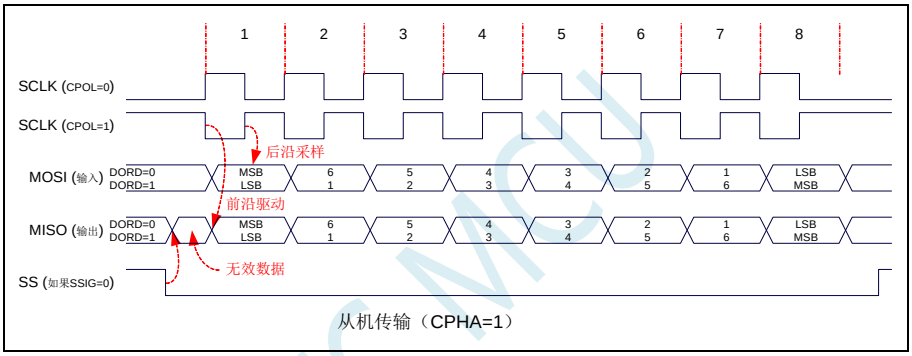
WCOL 可通过软件向其写入“1”清零。

20.5 数据模式

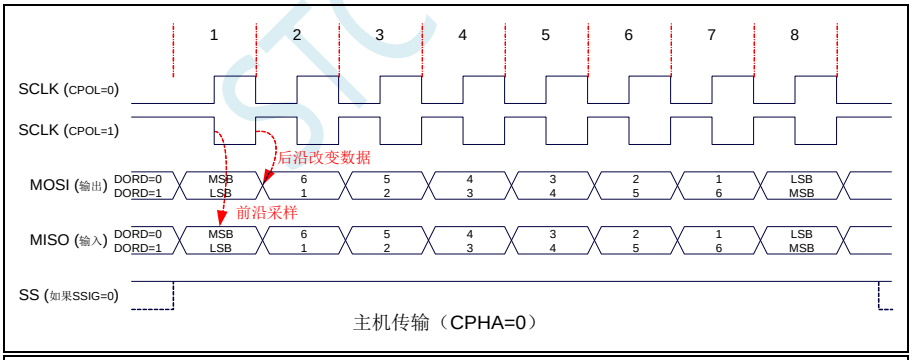
SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



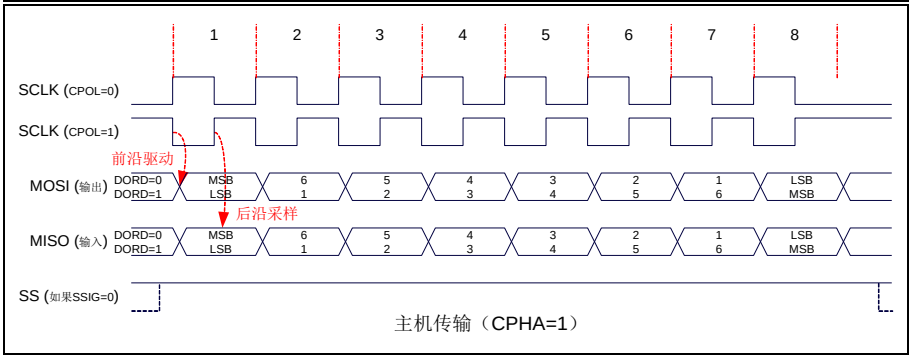
从机传输（CPHA=0）



从机传输（CPHA=1）



主机传输（CPHA=0）



主机传输（CPHA=1）

20.6 范例程序

20.6.1 SPI 单主单从系统主机程序（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;

bit         busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                          //清中断标志
    SS = 1;                                 //拉高从机的 SS 管脚
    busy = 0;
    LED = !LED;                             //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
```

```
    P5M1 = 0x00;

    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;                           //使能 SPI 主机模式
    SPSTAT = 0xc0;                          //清中断标志
    IE2 = ESPI;                             //使能 SPI 中断
    EA = 1;

    while (1)
    {
        while (busy);
        busy = 1;
        SS = 0;                             //拉低从机 SS 管脚
        SPDAT = 0x5a;                       //发送测试数据
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

BUSY        BIT     20H.0
SS          BIT     P1.0
LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            SETB    SS                  ;拉高从机的 SS 管脚
            CLR     BUSY
            CPL     LED
            RETI
```

```
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        SETB    LED
        SETB    SS
        CLR     BUSY

        MOV     SPCTL,#50H          ;使能 SPI 主机模式
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     IE2,#ESPI           ;使能 SPI 中断
        SETB    EA

LOOP:
        JB      BUSY,$
        SETB    BUSY
        CLR     SS                  ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH          ;发送测试数据
        JMP     LOOP

        END
```

20.6.2 SPI 单主单从系统从机程序（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    LED     =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    SPDAT = SPDAT;                  //将接收到的数据回传给主机
    LED = !LED;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                   //使能 SPI 从机模式
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
```

```
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            CPL     LED
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;使能 SPI 从机模式
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     IE2,#ESPI           ;使能 SPI 中断
            SETB    EA

            JMP     $

            END
```

20.6.3 SPI 单主单从系统主机程序（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;

    SPCTL = 0x50;                   //使能 SPI 主机模式
    SPSTAT = 0xc0;                  //清中断标志

    while (1)
    {
        SS = 0;                     //拉低从机 SS 管脚
        SPDAT = 0x5a;               //发送测试数据
        while (!(SPSTAT & 0x80));   //查询完成标志
        SPSTAT = 0xc0;              //清中断标志
        SS = 1;                     //拉高从机的 SS 管脚
        LED = !LED;                 //测试端口
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT  DATA    0CDH
SPCTL   DATA    0CEH
SPDAT   DATA    0CFH
IE2     DATA    0AFH
ESPI    EQU     02H

SS      BIT     P1.0
LED     BIT     P1.1
```

```
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    LED
            SETB    SS

            MOV     SPCTL,#50H          ;使能 SPI 主机模式
            MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
            CLR     SS                  ;拉低从机 SS 管脚
            MOV     SPDAT,#5AH          ;发送测试数据
            MOV     A,SPSTAT            ;查询完成标志
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;清中断标志
            SETB    SS
            CPL     LED
            JMP     LOOP

            END
```

20.6.4 SPI 单主单从系统从机程序（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz
```

```
#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        LED         =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                          //清中断标志
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                           //使能 SPI 从机模式
    SPSTAT = 0xc0;                          //清中断标志

    while (1)
    {
        while (!(SPSTAT & 0x80));           //查询完成标志
        SPSTAT = 0xc0;                      //清中断标志
        SPDAT = SPDAT;                      //将接收到的数据回传给主机
        LED = !LED;                         //测试端口
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;使能 SPI 从机模式
            MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
            MOV     A,SPSTAT            ;查询完成标志
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            CPL     LED
            JMP     LOOP

            END
```

20.6.5 SPI 互为主从系统程序（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;
sbit        KEY         =   P0^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                          //清中断标志
    if (SPCTL & 0x10)
    {                                       //主机模式
        SS = 1;                             //拉高从机的 SS 管脚
        SPCTL = 0x40;                       //重新设置为从机待机
    }
    else
    {                                       //从机模式
        SPDAT = SPDAT;                      //将接收到的数据回传给主机
    }
    LED = !LED;                             //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
```

```
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                   //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;

    while (1)
    {
        if (!KEY)                   //等待按键触发
        {
            SPCTL = 0x50;           //使能 SPI 主机模式
            SS = 0;                 //拉低从机 SS 管脚
            SPDAT = 0x5a;           //发送测试数据
            while (!KEY);           //等待按键释放
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
```

```
            PUSH    ACC
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     A,SPCTL
            JB      ACC.4,MASTER
SLAVE:
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            JMP     ISREXIT
MASTER:
            SETB    SS                  ;拉高从机的 SS 管脚
            MOV     SPCTL,#40H          ;重新设置为从机待机
ISREXIT:
            CPL     LED
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    SS
            SETB    LED
            SETB    KEY

            MOV     SPCTL,#40H          ;使能 SPI 从机模式进行待机
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     IE2,#ESPI           ;使能 SPI 中断
            SETB    EA

LOOP:
            JB      KEY,LOOP            ;等待按键触发
            MOV     SPCTL,#50H          ;使能 SPI 主机模式
            CLR     SS                  ;拉低从机 SS 管脚
            MOV     SPDAT,#5AH          ;发送测试数据
            JNB     KEY,$               ;等待按键释放
            JMP     LOOP

            END
```

20.6.6 SPI 互为主从系统程序（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;
sbit        KEY         =   P0^0;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                       //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                      //清中断标志

    while (1)
    {
        if (!KEY)                       //等待按键触发
        {
            SPCTL = 0x50;               //使能 SPI 主机模式
            SS = 0;                     //拉低从机 SS 管脚
            SPDAT = 0x5a;               //发送测试数据
            while (!KEY);               //等待按键释放
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;              //清中断标志
```

```
            if (SPCTL & 0x10)
            {                                   //主机模式
                SS = 1;                         //拉高从机的 SS 管脚
                SPCTL = 0x40;                   //重新设置为从机待机
            }
            else
            {                                   //从机模式
                SPDAT = SPDAT;                  //将接收到的数据回传给主机
            }
            LED = !LED;                         //测试端口
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H
```

```
        SETB    SS
        SETB    LED
        SETB    KEY

        MOV     SPCTL,#40H          ;使能 SPI 从机模式进行待机
        MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
        JB      KEY,SKIP            ;等待按键触发
        MOV     SPCTL,#50H          ;使能 SPI 主机模式
        CLR     SS                  ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH          ;发送测试数据
        JNB     KEY,$               ;等待按键释放
SKIP:
        MOV     A,SPSTAT
        JNB     ACC.7,LOOP
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     A,SPCTL
        JB      ACC.4,MASTER
SLAVE:
        MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
        CPL     LED
        JMP     LOOP
MASTER:
        SETB    SS                  ;拉高从机的 SS 管脚
        MOV     SPCTL,#40H          ;重新设置为从机待机
        CPL     LED
        JMP     LOOP

        END
```

21 I²C 总线

STC8A8K64D4 系列的单片机内部集成了一个 I²C 串行总线控制器。I²C 是一种高速同步通讯总线，通讯使用 SCL（时钟线）和 SDA（数据线）两线进行同步通讯。对于 SCL 和 SDA 的端口分配，STC8A8K64D4 系列的单片机提供了切换模式，可将 SCL 和 SDA 切换到不同的 I/O 口上，以方便用户将一组 I²C 总线当作多组进行分时复用。

与标准 I²C 协议相比较，忽略了如下两种机制：

- 发送起始信号（START）后不进行仲裁
- 时钟信号（SCL）停留在低电平时不进行超时检测

STC8A8K64D4 系列的 I²C 总线提供了两种操作模式：主机模式（SCL 为输出口，发送同步时钟信号）和从机模式（SCL 为输入口，接收同步时钟信号）

STC 创新：STC 的 I²C 串行总线控制器工作在从机模式时，SDA 管脚的下降沿信号可以唤醒进入掉电模式的 MCU。（注意：由于 I²C 传输速度比较快，MCU 唤醒后第一包数据一般是不正确的）

21.1 I2C 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

I2C_S[1:0]：I²C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P2.5	P2.4
10	P7.7	P7.6
11	P3.2	P3.3

21.2 I²C 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I²C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[5:0]						0000,0000
I2CMSCR	I²C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C 从机地址寄存器	FE85H	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I²C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I²C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I²C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

21.3 I²C 主机模式

21.3.1 I2C 配置寄存器（I2CCFG），总线速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL	MSSPEED[5:0]					

ENI2C：I²C 功能使能控制位

0：禁止 I²C 功能

1：允许 I²C 功能

MSSL：I²C 工作模式选择位

0：从机模式

1：主机模式

MSSPEED[5:0]：I²C 总线速度（等待时钟数）控制，**I2C 总线速度＝F_{OSC} / 2 / (MSSPEED * 2 + 4)**

MSSPEED[5:0]	对应的时钟数
0	4
1	6
2	8
…	…
x	2x+4
…	…
62	128
63	130

只有当 I²C 模块工作在主机模式时，MSSPEED 参数设置的等待参数才有效。此等待参数主要用于主机模式的以下几个信号：

T_{SSTA}：起始信号的建立时间（Setup Time of START）

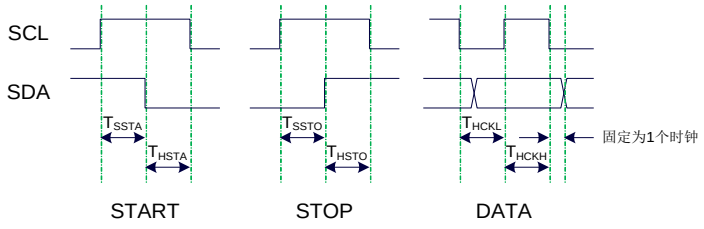
T_{HSTA}：起始信号的保持时间（Hold Time of START）

T_{SSTO}：停止信号的建立时间（Setup Time of STOP）

T_{HSTO}：停止信号的保持时间（Hold Time of STOP）

T_{HCKL}：时钟信号的低电平保持时间（Hold Time of SCL Low）

T_{HCKH}：时钟信号的高电平保持时间（Hold Time of SCL High）



例 1：当 MSSPEED＝10 时，T_{SSTA}＝T_{HSTA}＝T_{SSTO}＝T_{HSTO}＝T_{HCKL}＝24/FOSC

例 2：当 24MHz 的工作频率下需要 400K 的 I2C 总线速度时，

MSSPEED＝(24M / 400K / 2 - 4) / 2＝13

21.3.2 I2C 主机控制寄存器（I2CMSCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	MSCMD[3:0]			

EMSI：主机模式中断使能控制位

0：关闭主机模式的中断

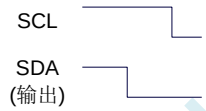
1：允许主机模式的中断

MSCMD[3:0]：主机命令

0000：待机，无动作。

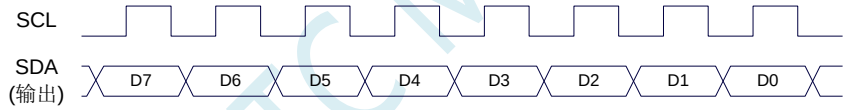
0001：起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态，即 MSBUSY（I2CMSST.7）为 0 时，写此命令会使控制器进入忙状态，硬件自动将 MSBUSY 状态位置 1，并开始发送 START 信号；**若当前 I²C 控制器处于忙状态，写此命令可触发发送 START 信号**。发送 START 信号的波形如下图所示：



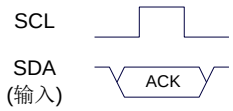
0010：发送数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上（先发送高位数据）。发送数据的波形如下图所示：



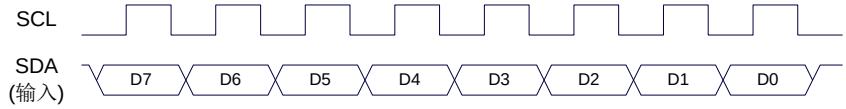
0011：接收 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将从 SDA 端口上读取的数据保存到 MSACKI（I2CMSST.1）。接收 ACK 的波形如下图所示：



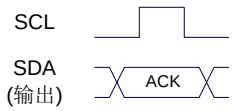
0100：接收数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器（先接收高位数据）。接收数据的波形如下图所示：



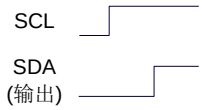
0101：发送 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将 MSACKO（I2CMSST.0）中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示：



0110：停止命令。

发送 STOP 信号。写此命令后，I²C 总线控制器开始发送 STOP 信号。信号发送完成后，硬件自动将 MSBUSY 状态位清零。STOP 信号的波形如下图所示：



0111：保留。

1000：保留。

1001：起始命令+发送数据命令+接收 ACK 命令。

此命令为命令 0001、命令 0010、命令 0011 三个命令的组合，下此命令后控制器会依次执行这三个命令。

1010：发送数据命令+接收 ACK 命令。

此命令为命令 0010、命令 0011 两个命令的组合，下此命令后控制器会依次执行这两个命令。

1011：接收数据命令+发送 ACK(0)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 ACK（0），不受 MSACKO 位的影响。

1100：接收数据命令+发送 NAK(1)命令。

此命令为命令 0100、命令 0101 两个命令的组合，下此命令后控制器会依次执行这两个命令。

注意：此命令所返回的应答信号固定为 NAK（1），不受 MSACKO 位的影响。

21.3.3 I2C 主机辅助控制寄存器（I2CMSAUX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA：主机模式时 I²C 数据自动发送允许位

0：禁止自动发送

1：使能自动发送

若自动发送功能被使能，当 MCU 执行完成对 I2CTXD 数据寄存器的写操作后，I²C 控制器会自动触发“1010”命令，即自动发送数据并接收 ACK 信号。

21.3.4 I2C 主机状态寄存器（I2CMSST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY：主机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

1：控制器处于忙碌状态

当 I²C 控制器处于主机模式时，在空闲状态下，发送完成 START 信号后，控制器便进入到忙碌状态，忙碌状态会一直维持到成功发送完成 STOP 信号，之后状态会再次恢复到空闲状态。

MSIF：主机模式的中断请求位（中断标志位）。当处于主机模式的 I²C 控制器执行完成寄存器 I2CMSCR

中 MSCMD 命令后产生中断信号，硬件自动将此位 1，向 CPU 发请求中断，响应中断后 MSIF 位必须用软件清零。

MSACKI：主机模式时，发送“0011”命令到 I2CMSCR 的 MSCMD 位后所接收到的 ACK 数据。

MSACKO：主机模式时，准备将要发送出去的 ACK 信号。当发送“0101”命令到 I2CMSCR 的 MSCMD 位后，控制器会自动读取此位的数据当作 ACK 发送到 SDA。

21.4 I²C 从机模式

21.4.1 I2C 从机控制寄存器（I2CSLCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI：从机模式时接收到 START 信号中断允许位

0：禁止从机模式时接收到 START 信号时发生中断

1：使能从机模式时接收到 START 信号时发生中断

ERXI：从机模式时接收到 1 字节数据后中断允许位

0：禁止从机模式时接收到数据后发生中断

1：使能从机模式时接收到 1 字节数据后发生中断

ETXI：从机模式时发送完成 1 字节数据后中断允许位

0：禁止从机模式时发送完成数据后发生中断

1：使能从机模式时发送完成 1 字节数据后发生中断

ESTOI：从机模式时接收到 STOP 信号中断允许位

0：禁止从机模式时接收到 STOP 信号时发生中断

1：使能从机模式时接收到 STOP 信号时发生中断

SLRST：复位从机模式

21.4.2 I2C 从机状态寄存器（I2CSLST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

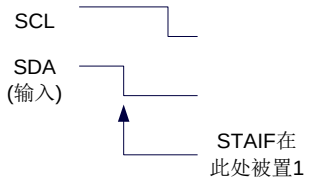
SLBUSY：从机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

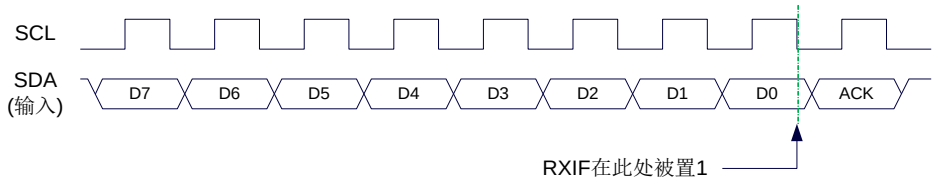
1：控制器处于忙碌状态

当 I²C 控制器处于从机模式时，在空闲状态下，接收到主机发送 START 信号后，控制器会继续检测之后的设备地址数据，若设备地址与当前 I2CSLADR 寄存器中所设置的从机地址相同时，控制器便进入到忙碌状态，忙碌状态会一直维持到成功接收到主机发送 STOP 信号，之后状态会再次恢复到空闲状态。

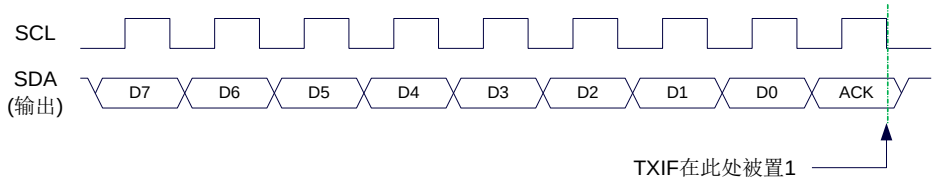
STAIF：从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示：



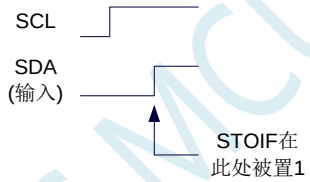
RXIF：从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后，在第 8 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示：



TXIF：从机模式时发送完成 1 字节的数据后的中断请求位。从机模式的 I²C 控制器发送完成 1 字节的数据并成功接收到 1 位 ACK 信号后，在第 9 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 TXIF 位必须用软件清零。TXIF 被置 1 的时间点如下图所示：

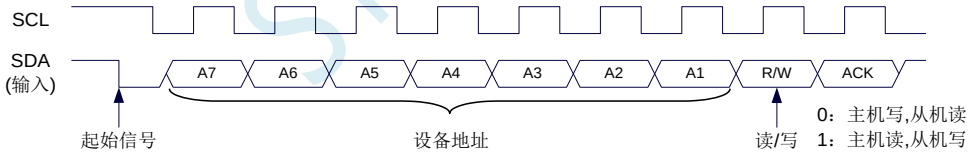


STOIF：从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：



SLACKI：从机模式时，接收到的 ACK 数据。

SLACKO：从机模式时，准备将要发送出去的 ACK 信号。



21.4.3 I2C 从机地址寄存器（I2CSLADR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H	I2CSLADR[7:1]							MA

I2CSLADR[7:1]：从机设备地址

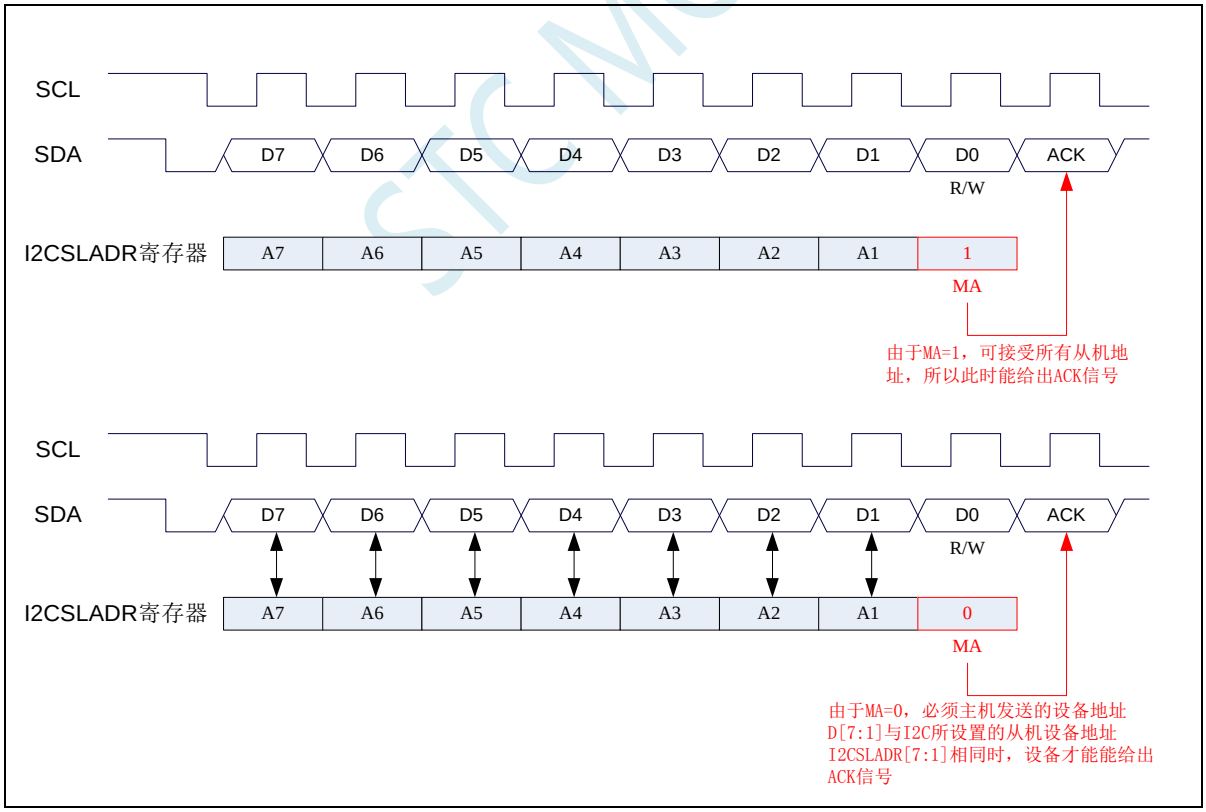
当 I²C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 I2CSLADR[7:1]中所设置的从机设备地址相同时，控制器才会向 CPU 发出中断求，请求 CPU 处理 I²C 事件；否则若设备地址不同，I²C 控制器继续监控，等待下一个起始信号，对下一个设备地址继续比较。

MA：从机设备地址比较控制

0：设备地址必须与 I2CSLADR[7:1]相同

1：忽略 I2CSLADR[7:1]中的设置，接受所有的设备地址

说明：I2C 总线协议规定 I2C 总线上最多可挂载 128 个 I2C 设备（理论值），不同的 I2C 设备用不同的 I2C 从机设备地址进行识别。I2C 主机发送完成起始信号后，发送的第一个数据（DATA0）的高 7 位即为从机设备地址（DATA0[7:1]为 I2C 设备地址），最低位为读写信号。当 I2C 设备从机地址寄存器 MA（I2CSLADR.0）为 1 时，表示 I2C 从机能够接受所有的设备地址，此时主机发送的任何设备地址，即 DATA0[7:1]为任何值，从机都能响应。当 I2C 设备从机地址寄存器 MA（I2CSLADR.0）为 0 时，主机发送的设备地址 DATA0[7:1]必须与从机的设备地址 I2CSLADR[7:1]相同时才能访问此从机设备



21.4.4 I2C 数据寄存器（I2CTXD，I2CRXD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD 是 I^2C 发送数据寄存器，存放将要发送的 I^2C 数据

I2CRXD 是 I^2C 接收数据寄存器，存放接收完成的 I^2C 数据

21.5 范例程序

21.5.1 I²C 主机模式访问 AT24C256（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

bit         busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                       //清中断标志
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81;                             //发送 START 命令
```

```
    while (busy);
}

void SendData(char dat)
{
    I2CTXD = dat;                       //写数据到数据缓冲区
    busy = 1;
    I2CMSCR = 0x82;                     //发送 SEND 命令
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;                     //发送读 ACK 命令
    while (busy);
}

char RecvData()
{
    busy = 1;
    I2CMSCR = 0x84;                     //发送 RECV 命令
    while (busy);
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                     //设置 ACK 信号
    busy = 1;
    I2CMSCR = 0x85;                     //发送 ACK 命令
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                     //设置 NAK 信号
    busy = 1;
    I2CMSCR = 0x85;                     //发送 ACK 命令
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86;                     //发送 STOP 命令
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
```

```
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;
    EA = 1;

    Start();    //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
    RecvACK();
    Stop();                                 //发送停止命令

    Delay();                                //等待设备写数据

    Start();                                //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0xa1);                         //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

BUSY        BIT     20H.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH

            MOV     DPTR,#I2CMSST       ;清中断标志
            MOVX    A,@DPTR
            ANL     A,#NOT 40H
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            CLR     BUSY                ;复位忙标志

            POP     DPH
            POP     DPL
            POP     ACC
            RETI

START:
            SETB    BUSY
```

```
                MOV     A,#10000001B            ;发送 START 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDDATA:
                MOV     DPTR,#I2CTXD            ;写数据到数据缓冲区
                MOVX    @DPTR,A
                SETB    BUSY
                MOV     A,#10000010B            ;发送 SEND 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
RECVACK:
                SETB    BUSY
                MOV     A,#10000011B            ;发送读 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
RECVDATA:
                SETB    BUSY
                MOV     A,#10000100B            ;发送 RECV 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                CALL    WAIT
                MOV     DPTR,#I2CRXD            ;从数据缓冲区读取数据
                MOVX    A,@DPTR
                RET
SENDACK:
                MOV     A,#00000000B            ;设置 ACK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                SETB    BUSY
                MOV     A,#10000101B            ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDNAK:
                MOV     A,#00000001B            ;设置 NAK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                SETB    BUSY
                MOV     A,#10000101B            ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                SETB    BUSY
                MOV     A,#10000110B            ;发送 STOP 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                JB      BUSY,$                  ;等待命令发送完成
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
```

```
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B            ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    EA

        CALL    START                   ;发送起始命令
        MOV     A,#0A0H
        CALL    SENDDATA                ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#000H                 ;发送存储地址高字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H                 ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H                  ;写测试数据 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H                  ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                    ;发送停止命令

        CALL    DELAY                   ;等待设备写数据

        CALL    START                   ;发送起始命令
        MOV     A,#0A0H                 ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H                 ;发送存储地址高字节
        CALL    SENDDATA
```

```
        CALL    RECVACK
        MOV     A,#000H                 ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START                   ;发送起始命令
        MOV     A,#0A1H                 ;发送设备地址+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA                ;读取数据 1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA                ;读取数据 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                    ;发送停止命令

        JMP     $

        END
```

21.5.2 I²C 主机模式访问 AT24C256（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;
```

```
void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                         //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                           //写数据到数据缓冲区
    I2CMSCR = 0x02;                         //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                         //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                         //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //设置 ACK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
```

```
            _nop_();
            _nop_();
            _nop_();
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        P_SW2 = 0x80;

        I2CCFG = 0xe0;                          //使能 I2C 主机模式
        I2CMSST = 0x00;

        Start();                                //发送起始命令
        SendData(0xa0);                         //发送设备地址+写命令
        RecvACK();
        SendData(0x00);                         //发送存储地址高字节
        RecvACK();
        SendData(0x00);                         //发送存储地址低字节
        RecvACK();
        SendData(0x12);                         //写测试数据 1
        RecvACK();
        SendData(0x78);                         //写测试数据 2
        RecvACK();
        Stop();                                 //发送停止命令

        Delay();                                //等待设备写数据

        Start();                                //发送起始命令
        SendData(0xa0);                         //发送设备地址+写命令
        RecvACK();
        SendData(0x00);                         //发送存储地址高字节
        RecvACK();
        SendData(0x00);                         //发送存储地址低字节
        RecvACK();
        Start();                                //发送起始命令
        SendData(0xa1);                         //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();                        //读取数据 1
        SendACK();
        P2 = RecvData();                        //读取数据 2
        SendNAK();
        Stop();                                 //发送停止命令

        P_SW2 = 0x00;
```

```
    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
```

```
            CALL    WAIT
            MOV     DPTR,#I2CRXD            ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B            ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B            ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B            ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B            ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B            ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            MOV     DPTR,#I2CMSST           ;清中断标志
            MOVX    A,@DPTR
            JNB     ACC.6,WAIT
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            RET

DELAY:
            MOV     R0,#0
            MOV     R1,#0
DELAY1:
            NOP
            NOP
            NOP
            NOP
            DJNZ    R1,DELAY1
            DJNZ    R0,DELAY1
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H
```

```
MOV     P_SW2,#80H

MOV     A,#11100000B        ;设置 I2C 模块为主机模式
MOV     DPTR,#I2CCFG
MOVX    @DPTR,A
MOV     A,#00000000B
MOV     DPTR,#I2CMSST
MOVX    @DPTR,A

CALL    START               ;发送起始命令
MOV     A,#0A0H
CALL    SENDDATA            ;发送设备地址+写命令
CALL    RECVACK
MOV     A,#000H             ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#12H              ;写测试数据 1
CALL    SENDDATA
CALL    RECVACK
MOV     A,#78H              ;写测试数据 2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP                ;发送停止命令

CALL    DELAY               ;等待设备写数据

CALL    START               ;发送起始命令
MOV     A,#0A0H             ;发送设备地址+写命令
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址高字节
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;发送存储地址低字节
CALL    SENDDATA
CALL    RECVACK
CALL    START               ;发送起始命令
MOV     A,#0A1H             ;发送设备地址+读命令
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA            ;读取数据 1
MOV     P0,A
CALL    SENDACK
CALL    RECVDATA            ;读取数据 2
MOV     P2,A
CALL    SENDNAK
CALL    STOP                ;发送停止命令

JMP     $

END
```

21.5.3 I²C 主机模式访问 PCF8563

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                             //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                               //写数据到数据缓冲区
    I2CMSCR = 0x02;                             //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                             //发送读 ACK 命令
```

```
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                         //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //设置 ACK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
```

```
    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0xa2);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x02);                         //发送存储地址
    RecvACK();
    SendData(0x00);                         //设置秒值
    RecvACK();
    SendData(0x00);                         //设置分钟值
    RecvACK();
    SendData(0x12);                         //设置小时值
    RecvACK();
    Stop();                                 //发送停止命令

    while (1)
    {
        Start();                            //发送起始命令
        SendData(0xa2);                     //发送设备地址+写命令
        RecvACK();
        SendData(0x02);                     //发送存储地址
        RecvACK();
        Start();                            //发送起始命令
        SendData(0xa3);                     //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();                    //读取秒值
        SendACK();
        P2 = RecvData();                    //读取分钟值
        SendACK();
        P3 = RecvData();                    //读取小时值
        SendNAK();
        Stop();                             //发送停止命令

        Delay();
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
```

```
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            CALL    WAIT
            MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B        ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
```

```
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST           ;清中断标志
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
                NOP
                DJNZ    R1,DELAY1
                DJNZ    R0,DELAY1
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     P_SW2,#80H

                MOV     A,#11100000B            ;设置 I2C 模块为主机模式
                MOV     DPTR,#I2CCFG
                MOVX    @DPTR,A
                MOV     A,#00000000B
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A

                CALL    START                   ;发送起始命令
                MOV     A,#0A2H
                CALL    SENDDATA                ;发送设备地址+写命令
                CALL    RECVACK
                MOV     A,#002H                 ;发送存储地址
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#00H                  ;设置秒值
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#00H                  ;设置分钟值
                CALL    SENDDATA
                CALL    RECVACK
```

```
                MOV     A,#12H                  ;设置小时值
                CALL    SENDDATA
                CALL    RECVACK
                CALL    STOP                    ;发送停止命令
LOOP:
                CALL    START                   ;发送起始命令
                MOV     A,#0A2H                 ;发送设备地址+写命令
                CALL    SENDDATA
                CALL    RECVACK
                MOV     A,#002H                 ;发送存储地址
                CALL    SENDDATA
                CALL    RECVACK
                CALL    START                   ;发送起始命令
                MOV     A,#0A3H                 ;发送设备地址+读命令
                CALL    SENDDATA
                CALL    RECVACK
                CALL    RECVDATA                ;读取秒值
                MOV     P0,A
                CALL    SENDACK
                CALL    RECVDATA                ;读取分钟值
                MOV     P2,A
                CALL    SENDACK
                CALL    RECVDATA                ;读取小时值
                MOV     P3,A
                CALL    SENDNAK
                CALL    STOP                    ;发送停止命令

                CALL    DELAY

                JMP     LOOP

                END
```

21.5.4 I²C 从机模式（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
```

```
sfr         P0M0        =   0x94;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SDA         =   P1^4;
sbit        SCL         =   P1^5;

bit         isda;                               //设备地址标志
bit         isma;                               //存储地址标志
unsigned char           addr;
unsigned char xdata     buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                       //处理 START 事件
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                       //处理 RECV 事件
        if (isda)
        {
            isda = 0;                           //处理 RECV 事件（RECV DEVICE ADDR）
        }
        else if (isma)
        {
            isma = 0;                           //处理 RECV 事件（RECV MEMORY ADDR）
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;            //处理 RECV 事件（RECV DATA）
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                       //处理 SEND 事件
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                      //接收到 NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr];            //接收到 ACK 则继续读取数据
        }
    }
    else if (I2CSLST & 0x08)
    {
```

```
            I2CSLST &= ~0x08;                       //处理 STOP 事件
            isda = 1;
            isma = 1;
        }

        _pop_(P_SW2);
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        P_SW2 = 0x80;

        I2CCFG = 0x81;                              //使能 I2C 从机模式
        I2CSLADR = 0x5a;                            //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                                    //即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                                    //由于 MA 为 0,主机发送的的设备地址必须与
                                                    //I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                                    //主机若需要写数据则要发送 5AH(0101_1010B)
                                                    //主机若需要读数据则要发送 5BH(0101_1011B)
        I2CSLST = 0x00;
        I2CSLCR = 0x78;                             //使能从机模式中断
        EA = 1;

        isda = 1;                                   //用户变量初始化
        isma = 1;
        addr = 0;
        I2CTXD = buffer[addr];

        while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H
```

```
SDA             BIT             P1.4
SCL             BIT             P1.5
ISDA            BIT             20H.0           ;设备地址标志
ISMA            BIT             20H.1           ;存储地址标志

ADDR            DATA            21H

P1M1            DATA            091H
P1M0            DATA            092H
P0M1            DATA            093H
P0M0            DATA            094H
P2M1            DATA            095H
P2M0            DATA            096H
P3M1            DATA            0B1H
P3M0            DATA            0B2H
P4M1            DATA            0B3H
P4M0            DATA            0B4H
P5M1            DATA            0C9H
P5M0            DATA            0CAH

                ORG             0000H
                LJMP            MAIN
                ORG             00C3H
                LJMP            I2CISR

                ORG             0100H
I2CISR:
                PUSH            ACC
                PUSH            PSW
                PUSH            DPL
                PUSH            DPH
                MOV             DPTR,#I2CSLST   ;检测从机状态
                MOVX            A,@DPTR
                JB              ACC.6,STARTIF
                JB              ACC.5,RXIF
                JB              ACC.4,TXIF
                JB              ACC.3,STOPIF
ISREXIT:
                POP             DPH
                POP             DPL
                POP             PSW
                POP             ACC
                RETI
STARTIF:
                ANL             A,#NOT 40H      ;处理 START 事件
                MOVX            @DPTR,A
                JMP             ISREXIT
RXIF:
                ANL             A,#NOT 20H      ;处理 RECV 事件
                MOVX            @DPTR,A
                MOV             DPTR,#I2CRXD
                MOVX            A,@DPTR
                JBC             ISDA,RXDA
                JBC             ISMA,RXMA
                MOV             R0,ADDR         ;处理 RECV 事件（RECV DATA）
                MOVX            @R0,A
                INC             ADDR
                JMP             ISREXIT
RXDA:
```

```
                JMP     ISREXIT         ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
                MOV     ADDR,A          ;处理 RECV 事件（RECV MEMORY ADDR）
                MOV     R0,A
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
TXIF:
                ANL     A,#NOT 10H      ;处理 SEND 事件
                MOVX    @DPTR,A
                JB      ACC.1,RXNAK
                INC     ADDR
                MOV     R0,ADDR
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
RXNAK:
                MOVX    A,#0FFH
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
STOPIF:
                ANL     A,#NOT 08H      ;处理 STOP 事件
                MOVX    @DPTR,A
                SETB    ISDA
                SETB    ISMA
                JMP     ISREXIT

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     P_SW2,#80H

                MOV     A,#10000001B    ;使能 I2C 从机模式
                MOV     DPTR,#I2CCFG
                MOVX    @DPTR,A
                MOV     A,#01011010B    ;设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                        ;即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                        ;由于 MA 为 0,主机发送的的设备地址必须与
                                        ;I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                        ;主机若需要写数据则要发送 5AH(0101_1010B)
                                        ;主机若需要读数据则要发送 5BH(0101_1011B)
                MOV     DPTR,#I2CSLADR
                MOVX    @DPTR,A
                MOV     A,#00000000B
```

```
        MOV     DPTR,#I2CSLST
        MOVX    @DPTR,A
        MOV     A,#01111000B            ;使能从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                    ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

        SETB    EA

        SJMP    $

        END
```

21.5.5 I²C 从机模式（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;
```

```
bit             isda;                                   //设备地址标志
bit             isma;                                   //存储地址标志
unsigned char               addr;
unsigned char xdata         buffer[256];

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                                      //使能 I2C 从机模式
    I2CSLADR = 0x5a;                                    //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                                        //即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                                        //由于 MA 为 0,主机发送的的设备地址必须与
                                                        //I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                                        //主机若需要写数据则要发送 5AH(0101_1010B)
                                                        //主机若需要读数据则要发送 5BH(0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x00;                                     //禁止从机模式中断

    isda = 1;                                           //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40;                           //处理 START 事件
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;                           //处理 RECV 事件
            if (isda)
            {
                isda = 0;                               //处理 RECV 事件（RECV DEVICE ADDR）
            }
            else if (isma)
            {
                isma = 0;                               //处理 RECV 事件（RECV MEMORY ADDR）
                addr = I2CRXD;
                I2CTXD = buffer[addr];
            }
            else
            {
```

```
                buffer[addr++] = I2CRXD;                //处理 RECV 事件（RECV DATA）
            }
        }
        else if (I2CSLST & 0x10)
        {
            I2CSLST &= ~0x10;                           //处理 SEND 事件
            if (I2CSLST & 0x02)
            {
                I2CTXD = 0xff;                          //接收到 NAK 则停止读取数据
            }
            else
            {
                I2CTXD = buffer[++addr];                //接收到 ACK 则继续读取数据
            }
        }
        else if (I2CSLST & 0x08)
        {
            I2CSLST &= ~0x08;                           //处理 STOP 事件
            isda = 1;
            isma = 1;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5
ISDA        BIT     20H.0       ;设备地址标志
ISMA        BIT     20H.1       ;存储地址标志

ADDR        DATA    21H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
```

```
        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#10000001B            ;使能 I2C 从机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#01011010B            ;设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                        ;即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                        ;由于 MA 为 0,主机发送的的设备地址必须与
                                        ;I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                        ;主机若需要写数据则要发送 5AH(0101_1010B)
                                        ;主机若需要读数据则要发送 5BH(0101_1011B)
        MOV     DPTR,#I2CSLADR
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CSLST
        MOVX    @DPTR,A
        MOV     A,#00000000B            ;禁止从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                    ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

LOOP:
        MOV     DPTR,#I2CSLST           ;检测从机状态
        MOVX    A,@DPTR
        JB      ACC.6,STARTIF
        JB      ACC.5,RXIF
        JB      ACC.4,TXIF
        JB      ACC.3,STOPIF
        JMP     LOOP
STARTIF:
        ANL     A,#NOT 40H              ;处理 START 事件
        MOVX    @DPTR,A
```

```
            JMP     LOOP
RXIF:
            ANL     A,#NOT 20H          ;处理 RECV 事件
            MOVX    @DPTR,A
            MOV     DPTR,#I2CRXD
            MOVX    A,@DPTR
            JBC     ISDA,RXDA
            JBC     ISMA,RXMA
            MOV     R0,ADDR             ;处理 RECV 事件（RECV DATA）
            MOVX    @R0,A
            INC     ADDR
            JMP     LOOP
RXDA:
            JMP     LOOP                ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
            MOV     ADDR,A              ;处理 RECV 事件（RECV MEMORY ADDR）
            MOV     R0,A
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
TXIF:
            ANL     A,#NOT 10H          ;处理 SEND 事件
            MOVX    @DPTR,A
            JB      ACC.1,RXNAK
            INC     ADDR
            MOV     R0,ADDR
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
RXNAK:
            MOVX    A,#0FFH
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
STOPIF:
            ANL     A,#NOT 08H          ;处理 STOP 事件
            MOVX    @DPTR,A
            SETB    ISDA
            SETB    ISMA
            JMP     LOOP

            END
```

21.5.6 测试 I²C 从机模式代码的主机代码

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;
```