

18.3.4.3 6 位 PWM 模式

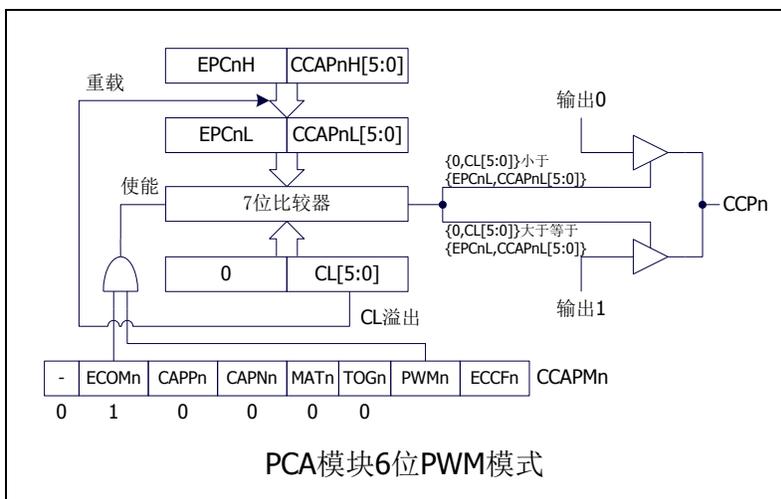
PCA_PWMn 寄存器中的 EBSn[1:0] 设置为 10 时, PCA 模块 n 工作于 6 位 PWM 模式, 此时将 {0, CL[5:0]} 与捕获寄存器 {EPCnL, CCAPnL[5:0]} 进行比较。当 PCA 模块工作于 6 位 PWM 模式时, 由于所有模块共用一个 PCA 计数器, 所有它们的输出频率相同。各个模块的输出占空比使用寄存器 {EPCnL, CCAPnL[5:0]} 进行设置。当 {0, CL[5:0]} 的值小于 {EPCnL, CCAPnL[5:0]} 时, 输出为低电平; 当 {0, CL[5:0]} 的值等于或大于 {EPCnL, CCAPnL[5:0]} 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。

PCA时钟输入源频率

8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:



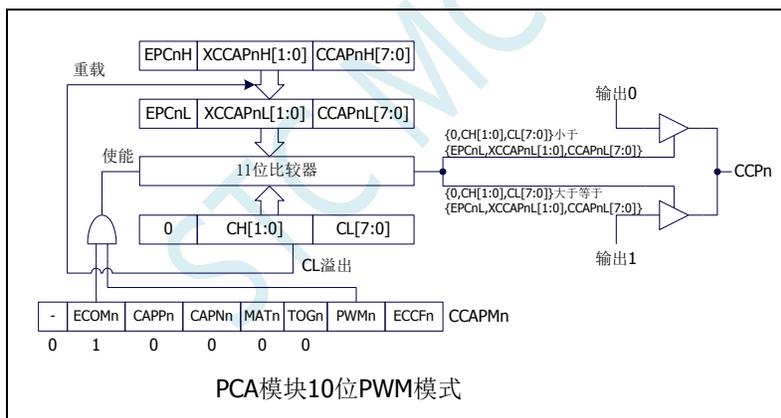
18.3.4.4 10 位 PWM 模式

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

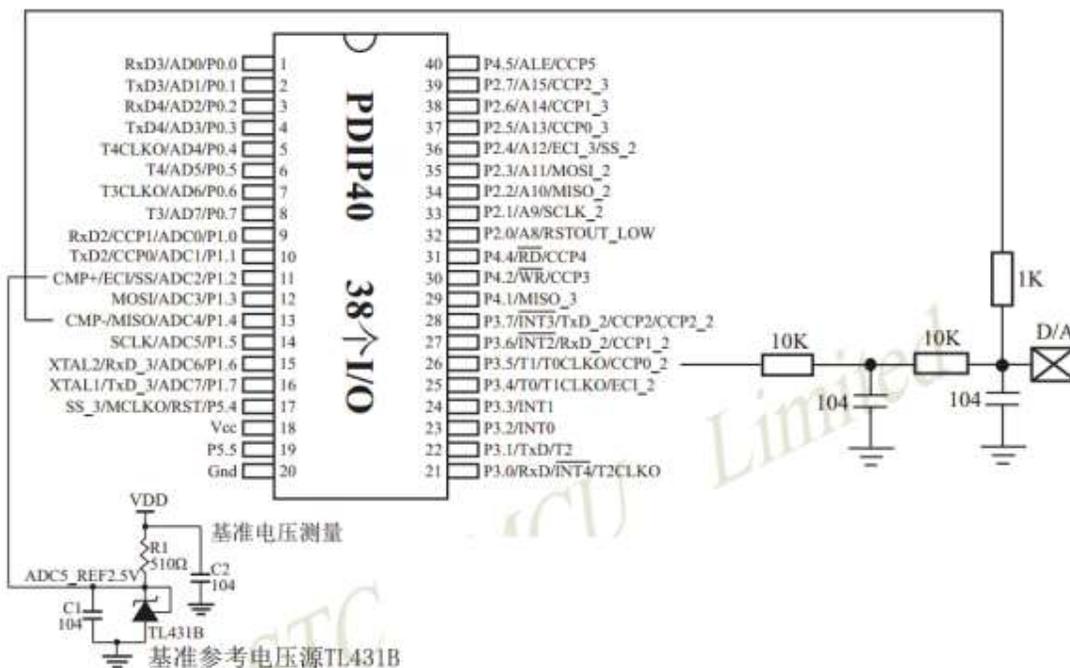


18.3.4.5 如何控制 PWM 固定输出高电平/低电平

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

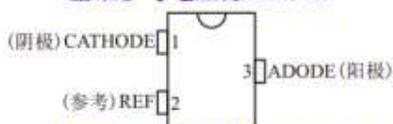


如应用简单, 可无需基准参考电压源, 直接与Vcc比较即可。

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) 如果要输出高精度电压, 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;

unsigned long count0;

unsigned long count1;

unsigned long length;

//存储 PCA 计时溢出次数

//记录上一次的捕获值

//记录本次的捕获值

//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

<i>CCF0</i>	<i>BIT</i>	<i>CCON.0</i>	
<i>CMOD</i>	<i>DATA</i>	<i>0D9H</i>	
<i>CL</i>	<i>DATA</i>	<i>0E9H</i>	
<i>CH</i>	<i>DATA</i>	<i>0F9H</i>	
<i>CCAPM0</i>	<i>DATA</i>	<i>0DAH</i>	
<i>CCAP0L</i>	<i>DATA</i>	<i>0EAH</i>	
<i>CCAP0H</i>	<i>DATA</i>	<i>0FAH</i>	
<i>PCA_PWM0</i>	<i>DATA</i>	<i>0F2H</i>	
<i>CCAPM1</i>	<i>DATA</i>	<i>0DBH</i>	
<i>CCAP1L</i>	<i>DATA</i>	<i>0EBH</i>	
<i>CCAP1H</i>	<i>DATA</i>	<i>0FBH</i>	
<i>PCA_PWM1</i>	<i>DATA</i>	<i>0F3H</i>	
<i>CCAPM2</i>	<i>DATA</i>	<i>0DCH</i>	
<i>CCAP2L</i>	<i>DATA</i>	<i>0ECH</i>	
<i>CCAP2H</i>	<i>DATA</i>	<i>0FCH</i>	
<i>PCA_PWM2</i>	<i>DATA</i>	<i>0F4H</i>	
<i>CNT</i>	<i>DATA</i>	<i>20H</i>	
<i>COUNT0</i>	<i>DATA</i>	<i>21H</i>	;3 bytes
<i>COUNT1</i>	<i>DATA</i>	<i>24H</i>	;3 bytes
<i>LENGTH</i>	<i>DATA</i>	<i>27H</i>	;3 bytes, (COUNT1-COUNT0)
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>003BH</i>	
	<i>LJMP</i>	<i>PCAIRS</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>PCAIRS:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>JNB</i>	<i>CF,CHECKCCF0</i>	
	<i>CLR</i>	<i>CF</i>	;清中断标志
	<i>INC</i>	<i>CNT</i>	;PCA 计时溢出次数+1
<i>CHECKCCF0:</i>			
	<i>JNB</i>	<i>CCF0,ISREXIT</i>	
	<i>CLR</i>	<i>CCF0</i>	
	<i>MOV</i>	<i>COUNT0,COUNT1</i>	;备份上一次的捕获值
	<i>MOV</i>	<i>COUNT0+1,COUNT1+1</i>	
	<i>MOV</i>	<i>COUNT0+2,COUNT1+2</i>	
	<i>MOV</i>	<i>COUNT1,CNT</i>	;保存本次的捕获值
	<i>MOV</i>	<i>COUNT1+1,CCAP0H</i>	
	<i>MOV</i>	<i>COUNT1+2,CCAP0L</i>	
	<i>CLR</i>	<i>C</i>	;计算两次的捕获差值
	<i>MOV</i>	<i>A,COUNT1+2</i>	
	<i>SUBB</i>	<i>A,COUNT0+2</i>	

```

MOV     LENGTH+2,A
MOV     A,COUNTI+I
SUBB    A,COUNT0+I
MOV     LENGTH+1,A
MOV     A,COUNTI
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     COUNTI,A
MOV     COUNTI+1,A
MOV     COUNTI+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

PCAISR:  ORG      0100H

        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

PCAISR:
                ORG    0100H
                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     CCAPMI   = 0xdb;
sfr     CCAPIL   = 0xeb;
sfr     CCAP1H   = 0xfb;
sfr     PCA_PWM1 = 0xf3;
sfr     CCAPM2   = 0xdc;
sfr     CCAP2L   = 0xec;
sfr     CCAP2H   = 0xfc;
sfr     PCA_PWM2 = 0xf4;

sfr     P0MI     = 0x93;
sfr     P0M0     = 0x94;
sfr     P1MI     = 0x91;
sfr     P1M0     = 0x92;
sfr     P2MI     = 0x95;
sfr     P2M0     = 0x96;
sfr     P3MI     = 0xb1;
sfr     P3M0     = 0xb2;
sfr     P4MI     = 0xb3;
sfr     P4M0     = 0xb4;
sfr     P5MI     = 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

PCAISR:   ORG          0100H

          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

ENPWMTA: 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)	$\text{PWM输出频率} = \frac{\text{系统工作频率SYSclk}}{(\text{PWM_PS} + 1) \times ([\text{PWMCH}, \text{PWMCL}] + 1)}$
SELT2=1 (定时器2的溢出脉冲)	$\text{PWM输出频率} = \frac{\text{定时器2的溢出脉冲频率}}{([\text{PWMCH}, \text{PWMCL}] + 1)}$

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
PWMCM = 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
MOVX    @DPTR, A               ;设置PWM 周期为1000H 个PWM 时钟
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
MOVX    @DPTR, A               ;使能PWM0 输出
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;

```

```

P5MI = 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; //PWM 时钟为系统时钟
PWMCKS = 0x00; //设置PWM 周期为0800H 个PWM 时钟
PWMC = 0x0800; //PWM0 在计数值为100H 地方输出低电平
PWM0T1= 0x0100; //PWM0 在计数值为700H 地方输出高电平
PWM0T2= 0x0700; //PWM1 在计数值为0080H 地方输出高电平
PWMIT2= 0x0080; //PWM1 在计数值为0780H 地方输出低电平
PWMIT1= 0x0780; //使能PWM0 输出
PWM0CR= 0x80; //使能PWM1 输出
PWMICR= 0x80;
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
PWMIT1H	EQU	0FF18H
PWMIT1L	EQU	0FF19H
PWMIT2H	EQU	0FF1AH
PWMIT2L	EQU	0FF1BH
PWMICR	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

MAIN:
          ORG         0100H

          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

<i>CYCLE</i>	<i>EQU</i>	<i>1000H</i>
<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>PWMSET</i>	<i>DATA</i>	<i>0F1H</i>
<i>PWMCFG</i>	<i>DATA</i>	<i>0F6H</i>
<i>PWMCH</i>	<i>EQU</i>	<i>0FF00H</i>
<i>PWMCL</i>	<i>EQU</i>	<i>0FF01H</i>
<i>PWMCKS</i>	<i>EQU</i>	<i>0FF02H</i>
<i>PWMTADCH</i>	<i>EQU</i>	<i>0FF03H</i>
<i>PWMTADCL</i>	<i>EQU</i>	<i>0FF04H</i>
<i>PWMIF</i>	<i>EQU</i>	<i>0FF05H</i>
<i>PWMFDCR</i>	<i>EQU</i>	<i>0FF06H</i>
<i>PWM0T1H</i>	<i>EQU</i>	<i>0FF10H</i>
<i>PWM0T1L</i>	<i>EQU</i>	<i>0FF11H</i>
<i>PWM0T2H</i>	<i>EQU</i>	<i>0FF12H</i>
<i>PWM0T2L</i>	<i>EQU</i>	<i>0FF13H</i>
<i>PWM0CR</i>	<i>EQU</i>	<i>0FF14H</i>
<i>PWM0HLD</i>	<i>EQU</i>	<i>0FF15H</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>
<i>DIR</i>	<i>BIT</i>	<i>20H.0</i>
<i>VALL</i>	<i>DATA</i>	<i>21H</i>
<i>VALH</i>	<i>DATA</i>	<i>22H</i>
	<i>ORG</i>	<i>0000H</i>
	<i>LJMP</i>	<i>MAIN</i>
	<i>ORG</i>	<i>00B3H</i>
	<i>LJMP</i>	<i>PWM0ISR</i>
	<i>ORG</i>	<i>0100H</i>
<i>PWMISR:</i>	<i>PUSH</i>	<i>ACC</i>
	<i>PUSH</i>	<i>PSW</i>
	<i>PUSH</i>	<i>DPL</i>
	<i>PUSH</i>	<i>DPH</i>
	<i>PUSH</i>	<i>P_SW2</i>
	<i>MOV</i>	<i>P_SW2,#80H</i>
	<i>MOV</i>	<i>A,PWMCFG</i>
	<i>JNB</i>	<i>ACC.3,JSREXIT</i>

```

        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 PWM0TI     (*(unsigned int volatile xdata *)0xFF10)
#define PWM0TIH    (*(unsigned char volatile xdata *)0xFF10)
#define PWM0TIL    (*(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
    PWMCM = 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

ADCISR:  ORG    0100H
        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
MOVX    @DPTR,A           ;设置PWM 周期为1000H 个PWM 时钟
MOV      A,#00H
MOV      DPTR,#PWMCL
MOVX    @DPTR,A
MOV      A,#01H
MOV      DPTR,#PWM0TIH    ;在计数值为100H 地方PWM0 通道输出低电平
MOVX    @DPTR,A
MOV      A,#00H
MOV      DPTR,#PWM0TIL
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）

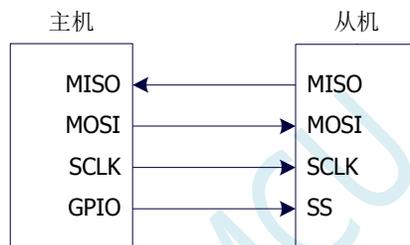
20.3.1 单主单从

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

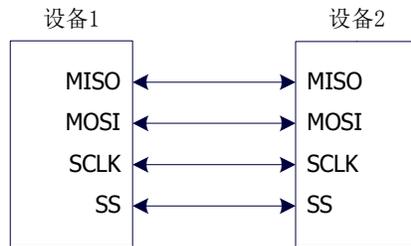
20.3.2 互为主从

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

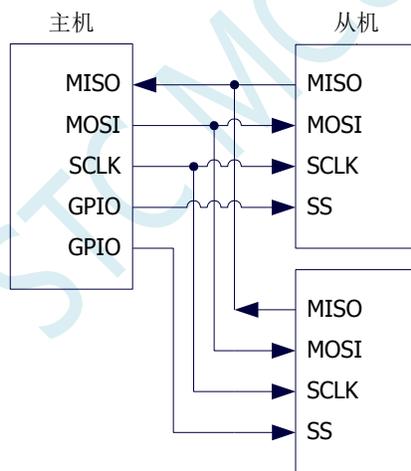
20.3.3 单主多从

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。

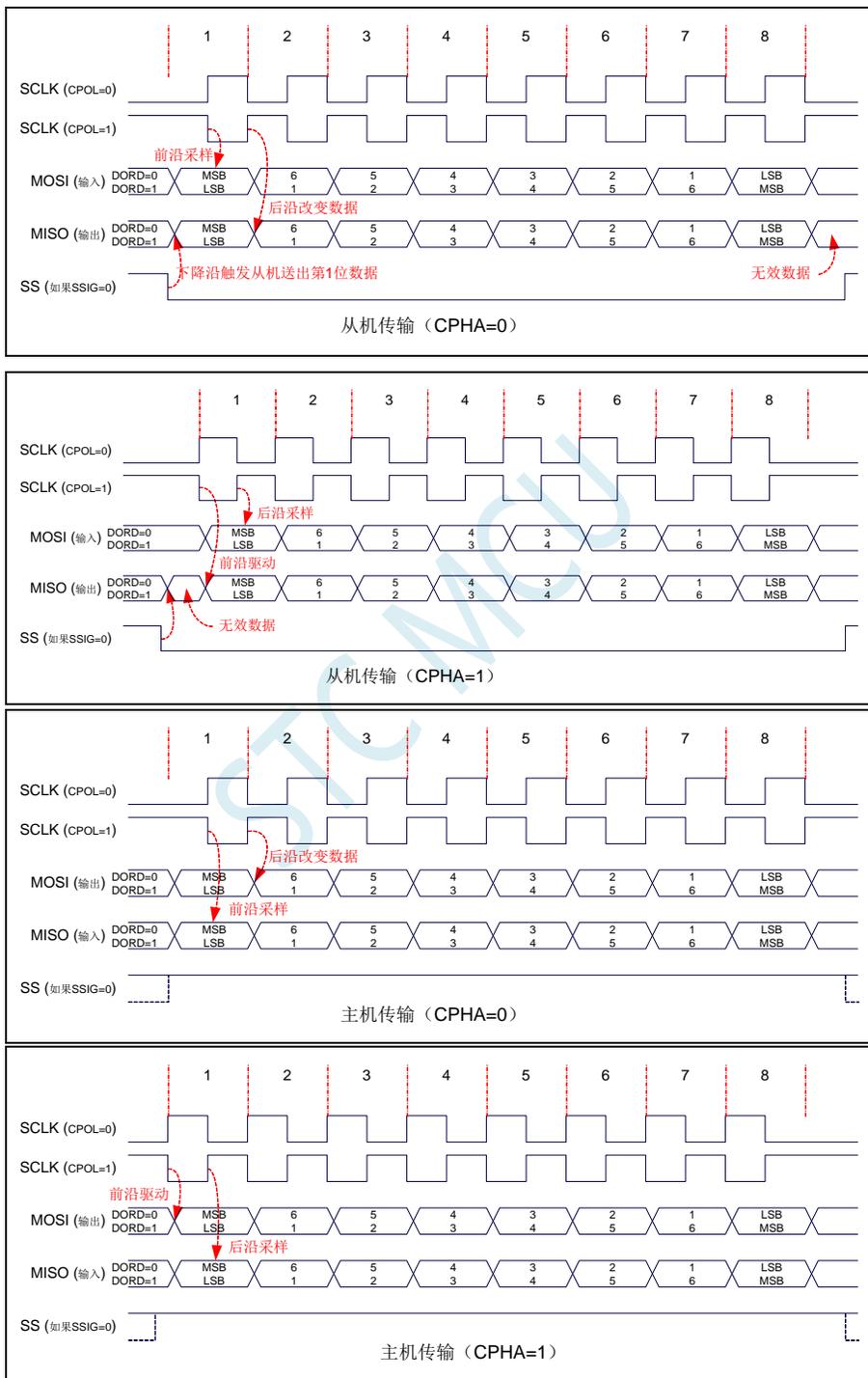
接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下一个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

WCOL 可通过软件向其写入“1”清零。

STC MCU

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。



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

MAIN:     ORG       0100H

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     SP,SPSTAT,#0C0H      ;清中断标志
    MOV     A,SPCTL
    JB     ACC.4,MASTER

SLAVE:
    MOV     SP,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)
        {
            SS = 1;           //主机模式
            SPCTL = 0x40;    //拉高从机的SS 管脚
                            //重新设置为从机待机
        }
        else
        {
            SPDAT = SPDAT;   //从机模式
                            //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	I ² C 配置寄存器	FE80H	ENI2C	MSSL	MSSPEED[5:0]						0000,0000
I2CMSCR	I ² C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I ² C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I ² C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I ² C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I ² C 从机地址寄存器	FE85H	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I ² C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I ² C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I ² C 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)

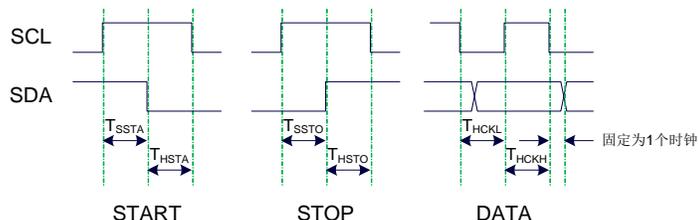
T_{HSTA}: 起始信号的保持时间 (Hold Time of START)

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, $T_{SSTA} = T_{HSTA} = T_{SSTO} = T_{HSTO} = T_{HCKL} = 24/F_{OSC}$

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

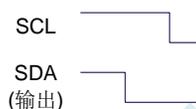
1: 允许主机模式的中断

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:



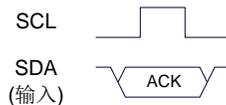
0010: 发送数据命令。

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:



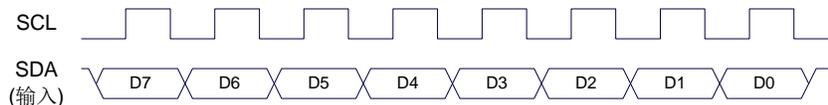
0011: 接收 ACK 命令。

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:



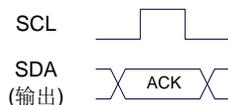
0100: 接收数据命令。

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:



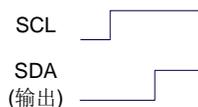
0101: 发送 ACK 命令。

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。

STC MCU

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)

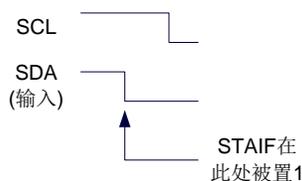
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

SLBUSY: 从机模式时 I²C 控制器状态位 (只读位)

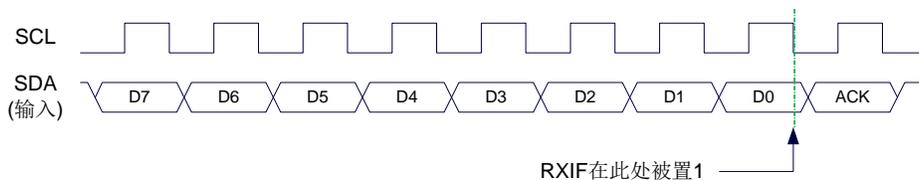
- 0: 控制器处于空闲状态
- 1: 控制器处于忙碌状态

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。

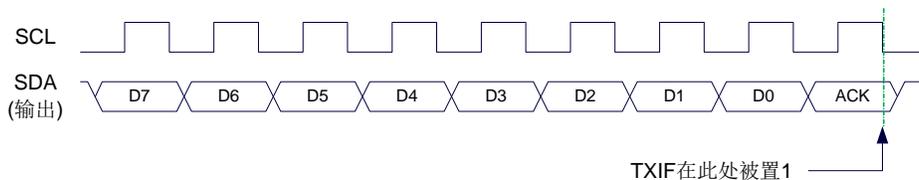
STAIF: 从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后, 硬件会自动将此位置 1, 并向 CPU 发请求中断, 响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示:



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:

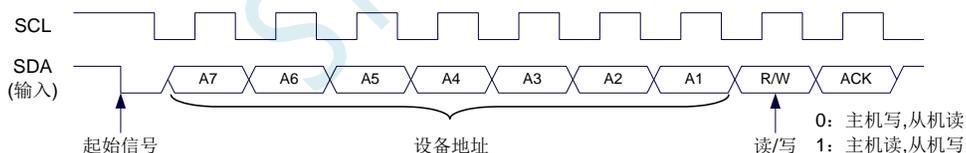


STOIF: 从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后, 硬件会自动将此位置 1, 并向 CPU 发请求中断, 响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示:



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

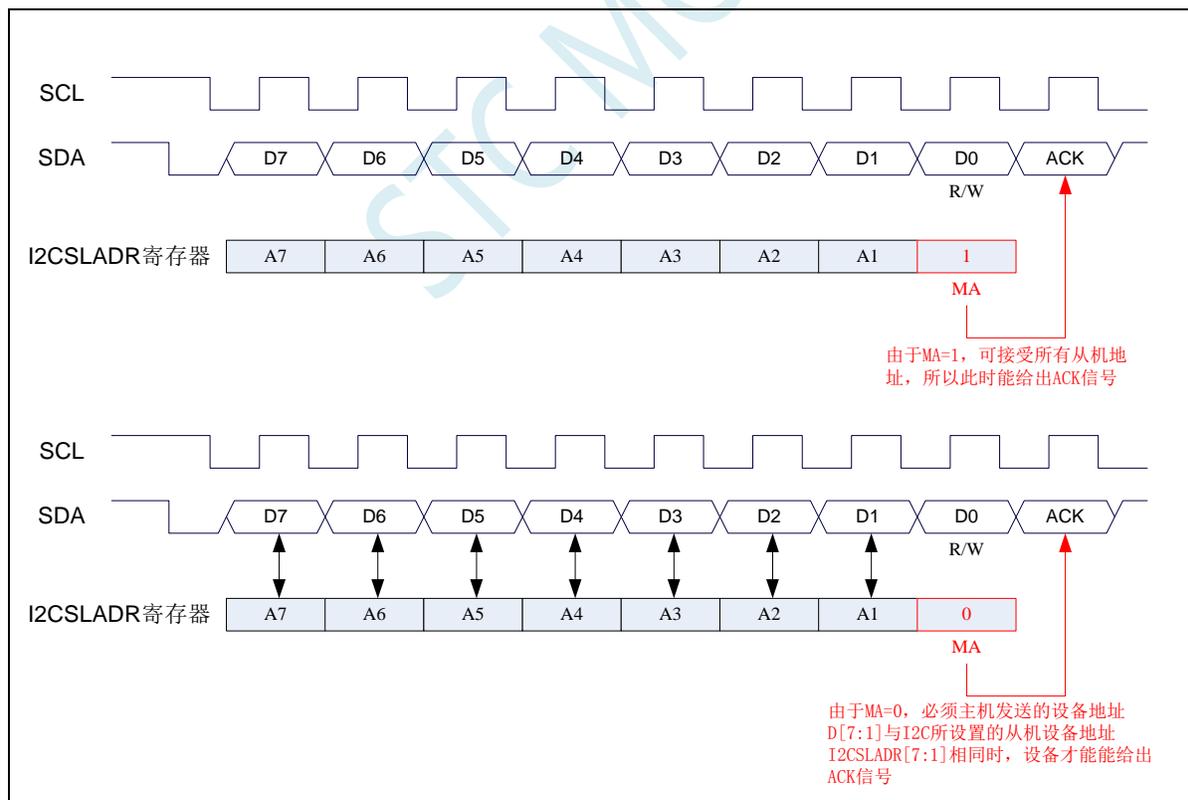
当 I²C 控制器处于从机模式时, 控制器在接收到 START 信号后, 会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 I2CSLADR[7:1]中所设置的从机设备地址相同时, 控制器才会向 CPU 发出中断求, 请求 CPU 处理 I²C 事件; 否则若设备地址不同, I²C 控制器继续监控, 等待下一个起始信号, 对下一个设备地址继续比较。

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 I²C 发送数据寄存器，存放将要发送的 I²C 数据

I2CRXD 是 I²C 接收数据寄存器，存放接收完成的 I²C 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

DELAYI:

```

```

NOP
NOP
NOP
NOP
DJNZ    RI,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,#0000000B      ;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,#1110000B      ;设置 I2C 模块为主机模式
MOV      DPTR,#I2CCFG
MOVX     @DPTR,A
MOV      A,#0000000B
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 I2CSLST     (*(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

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>
<i>I2CCFG</i>	<i>XDATA</i>	<i>0FE80H</i>
<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>
<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>

```

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

DELAYI:
NOP
NOP
NOP
NOP
DJNZ   R1,DELAYI
DJNZ   R0,DELAYI
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      PIMI       = 0x91;
sfr      PIM0       = 0x92;
sfr      P0MI       = 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

<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>	
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>	
<i>ISDA</i>	<i>BIT</i>	<i>20H.0</i>	;设备地址标志
<i>ISMA</i>	<i>BIT</i>	<i>20H.1</i>	;存储地址标志
<i>ADDR</i>	<i>DATA</i>	<i>21H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>00C3H</i>	
	<i>LJMP</i>	<i>I2CISR</i>	
<i>I2CISR:</i>	<i>ORG</i>	<i>0100H</i>	
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>PUSH</i>	<i>DPL</i>	
	<i>PUSH</i>	<i>DPH</i>	
	<i>MOV</i>	<i>DPTR,#I2CSLST</i>	;检测从机状态
	<i>MOVX</i>	<i>A,@DPTR</i>	
	<i>JB</i>	<i>ACC.6,STARTIF</i>	
	<i>JB</i>	<i>ACC.5,RXIF</i>	
	<i>JB</i>	<i>ACC.4,TXIF</i>	
	<i>JB</i>	<i>ACC.3,STOPIF</i>	
<i>ISREXIT:</i>	<i>POP</i>	<i>DPH</i>	
	<i>POP</i>	<i>DPL</i>	
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>STARTIF:</i>	<i>ANL</i>	<i>A,#NOT 40H</i>	;处理 START 事件
	<i>MOVX</i>	<i>@DPTR,A</i>	
	<i>JMP</i>	<i>ISREXIT</i>	
<i>RXIF:</i>	<i>ANL</i>	<i>A,#NOT 20H</i>	;处理 RECV 事件
	<i>MOVX</i>	<i>@DPTR,A</i>	
	<i>MOV</i>	<i>DPTR,#I2CRXD</i>	
	<i>MOVX</i>	<i>A,@DPTR</i>	
	<i>JBC</i>	<i>ISDA,RXDA</i>	
	<i>JBC</i>	<i>ISMA,RXMA</i>	
	<i>MOV</i>	<i>R0,ADDR</i>	;处理 RECV 事件 (RECV DATA)
	<i>MOVX</i>	<i>@R0,A</i>	
	<i>INC</i>	<i>ADDR</i>	
	<i>JMP</i>	<i>ISREXIT</i>	
<i>RXDA:</i>			

```

RXMA:
    JMP      ISREXIT          ;处理 RECV 事件 (RECV DEVICE ADDR)
    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,#1000001B      ;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

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>	
<i>I2CCFG</i>	<i>XDATA</i>	<i>0FE80H</i>	
<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>	
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>	
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>	
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>	
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>	
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>	
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>	
<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>	
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>	
<i>ISDA</i>	<i>BIT</i>	<i>20H.0</i>	;设备地址标志
<i>ISMA</i>	<i>BIT</i>	<i>20H.1</i>	;存储地址标志
<i>ADDR</i>	<i>DATA</i>	<i>21H</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	

```

        ORG      0000H
        LJMP     MAIN

MAIN:
        ORG      0100H

        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, #1000001B           ;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, #0000000B
        MOV     DPTR, #I2CSLST
        MOVX    @DPTR, A
        MOV     A, #0000000B       ;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;
```

```

#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 PIMI           = 0x91;
sfr PIM0           = 0x92;
sfr P0MI           = 0x93;
sfr P0M0           = 0x94;
sfr P2MI           = 0x95;
sfr P2M0           = 0x96;
sfr P3MI           = 0xb1;
sfr P3M0           = 0xb2;
sfr P4MI           = 0xb3;
sfr P4M0           = 0xb4;
sfr P5MI           = 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(0x5a);           //发送设备地址(010_1101B)+写命令(0B)
    RecvACK();
    SendData(0x00);           //发送存储地址
    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

    Start(); //发送起始命令
    SendData(0x5a); //发送设备地址(010_1101B)+写命令(0B)
    RecvACK();
    SendData(0x00); //发送存储地址高字节
    RecvACK();
    Start(); //发送起始命令
    SendData(0x5b); //发送设备地址(010_1101B)+读命令(1B)
    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

<i>P_SW2</i>	<i>DATA</i>	<i>0BAH</i>	
<i>I2CCFG</i>	<i>XDATA</i>	<i>0FE80H</i>	
<i>I2CMSCR</i>	<i>XDATA</i>	<i>0FE81H</i>	
<i>I2CMSST</i>	<i>XDATA</i>	<i>0FE82H</i>	
<i>I2CSLCR</i>	<i>XDATA</i>	<i>0FE83H</i>	
<i>I2CSLST</i>	<i>XDATA</i>	<i>0FE84H</i>	
<i>I2CSLADR</i>	<i>XDATA</i>	<i>0FE85H</i>	
<i>I2CTXD</i>	<i>XDATA</i>	<i>0FE86H</i>	
<i>I2CRXD</i>	<i>XDATA</i>	<i>0FE87H</i>	
<i>SDA</i>	<i>BIT</i>	<i>P1.4</i>	
<i>SCL</i>	<i>BIT</i>	<i>P1.5</i>	
<i>P1M1</i>	<i>DATA</i>	<i>091H</i>	
<i>P1M0</i>	<i>DATA</i>	<i>092H</i>	
<i>P0M1</i>	<i>DATA</i>	<i>093H</i>	
<i>P0M0</i>	<i>DATA</i>	<i>094H</i>	
<i>P2M1</i>	<i>DATA</i>	<i>095H</i>	
<i>P2M0</i>	<i>DATA</i>	<i>096H</i>	
<i>P3M1</i>	<i>DATA</i>	<i>0B1H</i>	
<i>P3M0</i>	<i>DATA</i>	<i>0B2H</i>	
<i>P4M1</i>	<i>DATA</i>	<i>0B3H</i>	
<i>P4M0</i>	<i>DATA</i>	<i>0B4H</i>	
<i>P5M1</i>	<i>DATA</i>	<i>0C9H</i>	
<i>P5M0</i>	<i>DATA</i>	<i>0CAH</i>	
	<i>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>START:</i>	<i>MOV</i>	<i>A,#0000001B</i>	;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, #5AH
CALL    SENDDATA                 ;发送设备地址(010_1101B)+写命令(0B)
CALL    RECVACK
MOV     A, #000H
CALL    SENDDATA                 ;发送存储地址
CALL    RECVACK
MOV     A, #12H
CALL    SENDDATA                 ;写测试数据1
CALL    RECVACK
MOV     A, #78H
CALL    SENDDATA                 ;写测试数据2
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, #5AH
CALL    SENDDATA                 ;发送设备地址(010_1101B)+写命令(0B)
CALL    RECVACK
MOV     A, #000H
CALL    SENDDATA                 ;发送存储地址
CALL    RECVACK
CALL    START                   ;发送起始命令
MOV     A, #5BH
CALL    SENDDATA                 ;发送设备地址(010_1101B)+读命令(1B)
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
```

STC MCU

22 LCM 接口

STC8A8K64D4 系列的单片机内部集成了一个 LCM 接口控制器, 可用于驱动目前流行的液晶显示屏模块。可驱动 I8080 接口和 M6800 接口彩屏, 支持 8 位和 16 位数据宽度

22.1 LCM 接口功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80
LCMIFCFG2	FE51H		LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

LCMIFCPS[1:0]: LCM 接口控制脚选择位

LCMIFCPS [1:0]	RS	I8080 的读信号 RD M6800 的使能信号 E	I8080 的写信号 WR M6800 的读写信号 RW
00	P4.1	P4.4	P4.3
01	P4.1	P3.7	P3.6
10	P4.1	P4.2	P4.0
11	P4.0	P3.7	P3.6

LCMIFDPS[1:0]: 8 位数据 LCM 接口数据脚选择位

LCMIFDPS [1:0]	D16_D8	数据字节 DAT[7:0]
00	0	P2[7:0]
01	0	P6[7:0]
10	0	P2[7:0]
11	0	P6[7:0]

LCMIFDPS[1:0]: 16 位数据 LCM 接口数据脚选择位

LCMIFDPS [1:0]	D16_D8	数据高字节 DAT[15:8]	数据低字节 DAT[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	P7[7:0]
11	1	P6[7:0]	P7[7:0]

22.2 LCM 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	
LCMIFCFG	LCM 接口配置寄存器	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFCFG2	LCM 接口配置寄存器 2	FE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000
LCMIFCR	LCM 接口控制寄存器	FE52H	ENLCMIF	-	-	-	-	CMD[2:0]			0xxx,x000
LCMIFSTA	LCM 接口状态寄存器	FE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
LCMIDDATL	LCM 接口低字节数据	FE54H	LCMIFDAT[7:0]								0000,0000
LCMIDDATH	LCM 接口高字节数据	FE55H	LCMIFDAT[15:8]								0000,0000

22.2.1 LCM 接口配置寄存器 (LCMIFCFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80

LCMIFIE: LCM 接口中断使能控制位

0: 禁止 LCM 接口中断

1: 允许 LCM 接口中断

LCMIFIP[1:0]: LCM 接口中断优先级控制位

LCMIFIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

LCMIFDPS[1:0]: LCM 接口数据脚选择位

LCMIFDPS [1:0]	D16_D8	数据高字节 DAT[15:8]	数据低字节 DAT[7:0]
00	0	N/A	P2[7:0]
01	0	N/A	P6[7:0]
10	0	N/A	P2[7:0]
11	0	N/A	P6[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	P7[7:0]
11	1	P6[7:0]	P7[7:0]

D16_D8: LCM 接口数据宽度控制位

0: 8 位数据宽度

1: 16 位数据宽度

M68_I80: LCM 接口模式选择位

0: I8080 模式

1: M6800 模式

22.2.2 LCM 接口配置寄存器 2 (LCMIFCFG2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG2	FE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]		HOLDT[1:0]		

LCMIFCPS[1:0]: LCM 接口控制脚选择位

LCMIFCPS [1:0]	RS	I8080 的读信号 RD M6800 的使能信号 E	I8080 的写信号 WR M6800 的读写信号 RW
00	P4.1	P4.4	P4.3
01	P4.1	P3.7	P3.6
10	P4.1	P4.2	P4.0
11	P4.0	P3.7	P3.6

SETUPT[2:0]: LCM 接口通讯的数据建立时间控制位 (详见后续章节的时序图)

HOLDT[1:0]: LCM 接口通讯的数据保持时间控制位 (详见后续章节的时序图)

22.2.3 LCM 接口控制寄存器 (LCMIFCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCR	FE52H	ENLCMIF	-	-	-	-	CMD[2:0]		

ELCMIF: LCM 接口使能控制位

0: 禁止 LCM 接口功能

1: 允许 LCM 接口功能

CMD[2:0]: LCM 接口触发命令

CMD[2:0]	触发命令
100	写命令
101	写数据
110	读命令/状态
111	读数据

22.2.4 LCM 接口状态寄存器 (LCMIFSTA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFSTA	FE53H	-	-	-	-	-	-	-	LCMIFIF

LCMIFIF: LCM 接口中断请求标志, 需软件清 0

22.2.5 LCM 接口数据寄存器 (LCMIFDATL, LCMIFDATH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFDATL	FE54H	LCMIFDAT[7:0]							
LCMIFDATH	FE55H	LCMIFDAT[15:8]							

LCMIFDAT: LCM 接口数据寄存器。

当数据宽度为 8 位数据时, 只有 LCMIFDATL 数据有效;

当数据宽度为 16 位数据时, 由 LCMIFDATL 和 LCMIFDATH 共同组合成 16 位数据

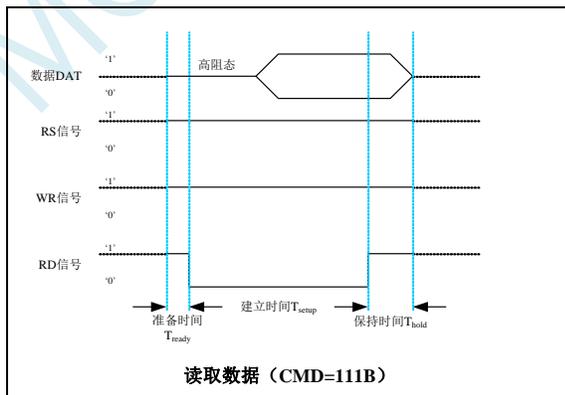
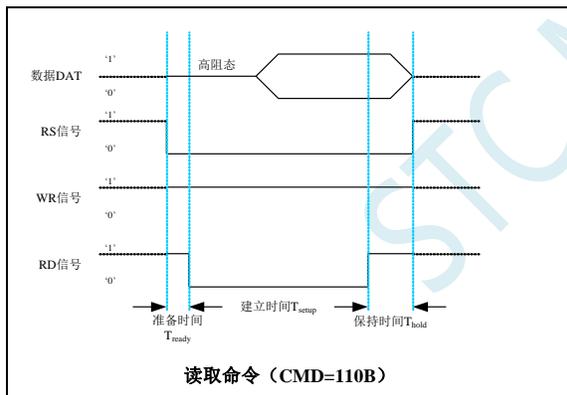
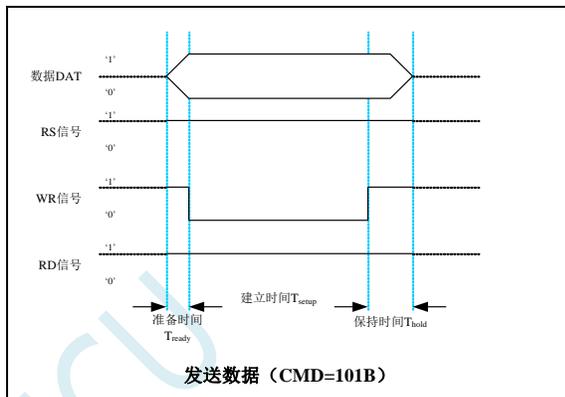
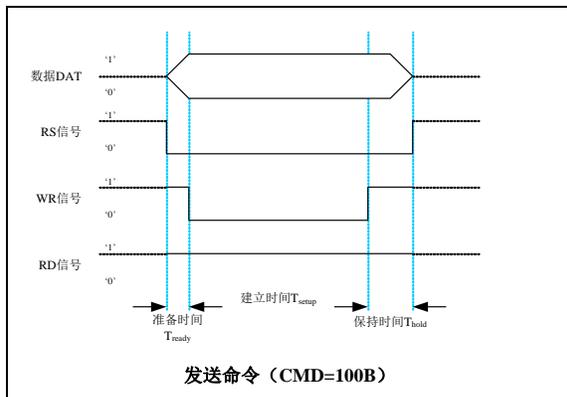
22.3 LCD 接口时序图

注: $T_{ready} = 1$ 个系统时钟

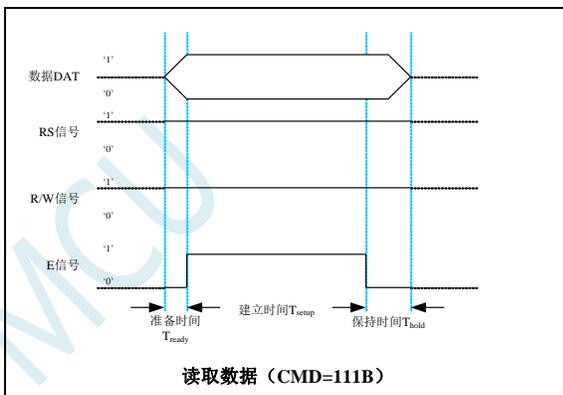
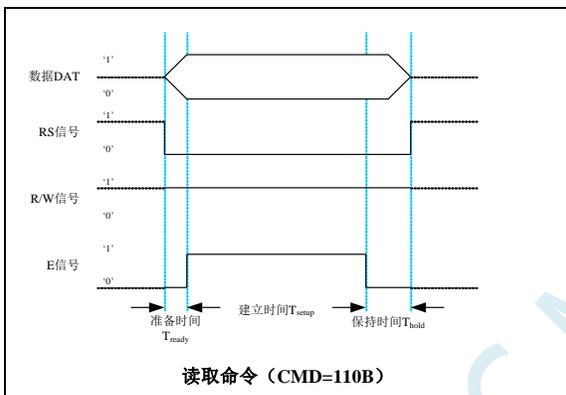
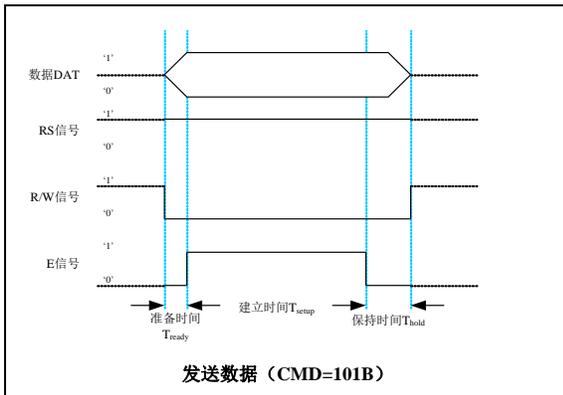
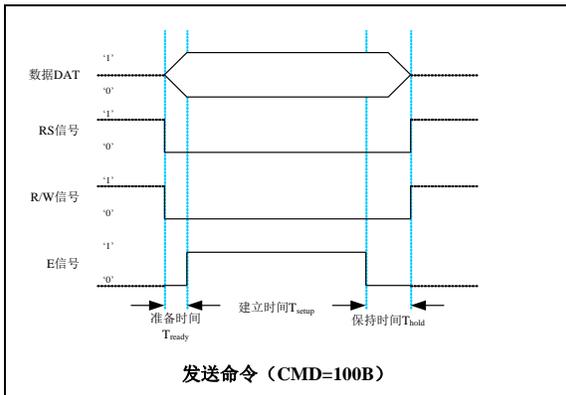
$T_{setup} = (SETUPT + 1)$ 个系统时钟

$T_{hold} = (HOLDT + 1)$ 个系统时钟

22.3.1 I8080 模式



22.3.2 M6800 模式



23 DMA

STC8A8K64D4 系列的单片机支持批量数据存储功能，即传统的 DMA。

支持如下几种 DMA 操作：

- M2M_DMA：XRAM 存储器到 XRAM 存储器的数据读写
- ADC_DMA：自动扫描使能的 ADC 通道并将转换的 ADC 数据自动存储到 XRAM 中
- SPI_DMA：自动将 XRAM 中的数据和 SPI 外设之间进行数据交换
- UR1T_DMA：自动将 XRAM 中的数据通过串口 1 发送出去
- UR1R_DMA：自动将串口 1 接收到的数据存储到 XRAM 中
- UR2T_DMA：自动将 XRAM 中的数据通过串口 2 发送出去
- UR2R_DMA：自动将串口 2 接收到的数据存储到 XRAM 中
- UR3T_DMA：自动将 XRAM 中的数据通过串口 3 发送出去
- UR3R_DMA：自动将串口 3 接收到的数据存储到 XRAM 中
- UR4T_DMA：自动将 XRAM 中的数据通过串口 4 发送出去
- UR4R_DMA：自动将串口 4 接收到的数据存储到 XRAM 中
- LCM_DMA：自动将 XRAM 中的数据和 LCM 设备之间进行数据交换

每次 DMA 数据传输最大数据量为 256 字节。

每种 DMA 对 XRAM 的读写操作都可设置 4 级访问优先级，硬件自动进行 XRAM 总线的访问仲裁，不会影响 CPU 的 XRAM 的访问。相同优先级下，不同 DMA 对 XRAM 的访问顺序如下：SPI_DMA，UR1R_DMA，UR1T_DMA，UR2R_DMA，UR2T_DMA，UR3R_DMA，UR3T_DMA，UR4R_DMA，UR4T_DMA，LCM_DMA，M2M_DMA，ADC_DMA

23.1 DMA 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	
DMA_M2M_CFG	M2M_DMA 配置寄存器	FA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_CR	M2M_DMA 控制寄存器	FA01H	ENM2M	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA 状态寄存器	FA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_M2M_AMT	M2M_DMA 传输总字节数	FA03H									0000,0000
DMA_M2M_DONE	M2M_DMA 传输完成字节数	FA04H									0000,0000
DMA_M2M_TXAH	M2M_DMA 发送高地址	FA05H									0000,0000
DMA_M2M_TXAL	M2M_DMA 发送低地址	FA06H									0000,0000
DMA_M2M_RXAH	M2M_DMA 接收高地址	FA07H									0000,0000
DMA_M2M_RXAL	M2M_DMA 接收低地址	FA08H									0000,0000
DMA_ADC_CFG	ADC_DMA 配置寄存器	FA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_CR	ADC_DMA 控制寄存器	FA11H	ENADC	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_ADC_STA	ADC_DMA 状态寄存器	FA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_ADC_RXAH	ADC_DMA 接收高地址	FA17H									0000,0000
DMA_ADC_RXAL	ADC_DMA 接收低地址	FA18H									0000,0000
DMA_ADC_CFG2	ADC_DMA 配置寄存器 2	FA19H	-	-	-	-	CVTIMESEL[3:0]			xxxx,0000	

DMA_ADC_CHSW0	ADC_DMA 通道使能	FA1AH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	1000,0000
DMA_ADC_CHSW1	ADC_DMA 通道使能	FA1BH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0	0000,0001
DMA_SPI_CFG	SPI_DMA 配置寄存器	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_CR	SPI_DMA 控制寄存器	FA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO	000x,xxx0
DMA_SPI_STA	SPI_DMA 状态寄存器	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_SPI_AMT	SPI_DMA 传输总字节数	FA23H									0000,0000
DMA_SPI_DONE	SPI_DMA 传输完成字节数	FA24H									0000,0000
DMA_SPI_TXAH	SPI_DMA 发送高地址	FA25H									0000,0000
DMA_SPI_TXAL	SPI_DMA 发送低地址	FA26H									0000,0000
DMA_SPI_RXAH	SPI_DMA 接收高地址	FA27H									0000,0000
DMA_SPI_RXAL	SPI_DMA 接收低地址	FA28H									0000,0000
DMA_SPI_CFG2	SPI_DMA 配置寄存器 2	FA29H	-	-	-	-	-	WRPSS	SSS[1:0]		xxxx,x000
DMA_UR1T_CFG	UR1T_DMA 配置寄存器	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_CR	UR1T_DMA 控制寄存器	FA31H	ENUR1T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA 状态寄存器	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1T_AMT	UR1T_DMA 传输总字节数	FA33H									0000,0000
DMA_UR1T_DONE	UR1T_DMA 传输完成字节数	FA34H									0000,0000
DMA_UR1T_TXAH	UR1T_DMA 发送高地址	FA35H									0000,0000
DMA_UR1T_TXAL	UR1T_DMA 发送低地址	FA36H									0000,0000
DMA_UR1R_CFG	UR1R_DMA 配置寄存器	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_CR	UR1R_DMA 控制寄存器	FA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR1R_STA	UR1R_DMA 状态寄存器	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR1R_AMT	UR1R_DMA 传输总字节数	FA3BH									0000,0000
DMA_UR1R_DONE	UR1R_DMA 传输完成字节数	FA3CH									0000,0000
DMA_UR1R_TXAH	UR1R_DMA 发送高地址	FA3DH									0000,0000
DMA_UR1R_TXAL	UR1R_DMA 发送低地址	FA3EH									0000,0000
DMA_UR2T_CFG	UR2T_DMA 配置寄存器	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_CR	UR2T_DMA 控制寄存器	FA41H	ENUR2T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA 状态寄存器	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA 传输总字节数	FA43H									0000,0000
DMA_UR2T_DONE	UR2T_DMA 传输完成字节数	FA44H									0000,0000
DMA_UR2T_TXAH	UR2T_DMA 发送高地址	FA45H									0000,0000
DMA_UR2T_TXAL	UR2T_DMA 发送低地址	FA46H									0000,0000
DMA_UR2R_CFG	UR2R_DMA 配置寄存器	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_CR	UR2R_DMA 控制寄存器	FA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR2R_STA	UR2R_DMA 状态寄存器	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR2R_AMT	UR2R_DMA 传输总字节数	FA4BH									0000,0000
DMA_UR2R_DONE	UR2R_DMA 传输完成字节数	FA4CH									0000,0000
DMA_UR2R_TXAH	UR2R_DMA 发送高地址	FA4DH									0000,0000
DMA_UR2R_TXAL	UR2R_DMA 发送低地址	FA4EH									0000,0000
DMA_UR3T_CFG	UR3T_DMA 配置寄存器	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_CR	UR3T_DMA 控制寄存器	FA51H	ENUR3T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA 状态寄存器	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA 传输总字节数	FA53H									0000,0000

DMA_UR3T_DONE	UR3T_DMA 传输完成字节数	FA54H										0000,0000
DMA_UR3T_TXAH	UR3T_DMA 发送高地址	FA55H										0000,0000
DMA_UR3T_TXAL	UR3T_DMA 发送低地址	FA56H										0000,0000
DMA_UR3R_CFG	UR3R_DMA 配置寄存器	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]			0xxx,0000
DMA_UR3R_CR	UR3R_DMA 控制寄存器	FA59H	ENUR3R	-	TRIG	-	-	-	-	-	CLRIFIFO	0x0x,xxx0
DMA_UR3R_STA	UR3R_DMA 状态寄存器	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF		xxxx,xx00
DMA_UR3R_AMT	UR3R_DMA 传输总字节数	FA5BH										0000,0000
DMA_UR3R_DONE	UR3R_DMA 传输完成字节数	FA5CH										0000,0000
DMA_UR3R_TXAH	UR3R_DMA 发送高地址	FA5DH										0000,0000
DMA_UR3R_TXAL	UR3R_DMA 发送低地址	FA5EH										0000,0000
DMA_UR4T_CFG	UR4T_DMA 配置寄存器	FA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]			0xxx,0000
DMA_UR4T_CR	UR4T_DMA 控制寄存器	FA61H	ENUR4T	TRIG	-	-	-	-	-	-		00xx,xxxx
DMA_UR4T_STA	UR4T_DMA 状态寄存器	FA62H	-	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA 传输总字节数	FA63H										0000,0000
DMA_UR4T_DONE	UR4T_DMA 传输完成字节数	FA64H										0000,0000
DMA_UR4T_TXAH	UR4T_DMA 发送高地址	FA65H										0000,0000
DMA_UR4T_TXAL	UR4T_DMA 发送低地址	FA66H										0000,0000
DMA_UR4R_CFG	UR4R_DMA 配置寄存器	FA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]			0xxx,0000
DMA_UR4R_CR	UR4R_DMA 控制寄存器	FA69H	ENUR4R	-	TRIG	-	-	-	-	-	CLRIFIFO	0x0x,xxx0
DMA_UR4R_STA	UR4R_DMA 状态寄存器	FA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF		xxxx,xx00
DMA_UR4R_AMT	UR4R_DMA 传输总字节数	FA6BH										0000,0000
DMA_UR4R_DONE	UR4R_DMA 传输完成字节数	FA6CH										0000,0000
DMA_UR4R_TXAH	UR4R_DMA 发送高地址	FA6DH										0000,0000
DMA_UR4R_TXAL	UR4R_DMA 发送低地址	FA6EH										0000,0000
DMA_LCM_CFG	LCM_DMA 配置寄存器	FA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]			0xxx,0000
DMA_LCM_CR	LCM_DMA 控制寄存器	FA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	-		0000,0xxx
DMA_LCM_STA	LCM_DMA 状态寄存器	FA72H	-	-	-	-	-	-	TXOVW	LCMIF		xxxx,xx00
DMA_LCM_AMT	LCM_DMA 传输总字节数	FA73H										0000,0000
DMA_LCM_DONE	LCM_DMA 传输完成字节数	FA74H										0000,0000
DMA_LCM_TXAH	LCM_DMA 发送高地址	FA75H										0000,0000
DMA_LCM_TXAL	LCM_DMA 发送低地址	FA76H										0000,0000
DMA_LCM_RXAH	LCM_DMA 接收高地址	FA77H										0000,0000
DMA_LCM_RXAL	LCM_DMA 接收低地址	FA78H										0000,0000

23.2 存储器与存储器之间的数据读写 (M2M_DMA)

23.2.1 M2M_DMA 配置寄存器 (DMA_M2M_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	FA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	

M2MIE: M2M_DMA 中断使能控制位

- 0: 禁止 M2M_DMA 中断
- 1: 允许 M2M_DMA 中断

TXACO: M2M_DMA 源地址 (读取地址) 改变方向

0: 数据读取完成后地址自动递增

1: 数据读取完成后地址自动递减

RXACO: M2M_DMA 目标地址 (写入地址) 改变方向

0: 数据写入完成后地址自动递增

1: 数据写入完成后地址自动递减

M2MIP[1:0]: M2M_DMA 中断优先级控制位

M2MIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

M2MPTY[1:0]: M2M_DMA 数据总线访问优先级控制位

M2MPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.2.2 M2M_DMA 控制寄存器 (DMA_M2M_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CR	FA01H	ENM2M	TRIG	-	-	-	-	-	-

ENM2M: M2M_DMA 功能使能控制位

0: 禁止 M2M_DMA 功能

1: 允许 M2M_DMA 功能

TRIG: M2M_DMA 数据读写触发控制位

0: 写 0 无效

1: 写 1 开始 M2M_DMA 操作,

23.2.3 M2M_DMA 状态寄存器 (DMA_M2M_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	FA02H	-	-	-	-	-	-	-	M2MIF

M2MIF: M2M_DMA 中断请求标志位, 当 M2M_DMA 操作完成后, 硬件自动将 M2MIF 置 1, 若使能 M2M_DMA 中断则进入中断服务程序。标志位需软件清零

23.2.4 M2M_DMA 传输总字节寄存器 (DMA_M2M_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_AMT	FA03H								

DMA_M2M_AMT: 设置需要进行数据读写的字节数。

注: 实际读写的字节数为 (DMA_M2M_AMT+1), 即当 DMA_M2M_AMT 设置为 0 时, 读写 1 字节, 当 DMA_M2M_AMT 设置 255 时, 读写 256 字节

23.2.5 M2M_DMA 传输完成字节寄存器 (DMA_M2M_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_DONE	FA04H								

DMA_M2M_DONE: 当前已经读写完成的字节数。

23.2.6 M2M_DMA 发送地址寄存器 (DMA_M2M_TXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_TXAH	FA05H	ADDR[15:8]							
DMA_M2M_TXAL	FA06H	ADDR[7:0]							

DMA_M2M_TXA: 设置进行数据读写时的源地址。执行 M2M_DMA 操作时会从这个地址开始读数据。

23.2.7 M2M_DMA 接收地址寄存器 (DMA_M2M_RXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_RXAH	FA07H	ADDR[15:8]							
DMA_M2M_RXAL	FA08H	ADDR[7:0]							

DMA_M2M_RXA: 设置进行数据读写时的目标地址。执行 M2M_DMA 操作时会从这个地址开始写入数据。

23.3 ADC 数据自动存储 (ADC_DMA)

23.3.1 ADC_DMA 配置寄存器 (DMA_ADC_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG	FA10H	ADCIE	-			ADCIP[1:0]		ADCPTY[1:0]	

ADCIE: ADC_DMA 中断使能控制位

0: 禁止 ADC_DMA 中断

1: 允许 ADC_DMA 中断

ADCIP[1:0]: ADC_DMA 中断优先级控制位

ADCIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

ADCPTY[1:0]: ADC_DMA 数据总线访问优先级控制位

ADCPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.3.2 ADC_DMA 控制寄存器 (DMA_ADC_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CR	FA11H	ENADC	TRIG	-	-	-	-	-	-

ENADC: ADC_DMA 功能使能控制位

0: 禁止 ADC_DMA 功能

1: 允许 ADC_DMA 功能

TRIG: ADC_DMA 操作触发控制位

0: 写 0 无效

1: 写 1 开始 ADC_DMA 操作,

23.3.3 ADC_DMA 状态寄存器 (DMA_ADC_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_STA	FA12H	-	-	-	-	-	-	-	ADCIF

ADCIF: ADC_DMA 中断请求标志位, 当 ADC_DMA 完成扫描所有使能的 ADC 通道后, 硬件自动将 ADCIF 置 1, 若使能 ADC_DMA 中断则进入中断服务程序。标志位需软件清零

23.3.4 ADC_DMA 接收地址寄存器 (DMA_ADC_RXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_RXAH	FA17H	ADDR[15:8]							

DMA_ADC_RXAL	FA18H	ADDR[7:0]
--------------	-------	-----------

DMA_ADC_RXA: 设置进行 ADC_DMA 操作时 ADC 转换数据的存储地址。

23.3.5 ADC_DMA 配置寄存器 2 (DMA_ADC_CFG2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG2	FA19H	-	-	-	-	CVTIMESEL[3:0]			

CVTIMESEL[3:0]: 设置进行 ADC_DMA 操作时, 对每个 ADC 通道进行 ADC 转换的次数

CVTIMESEL[3:0]	转换次数
0xxx	1 次
1000	2 次
1001	4 次
1010	8 次
1011	16 次
1100	32 次
1101	64 次
1110	128 次
1111	256 次

23.3.6 ADC_DMA 通道使能寄存器 (DMA_ADC_CHSWx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CHSW0	FA1AH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
DMA_ADC_CHSW1	FA1BH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

CHn: 设置 ADC_DMA 操作时, 自动扫描的 ADC 通道。通道扫描总是从编号小的通道开始。

23.3.7 ADC_DMA 的数据存储格式

注: ADC 转换速度和转换结果的对齐方式均由 ADC 相关寄存器进行设置

XRAM[DMA_ADC_RXA+0] = 使能的第 1 通道的第 1 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+1] = 使能的第 1 通道的第 1 次 ADC 转换结果的低字节;

XRAM[DMA_ADC_RXA+2] = 使能的第 1 通道的第 2 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+3] = 使能的第 1 通道的第 2 次 ADC 转换结果的低字节;

...

XRAM[DMA_ADC_RXA+2n-2] = 使能的第 1 通道的第 n 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+2n-1] = 使能的第 1 通道的第 n 次 ADC 转换结果的低字节;

XRAM[DMA_ADC_RXA+2n] = 第 1 通道的 ADC 通道号;

XRAM[DMA_ADC_RXA+2n+1] = 第 1 通道 n 次 ADC 转换结果取完平均值之后的余数;

XRAM[DMA_ADC_RXA+2n+2] = 第 1 通道 n 次 ADC 转换结果平均值的高字节;

XRAM[DMA_ADC_RXA+2n+3] = 第 1 通道 n 次 ADC 转换结果平均值的低字节;

XRAM[DMA_ADC_RXA+(2n+4)+0] = 使能的第 2 通道的第 1 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+(2n+4)+1] = 使能的第 2 通道的第 1 次 ADC 转换结果的低字节;

XRAM[DMA_ADC_RXA+(2n+4)+2] = 使能的第 2 通道的第 2 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+(2n+4)+3] = 使能的第 2 通道的第 2 次 ADC 转换结果的低字节;

...

XRAM[DMA_ADC_RXA+(2n+4)+2n-2] = 使能的第 2 通道的第 n 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+(2n+4)+2n-1] = 使能的第 2 通道的第 n 次 ADC 转换结果的低字节;

XRAM[DMA_ADC_RXA+(2n+4)+2n] = 第 2 通道的 ADC 通道号;

XRAM[DMA_ADC_RXA+(2n+4)+2n+1] = 第 2 通道的 n 次 ADC 转换结果取完平均值之后的余数;

XRAM[DMA_ADC_RXA+(2n+4)+2n+2] = 第 2 通道的 n 次 ADC 转换结果平均值的高字节;

XRAM[DMA_ADC_RXA+(2n+4)+2n+3] = 第 2 通道的 n 次 ADC 转换结果平均值的低字节;

...

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+0] = 使能的第 m 通道的第 1 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+1] = 使能的第 m 通道的第 1 次 ADC 转换结果的低字节;

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2] = 使能的第 m 通道的第 2 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+3] = 使能的第 m 通道的第 2 次 ADC 转换结果的低字节;

...

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n-2] = 使能的第 m 通道的第 n 次 ADC 转换结果的高字节;

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n-1] = 使能的第 m 通道的第 n 次 ADC 转换结果的低字节;

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n] = 第 m 通道的 ADC 通道号;

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n+1] = 第 m 通道的 n 次 ADC 转换结果取完平均值之后的余数;

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n+2] = 第 m 通道的 n 次 ADC 转换结果平均值的高字节;

XRAM[DMA_ADC_RXA+(m-1)(2n+4)+2n+3] = 第 m 通道的 n 次 ADC 转换结果平均值的低字节;

表格形式如下:

ADC 通道	偏移地址	数据
第 1 通道	0	使能的第 1 通道的第 1 次 ADC 转换结果的高字节
	1	使能的第 1 通道的第 1 次 ADC 转换结果的低字节
	2	使能的第 1 通道的第 2 次 ADC 转换结果的高字节
	3	使能的第 1 通道的第 2 次 ADC 转换结果的低字节

	$2n-2$	使能的第 1 通道的第 n 次 ADC 转换结果的高字节
	$2n-1$	使能的第 1 通道的第 n 次 ADC 转换结果的低字节
	$2n$	第 1 通道的 ADC 通道号
	$2n+1$	第 1 通道 n 次 ADC 转换结果取完平均值之后的余数
	$2n+2$	第 1 通道 n 次 ADC 转换结果平均值的高字节
	$2n+3$	第 1 通道 n 次 ADC 转换结果平均值的低字节
第 2 通道	$(2n+4) + 0$	使能的第 2 通道的第 1 次 ADC 转换结果的高字节
	$(2n+4) + 1$	使能的第 2 通道的第 1 次 ADC 转换结果的低字节
	$(2n+4) + 2$	使能的第 2 通道的第 2 次 ADC 转换结果的高字节
	$(2n+4) + 3$	使能的第 2 通道的第 2 次 ADC 转换结果的低字节

	$(2n+4) + 2n-2$	使能的第 2 通道的第 n 次 ADC 转换结果的高字节
	$(2n+4) + 2n-1$	使能的第 2 通道的第 n 次 ADC 转换结果的低字节
	$(2n+4) + 2n$	第 2 通道的 ADC 通道号
	$(2n+4) + 2n+1$	第 2 通道 n 次 ADC 转换结果取完平均值之后的余数
	$(2n+4) + 2n+2$	第 2 通道 n 次 ADC 转换结果平均值的高字节
	$(2n+4) + 2n+3$	第 2 通道 n 次 ADC 转换结果平均值的低字节
...	...	
第 m 通道	$(m-1)(2n+4) + 0$	使能的第 m 通道的第 1 次 ADC 转换结果的高字节
	$(m-1)(2n+4) + 1$	使能的第 m 通道的第 1 次 ADC 转换结果的低字节
	$(m-1)(2n+4) + 2$	使能的第 m 通道的第 2 次 ADC 转换结果的高字节
	$(m-1)(2n+4) + 3$	使能的第 m 通道的第 2 次 ADC 转换结果的低字节

	$(m-1)(2n+4) + 2n-2$	使能的第 m 通道的第 n 次 ADC 转换结果的高字节
	$(m-1)(2n+4) + 2n-1$	使能的第 m 通道的第 n 次 ADC 转换结果的低字节
	$(m-1)(2n+4) + 2n$	第 m 通道的 ADC 通道号
	$(m-1)(2n+4) + 2n+1$	第 m 通道 n 次 ADC 转换结果取完平均值之后的余数
	$(m-1)(2n+4) + 2n+2$	第 m 通道 n 次 ADC 转换结果平均值的高字节
	$(m-1)(2n+4) + 2n+3$	第 m 通道 n 次 ADC 转换结果平均值的低字节

23.4 SPI 与存储器之间的数据交换 (SPI_DMA)

23.4.1 SPI_DMA 配置寄存器 (DMA_SPI_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	

SPIIE: SPI_DMA 中断使能控制位

- 0: 禁止 SPI_DMA 中断
- 1: 允许 SPI_DMA 中断

ACT_TX: SPI_DMA 发送数据控制位

- 0: 禁止 SPI_DMA 发送数据。主机模式时, SPI 只发送时钟到 SCLK 端口, 但不从 XRAM 读取数据, 也不向 MOSI 端口上发送数据; 从机模式时, SPI 不从 XRAM 读取数据, 也不向 MISO 端口上发送数据
- 1: 允许 SPI_DMA 发送数据。主机模式时, SPI 发送时钟到 SCLK 端口, 同时从 XRAM 读取数据, 并将数据发送到 MOSI 端口; 从机模式时, SPI 从 XRAM 读取数据, 并将数据发送到 MISO 端口

ACT_RX: SPI_DMA 接收数据控制位

- 0: 禁止 SPI_DMA 接收数据。主机模式时, SPI 只发送时钟到 SCLK 端口, 但不从 MISO 端口读取数据, 也不向 XRAM 写数据; 从机模式时, SPI 不从 MOSI 端口读取数据, 也不向 XRAM 写数据。
- 1: 允许 SPI_DMA 接收数据。主机模式时, SPI 发送时钟到 SCLK 端口, 同时从 MISO 端口读取数据, 并将数据写入 XRAM; 从机模式时, SPI 从 MOSI 端口读取数据, 并写入 XRAM。

SPIIP[1:0]: SPI_DMA 中断优先级控制位

SPIIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

SPIPTY[1:0]: SPI_DMA 数据总线访问优先级控制位

SPIPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.4.2 SPI_DMA 控制寄存器 (DMA_SPI_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CR	FA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO

ENSPI: SPI_DMA 功能使能控制位

- 0: 禁止 SPI_DMA 功能
- 1: 允许 SPI_DMA 功能

TRIG_M: SPI_DMA 主机模式触发控制位

- 0: 写 0 无效
- 1: 写 1 开始 SPI_DMA 主机模式操作,

TRIG_S: SPI_DMA 从机模式触发控制位

0: 写 0 无效

1: 写 1 开始 SPI_DMA 从机模式操作,

CLRFIFO: 清除 SPI_DMA 接收 FIFO 控制位

0: 写 0 无效

1: 开始 SPI_DMA 操作前, 先清空 SPI_DMA 内置的 FIFO

23.4.3 SPI_DMA 状态寄存器 (DMA_SPI_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_STA	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF

SPIIF: SPI_DMA 中断请求标志位, 当 SPI_DMA 数据交换完成后, 硬件自动将 SPIIF 置 1, 若使能 SPI_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS: SPI_DMA 接收数据丢弃标志位。SPI_DMA 操作过程中, 当 XRAM 总线过于繁忙, 来不及清空 SPI_DMA 的接收 FIFO 导致 SPI_DMA 接收的数据自动丢弃时, 硬件自动将 RXLOSS 置 1。标志位需软件清零

TXOVW: SPI_DMA 数据覆盖标志位。SPI_DMA 正在数据传输过程中, 主机模式的 SPI 写 SPDAT 寄存器再次触发 SPI 数据传输时, 会导致数据传输失败, 此时硬件自动将 TXOVW 置 1。标志位需软件清零

23.4.4 SPI_DMA 传输总字节寄存器 (DMA_SPI_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_AMT	FA23H								

DMA_SPI_AMT: 设置需要进行数据读写的字节数。

注: 实际读写的字节数为 (DMA_SPI_AMT+1), 即当 DMA_SPI_AMT 设置为 0 时, 传输 1 字节, 当 DMA_SPI_AMT 设置 255 时, 传输 256 字节

23.4.5 SPI_DMA 传输完成字节寄存器 (DMA_SPI_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_DONE	FA24H								

DMA_SPI_DONE: 当前已经传输完成的字节数。

23.4.6 SPI_DMA 发送地址寄存器 (DMA_SPI_TXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_TXAH	FA25H	ADDR[15:8]							
DMA_SPI_TXAL	FA26H	ADDR[7:0]							

DMA_SPI_TXA: 设置进行数据传输时的源地址。执行 SPI_DMA 操作时会从这个地址开始读数据。

23.4.7 SPI_DMA 接收地址寄存器 (DMA_SPI_RXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_RXAH	FA27H	ADDR[15:8]							

DMA_SPI_RXAL	FA28H	ADDR[7:0]
--------------	-------	-----------

DMA_SPI_RXA: 设置进行数据传输时的目标地址。执行 SPI_DMA 操作时会从这个地址开始写入数据。

23.4.8 SPI_DMA 配置寄存 2 器 (DMA_SPI_CFG2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG2	FA29H	-	-	-	-	-	WRPSS	SSS[1:0]	

WRPSS: SPI_DMA 过程中使能 SS 脚控制位

0: SPI_DMA 传输过程中, 不自动控制 SS 脚

1: SPI_DMA 传输过程中, 自动拉低 SS 脚, 传输完成后, 自动恢复原始状态

SSS[1:0]: SPI_DMA 过程中, 自动控制 SS 选择位

SSS[1:0]	SS 脚
00	P1.2
01	P2.2
10	P7.4
11	P3.5

23.5 串口 1 与存储器之间的数据交换 (UR1T_DMA, UR1R_DMA)

23.5.1 UR1T_DMA 配置寄存器 (DMA_UR1T_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CFG	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	

UR1TIE: UR1T_DMA 中断使能控制位

0: 禁止 UR1T_DMA 中断

1: 允许 UR1T_DMA 中断

UR1TIP[1:0]: UR1T_DMA 中断优先级控制位

UR1TIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

UR1TPTY[1:0]: UR1T_DMA 数据总线访问优先级控制位

UR1TPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.5.2 UR1T_DMA 控制寄存器 (DMA_UR1T_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CR	FA31H	ENUR1T	TRIG	-	-	-	-	-	-

ENUR1T: UR1T_DMA 功能使能控制位

0: 禁止 UR1T_DMA 功能

1: 允许 UR1T_DMA 功能

TRIG: UR1T_DMA 串口 1 发送触发控制位

0: 写 0 无效

1: 写 1 开始 UR1T_DMA 自动发送数据

23.5.3 UR1T_DMA 状态寄存器 (DMA_UR1T_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_STA	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF

UR1TIF: UR1T_DMA 中断请求标志位, 当 UR1T_DMA 数据发送完成后, 硬件自动将 UR1TIF 置 1, 若使能 UR1T_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW: UR1T_DMA 数据覆盖标志位。UR1T_DMA 正在数据传输过程中, 串口写 SBUF 寄存器再次触发串口发送数据时, 会导致数据传输失败, 此时硬件自动将 TXOVW 置 1。标志位需软件清零

23.5.4 UR1T_DMA 传输总字节寄存器 (DMA_UR1T_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_AMT	FA33H								

DMA_UR1T_AMT: 设置需要自动发送数据的字节数。

注: 实际的字节数为 (DMA_UR1T_AMT+1), 即当 DMA_UR1T_AMT 设置为 0 时, 传输 1 字节, 当 DMA_UR1T_AMT 设置 255 时, 传输 256 字节

23.5.5 UR1T_DMA 传输完成字节寄存器 (DMA_UR1T_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_DONE	FA34H								

DMA_UR1T_DONE: 当前已经发送完成的字节数。

23.5.6 UR1T_DMA 发送地址寄存器 (DMA_UR1T_TXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_TXAH	FA35H	ADDR[15:8]							
DMA_UR1T_TXAL	FA36H	ADDR[7:0]							

DMA_UR1T_TXA: 设置自动发送数据的源地址。执行 UR1T_DMA 操作时会从这个地址开始读数据。

23.5.7 UR1R_DMA 配置寄存器 (DMA_UR1R_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CFG	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	

UR1RIE: UR1R_DMA 中断使能控制位

0: 禁止 UR1R_DMA 中断

1: 允许 UR1R_DMA 中断

UR1RIP[1:0]: UR1R_DMA 中断优先级控制位

UR1RIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

UR1RPTY[1:0]: UR1R_DMA 数据总线访问优先级控制位

UR1RPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.5.8 UR1R_DMA 控制寄存器 (DMA_UR1R_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CR	FA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIPO

ENUR1R: UR1R_DMA 功能使能控制位

0: 禁止 UR1R_DMA 功能

1: 允许 UR1R_DMA 功能

TRIG: UR1R_DMA 串口 1 接收触发控制位

0: 写 0 无效

1: 写 1 开始 UR1R_DMA 自动接收数据

CLRFIFO: 清除 UR1R_DMA 接收 FIFO 控制位

0: 写 0 无效

1: 开始 UR1R_DMA 操作前, 先清空 UR1R_DMA 内置的 FIFO

23.5.9 UR1R_DMA 状态寄存器 (DMA_UR1R_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_STA	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF

UR1RIF: UR1R_DMA 中断请求标志位, 当 UR1R_DMA 接收数据完成后, 硬件自动将 UR1RIF 置 1, 若使能 UR1R_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS: UR1R_DMA 接收数据丢弃标志位。UR1R_DMA 操作过程中, 当 XRAM 总线过于繁忙, 来不及清空 UR1R_DMA 的接收 FIFO 导致 UR1R_DMA 接收的数据自动丢弃时, 硬件自动将 RXLOSS 置 1。标志位需软件清零

23.5.10 UR1R_DMA 传输总字节寄存器 (DMA_UR1R_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_AMT	FA3BH								

DMA_UR1R_AMT: 设置需要自动接收数据的字节数。

注: 实际的字节数为 (DMA_UR1R_AMT+1), 即当 DMA_UR1R_AMT 设置为 0 时, 传输 1 字节, 当 DMA_UR1R_AMT 设置 255 时, 传输 256 字节

23.5.11 UR1R_DMA 传输完成字节寄存器 (DMA_UR1R_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_DONE	FA3CH								

DMA_UR1R_DONE: 当前已经接收完成的字节数。

23.5.12 UR1R_DMA 接收地址寄存器 (DMA_UR1R_RXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_RXAH	FA3DH	ADDR[15:8]							
DMA_UR1R_RXAL	FA3EH	ADDR[7:0]							

DMA_UR1R_RXA: 设置自动接收数据的目标地址。执行 UR1R_DMA 操作时会从这个地址开始写数据。

23.6 串口 2 与存储器之间的数据交换 (UR2T_DMA, UR2R_DMA)

23.6.1 UR2T_DMA 配置寄存器 (DMA_UR2T_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CFG	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	

UR2TIE: UR2T_DMA 中断使能控制位

0: 禁止 UR2T_DMA 中断

1: 允许 UR2T_DMA 中断

UR2TIP[1:0]: UR2T_DMA 中断优先级控制位

UR2TIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

UR2TPTY[1:0]: UR2T_DMA 数据总线访问优先级控制位

UR2TPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.6.2 UR2T_DMA 控制寄存器 (DMA_UR2T_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CR	FA41H	ENUR2T	TRIG	-	-	-	-	-	-

ENUR2T: UR2T_DMA 功能使能控制位

0: 禁止 UR2T_DMA 功能

1: 允许 UR2T_DMA 功能

TRIG: UR2T_DMA 串口 1 发送触发控制位

0: 写 0 无效

1: 写 1 开始 UR2T_DMA 自动发送数据

23.6.3 UR2T_DMA 状态寄存器 (DMA_UR2T_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_STA	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF

UR2TIF: UR2T_DMA 中断请求标志位, 当 UR2T_DMA 数据发送完成后, 硬件自动将 UR2TIF 置 1, 若使能 UR2T_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW: UR2T_DMA 数据覆盖标志位。UR2T_DMA 正在数据传输过程中, 串口写 SBUF 寄存器再次触发串口发送数据时, 会导致数据传输失败, 此时硬件自动将 TXOVW 置 1。标志位需软件清零

23.6.4 UR2T_DMA 传输总字节寄存器 (DMA_UR2T_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_AMT	FA43H								

DMA_UR2T_AMT: 设置需要自动发送数据的字节数。

注: 实际的字节数为 (DMA_UR2T_AMT+1), 即当 DMA_UR2T_AMT 设置为 0 时, 传输 1 字节, 当 DMA_UR2T_AMT 设置 255 时, 传输 256 字节

23.6.5 UR2T_DMA 传输完成字节寄存器 (DMA_UR2T_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_DONE	FA44H								

DMA_UR2T_DONE: 当前已经发送完成的字节数。

23.6.6 UR2T_DMA 发送地址寄存器 (DMA_UR2T_TXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_TXAH	FA45H	ADDR[15:8]							
DMA_UR2T_TXAL	FA46H	ADDR[7:0]							

DMA_UR2T_TXA: 设置自动发送数据的源地址。执行 UR2T_DMA 操作时会从这个地址开始读数据。

23.6.7 UR2R_DMA 配置寄存器 (DMA_UR2R_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CFG	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	

UR2RIE: UR2R_DMA 中断使能控制位

0: 禁止 UR2R_DMA 中断

1: 允许 UR2R_DMA 中断

UR2RIP[1:0]: UR2R_DMA 中断优先级控制位

UR2RIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

UR2RPTY[1:0]: UR2R_DMA 数据总线访问优先级控制位

UR2RPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.6.8 UR2R_DMA 控制寄存器 (DMA_UR2R_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CR	FA49H	ENUR2R	-	TRIG	-	-	-	-	CLRIFO

ENUR2R: UR2R_DMA 功能使能控制位

0: 禁止 UR2R_DMA 功能

1: 允许 UR2R_DMA 功能

TRIG: UR2R_DMA 串口 1 接收触发控制位

0: 写 0 无效

1: 写 1 开始 UR2R_DMA 自动接收数据

CLRFIFO: 清除 UR2R_DMA 接收 FIFO 控制位

0: 写 0 无效

1: 开始 UR2R_DMA 操作前, 先清空 UR2R_DMA 内置的 FIFO

23.6.9 UR2R_DMA 状态寄存器 (DMA_UR2R_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_STA	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF

UR2RIF: UR2R_DMA 中断请求标志位, 当 UR2R_DMA 接收数据完成后, 硬件自动将 UR2RIF 置 1, 若使能 UR2R_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS: UR2R_DMA 接收数据丢弃标志位。UR2R_DMA 操作过程中, 当 XRAM 总线过于繁忙, 来不及清空 UR2R_DMA 的接收 FIFO 导致 UR2R_DMA 接收的数据自动丢弃时, 硬件自动将 RXLOSS 置 1。标志位需软件清零

23.6.10 UR2R_DMA 传输总字节寄存器 (DMA_UR2R_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_AMT	FA4BH								

DMA_UR2R_AMT: 设置需要自动接收数据的字节数。

注: 实际的字节数为 (DMA_UR2R_AMT+1), 即当 DMA_UR2R_AMT 设置为 0 时, 传输 1 字节, 当 DMA_UR2R_AMT 设置 255 时, 传输 256 字节

23.6.11 UR2R_DMA 传输完成字节寄存器 (DMA_UR2R_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_DONE	FA4CH								

DMA_UR2R_DONE: 当前已经接收完成的字节数。

23.6.12 UR2R_DMA 接收地址寄存器 (DMA_UR2T_RXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_RXAH	FA4DH	ADDR[15:8]							
DMA_UR2R_RXAL	FA4EH	ADDR[7:0]							

DMA_UR2R_RXA: 设置自动接收数据的目标地址。执行 UR2R_DMA 操作时会从这个地址开始写数据。

23.7 串口 3 与存储器之间的数据交换 (UR3T_DMA, UR3R_DMA)

23.7.1 UR3T_DMA 配置寄存器 (DMA_UR3T_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CFG	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	

UR3TIE: UR3T_DMA 中断使能控制位

0: 禁止 UR3T_DMA 中断

1: 允许 UR3T_DMA 中断

UR3TIP[1:0]: UR3T_DMA 中断优先级控制位

UR3TIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

UR3TPTY[1:0]: UR3T_DMA 数据总线访问优先级控制位

UR3TPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.7.2 UR3T_DMA 控制寄存器 (DMA_UR3T_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CR	FA51H	ENUR3T	TRIG	-	-	-	-	-	-

ENUR3T: UR3T_DMA 功能使能控制位

0: 禁止 UR3T_DMA 功能

1: 允许 UR3T_DMA 功能

TRIG: UR3T_DMA 串口 1 发送触发控制位

0: 写 0 无效

1: 写 1 开始 UR3T_DMA 自动发送数据

23.7.3 UR3T_DMA 状态寄存器 (DMA_UR3T_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_STA	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF

UR3TIF: UR3T_DMA 中断请求标志位, 当 UR3T_DMA 数据发送完成后, 硬件自动将 UR3TIF 置 1, 若使能 UR3T_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW: UR3T_DMA 数据覆盖标志位。UR3T_DMA 正在数据传输过程中, 串口写 SBUF 寄存器再次触发串口发送数据时, 会导致数据传输失败, 此时硬件自动将 TXOVW 置 1。标志位需软件清零

23.7.4 UR3T_DMA 传输总字节寄存器 (DMA_UR3T_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_AMT	FA53H								

DMA_UR3T_AMT: 设置需要自动发送数据的字节数。

注: 实际的字节数为 (DMA_UR3T_AMT+1), 即当 DMA_UR3T_AMT 设置为 0 时, 传输 1 字节, 当 DMA_UR3T_AMT 设置 255 时, 传输 256 字节

23.7.5 UR3T_DMA 传输完成字节寄存器 (DMA_UR3T_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_DONE	FA54H								

DMA_UR3T_DONE: 当前已经发送完成的字节数。

23.7.6 UR3T_DMA 发送地址寄存器 (DMA_UR3T_TXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_TXAH	FA55H	ADDR[15:8]							
DMA_UR3T_TXAL	FA56H	ADDR[7:0]							

DMA_UR3T_TXA: 设置自动发送数据的源地址。执行 UR3T_DMA 操作时会从这个地址开始读数据。

23.7.7 UR3R_DMA 配置寄存器 (DMA_UR3R_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CFG	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	

UR3RIE: UR3R_DMA 中断使能控制位

0: 禁止 UR3R_DMA 中断

1: 允许 UR3R_DMA 中断

UR3RIP[1:0]: UR3R_DMA 中断优先级控制位

UR3RIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

UR3RPTY[1:0]: UR3R_DMA 数据总线访问优先级控制位

UR3RPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.7.8 UR3R_DMA 控制寄存器 (DMA_UR3R_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CR	FA59H	ENUR3R	-	TRIG	-	-	-	-	CLRIFO

ENUR3R: UR3R_DMA 功能使能控制位

0: 禁止 UR3R_DMA 功能

1: 允许 UR3R_DMA 功能

TRIG: UR3R_DMA 串口 1 接收触发控制位

0: 写 0 无效

1: 写 1 开始 UR3R_DMA 自动接收数据

CLRFIFO: 清除 UR3R_DMA 接收 FIFO 控制位

0: 写 0 无效

1: 开始 UR3R_DMA 操作前, 先清空 UR3R_DMA 内置的 FIFO

23.7.9 UR3R_DMA 状态寄存器 (DMA_UR3R_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_STA	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF

UR3RIF: UR3R_DMA 中断请求标志位, 当 UR3R_DMA 接收数据完成后, 硬件自动将 UR3RIF 置 1, 若使能 UR3R_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS: UR3R_DMA 接收数据丢弃标志位。UR3R_DMA 操作过程中, 当 XRAM 总线过于繁忙, 来不及清空 UR3R_DMA 的接收 FIFO 导致 UR3R_DMA 接收的数据自动丢弃时, 硬件自动将 RXLOSS 置 1。标志位需软件清零

23.7.10 UR3R_DMA 传输总字节寄存器 (DMA_UR3R_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_AMT	FA5BH								

DMA_UR3R_AMT: 设置需要自动接收数据的字节数。

注: 实际的字节数为 (DMA_UR3R_AMT+1), 即当 DMA_UR3R_AMT 设置为 0 时, 传输 1 字节, 当 DMA_UR3R_AMT 设置 255 时, 传输 256 字节

23.7.11 UR3R_DMA 传输完成字节寄存器 (DMA_UR3R_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_DONE	FA5CH								

DMA_UR3R_DONE: 当前已经接收完成的字节数。

23.7.12 UR3R_DMA 接收地址寄存器 (DMA_UR3T_RXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_RXAH	FA5DH	ADDR[15:8]							
DMA_UR3R_RXAL	FA5EH	ADDR[7:0]							

DMA_UR3R_RXA: 设置自动接收数据的目标地址。执行 UR3R_DMA 操作时会从这个地址开始写数据。

23.8 串口 4 与存储器之间的数据交换 (UR4T_DMA, UR4R_DMA)

23.8.1 UR4T_DMA 配置寄存器 (DMA_UR4T_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CFG	FA50H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	

UR4TIE: UR4T_DMA 中断使能控制位

0: 禁止 UR4T_DMA 中断

1: 允许 UR4T_DMA 中断

UR4TIP[1:0]: UR4T_DMA 中断优先级控制位

UR4TIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

UR4TPTY[1:0]: UR4T_DMA 数据总线访问优先级控制位

UR4TPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.8.2 UR4T_DMA 控制寄存器 (DMA_UR4T_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CR	FA51H	ENUR4T	TRIG	-	-	-	-	-	-

ENUR4T: UR4T_DMA 功能使能控制位

0: 禁止 UR4T_DMA 功能

1: 允许 UR4T_DMA 功能

TRIG: UR4T_DMA 串口 1 发送触发控制位

0: 写 0 无效

1: 写 1 开始 UR4T_DMA 自动发送数据

23.8.3 UR4T_DMA 状态寄存器 (DMA_UR4T_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_STA	FA52H	-	-	-	-	-	TXOVW	-	UR4TIF

UR4TIF: UR4T_DMA 中断请求标志位, 当 UR4T_DMA 数据发送完成后, 硬件自动将 UR4TIF 置 1, 若使能 UR4T_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW: UR4T_DMA 数据覆盖标志位。UR4T_DMA 正在数据传输过程中, 串口写 SBUF 寄存器再次触发串口发送数据时, 会导致数据传输失败, 此时硬件自动将 TXOVW 置 1。标志位需软件清零

23.8.4 UR4T_DMA 传输总字节寄存器 (DMA_UR4T_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_AMT	FA53H								

DMA_UR4T_AMT: 设置需要自动发送数据的字节数。

注: 实际的字节数为 (DMA_UR4T_AMT+1), 即当 DMA_UR4T_AMT 设置为 0 时, 传输 1 字节, 当 DMA_UR4T_AMT 设置 255 时, 传输 256 字节

23.8.5 UR4T_DMA 传输完成字节寄存器 (DMA_UR4T_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_DONE	FA54H								

DMA_UR4T_DONE: 当前已经发送完成的字节数。

23.8.6 UR4T_DMA 发送地址寄存器 (DMA_UR4T_TXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_TXAH	FA55H	ADDR[15:8]							
DMA_UR4T_TXAL	FA56H	ADDR[7:0]							

DMA_UR4T_TXA: 设置自动发送数据的源地址。执行 UR4T_DMA 操作时会从这个地址开始读数据。

23.8.7 UR4R_DMA 配置寄存器 (DMA_UR4R_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CFG	FA58H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	

UR4RIE: UR4R_DMA 中断使能控制位

0: 禁止 UR4R_DMA 中断

1: 允许 UR4R_DMA 中断

UR4RIP[1:0]: UR4R_DMA 中断优先级控制位

UR4RIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

UR4RPTY[1:0]: UR4R_DMA 数据总线访问优先级控制位

UR4RPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.8.8 UR4R_DMA 控制寄存器 (DMA_UR4R_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CR	FA59H	ENUR4R	-	TRIG	-	-	-	-	CLRFIPO

ENUR4R: UR4R_DMA 功能使能控制位

0: 禁止 UR4R_DMA 功能

1: 允许 UR4R_DMA 功能

TRIG: UR4R_DMA 串口 1 接收触发控制位

0: 写 0 无效

1: 写 1 开始 UR4R_DMA 自动接收数据

CLRFIFO: 清除 UR4R_DMA 接收 FIFO 控制位

0: 写 0 无效

1: 开始 UR4R_DMA 操作前, 先清空 UR4R_DMA 内置的 FIFO

23.8.9 UR4R_DMA 状态寄存器 (DMA_UR4R_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_STA	FA5AH	-	-	-	-	-	-	RXLOSS	UR4RIF

UR4RIF: UR4R_DMA 中断请求标志位, 当 UR4R_DMA 接收数据完成后, 硬件自动将 UR4RIF 置 1, 若使能 UR4R_DMA 中断则进入中断服务程序。标志位需软件清零

RXLOSS: UR4R_DMA 接收数据丢弃标志位。UR4R_DMA 操作过程中, 当 XRAM 总线过于繁忙, 来不及清空 UR4R_DMA 的接收 FIFO 导致 UR4R_DMA 接收的数据自动丢弃时, 硬件自动将 RXLOSS 置 1。标志位需软件清零

23.8.10 UR4R_DMA 传输总字节寄存器 (DMA_UR4R_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_AMT	FA5BH								

DMA_UR4R_AMT: 设置需要自动接收数据的字节数。

注: 实际的字节数为 (DMA_UR4R_AMT+1), 即当 DMA_UR4R_AMT 设置为 0 时, 传输 1 字节, 当 DMA_UR4R_AMT 设置 255 时, 传输 256 字节

23.8.11 UR4R_DMA 传输完成字节寄存器 (DMA_UR4R_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_DONE	FA5CH								

DMA_UR4R_DONE: 当前已经接收完成的字节数。

23.8.12 UR4R_DMA 接收地址寄存器 (DMA_UR4T_RXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_RXAH	FA5DH	ADDR[15:8]							
DMA_UR4R_RXAL	FA5EH	ADDR[7:0]							

DMA_UR4R_RXA: 设置自动接收数据的目标地址。执行 UR4R_DMA 操作时会从这个地址开始写数据。

23.9 LCM 与存储器之间的数据读写 (LCM_DMA)

23.9.1 LCM_DMA 配置寄存器 (DMA_LCM_CFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CFG	FA70H	LCMIE	ACT_TX	ACT_RX	-	LCMIP[1:0]		LCMPTY[1:0]	

LCMIE: LCM_DMA 中断使能控制位

0: 禁止 LCM_DMA 中断

1: 允许 LCM_DMA 中断

LCMIP[1:0]: LCM_DMA 中断优先级控制位

LCMIP[1:0]	中断优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

LCMPTY[1:0]: LCM_DMA 数据总线访问优先级控制位

LCMPTY [1:0]	总线优先级
00	最低级 (0)
01	较低级 (1)
10	较高级 (2)
11	最高级 (3)

23.9.2 LCM_DMA 控制寄存器 (DMA_LCM_CR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CR	FA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	CLRFIFO

ENLCM: LCM_DMA 功能使能控制位

0: 禁止 LCM_DMA 功能

1: 允许 LCM_DMA 功能

TRIGWC: LCM_DMA 发送命令模式触发控制位

0: 写 0 无效

1: 写 1 开始 LCM_DMA 发送命令模式操作

TRIGWD: LCM_DMA 发送数据模式触发控制位

0: 写 0 无效

1: 写 1 开始 LCM_DMA 发送数据模式操作

TRIGRC: LCM_DMA 读取命令模式触发控制位

0: 写 0 无效

1: 写 1 开始 LCM_DMA 读取命令模式操作

TRIGRD: LCM_DMA 读取数据模式触发控制位

0: 写 0 无效

1: 写 1 开始 LCM_DMA 读取数据模式操作

23.9.3 LCM_DMA 状态寄存器 (DMA_LCM_STA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_STA	FA72H	-	-	-	-	-	-	TXOVW	LCMIF

LCMIF: LCM_DMA 中断请求标志位, 当 LCM_DMA 数据交换完成后, 硬件自动将 LCMIF 置 1, 若使能 LCM_DMA 中断则进入中断服务程序。标志位需软件清零

TXOVW: LCM_DMA 数据覆盖标志位。LCM_DMA 正在数据传输过程中, LCMIF 写 LCMIFDATL 和 LCMIDDATH 寄存器时, 会导致数据传输失败, 此时硬件自动将 TXOVW 置 1。标志位需软件清零

23.9.4 LCM_DMA 传输总字节寄存器 (DMA_LCM_AMT)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_AMT	FA73H								

DMA_LCM_AMT: 设置需要进行数据读写的字节数。

注: 实际读写的字节数为 (DMA_LCM_AMT+1), 即当 DMA_LCM_AMT 设置为 0 时, 传输 1 字节, 当 DMA_LCM_AMT 设置 255 时, 传输 256 字节

23.9.5 LCM_DMA 传输完成字节寄存器 (DMA_LCM_DONE)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_DONE	FA74H								

DMA_LCM_DONE: 当前已经传输完成的字节数。

23.9.6 LCM_DMA 发送地址寄存器 (DMA_LCM_TXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_TXAH	FA75H	ADDR[15:8]							
DMA_LCM_TXAL	FA76H	ADDR[7:0]							

DMA_LCM_TXA: 设置进行数据传输时的源地址。执行 LCM_DMA 操作时会从这个地址开始读数据。

23.9.7 LCM_DMA 接收地址寄存器 (DMA_LCM_RXAx)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_RXAH	FA77H	ADDR[15:8]							
DMA_LCM_RXAL	FA78H	ADDR[7:0]							

DMA_LCM_RXA: 设置进行数据传输时的目标地址。执行 LCM_DMA 操作时会从这个地址开始写入数据。

23.10 范例程序

23.10.1 串口 1 中断模式与电脑收发测试 - DMA 接收超时中断

C 语言代码

```
//测试工作频率为22.1184MHz
```

```
/****** 功能说明 *****
```

串口 1 全双工中断方式收发通讯程序。通过 PC 向 MCU 发送数据, MCU 将收到的数据自动存入 DMA 空间。当一次性接收的内容存满设置的 DMA 空间后, 通过串口 1 的 DMA 自动发送功能把存储空间的数据输出。利用串口接收中断进行超时判断, 超时没有收到新的数据, 表示一串数据已经接收完毕, 将已接收的内容输出, 并清除 DMA 空间。用定时器做波特率发生器, 建议使用 1T 模式(除非低波特率用 12T), 并选择可被波特率整除的时钟频率, 以提高精度。

下载时, 选择时钟 22.1184MHz (用户可自行修改频率)。

```
*****/
```

```
#include "stdio.h"
```

```
#include "stc8a8k64d4.h"
```

```
#define MAIN_Fosc 22118400L //定义主时钟 (精确计算 115200 波特率)
```

```
#define Baudrate1 115200L
```

```
#define Timer0_Reload (65536UL -(MAIN_Fosc / 1000))
```

```
#define DMA_AMT_LEN 255 //设置传输总字节数(0~255) : DMA_AMT_LEN+1
```

```
bit B_1ms; //1ms 标志
```

```
bit DMATxFlag;
```

```
bit DMARxFlag;
```

```
bit BusyFlag;
```

```
u8 Rx_cnt;
```

```
u8 RXI_TimeOut;
```

```
u8 xdata DMABuffer[256];
```

```
void UART1_config(u8 brt);
```

```
void DMA_Config(void);
```

```
void UartPutc(unsigned char dat)
```

```
{
    BusyFlag = 1;
    SBUF = dat;
    while(BusyFlag);
}
```

```
char putchar(char c)
```

```
{
    UartPutc(c);
    return c;
}
```

```
void main(void)
```

```
{
    u16 i;
```

```

P0M1 = 0x00;   P0M0 = 0x00;           //设置为准双向口
P1M1 = 0x00;   P1M0 = 0x00;           //设置为准双向口
P2M1 = 0x00;   P2M0 = 0x00;           //设置为准双向口
P3M1 = 0x00;   P3M0 = 0x00;           //设置为准双向口
P4M1 = 0x00;   P4M0 = 0x00;           //设置为准双向口
P5M1 = 0x00;   P5M0 = 0x00;           //设置为准双向口
P6M1 = 0x00;   P6M0 = 0x00;           //设置为准双向口
P7M1 = 0x00;   P7M0 = 0x00;           //设置为准双向口

for(i=0; i<256; i++)
{
    DMABuffer[i] = i;
}

AUXR = 0x80;           //Timer0 set as 1T, 16 bits timer auto-reload,
TH0 = (u8)(Timer0_Reload / 256);
TL0 = (u8)(Timer0_Reload % 256);
ET0 = 1;               //Timer0 interrupt enable
TR0 = 1;               //Tiner0 run

UART1_config(1);      //使用Timer1 做波特率
DMA_Config();
EA = 1;                //允许总中断

printf("UART1 DMA Timeout Programme!\r\n"); //UART1 发送一个字符串
DMATxFlag = 0;
DMARxFlag = 0;

while (1)
{
    if((DMATxFlag) && (DMARxFlag)) //判断发送完成标志与接收完成标志
    {
        Rx_cnt = 0;
        RX1_TimeOut = 0;
        printf("\r\nUART1 DMA FULL!\r\n"); //UART1 发送一个字符串
        DMATxFlag = 0;
        DMA_URIT_CR = 0xc0; //bit7 1:使能 UART1_DMA,
                             //bit6 1:开始 UART1_DMA 自动发送

        DMARxFlag = 0;
        DMA_URIR_CR = 0xa1; //bit7 1:使能 UART1_DMA,
                             //bit5 1:开始 UART1_DMA 自动接收,
                             //bit0 1:清除 FIFO
    }

    if(B_1ms) //1ms 到
    {
        B_1ms = 0;
        if(RX1_TimeOut > 0) //超时计数
        {
            if(--RX1_TimeOut == 0)
            {
                DMA_URIR_CR = 0x00; //关闭 UART1_DMA
                printf("\r\nUART1 Timeout!\r\n"); //UART1 发送一个字符串

                for(i=0;i<Rx_cnt;i++) UartPutc(DMABuffer[i]);
                printf("\r\n");

                Rx_cnt = 0;
            }
        }
    }
}

```

```

        DMA_URIR_CR = 0xa1;           //bit7 1:使能 UART1_DMA,
                                     //bit5 1:开始 UART1_DMA 自动接收,
                                     //bit0 1:清除 FIFO
    }
}

void DMA_Config(void)
{
    P_SW2 = 0x80;
    DMA_URIT_CFG = 0x80;             //bit7 1:Enable Interrupt
    DMA_URIT_STA = 0x00;
    DMA_URIT_AMT = DMA_AMT_LEN;     //设置传输总字节数: n+1
    DMA_URIT_TXA = DMABuffer;
    DMA_URIT_CR = 0xc0;             //bit7 1:使能 UART1_DMA,
                                     //bit6 1:开始 UART1_DMA 自动发送

    DMA_URIR_CFG = 0x80;           //bit7 1:Enable Interrupt
    DMA_URIR_STA = 0x00;
    DMA_URIR_AMT = DMA_AMT_LEN;     //设置传输总字节数: n+1
    DMA_URIR_RXA = DMABuffer;
    DMA_URIR_CR = 0xa1;           //bit7 1:使能 UART1_DMA,
                                     //bit5 1:开始 UART1_DMA 自动接收,
                                     //bit0 1:清除 FIFO
}

void SetTimer2Baudrate(u16 dat)
{
    AUXR &= ~(1<<4);               //Timer stop
    AUXR &= ~(1<<3);               //Timer2 set As Timer
    AUXR |= (1<<2);                //Timer2 set as IT mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2 &= ~(1<<2);               //禁止中断
    AUXR |= (1<<4);               //Timer run enable
}

void UART1_config(u8 brt)
                                     //选择波特率:
                                     //2: 使用Timer2 做波特率,
                                     //其它值: 使用Timer1 做波特率
{
    /***** 波特率使用定时器2 *****/
    if(brt == 2)
    {
        AUXR |= 0x01;             //S1 BRT Use Timer2;
        SetTimer2Baudrate(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /***** 波特率使用定时器1 *****/
    else
    {
        TRI = 0;
        AUXR &= ~0x01;           //S1 BRT Use Timer1;
        AUXR |= (1<<6);         //Timer1 set as IT mode
        TMOD &= ~(1<<6);        //Timer1 set As Timer
        TMOD &= ~0x30;          //Timer1_16bitAutoReload;
        THI = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
    }
}

```

```

    TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
    ET1 = 0; //禁止中断
    INTCLKO &= ~0x02; //不输出时钟
    TRI = 1;
}
/*****/

SCON = (SCON & 0x3f) / 0x40; //UART1 模式:
//0x00: 同步移位输出,
//0x40: 8 位数据,可变波特率,
//0x80: 9 位数据,固定波特率,
//0xc0: 9 位数据,可变波特率
//高优先级中断
//允许中断
//允许接收

// PS = 1;
ES = 1; //允许中断
REN = 1; //允许接收
P_SW1 &= 0x3f;
P_SW1 /= 0x00; //UART1 switch to:
//0x00: P3.0 P3.1,
//0x40: P3.6 P3.7,
//0x80: P1.6 P1.7,
//0xC0: P4.3 P4.4

RX1_TimeOut = 0;
}

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        Rx_cnt++;
        if(Rx_cnt >= DMA_AMT_LEN) Rx_cnt = 0;
        RX1_TimeOut = 5; //如果 5ms 没收到新的数据, 判定一串数据接收完毕
    }

    if(TI)
    {
        TI = 0;
        BusyFlag = 0;
    }
}

void timer0 (void) interrupt 1
{
    B_1ms = 1; //1ms 标志
}

void UART1_DMA_Interrupt(void) interrupt 13
{
    if(DMA_URIT_STA & 0x01) //发送完成
    {
        DMA_URIT_STA &= ~0x01;
        DMATxFlag = 1;
    }
    if(DMA_URIT_STA & 0x04) //数据覆盖
    {
        DMA_URIT_STA &= ~0x04;
    }
}

```

```

if (DMA_URIR_STA & 0x01)           //接收完成
{
    DMA_URIR_STA &= ~0x01;
    DMARxFlag = 1;
}
if (DMA_URIR_STA & 0x02)           //数据丢弃
{
    DMA_URIR_STA &= ~0x02;
}
}

```

//文件: ISR.ASM

//中断号大于 31 的中断, 需要进行中断入口地址重映射处理

```

CSEG AT 012BH                       ;P0INT_VECTOR
JMP P0INT_ISR
CSEG AT 0133H                       ;P1INT_VECTOR
JMP P1INT_ISR
CSEG AT 013BH                       ;P2INT_VECTOR
JMP P2INT_ISR
CSEG AT 0143H                       ;P3INT_VECTOR
JMP P3INT_ISR
CSEG AT 014BH                       ;P4INT_VECTOR
JMP P4INT_ISR
CSEG AT 0153H                       ;P5INT_VECTOR
JMP P5INT_ISR
CSEG AT 015BH                       ;P6INT_VECTOR
JMP P6INT_ISR
CSEG AT 0163H                       ;P7INT_VECTOR
JMP P7INT_ISR
CSEG AT 016BH                       ;P8INT_VECTOR
JMP P8INT_ISR
CSEG AT 0173H                       ;P9INT_VECTOR
JMP P9INT_ISR
CSEG AT 017BH                       ;M2MDMA_VECTOR
JMP M2MDMA_ISR
CSEG AT 0183H                       ;ADCDMA_VECTOR
JMP ADCDMA_ISR
CSEG AT 018BH                       ;SPIDMA_VECTOR
JMP SPIDMA_ISR
CSEG AT 0193H                       ;UITXDMA_VECTOR
JMP UITXDMA_ISR
CSEG AT 019BH                       ;UIRXDMA_VECTOR
JMP UIRXDMA_ISR
CSEG AT 01A3H                       ;U2TXDMA_VECTOR
JMP U2TXDMA_ISR
CSEG AT 01ABH                       ;U2RXDMA_VECTOR
JMP U2RXDMA_ISR
CSEG AT 01B3H                       ;U3TXDMA_VECTOR
JMP U3TXDMA_ISR
CSEG AT 01BBH                       ;U3RXDMA_VECTOR
JMP U3RXDMA_ISR
CSEG AT 01C3H                       ;U4TXDMA_VECTOR
JMP U4TXDMA_ISR
CSEG AT 01CBH                       ;U4RXDMA_VECTOR
JMP U4RXDMA_ISR
CSEG AT 01D3H                       ;LCMDMA_VECTOR
JMP LCMDMA_ISR
CSEG AT 01DBH                       ;LCMIF_VECTOR

```

JMP *LCMIF_ISR*

P0INT_ISR:
P1INT_ISR:
P2INT_ISR:
P3INT_ISR:
P4INT_ISR:
P5INT_ISR:
P6INT_ISR:
P7INT_ISR:
P8INT_ISR:
P9INT_ISR:
M2MDMA_ISR:
ADC DMA_ISR:
SPIDMA_ISR:
U1TXDMA_ISR:
U1RXDMA_ISR:
U2TXDMA_ISR:
U2RXDMA_ISR:
U3TXDMA_ISR:
U3RXDMA_ISR:
U4TXDMA_ISR:
U4RXDMA_ISR:
LCMDMA_ISR:
LCMIF_ISR:

JMP *006BH*

END

23.10.2 串口 1 中断模式与电脑收发测试 - DMA 数据校验

C 语言代码

//测试工作频率为22.1184MHz

*/***** 功能说明 *****/*

串口 1 全双工中断方式收发通讯程序。通过 PC 向 MCU 发送数据, MCU 将收到的数据自动存入 DMA 空间。数据包的最后两个字节作为校验位, 例程以 `crc16_ccitt` 算法进行校验。当 DMA 空间存满设置大小的内容后, 对有效数据进行校验计算, 然后与最后两位校验位进行对比。通过串口 1 的 DMA 自动发送功能把存储空间的数据输出。用定时器做波特率发生器, 建议使用 1T 模式(除非低波特率用 12T), 并选择可被波特率整除的时钟频率, 以提高精度。

下载时, 选择时钟 22.1184MHz (用户可自行修改频率)。

******/*

#include "stdio.h"

#include "STC8Hxxxx.h"

#include "crc16.h"

#define MAIN_Fosc 22118400L

//定义主时钟 (精确计算 115200 波特率)

#define Baudrate1 115200L

#define DMA_AMT_LEN 255

//设置传输总字节数(0~255) : DMA_AMT_LEN+1

```
bit    DMATxFlag;
```

```
bit    DMARxFlag;
```

```
u8     xdata DMABuffer[256];
```

```
void UART1_config(u8 brt);
```

```
void DMA_Config(void);
```

```
void UartPutc(unsigned char dat)
```

```
{
    SBUF = dat;
    while(TI == 0);
    TI = 0;
}
```

```
char putchar(char c)
```

```
{
    UartPutc(c);
    return c;
}
```

```
/*CRC 计算函数*/
```

```
u16 crc16_ccitt(u8 *pbuf, u16 len)
```

```
{
    unsigned short code crc16_ccitt_table[256] =
    {
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
        0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
        0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
        0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
        0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
        0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
        0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
        0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
        0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
        0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
        0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
        0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
        0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
        0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
        0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
        0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
        0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
        0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
        0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
        0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
        0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
        0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
        0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
        0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
        0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
        0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
        0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
        0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CCI,
        0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
        0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
    }
}
```

```

};

u16 crc16 = 0x0000;
u16 crc_h8, crc_l8;

while( len-- ) {
    crc_h8 = (crc16 >> 8);
    crc_l8 = (crc16 << 8);
    crc16 = crc_l8 ^ crc16_ccitt_table[crc_h8 ^ *pbuf];
    pbuf++;
}

return crc16;
}

void main(void)
{
    u16 i;
    u16 CheckSum;

    P0M1 = 0x00;    P0M0 = 0x00;    //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;    //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;    //设置为准双向口
    P3M1 = 0x00;    P3M0 = 0x00;    //设置为准双向口
    P4M1 = 0x00;    P4M0 = 0x00;    //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;    //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;    //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;    //设置为准双向口

    for(i=0; i<256; i++)
    {
        DMABuffer[i] = i;
    }

    P_SW2 = 0x80;
    DMA_URIT_STA = 0x00;
    UART1_config(1);
    printf("UART1 DMA CRC Programme!\r\n");

    DMA_Config();
    EA = 1;    //允许总中断

    DMATxFlag = 0;
    DMARxFlag = 0;

    while (1)
    {
        if((DMATxFlag) && (DMARxFlag))
        {
            CheckSum = crc16_ccitt(DMABuffer,DMA_AMT_LEN-1);
            if(((u8)CheckSum == DMABuffer[DMA_AMT_LEN-1]) &&
                ((u8)(CheckSum>>8) == DMABuffer[DMA_AMT_LEN]))
            {
                printf("\r\nOK! CheckSum = %04x\r\n",CheckSum);
            }
            else
            {
                printf("\r\nERROR! CheckSum = %04x\r\n",CheckSum);
            }
        }
    }
}

```

```

        DMAxFlag = 0;
        DMA_URIT_CR = 0xc0; //bit7 1:使能 UART1_DMA,
                             //bit6 1:开始 UART1_DMA 自动发送

        DMAxFlag = 0;
        DMA_URIR_CR = 0xa1; //bit7 1:使能 UART1_DMA,
                             //bit5 1:开始 UART1_DMA 自动接收,
                             //bit0 1:清除 FIFO
    }
}

void DMA_Config(void)
{
    P_SW2 = 0x80;
    DMA_URIT_CFG = 0x80; //bit7 1:Enable Interrupt
    DMA_URIT_STA = 0x00;
    DMA_URIT_AMT = DMA_AMT_LEN; //设置传输总字节数: n+1
    DMA_URIT_TXA = DMABuffer;
    DMA_URIT_CR = 0xc0; //bit7 1:使能 UART1_DMA,
                        //bit6 1:开始 UART1_DMA 自动发送

    DMA_URIR_CFG = 0x80; //bit7 1:Enable Interrupt
    DMA_URIR_STA = 0x00;
    DMA_URIR_AMT = DMA_AMT_LEN; //设置传输总字节数: n+1
    DMA_URIR_RXA = DMABuffer;
    DMA_URIR_CR = 0xa1; //bit7 1:使能 UART1_DMA,
                        //bit5 1:开始 UART1_DMA 自动接收, bit0 1:清除 FIFO
}

void SetTimer2Baudrate(u16 dat) //选择波特率:
                                //2: 使用Timer2 做波特率,
                                //其它值: 使用Timer1 做波特率
{
    AUXR &= ~(1<<4); //Timer stop
    AUXR &= ~(1<<3); //Timer2 set As Timer
    AUXR |= (1<<2); //Timer2 set as IT mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2 &= ~(1<<2); //禁止中断
    AUXR |= (1<<4); //Timer run enable
}

void UART1_config(u8 brt) //选择波特率:
                           //2: 使用Timer2 做波特率
                           //其它值: 使用Timer1 做波特率
{
    /***** 波特率使用定时器2 *****/
    if(brt == 2)
    {
        AUXR |= 0x01; //S1 BRT Use Timer2;
        SetTimer2Baudrate(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /***** 波特率使用定时器1 *****/
    else
    {
        TRI = 0;
        AUXR &= ~0x01; //S1 BRT Use Timer1;
        AUXR |= (1<<6); //Timer1 set as IT mode
    }
}

```

```

    TMOD &= ~(1<<6);           //Timer1 set As Timer
    TMOD &= ~0x30;           //Timer1_16bitAutoReload;
    TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
    TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
    ET1 = 0;                 //禁止中断
    INTCLKO &= ~0x02;       //不输出时钟
    TRI = 1;
}
/*****/

SCON = (SCON & 0x3f) | 0x40; //UART1 模式,
//0x00: 同步移位输出,
//0x40: 8 位数据,可变波特率,
//0x80: 9 位数据,固定波特率,
//0xc0: 9 位数据,可变波特率
// PS = 1; //高优先级中断
// ES = 1; //允许中断
REN = 1; //允许接收
P_SW1 &= 0x3f;
P_SW1 |= 0x00;
}

void UART1_DMA_Interrupt(void) interrupt 13
{
    if (DMA_URIT_STA & 0x01) //发送完成
    {
        DMA_URIT_STA &= ~0x01;
        DMATxFlag = 1;
    }
    if (DMA_URIT_STA & 0x04) //数据覆盖
    {
        DMA_URIT_STA &= ~0x04;
    }

    if (DMA_URIR_STA & 0x01) //接收完成
    {
        DMA_URIR_STA &= ~0x01;
        DMARxFlag = 1;
    }
    if (DMA_URIR_STA & 0x02) //数据丢弃
    {
        DMA_URIR_STA &= ~0x02;
    }
}

```

//文件: ISR.ASM

//中断号大于 31 的中断, 需要进行中断入口地址重映射处理

```

CSEG AT 012BH ;P0INT_VECTOR
JMP P0INT_ISR
CSEG AT 0133H ;P1INT_VECTOR
JMP P1INT_ISR
CSEG AT 013BH ;P2INT_VECTOR
JMP P2INT_ISR
CSEG AT 0143H ;P3INT_VECTOR
JMP P3INT_ISR
CSEG AT 014BH ;P4INT_VECTOR
JMP P4INT_ISR
CSEG AT 0153H ;P5INT_VECTOR

```

```

JMP      P5INT_ISR
CSEG AT  015BH      ;P6INT_VECTOR
JMP      P6INT_ISR
CSEG AT  0163H     ;P7INT_VECTOR
JMP      P7INT_ISR
CSEG AT  016BH     ;P8INT_VECTOR
JMP      P8INT_ISR
CSEG AT  0173H     ;P9INT_VECTOR
JMP      P9INT_ISR
CSEG AT  017BH     ;M2MDMA_VECTOR
JMP      M2MDMA_ISR
CSEG AT  0183H     ;ADCDMA_VECTOR
JMP      ADCDMA_ISR
CSEG AT  018BH     ;SPIDMA_VECTOR
JMP      SPIDMA_ISR
CSEG AT  0193H     ;UITXDMA_VECTOR
JMP      UITXDMA_ISR
CSEG AT  019BH     ;UIRXDMA_VECTOR
JMP      UIRXDMA_ISR
CSEG AT  01A3H     ;U2TXDMA_VECTOR
JMP      U2TXDMA_ISR
CSEG AT  01ABH     ;U2RXDMA_VECTOR
JMP      U2RXDMA_ISR
CSEG AT  01B3H     ;U3TXDMA_VECTOR
JMP      U3TXDMA_ISR
CSEG AT  01BBH     ;U3RXDMA_VECTOR
JMP      U3RXDMA_ISR
CSEG AT  01C3H     ;U4TXDMA_VECTOR
JMP      U4TXDMA_ISR
CSEG AT  01CBH     ;U4RXDMA_VECTOR
JMP      U4RXDMA_ISR
CSEG AT  01D3H     ;LCMDMA_VECTOR
JMP      LCMDMA_ISR
CSEG AT  01DBH     ;LCMIF_VECTOR
JMP      LCMIF_ISR

```

P0INT_ISR:

PIINT_ISR:

P2INT_ISR:

P3INT_ISR:

P4INT_ISR:

P5INT_ISR:

P6INT_ISR:

P7INT_ISR:

P8INT_ISR:

P9INT_ISR:

M2MDMA_ISR:

ADCDMA_ISR:

SPIDMA_ISR:

UITXDMA_ISR:

UIRXDMA_ISR:

U2TXDMA_ISR:

U2RXDMA_ISR:

U3TXDMA_ISR:

U3RXDMA_ISR:

U4TXDMA_ISR:

U4RXDMA_ISR:

LCMDMA_ISR:

LCMIF_ISR:

JMP 006BH

END

代码测试方法

根据预定义的 DMA 数据包长度 (例如: 256 字节), 通过串口工具发送一包数据 (254 字节), 并在最后加上 2 个字节的 CCITT-CRC16 校验码:



MCU 收到整包数据 (256 字节) 之后对前面 254 字节数据进行 CRC16 校验, 得出来的校验码与最后两个字节进行比较, 如果数值相等, 打印 “OK!” 以及计算出来的校验码, 然后输出 DMA 空间收取的内容。



如果校验码数值不相等, 打印 “ERROR!” 以及计算出来的校验码。

24 增强型双数据指针

STC8A8K64D4 系列的单片机内部集成了两组 16 位的数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能

24.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	数据指针 (低字节)	82H									0000,0000
DPH	数据指针 (高字节)	83H									0000,0000
DPL1	第二组数据指针 (低字节)	E4H									0000,0000
DPH1	第二组数据指针 (高字节)	E5H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR 时序控制寄存器	AEH									0000,0000

24.1.1 第 1 组 16 位数据指针寄存器 (DPTR0)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL 为低 8 位数据 (低字节)

DPH 为高 8 位数据 (高字节)

DPL 和 DPH 组合为第一组 16 位数据指针寄存器 DPTR0

24.1.2 第 2 组 16 位数据指针寄存器 (DPTR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1 为低 8 位数据 (低字节)

DPH1 为高 8 位数据 (高字节)

DPL1 和 DPH1 组合为第二组 16 位数据指针寄存器 DPTR1

24.1.3 数据指针控制寄存器 (DPS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1: 控制 DPTR1 自动递增方式

0: DPTR1 自动递增

1: DPTR1 自动递减

ID0: 控制DPTR0自动递增方式

0: DPTR0 自动递增

1: DPTR0 自动递减

TSL: DPTR0/DPTR1 自动切换控制 (自动对SEL进行取反)

0: 关闭自动切换功能

1: 使能自动切换功能

当 TSL 位被置 1 后, 每当执行完成相关指令后, 系统会自动将 SEL 位取反。

与 TSL 相关的指令包括如下指令:

```
MOV    DPTR,#data16
INC    DPTR
MOVC   A,@A+DPTR
MOVX   A,@DPTR
MOVX   @DPTR,A
```

AU1/AU0: 使能DPTR1/DPTR0使用ID1/ID0控制位进行自动递增/递减控制

0: 关闭自动递增/递减功能

1: 使能自动递增/递减功能

注意: 在写保护模式下, AU0 和 AU1 位无法直接单独使能, 若单独使能 AU1 位, 则 AU0 位也会被自动使能, 若单独使能 AU0, 没有效果。若需要单独使能 AU1 或者 AU0, 则必须使用 TA 寄存器触发 DPS 的保护机制 (参考 TA 寄存器的说明)。另外, 只有执行下面的 3 条指令后才会对 DPTR0/DPTR1 进行自动递增/递减操作。3 条相关指令如下:

```
MOVC   A,@A+DPTR
MOVX   A,@DPTR
MOVX   @DPTR,A
```

SEL: 选择DPTR0/DPTR1作为当前的目标DPTR

0: 选择 DPTR0 作为目标 DPTR

1: 选择 DPTR1 作为目标 DPTR

SEL 选择目标 DPTR 对下面指令有效:

```
MOV    DPTR,#data16
INC    DPTR
MOVC   A,@A+DPTR
MOVX   A,@DPTR
MOVX   @DPTR,A
JMP    @A+DPTR
```

24.1.4 数据指针控制寄存器 (TA)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

TA 寄存器是对 DPS 寄存器中的 AU1 和 AU0 进行写保护的。由于程序无法对 DPS 中的 AU1 和 AU0 进行单独的写入, 所以当需要单独使能 AU1 或者 AU0 时, 必须使用 TA 寄存器进行触发。TA 寄存器是只写寄存器。当需要对 AU1 或者 AU0 进行单独使能时, 必须按照如下的步骤进行操作:

```
CLR    EA           ;关闭中断 (必需)
MOV    TA,#0AAH    ;写入触发命令序列 1
                           ;此处不能有其他任何指令
```

MOV	TA,#55H	;写入触发命令序列 2 ;此处不能有其他任何指令
MOV	DPS,#xxH	;写保护暂时关闭, 可向 DPS 中写入任何值 ;DSP 再次进行写保护状态
SETB	EA	;打开中断 (如有必要)

STC MCU

24.2 范例程序

24.2.1 示例代码 1

将程序空间 1000H~1003H 的 4 个字节数据反向复制到扩展 RAM 的 0100H~0103H 中, 即

```
C:1000H -> X:0103H
C:1001H -> X:0102H
C:1002H -> X:0101H
C:1003H -> X:0100H
```

汇编代码

;测试工作频率为 11.0592MHz

```

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

MAIN:
                ORG      0100H

                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      DPS,#00100000B      ;使能 TSL,并选择 DPTR0
                MOV      DPTR,#1000H        ;将 1000H 写入 DPTR0 后选择 DPTR1 为 DPTR
                MOV      DPTR,#0103H        ;将 0103H 写入 DPTR1 中
                MOV      DPS,#10111000B    ;设置 DPTR1 为递减模式,DPTR0 为递增模式,使能 TSL
                                           ;AU0 和 AU1,并选择 DPTR0 为当前的 DPTR
                MOV      R7,#4              ;设置数据复制个数

COPY_NEXT:
                CLR      A
                MOVC     A,@A+DPTR        ;从 DPTR0 所指的程序空间读取数据,
                                           ;完成后 DPTR0 自动加 1 并将 DPTR1 设置为 DPTR
                MOVB     @DPTR,A          ;将 ACC 的数据写入到 DPTR1 所指的 XDATA 中,

```

```

                                ;完成后DPTR1 自动减1 并将DPTR0 设置为DPTR
                                ;
                                DJNZ     R7,COPY_NEXT
                                SJMP     $
                                END

```

24.2.2 示例代码 2

将扩展 RAM 的 0100H~0103H 中的数据依次发送到 P0 口

汇编代码

;测试工作频率为 11.0592MHz

```

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

                                CLR      EA                                ;关闭中断
                                MOV      TA, #0AAH                        ;写入DPS 写保护触发命令1
                                MOV      TA, #55H                          ;写入DPS 写保护触发命令2
                                MOV      DPS, #00001000B                    ;DPTR0 递增,单独使能AU0,并选择DPTR0
                                SETB    EA                                ;打开中断
                                MOV      DPTR, #0100H                      ;将0100H 写入DPTR0 中
                                MOVX    A, @DPTR                          ;从DPTR0 所指的XRAM 读取数据后DPTR0 自动加1
                                MOV      P0, A                            ;数据输出到P0 口
                                MOVX    A, @DPTR                          ;从DPTR0 所指的XRAM 读取数据后DPTR0 自动加1
                                MOV      P0, A                            ;数据输出到P0 口
                                MOVX    A, @DPTR                          ;从DPTR0 所指的XRAM 读取数据后DPTR0 自动加1
                                MOV      P0, A                            ;数据输出到P0 口
                                MOVX    A, @DPTR                          ;从DPTR0 所指的XRAM 读取数据后DPTR0 自动加1

```

```
MOV      P0,A           ;数据输出到P0 口
SJMP    $
END
```

STC MCU

25 MDU16 硬件 16 位乘除法器

STC8A8K64D4 系列型号的单片机内部集成了 MDU16/16 位硬件乘除法器。

支持如下数据运算：

- 数据规格化（需要 3~20 个时钟的运算时间）
- 逻辑左移（需要 3~18 个时钟的运算时间）
- 逻辑右移（需要 3~18 个时钟的运算时间）
- 16 位乘以 16 位（需要 10 个时钟的运算时间）
- 16 位除以 16 位（需要 9 个时钟的运算时间）
- 32 位除以 16 位（需要 17 个时钟的运算时间）

所有的操作都是基于无符号整形数据类型。

25.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
MD3	MDU 数据寄存器	FCF0H	MD3[7:0]								0000,0000
MD2	MDU 数据寄存器	FCF1H	MD2[7:0]								0000,0000
MD1	MDU 数据寄存器	FCF2H	MD1[7:0]								0000,0000
MD0	MDU 数据寄存器	FCF3H	MD0[7:0]								0000,0000
MD5	MDU 数据寄存器	FCF4H	MD5[7:0]								0000,0000
MD4	MDU 数据寄存器	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU 模式控制寄存器	FCF6H	MODE[2:0]				SC[4:0]				0000,0000
OPCON	MDU 操作控制寄存器	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000

25.1.1 操作数 1 数据寄存器 (MD0~MD3)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MD3	FCF0H	MD3[7:0]							
MD2	FCF1H	MD2[7:0]							
MD1	FCF2H	MD1[7:0]							
MD0	FCF3H	MD0[7:0]							

25.1.2 操作数 2 数据寄存器 (MD4~MD5)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MD5	FCF4H	MD5[7:0]							
MD4	FCF5H	MD4[7:0]							

32位除以16位除法：

被除数：{MD3,MD2,MD1,MD0}

除数：{MD5,MD4}

商：{MD3,MD2,MD1,MD0}

余数：{MD5,MD4}

16位除以16位除法:

被除数: {MD1,MD0}

除数: {MD5,MD4}

商: {MD1,MD0}

余数: {MD5,MD4}

16位乘以16位乘法:

被乘数: {MD1,MD0}

乘数: {MD5,MD4}

积: {MD3,MD2,MD1,MD0}

32位逻辑左移/逻辑右移

操作数: {MD3,MD2,MD1,MD0}

32位数据规格化:

操作数: {MD3,MD2,MD1,MD0}

25.1.3 MDU 模式控制寄存器 (ARCON), 运算所需时钟数

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ARCON	FCF6H	MODE[2:0]				SC[4:0]			

MODE[2:0]: MDU模式选择

MODE[2:0]	模式	时钟数	操作说明
1	逻辑右移	3~18	将{MD3,MD2,MD1,MD0}中的数据右移SC[4:0]位, MD3的高位补0
2	逻辑左移	3~18	将{MD3,MD2,MD1,MD0}中的数据左移SC[4:0]位, MD0的低位补0
3	数据规格化	3~20	对{MD3,MD2,MD1,MD0}中的数据进行逻辑左移, 将数据高位的0全部移出, 使MD3的最高位为1, 逻辑左移的位数被记录在SC[4:0]中
4	16位×16位	10	{MD1,MD0} × {MD5,MD4} = {MD3,MD2,MD1,MD0}
5	16位÷16位	9	{MD1,MD0} ÷ {MD5,MD4} = {MD1,MD0} ··· {MD5,MD4}
6	32位÷16位	17	{MD3,MD2,MD1,MD0} ÷ {MD5,MD4} = {MD3,MD2,MD1,MD0} ··· {MD5,MD4}
其他	无效		

SC[4:0]: 数据移动位数

当 MDU 为移动模式时, SC 用于设置左移/右移的位数

当 MDU 为数据规格化模式时, SC 为数据规格化后数据所移动的实际位数

25.1.4 MDU 操作控制寄存器 (OPCON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OPCON	FCF7H	-	MDOV	-	-	-	-	RST	ENOP

MDOV: MDU溢出标志位 (只读标志位)

在如下几种情况时, MDOV 会被硬件自动置 1:

- 1、除数为 0 时;
- 2、乘法的积大于 0FFFFH 时;

当软件写 OPCODE.0 (EN) 或者写 ARCON 时, 硬件会自动清除 MDOV

RST: 软件复位 MDU 乘除单元。写 1 触发软件复位, MDU 复位完成后硬件自动清零。

注: 软件复位 MDU 乘除单元时, ARCON 寄存器的值会被清除。

ENOP: MDU 模块使能。写 1 触发 MDU 模块开始计算, 当 MDU 计算完成后, 硬件自动将 ENOP 清零。

软件可以在对 ENOP 置 1 后, 循环的查询 ENOP, 当 ENOP 由 1 变 0 则表示计算完成。

STC MCU

25.2 关于 MDU16 的网友应用杂谈（提供思路，仅供参考）

网友 1：“数据规格化用下面的一个简单例子说明”

- 1、有一个 7 位小数精度的数据：0.0000123，由于数据位宽有限，如果需要有效利用位宽，就需要把前面的数据左移，比如左移后数据为 0.123e-4，其中指数-4 保存在另一个寄存器，记录左移的次数就是记录指数的大小。原来寄存器数据转换为 0.123。这样就把数据右边的位宽腾出来，可以保证后续计算的精度。上面只是用十进制简单的说明规格化原理，二进制原理也是一样的。其中浮点和定点（整数）转换就必须使用规格化的原理，如果两个浮点数相加减时的指数不一样，也需要进行规格化处理（这个过程叫作对阶）。如果两个浮点数的指数相差非常大，相加减时就会出现大数吃小数的问题。比如： $0.123e+4 - 0.12e-4 = 0.123e+4 - 0.0000000012e+4 = 0.123e+4$ 。结果就是被减数，这是因为在减操作前，两个浮点数的指数需要完全一致（对阶），需要把指数小的浮点数进行移位，使指数变为+4。但是数据宽度是有限的 7 位小数精度，0.0000000012e+4 这个数右边的数据会被截断变为 $0.0000000e+4 = 0$ 。

网友 2：“关于 STC8C 的 MDU 功能，我分享一点自己的体会，有不对的请大家批评指教，共同提高。”

- 1、功能 1 和 2 对于缩减和扩展整数数据很有效。首先在进行双操作数运算时，如果两个数的长度不一样，需要转换为相同的长度进行才进行运算。比如 32 位整数乘 8 位整数，就要将 8 位转换为 32 位。其次对 AD 采样的结果，转换为指定的位数精度时也需要位移。最后，比如对网络通讯，需要提取数据的某几位出来进行命令解析或者数据分解合成，位移都是很重要的。由于 8051 只有移动 1 位的指令，多位移动需要借助额外的循环代码，需要很多个指令周期，因此使用 MDU 将比 51 汇编指令快数倍。
- 2、功能 3 是整数转换为浮点数必须的功能。对于满精度的 32 位整数，实现这个功能一般要超过 100 个指令周期，因此 MDU 对转速度的提升是比较大的。由于像 AD 设备输出、像各种三轴加速度输出，一般都是整数的（比如 16 位的），要进行实数运算，要进行三角函数运算，整数的输出必须要转换为浮点数，而且每次采集数据都要进行这数据类型转换，需要转换的次数就很多了。对于高速数据采集和像无人机控制这样的应用，如果采用 DMU 对整体性能的提高就很可观了。
- 3、功能 6 是定点实数运算必须的除法功能，功能 4 是功能 6 对应的 16 位 x16 位结果为 32 位的乘法运算。功能 6 的最常见应用是数据处理中的标度转换，比如对于将参考电压为 5 伏的 10 位 AD 采集的整数转换 3 位数码管的 2 位固定小数点进行显示的运算公式为： $N_{32} = ADN * 500 / 1023$ 。这时只要（1）将 AD 采样值 AND 送 MX（DM1MD0），（2）送 500 到 NX（MD5MD4），（3）执行功能 4，结果是 32 位的了，（4）送 1023 到 NX（MD5MD4），（3）执行功能 6，16 位的结果就在 MX 中了，取回来就行了。另一个常见的应用就是在 TFT 之类的点阵屏上画点和线，比如数字示波器，这些都需要进行坐标变换的乘除法-先乘为 32 位整数，再除以 16 位整数得到 16 位结果。
- 4、功能 4 和功能 6 的组合是实现离散卷积的硬件基础。如果不采用浮点加速硬件，实现浮点数的四则运算比实现整数的四则运算要慢一个量级，因此前辈们发明了用整数变量来实现卷积的方法。首先比如我们常见的将 JPG 图像数据转换为 RGB 图像数据或者相反，就需要进行傅里叶变换，由于图像数据的长度是固定的（8 位或者 16 位），因此就可以用离散傅里叶变换来实现，其中基本只用到 8 位或者 16 位的整数乘法和极少量的 32 位乘除法。这样，我们早期的数码相机才有可能实现。其次 PS 图像处理中常见的各种模板处理，也使用的是二维矩阵卷积方法，也是需要巨量的对整数的（8 位图像视图大小需要 16 位和 32 位的中间计算结果）乘加计算，使用离散卷积将极高的提高运算速度。因此有 MDU 的 STC8 单片机不仅可以用于实时采集和显示图像，也可以实时处理图像。最后人工智能也涉及大量的矢量和矩阵运算，比如神经网络卷积，这些都可

以用功能 4 和功能 6 的组合实现, MDU 应该可以在小型智能化场景中得到应用。只是要实现这些功能, 需要 STC8 的增强型双数据指针的配合, 需要专门的知识结构, 专门编制出函数库来提供给用户使用, 才能发挥 STC8 的 MDU 巨大优势。

STC MCU

25.3 范例程序

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

```
#define MD3U32      (*(unsigned long volatile xdata *)0xfc0)
#define MD3U16     (*(unsigned int volatile xdata *)0xfc0)
#define MD1U16     (*(unsigned int volatile xdata *)0xfc2)
#define MD5U16     (*(unsigned int volatile xdata *)0xfc4)
```

```
#define MD3         (*(unsigned char volatile xdata *)0xfc0)
#define MD2         (*(unsigned char volatile xdata *)0xfc1)
#define MD1         (*(unsigned char volatile xdata *)0xfc2)
#define MD0         (*(unsigned char volatile xdata *)0xfc3)
#define MD5         (*(unsigned char volatile xdata *)0xfc4)
#define MD4         (*(unsigned char volatile xdata *)0xfc5)
#define ARCON       (*(unsigned char volatile xdata *)0xfc6)
#define OPCON       (*(unsigned char volatile xdata *)0xfc7)
```

```
sfr      P_SW2      = 0xBA;
```

```
////////////////////////////////////
```

```
//16 位乘 16 位
```

```
////////////////////////////////////
```

```
unsigned long res;
unsigned int dat1, dat2;
```

```
P_SW2 |= 0x80; //访问扩展寄存器 xsfr
MD1U16 = dat1; //dat1 用户给定
MD5U16 = dat2; //dat2 用户给定
ARCON = 4 << 5; //16 位*16 位,乘法模式
OPCON = 1; //启动计算
while((OPCON & 1) != 0); //等待计算完成
res = MD3U32; //32 位结果
```

```
////////////////////////////////////
```

```
//32 位除以 16 位
```

```
////////////////////////////////////
```

```
unsigned long res;
unsigned long dat1;
unsigned int dat2;
```

```
P_SW2 |= 0x80; //访问扩展寄存器 xsfr
MD3U32 = dat1; //dat1 用户给定
MD5U16 = data2; //dat2 用户给定
ARCON = 6 << 5; //32 位/16 位,除法模式
OPCON = 1; //启动计算
while((OPCON & 1) != 0); //等待计算完成
res = MD3U32; //32 位商, 16 位余数在 MD5U16 中
```

```
////////////////////////////////////
```

//左移或右移:

////////////////////////////////////

unsigned long res;
unsigned long dat1;
unsigned char num;

//移位的位数, 用户给定

MD3U32 = dat1;
ARCON = (2 << 5) + num;
//ARCON = (1 << 5) + num;
OPCON = 1;
while((OPCON & 1) != 0);
res = MD3U32;

//dat1 用户给定
//32 位左移模式
//32 位右移模式
//启动计算
//等待计算完成
//32 位结果

STC MCU

附录A 编译器（汇编器）/仿真器/头文件使用指南

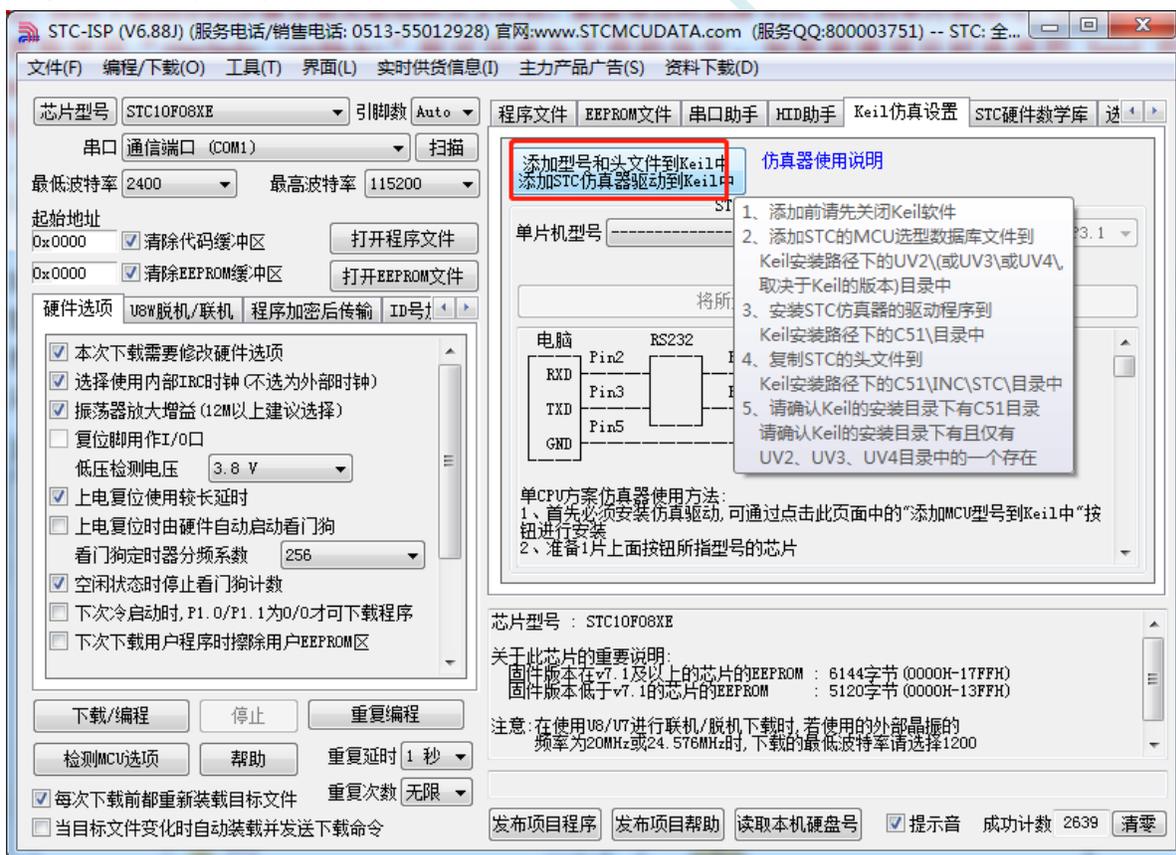
A: STC 单片机应使用何种编译器/汇编器？

Q: 任何老式的 8051 编译器/汇编器都可以支持，现流行使用 Keil C51

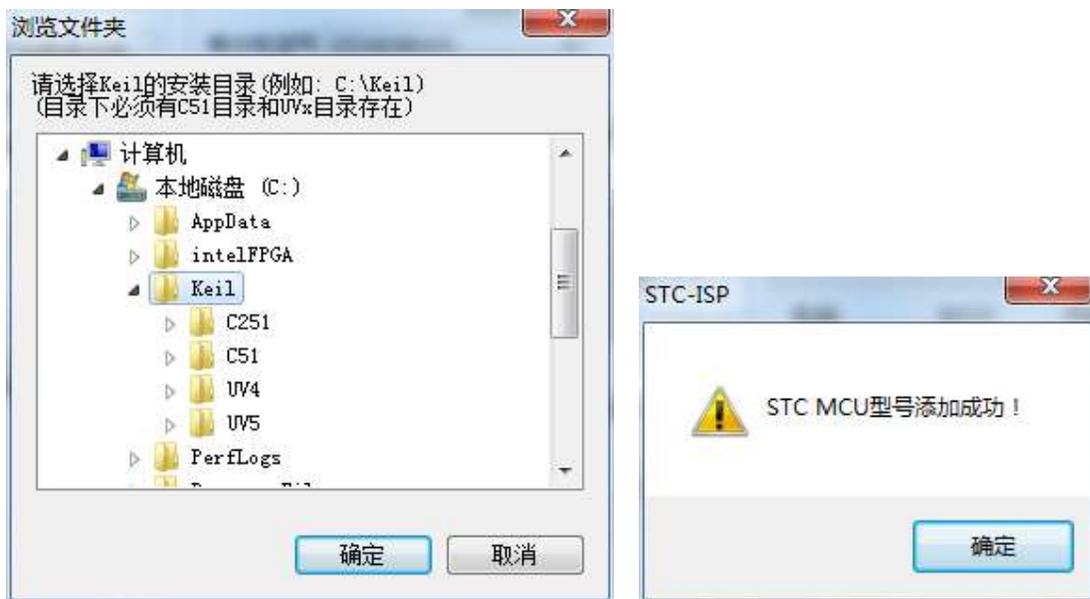
A: Keil 环境中，应如何包含头文件

Q: 按照下面图示的步骤安装完驱动和头文件后，新建项目时选择 STC 相应的单片机型号，在源文件中直接使用“`#include <STC8C.h>`”即可完成头文件的包含。如果新建项目时选择的 Intel 的 8052/87C52/87C54/87C58 或 Philips 的 P87C52/P87C54/P87C58 编译，头文件包含`<reg51.h>`即可，不过 STC 新增的特殊功能寄存器则需要用户自己声明。

1、安装 Keil 版本的仿真驱动

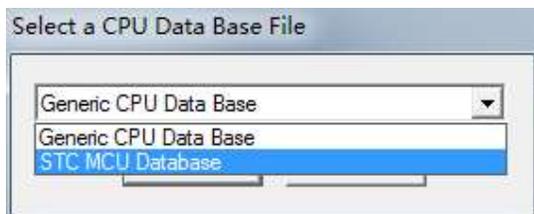


如上图，首先选择“Keil 仿真设置”页面，点击“添加 MCU 型号到 Keil 中”，在出现的如下的目录选择窗口中，定位到 Keil 的安装目录（一般可能为“C:\Keil\”），“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装 STC 的 Monitor51 仿真驱动 STCMON51.DLL，驱动与头文件的安装目录如上图所示。

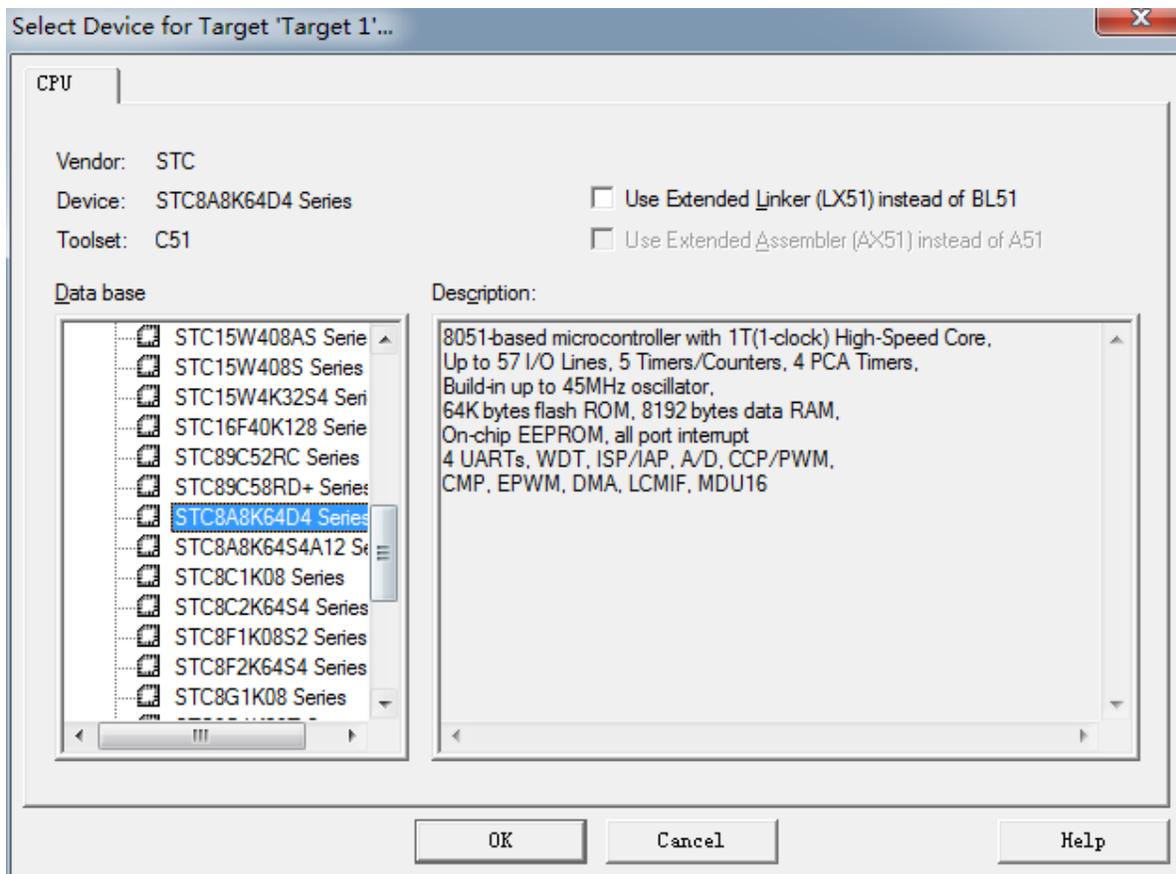


2、在 Keil 中创建项目

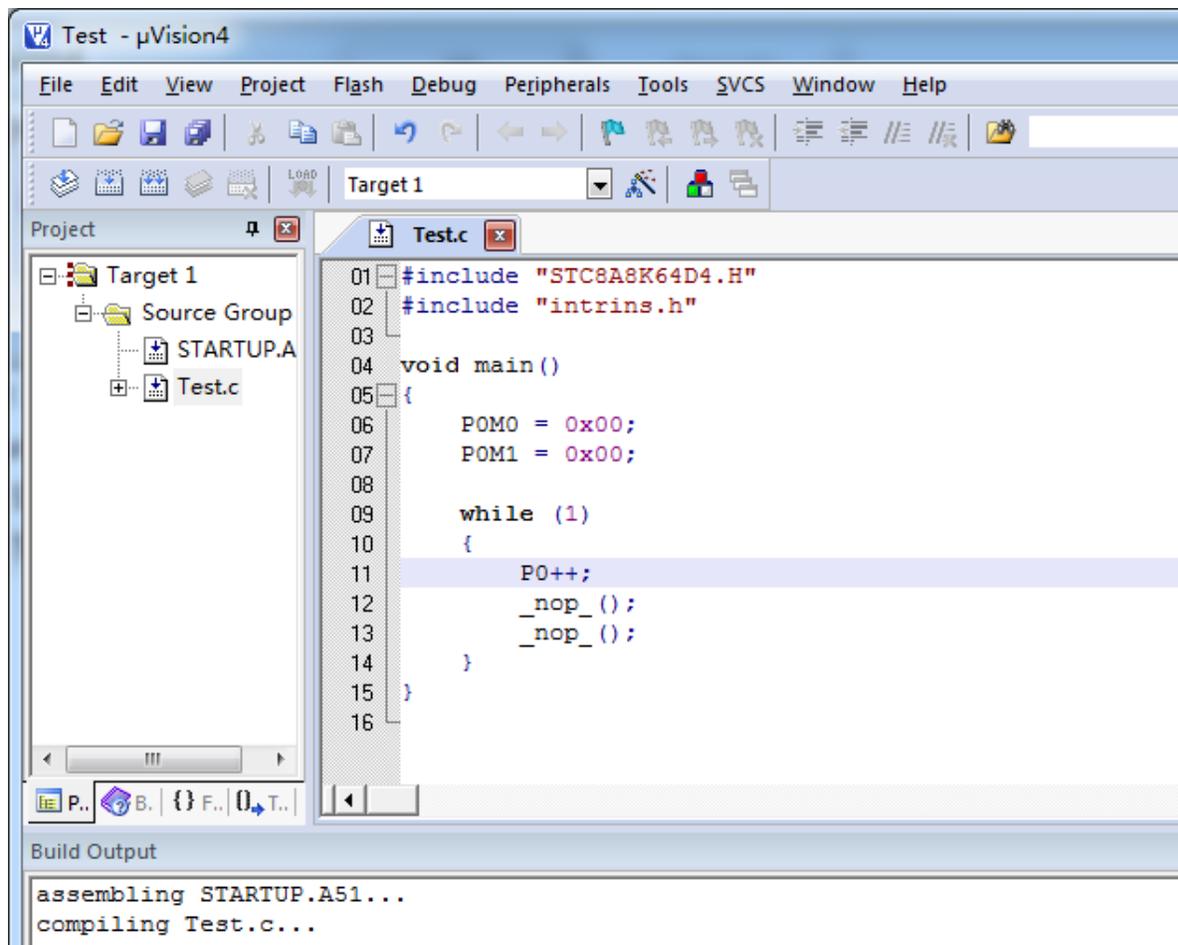
若第一步的驱动安装成功，则在 Keil 中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的 MCU 型号，我们在此选择“STC8A8K64D4”的型号，点击“确定”完成选择



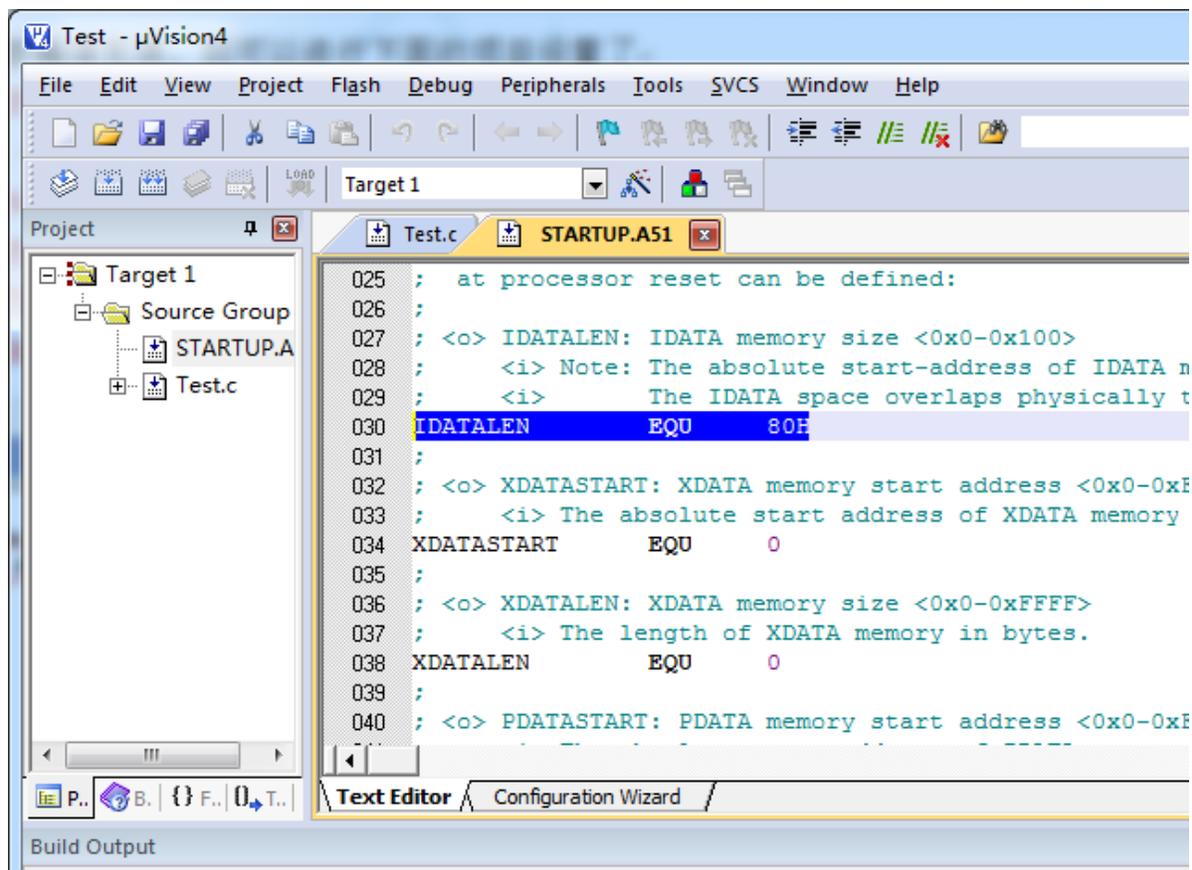
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

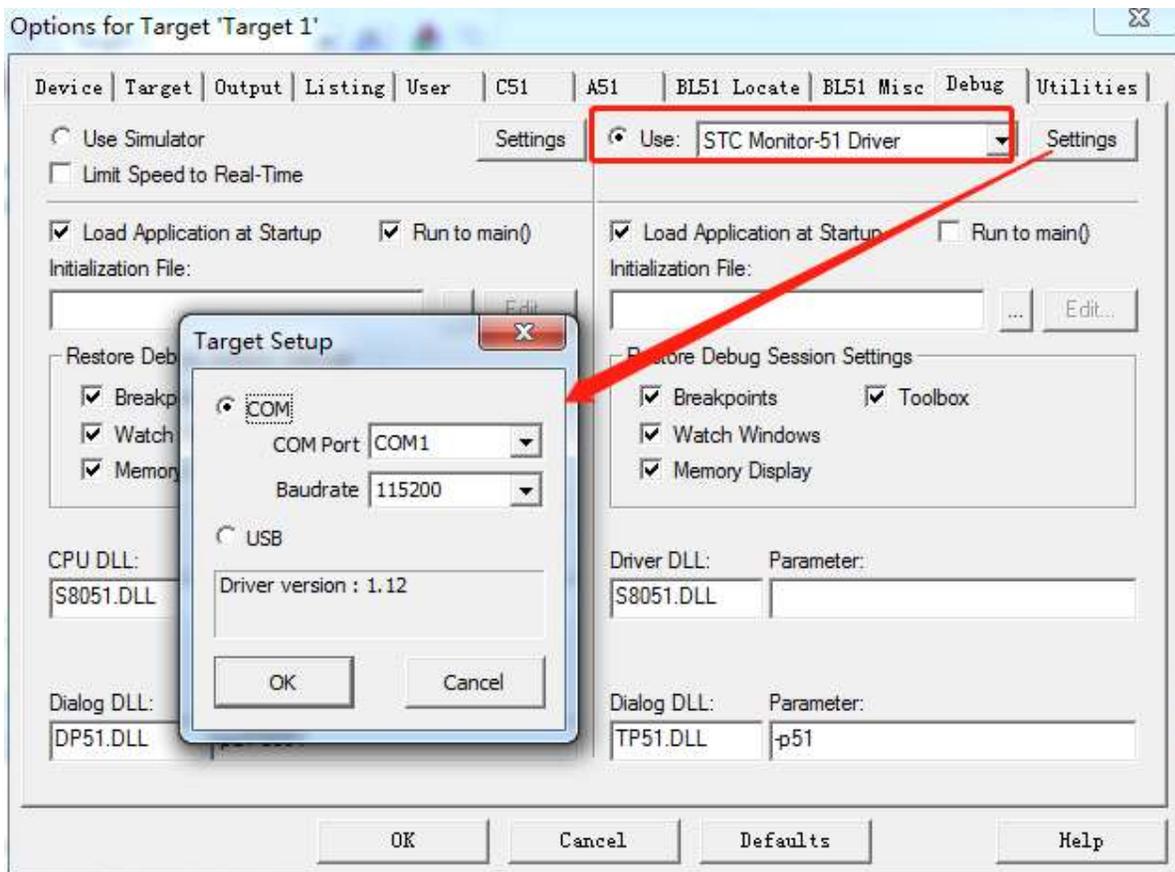
附加说明一点：

当创建的是 C 语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义 IDATA 大小的一个宏，默认值是 128，即十六进制的 80H，同时它也是启动文件中需要初始化为 0 的 IDATA 的大小。所以当 IDATA 定义为 80H，那么 STARTUP.A51 里面的代码则会将 IDATA 的 00-7F 的 RAM 初始化为 0；同样若将 IDATA 定义为 0FFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



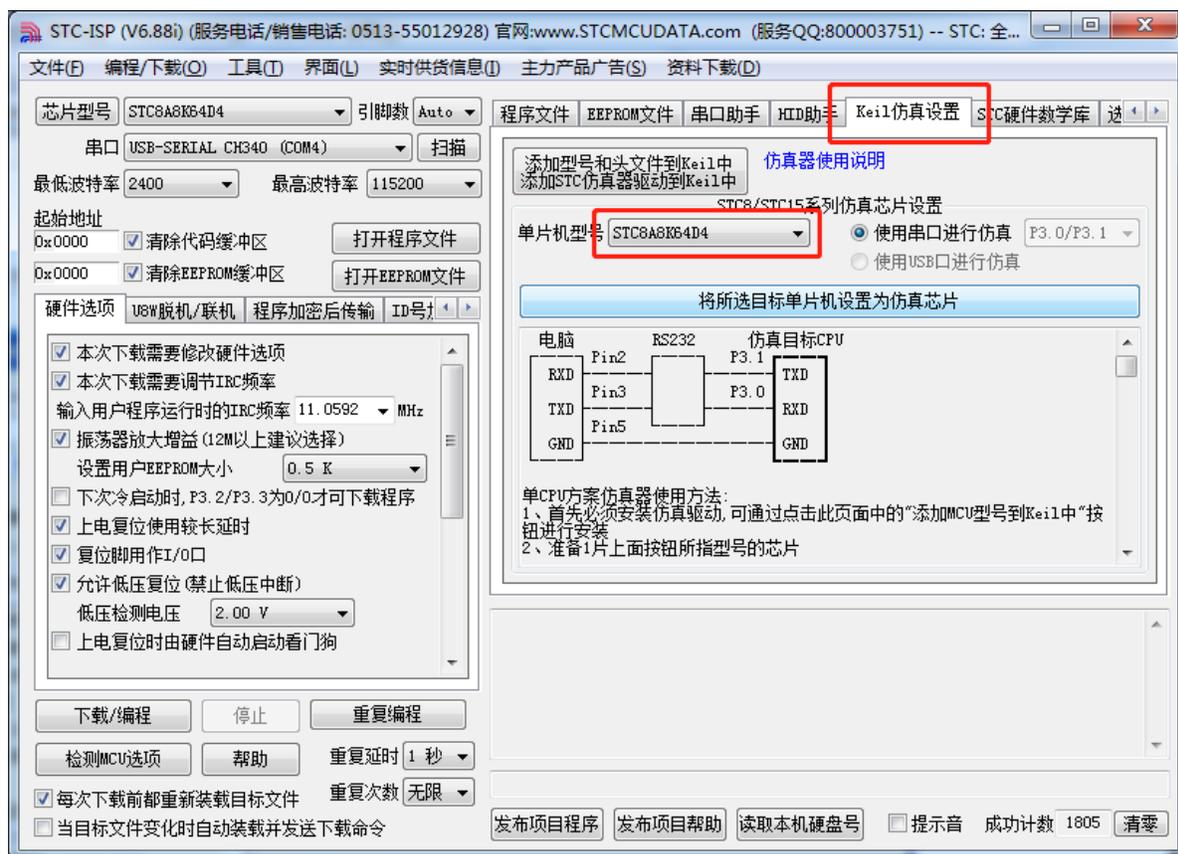
虽然 STC8 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

3、项目设置，选择 STC 仿真驱动



如上图，首先进入到项目的设置页面，选择“Debug”设置页，第2步选择右侧的硬件仿真“Use ...”，第3步，在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项，然后点击“Settings”按钮，进入下面的设置画面，对串口的端口号和波特率进行设置，波特率一般选择 115200。到此设置便完成了。

4、创建仿真芯片



准备一颗 STC8A 系列或者 STC8F 系列的芯片，并通过下载板连接到电脑的串口，然后如上图，选择正确的芯片型号，然后进入到“Keil 仿真设置”页面，点击相应型号的按钮，当程序下载完成后仿真器便制作完成了。

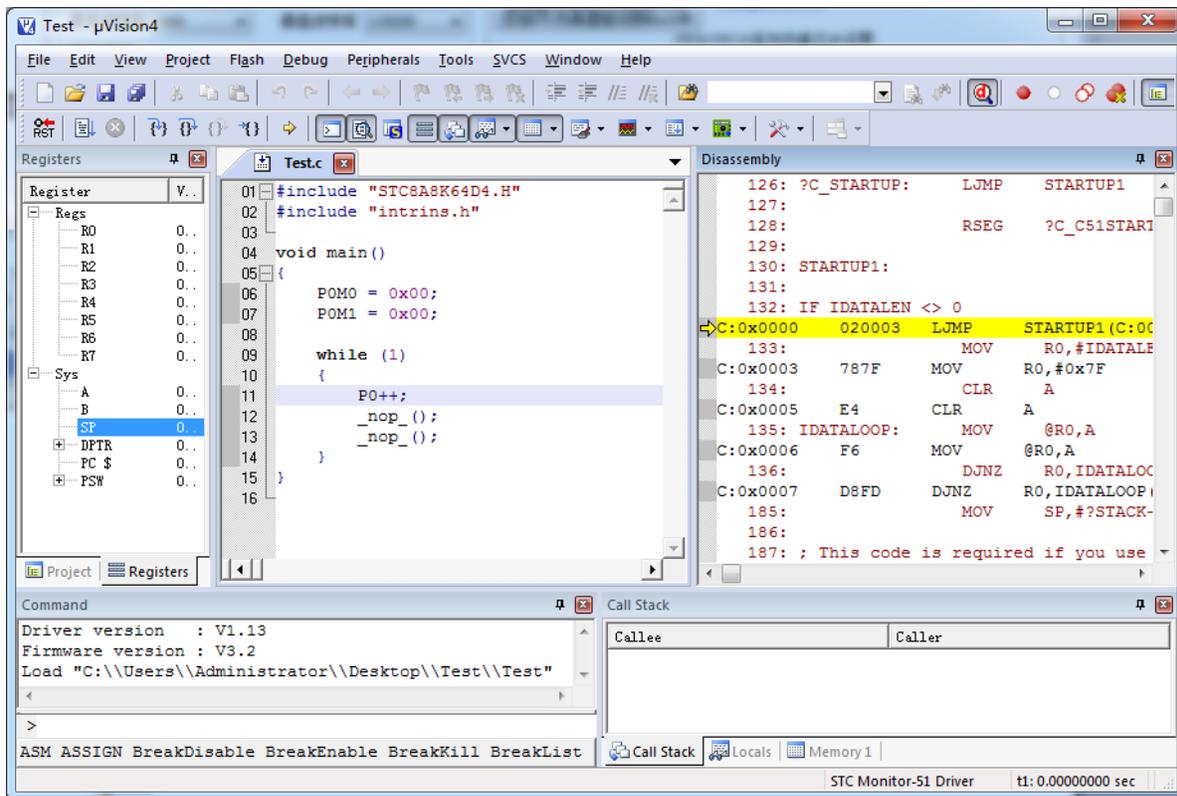
5、开始仿真

将制作完成的仿真芯片通过串口与电脑相连接。

将前面我们所创建的项目编译至没有错误后，按“Ctrl+F5”开始调试。

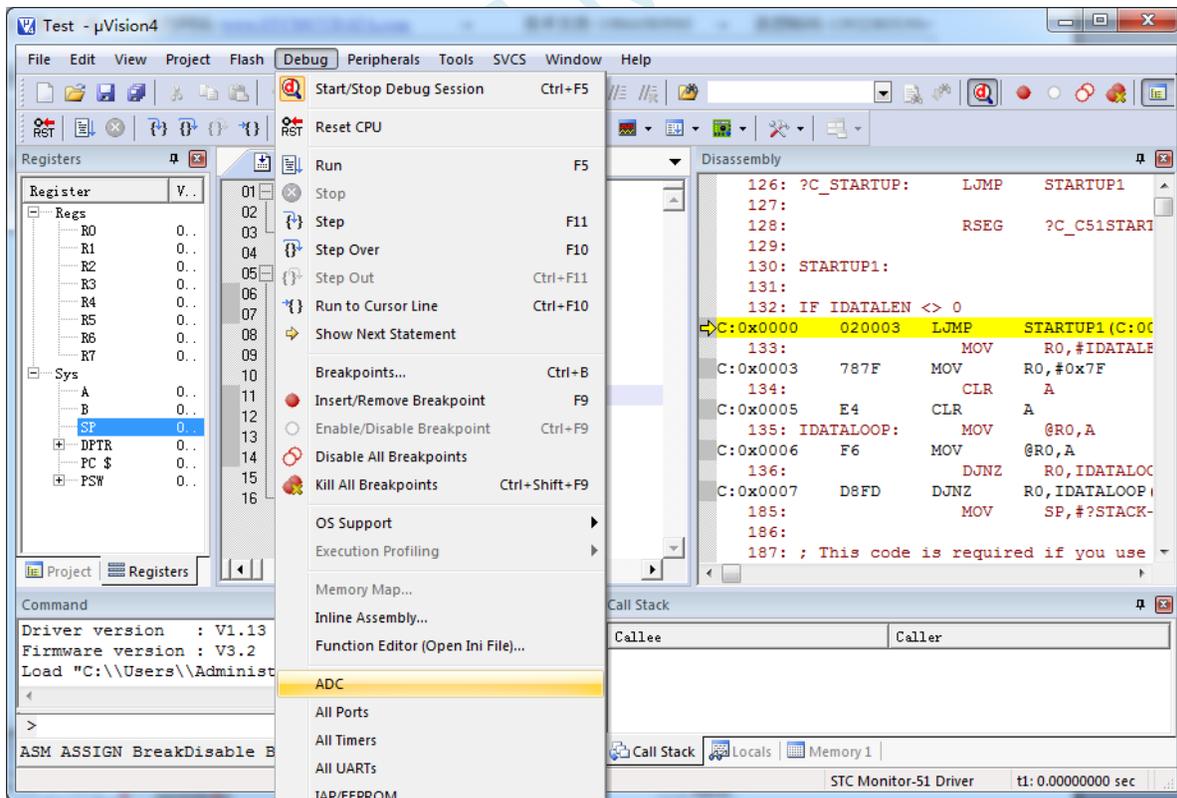
若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号

断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。



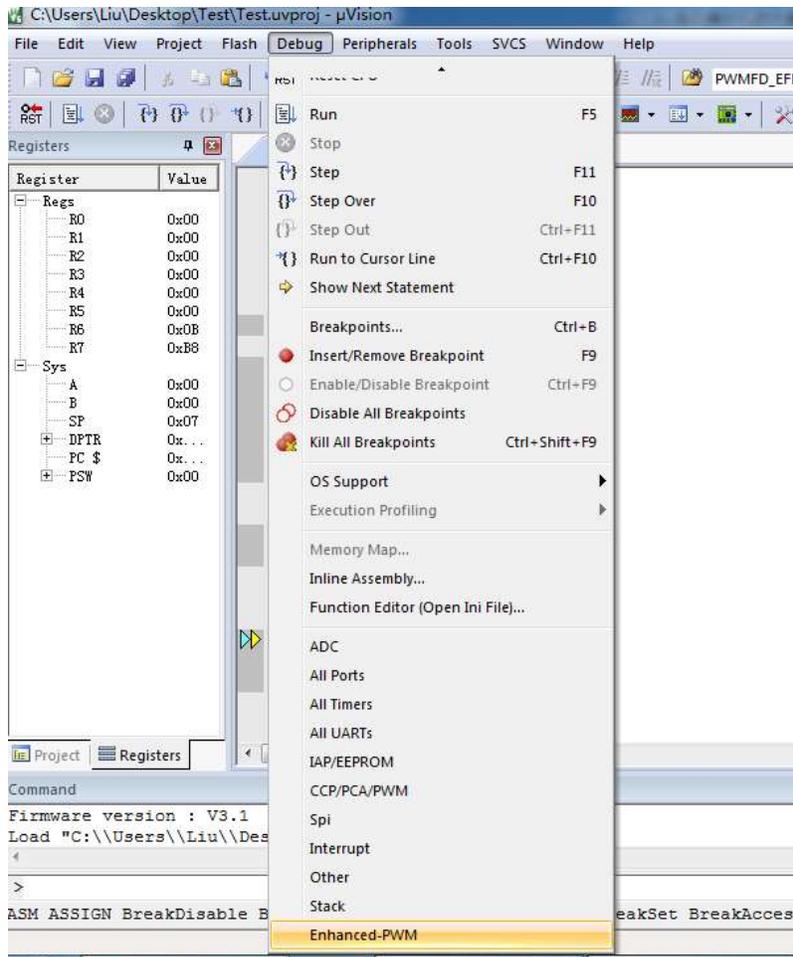
6、仿真过程中，寄存器的查看

在仿真的过程中，可查看 MCU 相关的寄存器。所有的寄存器列表在“Debug”菜单的底端。如下图所示：



在上图“Debug”菜单的最底端，还有一个黑色的小三角，这表示还有隐藏的项目（主要是由于显示版面大小的原因）

将鼠标仿真小三角上即可自动拖出所有的项，如下图：



仿真注意事项：

- 1、仿真监控程序占用 P3.0/P3.1 两个端口，但不占用串口 1，用户可以将串口 1 切换到 P3.6/P3.7 或者 P1.6/P1.7 再使用
- 2、仿真监控程序占用内部扩展 RAM(XDATA)的最后 768 字节，用户不可对这个区域的 XDATA 进行写操作

附录B STC-ISP 下载软件高级应用

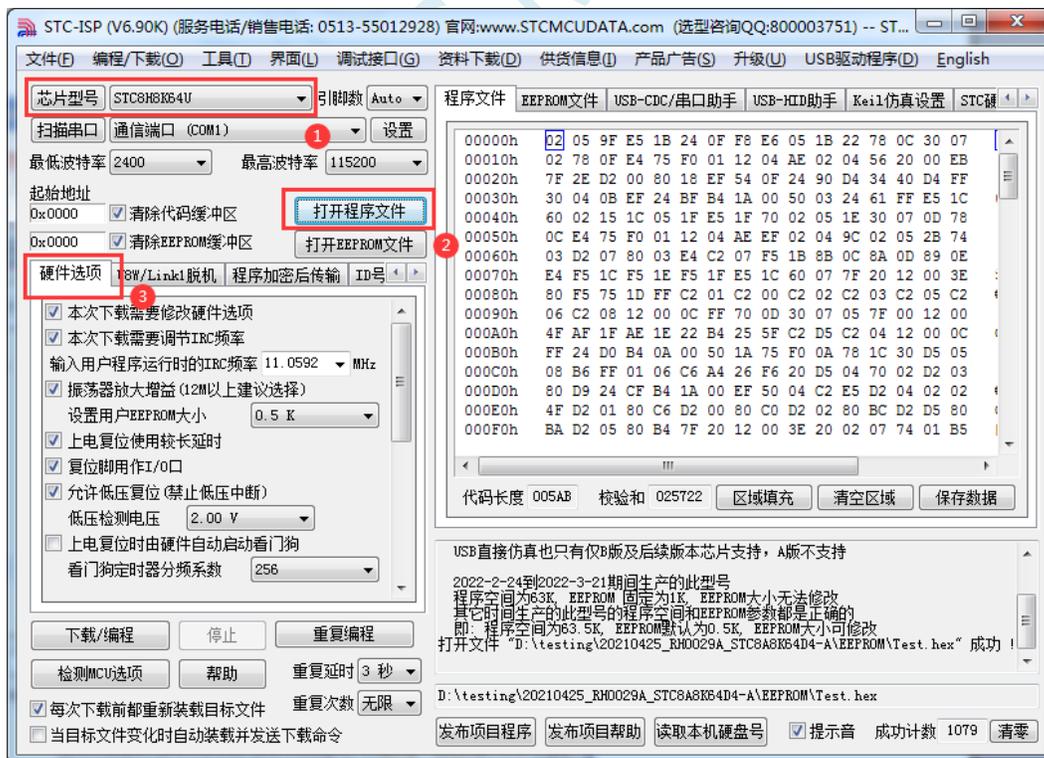
B.1 发布项目程序

发布项目程序功能主要是将用户的程序代码与相关的选项设置打包成为一个可以直接对目标芯片进行下载编程的超级简单的用户自己界面的可执行文件。

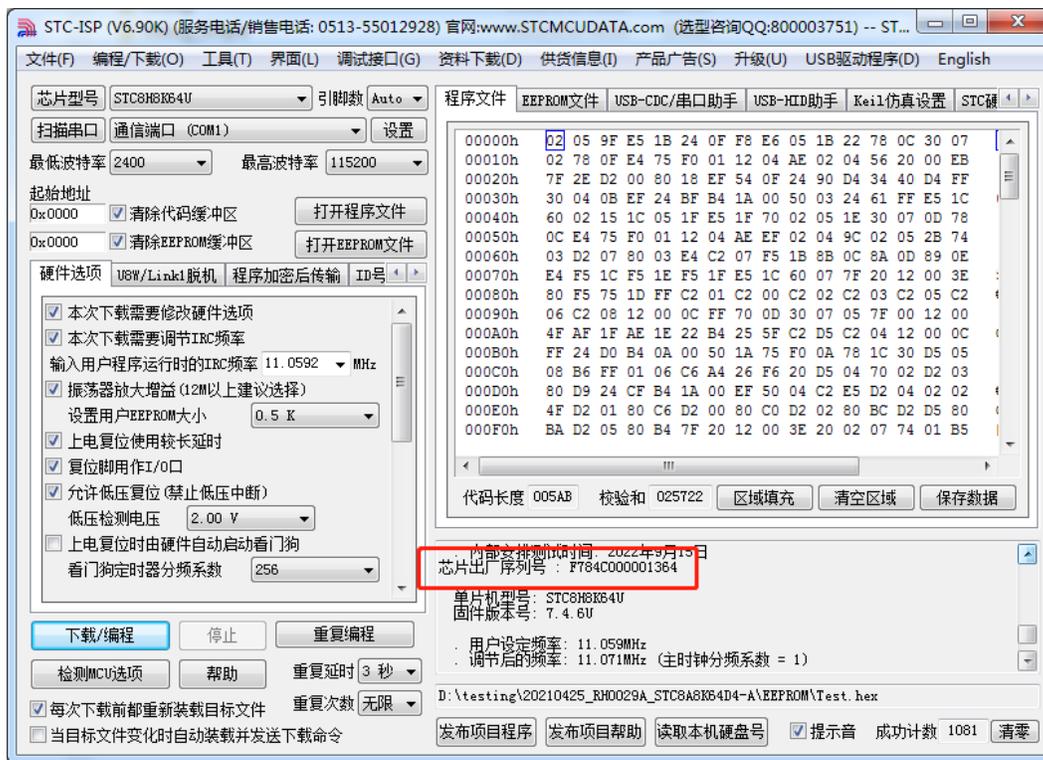
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘拷走，防不胜防，当然盗走你的电脑那就没办法那，所以 STC 的脱机下载工具比电脑烧录安全，能限制可烧录芯片数量，让前台文员小姐烧，让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---坦克，可以发给客户自己升级，不需冒着生命危险跑到战火纷飞的伊拉克升级软件啦），对于 ID 号不一致的其它芯片，不能进行下载编程。

发布项目程序详细的操作步骤如下：

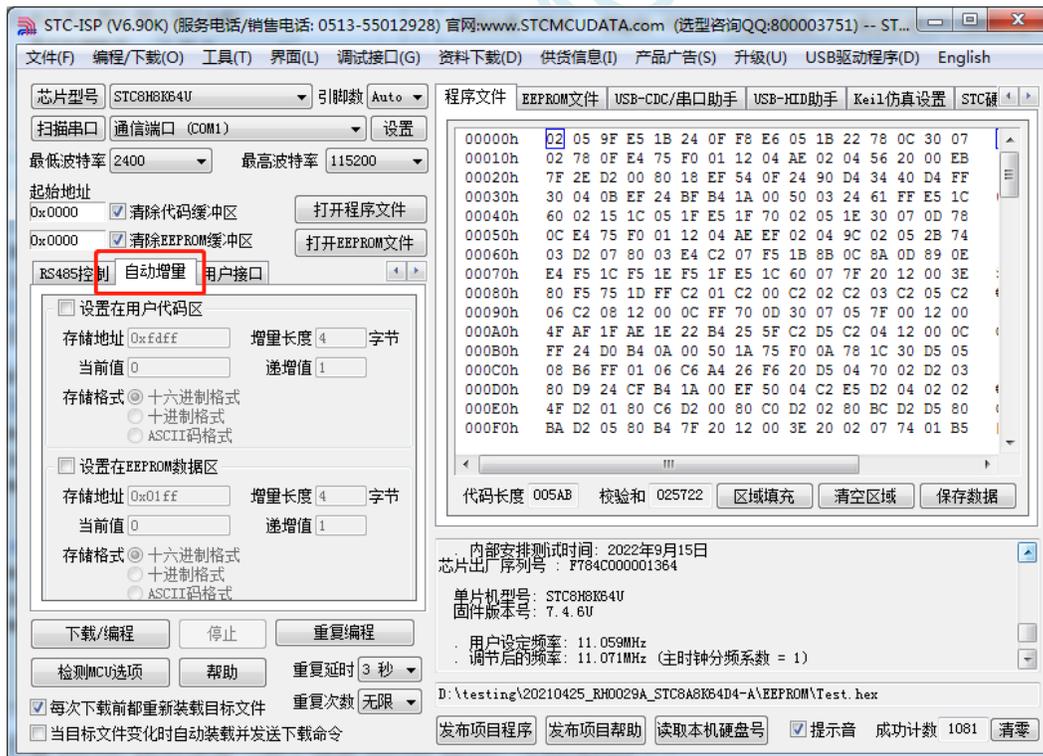
- 1、首先选择目标芯片的型号
- 2、打开程序代码文件
- 3、设置好相应的硬件选项



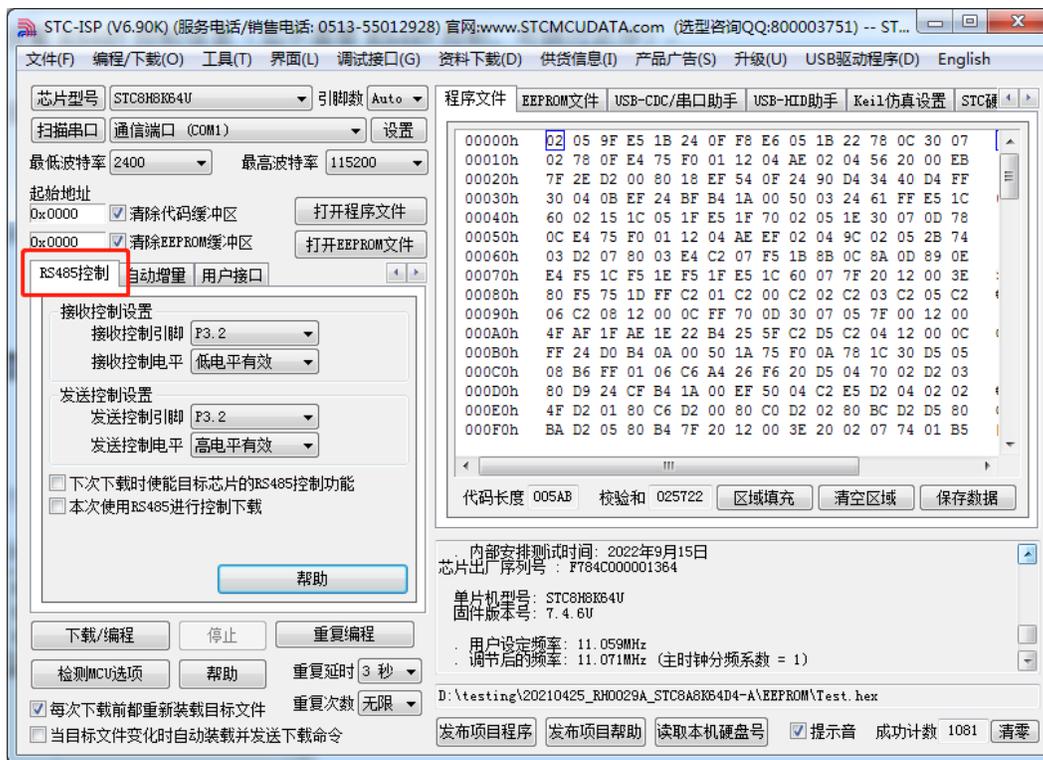
- 4、试烧一下芯片，并记下目标芯片的 ID 号，如下图所示，该芯片的 ID 号即为“F784C000001364”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



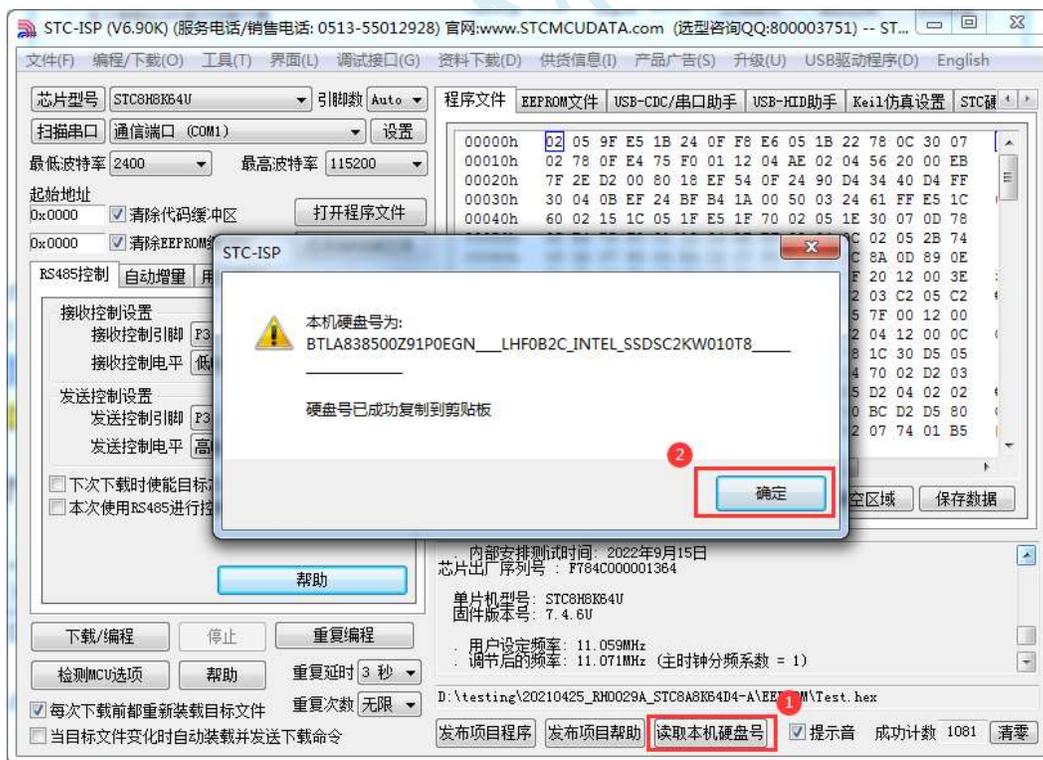
5、设置自动增量（如不需要自动增量，可跳过此步）



6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



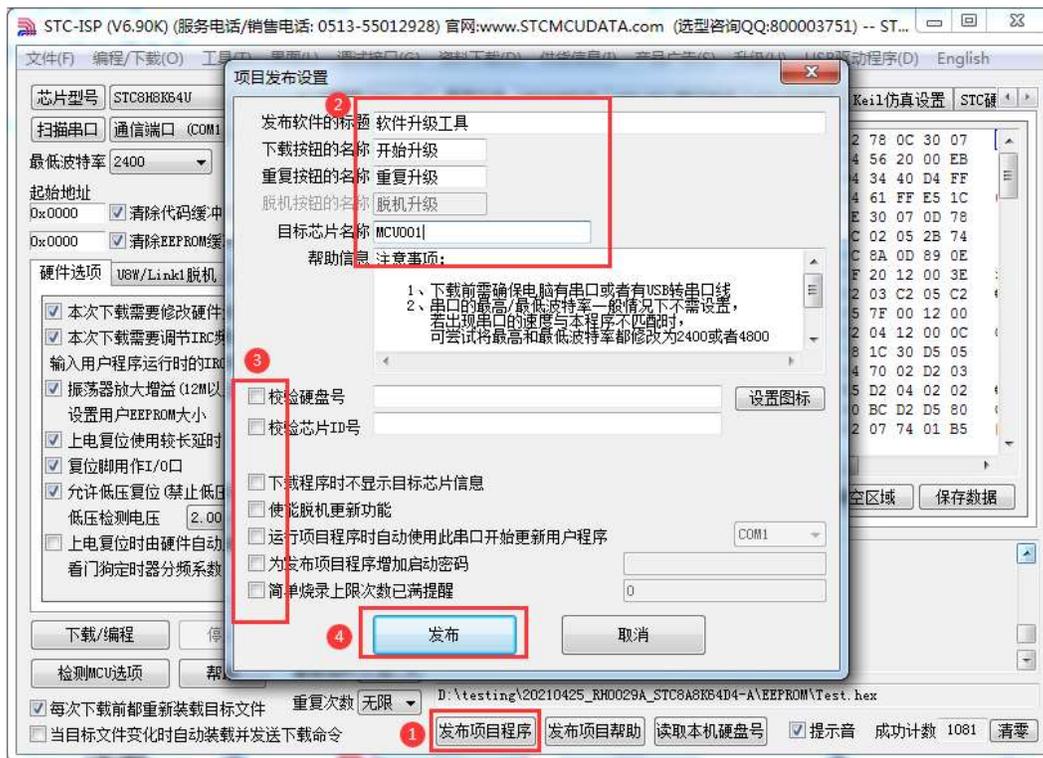
- 7、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）



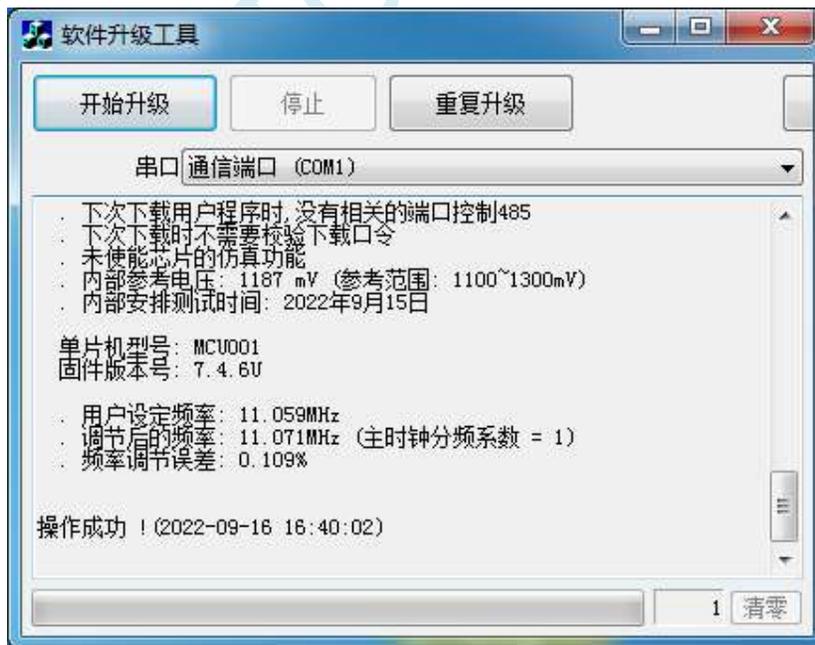
- 8、点击“发布项目程序”按钮，进入发布应用程序的设置界面。
 9、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息
 10、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记

下的目标电脑的硬盘号

- 11、若需要校验目标芯片的 ID 号，则需要勾选上“校验芯片 ID 号”，并在后面的文本框内输入前面所记下的目标芯片的 ID 号



- 12、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。发布的项目程序打界面如下图



B.2 程序加密后传输（防烧录时串口分析出程序）

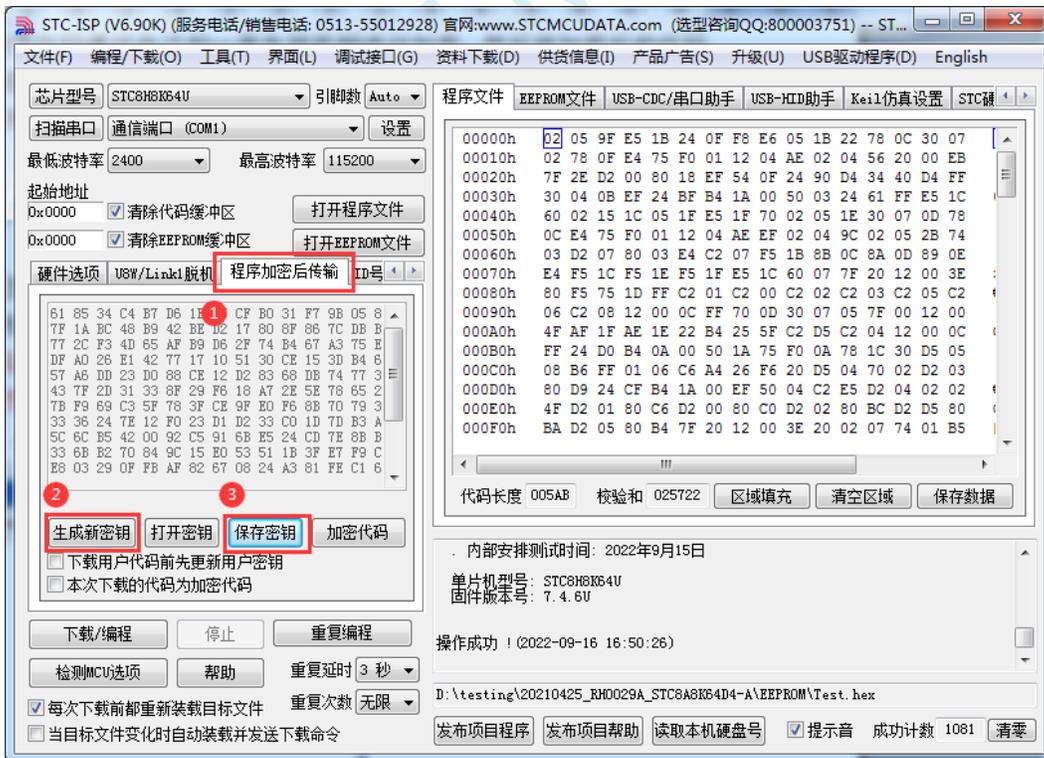
目前，所有的普通串口下载烧录编程都是采用**明码通信**的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。当然用 **STC 的脱机下载板烧程序总比用电脑烧程序强**（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘拷走，防不胜防，当然盗走你的电脑那就没办法那，所以 **STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以**）。即使是 **STC 全球首创的脱机下载工具**，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据分析出实际的用户程序代码，也是没有办法达到要求的，这就需要用到最新的 **STC 单片机所提供的程序加密后传输功能**。

程序加密后传输下载是用户先将程序代码通过自己的一套专用密钥进行加密，然后将加密后的代码再通过串口下载，此时下载传输的是加密文件，通过串口分析出来的是加密后的乱码，如不通过派人潜入你公司盗窃你电脑里面的加密密钥，就无任何价值，便可起到防止在烧录程序时被烧录人员通过监测串口分析出代码的目的。

程序加密后传输功能的使用需要如下的几个步骤：

1、生成并保存新的密钥

如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。

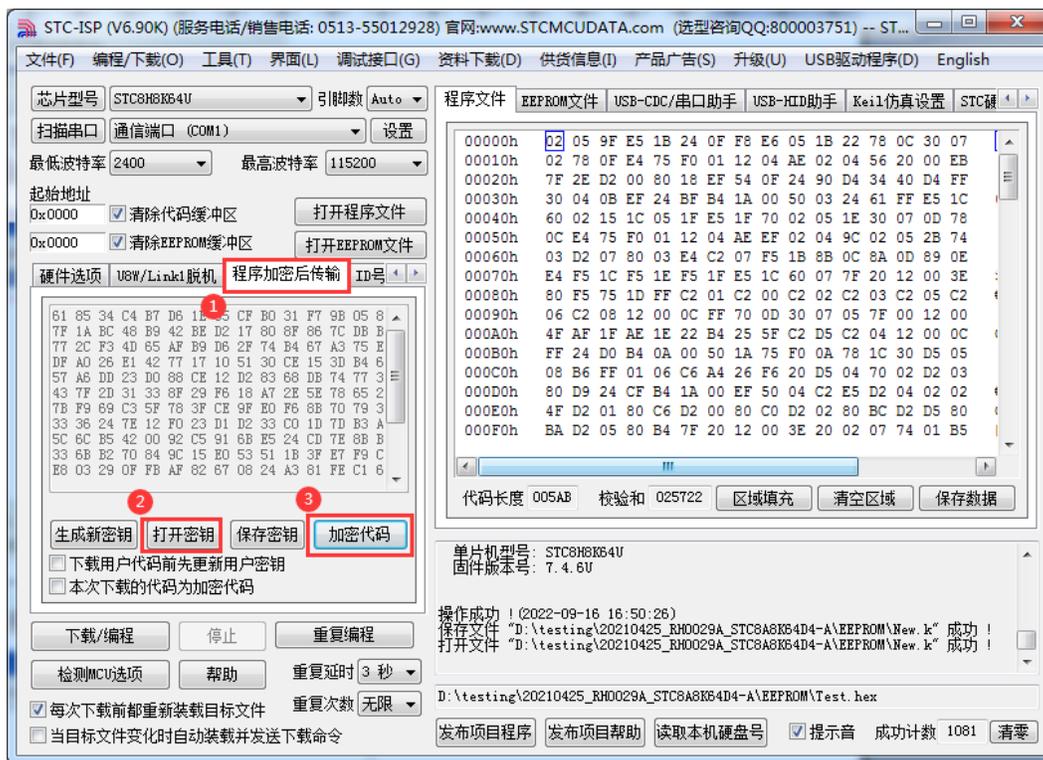


2、对代码文件加密

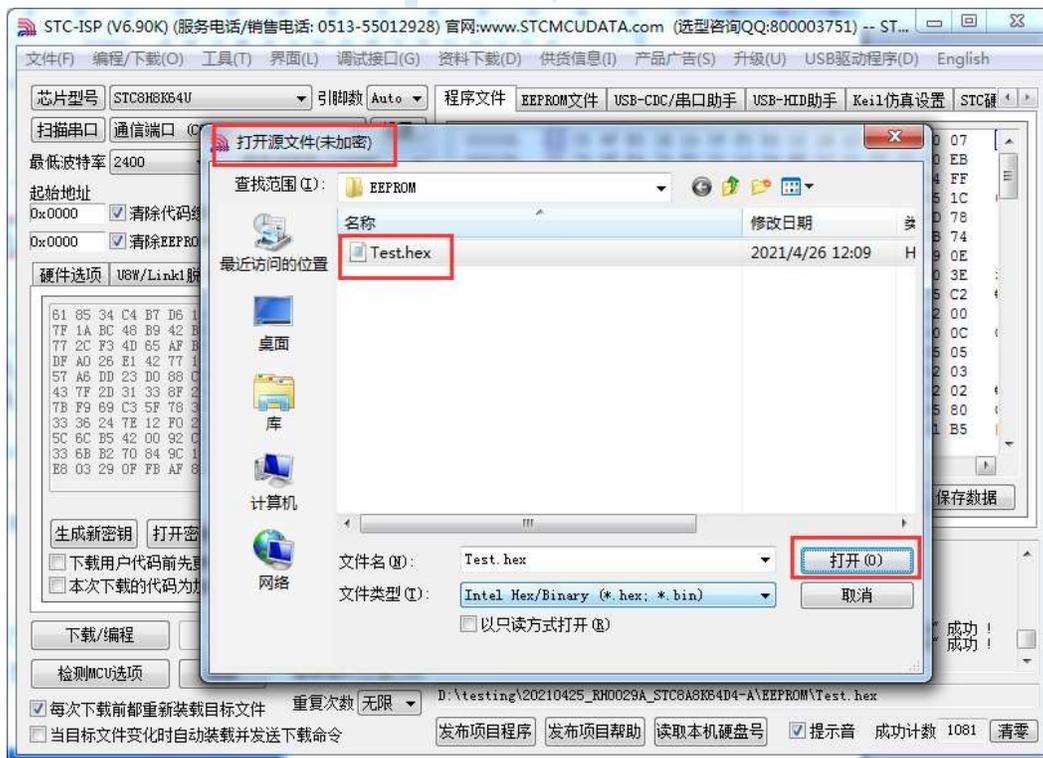
加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。

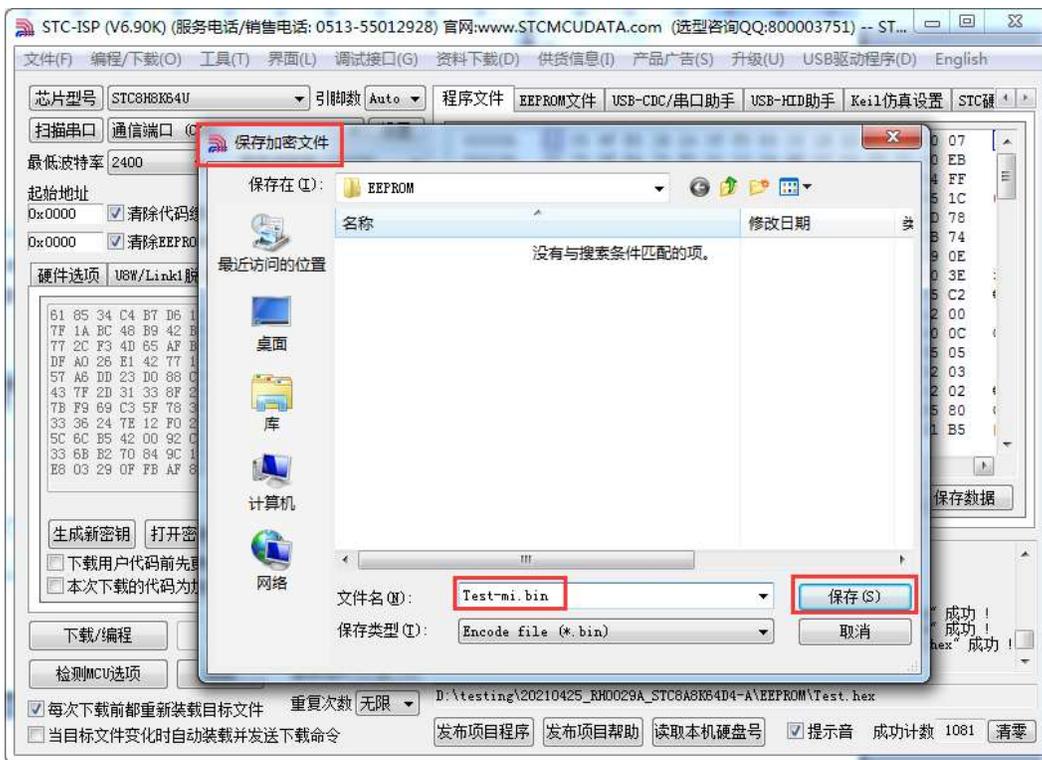
如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，

例如“New.k”，然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件



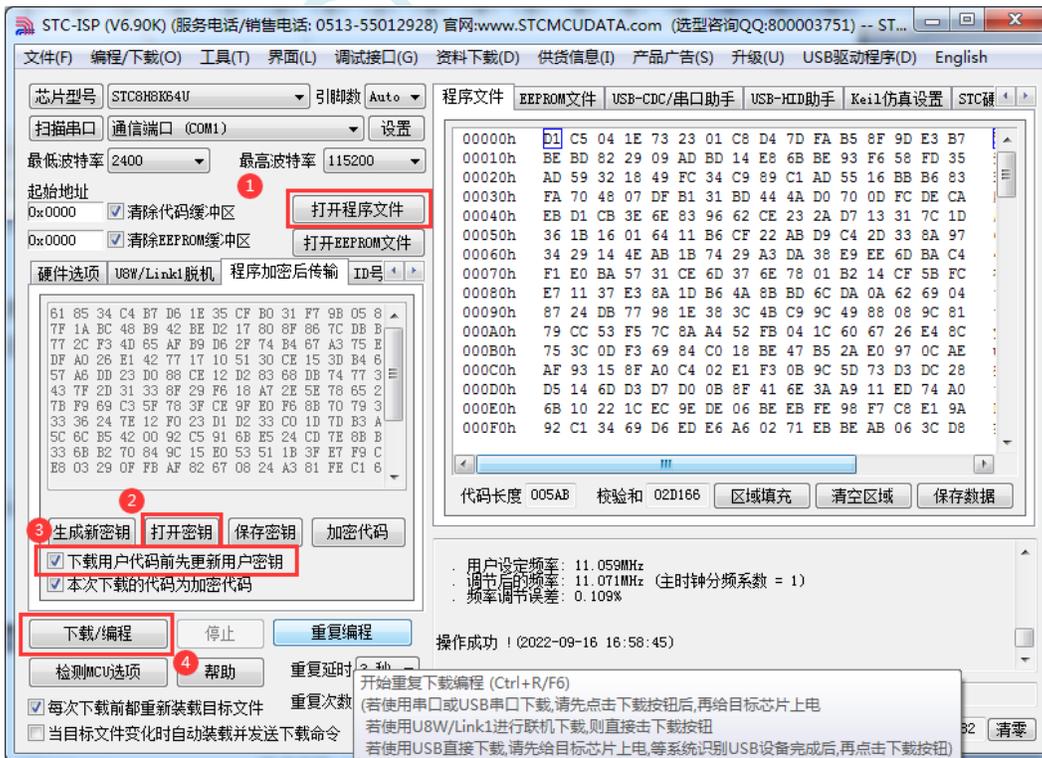
点击打开按钮后，马上会有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。





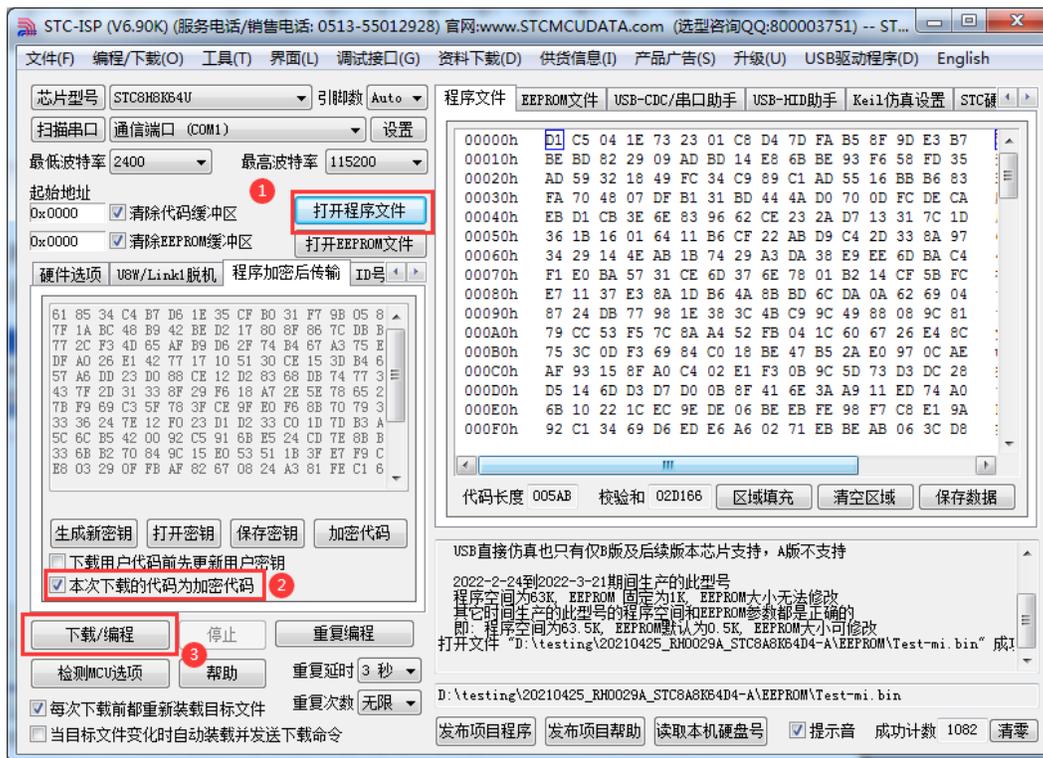
3、将用户密钥更新到目标芯片中

更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



4、加密更新用户代码

密钥更新成功后，目标芯片便具有接收加密代码并还原的功能。此时若需要再次升级/更新代码，则只需要参考第二步的方法，将目标代码进行加密，然后如下图



对于一片新的 STC 单片机，可将步骤 3 和步骤 4 合并完成，即将密钥更新到目标单片机的同时也可将加密后的代码一并下载到单片机中，若已经执行过步骤 3（即已经将密钥更新到目标芯片中了），则后续的代码更新就只需要按照步骤 4，只需要在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项（“下载用户代码前先更新用户密钥”选项不需要选了），然后打开我们之前加过加密后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的（防止在烧录程序时被烧录人员通过监测串口分析出代码的目的）。

B.3 发布项目程序+程序加密后传输结合使用

发布项目程序与程序加密后传输两项新的特殊功能可以结合在一起使用。首先程序加密后传输可以确保用户代码在烧录编程时串口通信传输过程当中的保密性，而发布项目程序可实现让最终使用者远程升级功能（方案公司的人员不需要亲自到场）。所以两项功能结合起来使用，非常适用于方案公司/生产商在软件需要更新时，让最终使用者自己对终端产品进行软件更新的目的，又确保现场烧录人员无法通过串口分析出有用程序，强烈建议方案公司使用。

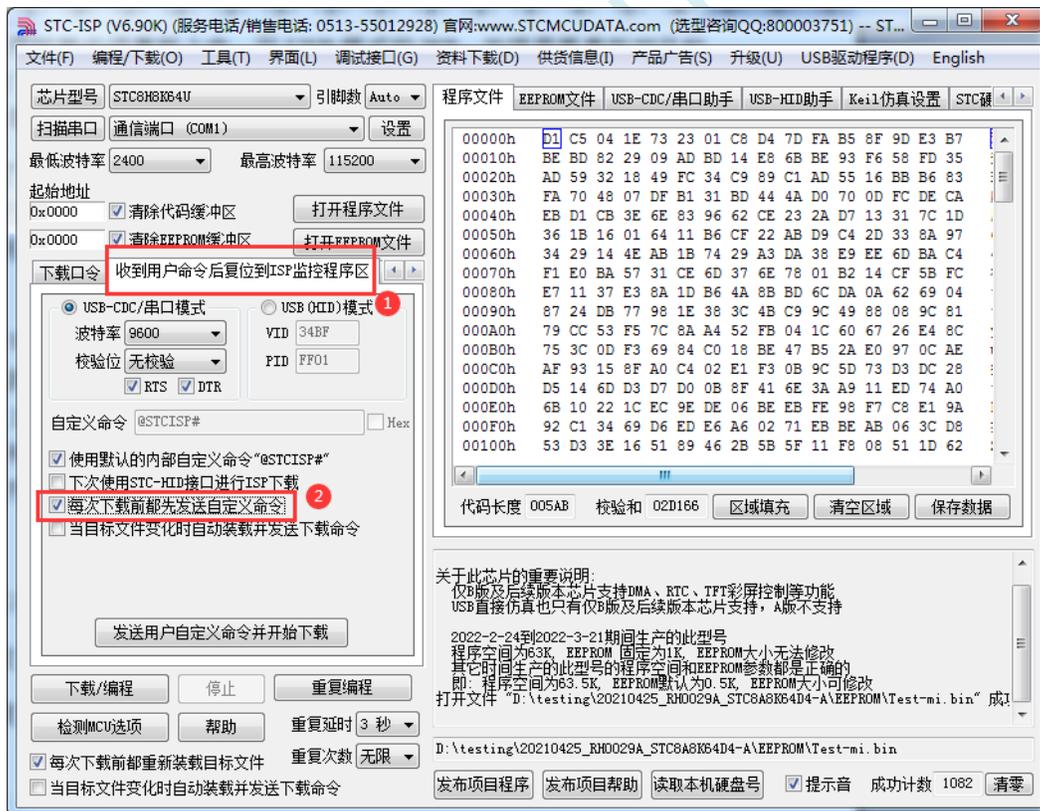
STC MCU

B.4 用户自定义下载（实现不停电下载）

将用户的目标程序下载到STC单片机是通过执行单片机内部的ISP系统代码和上位机进行串口或者USB通讯来实现的。但STC单片机内部的ISP系统代码只有在每次重新停电再上电时才会被执行，这就要求用户每次需要对目标单片机更新程序时就必须重新上电，而USB模式的ISP，处理需要重新对目标芯片上电外，还需要在上电时将P3.2口下拉到GND。对于处于开发阶段的项目，需要频繁的修改代码、更新代码，每次下载都需要重新上电会导致操作非常麻烦。

STC单片机在硬件设计时，增加了一个软复位寄存器（IAP_CONTR），让用户可以通过设置此寄存器来决定CPU复位后重新执行用户代码还是复位到ISP区执行ISP系统代码。当向IAP_CONTR寄存器写入0x20时，CPU复位后重新执行用户代码；当向IAP_CONTR寄存器写入0x60时，CPU复位后复位到ISP区执行ISP系统代码。

要实现不停电进行ISP下载，用户可以在程序中设计一段代码，例如检测一个特殊的按键、或者监控串口等待一个特殊的串口命令，当检测到满足下载条件时，就通过软件触发软复位寄存器复位到ISP区执行ISP系统代码，从而实现不停电ISP下载。当触发条件是外部按键时，则在用户代码中实时监控按键状态即可。若要实现STC-ISP软件和用户触发软复位完全同步，则需要使用STC-ISP软件中所提供的“收到用户命令后复位到ISP监控程序区”这个功能。



实现不停电 ISP 下载的步骤如下:

1、编写用户代码，并在用户代码中添加串口命令监控程序

(参考代码如下，测试单片机型号为 STC8H8K64U)

```
#include "stc8h.h"

#define FOSC 11059200UL
#define BAUD (65536 - FOSC/4/115200)

char code *STCISPCMD = "@STCISP#"; //自定义下载命令
char index;

void uart_isr() interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        dat = SBUF; //接收串口数据

        if (dat == STCISPCMD[index]) //判断接收的数据和当前的命令字符是否匹配
        {
            index++; //若匹配则索引+1
            if (STCISPCMD[index] == '\0') //判断命令是否配完成
                IAP_CONTR = 0x60; //若匹配完成则软复位到 ISP
        }
        else
        {
            index = 0; //若不匹配,则需要从头开始
            if (dat == STCISPCMD[index])
                index++;
        }
    }
}

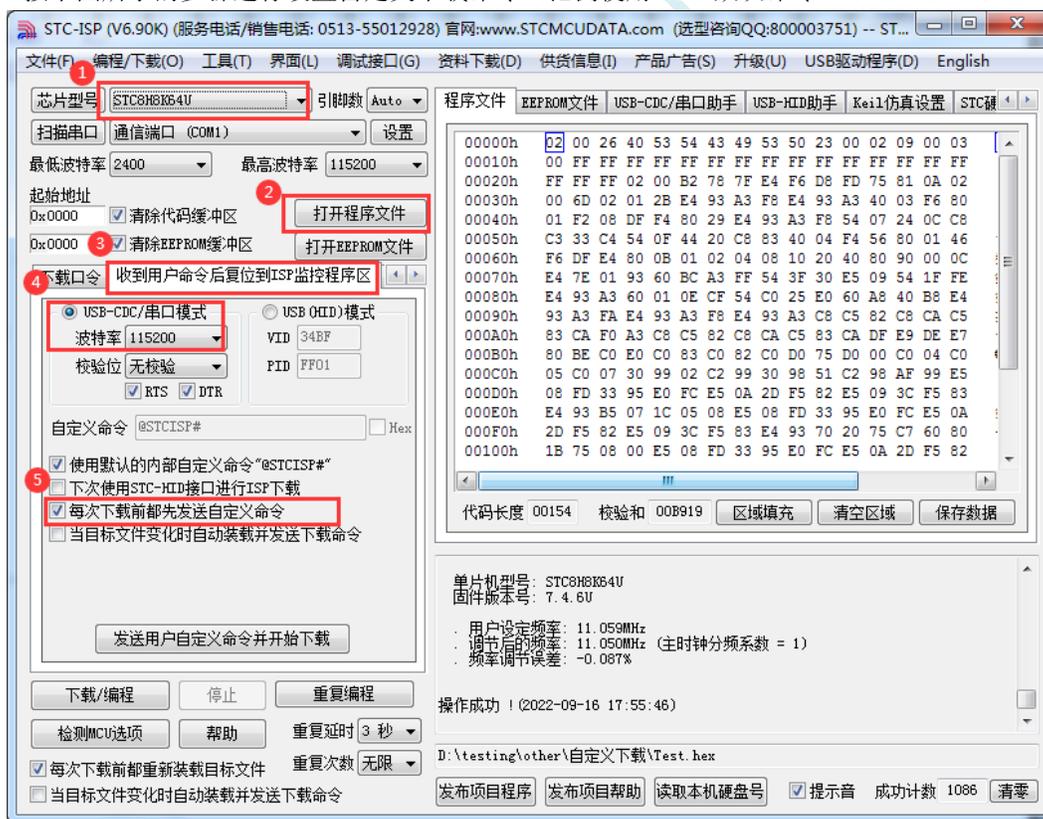
void main()
{
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
```

```
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
```

```
SCON = 0x50; //串口初始化
AUXR = 0x40;
TMOD = 0x00;
TH1 = BAUD >> 8;
TL1 = BAUD;
TR1 = 1;
ES = 1;
EA = 1;
```

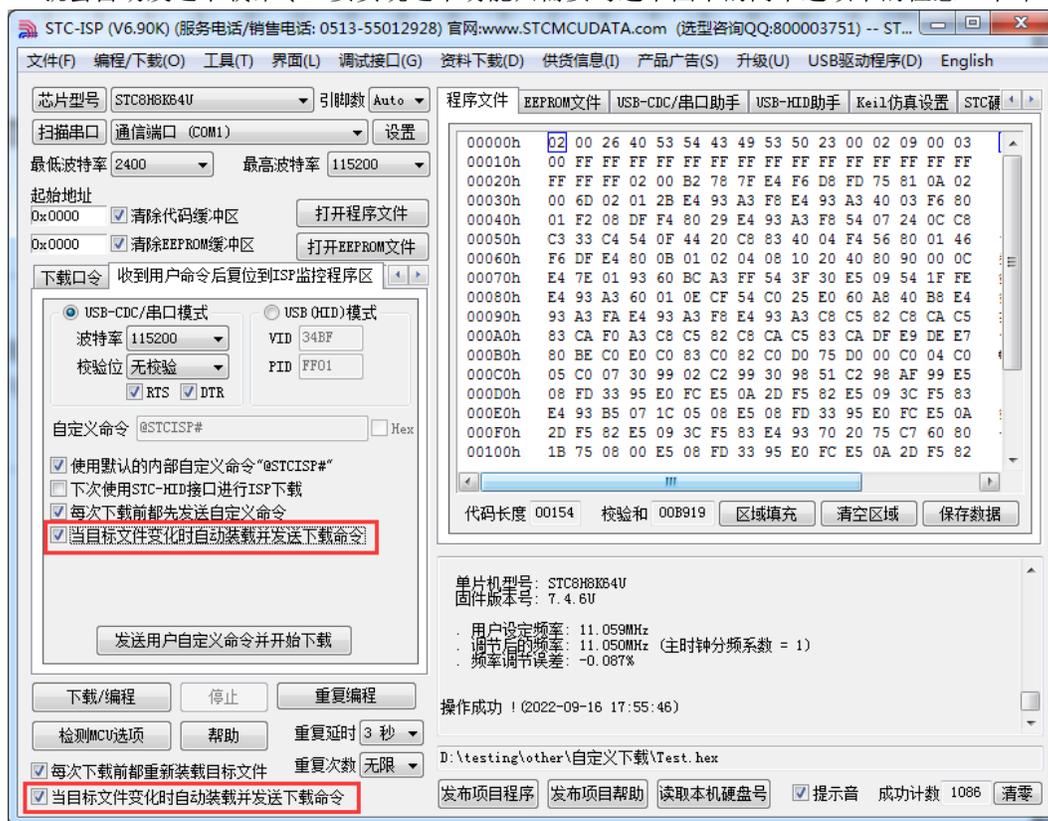
```
index = 0; //初始化命令
while (1);
}
```

2、按下图所示的步骤进行设置自定义下载命令（范例使用 STC 默认命令 “@STCISP#”）

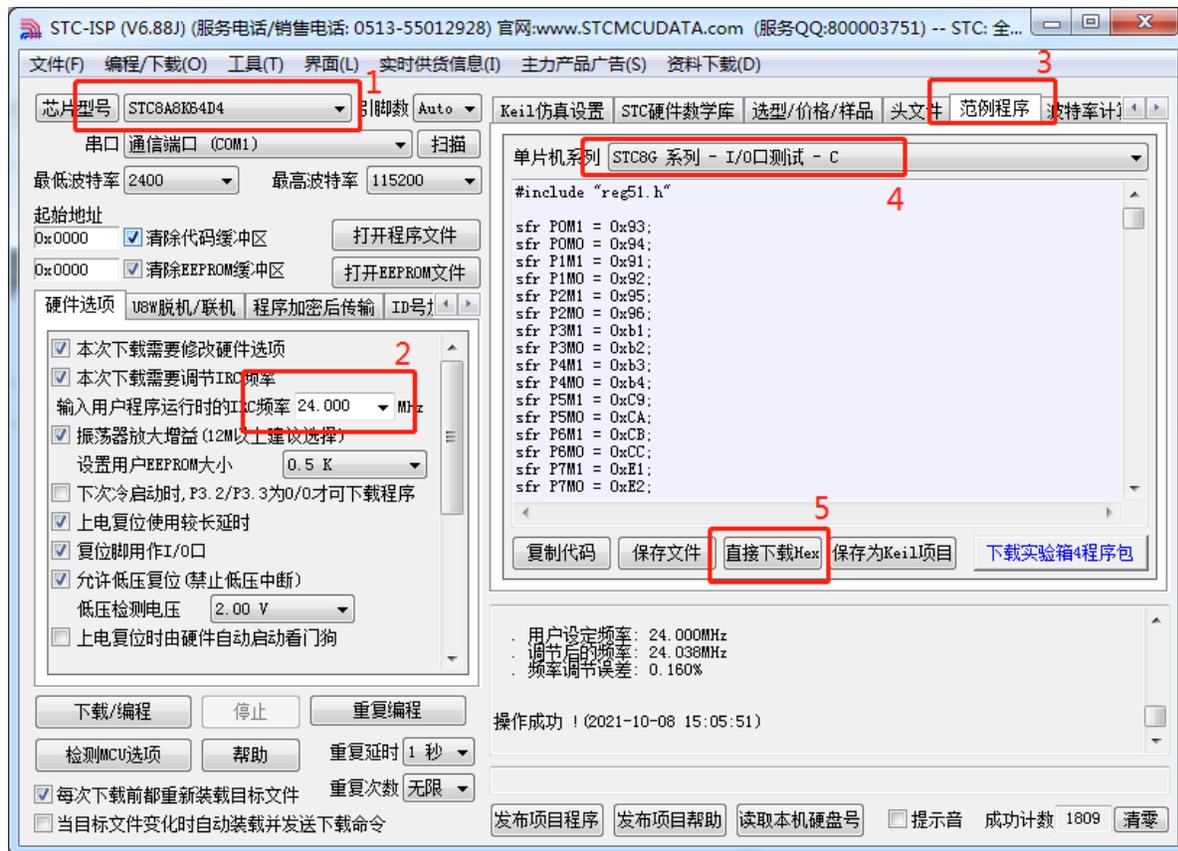


3、第一次下载时需要为目标单片机重新上电，之后的每次更新只需要点击下载软件中的“下载/编程”按钮，下载软件自动将下载命令发送给目标单片机，目标单片机接收到命令后自动复位到系统 ISP 区，即可实现不停电更新用户代码。

4、STC-ISP 还可实现项目开发阶段，完全自动下载功能，即当下载软件侦测到目标代码被更新了，就会自动发送下载命令。要实现这个功能只需要勾选下图中的两个选项中的任意一个即可



附录C 如何测试 I/O 口



测试 I/O 口步骤:

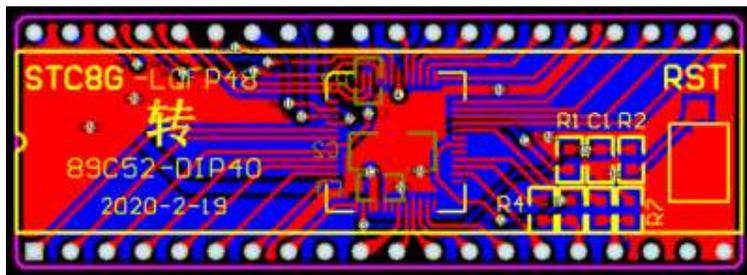
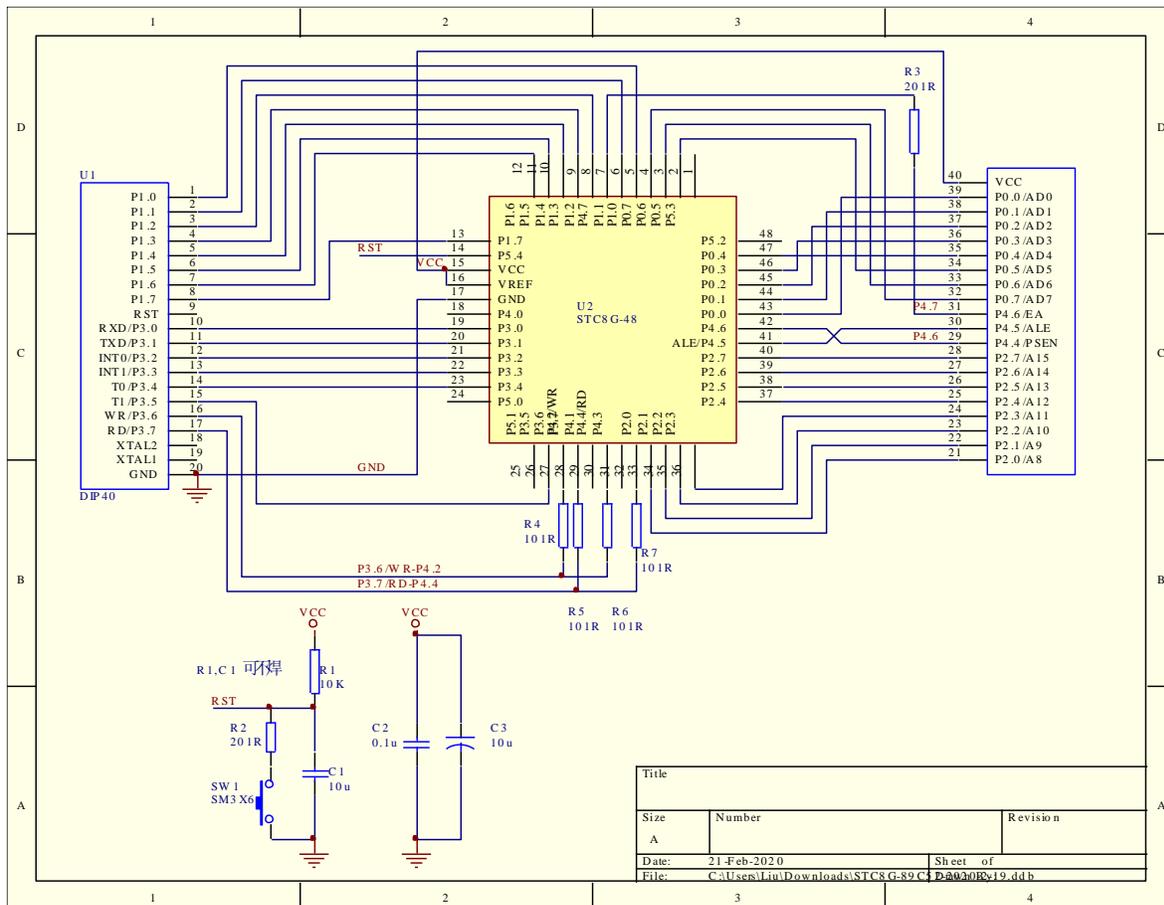
- 1、选择单片机型号
- 2、设置测试程序的工作频率（24MHz）
- 3、打开“范例程序”页
- 4、选择 STC8G 或者 STC8H 系列中的“I/O 口测试”程序
- 5、点击页面中的“直接下载 Hex”

下载完成后，会对所有的 I/O 口执行流水灯程序，此时可在 I/O 口接 LED 或者用示波器即可看到波形。

附录D 如何让传统的 8051 单片机学习板可仿真

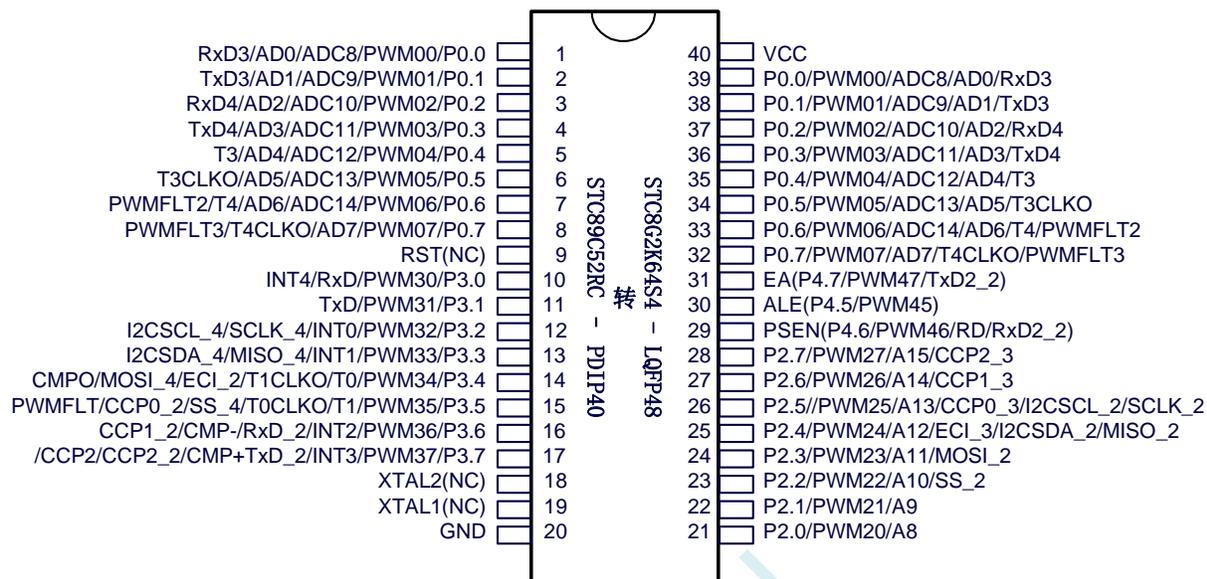
传统的 8051 单片机学习板不具有仿真功能，让传统的 8051 单片机学习板可仿真需要借助转换板，转换板的实物图如下图所示，转换后的引脚布排与传统 8051 的脚位基本一致，从而可以实现标准 8051 学习板的仿真功能。

下图是转换板的原理图和 PCB 版图



该转换板可进行 STC8A8K64D4 系列 LQFP48 转 STC89C52RC/STC89C58RD+ 系列仿真用。

下图为转换板功能示意图



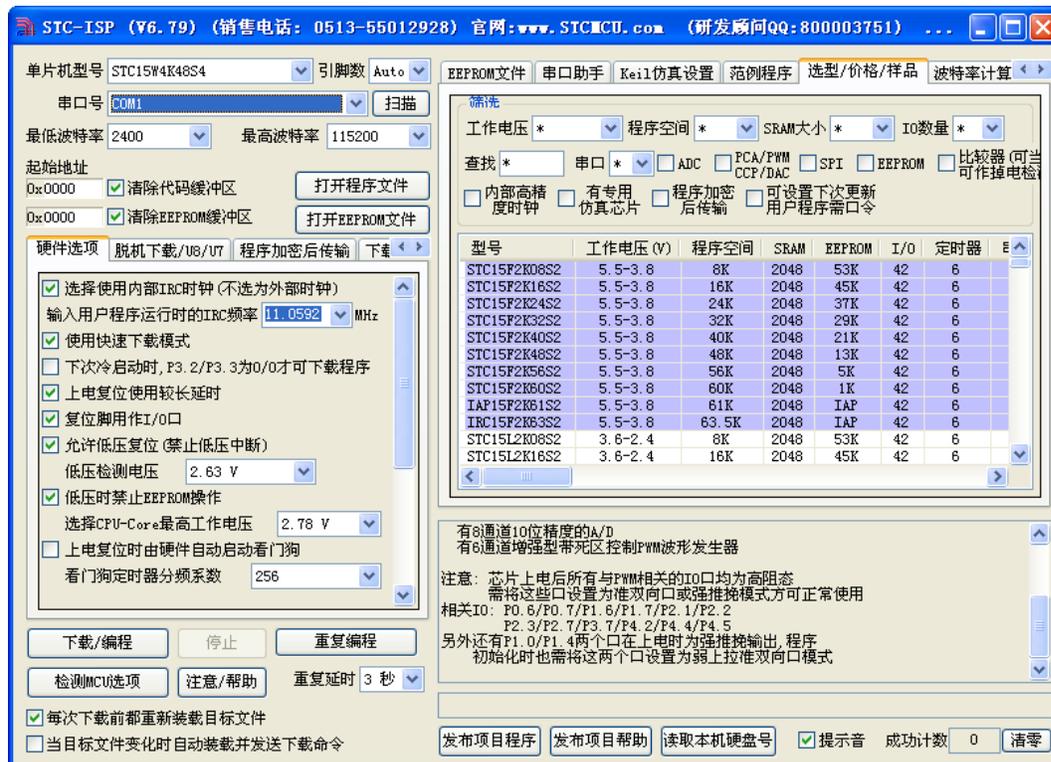
注意:

- ✓ 由于内置高精度 R/C 时钟，因此不需要外部晶振, XTAL1 和 XTAL2 是空的
- ✓ WR 和 RD 是 (WR/P4.2 和 RD/P4.4)，而不是传统的 (WR/P3.6 和 RD/P3.7)。
 (转换板中, P4.2 与 P3.6 连接在一起, P4.4 与 P3.7 连接在一起。当用户需要用此转换板访问外部总线时, 需要将 P3.6 和 P3.7 设置为高阻输入模式, 从而使 P4.2 和 P4.4 正常输出总线读写信号; 若不需要访问外部总线, 则需将 P4.2 和 P4.4 设置高阻输入模式, 3.6 和 P3.7 即为普通 I/O。)
- ✓ 由于 STC8A8K64D4 系列 MCU 是低电平复位, 与传统 8051 的高电平复位不兼容, 因此 RST 管脚是悬空, 而用转换板上的复位按键加复位电路取代

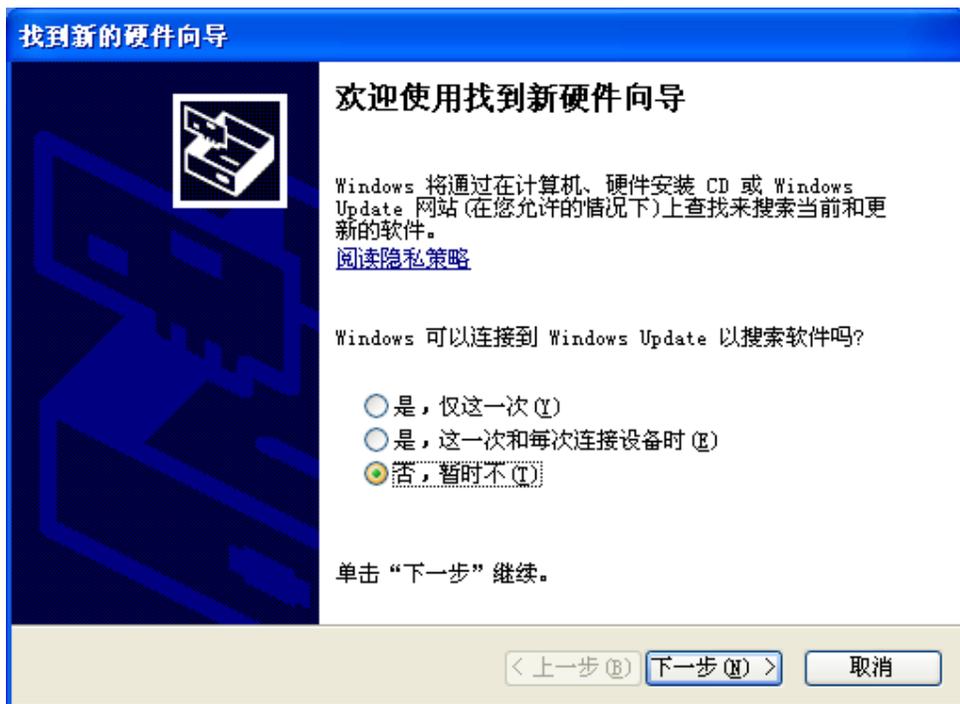
附录E STC-USB 驱动程序安装说明

Windows XP 安装方法

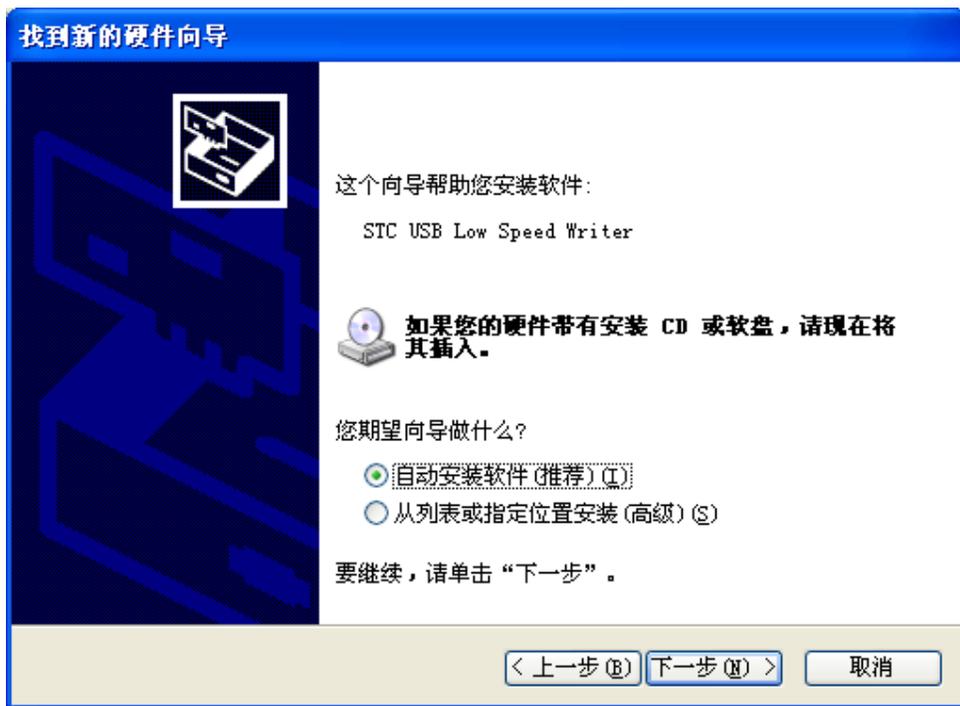
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



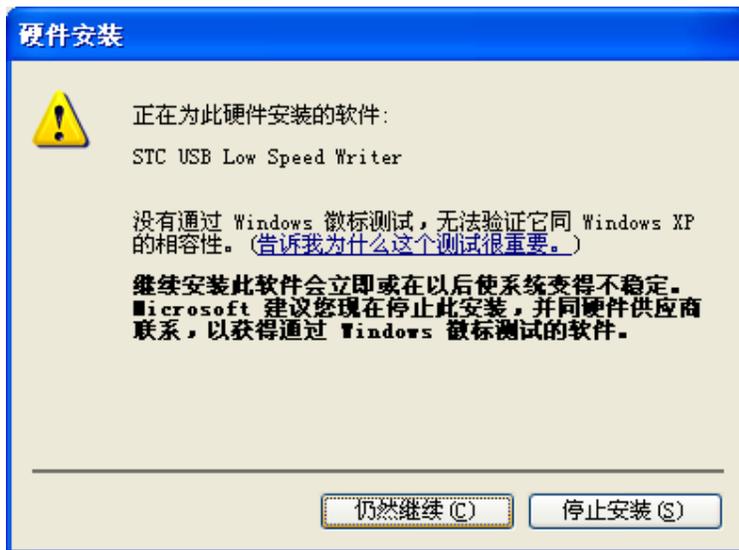
插入 USB 设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项



在下面的对话框中选择“自动安装软件(推荐)”项



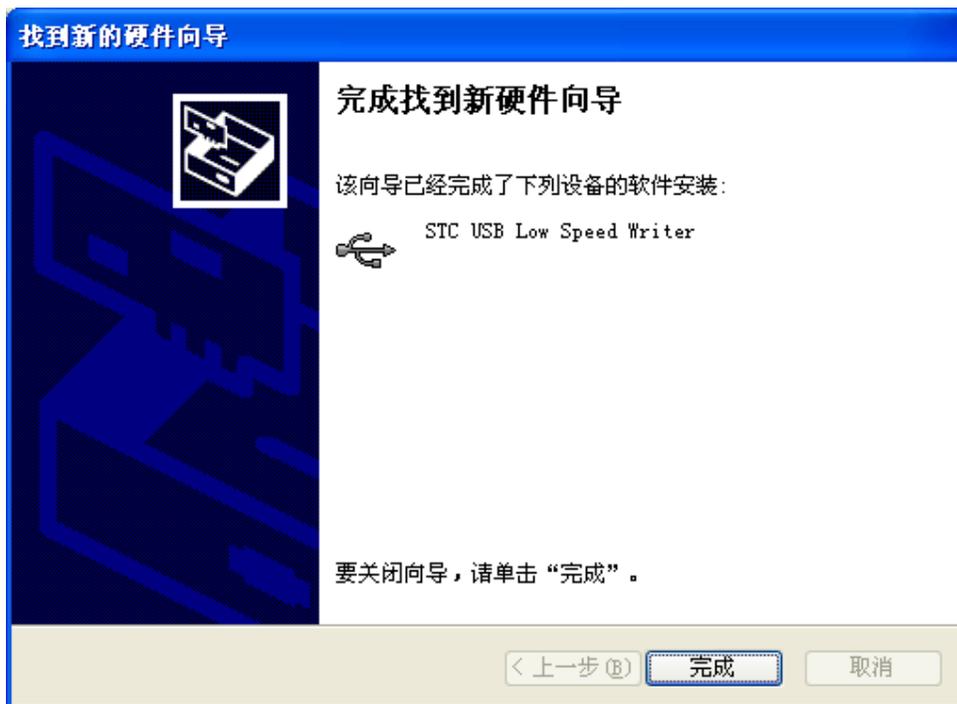
在弹出的下列对话框中, 选择“仍然继续”按钮



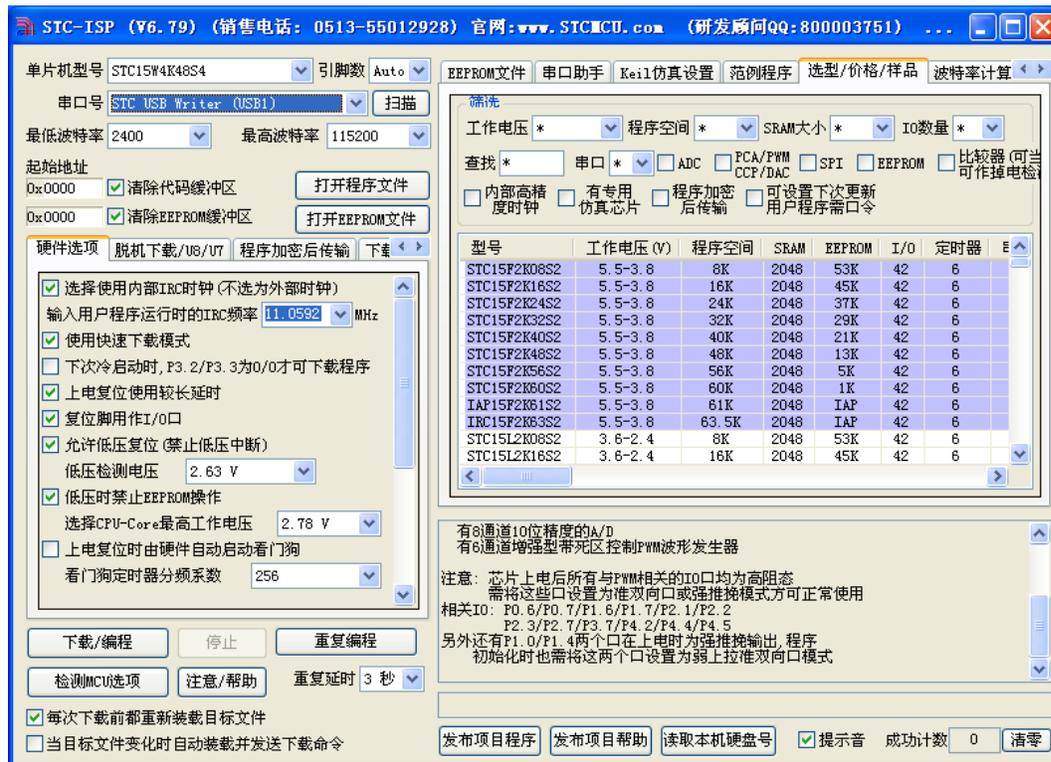
接下系统会自动安装驱动，如下图



出现下面的对话框表示驱动安装完成

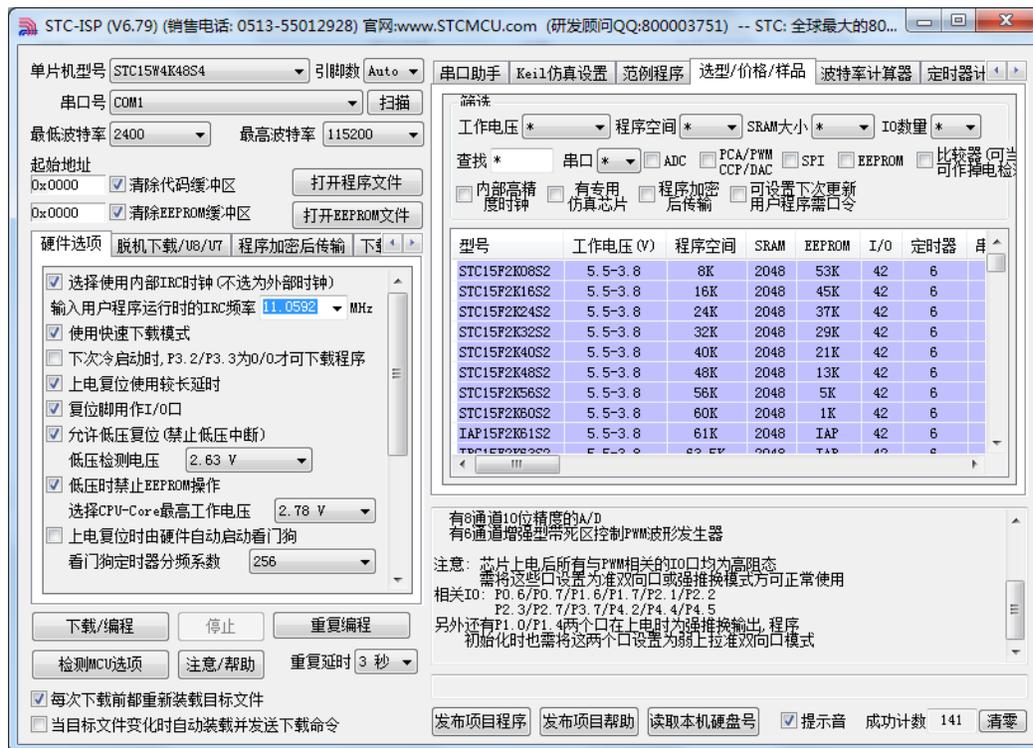


此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：



Windows 7 (32 位) 安装方法

打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录

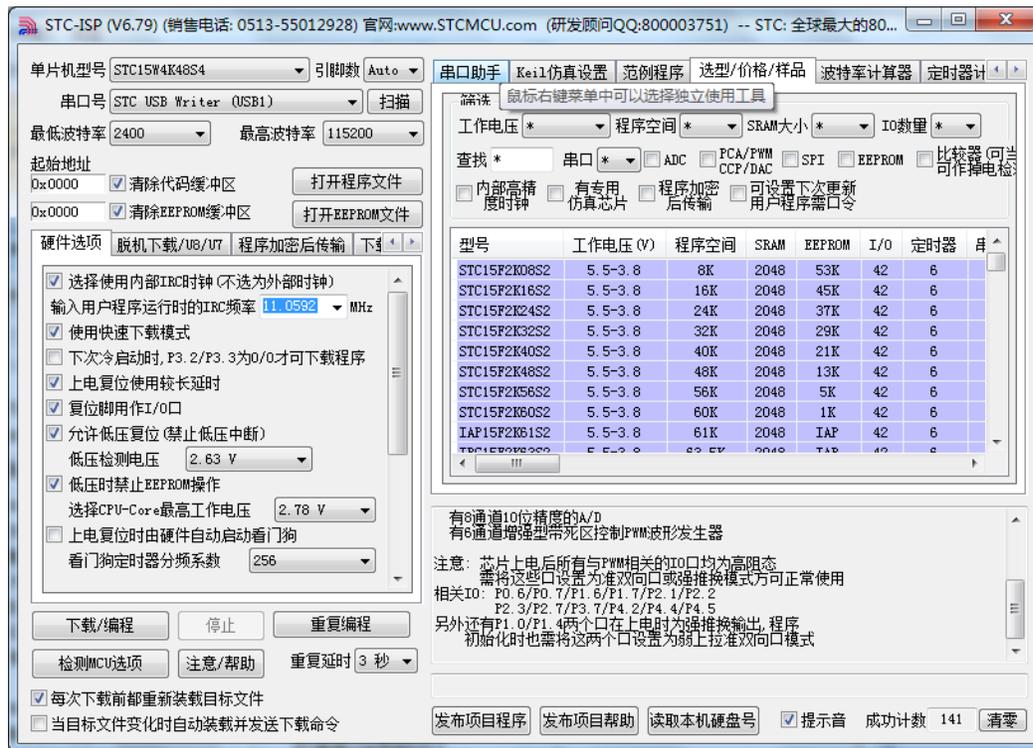


插入 USB 设备，系统找到设备后会安装驱动。安装完成后会有如下的提示框。



STC MCU

此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

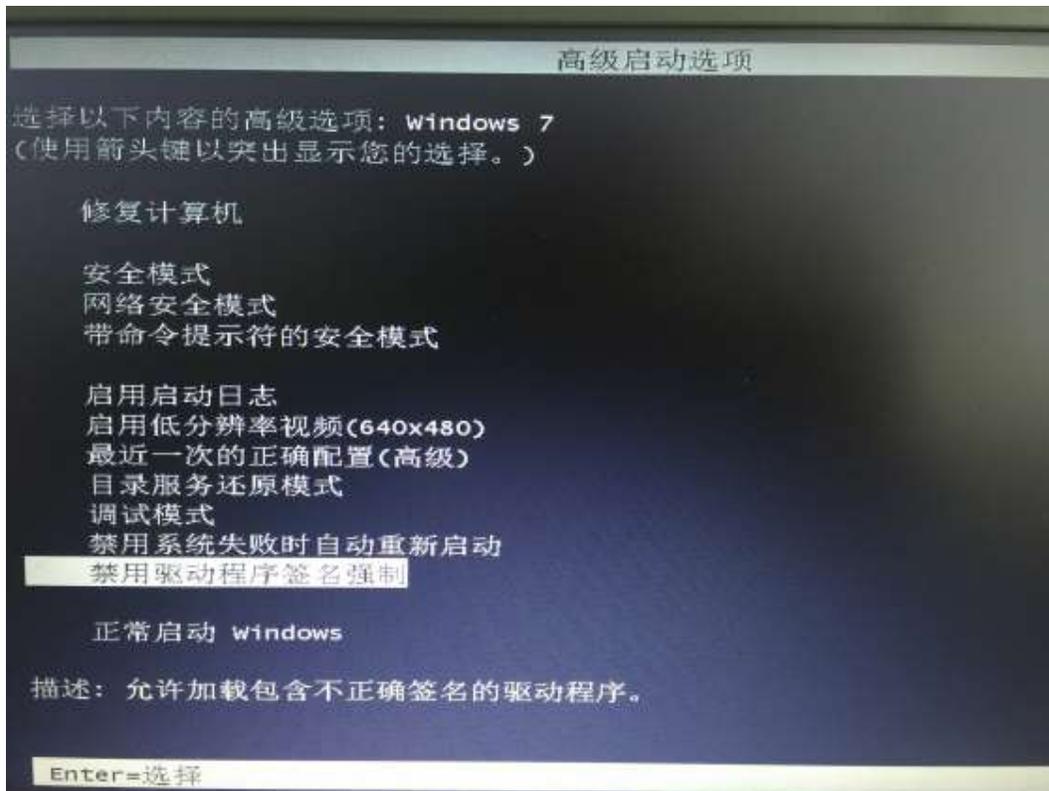


注：若 Windows 7 下，系统并没有自动安装驱动，则驱动的安装方法请参考 [Windows 8 \(32 位\) 的安装方法](#)

Windows 7 (64 位) 安装方法

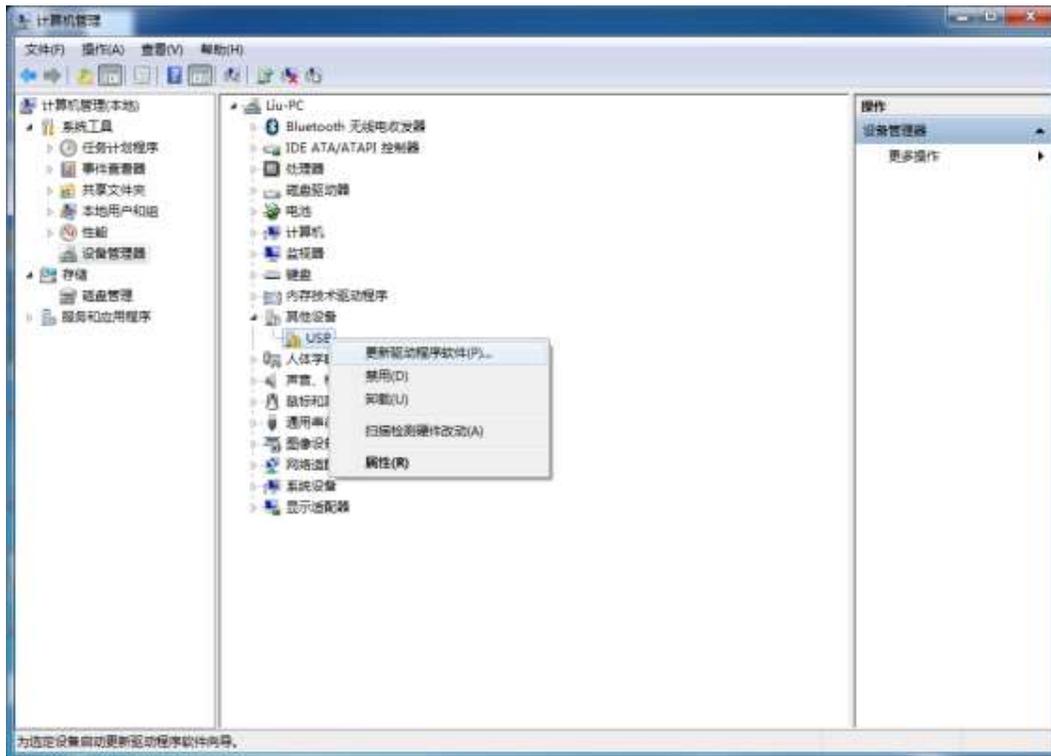
由于 Windows7 64 位操作系统在默认状态下, 对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前, 需要按照如下步骤, 暂时跳过数字签名, 即可顺利安装成功。

首先重启电脑, 并一直按住 F8, 直到出现下面启动画面



选择“禁用驱动程序签名强制”。启动后即可暂时关闭数字签名验证功能

插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的对话框中选择“浏览计算机以查找驱动程序软件”

