

```

        POP        ACC
        RETI

UART_INIT:
        MOV        SCON,#50H
        MOV        TMOD,#20H
        MOV        TL1,#0FDH           ;256-11059200/115200/32=0FDH
        MOV        TH1,#0FDH
        SETB       TRI
        MOV        AUXR,#40H
        CLR        BUSY
        MOV        WPTR,#00H
        MOV        RPTR,#00H
        RET

UART_SEND:
        JB         BUSY,$
        SETB       BUSY
        MOV        SBUF,A
        RET

UART_SENDSTR:
        CLR        A
        MOVC       A,@A+DPTR
        JZ         SENDEND
        LCALL      UART_SEND
        INC        DPTR
        JMP        UART_SENDSTR

SENDEND:
        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

        LCALL      UART_INIT
        SETB       ES
        SETB       EA

        MOV        DPTR,#STRING
        LCALL      UART_SENDSTR

LOOP:
        MOV        A,RPTR
        XRL        A,WPTR
        ANL        A,#0FH
        JZ         LOOP
        MOV        A,RPTR

```

```

        ANL        A,#0FH
        ADD        A,#BUFFER
        MOV        R0,A
        MOV        A,@R0
        LCALL     UART_SEND
        INC        RPTR
        JMP        LOOP

STRING:  DB        'Uart Test !',0DH,0AH,00H

        END

```

## 13.5.16 定时器 2（16 位自动重载），用作定时

### C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr      T2L      = 0xd7;
sfr      T2H      = 0xd6;
sfr      AUXR     = 0x8e;
sfr      IE2      = 0xaf;
#define   ET2      0x04
sfr      AUXINTIF = 0xef;
#define   T2IF     0x01

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

sbit     P10      = P1^0;

void TM2_Isr() interrupt 12
{
    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;

T2L = 0x66; //65536-11.0592M/12/1000
T2H = 0xfc;
AUXR = 0x10; //启动定时器
IE2 = ET2; //使能定时器中断
EA = 1;

while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

```

T2L      DATA      0D7H
T2H      DATA      0D6H
AUXR     DATA      8EH
IE2      DATA      0AFH
ET2      EQU        04H
AUXINTIF DATA      0EFH
T2IF     EQU        01H

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      0063H
LJMP    TM2ISR

TM2ISR:  ORG      0100H

CPL     P1.0 ;测试端口
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     T2L, #66H           ;65536-11.0592M/12/1000
MOV     T2H, #0FCH
MOV     AUXR, #10H        ;启动定时器
MOV     IE2, #ET2        ;使能定时器中断
SETB    EA

JMP     $

END

```

### 13.5.17 定时器 2（外部计数—扩展 T2 为外部下降沿中断）

#### C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

```

sfr     T2L      = 0xd7;
sfr     T2H      = 0xd6;
sfr     AUXR     = 0x8e;
sfr     IE2      = 0xaf;
#define  ET2      0x04
sfr     AUXINTIF = 0xef;
#define  T2IF     0x01

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

sbit    P10      = P1^0;

```

```
void TM2_Isr() interrupt 12
```

```
{
    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;

    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;           //设置外部计数模式并启动定时器
    IE2 = ET2;           //使能定时器中断
    EA = 1;

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

T2L	DATA	0D7H	
T2H	DATA	0D6H	
AUXR	DATA	8EH	
IE2	DATA	0AFH	
ET2	EQU	04H	
AUXINTIF	DATA	0EFH	
T2IF	EQU	01H	
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	0063H	
	LJMP	TM2ISR	
	ORG	0100H	
TM2ISR:			
	CPL	P1.0	;测试端口
	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     T2L, #0FFH
MOV     T2H, #0FFH
MOV     AUXR, #18H           ;设置外部计数模式并启动定时器
MOV     IE2, #ET2          ;使能定时器中断
SETB    EA

JMP     $

END

```

## 13.5.18 定时器 2，时钟分频输出

### C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

```
sfr     T2L      = 0xd7;
sfr     T2H      = 0xd6;
sfr     AUXR     = 0x8e;
sfr     INTCLKO  = 0x8f;
```

```
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;

T2L = 0x66; //65536-11.0592M/12/1000
T2H = 0xfc;
AUXR = 0x10; //启动定时器
INTCLKO = 0x04; //使能时钟输出

while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

```

T2L      DATA      0D7H
T2H      DATA      0D6H
AUXR     DATA      8EH
INTCLKO  DATA      8FH

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     T2L,#66H           ;65536-11.0592M/12/1000
MOV     T2H,#0FCH
MOV     AUXR,#10H        ;启动定时器
MOV     INTCLKO,#04H     ;使能时钟输出

JMP     $

END

```

## 13.5.19 定时器 2 做串口 1 波特率发生器

### C 语言代码

//测试工作频率为 11.0592MHz

```

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

```

```

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

```

```

sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;

```

```

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;

```

```

bit busy;
char wptr;
char rptr;
char buffer[16];

```

```

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

```



```
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

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;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

## 汇编代码

;测试工作频率为11.0592MHz

```

AUXR      DATA      8EH
T2H       DATA      0D6H
T2L       DATA      0D7H

BUSY      BIT         20H.0
WPTR      DATA      21H
RPTR      DATA      22H
BUFFER    DATA      23H                ;16 bytes

P0MI      DATA      093H
P0M0      DATA      094H
P1MI      DATA      091H
P1M0      DATA      092H
P2MI      DATA      095H
P2M0      DATA      096H
P3MI      DATA      0B1H
P3M0      DATA      0B2H
P4MI      DATA      0B3H
P4M0      DATA      0B4H
P5MI      DATA      0C9H
P5M0      DATA      0CAH

ORG       0000H
LJMP     MAIN
ORG       0023H
LJMP     UART_ISR

ORG       0100H

UART_ISR:
PUSH    ACC
PUSH    PSW
MOV     PSW,#08H

JNB     TI,CHKRI
CLR     TI
CLR     BUSY

CHKRI:
JNB     RI,UARTISR_EXIT
CLR     RI
MOV     A,WPTR
ANL    A,#0FH
ADD    A,#BUFFER
MOV     R0,A
MOV     @R0,SBUF
INC    WPTR

UARTISR_EXIT:
POP     PSW
POP     ACC
RETI

UART_INIT:
MOV     SCON,#50H
MOV     T2L,#0E8H                ;65536-11059200/115200/4=0FFE8H
MOV     T2H,#0FFH
MOV     AUXR,#15H

```

```

        CLR        BUSY
        MOV        WPTR,#00H
        MOV        RPTR,#00H
        RET

UART_SEND:
        JB         BUSY,$
        SETB      BUSY
        MOV        SBUF,A
        RET

UART_SENDSTR:
        CLR        A
        MOVC      A,@A+DPTR
        JZ         SENDEND
        LCALL     UART_SEND
        INC       DPTR
        JMP        UART_SENDSTR

SENDEND:
        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

        LCALL     UART_INIT
        SETB      ES
        SETB      EA

        MOV        DPTR,#STRING
        LCALL     UART_SENDSTR

LOOP:
        MOV        A,RPTR
        XRL       A,WPTR
        ANL       A,#0FH
        JZ         LOOP
        MOV        A,RPTR
        ANL       A,#0FH
        ADD       A,#BUFFER
        MOV        R0,A
        MOV        A,@R0
        LCALL     UART_SEND
        INC       RPTR
        JMP        LOOP

STRING:  DB        'Uart Test !',0DH,0AH,00H

```

---

---

END

---

---

## 13.5.20 定时器 2 做串口 2 波特率发生器

### C 语言代码

---

---

//测试工作频率为 11.0592MHz

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

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

```
sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;
sfr S2CON = 0x9a;
sfr S2BUF = 0x9b;
sfr IE2 = 0xaf;
```

```
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;
```

```
bit busy;
char wptr;
char rptr;
char buffer[16];
```

```
void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart2Init()
{
```

```
S2CON = 0x10;
T2L = BRT;
T2H = BRT >> 8;
AUXR = 0x14;
wptr = 0x00;
rptr = 0x00;
busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2SEND(*p++);
    }
}

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;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

---

## 汇编代码

---

;测试工作频率为11.0592MHz

AUXR

DATA

8EH

<i>T2H</i>	<i>DATA</i>	<i>0D6H</i>	
<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>	
<i>S2CON</i>	<i>DATA</i>	<i>9AH</i>	
<i>S2BUF</i>	<i>DATA</i>	<i>9BH</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</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>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0043H</i>	
	<i>LJMP</i>	<i>UART2_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART2_ISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>PSW,#08H</i>	
	<i>MOV</i>	<i>A,S2CON</i>	
	<i>JNB</i>	<i>ACC.1,CHKRI</i>	
	<i>ANL</i>	<i>S2CON,#NOT 02H</i>	
	<i>CLR</i>	<i>BUSY</i>	
<i>CHKRI:</i>			
	<i>JNB</i>	<i>ACC.0,UART2ISR_EXIT</i>	
	<i>ANL</i>	<i>S2CON,#NOT 01H</i>	
	<i>MOV</i>	<i>A,WPTR</i>	
	<i>ANL</i>	<i>A,#0FH</i>	
	<i>ADD</i>	<i>A,#BUFFER</i>	
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOV</i>	<i>@R0,S2BUF</i>	
	<i>INC</i>	<i>WPTR</i>	
<i>UART2ISR_EXIT:</i>			
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>UART2_INIT:</i>			
	<i>MOV</i>	<i>S2CON,#10H</i>	
	<i>MOV</i>	<i>T2L,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>T2H,#0FFH</i>	
	<i>MOV</i>	<i>AUXR,#14H</i>	

```

CLR      BUSY
MOV      WPTR,#00H
MOV      RPTR,#00H
RET

```

**UART2\_SEND:**

```

JB       BUSY,$
SETB    BUSY
MOV      S2BUF,A
RET

```

**UART2\_SENDSTR:**

```

CLR      A
MOVC    A,@A+DPTR
JZ       SEND2END
LCALL   UART2_SEND
INC     DPTR
JMP     UART2_SENDSTR

```

**SEND2END:**

```
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

LCALL   UART2_INIT
MOV     IE2,#01H
SETB    EA

MOV     DPTR,#STRING
LCALL  UART2_SENDSTR

```

**LOOP:**

```

MOV     A,RPTR
XRL    A,WPTR
ANL    A,#0FH
JZ     LOOP
MOV     A,RPTR
ANL    A,#0FH
ADD    A,#BUFFER
MOV     R0,A
MOV     A,@R0
LCALL  UART2_SEND
INC    RPTR
JMP    LOOP

```

```
STRING:  DB      'Uart Test !',0DH,0AH,00H
```

---



---

END

---



---

## 13.5.21 定时器 2 做串口 3 波特率发生器

### C 语言代码

---



---

```
//测试工作频率为 11.0592MHz
```

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

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

```
sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;
sfr S3CON = 0xac;
sfr S3BUF = 0xad;
sfr IE2 = 0xaf;
```

```
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;
```

```
bit busy;
char wptr;
char rptr;
char buffer[16];
```

```
void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart3Init()
{
```



```
S3CON = 0x10;
T2L = BRT;
T2H = BRT >> 8;
AUXR = 0x14;
wptr = 0x00;
rptr = 0x00;
busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

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;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

---

## 汇编代码

---

;测试工作频率为11.0592MHz

AUXR

DATA

8EH

<i>T2H</i>	<i>DATA</i>	<i>0D6H</i>	
<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>	
<i>S3CON</i>	<i>DATA</i>	<i>0ACH</i>	
<i>S3BUF</i>	<i>DATA</i>	<i>0ADH</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</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>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>008BH</i>	
	<i>LJMP</i>	<i>UART3_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART3_ISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>PSW,#08H</i>	
	<i>MOV</i>	<i>A,S3CON</i>	
	<i>JNB</i>	<i>ACC.1,CHKRI</i>	
	<i>ANL</i>	<i>S3CON,#NOT 02H</i>	
	<i>CLR</i>	<i>BUSY</i>	
<i>CHKRI:</i>			
	<i>JNB</i>	<i>ACC.0,UART3ISR_EXIT</i>	
	<i>ANL</i>	<i>S3CON,#NOT 01H</i>	
	<i>MOV</i>	<i>A,WPTR</i>	
	<i>ANL</i>	<i>A,#0FH</i>	
	<i>ADD</i>	<i>A,#BUFFER</i>	
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOV</i>	<i>@R0,S3BUF</i>	
	<i>INC</i>	<i>WPTR</i>	
<i>UART3ISR_EXIT:</i>			
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>UART3_INIT:</i>			
	<i>MOV</i>	<i>S3CON,#10H</i>	
	<i>MOV</i>	<i>T2L,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>T2H,#0FFH</i>	
	<i>MOV</i>	<i>AUXR,#14H</i>	

```

        CLR        BUSY
        MOV        WPTR,#00H
        MOV        RPTR,#00H
        RET

UART3_SEND:
        JB        BUSY,$
        SETB       BUSY
        MOV        S3BUF,A
        RET

UART3_SENDSTR:
        CLR        A
        MOVC       A,@A+DPTR
        JZ        SEND3END
        LCALL      UART3_SEND
        INC        DPTR
        JMP        UART3_SENDSTR

SEND3END:
        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

        LCALL      UART3_INIT
        MOV        IE2,#08H
        SETB       EA

        MOV        DPTR,#STRING
        LCALL      UART3_SENDSTR

LOOP:
        MOV        A,RPTR
        XRL        A,WPTR
        ANL        A,#0FH
        JZ        LOOP
        MOV        A,RPTR
        ANL        A,#0FH
        ADD        A,#BUFFER
        MOV        R0,A
        MOV        A,@R0
        LCALL      UART3_SEND
        INC        RPTR
        JMP        LOOP

STRING:  DB        'Uart Test !',0DH,0AH,00H

```

---



---

END

---



---

## 13.5.22 定时器 2 做串口 4 波特率发生器

### C 语言代码

---



---

```
//测试工作频率为 11.0592MHz
```

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

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

```
sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;
sfr S4CON = 0x84;
sfr S4BUF = 0x85;
sfr IE2 = 0xaf;
```

```
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;
```

```
bit busy;
char wptr;
char rptr;
char buffer[16];
```

```
void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart4Init()
{
```

```
S4CON = 0x10;
T2L = BRT;
T2H = BRT >> 8;
AUXR = 0x14;
wptr = 0x00;
rptr = 0x00;
busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

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;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

---

## 汇编代码

---

;测试工作频率为11.0592MHz

AUXR

DATA

8EH

<i>T2H</i>	<i>DATA</i>	<i>0D6H</i>	
<i>T2L</i>	<i>DATA</i>	<i>0D7H</i>	
<i>S4CON</i>	<i>DATA</i>	<i>84H</i>	
<i>S4BUF</i>	<i>DATA</i>	<i>85H</i>	
<i>IE2</i>	<i>DATA</i>	<i>0AFH</i>	
<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</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>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0093H</i>	
	<i>LJMP</i>	<i>UART4_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART4_ISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>PSW,#08H</i>	
	<i>MOV</i>	<i>A,S4CON</i>	
	<i>JNB</i>	<i>ACC.1,CHKRI</i>	
	<i>ANL</i>	<i>S4CON,#NOT 02H</i>	
	<i>CLR</i>	<i>BUSY</i>	
<i>CHKRI:</i>			
	<i>JNB</i>	<i>ACC.0,UART4ISR_EXIT</i>	
	<i>ANL</i>	<i>S4CON,#NOT 01H</i>	
	<i>MOV</i>	<i>A,WPTR</i>	
	<i>ANL</i>	<i>A,#0FH</i>	
	<i>ADD</i>	<i>A,#BUFFER</i>	
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOV</i>	<i>@R0,S4BUF</i>	
	<i>INC</i>	<i>WPTR</i>	
<i>UART4ISR_EXIT:</i>			
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>UART4_INIT:</i>			
	<i>MOV</i>	<i>S4CON,#10H</i>	
	<i>MOV</i>	<i>T2L,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>T2H,#0FFH</i>	
	<i>MOV</i>	<i>AUXR,#14H</i>	

```

        CLR        BUSY
        MOV        WPTR,#00H
        MOV        RPTR,#00H
        RET

UART4_SEND:
        JB         BUSY,$
        SETB      BUSY
        MOV        S4BUF,A
        RET

UART4_SENDSTR:
        CLR        A
        MOVC      A,@A+DPTR
        JZ         SEND4END
        LCALL     UART4_SEND
        INC        DPTR
        JMP        UART4_SENDSTR

SEND4END:
        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

        LCALL     UART4_INIT
        MOV        IE2,#10H
        SETB      EA

        MOV        DPTR,#STRING
        LCALL     UART4_SENDSTR

LOOP:
        MOV        A,RPTR
        XRL        A,WPTR
        ANL        A,#0FH
        JZ         LOOP
        MOV        A,RPTR
        ANL        A,#0FH
        ADD        A,#BUFFER
        MOV        R0,A
        MOV        A,@R0
        LCALL     UART4_SEND
        INC        RPTR
        JMP        LOOP

STRING:  DB        'Uart Test !',0DH,0AH,00H

```

---

---

END

---

---

### 13.5.23 定时器 3 (16 位自动重载), 用作定时

#### C 语言代码

---

---

```
//测试工作频率为 11.0592MHz
```

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

```
sfr    T4T3M    = 0xd1;  
sfr    T4L      = 0xd3;  
sfr    T4H      = 0xd2;  
sfr    T3L      = 0xd5;  
sfr    T3H      = 0xd4;  
sfr    T2L      = 0xd7;  
sfr    T2H      = 0xd6;  
sfr    AUXR     = 0x8e;  
sfr    IE2      = 0xaf;  
#define ET2     0x04  
#define ET3     0x20  
#define ET4     0x40  
sfr    AUXINTIF = 0xef;  
#define T2IF    0x01  
#define T3IF    0x02  
#define T4IF    0x04  
  
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   P10      = P1^0;
```

```
void TM3_Isr() interrupt 19
```

```
{  
    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;

T3L = 0x66;           //65536-11.0592M/12/1000
T3H = 0xfc;
T4T3M = 0x08;        //启动定时器
IE2 = ET3;           //使能定时器中断
EA = 1;

while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

T4T3M      DATA      0D1H
T4L        DATA      0D3H
T4H        DATA      0D2H
T3L        DATA      0D5H
T3H        DATA      0D4H
T2L        DATA      0D7H
T2H        DATA      0D6H
AUXR       DATA      8EH
IE2        DATA      0AFH
ET2        EQU        04H
ET3        EQU        20H
ET4        EQU        40H
AUXINTIF   DATA      0EFH
T2IF       EQU        01H
T3IF       EQU        02H
T4IF       EQU        04H

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

          ORG          0000H
          LJMP        MAIN
          ORG          009BH
          LJMP        TM3ISR

          ORG          0100H
TM3ISR:
          CPL          P1.0           ;测试端口
          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     T3L, #66H           ;65536-11.0592M/12/1000
MOV     T3H, #0FCH
MOV     T4T3M, #08H        ;启动定时器
MOV     IE2, #ET3          ;使能定时器中断
SETB    EA

JMP     $

END

```

### 13.5.24 定时器 3（外部计数—扩展 T3 为外部下降沿中断）

C 语言代码

//测试工作频率为 11.0592MHz

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

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

```

sfr     T4T3M    = 0xd1;
sfr     T4L      = 0xd3;
sfr     T4H      = 0xd2;
sfr     T3L      = 0xd5;
sfr     T3H      = 0xd4;
sfr     T2L      = 0xd7;
sfr     T2H      = 0xd6;
sfr     AUXR     = 0x8e;
sfr     IE2      = 0xaf;
#define  ET2      0x04
#define  ET3      0x20
#define  ET4      0x40
sfr     AUXINTIF = 0xef;
#define  T2IF     0x01
#define  T3IF     0x02
#define  T4IF     0x04

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   P10     = P1^0;

```

```

void TM3_Isr() interrupt 19

```

```

{
    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;

    T3L = 0x66;           //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x0c;        //设置外部计数模式并启动定时器
    IE2 = ET3;           //使能定时器中断
    EA = 1;

    while (1);
}

```

## 汇编代码

```

;测试工作频率为 11.0592MHz

```

```

T4T3M    DATA    0D1H
T4L      DATA    0D3H
T4H      DATA    0D2H
T3L      DATA    0D5H
T3H      DATA    0D4H
T2L      DATA    0D7H
T2H      DATA    0D6H
AUXR     DATA    8EH
IE2      DATA    0AFH
ET2      EQU      04H
ET3      EQU      20H
ET4      EQU      40H
AUXINTIF DATA    0EFH
T2IF     EQU      01H
T3IF     EQU      02H
T4IF     EQU      04H

```

```

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

        ORG      0000H
        LJMP    MAIN
        ORG      009BH
        LJMP    TM3ISR

TM3ISR:  ORG      0100H

        CPL     P1.0           ;测试端口
        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     T3L, #66H      ;65536-11.0592M/12/1000
        MOV     T3H, #0FCH
        MOV     T4T3M, #0CH   ;设置外部计数模式并启动定时器
        MOV     IE2, #ET3    ;使能定时器中断
        SETB    EA

        JMP     $

        END

```

### 13.5.25 定时器 3，时钟分频输出

#### C 语言代码

```
//测试工作频率为 11.0592MHz;
```

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

```

#include "intrins.h"

sfr    T4T3M    = 0xd1;
sfr    T4L      = 0xd3;
sfr    T4H      = 0xd2;
sfr    T3L      = 0xd5;
sfr    T3H      = 0xd4;
sfr    T2L      = 0xd7;
sfr    T2H      = 0xd6;

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;

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;

    T3L = 0x66;                //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;             //使能时钟输出并启动定时器

    while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

```

T4T3M    DATA    0D1H
T4L      DATA    0D3H
T4H      DATA    0D2H
T3L      DATA    0D5H
T3H      DATA    0D4H
T2L      DATA    0D7H
T2H      DATA    0D6H

P1M1     DATA    091H
P1M0     DATA    092H

```

```

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

          ORG          0000H
          LJMP        MAIN

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          T3L, #66H          ;65536-11.0592M/12/1000
          MOV          T3H, #0FCH
          MOV          T4T3M, #09H      ; 使能时钟输出并启动定时器

          JMP          $

          END

```

### 13.5.26 定时器 3 做串口 3 波特率发生器

#### C 语言代码

```
//测试工作频率为 11.0592MHz
```

```

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

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

sfr  T4T3M      = 0xd1;
sfr  T4L        = 0xd3;
sfr  T4H        = 0xd2;
sfr  T3L        = 0xd5;
sfr  T3H        = 0xd4;
sfr  T2L        = 0xd7;

```

```
sfr    T2H      = 0xd6;
sfr    S3CON    = 0xac;
sfr    S3BUF    = 0xad;
sfr    IE2      = 0xaf;

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;

bit    busy;
char   wptr;
char   rptr;
char   buffer[16];
```

```
void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}
```

```
void Uart3SendStr(char *p)
{
    while (*p)
```

```

    {
        Uart3SEND(*p++);
    }
}

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;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

<b>T4T3M</b>	<b>DATA</b>	<b>0D1H</b>	
<b>T4L</b>	<b>DATA</b>	<b>0D3H</b>	
<b>T4H</b>	<b>DATA</b>	<b>0D2H</b>	
<b>T3L</b>	<b>DATA</b>	<b>0D5H</b>	
<b>T3H</b>	<b>DATA</b>	<b>0D4H</b>	
<b>T2L</b>	<b>DATA</b>	<b>0D7H</b>	
<b>T2H</b>	<b>DATA</b>	<b>0D6H</b>	
<b>S3CON</b>	<b>DATA</b>	<b>0ACH</b>	
<b>S3BUF</b>	<b>DATA</b>	<b>0ADH</b>	
<b>IE2</b>	<b>DATA</b>	<b>0AFH</b>	
<b>BUSY</b>	<b>BIT</b>	<b>20H.0</b>	
<b>WPTR</b>	<b>DATA</b>	<b>21H</b>	
<b>RPTR</b>	<b>DATA</b>	<b>22H</b>	
<b>BUFFER</b>	<b>DATA</b>	<b>23H</b>	;16 bytes
<b>P0M1</b>	<b>DATA</b>	<b>093H</b>	
<b>P0M0</b>	<b>DATA</b>	<b>094H</b>	
<b>P1M1</b>	<b>DATA</b>	<b>091H</b>	
<b>P1M0</b>	<b>DATA</b>	<b>092H</b>	



```

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          008BH
          LJMP         UART3_ISR

          ORG          0100H

UART3_ISR:
          PUSH         ACC
          PUSH         PSW
          MOV          PSW,#08H

          MOV          A,S3CON
          JNB         ACC.1,CHKRI
          ANL         S3CON,#NOT 02H
          CLR         BUSY

CHKRI:
          JNB         ACC.0,UART3ISR_EXIT
          ANL         S3CON,#NOT 01H
          MOV          A,WPTR
          ANL         A,#0FH
          ADD         A,#BUFFER
          MOV          R0,A
          MOV          @R0,S3BUF
          INC         WPTR

UART3ISR_EXIT:
          POP          PSW
          POP          ACC
          RETI

UART3_INIT:
          MOV          S3CON,#50H
          MOV          T3L,#0E8H          ;65536-11059200/115200/4=0FFE8H
          MOV          T3H,#0FFH
          MOV          T4T3M,#0AH
          CLR         BUSY
          MOV          WPTR,#00H
          MOV          RPTR,#00H
          RET

UART3_SEND:
          JB          BUSY,$
          SETB        BUSY
          MOV          S3BUF,A
          RET

UART3_SENDSTR:
          CLR         A
          MOVC        A,@A+DPTR
          JZ          SEND3END

```

```

        LCALL    UART3_SEND
        INC     DPTR
        JMP     UART3_SENDSTR
SEND3END:
        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

        LCALL    UART3_INIT
        MOV     IE2, #08H
        SETB    EA

        MOV     DPTR, #STRING
        LCALL    UART3_SENDSTR

LOOP:
        MOV     A, RPTR
        XRL    A, WPTR
        ANL    A, #0FH
        JZ     LOOP
        MOV     A, RPTR
        ANL    A, #0FH
        ADD    A, #BUFFER
        MOV     R0, A
        MOV     A, @R0
        LCALL    UART3_SEND
        INC     RPTR
        JMP     LOOP

STRING:  DB     'Uart Test !', 0DH, 0AH, 00H

        END

```

### 13.5.27 定时器 4（16 位自动重载），用作定时

#### C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

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

```
sfr     T4T3M    = 0xd1;
```

```

sfr    T4L      = 0xd3;
sfr    T4H      = 0xd2;
sfr    T3L      = 0xd5;
sfr    T3H      = 0xd4;
sfr    T2L      = 0xd7;
sfr    T2H      = 0xd6;
sfr    AUXR     = 0x8e;
sfr    IE2      = 0xaf;
#define ET2      0x04
#define ET3      0x20
#define ET4      0x40
sfr    AUXINTIF = 0xef;
#define T2IF     0x01
#define T3IF     0x02
#define T4IF     0x04

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   P10      = P1^0;

void TM4_Isr() interrupt 20
{
    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;

    T4L = 0x66;    //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80; //启动定时器
    IE2 = ET4;    //使能定时器中断
    EA = 1;

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

T4T3M    DATA    0D1H
T4L      DATA    0D3H
T4H      DATA    0D2H
T3L      DATA    0D5H
T3H      DATA    0D4H
T2L      DATA    0D7H
T2H      DATA    0D6H
AUXR     DATA    8EH
IE2      DATA    0AFH
ET2      EQU      04H
ET3      EQU      20H
ET4      EQU      40H
AUXINTIF DATA    0EFH
T2IF     EQU      01H
T3IF     EQU      02H
T4IF     EQU      04H

```

```

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

```

```

ORG      0000H
LJMP     MAIN
ORG      00A3H
LJMP     TM4ISR

```

```

ORG      0100H

```

TM4ISR:

```

CPL      P1.0
RETI

```

;测试端口

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      T4L,#66H           ;65536-11.0592M/12/1000
MOV      T4H,#0FCH
MOV      T4T3M,#80H        ;启动定时器
MOV      IE2,#ET4          ;使能定时器中断
SETB     EA

JMP      $

END

```

## 13.5.28 定时器 4（外部计数—扩展 T4 为外部下降沿中断）

### C 语言代码

//测试工作频率为 11.0592MHz

```

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

sfr      T4T3M      = 0xd1;
sfr      T4L        = 0xd3;
sfr      T4H        = 0xd2;
sfr      T3L        = 0xd5;
sfr      T3H        = 0xd4;
sfr      T2L        = 0xd7;
sfr      T2H        = 0xd6;
sfr      AUXR       = 0x8e;
sfr      IE2        = 0xaf;
#define   ET2        0x04
#define   ET3        0x20
#define   ET4        0x40
sfr      AUXINTIF   = 0xef;
#define   T2IF       0x01
#define   T3IF       0x02
#define   T4IF       0x04

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     P10        = P1^0;

void TM4_Isr() interrupt 20
{
    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;

    T4L = 0x66;           //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0xc0;       //设置外部计数模式并启动定时器
    IE2 = ET4;          //使能定时器中断
    EA = 1;

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

T4T3M	DATA	0D1H
T4L	DATA	0D3H
T4H	DATA	0D2H
T3L	DATA	0D5H
T3H	DATA	0D4H
T2L	DATA	0D7H
T2H	DATA	0D6H
AUXR	DATA	8EH
IE2	DATA	0AFH
ET2	EQU	04H
ET3	EQU	20H
ET4	EQU	40H
AUXINTIF	DATA	0EFH
T2IF	EQU	01H
T3IF	EQU	02H
T4IF	EQU	04H
P1M1	DATA	091H
P1M0	DATA	092H
P0M1	DATA	093H
P0M0	DATA	094H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH

```

        ORG      0000H
        LJMP    MAIN
        ORG      00A3H
        LJMP    TM4ISR

TM4ISR:
        ORG      0100H
        CPL     P1.0           ;测试端口
        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     T4L, #66H      ;65536-11.0592M/12/1000
        MOV     T4H, #0FCH
        MOV     T4T3M, #0C0H  ;设置外部计数模式并启动定时器
        MOV     IE2, #ET4     ;使能定时器中断
        SETB    EA

        JMP     $

        END

```

### 13.5.29 定时器 4，时钟分频输出

#### C 语言代码

//测试工作频率为 11.0592MHz

```

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

```

```

sfr      T4T3M    = 0xd1;
sfr      T4L      = 0xd3;
sfr      T4H      = 0xd2;
sfr      T3L      = 0xd5;
sfr      T3H      = 0xd4;
sfr      T2L      = 0xd7;
sfr      T2H      = 0xd6;

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;

```

```
void main()
```

```
{
```

```
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
```

```
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
```

```
    P4M1 = 0x00;
```

```
    P5M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```
    T4L = 0x66;
```

```
//65536-11.0592M/12/1000
```

```
    T4H = 0xfc;
```

```
    T4T3M = 0x90;
```

```
//使能时钟输出并启动定时器
```

```
    while (1);
```

```
}
```

## 汇编代码

;测试工作频率为11.0592MHz

```

T4T3M    DATA    0D1H
T4L      DATA    0D3H
T4H      DATA    0D2H
T3L      DATA    0D5H
T3H      DATA    0D4H
T2L      DATA    0D7H
T2H      DATA    0D6H

```

```

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

```

```
    ORG    0000H
```

```
    LJMP   MAIN
```



```

        ORG          0100H

MAIN:

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

        MOV         T4L, #66H           ;65536-11.0592M/12/1000
        MOV         T4H, #0FCH
        MOV         T4T3M, #90H       ;使能时钟输出并启动定时器

        JMP         $

        END

```

### 13.5.30 定时器 4 做串口 4 波特率发生器

#### C 语言代码

//测试工作频率为 11.0592MHz

```

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

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

sfr      T4T3M      = 0xd1;
sfr      T4L        = 0xd3;
sfr      T4H        = 0xd2;
sfr      T3L        = 0xd5;
sfr      T3H        = 0xd4;
sfr      T2L        = 0xd7;
sfr      T2H        = 0xd6;
sfr      S4CON      = 0x84;
sfr      S4BUF      = 0x85;
sfr      IE2        = 0xaf;

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;
```

```
bit    busy;
char   wptr;
char   rptr;
char   buffer[16];
```

```
void Uart4Isr() interrupt 18
```

```
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart4Init()
```

```
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart4Send(char dat)
```

```
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}
```

```
void Uart4SendStr(char *p)
```

```
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}
```

```
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;

Uart4Init();
IE2 = 0x10;
EA = 1;
Uart4SENDStr("Uart Test !\r\n");

while (1)
{
    if (rptr != wptr)
    {
        Uart4SEND(buffer[rptr++]);
        rptr &= 0x0f;
    }
}
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

T4T3M	DATA	0D1H	
T4L	DATA	0D3H	
T4H	DATA	0D2H	
T3L	DATA	0D5H	
T3H	DATA	0D4H	
T2L	DATA	0D7H	
T2H	DATA	0D6H	
S4CON	DATA	84H	
S4BUF	DATA	85H	
IE2	DATA	0AFH	
BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes
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	0093H	
	LJMP	UART4_ISR	

```

        ORG          0100H

UART4_ISR:
        PUSH        ACC
        PUSH        PSW
        MOV         PSW,#08H

        MOV         A,S4CON
        JNB        ACC.1,CHKRI
        ANL        S4CON,#NOT 02H
        CLR        BUSY

CHKRI:
        JNB        ACC.0,UART4ISR_EXIT
        ANL        S4CON,#NOT 01H
        MOV         A,WPTR
        ANL        A,#0FH
        ADD        A,#BUFFER
        MOV         R0,A
        MOV         @R0,S4BUF
        INC        WPTR

UART4ISR_EXIT:
        POP         PSW
        POP         ACC
        RETI

UART4_INIT:
        MOV         S4CON,#50H
        MOV         T4L,#0E8H ;65536-11059200/115200/4=0FFE8H
        MOV         T4H,#0FFH
        MOV         T4T3M,#0A0H
        CLR        BUSY
        MOV         WPTR,#00H
        MOV         RPTR,#00H
        RET

UART4_SEND:
        JB         BUSY,$
        SETB       BUSY
        MOV         S4BUF,A
        RET

UART4_SENDSTR:
        CLR        A
        MOVC       A,@A+DPTR
        JZ         SEND4END
        LCALL      UART4_SEND
        INC        DPTR
        JMP        UART4_SENDSTR

SEND4END:
        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

LCALL  UART4_INIT
MOV    IE2, #10H
SETB  EA

MOV    DPTR, #STRING
LCALL  UART4_SENDSTR

LOOP:
MOV    A, RPTR
XRL   A, WPTR
ANL   A, #0FH
JZ    LOOP
MOV    A, RPTR
ANL   A, #0FH
ADD   A, #BUFFER
MOV    R0, A
MOV   A, @R0
LCALL UART4_SEND
INC   RPTR
JMP  LOOP

STRING: DB    'Uart Test !', 0DH, 0AH, 00H

END
```

## 14 串口通信

STC8A8K64D4 系列单片机具有 4 个全双工异步串行通信接口。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个相互独立的接收、发送缓冲器构成，可以同时发送和接收数据。

STC8A8K64D4 系列单片机的串口 1 有 4 种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。串口 2/串口 3/串口 4 都只有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

串口 1、串口 2、串口 3、串口 4 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

### 14.1 串口功能脚切换

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	-
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

S1\_S[1:0]: 串口 1 功能脚选择位

S1_S[1:0]	RxD	TxD
00	P3.0	P3.1
01	P3.6	P3.7
10	P1.6	P1.7
11	P4.3	P4.4

S4\_S: 串口 4 功能脚选择位

S4_S	RxD4	TxD4
0	P0.2	P0.3
1	P5.2	P5.3

S3\_S: 串口 3 功能脚选择位

S3_S	RxD3	TxD3
0	P0.0	P0.1
1	P5.0	P5.1

S2\_S: 串口 2 功能脚选择位

S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P4.0	P4.2

### 14.2 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000.0000

SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	ADH									0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	85H									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000

STC MCU

## 14.3 串口 1

### 14.3.1 串口 1 控制寄存器 (SCON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

**SM0/FE:** 当PCON寄存器中的SMOD0位为1时, 该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时, 通过UART接收器将该位置1, 必须由软件清零。当PCON寄存器中的SMOD0位为0时, 该位和SM1一起指定串口1的通信工作模式, 如下表所示:

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

**SM2:** 允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时, 如果 SM2 位为 1 且 REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 RB8) 来筛选地址帧, 若 RB8=1, 说明该帧是地址帧, 地址信息可以进入 SBUF, 并使 RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 RB8=0, 说明该帧不是地址帧, 应丢掉且保持 RI=0。在模式 2 或模式 3 中, 如果 SM2 位为 0 且 REN 位为 1, 接收机处于地址帧筛选被禁止状态, 不论收到的 RB8 为 0 或 1, 均可使接收到的信息进入 SBUF, 并使 RI=1, 此时 RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式, 在这两种方式时, SM2 应设置为 0。

**REN:** 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

**TB8:** 当串口 1 使用模式 2 或模式 3 时, TB8 为要发送的第 9 位数据, 按需要由软件置位或清 0。在模式 0 和模式 1 中, 该位不用。

**RB8:** 当串口 1 使用模式 2 或模式 3 时, RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中, 该位不用。

**TI:** 串口 1 发送中断请求标志位。在模式 0 中, 当串口发送数据第 8 位结束时, 由硬件自动将 TI 置 1, 向主机请求中断, 响应中断后 TI 必须用软件清零。在其他模式中, 则在停止位开始发送时由硬件自动将 TI 置 1, 向 CPU 发请求中断, 响应中断后 TI 必须用软件清零。

**RI:** 串口 1 接收中断请求标志位。在模式 0 中, 当串口接收第 8 位数据结束时, 由硬件自动将 RI 置 1, 向主机请求中断, 响应中断后 RI 必须用软件清零。在其他模式中, 串行接收到停止位的中间时刻由硬件自动将 RI 置 1, 向 CPU 发中断申请, 响应中断后 RI 必须由软件清零。

### 14.3.2 串口 1 数据寄存器 (SBUF)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

**SBUF:** 串口 1 数据接收/发送缓冲区。SBUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 SBUF 进行读操作, 实际是读取串口接收缓冲区, 对 SBUF 进行写操作则是触发串口开始发送数据。



### 14.3.3 电源管理寄存器 (PCON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD: 串口 1 波特率控制位

0: 串口 1 的各个模式的波特率都不加倍

1: 串口 1 模式 1 (使用模式 2 的定时器 1 作为波特率发生器时有效)、模式 2、模式 3 (使用模式 2 的定时器 1 作为波特率发生器时有效) 的波特率加倍

SMOD0: 帧错误检测控制位

0: 无帧错检测功能

1: 使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能, 即为帧错误检测标志位。

### 14.3.4 辅助寄存器 1 (AUXR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

UART\_M0x6: 串口 1 模式 0 的通讯速度控制

0: 串口 1 模式 0 的波特率不加倍, 固定为  $F_{osc}/12$

1: 串口 1 模式 0 的波特率 6 倍速, 即固定为  $F_{osc}/12*6 = F_{osc}/2$

S1ST2: 串口 1 波特率发生器选择位

0: 选择定时器 1 作为波特率发生器

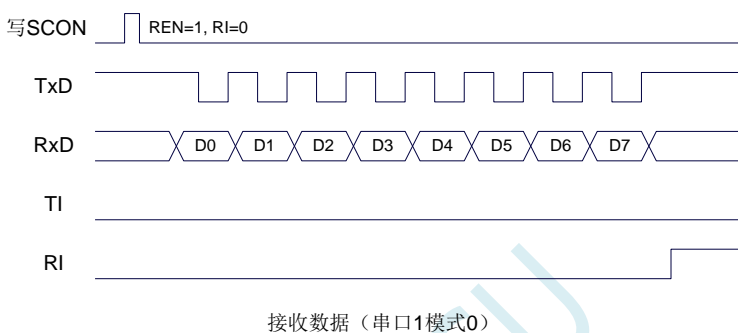
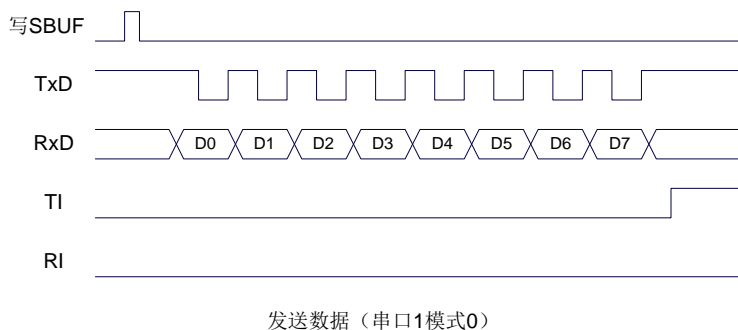
1: 选择定时器 2 作为波特率发生器

### 14.3.5 串口 1 模式 0, 模式 0 波特率计算公式

当串口 1 选择工作模式为模式 0 时, 串行通信接口工作在同步移位寄存器模式, 当串行口模式 0 的通信速度设置位 UART\_M0x6 为 0 时, 其波特率固定为系统时钟频率的 12 分频 ( $SYSClk/12$ ); 当设置 UART\_M0x6 为 1 时, 其波特率固定为系统时钟频率的 2 分频 ( $SYSClk/2$ )。RxD 为串行通讯的数据口, TxD 为同步移位脉冲输出脚, 发送、接收的是 8 位数据, 低位在先。

模式 0 的发送过程: 当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送, 串行口即将 8 位数据以  $SYSClk/12$  或  $SYSClk/2$  (由 UART\_M0x6 确定是 12 分频还是 2 分频) 的波特率从 RxD 管脚输出(从低位到高位), 发送完中断标志 TI 置 1, TxD 管脚输出同步移位脉冲信号。当写信号有效后, 相隔一个时钟, 发送控制端 SEND 有效(高电平), 允许 RxD 发送数据, 同时允许 TxD 输出同步移位脉冲。一帧(8 位)数据发送完毕时, 各控制端均恢复原状态, 只有 TI 保持高电平, 呈中断申请状态。在再次发送数据前, 必须用软件将 TI 清 0。

模式 0 的接收过程: 首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后, RxD 为串行数据输入端, TxD 为同步脉冲输出端。串行接收的波特率为  $SYSClk/12$  或  $SYSClk/2$  (由 UART\_M0x6 确定是 12 分频还是 2 分频)。当接收完成一帧数据 (8 位) 后, 控制信号复位, 中断标志 RI 被置 1, 呈中断申请状态。当再次接收时, 必须通过软件将 RI 清 0



工作于模式 0 时，必须清 0 多机通信控制位 SM2，使之不影响 TB8 位和 RB8 位。由于波特率固定为 SYSclk/12 或 SYSclk/2，无需定时器提供，直接由单片机的时钟作为同步移位脉冲。

串口 1 模式 0 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

UART_M0x6	波特率计算公式
0	波特率 = $\frac{\text{SYSclk}}{12}$
1	波特率 = $\frac{\text{SYSclk}}{2}$

### 14.3.6 串口 1 模式 1，模式 1 波特率计算公式

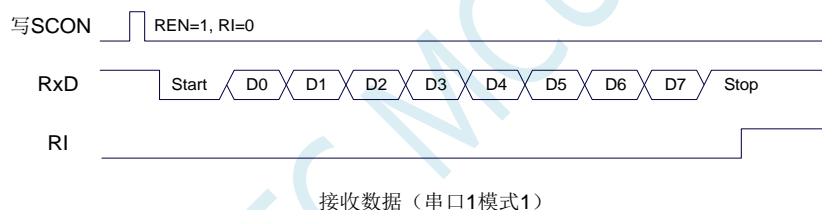
当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

模式 1 的接收过程: 当软件置位接收允许标志位 REN, 即 REN=1 时, 接收器便对 RxD 端口的信号进行检测, 当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据, 并立即复位波特率发生器的接收计数器, 将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入, 已装入的 1FFH 向左边移出, 当起始位“0”移到移位寄存器的最左边时, 使 RX 控制器作最后一次移位, 完成一帧的接收。若同时满足以下两个条件:

- RI=0;
- SM2=0 或接收到的停止位为 1。

则接收到的数据有效, 实现装载入 SBUF, 停止位进入 RB8, RI 标志位被置 1, 向主机请求中断, 若上述两条件不能同时满足, 则接收到的数据作废并丢失, 无论条件满足与否, 接收器重又检测 RxD 端口上的“1”→“0”的跳变, 继续下一帧的接收。接收有效, 在响应中断后, RI 标志位必须由软件清 0。通常情况下, 串行通信工作于模式 1 时, SM2 设置为“0”。



串口 1 的波特率是可变的, 其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时 (12 倍速), 相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示: (SYSclk 为系统工作频率)

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$
定时器1模式0	1T	定时器1重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器1重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$
定时器1模式2	1T	定时器1重载值 = $256 - \frac{2^{SMOD} \times SYSclk}{32 \times \text{波特率}}$
	12T	定时器1重载值 = $256 - \frac{2^{SMOD} \times SYSclk}{12 \times 32 \times \text{波特率}}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
		1T 模式	12T 模式	1T 模式	12T 模式	SMOD=1		SMOD=0	
						1T 模式	12T 模式	1T 模式	12T 模式
11.0592	115200	FFE8H	FFE8H	FFE8H	FFE8H	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

### 14.3.7 串口 1 模式 2, 模式 2 波特率计算公式

当 SM0、SM1 两位为 10 时, 串行口 1 工作在模式 2。串行口 1 工作模式 2 为 9 位数据异步通信 UART 模式, 其一帧的信息由 11 位组成: 1 位起始位, 8 位数据位 (低位在先), 1 位可编程位 (第 9 位数据) 和 1 位停止位。发送时可编程位 (第 9 位数据) 由 SCON 中的 TB8 提供, 可软件设置为 1 或 0, 或者可将 PSW 中的奇/偶校验位 P 值装入 TB8 (TB8 既可作为多机通信中的地址数据标志位, 又可作为数据的奇偶校验位)。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口, RxD 为接收端口, 以全双工模式进行接收/发送。

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频 (取决于 PCON 中 SMOD 的值)

串口 1 模式 2 的波特率计算公式如下表所示 (SYSclk 为系统工作频率):

SMOD	波特率计算公式
0	波特率 = $\frac{\text{SYSclk}}{64}$
1	波特率 = $\frac{\text{SYSclk}}{32}$

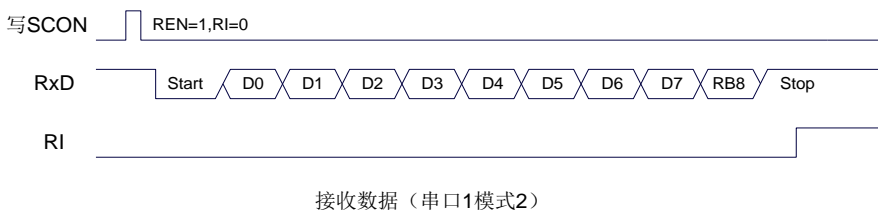
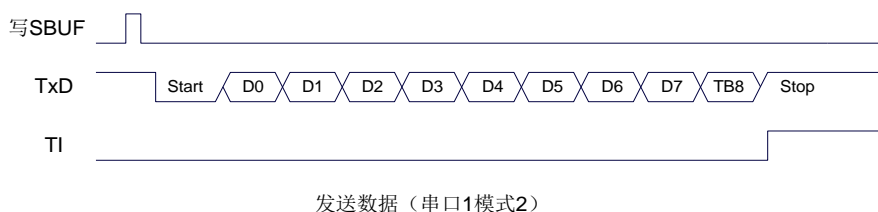
模式 2 和模式 1 相比, 除波特率发生源略有不同, 发送时由 TB8 提供给移位寄存器第 9 数据位不同外, 其余功能结构均基本相同, 其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件:

- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时, 才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中, RI 标志位被置 1, 并向主机请求中断处理。如果上述条件有一个不满足, 则刚接收到移位寄存器中的数据无效而丢失, 也不置位 RI。无论上述条件满足与否, 接收器又重新开始检测 RxD 输入端口的跳变信息, 接收下一帧的输入信息。在模式 2 中, 接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定, 为多机通信提供了方便。



### 14.3.8 串口 1 模式 3, 模式 3 波特率计算公式

当 SM0、SM1 两位为 11 时, 串行口 1 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART

模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

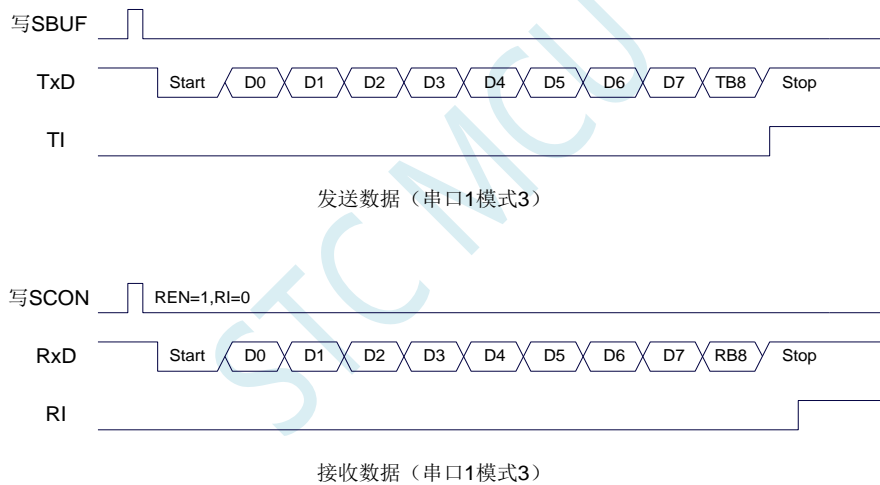
模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



串口 1 模式 3 的波特率计算公式与模式 1 是完全相同的。请参考模式 1 的波特率计算公式。

## 14.3.9 自动地址识别

### 14.3.10 串口 1 从机地址控制寄存器（SADDR，SADEN）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR：从机地址寄存器

SADEN：从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器 SADDR 和 SADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口中

断; 否则硬件自动丢弃串口数据, 而不产生中断。当众多处于空闲模式的从机链接在一起时, 只有从机地址相匹配的从机才会从空闲模式唤醒, 从而可以大大降低从机 MCU 的功耗, 即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能, 首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3 (通常都选择波特率可变的模式 3, 因为模式 2 的波特率是固定的, 不便于调节), 并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中, 第 9 位数据 (存放在 RB8 中) 为地址/数据的标志位, 当第 9 位数据为 1 时, 表示前面的 8 位数据 (存放在 SBUF 中) 为地址信息。当 SM2 被设置为 1 时, 从机 MCU 会自动过滤掉非地址数据 (第 9 位为 0 的数据), 而对 SBUF 中的地址数据 (第 9 位为 1 的数据) 自动与 SADDR 和 SADEN 所设置的本机地址进行比较, 若地址相匹配, 则会将 RI 置“1”, 并产生中断, 否则不予处理本次接收的串口数据。

从机地址的设置是通过 SADDR 和 SADEN 两个寄存器进行设置的。SADDR 为从机地址寄存器, 里面存放本机的从机地址。SADEN 为从机地址屏蔽位寄存器, 用于设置地址信息中的忽略位, 设置方法如下:

例如

SADDR = 11001010

SADEN = 10000001

则匹配地址为 1xxxxx0

即, 只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如

SADDR = 11001010

SADEN = 00001111

则匹配地址为 xxxx1010

即, 只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配, 而高 4 为被忽略, 可以为任意值。

主机可以使用广播地址 (FFH) 同时选中所有的从机来进行通讯。

## 14.4 串口 2

### 14.4.1 串口 2 控制寄存器 (S2CON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0: 指定串口2的通信工作模式, 如下表所示:

S2SM0	串口2工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S2SM2: 允许串口 2 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S2SM2 位为 1 且 S2REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S2RB8) 来筛选地址帧: 若 S2RB8=1, 说明该帧是地址帧, 地址信息可以进入 S2BUF, 并使 S2RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S2RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S2RI=0。在模式 1 中, 如果 S2SM2 位为 0 且 S2REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的 S2RB8 为 0 或 1, 均可使接收到的信息进入 S2BUF, 并使 S2RI=1, 此时 S2RB8 通常为校验位。模式 0 为非多机通信方式, 在这种方式时, 要设置 S2SM2 应为 0。

S2REN: 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

S2TB8: 当串口 2 使用模式 1 时, S2TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S2RB8: 当串口 2 使用模式 1 时, S2RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 中, 该位不用。

S2TI: 串口 2 发送中断请求标志位。在停止位开始发送时由硬件自动将 S2TI 置 1, 向 CPU 发请求中断, 响应中断后 S2TI 必须用软件清零。

S2RI: 串口 2 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S2RI 置 1, 向 CPU 发中断申请, 响应中断后 S2RI 必须由软件清零。

### 14.4.2 串口 2 数据寄存器 (S2BUF)

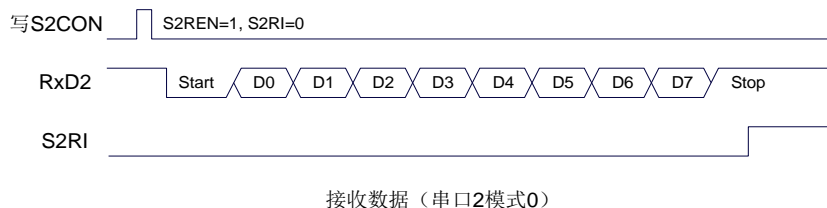
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

S2BUF: 串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S2BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S2BUF 进行写操作则是触发串口开始发送数据。

### 14.4.3 串口 2 模式 0, 模式 0 波特率计算公式

串行口 2 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD2 为数据发送口, RxD2 为数据接收口, 串行口全双工接受/发送。





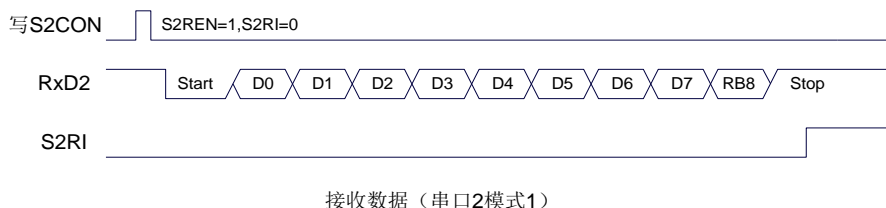
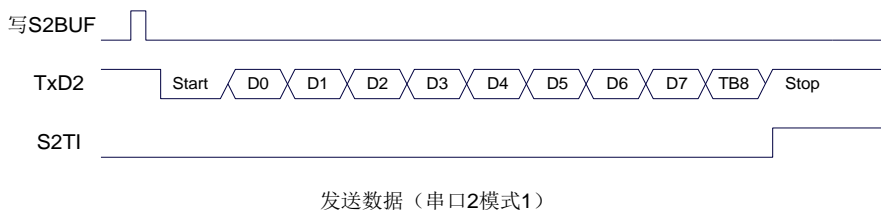
串口 2 的波特率是可变的, 其波特率由定时器 2 产生。当定时器采用 1T 模式时 (12 倍速), 相应的波特率的速度也会相应提高 12 倍。

串口 2 模式 0 的波特率计算公式如下表所示: (SYSclk 为系统工作频率)

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$

#### 14.4.4 串口 2 模式 1, 模式 1 波特率计算公式

串行口 2 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位: 1 位起始位, 9 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD2 为数据发送口, RxD2 为数据接收口, 串行口全双工接受/发送。



串口 2 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STC MCU

## 14.5 串口 3

### 14.5.1 串口 3 控制寄存器 (S3CON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0: 指定串口3的通信工作模式, 如下表所示:

S3SM0	串口3工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S3ST3: 选择串口 3 的波特率发生器

0: 选择定时器 2 为串口 3 的波特率发生器

1: 选择定时器 3 为串口 3 的波特率发生器

S3SM2: 允许串口 3 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S3SM2 位为 1 且 S3REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S3RB8) 来筛选地址帧: 若 S3RB8=1, 说明该帧是地址帧, 地址信息可以进入 S3BUF, 并使 S3RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S3RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S3RI=0。在模式 1 中, 如果 S3SM2 位为 0 且 S3REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的 S3RB8 为 0 或 1, 均可使接收到的信息进入 S3BUF, 并使 S3RI=1, 此时 S3RB8 通常为校验位。模式 0 为非多机通信方式, 在这种方式时, 要设置 S3SM2 应为 0。

S3REN: 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

S3TB8: 当串口 3 使用模式 1 时, S3TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S3RB8: 当串口 3 使用模式 1 时, S3RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 中, 该位不用。

S3TI: 串口 3 发送中断请求标志位。在停止位开始发送时由硬件自动将 S3TI 置 1, 向 CPU 发请求中断, 响应中断后 S3TI 必须用软件清零。

S3RI: 串口 3 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S3RI 置 1, 向 CPU 发中断申请, 响应中断后 S3RI 必须由软件清零。

### 14.5.2 串口 3 数据寄存器 (S3BUF)

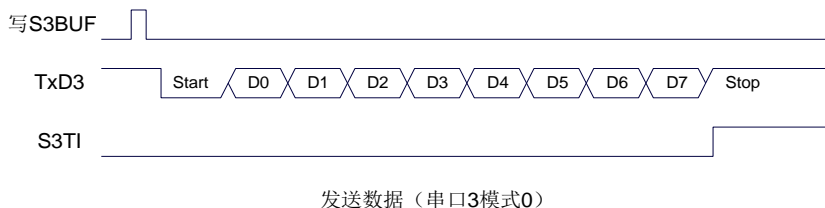
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

S3BUF: 串口 1 数据接收/发送缓冲区。S3BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S3BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S3BUF 进行写操作则是触发串口开始发送数据。

### 14.5.3 串口 3 模式 0, 模式 0 波特率计算公式

串口 3 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位,

8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。



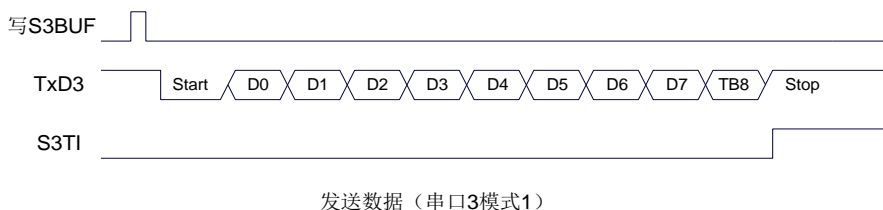
串口 3 的波特率是可变的，其波特率可由定时器 2 或定时器 3 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

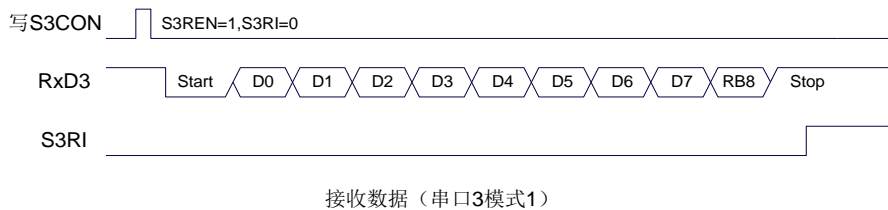
串口 3 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$
定时器3	1T	定时器3重载值 = $65536 - \frac{SYSclk}{4 \times \text{波特率}}$
	12T	定时器3重载值 = $65536 - \frac{SYSclk}{12 \times 4 \times \text{波特率}}$

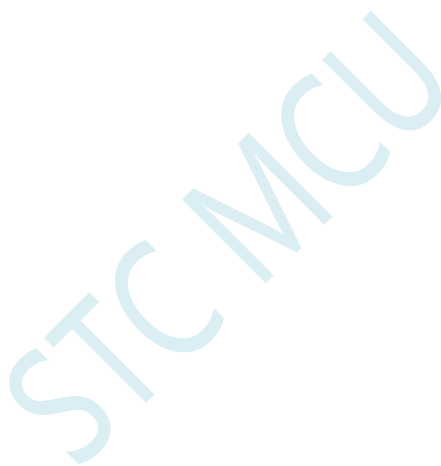
### 14.5.4 串口 3 模式 1，模式 1 波特率计算公式

串行口 3 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。





串口 3 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。



## 14.6 串口 4

### 14.6.1 串口 4 控制寄存器 (S4CON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0: 指定串口4的通信工作模式, 如下表所示:

S4SM0	串口4工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S4ST4: 选择串口 4 的波特率发生器

0: 选择定时器 2 为串口 4 的波特率发生器

1: 选择定时器 4 为串口 4 的波特率发生器

S4SM2: 允许串口 4 在模式 1 时允许多机通信控制位。在模式 1 时, 如果 S4SM2 位为 1 且 S4REN 位为 1, 则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位 (即 S4RB8) 来筛选地址帧: 若 S4RB8=1, 说明该帧是地址帧, 地址信息可以进入 S4BUF, 并使 S4RI 为 1, 进而在中断服务程序中再进行地址号比较; 若 S4RB8=0, 说明该帧不是地址帧, 应丢掉且保持 S4RI=0。在模式 1 中, 如果 S4SM2 位为 0 且 S4REN 位为 1, 接收机处于地址帧筛选被禁止状态。不论收到的 S4RB8 为 0 或 1, 均可使接收到的信息进入 S4BUF, 并使 S4RI=1, 此时 S4RB8 通常为校验位。模式 0 为非多机通信方式, 在这种方式时, 要设置 S4SM2 应为 0。

S4REN: 允许/禁止串口接收控制位

0: 禁止串口接收数据

1: 允许串口接收数据

S4TB8: 当串口 4 使用模式 1 时, S4TB8 为要发送的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位, 按需要由软件置位或清 0。在模式 0 中, 该位不用。

S4RB8: 当串口 4 使用模式 1 时, S4RB8 为接收到的第 9 位数据, 一般用作校验位或者地址帧/数据帧标志位。在模式 0 中, 该位不用。

S4TI: 串口 4 发送中断请求标志位。在停止位开始发送时由硬件自动将 S4TI 置 1, 向 CPU 发请求中断, 响应中断后 S4TI 必须用软件清零。

S4RI: 串口 4 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S4RI 置 1, 向 CPU 发中断申请, 响应中断后 S4RI 必须由软件清零。

### 14.6.2 串口 4 数据寄存器 (S4BUF)

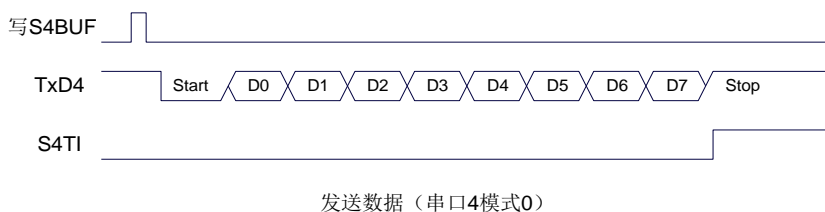
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	85H								

S4BUF: 串口 1 数据接收/发送缓冲区。S4BUF 实际是 2 个缓冲器, 读缓冲器和写缓冲器, 两个操作分别对应两个不同的寄存器, 1 个是只写寄存器 (写缓冲器), 1 个是只读寄存器 (读缓冲器)。对 S4BUF 进行读操作, 实际是读取串口接收缓冲区, 对 S4BUF 进行写操作则是触发串口开始发送数据。

### 14.6.3 串口 4 模式 0, 模式 0 波特率计算公式

串行口 4 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位: 1 位起始位, 8 位数据位 (低位在先) 和 1 位停止位。波特率可变, 可根据需要进行设置波特率。TxD4 为数据发送口,

RxD4 为数据接收口，串行口全双工接受/发送。



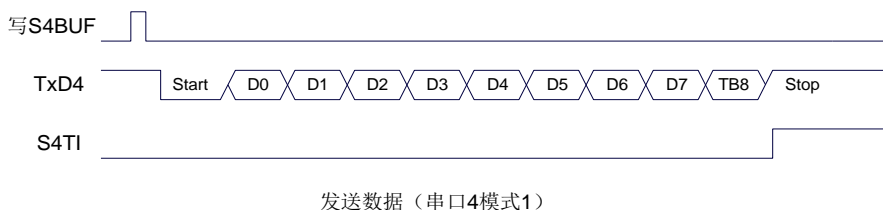
串口4的波特率是可变的，其波特率可由定时器2或定时器4产生。当定时器采用1T模式时（12倍速），相应的波特率的速度也会相应提高12倍。

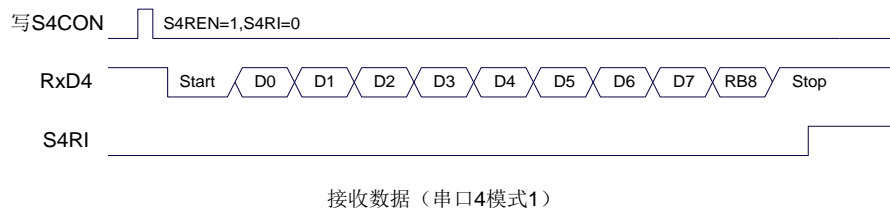
串口4模式0的波特率计算公式如下表所示：（SYSclk为系统工作频率）

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$
定时器4	1T	定时器4重载值 = $65536 - \frac{\text{SYSclk}}{4 \times \text{波特率}}$
	12T	定时器4重载值 = $65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{波特率}}$

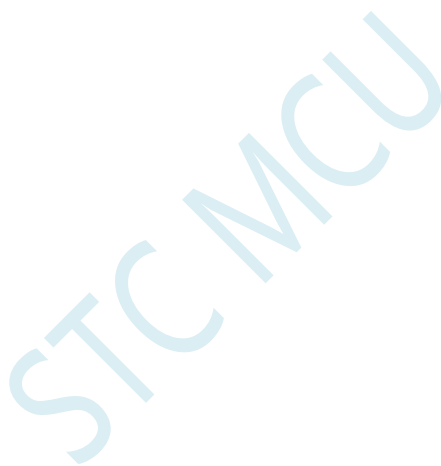
## 14.6.4 串口4模式1，模式1波特率计算公式

串行口4的模式1为9位数据位可变波特率UART工作模式。此模式一帧信息为11位：1位起始位，9位数据位（低位在先）和1位停止位。波特率可变，可根据需要进行设置波特率。TxD4为数据发送口，RxD4为数据接收口，串行口全双工接受/发送。





串口 4 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

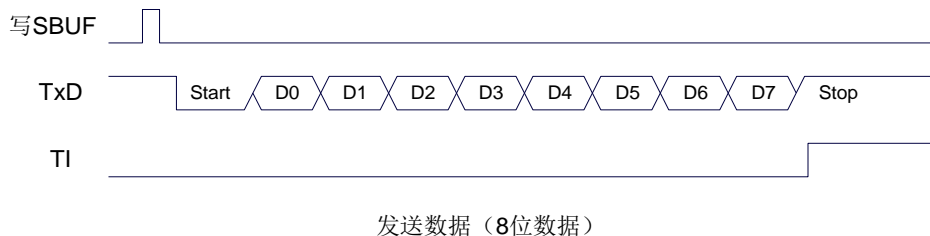




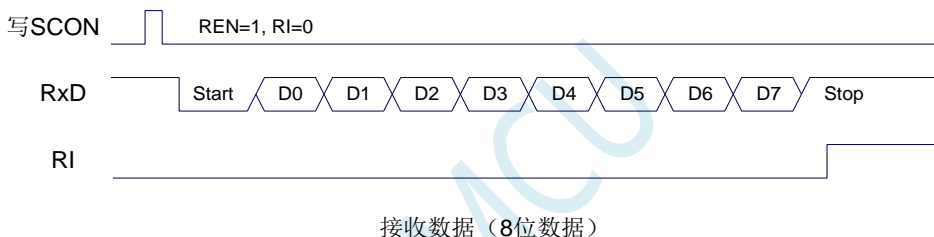
## 14.7 串口注意事项

关于串口中断请求有如下问题需要注意: (串口 1、串口 2、串口 3、串口 4 均类似, 下面以串口 1 为例进行说明)

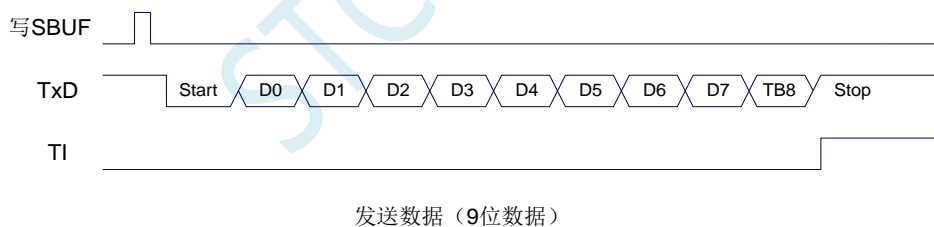
8 位数据模式时, 发送完成约 1/3 个停止位后产生 TI 中断请求, 如下图所示:



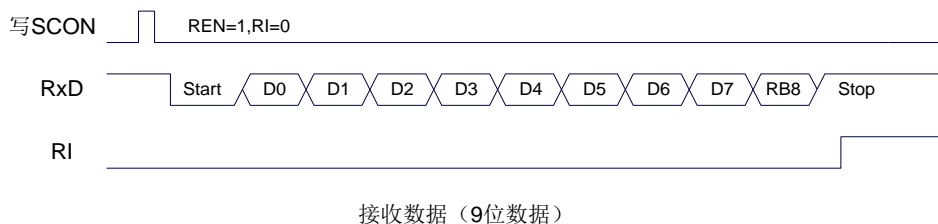
8 位数据模式时, 接收完成一半个停止位后产生 RI 中断请求, 如下图所示:



9 位数据模式时, 发送完成约 1/3 个停止位后产生 TI 中断请求:



9 位数据模式时, 一半个停止位后产生 RI 中断请求, 如下图所示:



## 14.8 范例程序

### 14.8.1 串口 1 使用定时器 2 做波特率发生器

#### C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

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

```
sfr AUXR = 0x8e;  
sfr T2H = 0xd6;  
sfr T2L = 0xd7;
```

```
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;
```

```
bit busy;  
char wptr;  
char rptr;  
char buffer[16];
```

```
void UartIsr() interrupt 4  
{  
    if (TI)  
    {  
        TI = 0;  
        busy = 0;  
    }  
    if (RI)  
    {  
        RI = 0;  
        buffer[wptr++] = SBUF;  
        wptr &= 0x0f;  
    }  
}
```

```
void UartInit()  
{  
    SCON = 0x50;  
    T2L = BRT;  
    T2H = BRT >> 8;
```

```

    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

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;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

---

## 汇编代码

---

;测试工作频率为 11.0592MHz

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H

<i>BUSY</i>	<i>BIT</i>	<i>20H.0</i>	
<i>WPTR</i>	<i>DATA</i>	<i>21H</i>	
<i>RPTR</i>	<i>DATA</i>	<i>22H</i>	
<i>BUFFER</i>	<i>DATA</i>	<i>23H</i>	<i>;16 bytes</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>ORG</i>	<i>0000H</i>	
	<i>LJMP</i>	<i>MAIN</i>	
	<i>ORG</i>	<i>0023H</i>	
	<i>LJMP</i>	<i>UART_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART_ISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>PSW,#08H</i>	
	<i>JNB</i>	<i>TI,CHKRI</i>	
	<i>CLR</i>	<i>TI</i>	
	<i>CLR</i>	<i>BUSY</i>	
<i>CHKRI:</i>			
	<i>JNB</i>	<i>RI,UARTISR_EXIT</i>	
	<i>CLR</i>	<i>RI</i>	
	<i>MOV</i>	<i>A,WPTR</i>	
	<i>ANL</i>	<i>A,#0FH</i>	
	<i>ADD</i>	<i>A,#BUFFER</i>	
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOV</i>	<i>@R0,SBUF</i>	
	<i>INC</i>	<i>WPTR</i>	
<i>UARTISR_EXIT:</i>			
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>UART_INIT:</i>			
	<i>MOV</i>	<i>SCON,#50H</i>	
	<i>MOV</i>	<i>T2L,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>T2H,#0FFH</i>	
	<i>MOV</i>	<i>AUXR,#15H</i>	
	<i>CLR</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>WPTR,#00H</i>	
	<i>MOV</i>	<i>RPTR,#00H</i>	
	<i>RET</i>		
<i>UART_SEND:</i>			
	<i>JB</i>	<i>BUSY,\$</i>	

```

        SETB     BUSY
        MOV      SBUF,A
        RET

UART_SENDSTR:
        CLR      A
        MOVC    A,@A+DPTR
        JZ       SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR

SENDEND:
        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

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL    A,WPTR
        ANL    A,#0FH
        JZ     LOOP
        MOV     A,RPTR
        ANL    A,#0FH
        ADD    A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART_SEND
        INC    RPTR
        JMP    LOOP

STRING:  DB      'Uart Test !',0DH,0AH,00H

        END

```

## 14.8.2 串口 1 使用定时器 1（模式 0）做波特率发生器

## C 语言代码

```
//测试工作频率为 11.0592MHz;
```

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

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

```
sfr AUXR = 0x8e;
```

```
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;
```

```
bit busy;
```

```
char wptr;
```

```
char rptr;
```

```
char buffer[16];
```

```
void UartIsr() interrupt 4
```

```
{  
    if (TI)  
    {  
        TI = 0;  
        busy = 0;  
    }  
    if (RI)  
    {  
        RI = 0;  
        buffer[wptr++] = SBUF;  
        wptr &= 0x0f;  
    }  
}
```

```
void UartInit()
```

```
{  
    SCON = 0x50;  
    TMOD = 0x00;  
    T1 = BRT;  
    TH1 = BRT >> 8;  
    TR1 = 1;  
    AUXR = 0x40;  
    wptr = 0x00;  
    rptr = 0x00;  
    busy = 0;  
}
```

```
void UartSend(char dat)
```

```
{
```

```

    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

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;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

<b>AUXR</b>	<b>DATA</b>	<b>8EH</b>	
<b>BUSY</b>	<b>BIT</b>	<b>20H.0</b>	
<b>WPTR</b>	<b>DATA</b>	<b>21H</b>	
<b>RPTR</b>	<b>DATA</b>	<b>22H</b>	
<b>BUFFER</b>	<b>DATA</b>	<b>23H</b>	;16 bytes
<b>P0M1</b>	<b>DATA</b>	<b>093H</b>	
<b>P0M0</b>	<b>DATA</b>	<b>094H</b>	
<b>P1M1</b>	<b>DATA</b>	<b>091H</b>	
<b>P1M0</b>	<b>DATA</b>	<b>092H</b>	
<b>P2M1</b>	<b>DATA</b>	<b>095H</b>	

<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>0023H</i>	
	<i>LJMP</i>	<i>UART_ISR</i>	
	<i>ORG</i>	<i>0100H</i>	
<i>UART_ISR:</i>			
	<i>PUSH</i>	<i>ACC</i>	
	<i>PUSH</i>	<i>PSW</i>	
	<i>MOV</i>	<i>PSW,#08H</i>	
	<i>JNB</i>	<i>TI,CHKRI</i>	
	<i>CLR</i>	<i>TI</i>	
	<i>CLR</i>	<i>BUSY</i>	
<i>CHKRI:</i>			
	<i>JNB</i>	<i>RI,UARTISR_EXIT</i>	
	<i>CLR</i>	<i>RI</i>	
	<i>MOV</i>	<i>A,WPTR</i>	
	<i>ANL</i>	<i>A,#0FH</i>	
	<i>ADD</i>	<i>A,#BUFFER</i>	
	<i>MOV</i>	<i>R0,A</i>	
	<i>MOV</i>	<i>@R0,SBUF</i>	
	<i>INC</i>	<i>WPTR</i>	
<i>UARTISR_EXIT:</i>			
	<i>POP</i>	<i>PSW</i>	
	<i>POP</i>	<i>ACC</i>	
	<i>RETI</i>		
<i>UART_INIT:</i>			
	<i>MOV</i>	<i>SCON,#50H</i>	
	<i>MOV</i>	<i>TMOD,#00H</i>	
	<i>MOV</i>	<i>TL1,#0E8H</i>	<i>;65536-11059200/115200/4=0FFE8H</i>
	<i>MOV</i>	<i>TH1,#0FFH</i>	
	<i>SETB</i>	<i>TR1</i>	
	<i>MOV</i>	<i>AUXR,#40H</i>	
	<i>CLR</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>WPTR,#00H</i>	
	<i>MOV</i>	<i>RPTR,#00H</i>	
	<i>RET</i>		
<i>UART_SEND:</i>			
	<i>JB</i>	<i>BUSY,\$</i>	
	<i>SETB</i>	<i>BUSY</i>	
	<i>MOV</i>	<i>SBUF,A</i>	
	<i>RET</i>		
<i>UART_SENDSTR:</i>			
	<i>CLR</i>	<i>A</i>	
	<i>MOVC</i>	<i>A,@A+DPTR</i>	
	<i>JZ</i>	<i>SENDEND</i>	



```

        LCALL    UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        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

        LCALL    UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR, #STRING
        LCALL    UART_SENDSTR

LOOP:
        MOV     A, RPTR
        XRL    A, WPTR
        ANL    A, #0FH
        JZ     LOOP
        MOV     A, RPTR
        ANL    A, #0FH
        ADD    A, #BUFFER
        MOV     R0, A
        MOV     A, @R0
        LCALL    UART_SEND
        INC    RPTR
        JMP    LOOP

STRING:  DB     'Uart Test !', 0DH, 0AH, 00H

        END

```

### 14.8.3 串口 1 使用定时器 1（模式 2）做波特率发生器

#### C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

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

```
#define FOSC 11059200UL
```

```
#define BRT (256 - FOSC / 115200 / 32)

sfr AUXR = 0x8e;

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;

bit busy;
char wptr;
char rptr;
char buffer[16];
```

```
void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}
```

```
void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TLI = BRT;
    TH1 = BRT;
    TRI = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}
```

```
void UartSendStr(char *p)
{
```

```

    while (*p)
    {
        UartSend(*p++);
    }
}

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;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

<b>AUXR</b>	<b>DATA</b>	<b>8EH</b>	
<b>BUSY</b>	<b>BIT</b>	<b>20H.0</b>	
<b>WPTR</b>	<b>DATA</b>	<b>21H</b>	
<b>RPTR</b>	<b>DATA</b>	<b>22H</b>	
<b>BUFFER</b>	<b>DATA</b>	<b>23H</b>	;16 bytes
<b>P0M1</b>	<b>DATA</b>	<b>093H</b>	
<b>P0M0</b>	<b>DATA</b>	<b>094H</b>	
<b>P1M1</b>	<b>DATA</b>	<b>091H</b>	
<b>P1M0</b>	<b>DATA</b>	<b>092H</b>	
<b>P2M1</b>	<b>DATA</b>	<b>095H</b>	
<b>P2M0</b>	<b>DATA</b>	<b>096H</b>	
<b>P3M1</b>	<b>DATA</b>	<b>0B1H</b>	
<b>P3M0</b>	<b>DATA</b>	<b>0B2H</b>	
<b>P4M1</b>	<b>DATA</b>	<b>0B3H</b>	
<b>P4M0</b>	<b>DATA</b>	<b>0B4H</b>	
<b>P5M1</b>	<b>DATA</b>	<b>0C9H</b>	
<b>P5M0</b>	<b>DATA</b>	<b>0CAH</b>	

```

    ORG      0000H
    LJMP     MAIN
    ORG      0023H
    LJMP     UART_ISR

    ORG      0100H

UART_ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#08H

    JNB     TI,CHKRI
    CLR     TI
    CLR     BUSY

CHKRI:
    JNB     RI,UARTISR_EXIT
    CLR     RI
    MOV     A,WPTR
    ANL    A,#0FH
    ADD    A,#BUFFER
    MOV     R0,A
    MOV     @R0,SBUF
    INC    WPTR

UARTISR_EXIT:
    POP     PSW
    POP     ACC
    RETI

UART_INIT:
    MOV     SCON,#50H
    MOV     TMOD,#20H
    MOV     T1L,#0FDH           ;256-11059200/115200/32=0FDH
    MOV     TH1,#0FDH
    SETB   TRI
    MOV     AUXR,#40H
    CLR     BUSY
    MOV     WPTR,#00H
    MOV     RPTR,#00H
    RET

UART_SEND:
    JB      BUSY,$
    SETB   BUSY
    MOV     SBUF,A
    RET

UART_SENDSTR:
    CLR     A
    MOVC   A,@A+DPTR
    JZ     SENDEND
    LCALL  UART_SEND
    INC    DPTR
    JMP    UART_SENDSTR

SENDEND:
    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

LCALL   UART_INIT
SETB    ES
SETB    EA

MOV     DPTR, #STRING
LCALL   UART_SENDSTR

LOOP:
MOV     A, RPTR
XRL    A, WPTR
ANL    A, #0FH
JZ     LOOP
MOV     A, RPTR
ANL    A, #0FH
ADD    A, #BUFFER
MOV     R0, A
MOV     A, @R0
LCALL  UART_SEND
INC    RPTR
JMP    LOOP

STRING:  DB      'Uart Test !', 0DH, 0AH, 00H

END

```

## 14.8.4 串口 2 使用定时器 2 做波特率发生器

### C 语言代码

```
//测试工作频率为 11.0592MHz;
```

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

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

```
sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;
sfr S2CON = 0x9a;
sfr S2BUF = 0x9b;
```

```
sfr    IE2        = 0xaf;

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;
```

```
bit    busy;
char   wptr;
char   rptr;
char   buffer[16];
```

```
void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}
```

```
void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2SEND(*p++);
    }
}
```

```

}

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;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

AUXR	DATA	8EH	
T2H	DATA	0D6H	
T2L	DATA	0D7H	
S2CON	DATA	9AH	
S2BUF	DATA	9BH	
IE2	DATA	0AFH	
BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes
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        0043H
           LJMP      UART2_ISR

           ORG        0100H

UART2_ISR:
           PUSH     ACC
           PUSH     PSW
           MOV      PSW,#08H

           MOV      A,S2CON
           JNB     ACC.1,CHKRI
           ANL     S2CON,#NOT 02H
           CLR     BUSY

CHKRI:
           JNB     ACC.0,UART2ISR_EXIT
           ANL     S2CON,#NOT 01H
           MOV     A,WPTR
           ANL     A,#0FH
           ADD     A,#BUFFER
           MOV     R0,A
           MOV     @R0,S2BUF
           INC     WPTR

UART2ISR_EXIT:
           POP     PSW
           POP     ACC
           RETI

UART2_INIT:
           MOV     S2CON,#10H
           MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
           MOV     T2H,#0FFH
           MOV     AUXR,#14H
           CLR     BUSY
           MOV     WPTR,#00H
           MOV     RPTR,#00H
           RET

UART2_SEND:
           JB      BUSY,$
           SETB   BUSY
           MOV     S2BUF,A
           RET

UART2_SENDSTR:
           CLR     A
           MOVC   A,@A+DPTR
           JZ     SEND2END
           LCALL  UART2_SEND
           INC     DPTR
           JMP     UART2_SENDSTR

SEND2END:
           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

LCALL   UART2_INIT
MOV     IE2, #01H
SETB    EA

MOV     DPTR, #STRING
LCALL   UART2_SENDSTR

LOOP:
MOV     A, RPTR
XRL    A, WPTR
ANL    A, #0FH
JZ     LOOP
MOV     A, RPTR
ANL    A, #0FH
ADD    A, #BUFFER
MOV     R0, A
MOV     A, @R0
LCALL   UART2_SEND
INC    RPTR
JMP    LOOP

STRING:  DB      'Uart Test !', 0DH, 0AH, 00H

END

```

## 14.8.5 串口 3 使用定时器 2 做波特率发生器

### C 语言代码

//测试工作频率为 11.0592MHz;

```

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

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

sfr AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;
sfr S3CON = 0xac;
sfr S3BUF = 0xad;

```

```
sfr    IE2        = 0xaf;

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;
```

```
bit    busy;
char   wptr;
char   rptr;
char   buffer[16];
```

```
void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

```
void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}
```

```
void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}
```

```

}

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;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

AUXR	DATA	8EH	
T2H	DATA	0D6H	
T2L	DATA	0D7H	
S3CON	DATA	0ACH	
S3BUF	DATA	0ADH	
IE2	DATA	0AFH	
BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes
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        008BH
           LJMP       UART3_ISR

           ORG        0100H

UART3_ISR:
           PUSH       ACC
           PUSH       PSW
           MOV        PSW,#08H

           MOV        A,S3CON
           JNB        ACC.1,CHKRI
           ANL        S3CON,#NOT 02H
           CLR        BUSY

CHKRI:
           JNB        ACC.0,UART3ISR_EXIT
           ANL        S3CON,#NOT 01H
           MOV        A,WPTR
           ANL        A,#0FH
           ADD        A,#BUFFER
           MOV        R0,A
           MOV        @R0,S3BUF
           INC        WPTR

UART3ISR_EXIT:
           POP        PSW
           POP        ACC
           RETI

UART3_INIT:
           MOV        S3CON,#10H
           MOV        T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
           MOV        T2H,#0FFH
           MOV        AUXR,#14H
           CLR        BUSY
           MOV        WPTR,#00H
           MOV        RPTR,#00H
           RET

UART3_SEND:
           JB         BUSY,$
           SETB       BUSY
           MOV        S3BUF,A
           RET

UART3_SENDSTR:
           CLR        A
           MOVC       A,@A+DPTR
           JZ         SEND3END
           LCALL      UART3_SEND
           INC        DPTR
           JMP        UART3_SENDSTR

SEND3END:
           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

LCALL   UART3_INIT
MOV     IE2, #08H
SETB    EA

MOV     DPTR, #STRING
LCALL   UART3_SENDSTR

LOOP:
MOV     A, RPTR
XRL    A, WPTR
ANL    A, #0FH
JZ     LOOP
MOV     A, RPTR
ANL    A, #0FH
ADD    A, #BUFFER
MOV     R0, A
MOV     A, @R0
LCALL   UART3_SEND
INC    RPTR
JMP    LOOP

STRING:  DB      'Uart Test !', 0DH, 0AH, 00H

END

```

## 14.8.6 串口 3 使用定时器 3 做波特率发生器

### C 语言代码

//测试工作频率为 11.0592MHz;

```

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

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

sfr T4T3M = 0xd1;
sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T3L = 0xd5;
sfr T3H = 0xd4;

```

```

sfr    T2L      = 0xd7;
sfr    T2H      = 0xd6;
sfr    S3CON    = 0xac;
sfr    S3BUF    = 0xad;
sfr    IE2      = 0xaf;

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;

```

```

bit    busy;
char   wptr;
char   rptr;
char   buffer[16];

```

```
void Uart3Isr() interrupt 17
```

```

{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

```

```
void Uart3Init()
```

```

{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

```

```
void Uart3Send(char dat)
```

```

{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

```

```
void Uart3SendStr(char *p)
```

```
{
```

```

    while (*p)
    {
        Uart3SEND(*p++);
    }
}

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;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

T4T3M	DATA	0D1H	
T4L	DATA	0D3H	
T4H	DATA	0D2H	
T3L	DATA	0D5H	
T3H	DATA	0D4H	
T2L	DATA	0D7H	
T2H	DATA	0D6H	
S3CON	DATA	0ACH	
S3BUF	DATA	0ADH	
IE2	DATA	0AFH	
BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes
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          008BH
          LJMP         UART3_ISR

          ORG          0100H

UART3_ISR:
          PUSH         ACC
          PUSH         PSW
          MOV          PSW,#08H

          MOV          A,S3CON
          JNB         ACC.1,CHKRI
          ANL         S3CON,#NOT 02H
          CLR         BUSY

CHKRI:
          JNB         ACC.0,UART3ISR_EXIT
          ANL         S3CON,#NOT 01H
          MOV          A,WPTR
          ANL         A,#0FH
          ADD         A,#BUFFER
          MOV          R0,A
          MOV          @R0,S3BUF
          INC         WPTR

UART3ISR_EXIT:
          POP          PSW
          POP          ACC
          RETI

UART3_INIT:
          MOV          S3CON,#50H
          MOV          T3L,#0E8H          ;65536-11059200/115200/4=0FFE8H
          MOV          T3H,#0FFH
          MOV          T4T3M,#0AH
          CLR         BUSY
          MOV          WPTR,#00H
          MOV          RPTR,#00H
          RET

UART3_SEND:
          JB          BUSY,$
          SETB        BUSY
          MOV          S3BUF,A
          RET

UART3_SENDSTR:
          CLR         A
          MOVC        A,@A+DPTR

```



```

        JZ          SEND3END
        LCALL       UART3_SEND
        INC         DPTR
        JMP         UART3_SENDSTR
SEND3END:
        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

        LCALL       UART3_INIT
        MOV        IE2, #08H
        SETB       EA

        MOV        DPTR, #STRING
        LCALL       UART3_SENDSTR

LOOP:
        MOV        A, RPTR
        XRL        A, WPTR
        ANL        A, #0FH
        JZ         LOOP
        MOV        A, RPTR
        ANL        A, #0FH
        ADD        A, #BUFFER
        MOV        R0, A
        MOV        A, @R0
        LCALL       UART3_SEND
        INC        RPTR
        JMP        LOOP

STRING:  DB        'Uart Test !', 0DH, 0AH, 00H

        END

```

## 14.8.7 串口 4 使用定时器 2 做波特率发生器

### C 语言代码

```
//测试工作频率为 11.0592MHz;
```

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

```

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

sfr AUXR        = 0x8e;
sfr T2H         = 0xd6;
sfr T2L         = 0xd7;
sfr S4CON       = 0x84;
sfr S4BUF       = 0x85;
sfr IE2         = 0xaf;

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;

bit busy;
char wptr;
char rptr;
char buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

```

```

}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

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;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

AUXR	DATA	8EH	
T2H	DATA	0D6H	
T2L	DATA	0D7H	
S4CON	DATA	84H	
S4BUF	DATA	85H	
IE2	DATA	0AFH	
BUSY	BIT	20H.0	
WPTR	DATA	21H	
RPTR	DATA	22H	
BUFFER	DATA	23H	;16 bytes
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          0093H
          LJMP         UART4_ISR

          ORG          0100H

UART4_ISR:
          PUSH        ACC
          PUSH        PSW
          MOV          PSW,#08H

          MOV          A,S4CON
          JNB         ACC.1,CHKRI
          ANL         S4CON,#NOT 02H
          CLR         BUSY

CHKRI:
          JNB         ACC.0,UART4ISR_EXIT
          ANL         S4CON,#NOT 01H
          MOV          A,WPTR
          ANL         A,#0FH
          ADD         A,#BUFFER
          MOV          R0,A
          MOV          @R0,S4BUF
          INC         WPTR

UART4ISR_EXIT:
          POP          PSW
          POP          ACC
          RETI

UART4_INIT:
          MOV          S4CON,#10H
          MOV          T2L,#0E8H          ;65536-11059200/115200/4=0FFE8H
          MOV          T2H,#0FFH
          MOV          AUXR,#14H
          CLR         BUSY
          MOV          WPTR,#00H
          MOV          RPTR,#00H
          RET

UART4_SEND:
          JB          BUSY,$
          SETB        BUSY
          MOV          S4BUF,A
          RET

UART4_SENDSTR:
          CLR         A
          MOVC        A,@A+DPTR

```

```

        JZ          SEND4END
        LCALL       UART4_SEND
        INC         DPTR
        JMP         UART4_SENDSTR
SEND4END:
        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

        LCALL       UART4_INIT
        MOV        IE2, #10H
        SETB       EA

        MOV        DPTR, #STRING
        LCALL       UART4_SENDSTR

LOOP:
        MOV        A, RPTR
        XRL        A, WPTR
        ANL        A, #0FH
        JZ         LOOP
        MOV        A, RPTR
        ANL        A, #0FH
        ADD        A, #BUFFER
        MOV        R0, A
        MOV        A, @R0
        LCALL       UART4_SEND
        INC        RPTR
        JMP        LOOP

STRING:  DB        'Uart Test !', 0DH, 0AH, 00H

        END

```

## 14.8.8 串口 4 使用定时器 4 做波特率发生器

### C 语言代码

```
//测试工作频率为 11.0592MHz;
```

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

```

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

sfr T4T3M      = 0xd1;
sfr T4L        = 0xd3;
sfr T4H        = 0xd2;
sfr T3L        = 0xd5;
sfr T3H        = 0xd4;
sfr T2L        = 0xd7;
sfr T2H        = 0xd6;
sfr S4CON      = 0x84;
sfr S4BUF      = 0x85;
sfr IE2        = 0xaf;

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;

bit busy;
char wptr;
char rptr;
char buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)

```

```

{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

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;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

T4T3M	DATA	0D1H
T4L	DATA	0D3H
T4H	DATA	0D2H
T3L	DATA	0D5H
T3H	DATA	0D4H
T2L	DATA	0D7H
T2H	DATA	0D6H
S4CON	DATA	84H
S4BUF	DATA	85H
IE2	DATA	0AFH

```

BUSY      BIT      20H.0
WPTR     DATA     21H
RPTR     DATA     22H
BUFFER    DATA     23H                                ;16 bytes

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      0093H
LJMP     UART4_ISR

ORG      0100H

UART4_ISR:
PUSH     ACC
PUSH     PSW
MOV      PSW,#08H

MOV      A,S4CON
JNB     ACC.1,CHKRI
ANL     S4CON,#NOT 02H
CLR     BUSY

CHKRI:
JNB     ACC.0,UART4ISR_EXIT
ANL     S4CON,#NOT 01H
MOV     A,WPTR
ANL     A,#0FH
ADD     A,#BUFFER
MOV     R0,A
MOV     @R0,S4BUF
INC     WPTR

UART4ISR_EXIT:
POP     PSW
POP     ACC
RETI

UART4_INIT:
MOV     S4CON,#50H
MOV     T4L,#0E8H                                ;65536-11059200/115200/4=0FFE8H
MOV     T4H,#0FFH
MOV     T4T3M,#0A0H
CLR     BUSY
MOV     WPTR,#00H
MOV     RPTR,#00H
RET

UART4_SEND:

```



```
        JB      BUSY,$
        SETB    BUSY
        MOV     S4BUF,A
        RET

UART4_SENDSTR:
        CLR     A
        MOVC   A,@A+DPTR
        JZ      SEND4END
        LCALL  UART4_SEND
        INC     DPTR
        JMP     UART4_SENDSTR

SEND4END:
        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

        LCALL  UART4_INIT
        MOV     IE2,#10H
        SETB    EA

        MOV     DPTR,#STRING
        LCALL  UART4_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL    A,WPTR
        ANL    A,#0FH
        JZ     LOOP
        MOV     A,RPTR
        ANL    A,#0FH
        ADD    A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL  UART4_SEND
        INC    RPTR
        JMP    LOOP

STRING:  DB     'Uart Test !',0DH,0AH,00H

        END
```

## 14.8.9 串口多机通讯

现参考 STC15 系列数据手册，后续补充

STC MCU

## 14.8.10 串口转 LIN 总线

### C 语言代码

```
//测试工作频率为22.1184MHz
```

```
***** 功能说明 *****
```

本例程基于 STC8H8K64U 为主控芯片的实验箱 8 进行编写测试, STC8G、STC8H 系列芯片可通用参考。  
通过 UART 接口连接 LIN 收发器实现 LIN 总线信号收发测试例程。

UART1 通过串口工具连接电脑。

UART2 外接 LIN 收发器(TJA1020/1), 连接 LIN 总线。

将电脑串口发送的数据转发到 LIN 总线; 从 LIN 总线接收到的数据转发到电脑串口。

默认传输速率: 9600 波特率, 发送 LIN 数据前切换波特率, 发送 13 个显性间隔信号。

下载时, 选择时钟 22.1184MHz (用户可自行修改频率)。

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

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

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

```
#define MAIN_Fosc 22118400L
```

```
typedef unsigned char u8;
```

```
typedef unsigned int u16;
```

```
typedef unsigned long u32;
```

```
sfr AUXR = 0x8E;
```

```
sfr S2CON = 0x9A;
```

```
sfr S2BUF = 0x9B;
```

```
sfr TH2 = 0xD6;
```

```
sfr TL2 = 0xD7;
```

```
sfr IE2 = 0xAF;
```

```
sfr INT_CLKO = 0x8F;
```

```
sfr P_SW1 = 0xA2;
```

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

```
sfr P4 = 0xC0;
```

```
sfr P5 = 0xC8;
```

```
sfr P6 = 0xE8;
```

```
sfr P7 = 0xF8;
```

```
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;
```

```
sfr P6M1 = 0xCB;
```

```
sfr P6M0 = 0xCC;
```

```
sfr P7M1 = 0xE1;
```

```
sfr P7M0 = 0xE2;
```

```
sbit P00 = P0^0;
```

```

sbit P01 = P0^1;
sbit P02 = P0^2;
sbit P03 = P0^3;
sbit P04 = P0^4;
sbit P05 = P0^5;
sbit P06 = P0^6;
sbit P07 = P0^7;
sbit P10 = P1^0;
sbit P11 = P1^1;
sbit P12 = P1^2;
sbit P13 = P1^3;
sbit P14 = P1^4;
sbit P15 = P1^5;
sbit P16 = P1^6;
sbit P17 = P1^7;
sbit P20 = P2^0;
sbit P21 = P2^1;
sbit P22 = P2^2;
sbit P23 = P2^3;
sbit P24 = P2^4;
sbit P25 = P2^5;
sbit P26 = P2^6;
sbit P27 = P2^7;
sbit P30 = P3^0;
sbit P31 = P3^1;
sbit P32 = P3^2;
sbit P33 = P3^3;
sbit P34 = P3^4;
sbit P35 = P3^5;
sbit P36 = P3^6;
sbit P37 = P3^7;
sbit P40 = P4^0;
sbit P41 = P4^1;
sbit P42 = P4^2;
sbit P43 = P4^3;
sbit P44 = P4^4;
sbit P45 = P4^5;
sbit P46 = P4^6;
sbit P47 = P4^7;
sbit P50 = P5^0;
sbit P51 = P5^1;
sbit P52 = P5^2;
sbit P53 = P5^3;
sbit P54 = P5^4;
sbit P55 = P5^5;
sbit P56 = P5^6;
sbit P57 = P5^7;

sbit SLP_N = P2^4; //0: Sleep

/***** 用户定义宏 *****/

#define Baudrate1 (65536UL - (MAIN_Fosc / 4) / 9600UL)
#define Baudrate2 (65536UL - (MAIN_Fosc / 4) / 9600UL)

#define Baudrate_Break (65536UL - (MAIN_Fosc / 4) / 6647UL) //发送显性间隔信号波特率

#define UART1_BUF_LENGTH 32
#define UART2_BUF_LENGTH 32

```

```

#define LIN_ID 0x31

u8 TX1_Cnt; //发送计数
u8 RX1_Cnt; //接收计数
u8 TX2_Cnt; //发送计数
u8 RX2_Cnt; //接收计数
bit B_TX1_Busy; //发送忙标志
bit B_TX2_Busy; //发送忙标志
u8 RX1_TimeOut;
u8 RX2_TimeOut;

u8 xdata RX1_Buffer[UART1_BUF_LENGTH]; //接收缓冲
u8 xdata RX2_Buffer[UART2_BUF_LENGTH]; //接收缓冲

void UART1_config(u8 brt);
void UART2_config(u8 brt);
void PrintString1(u8 *puts);
void delay_ms(u8 ms);
void UART1_TxByte(u8 dat);
void UART2_TxByte(u8 dat);
void Lin_Send(u8 *puts);
void SetTimer2Baudrate(u16 dat);

//=====
// 函数: void main(void)
// 描述: 主函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void main(void)
{
    u8 i;

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

    UART1_config(1);
    UART2_config(2);
    EA = 1; //允许全局中断
    SLP_N = 1;

    PrintString1("STC8H8K64U UART1 Test Programme!\r\n"); //UART1 发送一个字符串

    while (1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)
        {
            if(--RX1_TimeOut == 0) //超时,则串口接收结束

```

```

        {
            if(RX1_Cnt > 0)
            {
                Lin_Send(RX1_Buffer);           //将 UART1 收到的数据发送到 LIN 总线上
            }
            RX1_Cnt = 0;
        }
    }

    if(RX2_TimeOut > 0)
    {
        if(--RX2_TimeOut == 0)                 //超时,则串口接收结束
        {
            if(RX2_Cnt > 0)
            {
                for (i=0; I < RX2_Cnt; i++)     //遇到停止符0 结束
                {
                    UART1_TxByte(RX2_Buffer[i]); //从 LIN 总线收到的数据发送到 UART1
                }
            }
            RX2_Cnt = 0;
        }
    }
}
}
}
}
}

```

```

=====
// 函数: void delay_ms(unsigned char ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms 数, 这里只支持 1~255ms. 自动适应主时钟。
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
=====

```

```

void delay_ms(u8 ms)
{
    u16 i;
    do{
        i = MAIN_Fosc / 10000;
        while(--i);           //10T per loop
    }while(--ms);
}

```

```

=====
// 函数: u8 Lin_CheckPID(u8 id)
// 描述: ID 码加上校验符, 转成 PID 码。
// 参数: ID 码。
// 返回: PID 码。
// 版本: VER1.0
// 日期: 2020-12-2
// 备注:
=====

```

```

u8 Lin_CheckPID(u8 id)
{
    u8 returnpid;
    u8 P0;
    u8 P1;
}

```

```

    P0 = (((id)^(id>>1)^(id>>2)^(id>>4))&0x01)<<6;
    P1 = (((~((id>>1)^(id>>3)^(id>>4)^(id>>5)))&0x01)<<7;

    returnpid = id|P0|P1 ;

    return returnpid ;
}

//=====
// 函数: u8 LINCalcChecksum(u8 *dat)
// 描述: 计算校验码。
// 参数: 数据场传输的数据。
// 返回: 校验码。
// 版本: VER1.0
// 日期: 2020-12-2
// 备注:
//=====
static u8 LINCalcChecksum(u8 *dat)
{
    u16 sum = 0;
    u8 i;

    for(i = 0; i < 8; i++)
    {
        sum += dat[i];
        if(sum & 0xFF00)
        {
            sum = (sum & 0x00FF) + 1;
        }
    }
    sum ^= 0x00FF;
    return (u8)sum;
}

//=====
// 函数: void Lin_SendBreak(void)
// 描述: 发送显性间隔信号。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2020-12-2
// 备注:
//=====
void Lin_SendBreak(void)
{
    SetTimer2Baudrate(Baudrate_Break);
    UART2_TxByte(0);
    SetTimer2Baudrate(Baudrate2);
}

//=====
// 函数: void Lin_Send(u8 *puts)
// 描述: 发送 LIN 总线报文。
// 参数: 待发送的数据场内容。
// 返回: none.
// 版本: VER1.0
// 日期: 2020-12-2
// 备注:
//=====

```

```

void Lin_Send(u8 *puts)
{
    u8 i;

    Lin_SendBreak();                //Break
    UART2_TxByte(0x55);             //SYNC
    UART2_TxByte(Lin_CheckPID(LIN_ID)); //LIN ID
    for(i=0;i<8;i++)
    {
        UART2_TxByte(puts[i]);
    }
    UART2_TxByte(LINCalcChecksum(puts));
}

//=====
// 函数: void UART1_TxByte(u8 dat)
// 描述: 发送一个字节.
// 参数: 无
// 返回: 无
// 版本: V1.0, 2014-6-30
//=====
void UART1_TxByte(u8 dat)
{
    SBUF = dat;
    B_TX1_Busy = 1;
    while(B_TX1_Busy);
}

//=====
// 函数: void UART2_TxByte(u8 dat)
// 描述: 发送一个字节.
// 参数: 无
// 返回: 无
// 版本: V1.0, 2014-6-30
//=====
void UART2_TxByte(u8 dat)
{
    S2BUF = dat;
    B_TX2_Busy = 1;
    while(B_TX2_Busy);
}

//=====
// 函数: void PrintString1(u8 *puts)
// 描述: 串口1 发送字符串函数.
// 参数: puts:      字符串指针.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void PrintString1(u8 *puts)
{
    for (; *puts != 0; puts++) //遇到停止符0 结束
    {
        SBUF = *puts;
        B_TX1_Busy = 1;
        while(B_TX1_Busy);
    }
}

```



```

}

//=====================================================
// 函数: void PrintString2(u8 *puts)
// 描述: 串口2 发送字符串函数。
// 参数: puts:          字符串指针。
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====================================================
//void PrintString2(u8 *puts)
//{
//    for (; *puts != 0; puts++)                //遇到停止符0 结束
//    {
//        S2BUF = *puts;
//        B_TX2_Busy = 1;
//        while(B_TX2_Busy);
//    }
//}

//=====================================================
// 函数: SetTimer2Baudraye(u16 dat)
// 描述: 设置Timer2 做波特率发生器。
// 参数: dat: Timer2 的重装值。
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====================================================
void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                //Timer stop
    AUXR &= ~(1<<3);                //Timer2 set As Timer
    AUXR |= (1<<2);                 //Timer2 set as IT mode
    TH2 = dat / 256;
    TL2 = dat % 256;
    IE2 &= ~(1<<2);                //禁止中断
    AUXR |= (1<<4);                //Timer run enable
}

//=====================================================
// 函数: void UART1_config(u8 brt)
// 描述: UART1 初始化函数。
// 参数: brt: 选择波特率, 2: 使用Timer2 做波特率, 其它值: 使用Timer1 做波特率
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====================================================
void UART1_config(u8 brt)
{
    /***** 波特率使用定时器2 *****/
    if(brt == 2)
    {
        AUXR |= 0x01;                //S1 BRT Use Timer2;
        SetTimer2Baudraye(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 1T mode
    TMOD &= ~(1<<6);        //Timer1 set As Timer
    TMOD &= ~0x30;         //Timer1_16bitAutoReload;
    TH1 = (u8)(Baudrate1 / 256);
    TL1 = (u8)(Baudrate1 % 256);
    ET1 = 0;                //禁止中断
    INT_CLKO &= ~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 |= 0x80; //UART1switch to: 0x00: P3.0 P3.1,
//                               0x40: P3.6 P3.7,
//                               0x80: P1.6 P1.7,
//                               0xc0: P4.3 P4.4

B_TX1_Busy = 0;
TX1_Cnt = 0;
RX1_Cnt = 0;
}

//=====
// 函数: void UART2_config(u8 brt)
// 描述: UART2 初始化函数。
// 参数: brt: 选择波特率, 2: 使用Timer2 做波特率, 其它值: 无效
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void UART2_config(u8 brt)
{
    if(brt == 2)
    {
        SetTimer2Baudrate(Baudrate2);

        S2CON &= ~(1<<7); //8 位数据, 1 位起始位, 1 位停止位, 无校验
        IE2 |= 1;         //允许中断
        S2CON |= (1<<4);  //允许接收
        P_SW2 &= ~0x01;
// P_SW2 |= 1;         //UART2 switch to: 0: P1.0/P1.1, 1: P4.6/P4.7

        B_TX2_Busy = 0;
        TX2_Cnt = 0;
        RX2_Cnt = 0;
    }
}

```

```
}

//=====
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUF_LENGTH) RX1_Cnt = 0;
        RX1_Buffer[RX1_Cnt] = SBUF;
        RX1_Cnt++;
        RX1_TimeOut = 5;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

//=====
// 函数: void UART2_int (void) interrupt UART2_VECTOR
// 描述: UART2 中断函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void UART2_int (void) interrupt 8
{
    if((S2CON & 1) != 0)
    {
        S2CON &= ~1; //Clear Rx flag
        if(RX2_Cnt >= UART2_BUF_LENGTH) RX2_Cnt = 0;
        RX2_Buffer[RX2_Cnt] = S2BUF;
        RX2_Cnt++;
        RX2_TimeOut = 5;
    }

    if((S2CON & 2) != 0)
    {
        S2CON &= ~2; //Clear Tx flag
        B_TX2_Busy = 0;
    }
}
}
```

## 15 比较器，掉电检测，内部 1.19V 参考信号源

STC8A8K64D4 系列单片机内部集成了一个比较器。比较器的正极可以是 P3.7 端口、P5.0 端口、P5.1 端口或者 ADC 的模拟输入通道，负极可以是 P3.6 端口或者是内部 BandGap 经过 OP 后的 REFV 电压(内部固定比较电压)。[通过多路选择器和分时复用可实现多个比较器的应用](#)

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

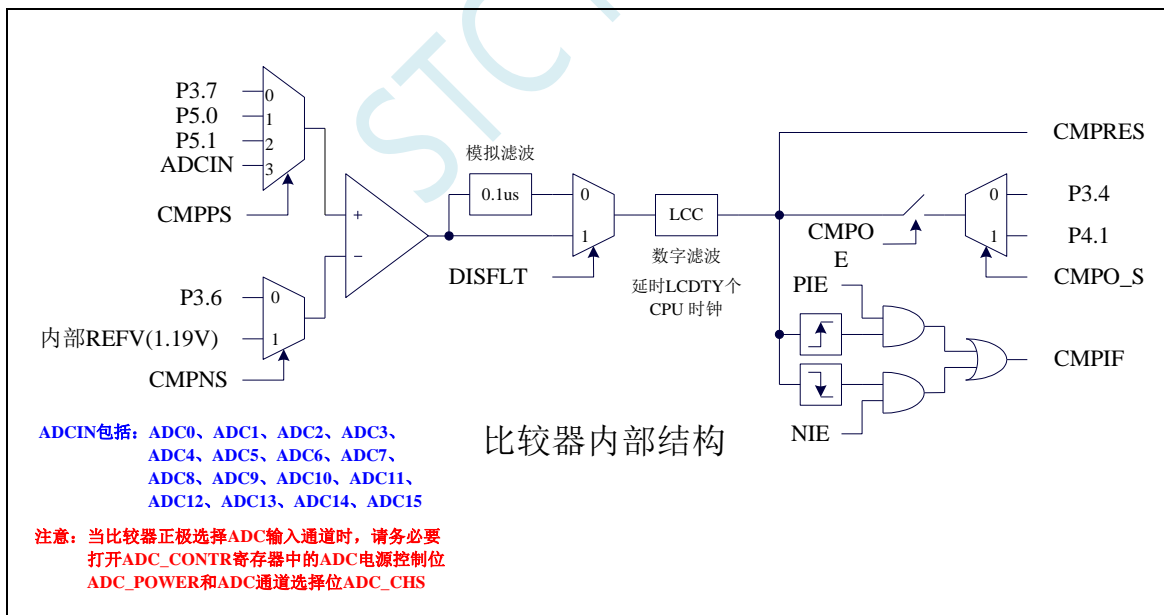
### 15.1 比较器输出功能脚切换

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

CMPO\_S：比较器输出脚选择位

CMPO_S	CMPO
0	P3.4
1	P4.1

### 15.2 比较器内部结构图



## 15.3 比较器相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器 1	E6H	COMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES	0000,xx00
CMPCR2	比较器控制寄存器 2	E7H	INVCMP0	DISFLT	LCDTY[5:0]						0000,0000
CMPEXCFG	比较器扩展配置寄存器	FEAEH	CHYS[1:0]		-	-	-	CM PNS	CM PPS[1:0]		00xx,x000

### 15.3.1 比较器控制寄存器 1 (CMPCR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	COMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES

COMPEN: 比较器模块使能位

0: 关闭比较功能

1: 使能比较功能

CMPIF: 比较器中断标志位。当 PIE 或 NIE 被使能后, 若产生相应的中断信号, 硬件自动将 CMPIF 置 1, 并向 CPU 提出中断请求。此标志位必须用户软件清零。

**(注意: 没有使能比较器中断时, 硬件不会设置此中断标志, 即使用查询方式访问比较器时, 不能查询此中断标志)**

PIE: 比较器上升沿中断使能位。

0: 禁止比较器上升沿中断。

1: 使能比较器上升沿中断。使能比较器的比较结果由 0 变成 1 时产生中断请求。

NIE: 比较器下降沿中断使能位。

0: 禁止比较器下降沿中断。

1: 使能比较器下降沿中断。使能比较器的比较结果由 1 变成 0 时产生中断请求。

CMPOE: 比较器结果输出控制位

0: 禁止比较器结果输出

1: 使能比较器结果输出。比较器结果输出到 P3.4 或者 P4.1 (由 P\_SW2 中的 CMPO\_S 进行设定)

CMPRES: 比较器的比较结果。此位为只读。

0: 表示 CMP+的电平低于 CMP-的电平

1: 表示 CMP+的电平高于 CMP-的电平

CMPRES 是经过数字滤波后的输出信号, 而不是比较器的直接输出结果。

### 15.3.2 比较器控制寄存器 2 (CMPCR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMP0	DISFLT	LCDTY[5:0]					

INVCMP0: 比较器结果输出控制

0: 比较器结果正向输出。若 CMPRES 为 0, 则 P3.4/P4.1 输出低电平, 反之输出高电平。

1: 比较器结果反向输出。若 CMPRES 为 0, 则 P3.4/P4.1 输出高电平, 反之输出低电平。

DISFLT: 模拟滤波功能控制

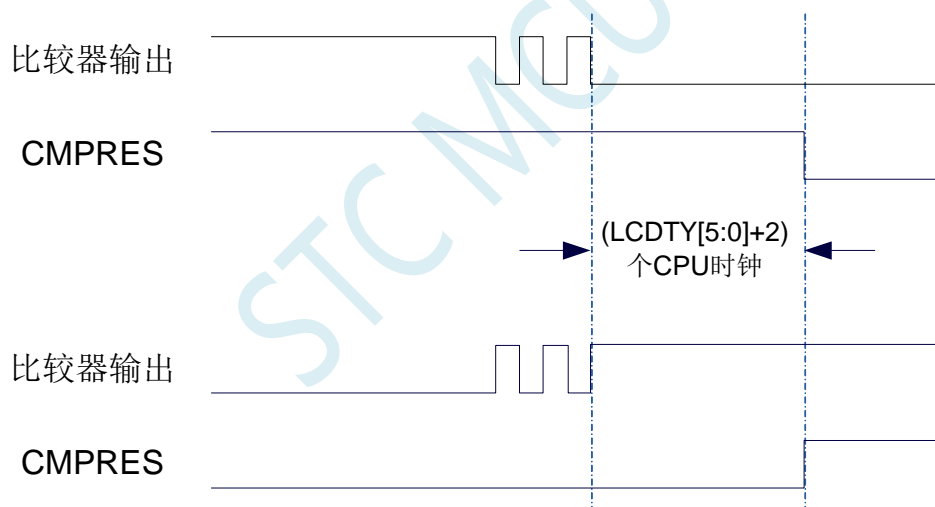
0: 使能 0.1us 模拟滤波功能

1: 关闭 0.1us 模拟滤波功能, 可略微提高比较器的比较速度。

LCDTY[5:0]: 数字滤波功能控制

数字滤波功能即为数字信号去抖动功能。当比较结果发生上升沿或者下降沿变化时, 比较器侦测变化后的信号必须维持 LCDTY 所设置的 CPU 时钟数不发生变化, 才认为数据变化是有效的; 否则将视同信号无变化。

**注意:** 当使能数字滤波功能后, 芯片内部实际的等待时钟需额外增加两个状态机切换时间, 即若 LCDTY 设置为 0 时, 为关闭数字滤波功能; 若 LCDTY 设置为非 0 值  $n$  ( $n=1\sim 63$ ) 时, 则实际的数字滤波时间为  $(n+2)$  个系统时钟



### 15.3.3 比较器扩展配置寄存器 (CMPEXCFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPEXCFG	FEAEH	CHYS[1:0]		-	-	-	CMPNS	CMPPS[1:0]	

CHYS[1:0]: 比较器 DC 迟滞输入选择

CHYS [1:0]	比较器 DC 迟滞输入选择
00	0mV
01	10mV
10	20mV
11	30mV

CMPNS: 比较器负端输入选择

0: P3.6

1: 选择内部 BandGap 经过 OP 后的电压 REFV 作为比较器负极输入源 (芯片在出厂时, 内部参考电压调整为 **1.19V**)

CMPPS[1:0]: 比较器正端输入选择

CMPPS[1:0]	比较器正端
00	P3.7
01	P5.0
10	P5.1
11	ADCIN

## 15.4 范例程序

### 15.4.1 比较器的使用（中断方式）

#### C 语言代码

```
//测试工作频率为 11.0592MHz

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

#define CMPEXCFG      (*(unsigned char volatile xdata *)0xfeae)
sfr      P_SW2      = 0xba;

sfr      CMPCR1     = 0xe6;
sfr      CMPCR2     = 0xe7;

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;
sbit     P11       = P1^1;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;           //清中断标志
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;          //上升沿中断测试端口
    }
    else
    {
        P11 = !P11;          //下降沿中断测试端口
    }
}

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;

P_SW2 = 0x80;
CMPEXCFG = 0x00;
CMPEXCFG &= ~0x03; //P3.7 为 CMP+ 输入脚
// CMPEXCFG |= 0x01; //P5.0 为 CMP+ 输入脚
// CMPEXCFG |= 0x02; //P5.1 为 CMP+ 输入脚
// CMPEXCFG |= 0x03; //ADC 输入脚为 CMP+ 输入脚
CMPEXCFG &= ~0x04; //P3.6 为 CMP- 输入脚
// CMPEXCFG |= 0x04; //内部1.19V 参考信号源为 CMP- 输入脚
P_SW2 = 0x00;

CMPCR2 = 0x00;
CMPCR2 &= ~0x80; //比较器正向输出
// CMPCR2 |= 0x80; //比较器反向输出
CMPCR2 &= ~0x40; //使能 0.1us 滤波
// CMPCR2 |= 0x40; //禁止 0.1us 滤波
// CMPCR2 &= ~0x3f; //比较器结果直接输出
CMPCR2 = 0x10; //比较器结果经过 16 个去抖时钟后输出
CMPCR1 = 0x00;
CMPCR1 |= 0x30; //使能比较器边沿中断
// CMPCR1 &= ~0x20; //禁止比较器上升沿中断
// CMPCR1 |= 0x20; //使能比较器上升沿中断
// CMPCR1 &= ~0x10; //禁止比较器下降沿中断
// CMPCR1 |= 0x10; //使能比较器下降沿中断
// CMPCR1 &= ~0x02; //禁止比较器输出
CMPCR1 = 0x02; //使能比较器输出
CMPCR1 |= 0x80; //使能比较器模块

EA = 1;

while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

CMPEXCFG	EQU	0FEFAH
P_SW2	DATA	0BAH
CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H
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      00ABH
    LJMP    CMPISR

CMPISR:
    ORG      0100H
    PUSH   ACC
    ANL    CMPCR1,#NOT 40H      ;清中断标志
    MOV    A,CMPCR1
    JB     ACC.0,RSING

FALLING:
    CPL    P1.0                ;下降沿中断测试端口
    POP    ACC
    RETI

RSING:
    CPL    P1.1                ;上升沿中断测试端口
    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    P_SW,#80H
    MOV    DPTR,#CMPEXCFG
    CLR    A
    ANL    A,#NOT 03H          ;P3.7 为 CMP+ 输入脚
    ; ORL    A,#01H            ;P5.0 为 CMP+ 输入脚
    ; ORL    A,#02H            ;P5.1 为 CMP+ 输入脚
    ; ORL    A,#03H            ;ADC 输入脚为 CMP+ 输入脚
    ANL    A,#NOT 04H          ;P3.6 为 CMP- 输入脚
    ; ORL    A,#04H            ;内部 1.19V 参考信号源为 CMP- 输入脚
    MOVX   @DPTR,A
    MOV    P_SW,#00H

    MOV    CMPCR2,#00H
    ANL    CMPCR2,#NOT 80H     ;比较器正向输出
    ; ORL    CMPCR2,#80H       ;比较器反向输出
    ; ANL    CMPCR2,#NOT 40H   ;使能 0.1us 滤波
    ; ORL    CMPCR2,#40H       ;禁止 0.1us 滤波
    ; ANL    CMPCR2,#NOT 3FH   ;比较器结果直接输出
    ORL    CMPCR2,#10H        ;比较器结果经过 16 个去抖时钟后输出
    MOV    CMPCR1,#00H
    ORL    CMPCR1,#30H        ;使能比较器边沿中断
    ; ANL    CMPCR1,#NOT 20H   ;禁止比较器上升沿中断
    ; ORL    CMPCR1,#20H       ;使能比较器上升沿中断

```

```

;      ANL      CMPCRI,#NOT 10H      ;禁止比较器下降沿中断
;      ORL      CMPCRI,#10H         ;使能比较器下降沿中断
;      ANL      CMPCRI,#NOT 08H     ;P3.7 为 CMP+ 输入脚
;      ORL      CMPCRI,#08H         ;ADC 输入脚为 CMP+ 输入脚
;      ANL      CMPCRI,#NOT 04H     ;内部 1.19V 参考信号源为 CMP- 输入脚
;      ORL      CMPCRI,#04H         ;P3.6 为 CMP- 输入脚
;      ANL      CMPCRI,#NOT 02H     ;禁止比较器输出
;      ORL      CMPCRI,#02H         ;使能比较器输出
;      ORL      CMPCRI,#80H         ;使能比较器模块
;      SETB     EA

      JMP      $

      END

```

## 15.4.2 比较器的使用（查询方式）

### C 语言代码

```
//测试工作频率为 11.0592MHz;
```

```

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

#define CMPEXCFG (*(unsigned char volatile xdata *)0xfeae)
sfr P_SW2 = 0xba;

sfr CMPCRI = 0xe6;
sfr CMPCR2 = 0xe7;

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;
sbit P11 = 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;

P_SW2 = 0x80;
CMPEXCFG = 0x00;
CMPEXCFG &= ~0x03; //P3.7 为 CMP+ 输入脚
// CMPEXCFG |= 0x01; //P5.0 为 CMP+ 输入脚
// CMPEXCFG |= 0x02; //P5.1 为 CMP+ 输入脚
// CMPEXCFG |= 0x03; //ADC 输入脚为 CMP+ 输入脚
CMPEXCFG &= ~0x04; //P3.6 为 CMP- 输入脚
// CMPEXCFG |= 0x04; //内部1.19V 参考信号源为 CMP- 输入脚
P_SW2 = 0x00;

CMPCR2 = 0x00;
CMPCR2 &= ~0x80; //比较器正向输出
// CMPCR2 |= 0x80; //比较器反向输出
CMPCR2 &= ~0x40; //使能 0.1us 滤波
// CMPCR2 |= 0x40; //禁止 0.1us 滤波
// CMPCR2 &= ~0x3f; //比较器结果直接输出
CMPCR2 = 0x10; //比较器结果经过 16 个去抖时钟后输出
CMPCR1 = 0x00;
CMPCR1 |= 0x30; //使能比较器边沿中断
// CMPCR1 &= ~0x20; //禁止比较器上升沿中断
// CMPCR1 |= 0x20; //使能比较器上升沿中断
// CMPCR1 &= ~0x10; //禁止比较器下降沿中断
// CMPCR1 |= 0x10; //使能比较器下降沿中断
// CMPCR1 &= ~0x02; //禁止比较器输出
CMPCR1 = 0x02; //使能比较器输出
CMPCR1 |= 0x80; //使能比较器模块

while (1)
{
    P10 = CMPCR1 & 0x01; //读取比较器比较结果
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

CMPEXCFG	EQU	0FEFAH
P_SW2	DATA	0BAH
CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H
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        P_SW, #80H
           MOV        DPTR, #CMPEXCFG
           CLR        A
           ANL        A, #NOT 03H           ;P3.7 为 CMP+ 输入脚
           ; ORL        A, #01H           ;P5.0 为 CMP+ 输入脚
           ; ORL        A, #02H           ;P5.1 为 CMP+ 输入脚
           ; ORL        A, #03H           ;ADC 输入脚为 CMP+ 输入脚
           ANL        A, #NOT 04H         ;P3.6 为 CMP- 输入脚
           ; ORL        A, #04H           ;内部 1.19V 参考信号源为 CMP- 输入脚
           MOVX       @DPTR, A
           MOV        P_SW, #00H

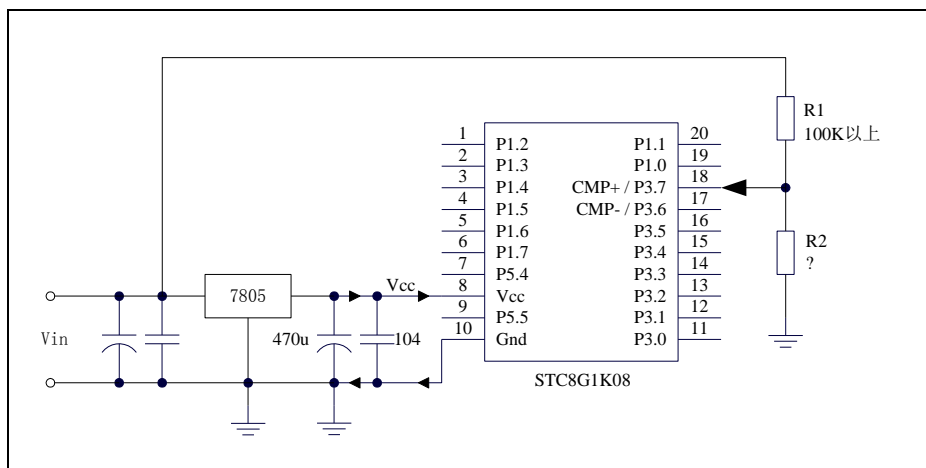
           MOV        CMPCR2, #00H
           ANL        CMPCR2, #NOT 80H    ;比较器正向输出
           ; ORL        CMPCR2, #80H      ;比较器反向输出
           ANL        CMPCR2, #NOT 40H    ;使能 0.1us 滤波
           ; ORL        CMPCR2, #40H      ;禁止 0.1us 滤波
           ; ANL        CMPCR2, #NOT 3FH  ;比较器结果直接输出
           ORL        CMPCR2, #10H        ;比较器结果经过 16 个去抖时钟后输出
           MOV        CMPCR1, #00H
           ORL        CMPCR1, #30H        ;使能比较器边沿中断
           ; ANL        CMPCR1, #NOT 20H  ;禁止比较器上升沿中断
           ; ORL        CMPCR1, #20H      ;使能比较器上升沿中断
           ; ANL        CMPCR1, #NOT 10H  ;禁止比较器下降沿中断
           ; ORL        CMPCR1, #10H      ;使能比较器下降沿中断
           ; ANL        CMPCR1, #NOT 02H  ;禁止比较器输出
           ORL        CMPCR1, #02H        ;使能比较器输出
           ORL        CMPCR1, #80H        ;使能比较器模块

LOOP:
           MOV        A, CMPCR1
           MOV        C, ACC.0
           MOV        P1.0, C           ;读取比较器比较结果
           JMP        LOOP

           END

```

### 15.4.3 比较器作外部掉电检测 (掉电过程中应及时保存用户数据到 EEPROM 中)

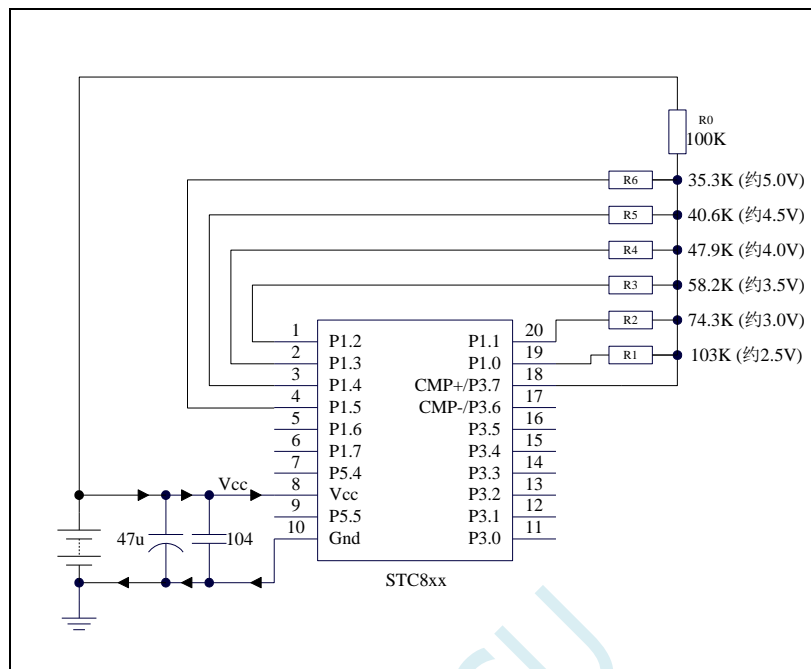


上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+ 的外部输入与内部 1.19V 参考信号源进行比较。

一般当交流电在 220V 时，稳压块 7805 前端的直流电压为 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直流电压为 8.5V。当稳压块 7805 前端的直流电压低于或等于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+ 端输入电压低于内部 1.19V 参考信号源，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直流电压高于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+ 端输入电压高于内部 1.19V 参考信号源，此时 CPU 可继续正常工作。

内部 1.19V 参考信号源即为内部 BandGap 经过 OP 后的电压 REFV (芯片在出厂时，内部参考电压调整为 1.19V)。具体的数值要通过读取内部 1.19V 参考信号源在内部 RAM 区或者 Flash 程序存储器 (ROM) 区所占用的地址的值获得。对于 STC8 系列，内部 1.19V 参考信号源值在 RAM 和 Flash 程序存储器 (ROM) 中的存储地址请参考“[存储器中的特殊参数](#)”章节

## 15.4.4 比较器检测工作电压（电池电压）



上图中，利用电阻分压的原理可以近似的测量出 MCU 的工作电压（选通的通道，MCU 的 I/O 口输出低电平，端口电压值接近 Gnd，未选通的通道，MCU 的 I/O 口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部 1.19V 参考信号源，正端选择通过电阻分压后输入到 CMP+ 管脚的电压值。

初始化时 P1.5~P1.0 口均设置为开漏模式，并输出高。首先 P1.0 口输出低电平，此时若 Vcc 电压低于 2.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 2.5V 则比较器的比较值为 1；

若确定 Vcc 高于 2.5V，则将 P1.0 口输出高，P1.1 口输出低电平，此时若 Vcc 电压低于 3.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 3.0V 则比较器的比较值为 1；

若确定 Vcc 高于 3.0V，则将 P1.1 口输出高，P1.2 口输出低电平，此时若 Vcc 电压低于 3.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 3.5V 则比较器的比较值为 1；

若确定 Vcc 高于 3.5V，则将 P1.2 口输出高，P1.3 口输出低电平，此时若 Vcc 电压低于 4.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 4.0V 则比较器的比较值为 1；

若确定 Vcc 高于 4.0V，则将 P1.3 口输出高，P1.4 口输出低电平，此时若 Vcc 电压低于 4.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 4.5V 则比较器的比较值为 1；

若确定 Vcc 高于 4.5V，则将 P1.4 口输出高，P1.5 口输出低电平，此时若 Vcc 电压低于 5.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 5.0V 则比较器的比较值为 1。

### C 语言代码

```
//测试工作频率为11.0592MHz
```

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

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

```
#define CMPEXCFG (*(unsigned char volatile xdata *)0xfeae)
```

```
sfr P_SW2 = 0xba;
```

```

sfr    CMPCR1    =    0xe6;
sfr    CMPCR2    =    0xe7;

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    P2M0      =    0x96;
sfr    P2M1      =    0x95;

```

```
void delay ()
```

```
{
    char i;

    for (i=0; i<20; i++);
}
```

```
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;

    unsigned char v;

    P1M0 = 0x3f; //P1.5~P1.0 初始化为开漏模式
    P1M1 = 0x3f;
    P1 = 0xff;

    P_SW2 = 0x80;
    CMPEXCFG = 0x00;
    CMPEXCFG &= ~0x03; //P3.7 为 CMP+ 输入脚
    CMPEXCFG |= 0x04; //内部1.19V 参考信号源为 CMP- 输入脚
    P_SW2 = 0x00;

    CMPCR2 = 0x10; //比较器结果经过16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x02; //禁止比较器输出
    CMPCR1 |= 0x80; //使能比较器模块

    while (1)

```



```

{
    v = 0x00; //电压<2.5V
    PI = 0xfe; //P1.0 输出 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x01; //电压>2.5V
    PI = 0xfd; //P1.1 输出 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x03; //电压>3.0V
    PI = 0xfb; //P1.2 输出 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x07; //电压>3.5V
    PI = 0xf7; //P1.3 输出 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x0f; //电压>4.0V
    PI = 0xef; //P1.4 输出 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x1f; //电压>4.5V
    PI = 0xdf; //P1.5 输出 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x3f; //电压>5.0V
ShowVol:
    PI = 0xff;
    P0 = ~v;
}
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

<b>CMPEXCFG</b>	<b>EQU</b>	<b>0FEFAH</b>
<b>P_SW2</b>	<b>DATA</b>	<b>0BAH</b>
<b>CMPCR1</b>	<b>DATA</b>	<b>0E6H</b>
<b>CMPCR2</b>	<b>DATA</b>	<b>0E7H</b>
<b>P2M0</b>	<b>DATA</b>	<b>096H</b>
<b>P2M1</b>	<b>DATA</b>	<b>095H</b>
<b>P0M1</b>	<b>DATA</b>	<b>093H</b>
<b>P0M0</b>	<b>DATA</b>	<b>094H</b>
<b>P1M1</b>	<b>DATA</b>	<b>091H</b>
<b>P1M0</b>	<b>DATA</b>	<b>092H</b>
<b>P2M1</b>	<b>DATA</b>	<b>095H</b>
<b>P2M0</b>	<b>DATA</b>	<b>096H</b>
<b>P3M1</b>	<b>DATA</b>	<b>0B1H</b>
<b>P3M0</b>	<b>DATA</b>	<b>0B2H</b>
<b>P4M1</b>	<b>DATA</b>	<b>0B3H</b>
<b>P4M0</b>	<b>DATA</b>	<b>0B4H</b>
<b>P5M1</b>	<b>DATA</b>	<b>0C9H</b>
<b>P5M0</b>	<b>DATA</b>	<b>0CAH</b>
	<b>ORG</b>	<b>0000H</b>
	<b>LJMP</b>	<b>MAIN</b>

```

ORG          0100H

MAIN:

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

MOV          P_SW, #80H
MOV          DPTR, #CMPEXCFG
CLR          A
ANL         A, #NOT 03H           ;P3.7 为 CMP+ 输入脚
ORL         A, #04H             ;内部 1.19V 参考信号源为 CMP- 输入脚
MOVX        @DPTR, A
MOV          P_SW, #00H

MOV          P1M0, #00111111B       ;P1.5~P1.0 初始化为开漏模式
MOV          P1M1, #00111111B
MOV          PI, #0FFH
MOV          CMPCR2, #10H           ;比较器结果经过 16 个去抖时钟后输出
MOV          CMPCR1, #00H
ANL         CMPCR1, #NOT 02H      ;禁止比较器输出
ORL         CMPCR1, #80H         ;使能比较器模块

LOOP:

MOV          R0, #00000000B         ;电压 < 2.5V
MOV          P1, #11111110B         ;P1.0 输出 0
CALL         DELAY
MOV          A, CMPCR1
JNB         ACC.0, SKIP
MOV          R0, #00000001B         ;电压 > 2.5V
MOV          P1, #11111101B         ;P1.1 输出 0
CALL         DELAY
MOV          A, CMPCR1
JNB         ACC.0, SKIP
MOV          R0, #00000011B         ;电压 > 3.0V
MOV          PI, #11111011B         ;P1.2 输出 0
CALL         DELAY
MOV          A, CMPCR1
JNB         ACC.0, SKIP
MOV          R0, #00000111B         ;电压 > 3.5V
MOV          P1, #11110111B         ;P1.3 输出 0
CALL         DELAY
MOV          A, CMPCR1
JNB         ACC.0, SKIP
MOV          R0, #00001111B         ;电压 > 4.0V
MOV          PI, #11110111B         ;P1.4 输出 0
CALL         DELAY
MOV          A, CMPCR1
JNB         ACC.0, SKIP
MOV          R0, #00011111B         ;电压 > 4.5V

```

---

```
      MOV      P1,#11011111B      ;P1.5 输出0
      CALL     DELAY
      MOV      A,CMPCRI
      JNB     ACC.0,SKIP
      MOV      R0,#00111111B      ;电压>5.0V
SKIP:
      MOV      P1,#11111111B
      MOV      A,R0
      CPL     A
      MOV      P0,A                ;P0.5~P0.0 口显示电压
      JMP     LOOP

DELAY:
      MOV      R0,#20
      DJNZ   R0,$
      RET

      END
```

---

## 16 IAP/EEPROM/DATA-FLASH

STC8A8K64D4 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。

注意：EEPROM 的写操作只能将字节中的 1 写为 0，当需要将字节中的 0 写为 1，则必须执行扇区擦除操作。EEPROM 的读/写操作是以 1 字节为单位进行，而 EEPROM 擦除操作是以 1 扇区（512 字节）为单位进行，在执行擦除操作时，如果目标扇区中有需要保留的数据，则必须预先将这些数据读取到 RAM 中暂存，待擦除完成后再将保存的数据和需要更新的数据一起再写回 EEPROM/DATA-FLASH。

所以在使用时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的（每扇区 512 字节）。

EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

### 16.1 EEPROM 操作时间

- 读取 1 字节：4 个系统时钟（使用 MOVC 指令读取更方便快捷）
- 编程 1 字节：约 30~40us（实际的编程时间为 6~7.5us，但还需要加上状态转换时间和各种控制信号的 SETUP 和 HOLD 时间）
- 擦除 1 扇区（512 字节）：约 4~6ms

EEPROM 操作所需时间是硬件自动控制的，用户只需要正确设置 IAP\_TPS 寄存器即可。

**IAP\_TPS = 系统工作频率 / 1000000（小数部分四舍五入进行取整）**

例如：系统工作频率为 12MHz，则 IAP\_TPS 设置为 12

又例如：系统工作频率为 22.1184MHz，则 IAP\_TPS 设置为 22

再例如：系统工作频率为 5.5296MHz，则 IAP\_TPS 设置为 6

### 16.2 EEPROM 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	
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	-	CMD[1:0]	xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAPTPS[5:0]						xx00,0000

#### 16.2.1 EEPROM 数据寄存器（IAP\_DATA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

在进行 EEPROM 的读操作时，命令执行完成后读出的 EEPROM 数据保存在 IAP\_DATA 寄存器中。

在进行 EEPROM 的写操作时, 在执行写命令前, 必须将待写入的数据存放在 IAP\_DATA 寄存器中, 再发送写命令。擦除 EEPROM 命令与 IAP\_DATA 寄存器无关。

## 16.2.2 EEPROM 地址寄存器 (IAP\_ADDR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

EEPROM 进行读、写、擦除操作的目标地址寄存器。IAP\_ADDRH 保存地址的高字节, IAP\_ADDRL 保存地址的低字节

## 16.2.3 EEPROM 命令寄存器 (IAP\_CMD)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[1:0]	

CMD[1:0]: 发送EEPROM操作命令

00: 空操作

01: 读 EEPROM 命令。读取目标地址所在的 1 字节。

10: 写 EEPROM 命令。写目标地址所在的 1 字节。**注意: 写操作只能将目标字节中的 1 写为 0, 而不能将 0 写为 1。一般当目标字节不为 FFH 时, 必须先擦除。**

11: 擦除 EEPROM。擦除目标地址所在的 1 页 (1 扇区/512 字节)。**注意: 擦除操作会一次擦除 1 个扇区 (512 字节), 整个扇区的内容全部变成 FFH。**

## 16.2.4 EEPROM 触发寄存器 (IAP\_TRIG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后, 需要向触发寄存器 IAP\_TRIG 依次写入 5AH、A5H (顺序不能交换) 两个触发命令来触发相应的读、写、擦除操作。操作完成后, EEPROM 地址寄存器 IAP\_ADDRH、IAP\_ADDRL 和 EEPROM 命令寄存器 IAP\_CMD 的内容不变。如果接下来要对下一个地址的数据进行操作, 需手动更新地址寄存器 IAP\_ADDRH 和寄存器 IAP\_ADDRL 的值。

**注意:** 每次 EEPROM 操作时, 都要对 IAP\_TRIG 先写入 5AH, 再写入 A5H, 相应的命令才会生效。写完触发命令后, CPU 会处于 IDLE 等待状态, 直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

## 16.2.5 EEPROM 控制寄存器 (IAP\_CONTR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

IAPEN: EEPROM操作使能控制位

0: 禁止 EEPROM 操作

1: 使能 EEPROM 操作

SWBS: 软件复位选择控制位, (需要与SWRST配合使用)

0: 软件复位后从用户代码开始执行程序

1: 软件复位后从系统 ISP 监控代码区开始执行程序

SWRST: 软件复位控制位

0: 无动作

1: 产生软件复位

CMD\_FAIL: EEPROM操作失败状态位, 需要软件清零

0: EEPROM 操作正确

1: EEPROM 操作失败

## 16.2.6 EEPROM 等待时间控制寄存器 (IAP\_TPS)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

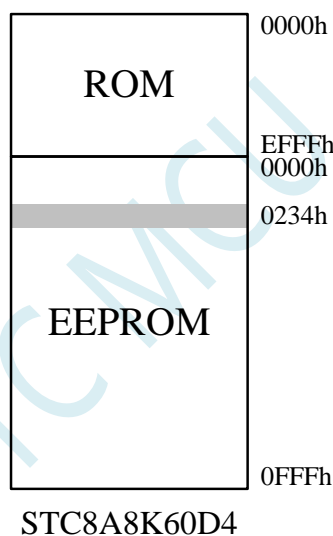
需要根据工作频率进行设置

若工作频率为12MHz, 则需要将IAP\_TPS设置为12; 若工作频率为24MHz, 则需要将IAP\_TPS设置为24, 其他频率以此类推。

## 16.3 EEPROM 大小及地址

STC8A8K64D4 系列单片机内部均有用于保存用户数据的 EEPROM。内部的 EEPROM 有 3 操作方式：读、写和擦除，其中擦除操作是以扇区为单位进行操作，每扇区为 512 字节，即每执行一次擦除命令就会擦除一个扇区，而读数据和写数据都是以字节为单位进行操作的，即每执行一次读或者写命令时只能读出或者写入一个字节。

STC8A8K64D4 系列单片机内部的 EEPROM 的访问方式有两种：IAP 方式和 MOVC 方式。IAP 方式可对 EEPROM 执行读、写、擦除操作，但 MOVC 只能对 EEPROM 进行读操作，而不能进行写和擦除操作。无论是使用 IAP 方式还是使用 MOVC 方式访问 EEPROM，首先都需要设置正确的目标地址。IAP 方式时，目标地址与 EEPROM 实际的物理地址是一致的，均是从地址 0000H 开始访问，但若使用 MOVC 指令进行读取 EEPROM 数据时，目标地址必须是在 EEPROM 实际的物理地址的基础上还有加上程序大小的偏移。下面以 STC8A8K60D4 这个型号为例，对目标地址进行详细说明：



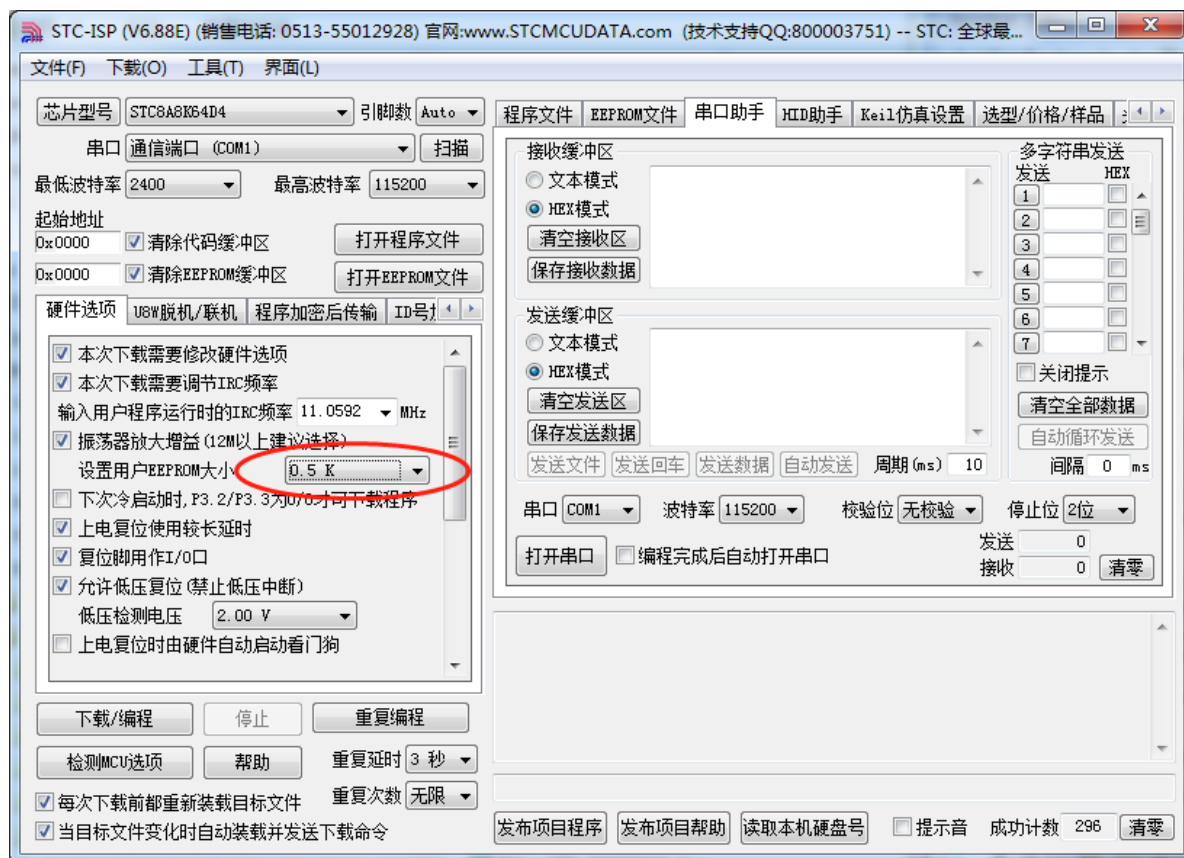
STC8A8K60D4 的程序空间为 60K 字节 (0000h~EFFFh)，EEPROM 空间为 4K (0000h~0FFFh)。当需要对 EEPROM 物理地址 0234h 的单元进行读、写、擦除时，若使用 IAP 方式进行访问时，设置的目标地址为 0234h，即 IAP\_ADDRH 设置 02h，IAP\_ADDRL 设置 34h，然后设置相应的触发命令即可对 0234h 单元进行正确操作了。但若是使用 MOVC 方式读取 EEPROM 的 0234h 单元，则必须在 0234h 的基础上还有加上 Flash 程序存储器（ROM）空间的大小 F000h，即必须将 DPTR 设置为 F234h，然后才能使用 MOVC 指令进行读取。

注意：由于擦除是以 512 字节为单位进行操作的，所以执行擦除操作时所设置的目标地址的低 9 位是无意义的。例如：执行擦除命令时，设置地址 0234H/0200H/0300H/03FFH，最终执行擦除的动作都是相同的，都是擦除 0200H~03FFH 这 512 字节。

不同型号内部 EEPROM 的大小及访问地址会存在差异, 针对各个型号 EEPROM 的详细大小和地址请参考下表

型号	大小	扇区	IAP 方式读/写/擦除		MOVC 读取	
			起始地址	结束地址	起始地址	结束地址
STC8A8K16D4	48K	96	0000h	BFFFh	4000h	FFFFh
STC8A8K32D4	32K	64	0000h	7FFFh	8000h	FFFFh
STC8A8K60D4	4K	8	0000h	0FFFh	F000h	FFFFh
STC8A8K64D4	用户自定义 <sup>[1]</sup>					

[1]: 这个为特殊型号, 这个型号的 EEPROM 大小是可用在 ISP 下载时用户自己设置的。如下图所示:



用户可用根据自己的需要在整个 FLASH 空间中规划出任意不超过 FLASH 大小的 EEPROM 空间, 但需要注意: **EEPROM 总是从后向前进行规划的。**

例如: STC8A8K64D4 这个型号的 FLASH 为 64K, 此时若用户想分出其中的 4K 作为 EEPROM 使用, 则 EEPROM 的物理地址即为 64K 的最后 4K, 物理地址为 F000h~FFFFh, 当然, 用户若使用 IAP 的方式进行访问, 目标地址仍然从 0000h 开始, 到 0FFFh 结束, 当使用 MOVC 读取则需要从 F000h 开始, 到 FFFFh 结束。



## 16.4 范例程序

### 16.4.1 EEPROM 基本操作

#### 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    IAP_DATA = 0xC2;
sfr    IAP_ADDRH = 0xC3;
sfr    IAP_ADDRL = 0xC4;
sfr    IAP_CMD = 0xC5;
sfr    IAP_TRIG = 0xC6;
sfr    IAP_CONTR = 0xC7;
sfr    IAP_TPS = 0xF5;
```

```
void IapIdle()
```

```
{
    IAP_CONTR = 0;           //关闭 IAP 功能
    IAP_CMD = 0;           //清除命令寄存器
    IAP_TRIG = 0;         //清除触发寄存器
    IAP_ADDRH = 0x80;     //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}
```

```
char IapRead(int addr)
```

```
{
    char dat;

    IAP_CONTR = 0x80;     //使能 IAP
    IAP_TPS = 12;        //设置等待参数 12MHz
    IAP_CMD = 1;         //设置 IAP 读命令
    IAP_ADDRL = addr;    //设置 IAP 低地址
    IAP_ADDRH = addr >> 8; //设置 IAP 高地址
    IAP_TRIG = 0x5a;     //写触发命令(0x5a)
    IAP_TRIG = 0xa5;     //写触发命令(0xa5)
    _nop_();
    dat = IAP_DATA;     //读 IAP 数据
    IapIdle();          //关闭 IAP 功能
}
```

```

    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;           //使能 IAP
    IAP_TPS = 12;              //设置等待参数 12MHz
    IAP_CMD = 2;               //设置 IAP 写命令
    IAP_ADDRL = addr;          //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;     //设置 IAP 高地址
    IAP_DATA = dat;            //写 IAP 数据
    IAP_TRIG = 0x5a;           //写触发命令(0x5a)
    IAP_TRIG = 0xa5;           //写触发命令(0xa5)
    _nop_();
    IapIdle();                 //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;           //使能 IAP
    IAP_TPS = 12;              //设置等待参数 12MHz
    IAP_CMD = 3;               //设置 IAP 擦除命令
    IAP_ADDRL = addr;          //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;     //设置 IAP 高地址
    IAP_TRIG = 0x5a;           //写触发命令(0x5a)
    IAP_TRIG = 0xa5;           //写触发命令(0xa5)
    _nop_();
    IapIdle();                 //关闭 IAP 功能
}

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;

    IapErase(0x0400);
    P0 = IapRead(0x0400);       //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);       //P1=0x12

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

```

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H

```

```

IAP_ADDRL DATA 0C4H
IAP_CMD DATA 0C5H
IAP_TRIG DATA 0C6H
IAP_CONTR DATA 0C7H
IAP_TPS DATA 0F5H

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

IAP_IDLE:
MOV IAP_CONTR,#0 ;关闭 IAP 功能
MOV IAP_CMD,#0 ;清除命令寄存器
MOV IAP_TRIG,#0 ;清除触发寄存器
MOV IAP_ADDRH,#80H ;将地址设置到非 IAP 区域
MOV IAP_ADDRL,#0
RET

IAP_READ:
MOV IAP_CONTR,#80H ;使能 IAP
MOV IAP_TPS,#12 ;设置等待参数 12MHz
MOV IAP_CMD,#1 ;设置 IAP 读命令
MOV IAP_ADDRL,DPL ;设置 IAP 低地址
MOV IAP_ADDRH,DPH ;设置 IAP 高地址
MOV IAP_TRIG,#5AH ;写触发命令(0x5a)
MOV IAP_TRIG,#0A5H ;写触发命令(0xa5)
NOP
MOV A,IAP_DATA ;读取 IAP 数据
LCALL IAP_IDLE ;关闭 IAP 功能
RET

IAP_PROGRAM:
MOV IAP_CONTR,#80H ;使能 IAP
MOV IAP_TPS,#12 ;设置等待参数 12MHz
MOV IAP_CMD,#2 ;设置 IAP 写命令
MOV IAP_ADDRL,DPL ;设置 IAP 低地址
MOV IAP_ADDRH,DPH ;设置 IAP 高地址
MOV IAP_DATA,A ;写 IAP 数据
MOV IAP_TRIG,#5AH ;写触发命令(0x5a)
MOV IAP_TRIG,#0A5H ;写触发命令(0xa5)
NOP
LCALL IAP_IDLE ;关闭 IAP 功能
RET

IAP_ERASE:

```

```

MOV      IAP_CONTR,#80H      ;使能 IAP
MOV      IAP_TPS,#12        ;设置等待参数 12MHz
MOV      IAP_CMD,#3         ;设置 IAP 擦除命令
MOV      IAP_ADDRL,DPL      ;设置 IAP 低地址
MOV      IAP_ADDRH,DPH      ;设置 IAP 高地址
MOV      IAP_TRIG,#5AH      ;写触发命令(0x5a)
MOV      IAP_TRIG,#0A5H     ;写触发命令(0xa5)
NOP
LCALL    IAP_IDLE           ;关闭 IAP 功能
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      DPTR,#0400H
LCALL    IAP_ERASE
MOV      DPTR,#0400H
LCALL    IAP_READ
MOV      P0,A                ;P0=0FFH
MOV      DPTR,#0400H
MOV      A,#12H
LCALL    IAP_PROGRAM
MOV      DPTR,#0400H
LCALL    IAP_READ
MOV      P1,A                ;P1=12H

SJMP     $

END

```

## 16.4.2 使用 MOVC 读取 EEPROM

### 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    IAP_DATA   = 0xc2;
sfr    IAP_ADDRH  = 0xc3;
sfr    IAP_ADDRL  = 0xc4;
sfr    IAP_CMD    = 0xc5;
sfr    IAP_TRIG   = 0xc6;
sfr    IAP_CONTR  = 0xc7;
sfr    IAP_TPS    = 0xf5;

#define IAP_OFFSET 0xF000H //STC8A8K60S4

void IapIdle()
{
    IAP_CONTR = 0; //关闭 IAP 功能
    IAP_CMD = 0; //清除命令寄存器
    IAP_TRIG = 0; //清除触发寄存器
    IAP_ADDRH = 0x80; //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    addr += IAP_OFFSET; //使用 MOVC 读取 EEPROM 需要加上相应的偏移
    return *(char code*)(addr); //使用 MOVC 读取数据
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80; //使能 IAP
    IAP_TPS = 12; //设置等待参数 12MHz
    IAP_CMD = 2; //设置 IAP 写命令
    IAP_ADDRL = addr; //设置 IAP 低地址
    IAP_ADDRH = addr >> 8; //设置 IAP 高地址
    IAP_DATA = dat; //写 IAP 数据
    IAP_TRIG = 0x5a; //写触发命令(0x5a)
    IAP_TRIG = 0xa5; //写触发命令(0xa5)
    _nop_();
    IapIdle(); //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80; //使能 IAP
    IAP_TPS = 12; //设置等待参数 12MHz
    IAP_CMD = 3; //设置 IAP 擦除命令
    IAP_ADDRL = addr; //设置 IAP 低地址
    IAP_ADDRH = addr >> 8; //设置 IAP 高地址
    IAP_TRIG = 0x5a; //写触发命令(0x5a)
    IAP_TRIG = 0xa5; //写触发命令(0xa5)
    _nop_();
    IapIdle(); //关闭 IAP 功能
}

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;

    IapErase(0x0400);
    P0 = IapRead(0x0400);           //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);         //P1=0x12

    while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

```

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

IAP_OFFSET  EQU      0F000H           ;STC8A8K60S4

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

IAP_IDLE:
            MOV     IAP_CONTR,#0       ;关闭 IAP 功能
            MOV     IAP_CMD,#0        ;清除命令寄存器
            MOV     IAP_TRIG,#0       ;清除触发寄存器
            MOV     IAP_ADDRH,#80H    ;将地址设置到非 IAP 区域

```

```

MOV      IAP_ADDRL,#0
RET

IAP_READ:
MOV      A,#LOW IAP_OFFSET      ;使用 MOVC 读取 EEPROM 需要加上相应的偏移
ADD      A,DPL
MOV      DPL,A
MOV      A,@HIGH IAP_OFFSET
ADDC    A,DPH
MOV      DPH,A
CLR      A
MOVC    A,@A+DPTR              ;使用 MOVC 读取数据
RET

IAP_PROGRAM:
MOV      IAP_CONTR,#80H        ;使能 IAP
MOV      IAP_TPS,#12          ;设置等待参数 12MHz
MOV      IAP_CMD,#2           ;设置 IAP 写命令
MOV      IAP_ADDRL,DPL        ;设置 IAP 低地址
MOV      IAP_ADDRH,DPH        ;设置 IAP 高地址
MOV      IAP_DATA,A           ;写 IAP 数据
MOV      IAP_TRIG,#5AH        ;写触发命令(0x5a)
MOV      IAP_TRIG,#0A5H       ;写触发命令(0xa5)
NOP
LCALL   IAP_IDLE              ;关闭 IAP 功能
RET

IAP_ERASE:
MOV      IAP_CONTR,#80H        ;使能 IAP
MOV      IAP_TPS,#12          ;设置等待参数 12MHz
MOV      IAP_CMD,#3           ;设置 IAP 擦除命令
MOV      IAP_ADDRL,DPL        ;设置 IAP 低地址
MOV      IAP_ADDRH,DPH        ;设置 IAP 高地址
MOV      IAP_TRIG,#5AH        ;写触发命令(0x5a)
MOV      IAP_TRIG,#0A5H       ;写触发命令(0xa5)
NOP
LCALL   IAP_IDLE              ;关闭 IAP 功能
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      DPTR,#0400H
LCALL   IAP_ERASE
MOV      DPTR,#0400H
LCALL   IAP_READ
MOV      P0,A                  ;P0=0FFH

```

```

MOV     DPTR,#0400H
MOV     A,#12H
LCALL  IAP_PROGRAM
MOV     DPTR,#0400H
LCALL  IAP_READ
MOV     P1,A           ;P1=12H

SJMP   $

END

```

### 16.4.3 使用串口送出 EEPROM 数据

#### C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

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

```

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 AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;

```

```

sfr IAP_DATA = 0xC2;
sfr IAP_ADDRH = 0xC3;
sfr IAP_ADDRL = 0xC4;
sfr IAP_CMD = 0xC5;
sfr IAP_TRIG = 0xC6;
sfr IAP_CONTR = 0xC7;
sfr IAP_TPS = 0xF5;

```

```

void UartInit()
{
    SCON = 0x5a;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
}

```

```
void UartSend(char dat)
```



```

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

void IapIdle()
{
    IAP_CONTR = 0;           //关闭 IAP 功能
    IAP_CMD = 0;           //清除命令寄存器
    IAP_TRIG = 0;         //清除触发寄存器
    IAP_ADDRH = 0x80;     //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = 0x80;     //使能 IAP
    IAP_TPS = 12;        //设置等待参数 12MHz
    IAP_CMD = 1;         //设置 IAP 读命令
    IAP_ADDRL = addr;    //设置 IAP 低地址
    IAP_ADDRH = addr >> 8; //设置 IAP 高地址
    IAP_TRIG = 0x5a;     //写触发命令(0x5a)
    IAP_TRIG = 0xa5;     //写触发命令(0xa5)
    _nop_();
    dat = IAP_DATA;     //读 IAP 数据
    IapIdle();          //关闭 IAP 功能

    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;     //使能 IAP
    IAP_TPS = 12;        //设置等待参数 12MHz
    IAP_CMD = 2;         //设置 IAP 写命令
    IAP_ADDRL = addr;    //设置 IAP 低地址
    IAP_ADDRH = addr >> 8; //设置 IAP 高地址
    IAP_DATA = dat;     //写 IAP 数据
    IAP_TRIG = 0x5a;     //写触发命令(0x5a)
    IAP_TRIG = 0xa5;     //写触发命令(0xa5)
    _nop_();
    IapIdle();          //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;     //使能 IAP
    IAP_TPS = 12;        //设置等待参数 12MHz
    IAP_CMD = 3;         //设置 IAP 擦除命令
    IAP_ADDRL = addr;    //设置 IAP 低地址
    IAP_ADDRH = addr >> 8; //设置 IAP 高地址
    IAP_TRIG = 0x5a;     //写触发命令(0x5a)
    IAP_TRIG = 0xa5;     //写触发命令(0xa5)
    _nop_();
    IapIdle();          //关闭 IAP 功能
}

```

```

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;

    UartInit();
    IapErase(0x0400);
    UartSend(IapRead(0x0400));
    IapProgram(0x0400, 0x12);
    UartSend(IapRead(0x0400));

    while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

AUXR	DATA	8EH
T2H	DATA	0D6H
T2L	DATA	0D7H
IAP_DATA	DATA	0C2H
IAP_ADDRH	DATA	0C3H
IAP_ADDRL	DATA	0C4H
IAP_CMD	DATA	0C5H
IAP_TRIG	DATA	0C6H
IAP_CONTR	DATA	0C7H
IAP_TPS	DATA	0F5H
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

**UART\_INIT:**

```

MOV     SCON,#5AH
MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
MOV     T2H,#0FFH
MOV     AUXR,#15H
RET

```

**UART\_SEND:**

```

JNB     TI,$
CLR     TI
MOV     SBUF,A
RET

```

**IAP\_IDLE:**

```

MOV     IAP_CONTR,#0       ;关闭 IAP 功能
MOV     IAP_CMD,#0        ;清除命令寄存器
MOV     IAP_TRIG,#0       ;清除触发寄存器
MOV     IAP_ADDRH,#80H    ;将地址设置到非 IAP 区域
MOV     IAP_ADDRL,#0
RET

```

**IAP\_READ:**

```

MOV     IAP_CONTR,#80H    ;使能 IAP
MOV     IAP_TPS,#12      ;设置等待参数 12MHz
MOV     IAP_CMD,#1       ;设置 IAP 读命令
MOV     IAP_ADDRL,DPL    ;设置 IAP 低地址
MOV     IAP_ADDRH,DPH    ;设置 IAP 高地址
MOV     IAP_TRIG,#5AH    ;写触发命令(0x5a)
MOV     IAP_TRIG,#0A5H   ;写触发命令(0xa5)
NOP
MOV     A,IAP_DATA       ;读取 IAP 数据
LCALL  IAP_IDLE         ;关闭 IAP 功能
RET

```

**IAP\_PROGRAM:**

```

MOV     IAP_CONTR,#80H    ;使能 IAP
MOV     IAP_TPS,#12      ;设置等待参数 12MHz
MOV     IAP_CMD,#2       ;设置 IAP 写命令
MOV     IAP_ADDRL,DPL    ;设置 IAP 低地址
MOV     IAP_ADDRH,DPH    ;设置 IAP 高地址
MOV     IAP_DATA,A       ;写 IAP 数据
MOV     IAP_TRIG,#5AH    ;写触发命令(0x5a)
MOV     IAP_TRIG,#0A5H   ;写触发命令(0xa5)
NOP
LCALL  IAP_IDLE         ;关闭 IAP 功能
RET

```

**IAP\_ERASE:**

```

MOV     IAP_CONTR,#80H    ;使能 IAP
MOV     IAP_TPS,#12      ;设置等待参数 12MHz
MOV     IAP_CMD,#3       ;设置 IAP 擦除命令
MOV     IAP_ADDRL,DPL    ;设置 IAP 低地址
MOV     IAP_ADDRH,DPH    ;设置 IAP 高地址
MOV     IAP_TRIG,#5AH    ;写触发命令(0x5a)
MOV     IAP_TRIG,#0A5H   ;写触发命令(0xa5)
NOP
LCALL  IAP_IDLE         ;关闭 IAP 功能
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

LCALL   UART_INIT
MOV     DPTR, #0400H
LCALL   IAP_ERASE
MOV     DPTR, #0400H
LCALL   IAP_READ
LCALL   UART_SEND
MOV     DPTR, #0400H
MOV     A, #12H
LCALL   IAP_PROGRAM
MOV     DPTR, #0400H
LCALL   IAP_READ
LCALL   UART_SEND

SJMP    $

END

```

## 16.4.4 串口 1 读写 EEPROM-带 MOVC 读

C 语言代码 (main.c)

//测试工作频率为 11.0592MHz;

/\* 本程序经过测试完全正常, 不提供电话技术支持, 如不能理解, 请自行补充相关基础. \*/

/\*\*\*\*\*\* 本程序功能说明 \*\*\*\*\*

STC8G 系列 EEPROM 通用测试程序

请先别修改程序, 直接下载"02-串口 1 读写 EEPROM-带 MOVC 读"里面的"UART-EEPROM.hex"测试, 下载时选择主频 11.0592MHZ.

PC 串口设置: 波特率 115200, 8, n, 1.

对 EEPROM 做扇区擦除、写入 64 字节、读出 64 字节的操作。

命令例子:

E 0 对 EEPROM 进行扇区擦除操作, E 表示擦除, 数字 0 为 0 扇区(十进制, 0~126, 看具体 IC).

W 0 对 EEPROM 进行写入操作, W 表示写入, 数字 0 为 0 扇区(十进制, 0~126, 看具体 IC). 从扇区的开始地址连续写 64 字节.

R 0 对 EEPROM 进行 IAP 读出操作, R 表示读出, 数字 0 为 0 扇区(十进制, 0~126, 看具体 IC). 从扇区的开始地址连续读 64 字节.

M 0 对 EEPROM 进行 MOVC 读出操作(操作地址为扇区\*512+偏移地址) 数字 0 为 0 扇区(十进制, 0~126, 看具体 IC). 从扇区的开始地址连续读 64 字节.

注意：为了通用，程序不识别扇区是否有效，用户自己根据具体的型号来决定。

日期: 2019-6-10

\*\*\*\*\*/

```
#include "config.H"
#include "EEPROM.h"

#define Baudrate1          115200L
#define UART1_BUF_LENGTH  10
#define EEADDR_OFFSET     (8 * 1024) //定义EEPROM用MOVC访问时加的偏移量,
//等于FLASH ROM的大小对于IAP或IRC开头的,
//则偏移量必须为0

#define TimeOutSet1       5

/***** 本地常量声明 *****/
u8 code T_Strings[]={"去年今日此门中，人面桃花相映红。人面不知何处去，桃花依旧笑春风。"};

/***** 本地变量声明 *****/
u8 xdatatmp[70];
u8 xdataRX1_Buffer[UART1_BUF_LENGTH];
u8 RX1_Cnt;
u8 RX1_TimeOut;
bit B_TX1_Busy;

/***** 本地函数声明 *****/
void UART1_config(void);
void TX1_write2buff(u8 dat); //写入发送缓冲
void PrintString1(u8 *puts); //发送一个字符串

/***** 外部函数和变量声明 *****/

/*****/

u8 CheckData(u8 dat)
{
    if((dat >= '0') && (dat <= '9')) return (dat-'0');
    if((dat >= 'A') && (dat <= 'F')) return (dat-'A'+10);
    if((dat >= 'a') && (dat <= 'f')) return (dat-'a'+10);
    return 0xff;
}

u16 GetAddress(void)
{
    u16 address;
    u8 i;

    address = 0;
    if(RX1_Cnt < 3) return 65535; //error
    if(RX1_Cnt <= 5) //5个字节以内是扇区操作，十进制
    //支持命令: E 0, E 12, E 120
    // W 0, W 12, W 120
    // R 0, R 12, R 120

    {
        for(i=2; i<RX1_Cnt; i++)
        {
            if(CheckData(RX1_Buffer[i]) > 9)
                return 65535; //error
            address = address * 10 + CheckData(RX1_Buffer[i]);
        }
    }
}

```

```

    }
    if(address < 124) //限制在0~123 扇区
    {
        address <<= 9;
        return (address);
    }
}
else if(RX1_Cnt == 8) //8 个字节直接地址操作，十六进制
//支持命令: E 0x1234, W 0x12b3, R 0x0A00
{
    if((RX1_Buffer[2] == '0') && ((RX1_Buffer[3] == 'x') || (RX1_Buffer[3] == 'X')))
    {
        for(i=4; i<8; i++)
        {
            if(CheckData(RX1_Buffer[i]) > 0x0F)
                return 65535; //error
            address = (address << 4) + CheckData(RX1_Buffer[i]);
        }
        if(address < 63488)
            return (address); //限制在0~123 扇区
    }
}

return 65535; //error
}

//=====
// 函数: void delay_ms(u8 ms)
// 描述: 延时函数。
// 参数: ms,要延时的ms 数 这里只支持1~255ms. 自动适应主时钟
// 返回: none.
// 版本: VER1.0
// 日期: 2013-4-1
// 备注:
//=====
void delay_ms(u8 ms)
{
    u16 i;
    do
    {
        i = MAIN_Fosc / 10000;
        while(--i) ;
    }while(--ms);
}

//使用MOVC 读EEPROM
void EEPROM_MOVC_read_n(u16 EE_address, u8 *DataAddress, u16 number)
{
    u8 code *pc;

    pc = EE_address + EEADDR_OFFSET;
    do
    {
        *DataAddress = *pc; //读出的数据
        DataAddress++;
        pc++;
    }while(--number);
}

```

```

/***** 主函数 *****/
void main(void)
{
    u8 i;
    u16 addr;

    UART1_config();           // 选择波特率 2: 使用Timer2 做波特率
                              //其它值: 使用Timer1 做波特率
    EA = 1;                   //允许总中断

    PrintStringI("STC8 系列MCU 用串口1 测试EEPROM 程序!\r\n"); //UART1 发送一个字符串

    while(1)
    {
        delay_ms(1);
        if(RXI_TimeOut > 0)           //超时计数
        {
            if(--RXI_TimeOut == 0)
            {
                if(RXI_Buffer[1] == ' ')
                {
                    addr = GetAddress();
                    if(addr < 63488)           //限制在0~123 扇区
                    {
                        if(RXI_Buffer[0] == 'E') //PC 请求擦除一个扇区
                        {
                            EEPROM_SectorErase(addr);
                            PrintStringI("扇区擦除完成!\r\n");
                        }
                        else if(RXI_Buffer[0] == 'W') //PC 请求写入EEPROM 64 字节数据
                        {
                            EEPROM_write_n(addr,T_Strings,64);
                            PrintStringI("写入操作完成!\r\n");
                        }
                        else if(RXI_Buffer[0] == 'R') //PC 请求返回64 字节EEPROM 数据
                        {
                            PrintStringI("IAP 读出的数据如下:\r\n");
                            EEPROM_read_n(addr,tmp,64);
                            for(i=0; i<64; i++)
                                TX1_write2buff(tmp[i]); //将数据返回给串口
                            TX1_write2buff(0x0d);
                            TX1_write2buff(0x0a);
                        }
                        else if(RXI_Buffer[0] == 'M') //PC 请求返回64 字节EEPROM 数据
                        {
                            PrintStringI("MOVC 读出的数据如下:\r\n");
                            EEPROM_MOVC_read_n(addr,tmp,64);
                            for(i=0; i<64; i++)
                                TX1_write2buff(tmp[i]); //将数据返回给串口
                            TX1_write2buff(0x0d);
                            TX1_write2buff(0x0a);
                        }
                        else PrintStringI("命令错误!\r\n");
                    }
                }
            }
        }
    }
}

```

```

        RXI_Cnt = 0;
    }
}
}
}
}
/*****/

/*****/ 发送一个字节 *****/
void TX1_write2buff(u8 dat)           //写入发送缓冲
{
    B_TX1_Busy = 1;                   //标志发送忙
    SBUF = dat;                       //发送一个字节
    while(B_TX1_Busy);               //等待发送完毕
}

//=====
// 函数: void PrintString1(u8 *puts)
// 描述: 串口1 发送字符串函数。
// 参数: puts: 字符串指针。
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void PrintString1(u8 *puts)           //发送一个字符串
{
    for (; *puts != 0; puts++)       //遇到停止符0 结束
    {
        TX1_write2buff(*puts);
    }
}

//=====
// 函数: void UART1_config(void)
// 描述: UART1 初始化函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void UART1_config(void)
{
    TRI = 0;
    AUXR &= ~0x01;                   //S1 BRT Use Timer1;
    AUXR |= (1<<6);                  //Timer1 set as 1T mode
    TMOD &= ~(1<<6);                  //Timer1 set As Timer
    TMOD &= ~0x30;                    //Timer1_16bitAutoReload;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0;                          // 禁止Timer1 中断
    INT_CLKO &= ~0x02;                // Timer1 不输出高速时钟
    TRI = 1;                          // 运行Timer1

    S1_USE_P30P31(); P3n_standard(0x03); //切换到 P3.0 P3.1
    //S1_USE_P36P37(); P3n_standard(0xc0); //切换到 P3.6 P3.7
    //S1_USE_P16P17(); P1n_standard(0xc0); //切换到 P1.6 P1.7

    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

B_TX1_Busy = 0;
RX1_Cnt = 0;
}

//=====
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUF_LENGTH)
            RX1_Cnt = 0; //防溢出
        RX1_Buffer[RX1_Cnt++] = SBUF;
        RX1_TimeOut = TimeOutSet1;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

```

## C 语言代码 (EEPROM.c)

```

//测试工作频率为 11.0592MHz

// 本程序是 STC 系列的内置 EEPROM 读写程序。

#include "config.h"
#include "eeprom.h"

//=====
// 函数: void ISP_Disable(void)
// 描述: 禁止访问 ISP/IAP.
// 参数: non.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====
void DisableEEPROM(void)
{
    ISP_CONTR = 0; //禁止 ISP/IAP 操作
    IAP_TPS = 0;
    ISP_CMD = 0; //去除 ISP/IAP 命令
}

```

```

    ISP_TRIG = 0; //防止 ISP/IAP 命令误触发
    ISP_ADDRH = 0xff; //清0 地址高字节
    ISP_ADDRL = 0xff; //清0 地址低字节, 指向非EEPROM 区, 防止误操作
}

//=====================================================
// 函数: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 从指定EEPROM 首地址读出n 个字节放指定的缓冲
// 参数: EE_address: 读出EEPROM 的首地址
//       DataAddress: 读出数据放缓冲的首地址
//       number: 读出的字节长度
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====================================================
void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0; //禁止中断
    ISP_CONTR = ISP_EN; //允许ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //工作频率设置
    ISP_READ(); //送字节读命令, 命令不需改变时, 不需重新送命令
    do
    {
        ISP_ADDRH = EE_address / 256; //送地址高字节 (地址需要改变时才需重新送地址)
        ISP_ADDRL = EE_address % 256; //送地址低字节
        ISP_TRIG(); //先送5AH, 再送A5H 到ISP/IAP 触发寄存器,
        //每次都需要如此
        //送完A5H 后, ISP/IAP 命令立即被触发启动
        //CPU 等待IAP 完成后, 才会继续执行程序。

        _nop();
        _nop();
        _nop();
        *DataAddress = ISP_DATA; //读出的数据送往
        EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = 1; //重新允许中断
}

/***** 扇区擦除函数 *****/
//=====================================================
// 函数: void EEPROM_SectorErase(u16 EE_address)
// 描述: 把指定地址的EEPROM 扇区擦除
// 参数: EE_address: 要擦除的扇区EEPROM 的地址
// 返回: non.
// 版本: V1.0, 2013-5-10
//=====================================================
void EEPROM_SectorErase(u16 EE_address)
{
    EA = 0; //禁止中断
    //只有扇区擦除, 没有字节擦除, 512 字节/扇区。
    //扇区中任意一个字节地址都是扇区地址。
    ISP_ADDRH = EE_address / 256; //送扇区地址高字节 (地址需要改变时才需重新送地址)
    ISP_ADDRL = EE_address % 256; //送扇区地址低字节
    ISP_CONTR = ISP_EN; //允许ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //工作频率设置
    ISP_ERASE(); //送扇区擦除命令, 命令不需改变时, 不需重新送命令
    ISP_TRIG();
}

```

```

    _nop_();
    _nop_();
    _nop_();
    DisableEEPROM();
    EA = I; //重新允许中断
}

//=====
// 函数: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 把缓冲的n 个字节写入指定首地址的EEPROM.
// 参数: EE_address: 写入EEPROM 的首地址.
//       DataAddress: 写入源数据的缓冲的首地址.
//       number:     写入的字节长度.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====
void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0; //禁止中断

    ISP_CONTR = ISP_EN; //允许ISP/IAP 操作
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //工作频率设置
    ISP_WRITE(); //送字节写命令, 命令不需改变时, 不需重新送命令
    do
    {
        ISP_ADDRH = EE_address / 256; //送地址高字节 (地址需要改变时才需重新送地址)
        ISP_ADDRL = EE_address % 256; //送地址低字节
        ISP_DATA = *DataAddress; //送数据到ISP_DATA, 只有数据改变时才需重新送
        ISP_TRIG();
        _nop_();
        _nop_();
        _nop_();
        EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = I; //重新允许中断
}

```

## 16.4.5 口令擦除写入-多扇区备份-串口 1 操作

### C 语言代码 (main.c)

```

//测试工作频率为 11.0592MHz

/* 本程序经过测试完全正常, 不提供电话技术支持, 如不能理解, 请自行补充相关基础 */

/***** 本程序功能说明 *****/
STC8G 系列 STC8H 系列 STC8C 系列 EEPROM 通用测试程序, 演示多扇区备份、有扇区错误则用正确扇区数据写入、
全部扇区错误(比如第一次运行程序)则写入默认值.

每次写都写入 3 个扇区, 即冗余备份.
每个扇区写一条记录, 写入完成后读出保存的数据和校验值跟源数据和校验值比较, 并从串口 I(P3.0 P3.1)返回结果(正确
或错误提示).
每条记录自校验, 64 字节数据, 2 字节校验值, 校验值 = 64 字节数据累加和 ^ 0x5555. ^0x5555 是为了保证写入的 66 个数
据不全部为 0.
如果有扇区错误, 则将正确扇区的数据写入错误扇区, 如果 3 个扇区都错误, 则均写入默认值.

```

擦除、写入、读出操作前均需要设置口令, 如果口令不对则退出操作, 并且每次退出操作都会清除口令。

请先别修改程序, 直接下载"03-口令擦除写入-多扇区备份-串口1 操作"里面的"UART-EEPROM.hex"测试, 下载时选择主频11.0592MHZ。

PC 串口设置: 波特率115200,8,n,1.

对EEPROM 做扇区擦除、写入 64 字节、读出 64 字节的操作。

命令例子:

使用串口助手发单个字符, 大小写均可。

发 E 或 e: 对 EEPROM 进行扇区擦除操作, E 表示擦除, 会擦除扇区 0、1、2。

发 W 或 w: 对 EEPROM 进行写入操作, W 表示写入, 会写入扇区 0、1、2, 每个扇区连续写 64 字节, 扇区 0 写入 0x0000~0x003f, 扇区 1 写入 0x0200~0x023f, 扇区 2 写入 0x0400~0x043f。

发 R 或 r: 对 EEPROM 进行读出操作, R 表示读出, 会读出扇区 0、1、2, 每个扇区连续读 64 字节, 扇区 0 读出 0x0000~0x003f, 扇区 1 读出 0x0200~0x023f, 扇区 2 读出 0x0400~0x043f。

注意: 为了通用, 程序不识别扇区是否有效, 用户自己根据具体的型号来决定。

日期: 2021-11-5

\*\*\*\*\*/

```
#include "config.H"
```

```
#include "EEPROM.h"
```

```
#define Baudrate1 115200L
```

```
***** 本地常量声明 *****/
```

```
u8 code T_StringD[]={"去年今日此门中,人面桃花相映红。人面不知何处去,桃花依旧笑春风。"};
```

```
u8 code T_StringW[]={"横看成岭侧成峰,远近高低各不同。不识庐山真面目,只缘身在此山中。"};
```

```
***** 本地变量声明 *****/
```

```
u8 xdatatmp[70]; //通用数据
```

```
u8 xdataSaveTmp[70]; //要写入的数组
```

```
bit B_TX1_Busy;
```

```
u8 cmd; //串口单字符命令
```

```
***** 本地函数声明 *****/
```

```
void UART1_config(void);
```

```
void TX1_write2buff(u8 dat); //写入发送缓冲
```

```
void PrintString1(u8 *puts); //发送一个字符串
```

```
***** 外部函数和变量声明 *****/
```

```
***** 读取EEPROM 记录, 并且校验, 返回校验结果, 0 为正确, 1 为错误 *****/
```

```
u8 ReadRecord(u16 addr)
```

```
{
```

```
u8 i;
```

```
u16 ChckSum; //计算的累加和
```

```
u16 j; //读取的累加和
```

```
for(i=0; i<66; i++) tmp[i] = 0; //清除缓冲
```

```
PassWord = D_PASSWORD; //给定口令
```

```

    EEPROM_read_n(addr,tmp,66);           //读出扇区0
    for(ChckSum=0, i=0; i<64; i++)
        ChckSum += tmp[i];               //计算累加和
    j = ((u16)tmp[64]<<8) + (u16)tmp[65]; //读取记录的累加和
    j ^= 0x5555;                          //隔位取反, 避免全0
    if(ChckSum !=j)    return 1;          //累加和错误, 返回1
    return 0;                              //累加和正确, 返回0
}

/***** 写入EEPROM 记录, 并且校验, 返回校验结果 0 为正确, 1 为错误 *****/
u8 SaveRecord(u16 addr)
{
    u8 i;
    u16 ChckSum;                          //计算的累加和

    for(ChckSum=0, i=0; i<64; i++)
        ChckSum += SaveTmp[i];           //计算累加和
    ChckSum ^= 0x5555;                   //隔位取反, 避免全0
    SaveTmp[64] = (u8)(ChckSum >> 8);
    SaveTmp[65] = (u8)ChckSum;

    PassWord = D_PASSWORD;               //给定口令
    EEPROM_SectorErase(addr);             //擦除一个扇区
    PassWord = D_PASSWORD;               //给定口令
    EEPROM_write_n(addr, SaveTmp, 66);    //写入扇区

    for(i=0; i<66; i++)
        tmp[i] = 0;                      //清除缓冲
    PassWord = D_PASSWORD;               //给定口令
    EEPROM_read_n(addr,tmp,66);          //读出扇区0
    for(i=0; i<66; i++)
        {
            if(SaveTmp[i] != tmp[i])
                return 1;                 //数据有错误, 返回1
        }
    return 0;                             //累加和正确, 返回0
}

/***** 主函数 *****/
void main(void)
{
    u8 i;
    u8 status;                            //状态

    UART1_config();                       //选择波特率 2: 使用Timer2 做波特率
                                           //其它值: 使用Timer1 做波特率
    EA = 1;                               //允许总中断

    PrintString1("STC8G-8H-8C 系列 MCU 用串口1 测试EEPROM 程序!\r\n"); //UART1 发送一个字符串

                                           //上电读取3个扇区并校验, 如果有扇区错误则将正确的
                                           //扇区写入错误扇区, 如果3个扇区都错误, 则写入默认值

    status = 0;
    if(ReadRecord(0x0000) == 0)           //读扇区0
    {
        status |= 0x01;                   //正确则标记 status.0=1
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i];          //保存在写缓冲
    }
}

```

```

if(ReadRecord(0x0200) == 0) //读扇区1
{
    status |= 0x02; //正确则标记 status.1=1
    for(i=0; i<64; i++)
        SaveTmp[i] = tmp[i]; //保存在写缓冲
}
if(ReadRecord(0x0400) == 0) //读扇区2
{
    status |= 0x04; //正确则标记 status.2=1
    for(i=0; i<64; i++)
        SaveTmp[i] = tmp[i]; //保存在写缓冲
}

if(status == 0) //所有扇区都错误 则写入默认值
{
    for(i=0; i<64; i++)
        SaveTmp[i] = T_StringD[i]; //读取默认值
}
else PrintString1("上电读取3个扇区数据均正确!\r\n"); //UART1 发送一个字符串提示

if((status & 0x01) == 0) //扇区0 错误 则写入默认值
{
    if(SaveRecord(0x0000) == 0)
        PrintString1("写入扇区0 正确!\r\n"); //写入记录0 扇区正确
    else
        PrintString1("写入扇区0 错误!\r\n"); //写入记录0 扇区错误
}
if((status & 0x02) == 0) //扇区1 错误 则写入默认值
{
    if(SaveRecord(0x0200) == 0)
        PrintString1("写入扇区1 正确!\r\n"); //写入记录1 扇区正确
    else
        PrintString1("写入扇区1 错误!\r\n"); //写入记录1 扇区错误
}
if((status & 0x04) == 0) //扇区2 错误 则写入默认值
{
    if(SaveRecord(0x0400) == 0)
        PrintString1("写入扇区2 正确!\r\n"); //写入记录2 扇区正确
    else
        PrintString1("写入扇区2 错误!\r\n"); //写入记录2 扇区错误
}

while(1)
{
    if(cmd != 0) //有串口命令
    {
        if((cmd >= 'a') && (cmd <= 'z'))
            cmd -= 0x20; //小写转大写

        if(cmd == 'E') //PC 请求擦除一个扇区
        {
            PassWord = D_PASSWORD; //给定口令
            EEPROM_SectorErase(0x0000); //擦除一个扇区
            PassWord = D_PASSWORD; //给定口令
            EEPROM_SectorErase(0x0200); //擦除一个扇区
            PassWord = D_PASSWORD; //给定口令
            EEPROM_SectorErase(0x0400); //擦除一个扇区
            PrintString1("扇区擦除完成!\r\n");
        }
    }
}

```

```

}

else if(cmd == 'W') //PC 请求写入EEPROM 64 字节数据
{
    for(i=0; i<64; i++)
        SaveTmp[i] = T_StringW[i]; //写入数值
    if(SaveRecord(0x0000) == 0)
        PrintString1("写入扇区0 正确!\r\n"); //写入记录0 扇区正确
    else
        PrintString1("写入扇区0 错误!\r\n"); //写入记录0 扇区错误
    if(SaveRecord(0x0200) == 0)
        PrintString1("写入扇区1 正确!\r\n"); //写入记录1 扇区正确
    else
        PrintString1("写入扇区1 错误!\r\n"); //写入记录1 扇区错误
    if(SaveRecord(0x0400) == 0)
        PrintString1("写入扇区2 正确!\r\n"); //写入记录2 扇区正确
    else
        PrintString1("写入扇区2 错误!\r\n"); //写入记录2 扇区错误
}

else if(cmd == 'R') //PC 请求返回64 字节EEPROM 数据
{
    if(ReadRecord(0x0000) == 0) //读出扇区0 的数据
    {
        PrintString1("读出扇区0 的数据如下:\r\n");
        for(i=0; i<64; i++)
            TX1_write2buff(tmp[i]); //将数据返回给串口
        TX1_write2buff(0x0d); //回车换行
        TX1_write2buff(0x0a);
    }
    else PrintString1("读出扇区0 的数据错误!\r\n");

    if(ReadRecord(0x0200) == 0) //读出扇区1 的数据
    {
        PrintString1("读出扇区1 的数据如下:\r\n");
        for(i=0; i<64; i++)
            TX1_write2buff(tmp[i]); //将数据返回给串口
        TX1_write2buff(0x0d); //回车换行
        TX1_write2buff(0x0a);
    }
    else PrintString1("读出扇区1 的数据错误!\r\n");

    if(ReadRecord(0x0400) == 0) //读出扇区2 的数据
    {
        PrintString1("读出扇区2 的数据如下:\r\n");
        for(i=0; i<64; i++)
            TX1_write2buff(tmp[i]); //将数据返回给串口
        TX1_write2buff(0x0d); //回车换行
        TX1_write2buff(0x0a);
    }
    else PrintString1("读出扇区2 的数据错误!\r\n");
}
else PrintString1("命令错误!\r\n");
cmd = 0;
}
}

}

}
/*****/

```

```

/***** 发送一个字节 *****/
void TX1_write2buff(u8 dat)           //写入发送缓冲
{
    B_TX1_Busy = 1;                 //标志发送忙
    SBUF = dat;                     //发送一个字节
    while(B_TX1_Busy);              //等待发送完毕
}

//=====
// 函数: void PrintString1(u8 *puts)
// 描述: 串口1 发送字符串函数。
// 参数: puts: 字符串指针。
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void PrintString1(u8 *puts)           //发送一个字符串
{
    for (; *puts != 0; puts++)       //遇到停止符0 结束
    {
        TX1_write2buff(*puts);
    }
}

//=====
// 函数: void UART1_config(void)
// 描述: UART1 初始化函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01;                  //SI BRT Use Timer1;
    AUXR |= (1<<6);                  //Timer1 set as IT mode
    TMOD &= ~(1<<6);                 //Timer1 set As Timer
    TMOD &= ~0x30;                   //Timer1_16bitAutoReload;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0;                          // 禁止Timer1 中断
    INT_CLKO &= ~0x02;                // Timer1 不输出高速时钟
    TR1 = 1;                            // 运行Timer1

    SI_USE_P30P31(); P3n_standard(0x03); //切换到 P3.0 P3.1
    //SI_USE_P36P37(); P3n_standard(0xc0); //切换到 P3.6 P3.7
    //SI_USE_P16P17(); P1n_standard(0xc0); //切换到 P1.6 P1.7

    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

```



```

    B_TXI_Busy = 0;
}

//=====
// 函数: void UART1_int (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 日期: 2014-11-28
// 备注:
//=====
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        cmd = SBUF;
    }

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

```

## C 语言代码 (EEPROM.c)

//测试工作频率为 11.0592MHz

// 本程序是 STC 系列的内置 EEPROM 读写程序。

```

#include "config.h"
#include "EEPROM.h"

u32    PassWord;                //擦除 写入时需要的口令

//=====
// 函数: void  ISP_Disable(void)
// 描述: 禁止访问ISP/IAP.
// 参数: non.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====
void DisableEEPROM(void)
{
    ISP_CONTR = 0;                //禁止 ISP/IAP 操作
    IAP_TPS   = 0;
    ISP_CMD   = 0;                //去除 ISP/IAP 命令
    ISP_TRIG  = 0;                //防止 ISP/IAP 命令误触发
    ISP_ADDRH = 0xff;            //清0 地址高字节
    ISP_ADDRL = 0xff;            //清0 地址低字节, 指向非EEPROM 区, 防止误操作
}

//=====
// 函数: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 从指定EEPROM 首地址读出n 个字节放指定的缓冲.
// 参数: EE_address: 读出EEPROM 的首地址.

```

```

//      DataAddress: 读出数据放缓冲的首地址.
//      number:      读出的字节长度.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====
void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    if(PassWord == D_PASSWORD)                //口令正确才会操作EEPROM
    {
        EA = 0;                               //禁止中断
        ISP_CONTR = ISP_EN;                   //允许ISP/IAP 操作
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //工作频率设置
        ISP_READ();                           //送字节读命令, 命令不需改变时, 不需重新送命令
        do
        {
            ISP_ADDRH = EE_address / 256;     //送地址高字节 (地址需要改变时才需重新送地址)
            ISP_ADDRL = EE_address % 256;     //送地址低字节
            if(PassWord == D_PASSWORD)        //口令正确才触发操作
            {
                ISP_TRIG = 0x5A;              //先送5AH, 再送A5H 到ISP/IAP 触发寄存器,
                //每次都需要如此
                ISP_TRIG = 0xA5;              //送完A5H 后, ISP/IAP 命令立即被触发启动
                //CPU 等待 IAP 完成后, 才会继续执行程序。
            }
            _nop_();
            _nop_();
            _nop_();
            *DataAddress = ISP_DATA;          //读出的数据送往
            EE_address++;
            DataAddress++;
        }while(--number);

        DisableEEPROM();
        EA = 1;                               //重新允许中断
    }
    PassWord = 0;                             //清除口令
}

/***** 扇区擦除函数 *****/
//=====
// 函数: void EEPROM_SectorErase(u16 EE_address)
// 描述: 把指定地址的EEPROM 扇区擦除
// 参数: EE_address: 要擦除的扇区EEPROM 的地址.
// 返回: non.
// 版本: V1.0, 2013-5-10
//=====
void EEPROM_SectorErase(u16 EE_address)
{
    if(PassWord == D_PASSWORD)                //口令正确才会操作EEPROM
    {
        EA = 0;                               //禁止中断
        //只有扇区擦除, 没有字节擦除, 512 字节/扇区。
        //扇区中任意一个字节地址都是扇区地址。
        ISP_ADDRH = EE_address / 256;         //送扇区地址高字节 (地址需要改变时才需重新送地址)
        ISP_ADDRL = EE_address % 256;         //送扇区地址低字节
        ISP_CONTR = ISP_EN;                   //允许ISP/IAP 操作
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //工作频率设置
        ISP_ERASE();                           //送扇区擦除命令, 命令不需改变时, 不需重新送命令
        if(PassWord == D_PASSWORD)            //口令正确才触发操作
        {

```

```

        ISP_TRIG = 0x5A;           //先送 5AH，再送 A5H 到 ISP/IAP 触发寄存器，
                                   //每次都需要如此
        ISP_TRIG = 0xA5;           //送完 A5H 后，ISP/IAP 命令立即被触发启动
                                   //CPU 等待 IAP 完成后，才会继续执行程序。
    }
    _nop_();
    _nop_();
    _nop_();
    DisableEEPROM();
    EA = 1;                         //重新允许中断
}
PassWord = 0;                       //清除口令
}

//=====================================================
// 函数: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// 描述: 把缓冲的 n 个字节写入指定首地址的 EEPROM.
// 参数: EE_address: 写入 EEPROM 的首地址.
//       DataAddress: 写入源数据的缓冲的首地址.
//       number:     写入的字节长度.
// 返回: non.
// 版本: V1.0, 2012-10-22
//=====================================================
void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    if(PassWord == D_PASSWORD)       //口令正确才会操作 EEPROM
    {
        EA = 0;                       //禁止中断

        ISP_CONTR = ISP_EN;           //允许 ISP/IAP 操作
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //工作频率设置
        ISP_WRITE();                 //送字节写命令，命令不需改变时，不需重新送命令
        do
        {
            ISP_ADDRH = EE_address / 256; //送地址高字节（地址需要改变时才需重新送地址）
            ISP_ADDRL = EE_address % 256; //送地址低字节
            ISP_DATA = *DataAddress;      //送数据到 ISP_DATA，只有数据改变时才需重新送
            if(PassWord == D_PASSWORD)   //口令正确才触发操作
            {
                ISP_TRIG = 0x5A;           //先送 5AH，再送 A5H 到 ISP/IAP 触发寄存器，
                                           //每次都需要如此
                ISP_TRIG = 0xA5;           //送完 A5H 后，ISP/IAP 命令立即被触发启动
                                           //CPU 等待 IAP 完成后，才会继续执行程序。
            }
            _nop_();
            _nop_();
            _nop_();
            EE_address++;
            DataAddress++;
        }while(--number);

        DisableEEPROM();
        EA = 1;                         //重新允许中断
    }
    PassWord = 0;                       //清除口令
}

```

## 17 ADC 模数转换, 内部 1.19V 参考信号源

STC8A8K64D4 系列单片机内部集成了一个 12 位高速 A/D 转换器。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频(ADC 的工作时钟频率范围为 SYSclk/2/1 到 SYSclk/2/16)。

STC8A8K64D4 系列的 ADC 最快速度: **800K (每秒进行 80 万次 ADC 转换)**

ADC 转换结果的数据格式有两种: 左对齐和右对齐。可方便用户程序进行读取和引用。

**注意: ADC 的第 15 通道只能用于检测内部参考信号源, 参考信号源值出厂时校准为 1.19V, 由于制造误差以及测量误差, 导致实际的内部参考信号源相比 1.19V, 大约有 ±1% 的误差。如果用户需要知道每一颗芯片的准确内部参考信号源值, 可外接精准参考信号源, 然后利用 ADC 的第 15 通道进行测量标定。**

**如果芯片有 ADC 的外部参考电源管脚 ADC\_VRef+, 则一定不能浮空, 必须接外部参考电源或者直接连接到 VCC**

### 17.1 ADC 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	
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]			000x,0000
ADC_RES	ADC 转换结果高位寄存器	BDH								0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH								0000,0000
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]			xx0x,0000

符号	描述	地址	位地址与符号							复位值
			B7	B6	B5	B4	B3	B2	B1	
ADCTIM	ADC 时序控制寄存器	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				0010,1010
ADCEXCFG	ADC 扩展配置寄存器	FEADH	-	-	ADCETRS[1:0]	-	CVTIMESEL[2:0]			xx00,x000

#### 17.1.1 ADC 控制寄存器 (ADC\_CONTR), PWM 触发 ADC 控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]			

ADC\_POWER: ADC 电源控制位

- 0: 关闭 ADC 电源
- 1: 打开 ADC 电源。

建议进入空闲模式和掉电模式前将 ADC 电源关闭, 以降低功耗

**特别注意:**

- 1、给 MCU 的内部 ADC 模块电源打开后, 需等待约 1ms, 等 MCU 内部的 ADC 电源稳定后再让 ADC 工作;
- 2、适当加长对外部信号的采样时间, 就是对 ADC 内部采样保持电容的充电或放电时间, 时间够, 内部才能和外部电势相等。

ADC\_START: ADC 转换启动控制位。写入 1 后开始 ADC 转换, 转换完成后硬件自动将此位清零。

0: 无影响。即使 ADC 已经开始转换工作, 写 0 也不会停止 A/D 转换。

1: 开始 ADC 转换, 转换完成后硬件自动将此位清零。

ADC\_FLAG: ADC 转换结束标志位。当 ADC 完成一次转换后, 硬件会自动将此位置 1, 并向 CPU 提出中断请求。此标志位必须软件清零。

**ADC\_EPWMT: 使能 PWM 实时触发 ADC 功能。详情请参考 PWM 章节**

ADC\_CHS[3:0]: ADC 模拟通道选择位

(注意: 被选择为 ADC 输入通道的 I/O 口, 必须设置 PxM0/PxM1 寄存器将 I/O 口模式设置为高阻输入模式。另外如果 MCU 进入掉电模式/时钟停振模式后, 仍需要使能 ADC 通道, 则需要设置 PxIE 寄存器关闭数字输入通道, 以防止外部模拟输入信号忽高忽低而产生额外的功耗)

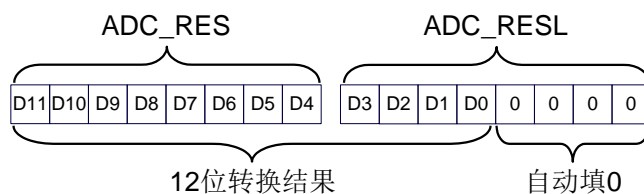
ADC_CHS[3:0]	ADC 通道
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P1.2/ADC2
0011	P1.3/ADC3
0100	P1.4/ADC4
0101	P1.5/ADC5
0110	P1.6/ADC6
0111	P1.7/ADC7
1000	P0.0/ADC8
1001	P0.1/ADC9
1010	P0.2/ADC10
1011	P0.3/ADC11
1100	P0.4/ADC12
1101	P0.5/ADC13
1110	P0.6/ADC14
1111	测试内部 1.19V

### 17.1.2 ADC 配置寄存器 (ADCCFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

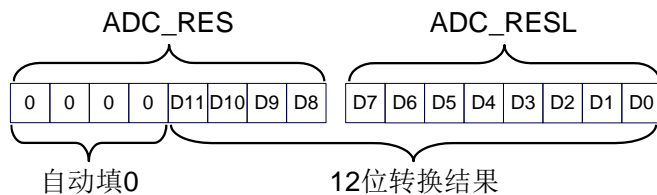
RESFMT: ADC 转换结果格式控制位

0: 转换结果左对齐。ADC\_RES 保存结果的高 8 位, ADC\_RESL 保存结果的低 4 位。格式如下:



RESFMT=0

1: 转换结果右对齐。ADC\_RES 保存结果的高 4 位, ADC\_RESL 保存结果的低 8 位。格式如下:



SPEED[3:0]: 设置 ADC 工作时钟频率 { $F_{ADC} = \text{SYSclk}/2/(\text{SPEED}+1)$ }

SPEED[3:0]	给 ADC 的工作时钟频率
0000	SYSclk/2/1
0001	SYSclk/2/2
0010	SYSclk/2/3
...	...
1101	SYSclk/2/14
1110	SYSclk/2/15
1111	SYSclk/2/16

### 17.1.3 ADC 转换结果寄存器 (ADC\_RES, ADC\_RESL)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

当 A/D 转换完成后, 10 位/12 位的转换结果会自动保存到 ADC\_RES 和 ADC\_RESL 中。保存结果的数据格式请参考 ADC\_CFG 寄存器中的 RESFMT 设置。

## 17.1.4 ADC 时序控制寄存器 (ADCTIM)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCTIM	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				

CSSETUP: ADC 通道选择时间控制  $T_{setup}$

CSSETUP	占用 ADC 工作时钟数
0	1 (默认值)
1	2

CSHOLD[1:0]: ADC 通道选择保持时间控制  $T_{hold}$

CSHOLD[1:0]	占用 ADC 工作时钟数
00	1
01	2 (默认值)
10	3
11	4

SMPDUTY[4:0]: ADC 模拟信号采样时间控制  $T_{duty}$  (注意: SMPDUTY 一定不能设置小于 01010B)

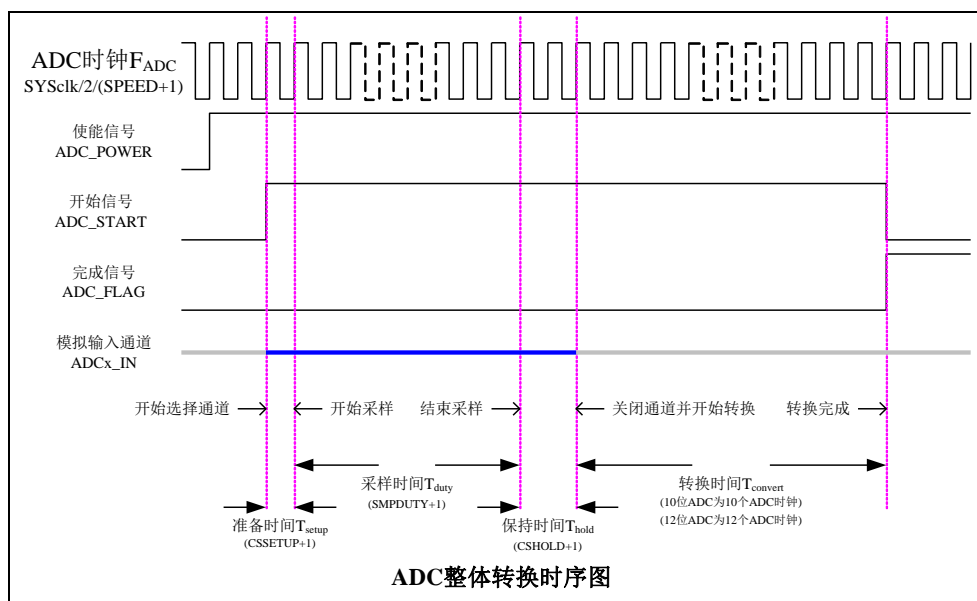
SMPDUTY[4:0]	占用 ADC 工作时钟数
00000	1
00001	2
...	...
01010	11 (默认值)
...	...
11110	31
11111	32

ADC 数模转换时间:  $T_{convert}$

10 位 ADC 的转换时间固定为 10 个 ADC 工作时钟

12 位 ADC 的转换时间固定为 12 个 ADC 工作时钟

一个完整的 ADC 转换时间为:  $T_{setup} + T_{duty} + T_{hold} + T_{convert}$ , 如下图所示



## 17.1.5 ADC 扩展配置寄存器 (ADCEXCFG)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCEXCFG	FEADH	-	-	ADCETRS[1:0]		-	CVTIMESEL[2:0]		

ADCETRS[1:0]: ADC 外部触发脚 ADC\_ETR 控制位

ADCETRS[1:0]	ADC_ETR 设置
0x	禁止 ETR 功能
10	使能 ADC_ETR 的上升沿触发 ADC
11	使能 ADC_ETR 的下降沿触发 ADC

**注: 使用此功能前, 必须打开 ADC\_CONTR 中的 ADC 电源开关, 并设置好相应的 ADC 通道**

CVTIMESEL[2:0]: ADC 自动转换次数选择

CVTIMESEL [2:0]	ADC 自动转换次数
0xx	转换 1 次
100	转换 2 次并取平均值
101	转换 4 次并取平均值
110	转换 8 次并取平均值
111	转换 16 次并取平均值

**注: 当使能 ADC 自动转换多次功能后, ADC 中断标志只会在 ADC 自动转换到设置的次数后, 才会被置 1 (例如: 设置 CVTIMESEL 为 101B, 即 ADC 自动转换 4 次并取平均值, 则 ADC 中断标志位每完成 4 次 ADC 转换才会被置 1)**



## 17.2 ADC 相关计算公式

### 17.2.1 ADC 速度计算公式

ADC 的转换速度由 ADCCFG 寄存器中的 SPEED 和 ADCTIM 寄存器共同控制。转换速度的计算公式如下所示:

$$12\text{位ADC转换速度} = \frac{\text{MCU工作频率SYSclk}}{2 \times (\text{SPEED}[3:0] + 1) \times (\text{CSSETUP} + 1) + (\text{CSHOLD} + 1) + (\text{SMPDUTY} + 1) + 12}$$

#### 注意:

- 12 位 ADC 的速度不能高于 800KHz
- SMPDUTY 的值不能小于 10, 建议设置为 15
- CSSETUP 可使用上电默认值 0
- CHOLD 可使用上电默认值 1 (ADCTIM 建议设置为 3FH)

### 17.2.2 ADC 转换结果计算公式

$$12\text{位ADC转换结果} = 4096 \times \frac{\text{ADC被转换通道的输入电压}V_{in}}{\text{ADC外部参考源的电压}} \quad (\text{有独立ADC\_Vref+管脚})$$

### 17.2.3 反推 ADC 输入电压计算公式

$$\text{ADC被转换通道的输入电压}V_{in} = \text{ADC外部参考源的电压} \times \frac{12\text{位ADC转换结果}}{4096} \quad (\text{有独立ADC\_Vref+管脚})$$

## 17.2.4 反推工作电压计算公式

当需要使用 ADC 输入电压和 ADC 转换结果反推工作电压时，若目标芯片无独立的 ADC\_Vref+管脚，则可直接测量并使用下面公式，若目标芯片有独立 ADC\_Vref+管脚时，则必须将 ADC\_Vref+管脚连接到 Vcc 管脚。

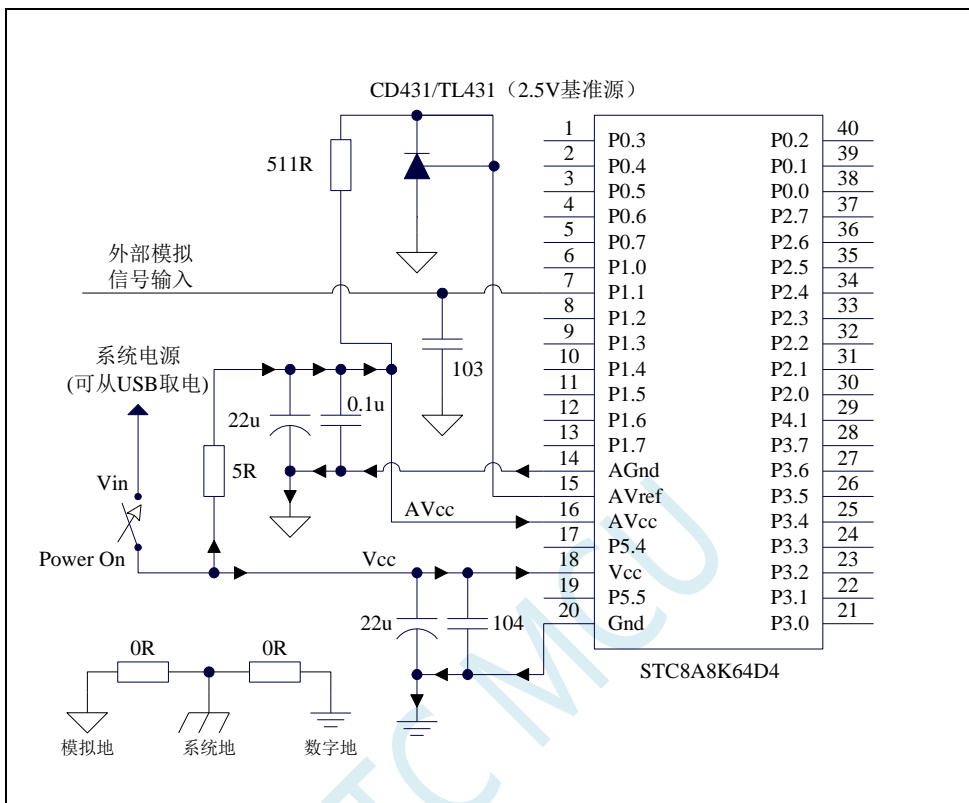
$$\text{MCU工作电压Vcc} = 4096 \times \frac{\text{ADC被转换通道的输入电压Vin}}{\text{12位ADC转换结果}}$$

## 17.3 12 位 ADC 静态特性

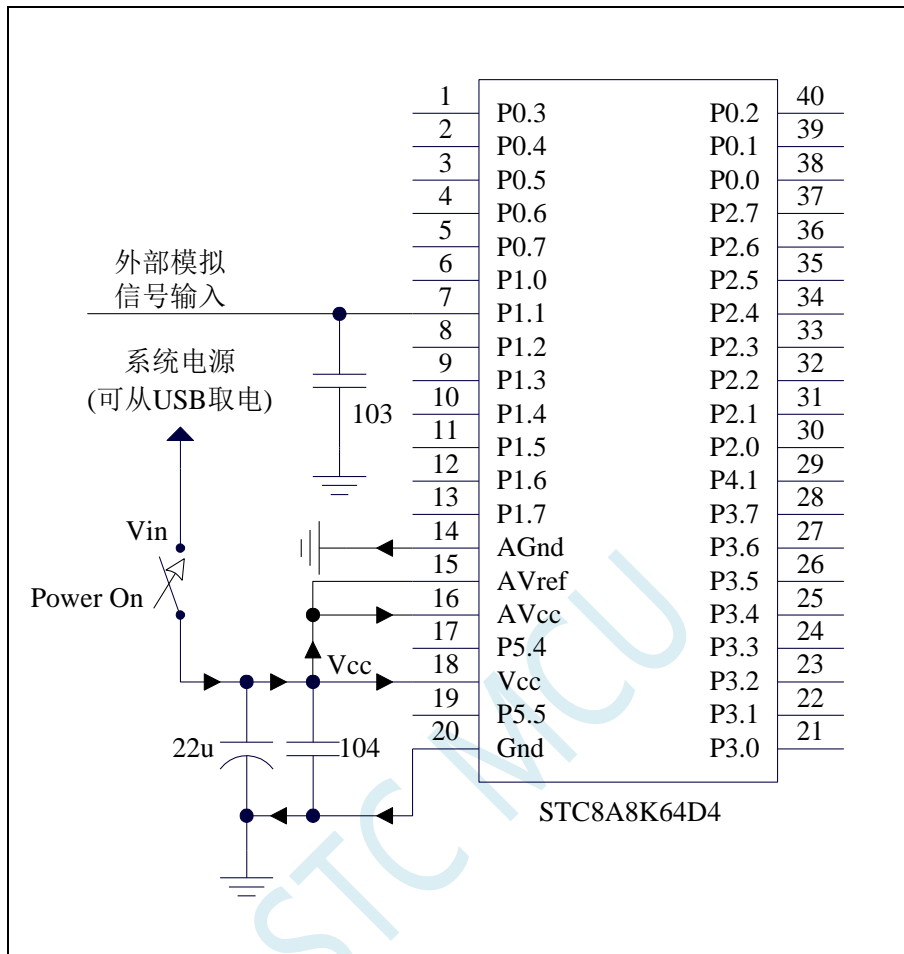
符号	描述	最小值	典型值	最大值	单位
RES	分辨率	-	12	-	Bits
E <sub>T</sub>	整体误差	-	0.5	1	LSB
E <sub>O</sub>	偏移误差	-	-0.1	1	LSB
E <sub>G</sub>	增益误差	-	0	1	LSB
E <sub>D</sub>	微分非线性误差	-	0.7	1.5	LSB
E <sub>I</sub>	积分非线性误差	-	1	2	LSB
R <sub>AIN</sub>	通道等效电阻	-	∞	-	ohm
R <sub>ESD</sub>	采样保持电容前串接的抗静电电阻	-	700	-	ohm
C <sub>ADC</sub>	内部采样保持电容	-	16.5	-	pF

## 17.4 ADC 应用参考线路图

### 17.4.1 高精度 ADC 应用



### 17.4.2 ADC 一般应用（对 ADC 精度要求不高的应用）



## 17.5 范例程序

### 17.5.1 ADC 基本操作（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    ADC_CONTR  =    0xbc;
sfr    ADC_RES    =    0xbd;
sfr    ADC_RESL   =    0xbe;
sfr    ADCCFG     =    0xde;
```

```
sfr    P_SW2     =    0xba;
#define ADCTIM    (*(unsigned char volatile xdata *)0xfea8)
```

```
sfr    P0M0      =    0x91;
sfr    P0M1      =    0x92;
sfr    P0M2      =    0x93;
sfr    P0M3      =    0x94;
sfr    P2M0      =    0x95;
sfr    P2M1      =    0x96;
sfr    P3M0      =    0xb1;
sfr    P3M1      =    0xb2;
sfr    P4M0      =    0xb3;
sfr    P4M1      =    0xb4;
sfr    P5M0      =    0xc9;
sfr    P5M1      =    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;
```

```
    P1M0 = 0x00;
```

```
//设置 P1.0 为 ADC 口
```

```
    P1M1 = 0x01;
```

```
    P_SW2 /= 0x80;
```

```
//设置 ADC 内部时序
```

```
    ADCTIM = 0x3f;
```

```
    P_SW2 &= 0x7f;
```

```
    ADCCFG = 0x0f;
```

```
//设置 ADC 时钟为系统时钟/2/16
```

```
    ADC_CONTR = 0x80;
```

```
//使能 ADC 模块
```

```
    while (1)
```

```

{
    ADC_CONTR |= 0x40;           //启动AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20)); //查询ADC 完成标志
    ADC_CONTR &= ~0x20;        //清完成标志
    P2 = ADC_RES;              //读取ADC 结果
}
}

```

## 汇编代码

;测试工作频率为11.0592MHz

```

ADC_CONTR  DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P_SW2       DATA    0BAH
ADCTIM      XDATA    0FEA8H

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

                ORG    0000H
                LJMP   MAIN

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

                MOV    P1M0, #00H           ;设置P1.0 为ADC 口
                MOV    P1M1, #01H
                MOV    P_SW2, #80H
                MOV    DPTR, #ADCTIM      ;设置ADC 内部时序
                MOV    A, #3FH

```

```

MOVX    @DPTR,A
MOV     P_SW2,#00H
MOV     ADCCFG,#0FH      ;设置ADC 时钟为系统时钟/2/16
MOV     ADC_CONTR,#80H   ;使能ADC 模块
LOOP:
ORL     ADC_CONTR,#40H   ;启动AD 转换
NOP
NOP
MOV     A,ADC_CONTR      ;查询ADC 完成标志
JNB     ACC.5,$-2
ANL     ADC_CONTR,#NOT 20H ;清完成标志
MOV     P2,ADC_RES       ;读取ADC 结果

SJMP    LOOP

END

```

## 17.5.2 ADC 基本操作（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     ADC_CONTR = 0xbc;
sfr     ADC_RES  = 0xbd;
sfr     ADC_RESL  = 0xbe;
sfr     ADCCFG   = 0xde;

sfr     P_SW2    = 0xba;
#define  ADCTIM  (*(unsigned char volatile xdata *)0xfea8)

sbit    EADC     = IE^5;

sfr     P1MI     = 0x91;
sfr     P1M0     = 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;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;      //清中断标志
    P2 = ADC_RES;           //读取ADC 结果
    ADC_CONTR |= 0x40;      //继续AD 转换
}

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;

    P1M0 = 0x00;           //设置P1.0 为ADC 口
    P1M1 = 0x01;
    P_SW2 /= 0x80;
    ADCTIM = 0x3f;        //设置ADC 内部时序
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;       //设置ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x80;    //使能ADC 模块
    EADC = 1;           //使能ADC 中断
    EA = 1;
    ADC_CONTR /= 0x40;   //启动AD 转换

    while (1);
}

```

## 汇编代码

;测试工作频率为 11.0592MHz

ADC_CONTR	DATA	0BCH
ADC_RES	DATA	0BDH
ADC_RESL	DATA	0BEH
ADCCFG	DATA	0DEH
P_SW2	DATA	0BAH
ADCTIM	XDATA	0FEA8H
EADC	BIT	IE.5
P1M1	DATA	091H
P1M0	DATA	092H
P0M1	DATA	093H
P0M0	DATA	094H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
	ORG	0000H
	LJMP	MAIN
	ORG	002BH
	LJMP	ADCISR



```

ORG      0100H

ADCISR:
ANL      ADC_CONTR,#NOT 20H    ;清完成标志
MOV      P2,ADC_RES           ;读取ADC 结果
ORL      ADC_CONTR,#40H       ;继续AD 转换
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      P1M0,#00H             ;设置P1.0 为ADC 口
MOV      P1M1,#01H
MOV      P_SW2,#80H
MOV      DPTR,#ADCTIM         ;设置ADC 内部时序
MOV      A,#3FH
MOVX     @DPTR,A
MOV      P_SW2,#00H
MOV      ADCCFG,#0FH          ;设置ADC 时钟为系统时钟/2/16
MOV      ADC_CONTR,#80H      ;使能ADC 模块
SETB     EADC                 ;使能ADC 中断
SETB     EA
ORL      ADC_CONTR,#40H       ;启动AD 转换

SJMP     $

END

```

### 17.5.3 格式化 ADC 转换结果

#### C 语言代码

```

//测试工作频率为11.0592MHz

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

sfr      ADC_CONTR  = 0xbc;
sfr      ADC_RES    = 0xbd;
sfr      ADC_RESL   = 0xbe;
sfr      ADCCFG     = 0xde;

sfr      P_SW2     = 0xba;
#define   ADCTIM    (*(unsigned char volatile xdata *)0xfea8)

```

```

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;

```

```

void main()
{
    P0M0 = 0x00;
    P0MI = 0x00;
    P1M0 = 0x00;
    P1MI = 0x00;
    P2M0 = 0x00;
    P2MI = 0x00;
    P3M0 = 0x00;
    P3MI = 0x00;
    P4M0 = 0x00;
    P4MI = 0x00;
    P5M0 = 0x00;
    P5MI = 0x00;

    P1M0 = 0x00;           //设置P1.0 为ADC 口
    P1MI = 0x01;
    P_SW2 /= 0x80;
    ADCTIM = 0x3f;        //设置ADC 内部时序
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;       //设置ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x80;    //使能ADC 模块
    ADC_CONTR /= 0x40;   //启动AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20)); //查询ADC 完成标志
    ADC_CONTR &= ~0x20;       //清完成标志

    ADCCFG = 0x00;         //设置结果左对齐
    ACC = ADC_RES;        //A 存储ADC 的10 位结果的高8 位
    B = ADC_RESL;         //B[7:6]存储ADC 的10 位结果的低2 位,B[5:0]为0

    // ADCCFG = 0x20;     //设置结果右对齐
    // ACC = ADC_RES;    //A[1:0]存储ADC 的10 位结果的高2 位,A[7:2]为0
    // B = ADC_RESL;     //B 存储ADC 的10 位结果的低8 位

    while (1);
}

```

## 汇编代码

;测试工作频率为11.0592MHz

```

ADC_CONTR  DATA    0BCH
ADC_RES    DATA    0BDH
ADC_RESL   DATA    0BEH

```

```

ADCCFG      DATA      0DEH

P_SW2       DATA      0BAH
ADCTIM      XDATA      0FEA8H

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

                ORG      0000H
                LJMP     MAIN

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

                MOV      P1M0, #00H           ;设置P1.0为ADC口
                MOV      P1M1, #01H
                MOV      P_SW2, #80H
                MOV      DPTR, #ADCTIM       ;设置ADC内部时序
                MOV      A, #3FH
                MOVX     @DPTR, A
                MOV      P_SW2, #00H
                MOV      ADCCFG, #0FH       ;设置ADC时钟为系统时钟/2/16
                MOV      ADC_CONTR, #80H    ;使能ADC模块

                ORL      ADC_CONTR, #40H    ;启动AD转换
                NOP
                NOP
                MOV      A, ADC_CONTR       ;查询ADC完成标志
                JNB     ACC.5, $-2
                ANL     ADC_CONTR, #NOT 20H ;清完成标志

                MOV      ADCCFG, #00H       ;设置结果左对齐
                MOV      A, ADC_RES         ;A存储ADC的10位结果的高8位
                MOV      B, ADC_RES         ;B[7:6]存储ADC的10位结果的低2位, B[5:0]为0
;
                MOV      ADCCFG, #20H       ;设置结果右对齐

```

```

;          MOV      A,ADC_RES      ;A[3:0]存储ADC 的10 位结果的高2 位,A[7:2]为0
;          MOV      B,ADC_RESL    ;B 存储ADC 的10 位结果的低8 位

          SJMP      $

          END

```

## 17.5.4 ADC 自动转换多次取平均值

### C 语言代码

```
//测试工作频率为11.0592MHz;
```

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

```
sfr      ADC_CONTR = 0xbc;
sfr      ADC_RES   = 0xbd;
sfr      ADC_RESL  = 0xbe;
sfr      ADCCFG    = 0xde;
```

```
sfr      P_SW2     = 0xba;
#define   ADCTIM    (*(unsigned char volatile xdata *)0xfea8)
#define   ADCEXCFG  (*(unsigned char volatile xdata *)0xfead)
```

```
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;
```

```
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;
```

```
    P1M0 = 0x00;
```

```
    P1M1 = 0x01;
```

```
    P_SW2 /= 0x80;
```

```
    ADCTIM = 0x3f;
```

```
// ADCEXCFG = 0x04;
```

```
//设置P1.0 为ADC 口
```

```
//设置ADC 内部时序
```

```
//设置ADC 自动转换2 此取平均值
```

```

// ADCEXCFG = 0x05;           //设置ADC 自动转换4 此取平均值
ADCEXCFG = 0x06;           //设置ADC 自动转换8 此取平均值
// ADCEXCFG = 0x07;           //设置ADC 自动转换16 此取平均值
P_SW2 &= 0x7f;
ADCCFG = 0x0f;              //设置ADC 时钟为系统时钟/2/16
ADC_CONTR = 0x80;          //使能ADC 模块

while (1)
{
    ADC_CONTR |= 0x40;       //启动AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20)); //查询ADC 完成标志
    ADC_CONTR &= ~0x20;     //清完成标志
    P2 = ADC_RES;           //读取ADC 结果
}
}

```

## 汇编代码

;测试工作频率为11.0592MHz

ADC_CONTR	DATA	0BCH
ADC_RES	DATA	0BDH
ADC_RESL	DATA	0BEH
ADCCFG	DATA	0DEH
P_SW2	DATA	0BAH
ADCTIM	XDATA	0FEA8H
ADCEXCFG	XDATA	0FEAdH
P1M1	DATA	091H
P1M0	DATA	092H
P0M1	DATA	093H
P0M0	DATA	094H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
	ORG	0000H
	LJMP	MAIN
	ORG	0100H
MAIN:	MOV	SP, #5FH
	MOV	P0M0, #00H
	MOV	P0M1, #00H
	MOV	P1M0, #00H
	MOV	P1M1, #00H
	MOV	P2M0, #00H
	MOV	P2M1, #00H
	MOV	P3M0, #00H
	MOV	P3M1, #00H
	MOV	P4M0, #00H

```

MOV      P4M1, #00H
MOV      P5M0, #00H
MOV      P5M1, #00H

MOV      P1M0, #00H          ;设置 P1.0 为 ADC 口
MOV      P1M1, #01H
MOV      P_SW2, #80H
MOV      DPTR, #ADCTIM      ;设置 ADC 内部时序
MOV      A, #3FH
MOVX     @DPTR, A
MOV      DPTR, #ADCEXCFG
; MOV      A, #04H          ;设置 ADC 自动转换 2 此取平均值
MOV      A, #05H          ;设置 ADC 自动转换 4 此取平均值
; MOV      A, #06H          ;设置 ADC 自动转换 8 此取平均值
; MOV      A, #07H          ;设置 ADC 自动转换 16 此取平均值
MOVX     @DPTR, A
MOV      P_SW2, #00H
MOV      ADCCFG, #0FH      ;设置 ADC 时钟为系统时钟/2/16
MOV      ADC_CONTR, #80H   ;使能 ADC 模块

LOOP:    ORL      ADC_CONTR, #40H   ;启动 AD 转换
NOP
NOP
MOV      A, ADC_CONTR      ;查询 ADC 完成标志
JNB     ACC.5, $-2
ANL     ADC_CONTR, #NOT 20H ;清完成标志
MOV      P2, ADC_RES       ;读取 ADC 结果

SJMP    LOOP

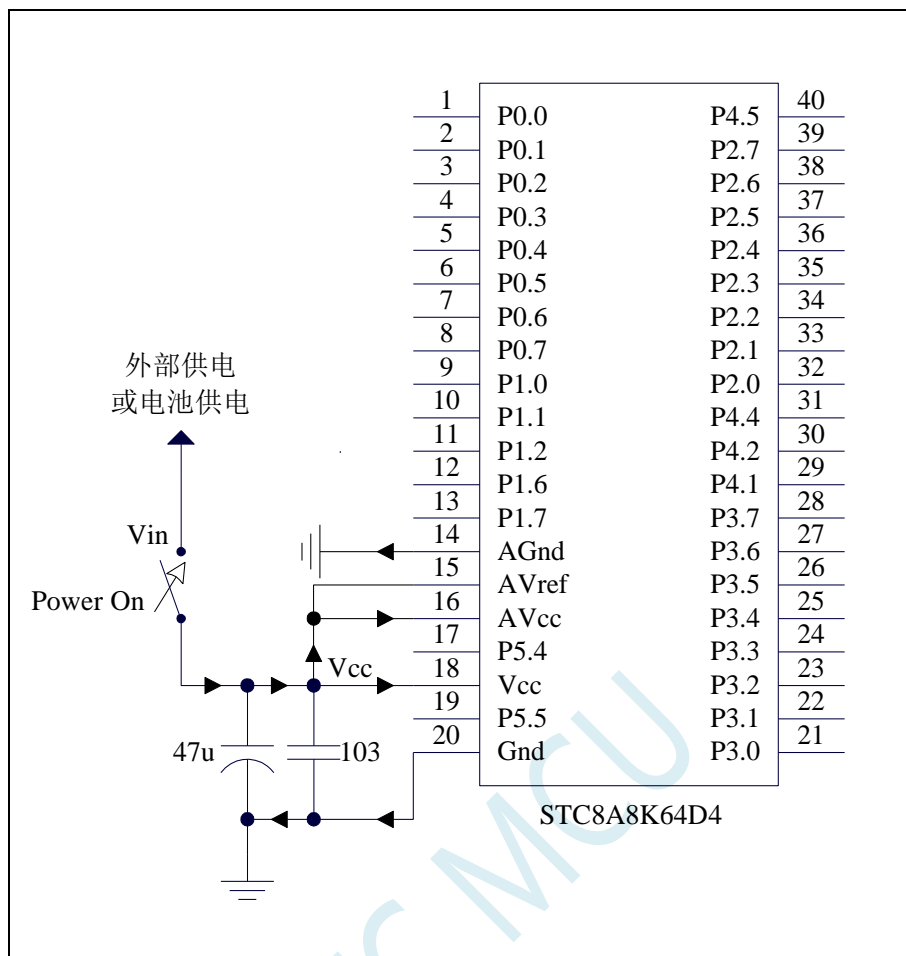
END

```

## 17.5.5 利用 ADC 第 15 通道测量外部电压或电池电压

STC8A8K64D4 系列 ADC 的第 15 通道用于测量内部参考信号源，由于内部参考信号源很稳定，约为 1.19V，且不会随芯片的工作电压的改变而变化，所以可以通过测量内部 1.19V 参考信号源，然后通过 ADC 的值便可反推出外部电压或外部电池电压。

下图为参考线路图:



## C 语言代码

//测试工作频率为11.0592MHz;

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

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

```
sfr AUXR = 0x8e;
```

```
sfr ADC_CONTR = 0xbc;
```

```
sfr ADC_RES = 0xbd;
```

```
sfr ADC_RESL = 0xbe;
```

```
sfr ADCCFG = 0xde;
```

```
sfr P_SW2 = 0xba;
```

```
#define ADCTIM (*(unsigned char volatile xdata *)0xfea8)
```

```
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;

int    *BGV;

//内部参考信号源值存放在idata 中
//idata 的EFH 地址存放高字节
//idata 的FOH 地址存放低字节
//电压单位为毫伏(mV)

bit    busy;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void ADCInit()
{
    P_SW2 /= 0x80;
    ADCTIM = 0x3f;
    P_SW2 &= 0x7f;

    //设置ADC 内部时序

    ADCCFG = 0x2f;
    ADC_CONTR = 0x8f;

    //设置ADC 时钟为系统时钟/2/16
    //使能ADC 模块,并选择第15 通道
}

int    ADCRead()
{
    int res;
}

```



```

    ADC_CONTR |= 0x40;           //启动AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20)); //查询ADC 完成标志
    ADC_CONTR &= ~0x20;         //清完成标志
    res = (ADC_RES << 8) / ADC_RES_L; //读取ADC 结果

    return res;
}

void main()
{
    int res;
    int vcc;
    int i;

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    BGV = (int idata *)0xef;
    ADCInit();                 //ADC 初始化
    UartInit();                 //串口初始化

    ES = 1;
    EA = 1;

    // ADCRead();
    // ADCRead();               //前两个数据建议丢弃

    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();      //读取 8 次数据
    }
    res >>= 3;                 //取平均值

    vcc = (int)(4096L * *BGV / res); //((12 位ADC 算法)计算 VREF 管脚电压,即电池电压)
    // vcc = (int)(1024L * *BGV / res); //((10 位ADC 算法)计算 VREF 管脚电压,即电池电压)
    //注意,此电压的单位为毫伏(mV)

    UartSend(vcc >> 8);        //输出电压值到串口
    UartSend(vcc);

    while (1);
}

```

上面的方法是使用 ADC 的第 15 通道反推外部电池电压的。在 ADC 测量范围内, ADC 的外部测量电压与 ADC 的测量值是成正比例的, 所以也可以使用 ADC 的第 15 通道反推外部通道输入电压, 假设当前已获取了内部参考信号源电压为 BGV, 内部参考信号源的 ADC 测量值为  $res_{bg}$ , 外部通道输入电压

的 ADC 测量值为  $res_x$ ，则外部通道输入电压  $V_x = BGV / res_{bg} * res_x$ ；

## 17.5.6 ADC 做电容感应触摸按键

按键是电路最常用的零件之一，是人机界面重要的输入方式，我们最熟悉的是机械式按键，但是机械按键有一个缺点（特别是便宜的按键），触点有寿命，很容易出现接触不良而失效。而非接触的按键则没有机械触点，寿命长，使用方便。

非接触的按键有多种方案，而电容感应按键则是低成本方案，多年前一般是使用专门的 IC 来实现，随着 MCU 功能的加强，以及广大用户的实践经验，直接使用 MCU 来做电容感应按键的技术已经成熟，其中最典型最可靠的是使用 ADC 做的方案。

本文档详述使用 STC 带 ADC 的系列 MCU 做的方案，可以使用任何带 ADC 功能的 MCU 来实现。下面前 3 个图是用得最多的方式，原理都一样，本文使用第 2 个图。

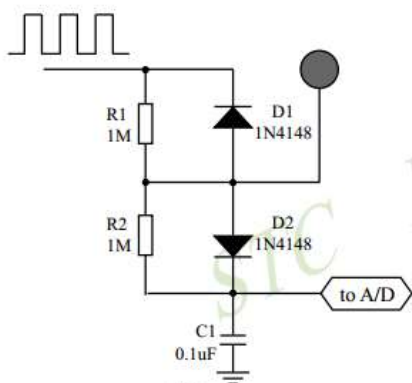


图1

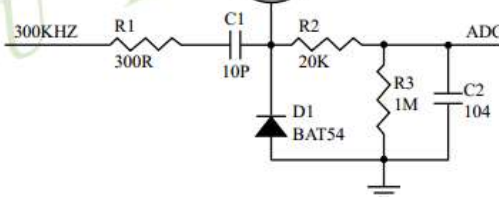


图2

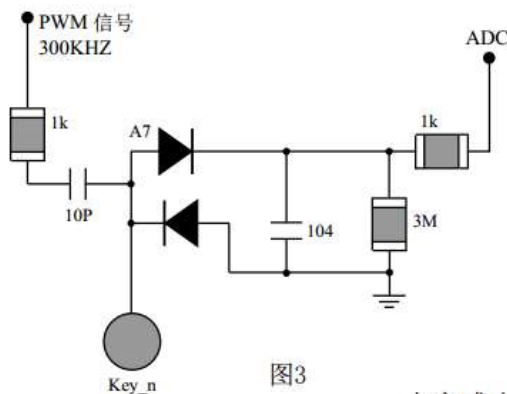


图3

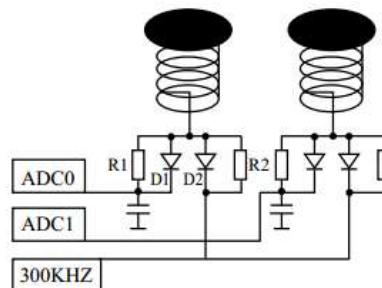
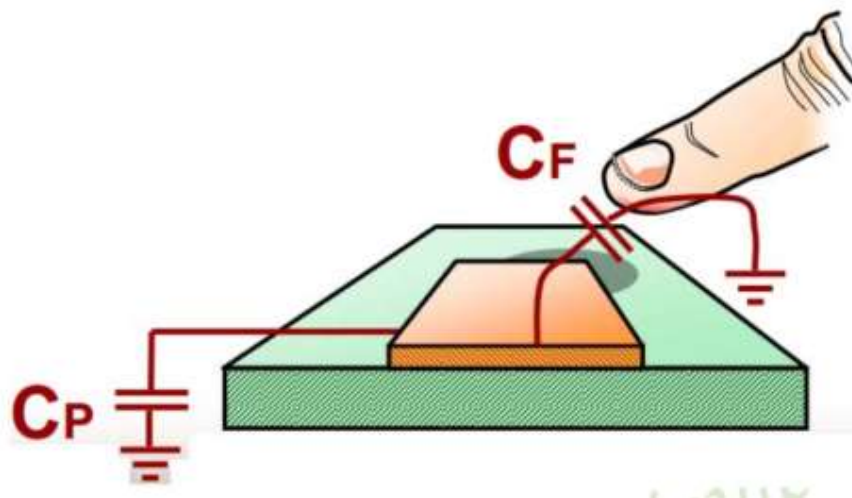


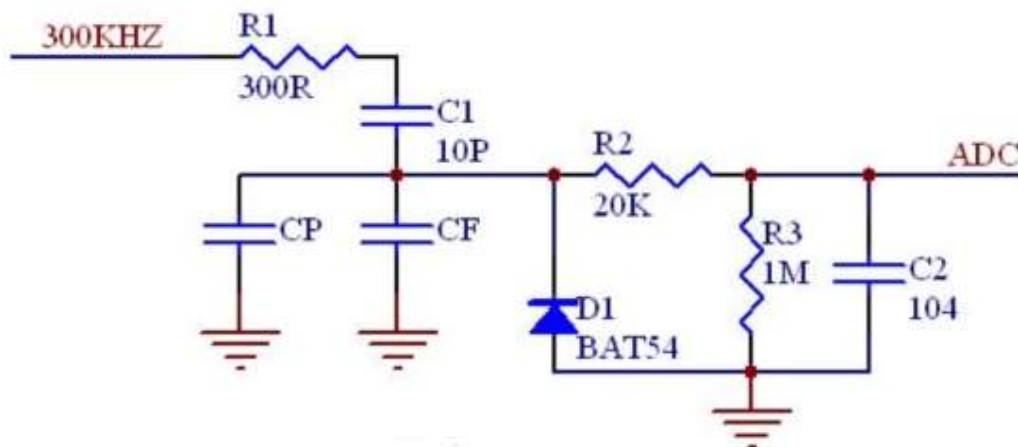
图4 加了感应弹簧

### 电容感应按键取样电路

一般实际应用时，都使用图 4 所示的感应弹簧来加大手指按下的面积。感应弹簧等效一块对地的金属板，对地有一个电容  $C_P$ ，而手指按下后，则再并联一个对地的电容  $C_F$ ，如下图所示。



下面为电路图的说明，CP 为金属板和分布电容，CF 为手指电容，并联在一起与 C1 对输入的 300KHZ 方波进行分压，经过 D1 整流，R2、C2 滤波后送 ADC，当手指压上去后，送去 ADC 的电压降低，程序就可以检测出按键动作。



## C 语言代码

//测试工作频率为 24MHz

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

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

```
#define MAIN_Fosc 2400000UL
```

//定义主时钟

```
#define Timer0_Reload (65536UL-(MAIN_Fosc / 60000)) //Timer 0 重装值, 对应300KHZ
```

```
typedef unsigned char u8;
```

```
typedef unsigned int u16;
```

```
typedef unsigned long u32;
```

```
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    ADC_CONTR = 0xBC;           //带AD 系列
sfr    ADC_RES   = 0xBD;           //带AD 系列
sfr    ADC_RESL  = 0xBE;           //带AD 系列
sfr    AUXR      = 0x8E;
sfr    AUXR2     = 0x8F;

#define CHANNEL 8                   //ADC 通道数
#define ADC_90T (3<<5)              //ADC 时间 90T
#define ADC_180T (2<<5)              //ADC 时间 180T
#define ADC_360T (1<<5)              //ADC 时间 360T
#define ADC_540T 0                   //ADC 时间 540T
#define ADC_FLAG (1<<4)              //软件清0
#define ADC_START (1<<3)             //自动清0

sbit   P_LED7   = P2^7;
sbit   P_LED6   = P2^6;
sbit   P_LED5   = P2^5;
sbit   P_LED4   = P2^4;
sbit   P_LED3   = P2^3;
sbit   P_LED2   = P2^2;
sbit   P_LED1   = P2^1;
sbit   P_LED0   = P2^0;

u16 idata adc[TOUCH_CHANNEL];      //当前ADC 值
u16 idata adc_prev[TOUCH_CHANNEL]; //上一个ADC 值
u16 idata TouchZero[TOUCH_CHANNEL]; //0 点ADC 值
u8 idata TouchZeroCnt[TOUCH_CHANNEL]; //0 点自动跟踪计数
u8 cnt_250ms;

void delay_ms(u8 ms);
void ADC_init(void);
u16 Get_ADC10bitResult(u8 channel);
void AutoZero(void);
u8 check_adc(u8 index);
void ShowLED(void);

void main(void)
{
    u8 i;

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

```

```

delay_ms(50);
ET0 = 0; //初始化 Timer0 输出一个 300KHZ 时钟
TR0 = 0;
AUXR |= 0x80; //Timer0 set as 1T mode
AUXR2 |= 0x01; //允许输出时钟
TMOD = 0; //Timer0 set as Timer, 16 bits Auto Reload.
TH0 = (u8)(Timer0_Reload >> 8);
TL0 = (u8)Timer0_Reload;
TR0 = 1;
ADC_init(); //ADC 初始化
delay_ms(50); //延时 50ms
for (i=0; i<TOUCH_CHANNEL; i++) //初始化 0 点和上一个值和 0 点自动跟踪计数
{
    adc_prev[i] = 1023;
    TouchZero[i] = 1023;
    TouchZeroCnt[i] = 0;
}
cnt_250ms = 0;
while (1)
{
    delay_ms(50); //每隔 50ms 处理一次按键
    ShowLED();
    if (++cnt_250ms >= 5)
    {
        cnt_250ms = 0;
        AutoZero(); //每隔 250ms 处理一次 0 点自动跟踪
    }
}

void delay_ms(u8 ms)
{
    unsigned int i;

    do
    {
        i = MAIN_Fosc / 10000;
        while(--i);
    } while(--ms);
}

void ADC_init(void)
{
    P1M0 = 0x00; //8 路 ADC
    P1M1 = 0xff;
    ADC_CONTR = 0x80; //允许 ADC
}

u16 Get_ADC10bitResult(u8 channel)
{
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_90T | ADC_START | channel; //触发 ADC
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    while((ADC_CONTR & ADC_FLAG) == 0); //等待 ADC 转换结束
}

```

```

    ADC_CONTR = 0x80; //清除标志
    return(((u16)ADC_RES << 2) | ((u16)ADC_RES1 & 3)); //返回ADC 结果
}

void AutoZero(void) //250ms 调用一次
//这是使用相邻 2 个采样的差的绝对值之和来检测。
{
    u8 i;
    u16 j,k;

    for(i=0; i<TOUCH_CHANNEL; i++) //处理 8 个通道
    {
        j = adc[i];
        k = j - adc_prev[i]; //减前一个读数
        F0 = 0; //按下
        if(k & 0x8000) F0 = 1, k = 0 - k; //释放 求出两次采样的差值
        if(k >= 20) //变化比较大
        {
            TouchZeroCnt[i] = 0; //如果变化比较大, 则清 0 计数器
            if(F0) TouchZero[i] = j; //如果是释放, 并且变化比较大, 则直接替代
        }
        else //变化比较小, 则蠕动, 自动 0 点跟踪
        {
            if(++TouchZeroCnt[i] >= 20) //连续检测到小变化 20 次/4 = 5 秒
            {
                TouchZeroCnt[i] = 0;
                TouchZero[i] = adc_prev[i]; //变化缓慢的值作为 0 点
            }
            adc_prev[i] = j; //保存这一次的采样值
        }
    }
}

u8 check_adc(u8 index) //获取触摸信息函数 50ms 调用 1 次
//判断键按下或释放, 有回差控制
{
    u16 delta;

    adc[index] = 1023 - Get_ADC10bitResult(index); //获取ADC 值, 转成按下键, ADC 值增加
    if(adc[index] < TouchZero[index]) return 0; //比 0 点还小的值, 则认为是键释放
    delta = adc[index] - TouchZero[index];
    if(delta >= 40) return 1; //键按下
    if(delta <= 20) return 0; //键释放
    return 2; //保持原状态
}

void ShowLED(void)
{
    u8 i;

    i = check_adc(0);
    if(i == 0) P_LED0 = 1; //指示灯灭
    if(i == 1) P_LED0 = 0; //指示灯亮
    i = check_adc(1);
    if(i == 0) P_LED1 = 1; //指示灯灭
    if(i == 1) P_LED1 = 0; //指示灯亮
    i = check_adc(2);
    if(i == 0) P_LED2 = 1; //指示灯灭
    if(i == 1) P_LED2 = 0; //指示灯亮
    i = check_adc(3);
}

```

```

    if(i == 0) P_LED3 = 1;           //指示灯灭
    if(i == 1) P_LED3 = 0;           //指示灯亮
    i = check_adc(4);
    if(i == 0) P_LED4 = 1;           //指示灯灭
    if(i == 1) P_LED4 = 0;           //指示灯亮
    i = check_adc(5);
    if(i == 0) P_LED5 = 1;           //指示灯灭
    if(i == 1) P_LED5 = 0;           //指示灯亮
    i = check_adc(6);
    if(i == 0) P_LED6 = 1;           //指示灯灭
    if(i == 1) P_LED6 = 0;           //指示灯亮
    i = check_adc(7);
    if(i == 0) P_LED7 = 1;           //指示灯灭
    if(i == 1) P_LED7 = 0;           //指示灯亮
}

```

## 汇编代码

;测试工作频率为24MHz

<i>Fosc_KHZ</i>	<i>EQU</i>	24000	;定义主时钟 KHZ
<i>Reload</i>	<i>EQU</i>	(65536 - <i>Fosc_KHZ</i> /600)	;Timer 0 重装值,对应 300KHZ
<i>ADC_CONTR</i>	<i>DATA</i>	0xBC	;带AD 系列
<i>ADC_RES</i>	<i>DATA</i>	0xBD	;带AD 系列
<i>ADC_RESL</i>	<i>DATA</i>	0xBE	;带AD 系列
<i>AUXR</i>	<i>DATA</i>	0x8E	
<i>AUXR2</i>	<i>DATA</i>	0x8F	
<i>P0M1</i>	<i>DATA</i>	093H	
<i>P0M0</i>	<i>DATA</i>	094H	
<i>P1M1</i>	<i>DATA</i>	091H	
<i>P1M0</i>	<i>DATA</i>	092H	
<i>P2M1</i>	<i>DATA</i>	095H	
<i>P2M0</i>	<i>DATA</i>	096H	
<i>P3M1</i>	<i>DATA</i>	0B1H	
<i>P3M0</i>	<i>DATA</i>	0B2H	
<i>P4M1</i>	<i>DATA</i>	0B3H	
<i>P4M0</i>	<i>DATA</i>	0B4H	
<i>P5M1</i>	<i>DATA</i>	0C9H	
<i>P5M0</i>	<i>DATA</i>	0CAH	
<i>CHANNEL</i>	<i>EQU</i>	8	;ADC 通道数
<i>ADC_90T</i>	<i>EQU</i>	(3 SHL 5)	;ADC 时间 90T
<i>ADC_180T</i>	<i>EQU</i>	(2 SHL 5)	;ADC 时间 180T
<i>ADC_360T</i>	<i>EQU</i>	(1 SHL 5)	;ADC 时间 360T
<i>ADC_540T</i>	<i>EQU</i>	0	;ADC 时间 540T
<i>ADC_FLAG</i>	<i>EQU</i>	(1 SHL 4)	;软件清0
<i>ADC_START</i>	<i>EQU</i>	(1 SHL 3)	;自动清0
<i>P_LED7</i>	<i>BIT</i>	P2.7;	
<i>P_LED6</i>	<i>BIT</i>	P2.6;	
<i>P_LED5</i>	<i>BIT</i>	P2.5;	
<i>P_LED4</i>	<i>BIT</i>	P2.4;	
<i>P_LED3</i>	<i>BIT</i>	P2.3;	
<i>P_LED2</i>	<i>BIT</i>	P2.2;	
<i>P_LED1</i>	<i>BIT</i>	P2.1;	
<i>P_LED0</i>	<i>BIT</i>	P2.0;	

```

adc      EQU      30H      ;当前ADC 值 30H~3FH,两字节一个值
adc_prev EQU      40H      ;上一个ADC 值 40H~4FH,两字节一个值
TouchZero EQU      50H      ;0 点ADC 值 50H~5FH,两字节一个值
TouchZeroCnt EQU      60H      ;0 点自动跟踪计数 60H~67H
cnt_250ms DATA      68H

ORG      0000H
LJMP     MAIN

MAIN:
ORG      0100H

MOV      SP,#0D0H
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      R7,#50
LCALL    F_delay_ms
CLR      ET0      ;初始化Timer0 输出一个300KHZ 时钟
CLR      TR0
ORL      AUXR,#080H ;Timer0 set as 1T mode
ORL      AUXR2,#01H ;允许输出时钟
MOV      TMOD,#0   ;Timer0 set as Timer,16 bits Auto Reload.
MOV      TH0,#HIGH Reload
MOV      TL0,#LOW Reload
SETB     TR0
LCALL    F_ADC_init
MOV      R7,#50
LCALL    F_delay_ms
MOV      R0,#adc_prev ;初始化上一个ADC 值

L_Init_Loop1:
MOV      @R0,#03H
INC      R0
MOV      @R0,#0FFH
INC      R0
MOV      A,R0
CJNE     A,#(adc_prev + CHANNEL * 2),L_Init_Loop1
MOV      R0,#TouchZero ;初始化0 点ADC 值

L_Init_Loop2:
MOV      @R0,#03H
INC      R0
MOV      @R0,#0FFH
INC      R0
MOV      A,R0
CJNE     A,#(TouchZero+CHANNEL * 2),L_Init_Loop2
MOV      R0,#TouchZeroCnt ;初始化自动跟踪计数值

L_Init_Loop3:
MOV      @R0,#0
INC      R0
MOV      A,R0

```



```

        CJNE    A,#(TouchZeroCnt + CHANNEL),L_Init_Loop3
        MOV     cnt_250ms,#5

L_MainLoop:
        MOV     R7,#50                ;延时 50ms
        LCALL  F_delay_ms
        LCALL  F_ShowLED             ;处理一次触摸键值
        DJNZ   cnt_250ms,L_MainLoop
        MOV     cnt_250ms,#5         ;250ms 处理一次0 点自动跟踪
        LCALL  F_AutoZero           ;自动跟踪零点
        SJMP   L_MainLoop

F_ADC_init:
        MOV     PIM0,#00H           ;8 路ADC
        MOV     PIM1,#0FFH
        MOV     ADC_CONTR,#080H    ;允许ADC
        RET

F_Get_ADC10bitResult:
        MOV     ADC_RES,#0
        MOV     ADC_RESL,#0
        MOV     A,R7
        ORL    A,#0E8H             ;触发ADC
        MOV     ADC_CONTR,A
        NOP
        NOP
        NOP
        NOP

L_10bitADC_Loop1:
        MOV     A,ADC_CONTR
        JNB    ACC.4,L_10bitADC_Loop1 ;等待 ADC 转换结束
        MOV     ADC_CONTR,#080H    ;清除标志
        MOV     A,ADC_RES
        MOV     B,#04H
        MUL   AB
        MOV     R7,A
        MOV     R6,B
        MOV     A,ADC_RESL
        ANL   A,#03H
        ORL   A,R7
        MOV     R7,A
        RET

F_AutoZero:
        ;250ms 调用一次
        ;这是使用相邻 2 个采样的差的绝对值之和来检测。

        CLR    A
        MOV     R5,A

L_AutoZero_Loop:
        MOV     A,R5
        ADD    A,ACC
        ADD    A,#LOW(adc)
        MOV     R0,A
        MOV     A,@R0
        MOV     R6,A
        INC    R0
        MOV     A,@R0
        MOV     R7,A
        MOV     A,R5
        ADD    A,ACC
        ADD    A,#LOW(adc_prev+01H)

```

```

MOV      R0,A
CLR      C
MOV      A,R7
SUBB    A,@R0
MOV      R3,A
MOV      A,R6
DEC      R0
SUBB    A,@R0
MOV      R2,A
CLR      F0 ;按下
JNB     ACC.7,L_AutoZero_1
SETB    F0
CLR      C
CLR      A
SUBB    A,R3
MOV      R3,A
MOV      A,R3
CLR      A
SUBB    A,R2
MOV      R2,A

L_AutoZero_1:
CLR      C ;计算 [R2 R3] - #20,if(k >= 20)
MOV      A,R3
SUBB    A,#20
MOV      A,R2
SUBB    A,#00H
JC      L_AutoZero_2 ;[R2 R3],20,转
MOV      A,#LOW(TouchZeroCnt) ;如果变化比较大, 则清0 计数器 TouchZeroCnt[i] = 0;
ADD      A,R5
MOV      R0,A
MOV      @R0,#0
JNB     F0,L_AutoZero_3
MOV      A,R5
ADD      A,ACC
ADD      A,#LOW(TouchZero)
MOV      R0,A
MOV      @R0,6
INC      R0
MOV      @R0,7
SJMP    L_AutoZero_3

L_AutoZero_2:
;变化比较小, 则蠕动, 自动0 点跟踪
;连续检测到小变化 20 次/4 = 5 秒.
MOV      A,#LOW(TouchZeroCnt)
ADD      A,R5
MOV      R0,A
INC      @R0
MOV      A,@R0
CLR      C
SUBB    A,#20
JC      L_AutoZero_3 ;if(TouchZeroCnt[i] < 20), 转
MOV      @R0,#0 ;TouchZeroCnt[i] = 0;
MOV      A,R5 ;变化缓慢的值作为0 点
ADD      A,ACC
ADD      A,#LOW(adc_prev)
MOV      R0,A
MOV      A,@R0
MOV      R2,A
INC      R0
MOV      A,@R0

```

```

MOV R3,A
MOV A,R5
ADD A,ACC
ADD A,#LOW (TouchZero)
MOV R0,A
MOV @R0,2
INC R0
MOV @R0,3
L_AutoZero_3: ;保存采样值 adc_prev[i] = j;
MOV A,R5
ADD A,ACC
ADD A,#LOW (adc_prev)
MOV R0,A
MOV @R0,6
INC R0
MOV @R0,7
INC R5
MOV A,R5
XRL A,#08H
JZ $ + 5H
LJMP L_AutoZero_Loop
RET

F_check_adc: ;判断键按下或释放,有回差控制
MOV R4,7
LCALL F_Get_ADC10bitResult ;返回的ADC 值在 [R6 R7]
CLR C
MOV A,#0FFH
SUBB A,R7
MOV R7,A
MOV A,#03H
SUBB A,R6
MOV R6,A
MOV A,R4 ;保存 adc[index]
ADD A,ACC
ADD A,#LOW (adc)
MOV R0,A
MOV @R0,6
INC R0
MOV @R0,7
MOV A,R4
ADD A,ACC
ADD A,#LOW (TouchZero+01H)
MOV R1,A
MOV A,R4
ADD A,ACC
ADD A,#LOW (adc)
MOV R0,A
MOV A,@R0
MOV R6,A
INC R0
MOV A,@R0
CLR C
SUBB A,@R1 ;计算 adc[index] - TouchZero[index]
MOV A,R6
DEC R1
SUBB A,@R1
JNC L_check_adc_1
MOV R7,#00H

```

```

    RET
L_check_adc_1:
    MOV     A,R4
    ADD     A,ACC
    ADD     A,#LOW (TouchZero+01H)
    MOV     RI,A
    MOV     A,R4
    ADD     A,ACC
    ADD     A,#LOW (adc+01H)
    MOV     R0,A
    CLR     C
    MOV     A,@R0
    SUBB   A,@R1
    MOV     R7,A
    DEC     R0
    MOV     A,@R0
    DEC     R1
    SUBB   A,@R1
    MOV     R6,A
    CLR     C
    MOV     A,R7
    SUBB   A,#40
    MOV     A,R6
    SUBB   A,#00H
    JC      L_check_adc_2           ;if(delta < 40), 转
    MOV     R7,#1                 ;if(delta >= 40) return 1; //键按下 返回1
    RET
L_check_adc_2:
    SETB   C
    MOV     A,R7
    SUBB   A,#20
    MOV     A,R6
    SUBB   A,#00H
    JNC    L_check_adc_3
    MOV     R7,#0
    RET
L_check_adc_3:
    MOV     R7,#2
    RET

F_ShowLED:
    MOV     R7,#0
    LCALL  F_check_adc
    MOV     A,R7
    ANL    A,#0FEH
    JNZ    L_QuitCheck0
    MOV     A,R7
    MOV     C,ACC.0
    CPL    C
    MOV     P_LED0,C

L_QuitCheck0:
    MOV     R7,#1
    LCALL  F_check_adc
    MOV     A,R7
    ANL    A,#0FEH
    JNZ    L_QuitCheck1
    MOV     A,R7
    MOV     C,ACC.0
    CPL    C

```

```
        MOV        P_LED1,C
L_QuitCheck1:
        MOV        R7,#2
        LCALL     F_check_adc
        MOV        A,R7
        ANL        A,#0FEH
        JNZ        L_QuitCheck2
        MOV        A,R7
        MOV        C,ACC.0
        CPL        C
        MOV        P_LED2,C
L_QuitCheck2:
        MOV        R7,#3
        LCALL     F_check_adc
        MOV        A,R7
        ANL        A,#0FEH
        JNZ        L_QuitCheck3
        MOV        A,R7
        MOV        C,ACC.0
        CPL        C
        MOV        P_LED3,C
L_QuitCheck3:
        MOV        R7,#4
        LCALL     F_check_adc
        MOV        A,R7
        ANL        A,#0FEH
        JNZ        L_QuitCheck4
        MOV        A,R7
        MOV        C,ACC.0
        CPL        C
        MOV        P_LED4,C
L_QuitCheck4:
        MOV        R7,#5
        LCALL     F_check_adc
        MOV        A,R7
        ANL        A,#0FEH
        JNZ        L_QuitCheck5
        MOV        A,R7
        MOV        C,ACC.0
        CPL        C
        MOV        P_LED5,C
L_QuitCheck5:
        MOV        R7,#6
        LCALL     F_check_adc
        MOV        A,R7
        ANL        A,#0FEH
        JNZ        L_QuitCheck6
        MOV        A,R7
        MOV        C,ACC.0
        CPL        C
        MOV        P_LED6,C
L_QuitCheck6:
        MOV        R7,#7
        LCALL     F_check_adc
        MOV        A,R7
        ANL        A,#0FEH
        JNZ        L_QuitCheck7
        MOV        A,R7
        MOV        C,ACC.0
```

```

        CPL            C
        MOV            P_LED7,C
L_QuitCheck7:
        RET

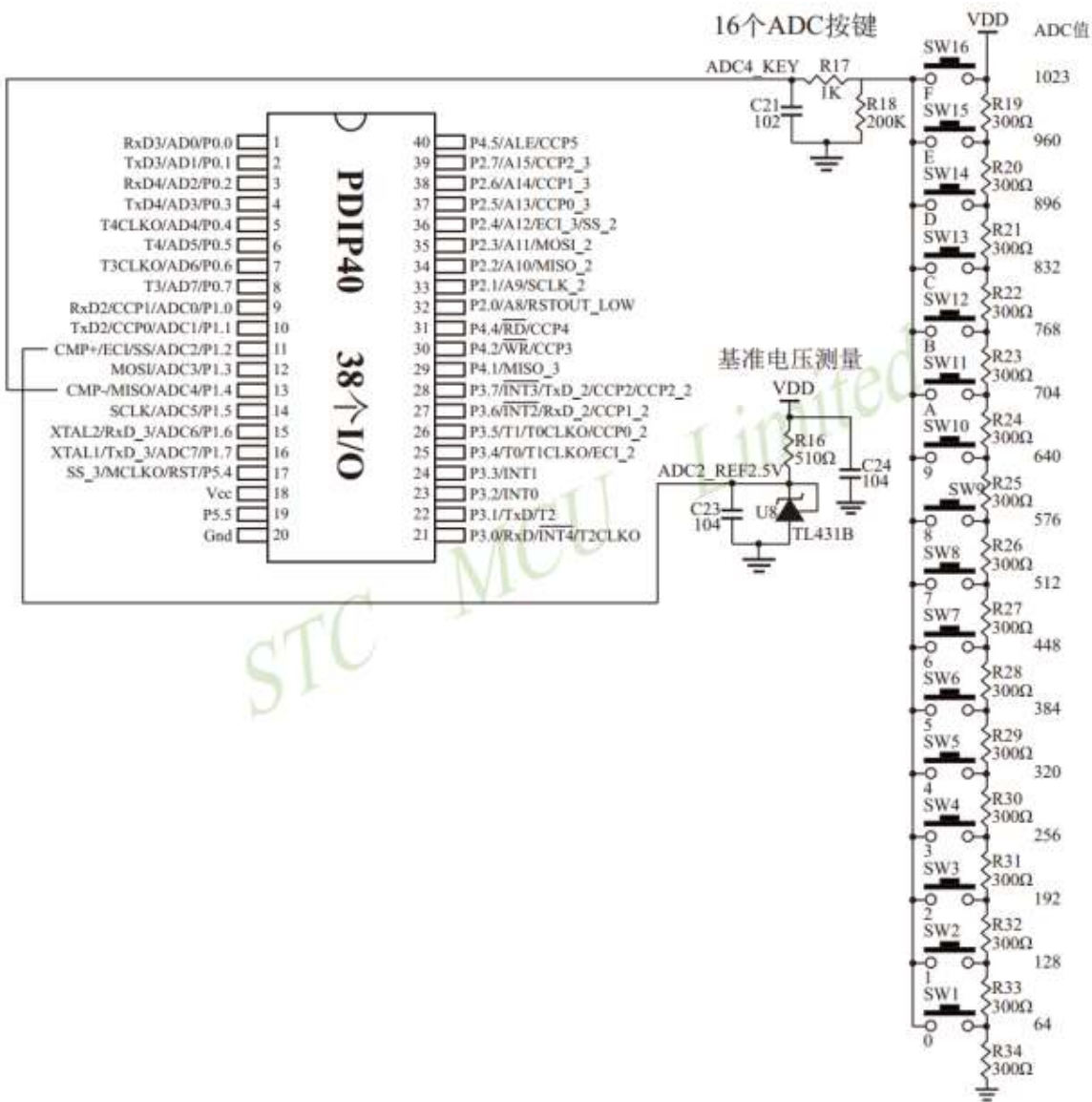
F_delay_ms:
        PUSH          3
        PUSH          4
L_delay_ms_1:
        MOV            R3,#HIGH (Fosc_KHZ / 13)
        MOV            R4,#LOW (Fosc_KHZ / 13)
L_delay_ms_2:
        MOV            A,R4
        DEC            R4
        JNZ            L_delay_ms_3
        DEC            R3
L_delay_ms_3:
        DEC            A
        ORL            A,R3
        JNZ            L_delay_ms_2
        DJNZ          R7,L_delay_ms_1
        POP            4
        POP            3
        RET

        END

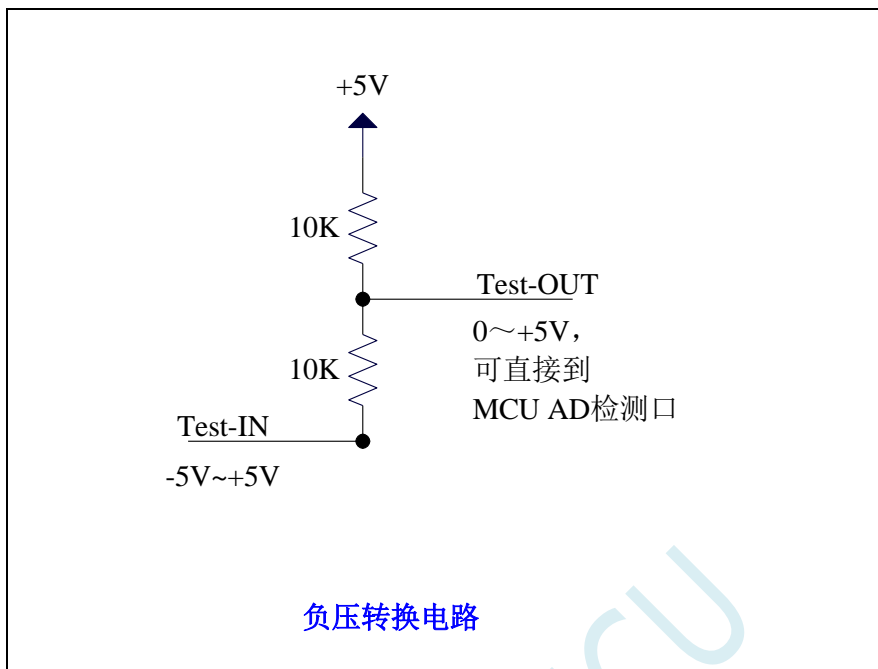
```

## 17.5.7 ADC 作按键扫描应用线路图

读 ADC 键的方法：每隔 10ms 左右读一次 ADC 值，并且保存最后 3 次的读数，其变化比较小时再判断键。判断键有效时，允许一定的偏差，比如 ±16 个字的偏差。

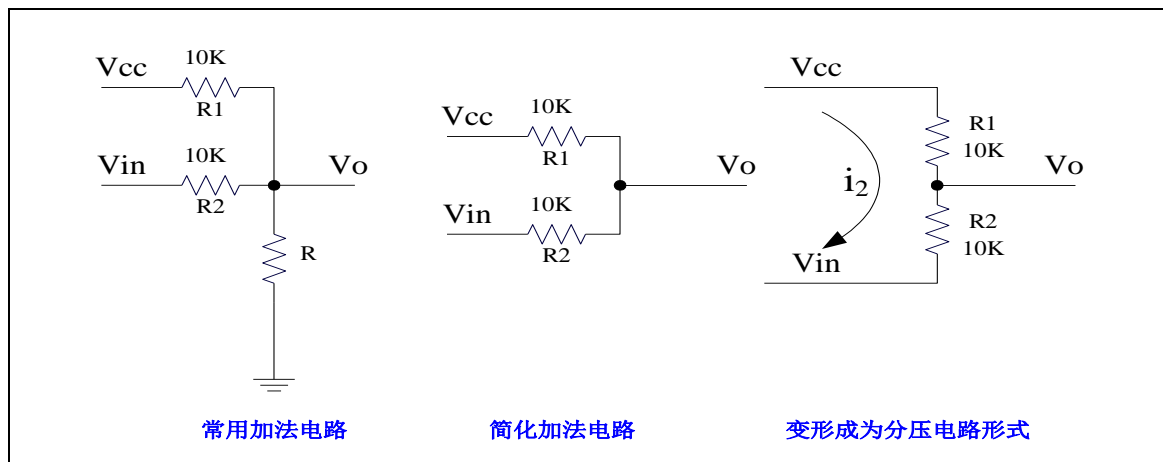


### 17.5.8 检测负电压参考线路图





## 17.5.9 常用加法电路在 ADC 中的应用



参照分压电路得到公式 1

$$\text{公式 1: } V_o = V_{in} + i_2 * R_2$$

$$\text{公式 2: } i_2 = (V_{cc} - V_{in}) / (R_1 + R_2) \{ \text{条件: 流向 } V_o \text{ 的电流 } \approx 0 \}$$

将  $R_1=R_2$  代入公式 2 得公式 3

$$\text{公式 3: } i_2 = (V_{cc} - V_{in}) / 2R_2$$

将公式 3 代入公式 1 得公式 4

$$\text{公式 4: } V_o = (V_{cc} + V_{in}) / 2$$

根据公式 4, 可以将以上电路看成加法电路。

在单片机的模数转换测量中, 要求被测电压大于 0 并且小于  $V_{CC}$ 。如果被测电压小于 0V, 可以利用加法电路将被测电压提升到 0V 以上。此时对被测电压的变化范围有一定的要求:

把上述条件代入公式 4 可得到下面 2 式

$$(V_{cc} + V_{in}) / 2 > 0 \quad \text{即 } V_{in} > -V_{cc}$$

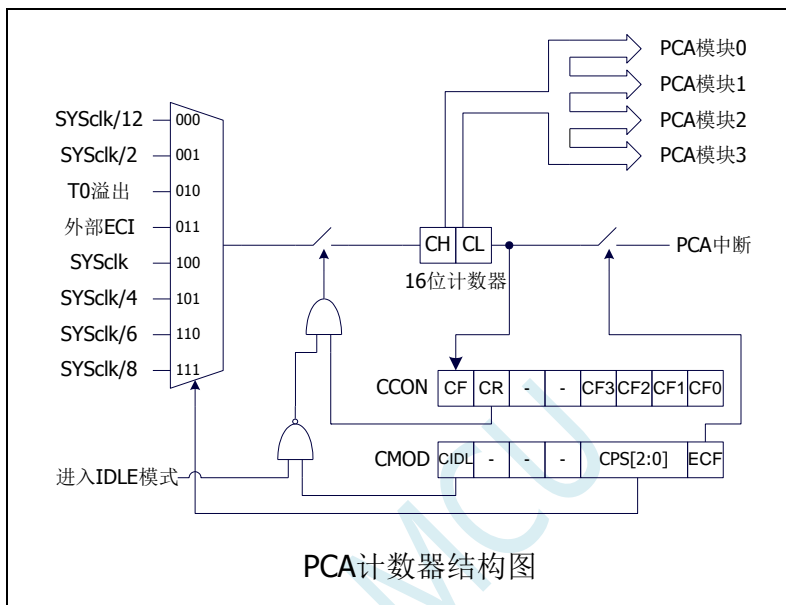
$$(V_{cc} + V_{in}) / 2 < V_{cc} \quad \text{即 } V_{in} < V_{cc}$$

$$\text{上面 2 式可以合起来: } -V_{cc} < V_{in} < V_{cc}$$

## 18 PCA/CCP/PWM 应用

STC8A8K64D4 系列单片机内部集成了 4 组可编程计数器阵列 (PCA/CCP/PWM) 模块, 可用于软件定时器、外部脉冲捕获、高速脉冲输出和 PWM 脉宽调制输出。

PCA 内部含有一个特殊的 16 位计数器, 4 组 PCA 模块均与之相连接。PCA 计数器的结构图如下:



### 18.1 PCA 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	-

CCP\_S[1:0]: PCA 功能脚选择位

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2	CCP3
00	P1.2	P1.7	P1.6	P1.5	P1.4
01	P2.2	P2.3	P2.4	P2.5	P2.6
10	P7.4	P7.0	P7.1	P7.2	P7.3
11	P3.5	P3.3	P3.2	P3.1	P3.0

### 18.2 PCA 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	
CCON	PCA 控制寄存器	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA 模式寄存器	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 模块 0 模式控制寄存器	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 模块 1 模式控制寄存器	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 模块 2 模式控制寄存器	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000

CCAPM3	PCA 模块 3 模式控制寄存器	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CL	PCA 计数器低字节	E9H									0000,0000
CCAP0L	PCA 模块 0 低字节	EAH									0000,0000
CCAP1L	PCA 模块 1 低字节	EBH									0000,0000
CCAP2L	PCA 模块 2 低字节	ECH									0000,0000
CCAP3L	PCA 模块 3 低字节	FD55H									0000,0000
PCA_PWM0	PCA0 的 PWM 模式寄存器	F2H	EBS0[1:0]		XCCAP0H[1:0]	XCCAP0L[1:0]	EPC0H	EPC0L			0000,0000
PCA_PWM1	PCA1 的 PWM 模式寄存器	F3H	EBS1[1:0]		XCCAP1H[1:0]	XCCAP1L[1:0]	EPC1H	EPC1L			0000,0000
PCA_PWM2	PCA2 的 PWM 模式寄存器	F4H	EBS2[1:0]		XCCAP2H[1:0]	XCCAP2L[1:0]	EPC2H	EPC2L			0000,0000
PCA_PWM3	PCA3 的 PWM 模式寄存器	FD57H	EBS3[1:0]		XCCAP3H[1:0]	XCCAP3L[1:0]	EPC3H	EPC3L			0000,0000
CH	PCA 计数器高字节	F9H									0000,0000
CCAP0H	PCA 模块 0 高字节	FAH									0000,0000
CCAP1H	PCA 模块 1 高字节	FBH									0000,0000
CCAP2H	PCA 模块 2 高字节	FCH									0000,0000
CCAP3H	PCA 模块 3 高字节	FD56H									0000,0000

## 18.2.1 PCA 控制寄存器 (CCON)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0

CF: PCA 计数器溢出中断标志。当 PCA 的 16 位计数器计数发生溢出时, 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。

CR: PCA 计数器允许控制位。

0: 停止 PCA 计数

1: 启动 PCA 计数

CCFn (n=0,1,2,3): PCA 模块中断标志。当 PCA 模块发生匹配或者捕获时, 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。

## 18.2.2 PCA 模式寄存器 (CMOD)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-		CPS[2:0]		ECF

CIDL: 空闲模式下是否停止 PCA 计数。

0: 空闲模式下 PCA 继续计数

1: 空闲模式下 PCA 停止计数

CPS[2:0]: PCA 计数脉冲源选择位

CPS[2:0]	PCA 的输入时钟源
000	系统时钟/12
001	系统时钟/2
010	定时器 0 的溢出脉冲
011	ECI 脚的外部输入时钟
100	系统时钟
101	系统时钟/4
110	系统时钟/6

111	系统时钟 8
-----	--------

ECF: PCA 计数器溢出中断允许位。

0: 禁止 PCA 计数器溢出中断

1: 使能 PCA 计数器溢出中断

### 18.2.3 PCA 计数器寄存器 (CL, CH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CL	E9H								
CH	F9H								

由 CL 和 CH 两个字节组合成一个 16 位计数器, CL 为低 8 位计数器, CH 为高 8 位计数器。每个 PCA 时钟 16 位计数器自动加 1。

### 18.2.4 PCA 模块模式控制寄存器 (CCAPMn)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2
CCAPM3	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

ECOMn: 允许 PCA 模块 n 的比较功能

CCAPPn: 允许 PCA 模块 n 进行上升沿捕获

CCAPNn: 允许 PCA 模块 n 进行下降沿捕获

MATn: 允许 PCA 模块 n 的匹配功能

TOGn: 允许 PCA 模块 n 的高速脉冲输出功能

PWMn: 允许 PCA 模块 n 的脉宽调制输出功能

ECCFn: 允许 PCA 模块 n 的匹配/捕获中断

### 18.2.5 PCA 模块模式捕获值/比较值寄存器 (CCAPnL, CCAPnH)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CCAP0L	EAH								
CCAP1L	EBH								
CCAP2L	ECH								
CCAP3L	FD55H								
CCAP0H	FAH								
CCAP1H	FBH								
CCAP2H	FCH								
CCAP3H	FD56H								

当 PCA 模块捕获功能使能时, CCAPnL 和 CCAPnH 用于保存发生捕获时的 PCA 的计数值 (CL 和 CH);

当 PCA 模块比较功能使能时, PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较, 并给出比较结果; 当 PCA 模块匹配功能使能时, PCA 控制器会将当前 CL 和 CH 中的计数值与保存在 CCAPnL 和 CCAPnH 中的值进行比较, 看是否匹配 (相等), 并给出匹配结果。

## 18.2.6 PCA 模块 PWM 模式控制寄存器 (PCA\_PWMn)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L
PCA_PWM1	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L
PCA_PWM2	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L
PCA_PWM3	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L

EBSn[1:0]: PCA 模块 n 的 PWM 位数控制

EBSn[1:0]	PWM 位数	重载值	比较值
00	8 位 PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7 位 PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6 位 PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10 位 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]: 10 位 PWM 的第 9 位和第 10 位的重载值

XCCAPnL[1:0]: 10 位 PWM 的第 9 位和第 10 位的比较值

EPCnH: PWM 模式下, 重载值的最高位 (8 位 PWM 的第 9 位, 7 位 PWM 的第 8 位, 6 位 PWM 的第 7 位, 10 位 PWM 的第 11 位)

EPCnL: PWM 模式下, 比较值的最高位 (8 位 PWM 的第 9 位, 7 位 PWM 的第 8 位, 6 位 PWM 的第 7 位, 10 位 PWM 的第 11 位)

注意: 在更新 10 位 PWM 的重载值时, 必须先写高两位 XCCAPnH[1:0], 再写低 8 位 CCAPnH[7:0]。

## 18.3 PCA 工作模式

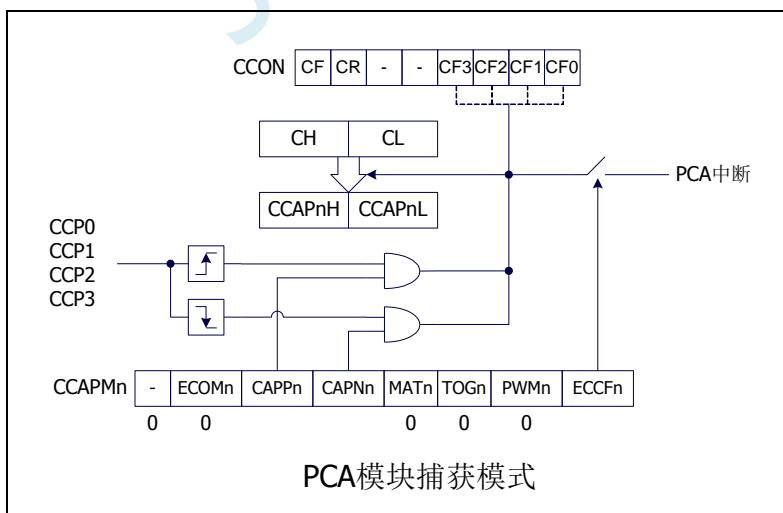
STC12H 系列单片机共有 4 组 PCA 模块，每组模块都可独立设置工作模式。模式设置如下所示：

CCAPMn								模块功能
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	
-	0	0	0	0	0	0	0	无操作
-	1	0	0	0	0	1	0	6/7/8/10 位 PWM 模式，无中断
-	1	1	0	0	0	1	1	6/7/8/10 位 PWM 模式，产生上升沿中断
-	1	0	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生下降沿中断
-	1	1	1	0	0	1	1	6/7/8/10 位 PWM 模式，产生边沿中断
-	0	1	0	0	0	0	x	16 位上升沿捕获
-	0	0	1	0	0	0	x	16 位下降沿捕获
-	0	1	1	0	0	0	x	16 位边沿捕获
-	1	0	0	1	0	0	x	16 位软件定时器
-	1	0	0	1	1	0	x	16 位高速脉冲输出

### 18.3.1 捕获模式

要使一个 PCA 模块工作在捕获模式，寄存器 CCAPMn 中的 CAPNn 和 CAPPn 至少有一位必须置 1（也可两位都置 1）。PCA 模块工作于捕获模式时，对模块的外部 CCP0/CCP1/CCP2 管脚的输入跳变进行采样。当采样到有效跳变时，PCA 控制器立即将 PCA 计数器 CH 和 CL 中的计数值装载到模块的捕获寄存器中 CCAPnH 和 CCAPnL，同时将 CCON 寄存器中相应的 CCFn 置 1。若 CCAPMn 中的 ECCFn 位被设置为 1，将产生中断。由于所有 PCA 模块的中断入口地址是共享的，所以在中断服务程序中需要判断是哪一个模块产生了中断，并注意中断标志位需要软件清零。

PCA 模块工作于捕获模式的结构图如下图所示：

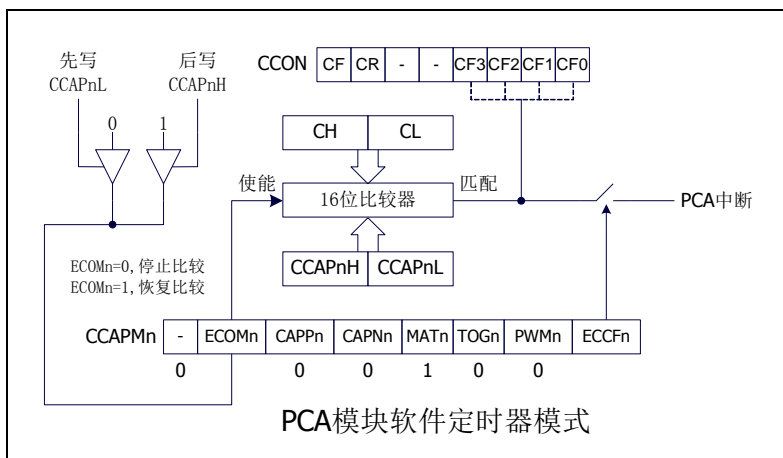


### 18.3.2 软件定时器模式

通过置位 CCAPMn 寄存器的 ECOM 和 MAT 位，可使 PCA 模块用作软件定时器。PCA 计数器值 CL 和 CH 与模块捕获寄存器的值 CCAPnL 和 CCAPnH 相比较，当两者相等时，CCON 中的 CCFn 会被

置 1, 若 CCAPm 中的 ECCFn 被设置为 1 时将产生中断。CCFn 标志位需要软件清零。

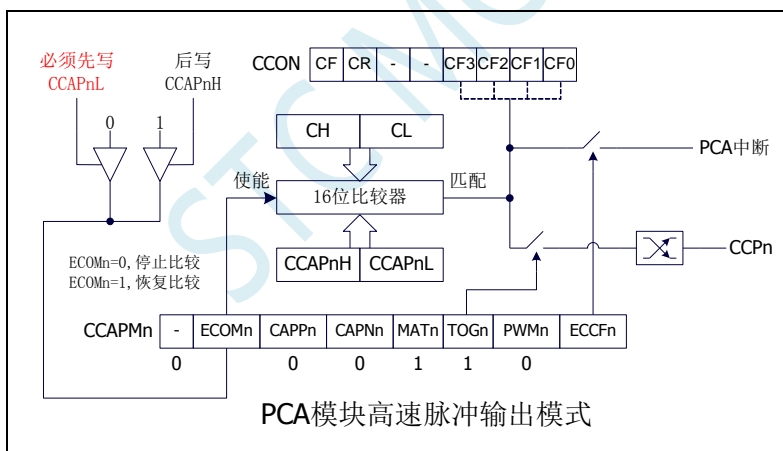
PCA 模块工作于软件定时器模式的结构图如下图所示:



### 18.3.3 高速脉冲输出模式

当 PCA 计数器的计数值与模块捕获寄存器的值相匹配时,PCA 模块的 CCPn 输出将发生翻转。要激活高速脉冲输出模式, CCAPm 寄存器的 TOGn、MATn 和 ECOMn 位必须都置 1。

PCA 模块工作于高速脉冲输出模式的结构图如下图所示:



### 18.3.4 PWM 脉宽调制模式及频率计算公式

#### 18.3.4.1 8 位 PWM 模式

脉宽调制是使用程序来控制波形的占空比、周期、相位波形的一种技术, 在三相电机驱动、D/A 转换等场合有广泛的应用。STC8 系列单片机的 PCA 模块可以通过设定各自的 PCA\_PWMn 寄存器使其工作于 8 位 PWM 或 7 位 PWM 或 6 位 PWM 或 10 位 PWM 模式。要使能 PCA 模块的 PWM 功能, 模块寄存器 CCAPm 的 PWMn 和 ECOMn 位必须置 1。

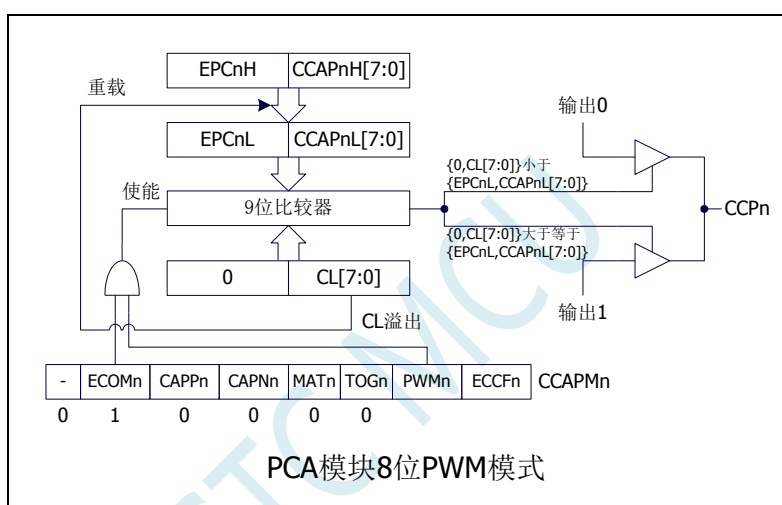
PCA\_PWMn 寄存器中的 EBSn[1:0] 设置为 00 时, PCA 模块 n 工作于 8 位 PWM 模式, 此时将 {0, CL[7:0]} 与捕获寄存器 {EPCnL, CCAPnL[7:0]} 进行比较。当 PCA 模块工作于 8 位 PWM 模式时, 由于所有模块共

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[7:0]}进行设置。当{0,CL[7:0]}的值小于{EPCnL,CCAPnL[7:0]}时,输出为低电平;当{0,CL[7:0]}的值等于或大于{EPCnL,CCAPnL[7:0]}时,输出为高电平。当CL[7:0]的值由FF变为00溢出时,{EPCnH,CCAPnH[7:0]}的内容重新装载到{EPCnL,CCAPnL[7:0]}中。这样就可实现无干扰地更新PWM。

$$\text{8位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{256}$$

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 8 位 PWM 模式的结构图如下图所示:



### 18.3.4.2 7 位 PWM 模式

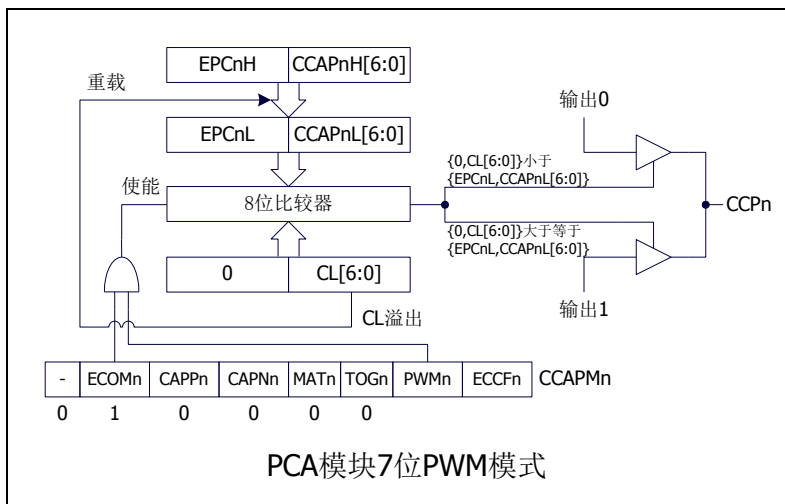
PCA\_PWMn 寄存器中的EBSn[1:0]设置为01时,PCA 模块 n 工作于 7 位 PWM 模式,此时将{0,CL[6:0]}与捕获寄存器{EPCnL,CCAPnL[6:0]}进行比较。当 PCA 模块工作于 7 位 PWM 模式时,由于所有模块共用一个PCA计数器,所有它们的输出频率相同。各个模块的输出占空比使用寄存器{EPCnL,CCAPnL[6:0]}进行设置。当{0,CL[6:0]}的值小于{EPCnL,CCAPnL[6:0]}时,输出为低电平;当{0,CL[6:0]}的值等于或大于{EPCnL,CCAPnL[6:0]}时,输出为高电平。当CL[6:0]的值由7F变为00溢出时,{EPCnH,CCAPnH[6:0]}的内容重新装载到{EPCnL,CCAPnL[6:0]}中。这样就可实现无干扰地更新PWM。

$$\text{7位模式的PWM频率} = \frac{\text{PCA时钟输入源频率}}{128}$$

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 7 位 PWM 模式的结构图如下图所示:





### 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:

