

```
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

14.7.4 UART2 using T2 as baud rate generator

C language code

//Operating frequency for test is 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;
    }
}

```

Assembly code

;Operating frequency for test is 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
        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
```

14.7.5 UART3 using T2 as baud rate generator

C language code

//Operating frequency for test is 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;
    }
}

```

Assembly code

;Operating frequency for test is 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
        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

```

14.7.6 UART3 using T3 as baud rate generator

C language code

//Operating frequency for test is 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;
    }
}

```

Assembly code

;Operating frequency for test is 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
MOV      T3H,#0FFH
MOV      T4T3M,#0AH
CLR      BUSY
MOV      WPTR,#00H
MOV      RPTR,#00H
RET

```

;65536-11059200/115200/4=0FFE8H

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

14.7.7 UART4 using T2 as baud rate generator

C language code

//Operating frequency for test is 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;
    }
}

```

Assembly code

;Operating frequency for test is 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
        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
```

14.7.8 UART4 using T4 as baud rate generator

C language code

//Operating frequency for test is 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;
    }
}

```

Assembly code

;Operating frequency for test is 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
        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

14.7.9 Serial multi-MCUs communication

Now refer to the STC15 series data sheet, which will be supplemented later.

14.7.10 UART to LIN bus

C language code

// Operating frequency for test is 22.1184MHz

/****** Function Description *****/

This routine is based on the experiment box 8 to program and test, whose main control chip is STC8H8K64U.

It can be used for general reference when using STC8G and STC8H series chips.

Connect the LIN transceiver through the UART interface to realize the LIN bus signal transceiver test routine. UART1 is connected to the computer through the serial port tool.

UART2 is connected to an external LIN transceiver (TJA1020/1) and connected to the LIN bus.

Forward the data sent by the computer serial port to the LIN bus; forward the data received from the LIN bus to the computer serial port.

Default transmission rate: 9600 baud rate, switch baud rate before sending LIN data, send 13 dominant interval signals.

When downloading, select the clock 22.1184MHz (users can modify the frequency by themselves).

*****/

```
#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

```

```

/***** user-defined macro *****/

```

```

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

```

```

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

```

```

#define    Baudrate_Break    (65536UL - (MAIN_Fosc / 4) / 6647UL)    // Baud Rate when Transmitting Dominant
Interval Signal

```

```

#define    UART1_BUF_LENGTH    32
#define    UART2_BUF_LENGTH    32

#define    LIN_ID                0x31

u8 TX1_Cnt;                      // count of sendding
u8 RX1_Cnt;                      //count of recieving
u8 TX2_Cnt;                      // count of sendding
u8 RX2_Cnt;                      // count of recieving
bit B_TX1_Busy;                 // busy flag of sendding
bit B_TX2_Busy;                 // busy flag of sendding
u8 RX1_TimeOut;
u8 RX2_TimeOut;

u8 xdata RX1_Buffer[UART1_BUF_LENGTH];    //buffer if recieving
u8 xdata RX2_Buffer[UART2_BUF_LENGTH];    // buffer if recieving

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);

//=====
// function: void main(void)
// description: main function
// parameters: none.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//=====
void main(void)
{
    u8 i;

    P0M1 = 0; P0M0 = 0;          //set as quasi-bidirectional port
    P1M1 = 0; P1M0 = 0;          //set as quasi-bidirectional port
    P2M1 = 0; P2M0 = 0;          //set as quasi-bidirectional port
    P3M1 = 0; P3M0 = 0;          //set as quasi-bidirectional port
    P4M1 = 0; P4M0 = 0;          //set as quasi-bidirectional port
    P5M1 = 0; P5M0 = 0;          //set as quasi-bidirectional port
    P6M1 = 0; P6M0 = 0;          //set as quasi-bidirectional port
    P7M1 = 0; P7M0 = 0;          //set as quasi-bidirectional port

    UART1_config(1);
    UART2_config(2);
    EA = 1;                      // Enable global interrupt
    SLP_N = 1;

    PrintString1("STC8H8K64U UART1 Test Programme!\r\n"); //UART1 sends a string

    while (1)
    {
        delay_ms(1);
    }
}

```



```

    if(RX1_TimeOut > 0)
    {
        if(--RX1_TimeOut == 0) // If it times out, the serial port reception ends
        {
            if(RX1_Cnt > 0)
            {
                Lin_Send(RX1_Buffer); // Send the data received by UART1 to the LIN bus
            }
            RX1_Cnt = 0;
        }
    }

    if(RX2_TimeOut > 0)
    {
        if(--RX2_TimeOut == 0) // If it times out, the serial port reception ends
        {
            if(RX2_Cnt > 0)
            {
                for (i=0; i < RX2_Cnt; i++) // End with stop 0
                {
                    UART1_TxByte(RX2_Buffer[i]); // Send data received from LIN bus to UART1
                }
            }
            RX2_Cnt = 0;
        }
    }
}

```

```

//=====
// function: void delay_ms(unsigned char ms)
// description: delay function
// parameters: ms, number of ms to delay, only support 1~255ms, and automatically adapt to master clock.
// return: none.
// version: VER1.0
// date: 2013-4-1
// remark:
//=====

```

```

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

```

```

//=====
// function: u8 Lin_CheckPID(u8 id)
// description: The ID code plus the check character is converted into a PID code.
// parameters: IDcode.
// return: PIDcode.
// version: VER1.0
// date: 2020-12-2
// remark:
//=====

```

```

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 ;
}

//=====================================================
// function: u8 LINCalcChecksum(u8 *dat)
// description: Calculate the checksum.
// parameters: The data transmitted by the data field.
// return: checksum.
// version: VER1.0
// date: 2020-12-2
// remark:
//=====================================================
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;
}

//=====================================================
// function: void Lin_SendBreak(void)
// description: Send a dominant interval signal.
// parameters: none.
// return: none.
// version: VER1.0
// date: 2020-12-2
// remark:
//=====================================================
void Lin_SendBreak(void)
{
    SetTimer2Baudraye(Baudrate_Break);
    UART2_TxByte(0);
    SetTimer2Baudraye(Baudrate2);
}

//=====================================================
// function: void Lin_Send(u8 *puts)
// description: Send LIN bus message.
// parameters: The content of the data field to be sent.
// return: none.
// version: VER1.0

```

```
// date: 2020-12-2
```

```
// remark:
```

```
//=====
```

```
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));
```

```
}
```

```
//=====
```

```
// function: void UART1_TxByte(u8 dat)
```

```
// description: Send a byte.
```

```
// parameters: none.
```

```
// return: none..
```

```
// version: V1.0, 2014-6-30
```

```
//=====
```

```
void UART1_TxByte(u8 dat)
```

```
{
```

```
    SBUF = dat;
```

```
    B_TX1_Busy = 1;
```

```
    while(B_TX1_Busy);
```

```
}
```

```
//=====
```

```
// function: void UART2_TxByte(u8 dat)
```

```
// description: Send a byte.
```

```
// parameters: none.
```

```
// return: none.
```

```
// version: V1.0, 2014-6-30
```

```
//=====
```

```
void UART2_TxByte(u8 dat)
```

```
{
```

```
    S2BUF = dat;
```

```
    B_TX2_Busy = 1;
```

```
    while(B_TX2_Busy);
```

```
}
```

```
//=====
```

```
// function: void PrintString1(u8 *puts)
```

```
// description: UART1 sends a string function
```

```
// parameters: puts:      String pointer.
```

```
// return: none.
```

```
// version: VER1.0
```

```
// date: 2014-11-28
```

```
// remark:
```

```
//=====
```

```
void PrintString1(u8 *puts)
```

```
{
```

```
    for (; *puts != 0; puts++)
```

```
    // End with stop 0
```

```
    {
```

```
        SBUF = *puts;
```

```

        B_TX1_Busy = 1;
        while(B_TX1_Busy);
    }
}

```

```

//=====================================================
// function: void PrintString2(u8 *puts)
// description: UART1 sends a string function
// parameters: puts:      String pointer.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//=====================================================

```

```

//void PrintString2(u8 *puts)
//{
//    for (; *puts != 0; puts++)                //End with stop 0
//    {
//        S2BUF = *puts;
//        B_TX2_Busy = 1;
//        while(B_TX2_Busy);
//    }
//}

```

```

//=====================================================
// function: SetTimer2Baudraye(u16 dat)
// description: Set Timer2 as baud rate generator.
// parameters: dat: Reload value of Timer2
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//=====================================================

```

```

void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                //Timer stop
    AUXR &= ~(1<<3);                //Timer2 set As Timer
    AUXR |= (1<<2);                  //Timer2 set as 1T mode
    TH2 = dat / 256;
    TL2 = dat % 256;
    IE2 &= ~(1<<2);                //Disable interrupt
    AUXR |= (1<<4);                //Timer run enable
}

```

```

//=====================================================
// function: void UART1_config(u8 brt)
// description: UART1 initialization function
// parameters: brt: baud rate selected, 2: select Timer2 as baud rate generator, other values: select Timer1 as baud rate generator
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//=====================================================

```

```

void UART1_config(u8 brt)
{
    /***** select Timer2 as baud rate generator *****/
    if(brt == 2)
    {

```

```

    AUXR |= 0x01;                                //S1 BRT Use Timer2;
    SetTimer2Baudrate(Baudrate1);
}

/*****select Timer1 as baud rate generator *****/
else
{
    TR1 = 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;                                        //diable interrupt
    INT_CLKO &= ~0x02;                            //does not output clock
    TR1 = 1;
}
/*****/

SCON = (SCON & 0x3f) | 0x40;                    //UART1 mode: 0x00: Synchronous shift output,
//                                                // 0x40: 8-bit data, variable baud rate,
//                                                // 0x80: 9-bit data, fixed baud rate,
//                                                // 0xc0: 9-bit data, variable baud rate
// PS = 1;                                        //High priority
// ES = 1;                                        //enable interrupt
// REN = 1;                                       //enable recieving
// 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;
}

//=====
// function: void UART2_config(u8 brt)
// description: UART2 initialization function
// parameters: brt: baud rate selected, 2: select Timer2 as baud rate generator, other values: not valid.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//=====
void UART2_config(u8 brt)
{
    if(brt == 2)
    {
        SetTimer2Baudrate(Baudrate2);

        S2CON &= ~(1<<7);                        //8-bit data, 1 start bit, 1 stop bit, no checking
        IE2 |= 1;                                //enable interrupt
        S2CON |= (1<<4);                          //enable recieving
        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;
    }
}

//=====================================================
// function: void UART1_int(void) interrupt UART1_VECTOR
// description: UART1 interrupt function。
// parameters: none.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//=====================================================
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;
    }
}

//=====================================================
// function: void UART2_int(void) interrupt UART2_VECTOR
// description: UART2 interrupt function
// parameters: none.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//=====================================================
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;
    }
}
```

STC MCU

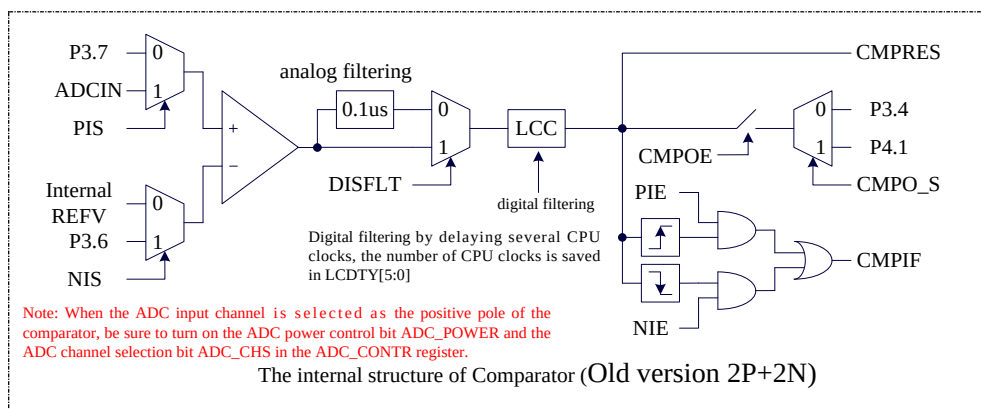
15 Comparator, Power-down Detection, Internal Reference Voltage

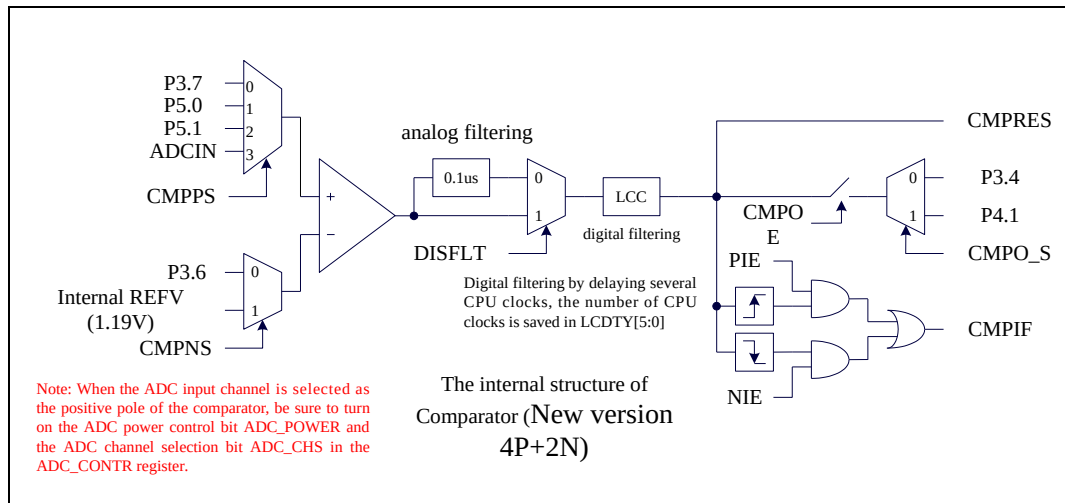
Product line	Comparator	
	Old version(2P+2N)	New version(4P+2N)
STC8H1K08 family	●	
STC8H1K28 family	●	
STC8H3K64S4 family	●	
STC8H3K64S2 family	●	
STC8H8K64U family A version	●	
STC8H8K64U family version		●
STC8H2K64T family	●	
STC8H4K64TLR family		●
STC8H4K64TLCD family		●
STC8H4K64LCD family		●

A comparator is integrated in STC8H series of microcontrollers. The positive terminal of the comparator can be P3.7 or ADC analog input (The positive pole of the new version of the comparator can be the P3.7 port, the P5.0 port, the P5.1 port or the analog input channel of the ADC), and the negative can be P3.6 or the REFV voltage of the internal BandGap after amplified. The application of multiple comparators can be realized through multiplexer and time division multiplexing.

There are two stage programmable filterings inside the comparator: analog filtering and digital filtering. Analog filtering can filter out glitches in the input signal, and digital filtering can wait for the input signal to stabilize before making a comparison. The result of the comparison can be obtained directly by reading the internal register bits or output the result of the comparator forward or reverse to the external port. Outputting the comparison result to the external port can be used as the trigger signal of external events and the feedback signal to expand the scope of application.

15.1 Internal Structure of Comparator





15.2 Registers Related to Comparator

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	Comparator Control Register 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	Comparator Control Register 2	E7H	INVCMP0	DISFLT	LCDTY[5:0]						0000,0000
CMPEXCFG	Comparator Extended Configuration Register	FEAEH	CHYS[1:0]		-	-	-	CMPNS	CMPPS[1:0]		00xx,x000

15.2.1 Comparator control register 1

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN: Comparator enable bit

- 0: disable comparator
- 1: enable comparator

CMPIF: Comparator interrupt flag. When PIE or NIE is enabled, if the corresponding interrupt signal is generated, the hardware will automatically set CMPIF and request interrupt to CPU. This flag must be cleared by software. **(Note: When the comparator interrupt is not enabled, this interrupt flag will not be set by the hardware, that is, this interrupt flag cannot be queried when the comparator is accessed in query mode.)**

PIE: Comparator rising edge interrupt enable bit

- 0: disable comparator rising edge interrupt
- 1: enable comparator rising edge interrupt. Enable the interrupt request when the compare result of the comparator changes from 0 to 1.

NIE: Comparator falling edge interrupt enable bit

- 0: disable comparator falling edge interrupt
- 1: enable comparator falling edge interrupt. Enable the interrupt request when the compare result of the comparator changes from 1 to 0.

PIS: Positive of comparator selection bit

- 0: Select P3.7 as the comparator positive input source.
- 1: The analog input of the ADC selected by the `ADC_CHS` bits in `ADC_CONTR` is selected as the comparator positive input source.

(Note 1: When the ADC input channel is selected as the positive of the comparator, please make sure to turn on the ADC power control bit `ADC_POWER` and ADC channel selection bit `ADC_CHS` in the `ADC_CONTR` register)

(Note 2: When the comparator interrupt needs to be used to wake up the CPU in power-down mode/clock stop mode, we must select P3.7 as the positive of the comparator, and the ADC input channel cannot be used)

NIS: Negative of comparator selection bit

- 0: The REFV voltage of the internal BandGap after amplified is selected as the comparator negative input

source. (When the chip is shipped, the internal reference voltage is adjusted to 1.19V)

1: Select P3.6 as the comparator negative input source.

CMPOE: Comparator result output control bit

0: disable comparator result output.

1: enable comparator result output. The comparator result is output to P3.4 or P4.1, which is selected by CMPO_S in P_SW2.

COMPRES: Flag bit of comparator result. (Read-only)

0: the level of CMP+ is lower than CMP-.

1: the level of CMP+ is higher than CMP-.

COMPRES is the digitally filtered output signal, not the comparator's direct output.

15.2.2 Comparator control register 2

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

INVCMP0: Inverse comparator output control bit

0: Normal output the result of comparator. If COMPRES is 0, P3.4 / P4.1 output low, and vice versa output high.

1: Output the result of comparator after it is inversed. If COMPRES is 0, P3.4 / P4.1 output high, and vice versa output low.

DISFLT: Analog filtering function control bit

0: enable 0.1us analog filtering function

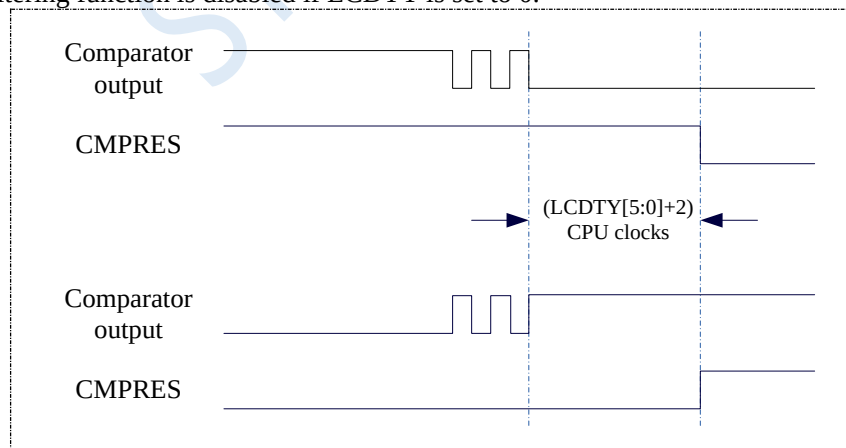
1: disable 0.1us analog filtering function, which can speed up the comparator slightly.

LCDTY[5:0]: Digital filtering function control bit

Digital filtering is the debouncing function of the digital signal. When the comparison result changes at the rising edge or falling edge, the data changing is considered be valid only if the signal the comparator detected does not change and maintains the number of CPU clocks set in LCDTY. Otherwise, the signal will be treated as no change.

Note: When the digital filtering function is enabled, the actual waiting clock inside the chip needs to add two additional state machine switching times, that is, if LCDTY is set to 0, the digital filtering function is turned off; if LCDTY is set to a non-zero value n (n = 1~63), the actual digital filtering time is (n+2) system clocks

The digital filtering function is disabled if LCDTY is set to 0.



15.2.3 Comparator Extended Configuration Register (CMPEXCFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CMPEXCFG	FEA7H	CHYS[1:0]		-	-	-	CMPPNS	CMPPS[1:0]	

CHYS[1:0]: Comparator DC Hysteresis Input Selection

CHYS [1:0]	Comparator DC Hysteresis Input Selection
00	0mV
01	10mV

10	20mV
11	30mV

CMPNS: Comparator negative input selection

0: P3.6

1: The REFV voltage of the internal BandGap after amplified is selected as the comparator negative input source. (When the chip is shipped, the internal reference voltage is adjusted to 1.19V)

CMPPS[1:0]: Positive of comparator selection bit

CMPPS[1:0]	Positive of comparator
00	P3.7
01	P5.0
10	P5.1
11	ADCIN

STC MCU

15.3 Example Routines

15.3.1 Using Old Version Comparator (Interrupt Mode)

C language code

//Operating frequency for test is 11.0592MHz

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

```
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;           //Clear interrupt flag
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;           //Falling edge interrupt test port
    }
    else
    {
        P11 = !P11;           //Rising edge interrupt test port
    }
}
```

```
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;
}
```

```

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;
//    CMPCR2 |= 0x80;
    CMPCR2 &= ~0x40;
//    CMPCR2 |= 0x40;
//    CMPCR2 &= ~0x3f;
    CMPCR2 = 0x10;
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;
//    CMPCR1 &= ~0x20;
//    CMPCR1 |= 0x20;
//    CMPCR1 &= ~0x10;
//    CMPCR1 |= 0x10;
    CMPCR1 &= ~0x08;
//    CMPCR1 |= 0x08;
//    CMPCR1 &= ~0x04;
    CMPCR1 |= 0x04;
//    CMPCR1 &= ~0x02;
    CMPCR1 = 0x02;
    CMPCR1 = 0x80;

    EA = 1;

    while (1);
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

```

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

                ORG    0100H
CMPISR:
    PUSH    ACC
    ANL     CMPCR1,#NOT 40H    ;Clear interrupt flag
    MOV     A,CMPCR1
    JB      ACC.0,RSING

FALLING:
    CPL     P1.0
    POP     ACC
                ;Falling edge interrupt test port

```

```

RSING:
    RETI
    CPL      P1.1      ;Rising edge interrupt test port
    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      CMPCR2, #00H
    ANL      CMPCR2, #NOT 80H      ;Comparator forward output
    ; ORL     CMPCR2, #80H      ;Comparator inverted output
    ANL      CMPCR2, #NOT 40H      ;Disable 0.1us filtering
    ; ORL     CMPCR2, #40H      ;Enable 0.1us filtering
    ; ANL     CMPCR2, #NOT 3FH    ;Output comparator result directly
    ORL      CMPCR2, #10H      ;Output comparator result after 16 debounce clocks
    MOV      CMPCR1, #00H
    ORL      CMPCR1, #30H      ;Enable edge interrupt of comparator
    ; ANL     CMPCR1, #NOT 20H    ;Disable comparator rising edge interrupt
    ; ORL     CMPCR1, #20H      ;Enable comparator rising edge interrupt
    ; ANL     CMPCR1, #NOT 10H    ;Disable comparator falling edge interrupt
    ; ORL     CMPCR1, #10H      ;Enable comparator falling edge interrupt
    ANL      CMPCR1, #NOT 08H    ;P3.7 is CMP+ input pin
    ; ORL     CMPCR1, #08H      ;ADC input pin is CMP+ input pin
    ; ANL     CMPCR1, #NOT 04H    ;Internal reference voltage is CMP- input pin
    ORL      CMPCR1, #04H      ;P3.6 is CMP- input pin
    ; ANL     CMPCR1, #NOT 02H    ;Disable comparator output
    ORL      CMPCR1, #02H      ;Enable Comparator output
    ORL      CMPCR1, #80H      ;Enable comparator module
    SETB     EA

    JMP      $

    END

```

15.3.2 Using Old Version Comparator (Polling Mode)

C language code

//Operating frequency for test is 11.0592MHz

```

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

```

```

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 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;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;           //Comparator forward output
    // CMPCR2 |= 0x80;         //Comparator inverted output
    CMPCR2 &= ~0x40;           //Disable 0.1us filtering
    // CMPCR2 |= 0x40;         //Enable 0.1us filtering
    // CMPCR2 &= ~0x3f;       //Output comparator result directly
    CMPCR2 |= 0x10;           //Output comparator result after 16 debounce clocks
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;           //Enable edge interrupt of comparator
    // CMPCR1 &= ~0x20;       //Disable comparator rising edge interrupt
    // CMPCR1 |= 0x20;         //Enable comparator rising edge interrupt
    // CMPCR1 &= ~0x10;       //Disable comparator falling edge interrupt
    // CMPCR1 |= 0x10;         //Enable comparator falling edge interrupt
    CMPCR1 &= ~0x08;         //P3.7 is CMP+ input pin
    // CMPCR1 |= 0x08;         //ADC input pin is CMP+ input pin
    // CMPCR1 &= ~0x04;       //Internal reference voltage is CMP- input pin
    CMPCR1 |= 0x04;           //P3.6 is CMP- input pin
    // CMPCR1 &= ~0x02;       //Disable comparator output
    CMPCR1 |= 0x02;           //Enable Comparator output
    CMPCR1 |= 0x80;           //Enable comparator module

    while (1)
    {
        P10 = CMPCR1 & 0x01; //Read comparator comparison result
    }
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

```

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

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     CMPCR2, #00H
                ANL     CMPCR2, #NOT 80H    ;Comparator forward output
;                ORL     CMPCR2, #80H        ;Comparator inverted output
                ANL     CMPCR2, #NOT 40H    ;Disable 0.1us filtering
;                ORL     CMPCR2, #40H        ;Enable 0.1us filtering
;                ANL     CMPCR2, #NOT 3FH    ;Output comparator result directly
                ORL     CMPCR2, #10H        ;Output comparator result after 16 debounce clocks
                MOV     CMPCR1, #00H
                ORL     CMPCR1, #30H        ;Enable edge interrupt of comparator
;                ANL     CMPCR1, #NOT 20H    ;Disable comparator rising edge interrupt
;                ORL     CMPCR1, #20H        ;Enable comparator rising edge interrupt
;                ANL     CMPCR1, #NOT 10H    ;Disable comparator falling edge interrupt
;                ORL     CMPCR1, #10H        ;Enable comparator falling edge interrupt
                ANL     CMPCR1, #NOT 08H    ;P3.7 is CMP+ input pin
;                ORL     CMPCR1, #08H        ;ADC input pin is CMP+ input pin
;                ANL     CMPCR1, #NOT 04H    ;Internal reference voltage is CMP- input pin
                ORL     CMPCR1, #04H        ;P3.6 is CMP- input pin
;                ANL     CMPCR1, #NOT 02H    ;Disable comparator output
                ORL     CMPCR1, #02H        ;Enable Comparator output
                ORL     CMPCR1, #80H        ;Enable comparator module

```


LOOP:

```

MOV    A,CMPCR1
MOV    C,ACC.0
MOV    P1.0,C           ;Read comparator comparison result
JMP    LOOP

```

END

15.3.3 Using New Version Comparator (Interrupt Mode)

C language code

// Operating frequency for test is 11.0592MHz

```

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

```

```

sfr     P_SW2      = 0xba;
sfr     CMPCR1     = 0xe6;
sfr     CMPCR2     = 0xe7;

```

```

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;
sbit    P11        = P1^1;

```

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

```
void CMP_Isr() interrupt 21
```

```

{
    CMPCR1 &= ~0x40;           // Clear interrupt flag
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;           // Rising edge interrupt test port
    }
    else
    {
        P11 = !P11;           // Falling edge interrupt test port
    }
}

```

```
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; // Enable XFR access
CMPEXCFG = 0x00; // Comparator DC Hysteresis Input Selection
// CMPEXCFG |= 0x40; // 0:0mV; 0x40:10mV; 0x80:20mV; 0xc0:30mV

CMPEXCFG &= ~0x03; // P3.7 is CMP+ input pin
// CMPEXCFG |= 0x01; // P5.0 is CMP+ input pin
// CMPEXCFG |= 0x02; // P5.1 is CMP+ input pin
// CMPEXCFG |= 0x03; // ADC input pin is CMP+ input pin
CMPEXCFG &= ~0x04; // P3.6 is CMP- input pin
// CMPEXCFG |= 0x04; // The internal 1.19V reference voltage is the CMP- input
pin

P_SW2 &= 0x7f; // Disable XFR access

CMPCR2 = 0x00;
CMPCR2 &= ~0x80; // Comparator forward output
// CMPCR2 |= 0x80; // Comparator inverted output
CMPCR2 &= ~0x40; // Enable 0.1us filtering
// CMPCR2 |= 0x40; // Disable 0.1us filtering
// CMPCR2 &= ~0x3f; // Output comparator result directly
CMPCR2 |= 0x10; // Output comparator result after 16 debounce clocks
CMPCR1 = 0x00;
CMPCR1 |= 0x30; // Enable edge interrupt of comparator
// CMPCR1 &= ~0x20; // Disable comparator rising edge interrupt
// CMPCR1 |= 0x20; // Enable comparator rising edge interrupt
// CMPCR1 &= ~0x10; // Disable comparator falling edge interrupt
// CMPCR1 |= 0x10; // Enable comparator falling edge interrupt
// CMPCR1 &= ~0x02; // Disable comparator output
CMPCR1 |= 0x02; // Enable Comparator output
CMPCR1 |= 0x80; // Enable comparator module

EA = 1;

while (1);
}

```

Assembly code

; Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH
CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H
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

CMPEXCFG  XDATA      0FEAEH

          ORG          0000H
          LJMP         MAIN
          ORG          00ABH
          LJMP         CMPISR

          ORG          0100H
CMPISR:   PUSH        ACC
          ANL          CMPCR1,#NOT 40H      ; Clear interrupt flag
          MOV          A,CMPCR1
          JB           ACC.0,RSING

FALLING:  CPL          P1.0                ; Falling edge interrupt test port
          POP          ACC
          RETI

RSING:    CPL          P1.1                ; Rising edge interrupt test port
          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_SW2,#80H
          MOV          DPTR,# CMPEXCFG
          CLR          A
          ANL          A,#NOT 03H          ; P3.7 is CMP+ input pin
;          ORL          A,#01H             ; P5.0 is CMP+ input pin
;          ORL          A,#02H             ; P5.1 is CMP+ input pin
;          ORL          A,#03H             ; ADC input pin is CMP+ input pin
          ANL          A,#NOT 04H          ; P3.6 is CMP- input pin
;          ORL          A,# 04H             ; Internal reference voltage is CMP- input pin
          MOVX         @DPTR,A
          MOV          P_SW2,#00H

          MOV          CMPCR2,#00H
          ANL          CMPCR2,#NOT 80H     ; Comparator forward output
;          ORL          CMPCR2,#80H        ; Comparator inverted output

```

```

        ANL          CMPCR2,#NOT 40H          ; Enable 0.1us filtering
;        ORL          CMPCR2,#40H             ; Disable 0.1us filtering
;        ANL          CMPCR2,#NOT 3FH         ; Output comparator result directly
        ORL          CMPCR2,#10H              ; Output comparator result after 16 debounce clocks
        MOV          CMPCR1,#00H
        ORL          CMPCR1,#30H              ; Enable edge interrupt of comparator
;        ANL          CMPCR1,#NOT 20H         ; Disable comparator rising edge interrupt
;        ORL          CMPCR1,#20H             ; Enable comparator rising edge interrupt
;        ANL          CMPCR1,#NOT 10H         ; Disable comparator falling edge interrupt
;        ORL          CMPCR1,#10H             ; Enable comparator falling edge interrupt
;        ANL          CMPCR1,#NOT 02H         ; Disable comparator output
        ORL          CMPCR1,#02H              ; Enable Comparator output
        ORL          CMPCR1,#80H              ; Enable comparator module
        SETB         EA

        JMP          $

END

```

15.3.4 Using New Version Comparator (Polling Mode)

C language code

// Operating frequency for test is 11.0592MHz

```

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

sfr      P_SW2      = 0xba;
sfr      CMPCR1     = 0xe6;
sfr      CMPCR2     = 0xe7;

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;
sbit     P11        = P1^1;

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

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; // Enable XFR access
CMPEXCFG = 0x00;
// CMPEXCFG |= 0x40; // Comparator DC Hysteresis Input Selection
// //0:0mV; 0x40:10mV; 0x80:20mV; 0xc0:30mV

CMPEXCFG &= ~0x03; //P3.7 is CMP+ input pin
// CMPEXCFG |= 0x01; //P5.0 is CMP+ input pin
// CMPEXCFG |= 0x02; //P5.1 is CMP+ input pin
// CMPEXCFG |= 0x03; // ADC input pin is CMP+ input pin
CMPEXCFG &= ~0x04; //P3.6 is CMP- input pin
// CMPEXCFG |= 0x04; // Internal reference voltage is CMP- input pin
P_SW2 &= 0x7f; // Disable XFR access

CMPCR2 = 0x00;
CMPCR2 &= ~0x80; // Comparator forward output
// CMPCR2 |= 0x80; // Comparator inverted output
CMPCR2 &= ~0x40; // Enable 0.1us filtering
// CMPCR2 |= 0x40; // Disable 0.1us filtering
// CMPCR2 &= ~0x3f; // Output comparator result directly
CMPCR2 |= 0x10; // Output comparator result after 16 debounce clocks
CMPCR1 = 0x00;
CMPCR1 |= 0x30; // Enable edge interrupt of comparator
// CMPCR1 &= ~0x20; // Disable comparator rising edge interrupt
// CMPCR1 |= 0x20; // Enable comparator rising edge interrupt
// CMPCR1 &= ~0x10; // Disable comparator falling edge interrupt
// CMPCR1 |= 0x10; // Enable comparator falling edge interrupt
// CMPCR1 &= ~0x02; // Disable comparator output
CMPCR1 |= 0x02; // Enable Comparator output
CMPCR1 |= 0x80; // Enable comparator module

while (1)
{
    P10 = CMPCR1 & 0x01; // Read comparator comparison result
}

```

Assembly code

; Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH
CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H
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

CMPEXCFG  XDATA      0FEAEH

          ORG          0000H
          LJMP         MAIN

MAIN:      ORG          0100H

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

          MOV          P_SW2, #80H
          MOV          DPTR, # CMPEXCFG
          CLR          A
          ANL          A, #NOT 03H      ; P3.7 is CMP+ input pin
;          ORL          A, #01H          ; P5.0 is CMP+ input pin
;          ORL          A, #02H          ; P5.1 is CMP+ input pin
;          ORL          A, #03H          ; ADC input pin is CMP+ input pin
          ANL          A, #NOT 04H      ; P3.6 is CMP- input pin
;          ORL          A, # 04H          ; Internal reference voltage is CMP- input pin
          MOVX         @DPTR, A
          MOV          P_SW2, #00H

          MOV          CMPCR2, #00H
          ANL          CMPCR2, #NOT 80H ; Comparator forward output
;          ORL          CMPCR2, #80H      ; Comparator inverted output
          ANL          CMPCR2, #NOT 40H ; Enable 0.1us filtering
;          ORL          CMPCR2, #40H      ; Disable 0.1us filtering
;          ANL          CMPCR2, #NOT 3FH   ; Output comparator result directly
          ORL          CMPCR2, #10H      ; Output comparator result after 16 debounce clocks
          MOV          CMPCR1, #00H
          ORL          CMPCR1, #30H      ; Enable edge interrupt of comparator
;          ANL          CMPCR1, #NOT 20H   ; Disable comparator rising edge interrupt
;          ORL          CMPCR1, #20H      ; Enable comparator rising edge interrupt
;          ANL          CMPCR1, #NOT 10H   ; Disable comparator falling edge interrupt
;          ORL          CMPCR1, #10H      ; Enable comparator falling edge interrupt
;          ANL          CMPCR1, #NOT 02H   ; Disable comparator output
          ORL          CMPCR1, #02H      ; Enable Comparator output
          ORL          CMPCR1, #80H      ; Enable comparator module

LOOP:      MOV          A, CMPCR1
          MOV          C, ACC.0
          MOV          P1.0, C          ; Read comparator comparison result

```

JMP LOOP

END

15.3.5 Multiplexing Application of Old Version Comparator (Comparator+ADC input channel)

Since the analog input channel of the ADC can be selected as the positive of the comparator, the application of multiple comparators can be realized through the multiplexer and time-division multiplexing.

Note: When the ADC input channel is selected as the positive of the comparator, please make sure to turn on the ADC power control bit **ADC_POWER** and the ADC channel selection bit **ADC_CHS** in the **ADC_CONTR** register.

C language code

// Operating frequency for test is 11.0592MHz

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

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

```
sfr      ADC_CONTR = 0xbc;
```

```
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;
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;
```

```

P1M0 &= 0xfe;           //Set P1.0 to input mode
P1M1 |= 0x01;
ADC_CONTR = 0x80;       // Enable ADC module and select P1.0 as ADC input pin

CMPCR2 = 0x00;
CMPCR1 = 0x00;

CMPCR1 |= 0x08;         // Select ADC input pin as CMP+ input pin
CMPCR1 |= 0x04;         //P3.6 is CMP- input pin
CMPCR1 |= 0x02;         // Enable Comparator output
CMPCR1 |= 0x80;         // Enable comparator module

while (1);
}

```

Assembly code

; Operating frequency for test is 11.0592MHz

```

CMPCR1    DATA    0E6H
CMPCR2    DATA    0E7H
ADC_CONTR DATA    0BCH

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

    ANL        P1M0, #0FEH    ; Set P1.0 to input mode
    ORL        P1M1, #01H
    MOV        ADC_CONTR, #80H    ; Enable ADC module and select P1.0 as ADC input pin

    MOV        CMPCR2, #00H

```



```

MOV      CMPCR1,#00H

ORL      CMPCR1,#08H      ; Select ADC input pin as CMP+ input pin
ORL      CMPCR1,#04H      ; P3.6 is CMP- input pin
ORL      CMPCR1,#02H      ; Enable Comparator output
ORL      CMPCR1,#80H      ; Enable comparator module

LOOP:

JMP      LOOP

END

```

15.3.6 Multiplexing Application of New Version Comparator (Comparator+ADC input channel)

Since the analog input channel of the ADC can be selected as the positive of the comparator, the application of multiple comparators can be realized through the multiplexer and time-division multiplexing.

Note: When the ADC input channel is selected as the positive of the comparator, please make sure to turn on the ADC power control bit **ADC_POWER** and the ADC channel selection bit **ADC_CHS** in the **ADC_CONTR** register.

C language code

// Operating frequency for test is 11.0592MHz

```

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

sfr      P_SW2      = 0xba;
sfr      CMPCR1     = 0xe6;
sfr      CMPCR2     = 0xe7;

sfr      ADC_CONTR  = 0xbc;

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;
sbit      P11       = P1^1;

#define    CMPEXC_CFG (*(unsigned char volatile xdata *)0xfeae)

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 &= 0xfe;           // Set P1.0 to input mode
P1M1 |= 0x01;
ADC_CONTR = 0x80;       // Enable ADC module and select P1.0 as ADC input pin

P_SW2 |= 0x80;          // Enable XFR access
CMPEXCFG = 0x00;
// CMPEXCFG &= ~0x03;    //P3.7 is CMP+ input pin
// CMPEXCFG |= 0x01;     //P5.0 is CMP+ input pin
// CMPEXCFG |= 0x02;     //P5.1 is CMP+ input pin
CMPEXCFG |= 0x03;       //ADC input pin is CMP+ input pin
CMPEXCFG &= ~0x04;     //P3.6 is CMP- input pin
// CMPEXCFG |= 0x04;     // The internal 1.19V reference voltage is the CMP- input
pin
P_SW2 &= 0x7f;          // Disable XFR access

CMPCR2 = 0x00;
CMPCR1 = 0x00;
CMPCR1 |= 0x02;         // Enable Comparator output
CMPCR1 |= 0x80;         // Enable comparator module

while (1);
}

```

Assembly code

; Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH
CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H
ADC_CONTR	DATA	0BCH
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
CMPEXCFG	XDATA	0FEAEH
	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

ANL        P1M0, #0FEH          ; Set P1.0 to input mode
ORL        P1M1, #01H
MOV        ADC_CONTR, #80H      ; Enable ADC module and select P1.0 as ADC input pin

MOV        P_SW2, #80H
MOV        DPTR, # CMPEXCFG
CLR        A
; ANL        A, #NOT 03H          ; P3.7 is CMP+ input pin
; ORL        A, #01H             ; P5.0 is CMP+ input pin
; ORL        A, #02H             ; P5.1 is CMP+ input pin
ORL        A, #03H             ; ADC input pin is CMP+ input pin
ANL        A, #NOT 04H          ; P3.6 is CMP- input pin
; ORL        A, # 04H           ; The internal 1.19V reference voltage is the CMP- input
; pin

MOVX       @DPTR, A
MOV        P_SW2, #00H

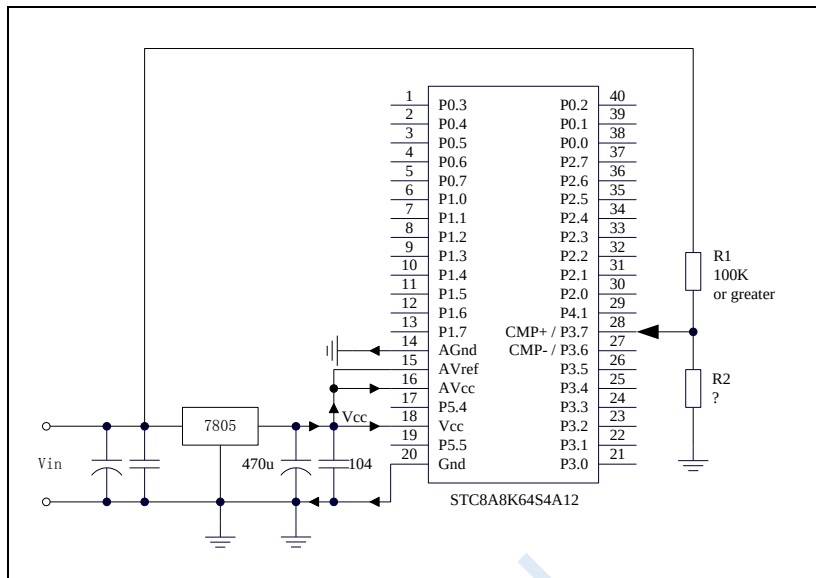
MOV        CMPCR2, #00H
MOV        CMPCR1, #00H
ORL        CMPCR1, #02H          ; Enable Comparator output
ORL        CMPCR1, #80H          ; Enable comparator module

LOOP:      JMP          LOOP

END

```

15.3.7 Comparator is Used for External Power-down Detection (User data should be saved to EEPROM in time during power down)

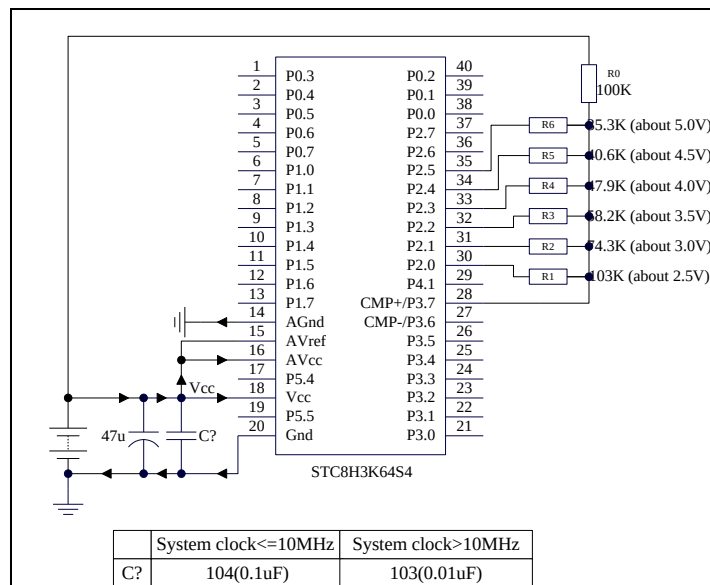


In the figure above, the resistor R1 and R2 divide the front-end voltage of the voltage regulator 7805. The divided voltage is used as the external input of the comparator CMP+ to compare with the internal reference voltage.

When the AC power is at 220V, the DC voltage at the front end of the voltage regulator block 7805 is 11V, and when the AC voltage drops to 160V, the DC voltage at the front end of the voltage regulator 7805 is 8.5V. When the dc voltage at the front end of the voltage regulator 7805 is lower than or equal to 8.5V, the dc voltage at the front end is divided by the resistors R1 and R2, and added to the comparator positive input terminal CMP+. The input voltage at the CMP+ terminal is lower than the internal reference voltage. A comparator interrupt can be generated at this time, so that there is sufficient time to save the data to the EEPROM during power-down detection. When the DC voltage of the front end of the voltage regulator 7805 is higher than 8.5V, the DC voltage input by the front end is divided by the resistors R1 and R2, and connected to the comparator positive input terminal CMP+. The input voltage of the CMP+ terminal is higher than the internal reference voltage. At this time, the CPU Can continue to work normally.

The internal reference voltage is the REPV of the internal BandGap after the amplified (the internal reference voltage is adjusted to 1.19V when the chip is shipped from the factory). The specific value should be obtained by reading the value occupied by the internal reference voltage in the internal RAM area or the Flash program memory (ROM) area. For STC8 series, the storage address of the internal reference voltage value in RAM and Flash program memory (ROM), please refer to the Chapter of "Special Parameters in Memory".

15.3.8 Comparator is Used to Detect the Operation Voltage (Battery Voltage)



In the figure above, the working voltage of the MCU can be approximately measured using the principle of resistance voltage division. The I/O port of selected channel outputs low level, which is close to GND, and the I/O port of unselected channel working in open-drain mode outputs high. Other channels are not affected.

The negative terminal of the comparator selects the internal reference voltage, and the positive terminal selects the voltage value got from the voltage divided by a resistor as the input to the CMP+ pin.

In initialization, P2.5 ~ P2.0 are set to open-drain mode and output high. Firstly, P2.0 outputs a low level. At this time, if the VCC voltage is lower than 2.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 2.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 2.5V, then make the output of P2.0 high and the output of P2.1 low. At this time, if the VCC voltage is lower than 3.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 3.0V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 3.0V, then make the output of P2.1 high and the output of P2.2 low. At this time, if the VCC voltage is lower than 3.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 3.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 3.5V, then make the output of P2.2 high and the output of P2.3 low. At this time, if the VCC voltage is lower than 4.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 4.0V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 4.0V, then make the output of P2.3 high and the output of P2.4 low. At this time, if the VCC voltage is lower than 4.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 4.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 4.5V, then make the output of P2.4 high and the output of P2.5 low. At this time, if the VCC voltage is lower than 5.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 5.0V, the comparison value of the comparator is 1.

C language code

//Operating frequency for test is 11.0592MHz

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

```
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;

```

```

    P2M0 = 0x3f;
    P2M1 = 0x3f;
    P2 = 0xff;

```

//P2.5 ~ P2.0 are initialized to open-drain mode

```

    CMPCR2 = 0x10;
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x08;
    CMPCR1 &= ~0x04;
    CMPCR1 &= ~0x02;
    CMPCR1 |= 0x80;

```

//Output comparator result after 16 debounce clocks

//P3.7 is CMP + input pin

//Internal reference voltage is CMP- input pin

//Disable comparator output

//Enable comparator module

```

    while (1)
    {
        v = 0x00;
        P2 = 0xfe;
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x01;
        P2 = 0xfd;
        delay();

```

//Voltage <2.5V

//P2.0 outputs 0

//Voltage >2.5V

//P2.1 outputs 0

```

    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x03;                                //Voltage>3.0V
    P2 = 0xfb;                                //P2.2 outputs 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x07;                                //Voltage>3.5V
    P2 = 0xf7;                                //P2.3 outputs 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x0f;                                //Voltage>4.0V
    P2 = 0xef;                                //P2.4 outputs 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x1f;                                //Voltage>4.5V
    P2 = 0xdf;                                //P2.5 outputs 0
    delay();
    if (!(CMPCR1 & 0x01)) goto ShowVol;
    v = 0x3f;                                //Voltage>5.0V
ShowVol:
    P2 = 0xff;
    P0 = ~v;
}
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

CMPCR1	DATA	0E6H
CMPCR2	DATA	0E7H
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M0	DATA	096H
P2M1	DATA	095H
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      P2M0, #00111111B      ;P2.5 ~ P2.0 are initialized to open-drain mode
MOV      P2M1, #00111111B
MOV      P2, #0FFH
MOV      CMPCR2, #10H          ;Output comparator result after 16 debounce clocks
MOV      CMPCR1, #00H
ANL      CMPCR1, #NOT 08H      ;P3.7 is CMP+ input pin
ANL      CMPCR1, #NOT 04H      ;Internal reference voltage is CMP- input pin
ANL      CMPCR1, #NOT 02H      ;Disable comparator output
ORL      CMPCR1, #80H          ;Enable comparator module

LOOP:
MOV      R0, #00000000B        ;Voltage <2.5V
MOV      P2, #11111110B        ;P2.0 outputs 0
CALL     DELAY
MOV      A, CMPCR1
JNB      ACC.0, SKIP
MOV      R0, #00000001B        ;Voltage >2.5V
MOV      P2, #11111101B        ;P2.1 outputs 0
CALL     DELAY
MOV      A, CMPCR1
JNB      ACC.0, SKIP
MOV      R0, #00000011B        ;Voltage >3.0V
MOV      P2, #11111011B        ;P2.2 outputs 0
CALL     DELAY
MOV      A, CMPCR1
JNB      ACC.0, SKIP
MOV      R0, #00000111B        ;Voltage >3.5V
MOV      P2, #11110111B        ;P2.3 outputs 0
CALL     DELAY
MOV      A, CMPCR1
JNB      ACC.0, SKIP
MOV      R0, #00001111B        ;Voltage >4.0V
MOV      P2, #11101111B        ;P2.4 outputs 0
CALL     DELAY
MOV      A, CMPCR1
JNB      ACC.0, SKIP
MOV      R0, #00011111B        ;Voltage >4.5V
MOV      P2, #11011111B        ;P2.5 outputs 0
CALL     DELAY
MOV      A, CMPCR1
JNB      ACC.0, SKIP
MOV      R0, #00111111B        ;Voltage >5.0V

SKIP:
MOV      P2, #11111111B
MOV      A, R0
CPL      A
MOV      P0, A                  ;P0.5 ~ P0.0 display voltage
JMP      LOOP

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

END

```


16 IAP/EEPROM/DATA-FLASH

Large capacity of internal EEPROM is integrated in STC8H series of microcontrollers. The internal Data Flash can be used as EEPROM by using ISP / IAP technology. And it can be repeatedly erased more than 100,000 times. EEPROM can be divided into several sectors, each sector contains 512 bytes.

Note: The EEPROM write operation can only write the 1 in the byte as 0. When the 0 in the byte needs to be written as 1, the sector erase operation must be performed. The read/write operation of EEPROM is performed in units of 1 byte, while the erase operation of EEPROM is performed in units of 1 sector (512 bytes). During the erasing operation, if there is something that needs to be reserved in the target sector Data, these data must be read into RAM for temporary storage in advance, and then the saved data and the data to be updated will be written back to EEPROM/DATA-FLASH after erasing is completed.

When EEPROM is used, it is recommended that the data modified at the same time be stored in the same sector, and data modified at different time be stored in different sectors, and not necessarily full. Data memory is erased sector by sector.

EEPROM can be used to save some parameters which need to be modified in the application and need be kept when power down takes place. In the user program, byte read / byte programming / sector erase can be performed to the EEPROM. When the operating voltage is low, it is recommended not to carry out EEPROM operation to avoid data loss.

16.1 EEPROM operation time

- Read 1 byte: 4 system clocks (use MOVC instruction to read more convenient and fast)
- Programming 1 byte: about 30~40us (the actual programming time is 6~7.5us, but state conversion time and various control SETUP and HOLD time of the control signal)
- Erase 1 sector (512 bytes): about 4~6ms

The time required for EEPROM operation is automatically controlled by the hardware, and the user only needs to set the IAP_TPS register correctly.

IAP_TPS=System operating frequency/1000000 (the decimal part is rounded to the nearest whole number)

For example: the operating frequency of the system is 12MHz, then IAP_TPS is set to 12

Another example: the system operating frequency is 22.1184MHz, then IAP_TPS is set to 22

Another example: the system operating frequency is 5.5296MHz, then IAP_TPS is set to 6

16.2 Registers Related to EEPROM

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP Flash Data Register	C2H									1111,1111
IAP_ADDRH	IAP Flash Address High Byte	C3H									0000,0000
IAP_ADDRL	IAP Flash Address Low Byte	C4H									0000,0000
IAP_CMD	IAP Flash Command Register	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP Flash Trigger register	C6H									0000,0000
IAP_CONTR	IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
IAP_TPS	IAP Waiting Time Control Register	F5H	-	-			IAPTPS[5:0]				xx00,0000

16.2.1 EEPROM data register (IAP_DATA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

During EEPROM being read operation, the EEPROM data be read after the command execution is

completed is stored in the IAP_DATA register. When writing the EEPROM, the data to be written must be stored in the IAP_DATA register before the write command is sent. The erase EEPROM command is not related to the IAP_DATA register.

16.2.2 EEPROM address registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

The target address register of EEPROM for reading, writing and erasing operation. IAP_ADDRH is the high byte address, and IAP_ADDRL is the low byte of the address.

16.2.3 EEPROM command register (IAP_CMD)

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

CMD[1:0]: EEPROM operation command to be sent.

00: No operation.

01: EEPROM reading command. Read one byte from the destination address. **Note: Writing operations can only write 1 as 0 in the destination byte, not 0 as 1. Generally, when the target byte is not FFH, it must be erased first.**

10: EEPROM writing command. Write one byte from the destination address.

11: EEPROM erasing command. Write one sector from the destination address. **Note: The erase operation will erase 1 sector (512 bytes) at a time, and the content of the entire sector will become FFH.**

16.2.4 EEPROM trigger register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

After setting the command register, address register, data register and control register of EEPROM for reading, writing and erasing operation, 5AH and A5H are written to the trigger register IAP_TRIG sequentially to trigger the corresponding operation. The order of 5AH and A5H can not be changed. After the operation is completed, the contents of the EEPROM address registers IAP_ADDRH, IAP_ADDRL and the EEPROM command register IAP_CMD do not change. The value of the IAP_ADDRH and IAP_ADDRL registers must be updated manually if the datum of the next address needs to be operated.

Note: For every EEPROM operation, 5AH should be written to IAP_TRIG firstly and then A5H to take effect the corresponding command. After the trigger command has been written, the CPU is in IDLE state until the corresponding IAP operation completes. And then the CPU will return to the normal state from the IDLE and resume executing the CPU instructions.

16.2.5 EEPROM control register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

IAPEN: EEPROM operation enable control bit.

0: disable EEPROM operation.

1: Enable EEPROM operation.

SWBS: Software reset selection control bit, which should be used with SWRST.

0: Execute the program from the user code area after the software reset.

1: Execute the program from the ISP memory area after the software reset.

SWRST: Software reset control bit.

0: No operation.

1: Generate software reset.

CMD_FAIL: Command fail status bit for EEPROM operation which should be cleared by software.

0: EEPROM operation is right.

1: EEPROM operation fails.

16.2.6 EEPROM erase wait time control register

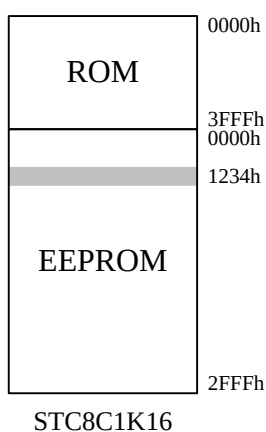
Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

Need to be set according to the operating frequency. If the working frequency is 12MHz, IAP_TPS needs to be set to 12; if the working frequency is 24MHz, IAP_TPS needs to be set to 24, and so on for other frequencies.

16.3 EEPROM Size and Address

There is EEPROM for saving user data in all STC8H series of microcontrollers. There are three operation modes for the internal EEPROM: reading, writing, and erasing. The erasing operation is performed in sectors. Each sector has 512 bytes. That is, each time an erasing command will erase a sector when it executes. The reading and writing operations are in bytes, that is, only one byte can be read or written each time when a reading or writing command is executed.

There are two ways to access the internal EEPROM of STC8H series microcontrollers: IAP mode and MOVC mode. The IAP mode can perform reading, writing and erasing operations on the EEPROM. MOVC can only perform reading operations on the EEPROM, cannot perform writing and erasing operations. Regardless of whether IAP or MOVC is used to access the EEPROM, the correct target address must be set firstly. In IAP mode, the target address is the same as the actual physical address of the EEPROM. Both of them are accessed from address 0000H. However, when using MOVC instruction to read EEPROM data, the target address must be the actual physical address of the EEPROM plus a program size offset. STC8H1K16 is used as an example to describe the target address in detail as following:



The program space of STC8H1K16 is 16K bytes (0000h ~ 3FFFh), and the EEPROM space is 12K bytes (0000h ~ 2FFFh). When you need to read, write, and erase the unit with EEPROM physical address 1234h, if you use IAP to access, set the target address to 1234h, that is, IAP_ADDRH is set to 12h, IAP_ADDRL is set to 34h, and then the corresponding trigger command can be set and the 1234h can be operated correctly. However, if the 1234h unit of the EEPROM is read by MOVC, the flash program memory (ROM) space must be added in addition to 4000h. That is, the DPTR must be set to 5234h before the MOVC instruction can be used for reading.

Note: Because the erasing is performed in 512-byte units, the lower 9 bits of the target address set when performing the erasing operation are meaningless. For example, if the target address is set to 1234H, 1200H, 1300H or 13FFH when executing the erasing command, the final erasing operation is the same, and the 512

bytes of 1200H ~ 13FFH are erased.

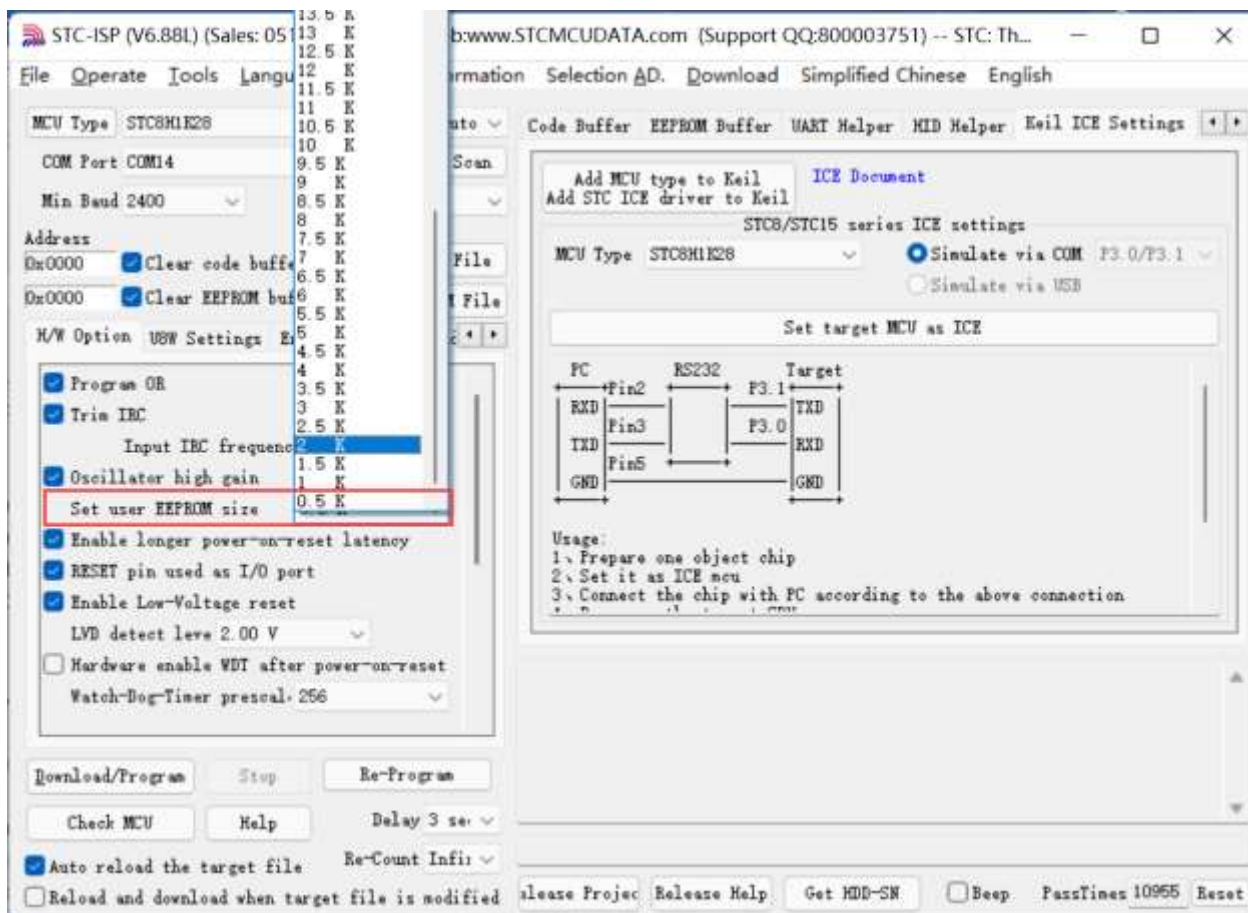
The size and access address of the internal EEPROM are different for different models. The size and address of EEPROM of each model are listed in the table below.

Model	Size	Sectors	Reading, writing, erasing in IAP mode		Reding using MOVC	
			Beginning adress	End address	Beginning adress	End address
STC8H1K16	12K	24	0000h	2FFFh	4000h	6FFFh
STC8H1K24	4K	8	0000h	0FFFh	6000h	6FFFh
STC8H1K28	User defined ^[1]					
STC8H1K33	User defined ^[1]					
STC8H1K08	4K	8	0000h	0FFFh	2000h	2FFFh
STC8H1K12	User defined ^[1]					
STC8H1K17	User defined ^[1]					
STC8H3K32S4	32K	64	0000h	7FFFh	8000h	FFFFh
STC8H3K48S4	16K	32	0000h	3FFFh	C000h	FFFFh
STC8H3K60S4	4K	8	0000h	0FFFh	F000h	FFFFh
STC8H3K64S4	User defined ^[1]					
STC8H3K32S2	32K	64	0000h	7FFFh	8000h	FFFFh
STC8H3K48S2	16K	32	0000h	3FFFh	C000h	FFFFh
STC8H3K60S2	4K	8	0000h	0FFFh	F000h	FFFFh
STC8H3K64S2	User defined ^[1]					
STC8H8K32U	32K	64	0000h	7FFFh	8000h	FFFFh
STC8H8K48U	16K	32	0000h	3FFFh	C000h	FFFFh
STC8H8K60U	4K	8	0000h	0FFFh	F000h	FFFFh
STC8H8K64U	User defined ^[1]					
STC8H2K32T	32K	64	0000h	7FFFh	8000h	FFFFh
STC8H2K48T	16K	32	0000h	3FFFh	C000h	FFFFh
STC8H2K60T	4K	8	0000h	0FFFh	F000h	FFFFh
STC8H2K64T	User defined ^[1]					
STC8H4K32TLR	32K	64	0000h	7FFFh	8000h	FFFFh
STC8H4K48TLR	16K	32	0000h	3FFFh	C000h	FFFFh
STC8H4K60TLR	4K	8	0000h	0FFFh	F000h	FFFFh
STC8H4K64TLR	User defined ^[1]					
STC8H4K32TLCD	32K	64	0000h	7FFFh	8000h	FFFFh
STC8H4K48TLCD	16K	32	0000h	3FFFh	C000h	FFFFh
STC8H4K60TLCD	4K	8	0000h	0FFFh	F000h	FFFFh
STC8H4K64TLCD	User defined ^[1]					
STC8H4K32LCD	32K	64	0000h	7FFFh	8000h	FFFFh
STC8H4K48LCD	16K	32	0000h	3FFFh	C000h	FFFFh
STC8H4K60LCD	4K	8	0000h	0FFFh	F000h	FFFFh
STC8H4K64LCD	User defined ^[1]					

[1]: This is a special model. The EEPROM size of this model can be set by the user when downloading by the ISP, as shown below:

Users can plan any EEPROM space in the entire FLASH space provided that the size does not exceed the FLASH size according to their own needs. It should be noted that **the EEPROM is always planned from the back to the front.**

For example, the size of FLASH in STC8H1K28 is 28K. If user wants to allocate 8K of it as EEPROM, the physical address of EEPROM is the last 8K of 28K, and the physical address is 5000h ~ 6FFFh. Of course, if the user uses IAP to access, the target address still starts from 0000h and ends at 1FFFh. When using MOVC to read, the target address is in the range from 5000h to 6FFFh.



16.4 Example Routines

16.4.1 EEPROM Basic Operation

C language code

//Operating frequency for test is 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;           //Disable IAP function
    IAP_CMD = 0;             //Clear command register
    IAP_TRIG = 0;           //Clear trigger register
    IAP_ADDRH = 0x80;        //Set the address to a non-IAP area
    IAP_ADDRL = 0;
}
```

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

```
{
    char dat;

    IAP_CONTR = 0x80;        //Enable IAP
    IAP_TPS = 12;            //Set the erasing wait parameter of 12MHz
    IAP_CMD = 1;             //Set IAP read command
    IAP_ADDRL = addr;        //Set IAP low address
    IAP_ADDRH = addr >> 8;   //Set IAP high address
    IAP_TRIG = 0x5a;         //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;         //Write trigger command (0xa5)
    _nop_();
    dat = IAP_DATA;          //Read IAP data
    IapIdle();               //Disable IAP function

    return dat;
}
```

```
void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;           //Enable IAP
    IAP_TPS = 12;               //Set the erasing wait parameter of 12MHz
    IAP_CMD = 2;                //Set IAP writing command
    IAP_ADDRRL = addr;          //Set IAP low address
    IAP_ADDRH = addr >> 8;      //Set IAP high address
    IAP_DATA = dat;             //Write IAP data
    IAP_TRIG = 0x5a;            //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;            //Write trigger command (0xa5)
    _nop_();
    IapIdle();                  //Disable IAP function
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;           //Enable IAP
    IAP_TPS = 12;               //Set the erasing wait parameter of 12MHz
    IAP_CMD = 3;                //Set IAP erasing command
    IAP_ADDRRL = addr;          //Set IAP low address
    IAP_ADDRH = addr >> 8;      //Set IAP high address
    IAP_TRIG = 0x5a;            //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;            //Write trigger command (0xa5)
    _nop_();                    //
    IapIdle();                  //Disable IAP function
}

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);
}
```

Assembly code

;Operating frequency for test is 11.0592MHz

IAP_DATA	DATA	0C2H
IAP_ADDRH	DATA	0C3H
IAP_ADDRRL	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	;Disable IAP function
	MOV	IAP_CMD,#0	;Clear command register
	MOV	IAP_TRIG,#0	;Clear trigger register
	MOV	IAP_ADDRH,#80H	;Set the address to a non-IAP area
	MOV	IAP_ADDRL,#0	
	RET		
IAP_READ:			
	MOV	IAP_CONTR,#80H	;Enable IAP
	MOV	IAP_TPS,#12	;Set the erasing wait parameter of 12MHz
	MOV	IAP_CMD,#1	;Set IAP read command
	MOV	IAP_ADDRL,DPL	;Set IAP low address
	MOV	IAP_ADDRH,DPH	;Set IAP high address
	MOV	IAP_TRIG,#5AH	;Write trigger command (0x5a)
	MOV	IAP_TRIG,#0A5H	;Write trigger command (0xa5)
	NOP		
	MOV	A,IAP_DATA	;Read IAP data
	LCALL	IAP_IDLE	;Disable IAP function
	RET		
IAP_PROGRAM:			
	MOV	IAP_CONTR,#80H	;Enable IAP
	MOV	IAP_TPS,#12	;Set the erasing wait parameter of 12MHz
	MOV	IAP_CMD,#2	;Set IAP writing command
	MOV	IAP_ADDRL,DPL	;Set IAP low address
	MOV	IAP_ADDRH,DPH	;Set IAP high address
	MOV	IAP_DATA,A	;Write IAP data
	MOV	IAP_TRIG,#5AH	;Write trigger command (0x5a)
	MOV	IAP_TRIG,#0A5H	;Write trigger command (0xa5)
	NOP		
	LCALL	IAP_IDLE	;Disable IAP function
	RET		
IAP_ERASE:			
	MOV	IAP_CONTR,#80H	;Enable IAP
	MOV	IAP_TPS,#12	;Set the erasing wait parameter of 12MHz
	MOV	IAP_CMD,#3	;Set IAP erasing command


```

MOV      IAP_ADDRL,DPL      ;Set IAP low address
MOV      IAP_ADDRH,DPH      ;Set IAP high address
MOV      IAP_TRIG,#5AH      ;Write trigger command (0x5a)
MOV      IAP_TRIG,#0A5H     ;Write trigger command (0xa5)
NOP
LCALL    IAP_IDLE            ;Disable IAP function
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 Read EEPROM using MOVC

C language code

//Operating frequency for test is 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 0x4000H //STC8H1K16
```

```
void IapIdle()
```

```

{
    IAP_CONTR = 0;           //Disable IAP function
    IAP_CMD = 0;             //Clear command register
    IAP_TRIG = 0;           //Clear trigger register
    IAP_ADDRH = 0x80;       //Set the address to a non-IAP area
    IAP_ADDRL = 0;
}

```

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

```

{
    addr += IAP_OFFSET;     //Using MOVC to read the EEPROM needs to add the
corresponding offset
    return *(char code *)(addr); //Read data using MOVC
}

```

```
void IapProgram(int addr, char dat)
```

```

{
    IAP_CONTR = 0x80;       //Enable IAP
    IAP_TPS = 12;           //Set the erasing wait parameter of 12MHz
    IAP_CMD = 2;            //Set IAP writing command
    IAP_ADDRL = addr;       //Set IAP low address
    IAP_ADDRH = addr >> 8;  //Set IAP high address
    IAP_DATA = dat;         //Write IAP data
    IAP_TRIG = 0x5a;        //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;        //Write trigger command (0xa5)
    _nop_();
    IapIdle();              //Disable IAP function
}

```

```
void IapErase(int addr)
```

```

{
    IAP_CONTR = 0x80;       //Enable IAP
    IAP_TPS = 12;           //Set the erasing wait parameter of 12MHz
    IAP_CMD = 3;            //Set IAP erasing command
    IAP_ADDRL = addr;       //Set IAP low address
    IAP_ADDRH = addr >> 8;  //Set IAP high address
    IAP_TRIG = 0x5a;        //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;        //Write trigger command (0xa5)
    _nop_();
    IapIdle();              //Disable IAP function
}

```

```
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);
}

```

Assembly code

;Operating frequency for test is 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      4000H           ;STC8H1K16

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           ;Disable IAP function
    MOV        IAP_CMD,#0             ;Clear command register
    MOV        IAP_TRIG,#0            ;Clear trigger register
    MOV        IAP_ADDRH,#80H         ;Set the address to a non-IAP area
    MOV        IAP_ADDRL,#0
    RET

```

IAP_READ:

```

MOV      A,#LOW IAP_OFFSET      ;Using MOVC to read the EEPROM needs to add the
corresponding offset
ADD      A,DPL
MOV      DPL,A
MOV      A,@HIGH IAP_OFFSET
ADDC     A,DPH
MOV      DPH,A
CLR      A
MOVC     A,@A+DPTR              ;Read data using MOVC
RET

```

IAP_PROGRAM:

```

MOV      IAP_CONTR,#80H        ;Enable IAP
MOV      IAP_TPS,#12           ;Set the erasing wait parameter of 12MHz
MOV      IAP_CMD,#2            ;Set IAP writing command
MOV      IAP_ADDRL,DPL         ;Set IAP low address
MOV      IAP_ADDRH,DPH        ;Set IAP high address
MOV      IAP_DATA,A           ;Write IAP data
MOV      IAP_TRIG,#5AH         ;Write trigger command (0x5a)
MOV      IAP_TRIG,#0A5H        ;Write trigger command (0xa5)
NOP
LCALL    IAP_IDLE              ;Disable IAP function
RET

```

IAP_ERASE:

```

MOV      IAP_CONTR,#80H        ;Enable IAP
MOV      IAP_TPS,#12           ;Set the erasing wait parameter of 12MHz
MOV      IAP_CMD,#3            ;Set IAP erasing command
MOV      IAP_ADDRL,DPL         ;Set IAP low address
MOV      IAP_ADDRH,DPH        ;Set IAP high address
MOV      IAP_TRIG,#5AH         ;Write trigger command (0x5a)
MOV      IAP_TRIG,#0A5H        ;Write trigger command (0xa5)
NOP
LCALL    IAP_IDLE              ;Disable IAP function
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 Send Out the Data in EEPROM Using UART

C language code

//Operating frequency for test is 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;           //Disable IAP function
    IAP_CMD = 0;             //Clear command register
    IAP_TRIG = 0;            //Clear trigger register
    IAP_ADDRH = 0x80;        //Set the address to a non-IAP area
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = 0x80;        //Enable IAP
    IAP_TPS = 12;            //Set the erasing wait parameter of 12MHz
    IAP_CMD = 1;             //Set IAP read command
    IAP_ADDRL = addr;        //Set IAP low address
    IAP_ADDRH = addr >> 8;   //Set IAP high address
    IAP_TRIG = 0x5a;         //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;         //Write trigger command (0xa5)
    _nop_();
    dat = IAP_DATA;          //Read IAP data
    IapIdle();               //Disable IAP function

    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;        //Enable IAP
    IAP_TPS = 12;            //Set the erasing wait parameter of 12MHz
    IAP_CMD = 2;             //Set IAP writing command
    IAP_ADDRL = addr;        //Set IAP low address
    IAP_ADDRH = addr >> 8;   //Set IAP high address
    IAP_DATA = dat;          //Write IAP data
    IAP_TRIG = 0x5a;         //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;         //Write trigger command (0xa5)
    _nop_();
    IapIdle();               //Disable IAP function
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;        //Enable IAP
    IAP_TPS = 12;            //Set the erasing wait parameter of 12MHz
    IAP_CMD = 3;             //Set IAP erasing command
    IAP_ADDRL = addr;        //Set IAP low address
    IAP_ADDRH = addr >> 8;   //Set IAP high address
    IAP_TRIG = 0x5a;         //Write trigger command (0x5a)
    IAP_TRIG = 0xa5;         //Write trigger command (0xa5)
    _nop_();
    IapIdle();               //Disable IAP function
}

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);
}

```

Assembly code

;Operating frequency for test is 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
MOV	T2H,#0FFH

;65536-11059200/115200/4=0FFE8H

```

MOV    AUXR,#15H
RET

UART_SEND:
JNB    TI,$
CLR    TI
MOV    SBUF,A
RET

IAP_IDLE:
MOV    IAP_CONTR,#0           ;Disable IAP function
MOV    IAP_CMD,#0            ;Clear command register
MOV    IAP_TRIG,#0           ;Clear trigger register
MOV    IAP_ADDRH,#80H        ;Set the address to a non-IAP area
MOV    IAP_ADDRL,#0
RET

IAP_READ:
MOV    IAP_CONTR,#80H        ;Enable IAP
MOV    IAP_TPS,#12           ;Set the erasing wait parameter of 12MHz
MOV    IAP_CMD,#1            ;Set IAP read command
MOV    IAP_ADDRL,DPL         ;Set IAP low address
MOV    IAP_ADDRH,DPH         ;Set IAP high address
MOV    IAP_TRIG,#5AH         ;Write trigger command (0x5a)
MOV    IAP_TRIG,#0A5H        ;Write trigger command (0xa5)
NOP
MOV    A,IAP_DATA             ;Read IAP data
LCALL  IAP_IDLE               ;Disable IAP function
RET

IAP_PROGRAM:
MOV    IAP_CONTR,#80H        ;Enable IAP
MOV    IAP_TPS,#12           ;Set the erasing wait parameter of 12MHz
MOV    IAP_CMD,#2            ;Set IAP writing command
MOV    IAP_ADDRL,DPL         ;Set IAP low address
MOV    IAP_ADDRH,DPH         ;Set IAP high address
MOV    IAP_DATA,A            ;Write IAP data
MOV    IAP_TRIG,#5AH         ;Write trigger command (0x5a)
MOV    IAP_TRIG,#0A5H        ;Write trigger command (0xa5)
NOP
LCALL  IAP_IDLE               ;Disable IAP function
RET

IAP_ERASE:
MOV    IAP_CONTR,#80H        ;Enable IAP
MOV    IAP_TPS,#12           ;Set the erasing wait parameter of 12MHz
MOV    IAP_CMD,#3            ;Set IAP erasing command
MOV    IAP_ADDRL,DPL         ;Set IAP low address
MOV    IAP_ADDRH,DPH         ;Set IAP high address
MOV    IAP_TRIG,#5AH         ;Write trigger command (0x5a)
MOV    IAP_TRIG,#0A5H        ;Write trigger command (0xa5)
NOP
LCALL  IAP_IDLE               ;Disable IAP function
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 UART1 reads and writes EEPROM - Read using MOVC

C language code (main.c)

//Operating frequency for test is 11.0592MHz

/ This program is completely normal after testing, and does not provide telephone technical support. If you cannot understand it, please supplement the relevant basis yourself. */*

*/****** 本程序功能说明 ******

STC8G family MCUs EEPROM general test program.

Please do not modify the program firstly, and download the "UART-EEPROM.hex" directly to test "02-UART 1 reads and writes EEPROM- using MOVC to read". Select the frequency 11.0592MHZ when downloading.

PC serial port setting: baud rate 115200,8,n,1.

Do sector erase, write 64 bytes, and read 64 bytes of EEPROM.

Command example:

E 0 Perform sector erasing operation on EEPROM, E means erasing, the number 0 is 0 sector (decimal, 0~126, see the specific IC).

W 0 Write operation to EEPROM, W means write, number 0 is 0 sector (decimal, 0~126, see the specific IC). Write 64 bytes continuously from the start address of the sector.

R 0 Perform IAP read operation on EEPROM, R means read out, the number 0 is 0 sector (decimal, 0~126, see the specific IC). Read 64 bytes continuously from the start address of the sector.

*M 0 Perform MOVC read operation on EEPROM (operation address is sector*512+offset address), number 0 is sector 0 (decimal, 0~126, see the specific IC). Read 64 bytes continuously from the start address of the sector .*

Note: For general purpose, the program does not recognize whether the sector is valid or not, and the user decides according to the specific model.

Date: 2019-6-10

```

/*****

```

```

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

```

```

#define Baudrate1          115200L
#define UART1_BUF_LENGTH  10
#define EEADDR_OFFSET      (8 * 1024)    //Define the offset added when EEPROM is accessed using
MOVC                                     // Equal to the size of the FLASH ROM.
                                     // For IAP or IRC at the beginning, the offset must be 0

#define TimeOutSet1        5

```

```

/***** local constant declaration *****/

```

```

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。"};

```

```

/***** local variable declaration *****/

```

```

u8 xdatatmp[70];
u8 xdataRX1_Buffer[UART1_BUF_LENGTH];
u8 RX1_Cnt;
u8 RX1_TimeOut;
bit B_TX1_Busy;

```

```

/***** local function declaration *****/

```

```

void UART1_config(void);
void TX1_write2buff(u8 dat);           //write send buffer
void PrintString1(u8 *puts);          //send a string

```

```

/***** External function and variable declarations *****/

```

```

/*****

```

```

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)                       //Within 5 bytes is sector operation, decimal,
                                           //Command supported: 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)                                     //Limited to sectors 0~123
    {
        address <= 9;
        return (address);
    }
}
else if(RX1_Cnt == 8)                                     //8 bytes direct address operation, hex,
                                                         //Command supported: 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);                             //Limited to sectors 0~123
    }
}

return 65535;                                             //error
}

```

```

//=====
// Function: void delay_ms(u8 ms)
// Description: delay function
// Parameters: ms, the number of ms to delay, here only supports 1~255ms. Automatically adapt to the main clock.
// Return: none.
// Version: VER1.0
// Date: 2013-4-1
// Remark:
//=====

```

```

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

```

//Read EEPROM using MOVC

```

void EEPROM_MOVC_read_n(u16 EE_address, u8 *DataAddress, u16 number)
{
    u8 code *pc;

    pc = EE_address + EEADDR_OFFSET;
    do
    {
        *DataAddress = *pc;                             //Data be read
        DataAddress++;
        pc++;
    }while(--number);
}

```

```

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

    UART1_config0; // select baud rate, 2: Use Timer2 as baud rate generator,
                  // Other values: Use Timer1 as baud rate generator,
    EA = 1;        // Enable CPU interrupt

    PrintString1("STC8 family MCU 用串口1 测试EEPROM 程序!\r\n"); //UART1 send a string

    while(1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0) //timeout count
        {
            if(--RX1_TimeOut == 0)
            {
                if(RX1_Buffer[1] == ' ')
                {
                    addr = GetAddress();
                    if(addr < 63488) //Limited to sectors 0~123
                    {
                        if(RX1_Buffer[0] == 'E') //PC requests to erase a sector
                        {
                            EEPROM_SectorErase(addr);
                            PrintString1("扇区擦除完成!\r\n");
                        }

                        else if(RX1_Buffer[0] == 'W') //PC request to write 64 bytes data to EEPROM
                        {
                            EEPROM_write_n(addr, T_Strings, 64);
                            PrintString1("写入操作完成!\r\n");
                        }

                        else if(RX1_Buffer[0] == 'R') //PC requests to return 64 bytes of EEPROM data
                        {
                            PrintString1("IAPData be read 如下:\r\n");
                            EEPROM_read_n(addr, tmp, 64);
                            for(i=0; i<64; i++)
                                TX1_write2buff(tmp[i]); // Return data to serial port
                            TX1_write2buff(0x0d);
                            TX1_write2buff(0x0a);
                        }

                        else if(RX1_Buffer[0] == 'M') //PC requests to return 64 bytes of EEPROM data
                        {
                            PrintString1("MOVCData be read 如下:\r\n");
                            EEPROM_MOVC_read_n(addr, tmp, 64);
                            for(i=0; i<64; i++)
                                TX1_write2buff(tmp[i]); // Return data to serial port
                            TX1_write2buff(0x0d);
                            TX1_write2buff(0x0a);
                        }

                        else PrintString1("命令错误!\r\n");
                    }
                }
            }
        }
    }
}

```

```

        RX1_Cnt = 0;
    }
}

}

}

}

/*****/

/***** send a byte *****/

void TX1_write2buff(u8 dat)                //write send buffer
{
    B_TX1_Busy = 1;                        //Set sending busy flag
    SBUF = dat;                            //send a byte
    while(B_TX1_Busy);                     //wait for sending complish
}

//=====
// Function: void PrintString1(u8 *puts)
// Description: UART1 send string function.
// Parameters:puts: String pointer.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//=====
void PrintString1(u8 *puts)                //send a string
{
    for (; *puts != 0; puts++)            //End with stop 0
    {
        TX1_write2buff(*puts);
    }
}

//=====
// Function: void UART1_config(void)
// Description: UART1 initialization function.
// Parameters:none.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//=====
void UART1_config(void)
{
    TR1 = 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;                              // Diable Timer1 interrupt
    INT_CLKO &= ~0x02;                    // Timer1 does not output high-speed clock
    TR1 = 1;                              // Start Timer1

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

    SCON = (SCON & 0x3f) | 0x40;          //UART1 mode, 0x00: synchronized shift output,

```

```

//          0x40: 8-bit data, variable baud rate,
//          0x80: 9-bit data, fixed baud rate,
//          0xc0: 9-bit data, variable baud rate
// PS = 1;
// ES = 1;
// REN = 1;
//high priority interrupt
//enable interrupt
//enable receiving

B_TX1_Busy = 0;
RX1_Cnt = 0;
}

//=====================================================
// Function: void UART1_int (void) interrupt UART1_VECTOR
// Description: UART1 interrupt function
// Parameters:nine.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//=====================================================
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUF_LENGTH)
            RX1_Cnt = 0; //avoid overflow
        RX1_Buffer[RX1_Cnt++] = SBUF;
        RX1_TimeOut = TimeOutSet1;
    }

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

```

C language code (EEPROM.c)

//Operating frequency for test is 11.0592MHz

// This program is the built-in EEPROM read and write program of STC series.

```

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

```

```

//=====================================================
// Function: voidISP_Disable(void)
// Description: Disable access ISP/IAP.
// Parameters:non.
// Return: non.
// Version: V1.0, 2012-10-22
//=====================================================
void DisableEEPROM(void)
{
    ISP_CONTR = 0; //Disable ISP/IAP operation
    IAP_TPS = 0;
    ISP_CMD = 0; //Remove ISP/IAP commands
    ISP_TRIG = 0; //Prevent false triggering of ISP/IAP commands
}

```

```

    ISP_ADDRH = 0xff;           //Clear address high byte
    ISP_ADDRL = 0xff;           //Clear address low byte, point to non-EEPROM area to prevent
misoperation
}

//=====================================================
// Function: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// Description: Read n bytes from the specified EEPROM first address and put them in the specified buffer.
// Parameters:EE_address: The first address of the EEPROM to read.
//   DataAddress: The first address of the data buffer.
//   number:     The length of bytes to read.
// Return: non.
// Version: V1.0, 2012-10-22
//=====================================================
void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0;                     //Disable interrupts
    ISP_CONTR = ISP_EN;          //Allow ISP/IAP operation
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Working frequency setting
    ISP_READ();                  //Send byte read command, when the command does not need to
be changed, there is no need to send the command again
    do
    {
        ISP_ADDRH = EE_address / 256; //Send the high byte of the address (the address needs to be re-
sent when the address needs to be changed)
        ISP_ADDRL = EE_address % 256; //Send the low byte of the address
        ISP_TRIG();                  //Send 5AH firstly, then send A5H to the ISP/IAP trigger register,
//Do this every time
//After sending A5H, the ISP/IAP command is triggered to start
immediately
//The CPU waits for the IAP to complete before continuing to
execute the program.
        _nop_0;
        _nop_0;
        _nop_0;
        *DataAddress = ISP_DATA; //Data be read
        EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = 1;                     //re-enable interrupt
}

/***** Sector Erase Function *****/
//=====================================================
// Function: void EEPROM_SectorErase(u16 EE_address)
// Description: Erase the EEPROM sector at the specified address.
// Parameters:EE_address: The address of the sector EEPROM to be erased.
// Return: non.
// Version: V1.0, 2013-5-10
//=====================================================
void EEPROM_SectorErase(u16 EE_address)
{
    EA = 0;                     //Disable interrupts
//Only sector erase, no byte erase, 512 bytes per sector.
//Any byte address in a sector is sector address.
    ISP_ADDRH = EE_address / 256; //Send the high byte of the sector address (the address needs to be
re-sent when the address needs to be changed)

```

```

    ISP_ADDRL = EE_address % 256;           //Send the low byte of the sector address
    ISP_CONTR = ISP_EN;                     //Allow ISP/IAP operation
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L);   //Working frequency setting
    ISP_ERASE();                             //Send sector erase command. When the command does not need
to be changed, it is not necessary to send the command again
    ISP_TRIG();
    _nop_();
    _nop_();
    _nop_();
    DisableEEPROM();
    EA = 1;                                //re-enable interrupt
}

//=====================================================
// Function: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// Description: Write the buffered n bytes into the EEPROM of the specified first address.
// Parameters:EE_address: Write the first address of the EEPROM.
//   DataAddress: The first address of the buffer where the source data is written.
//   number:     The length of bytes written.
// Return: non.
// Version: V1.0, 2012-10-22
//=====================================================
void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0;                                //Disable interrupts

    ISP_CONTR = ISP_EN;                     //Allow ISP/IAP operation
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L);   //Working frequency setting
    ISP_WRITE();                             //Send byte write command. When the command does not need to
be changed, no need to send the command again
    do
    {
        ISP_ADDRH = EE_address / 256;       //Send the high byte of the address (the address needs to be re-
sent when the address needs to be changed)
        ISP_ADDRL = EE_address % 256;       //Send the low byte of the address
        ISP_DATA = *DataAddress;            //Send data to ISP_DATA, and only need to send it again when
the data changes.
        ISP_TRIG();
        _nop_();
        _nop_();
        _nop_();
        EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = 1;                                //re-enable interrupt
}

```

16.4.5 Password erasing and writing - multi-sector backup - UART1 operation

C language code (main.c)

//Operating frequency for test is 11.0592MHz

/* This program is completely normal after testing, and does not provide telephone technical support. If you cannot understand it, please supplement the relevant basis yourself. */

/****** Function description of this program *****/

STC8G family, STC8H family and STC8C family's EEPROM general test program to demonstrate multi-sector backup, writing with correct sector data if there is a sector error, and writing the default value for all sector errors (such as the first time the program is run).

Each writing writes 3 sectors, that is, redundant backup.

Write a record in each sector, after the writing is completed, read the saved data and check value and compare it with the source data and check value, and return the result (correct or wrong) from UART1 (P3.0 P3.1) .

Each record is self-checked, 64-byte data, 2-byte check value, check value = 64 bytes data's cumulative sum ^ 0x5555. ^0x5555 is to ensure that the written 66 data are not all 0.

If there is a sector error, the data of the correct sector will be written to the wrong sector, and if all three sectors are wrong, the default value will be written.

A password needs to be set before erasing, writing, and reading operations. If the password is incorrect, the operation will be exited, and the password will be cleared each time when an exit operation is performed.

Please do not modify the program firstly, and download the "UART-EEPROM.hex" directly to test "03-Password erasing and writing-multi-sector backup-UART 1 operation ". Select the frequency 11.0592MHZ when downloading.

PC serial port setting: baud rate 115200,8,n,1.

Do sector erase, write 64 bytes, and read 64 bytes of EEPROM.

Command example:

Use the serial port assistant to send a single character, both upper and lower case.

E or e: Perform sector erase operation on EEPROM, E means erase, it will erase sectors 0, 1, 2.

W or w: Write operation to EEPROM, W means write, will write to sectors 0, 1, 2, each sector writes 64 bytes continuously, sector 0 writes 0x0000~0x003f, sector 1 writes 0x0200~0x023f, write 0x0400~0x043f in sector 0.

R or r: Read data from the EEPROM, R means read, it will read sectors 0, 1, 2, each sector reads 64 bytes continuously, sector 0 reads 0x0000~0x003f, sector 1 reads 0x0200~0x023f, Sector 0 reads 0x0400~0x043f.

Note: For general purpose, the program does not recognize whether the sector is valid or not, and the user decides according to the specific model.

Date: 2021-11-5

*****/

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

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

```
#define Baudrate1 115200L
```

/****** local constant declaration *****/

```
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。"};
```

/****** local variable declaration *****/

```
u8 xdatatmp[70]; //General data
```

```
u8 xdataSaveTmp[70]; // array to write
```

```
bit B_TX1_Busy;
```

```
u8 cmd; // single character command of UART
```

```

/***** local function declaration *****/

```

```

void UART1_config(void);
void TX1_write2buff(u8 dat);           //write send buffer
void PrintString1(u8 *puts);          //send a string

```

```

/***** External function and variable declarations *****/

```

```

/***** Read the EEPROM record, and verify, return the verification result, 0 is correct, 1 is wrong *****/

```

```

u8 ReadRecord(u16 addr)
{
    u8 i;
    u16 ChckSum;           //Calculated cumulative sum
    u16 j;                 //Accumulated sum of reading

    for(i=0; i<66; i++) tmp[i] = 0; //clear buffer
    PassWord = D_PASSWORD; //given password
    EEPROM_read_n(addr,tmp,66); //read sector 0
    for(ChckSum=0, i=0; i<64; i++)
        ChckSum += tmp[i]; //Calculate the cumulative sum
    j = ((u16)tmp[64]<<8) + (u16)tmp[65]; //Read cumulative sum of records
    j ^= 0x5555; //Invert every other bit, avoid all 0s
    if(ChckSum != j) return 1; //If the cumulative sum is wrong, return 1
    return 0; // If the cumulative sum is right, return 0
}

```

```

/***** Write the EEPROM record, and verify, return the verification result, 0 is correct, 1 is wrong *****/

```

```

u8 SaveRecord(u16 addr)
{
    u8 i;
    u16 ChckSum;           //Calculated cumulative sum

    for(ChckSum=0, i=0; i<64; i++)
        ChckSum += SaveTmp[i]; //Calculate the cumulative sum
    ChckSum ^= 0x5555; //Invert every other bit, avoid all 0s
    SaveTmp[64] = (u8)(ChckSum >> 8);
    SaveTmp[65] = (u8)ChckSum;

    PassWord = D_PASSWORD; //given password
    EEPROM_SectorErase(addr); //Erase a sector
    PassWord = D_PASSWORD; //given password
    EEPROM_write_n(addr, SaveTmp, 66); //write a sector

    for(i=0; i<66; i++)
        tmp[i] = 0; //clear buffer
    PassWord = D_PASSWORD; //given password
    EEPROM_read_n(addr,tmp,66); //read sector 0
    for(i=0; i<66; i++) //data comparison
    {
        if(SaveTmp[i] != tmp[i])
            return 1; //If there is an error in the data, return 1
    }
    return 0; // If the cumulative sum is right, return 0
}

```

```

/***** main function *****/

```

```

void main(void)

```

```

{
    u8 i;
    u8 status; //Status

    UART1_config0; // select baud rate, 2: Use Timer2 as baud rate generator,
                    //Other values: Use Timer1 as baud rate generator,
    EA = 1; //Enable CPU interrupt

    PrintString1("STC8G-8H-8C familyMCU 用串口1 测试EEPROM 程序!\r\n"); //UART1send a string

    // Power on and read 3 sectors and verify, if there is a sector error,
    write the correct sector into the wrong sector, if all 3 sectors are wrong, write the default value.
    status = 0;
    if(ReadRecord(0x0000) == 0) //read sector 0
    {
        status |= 0x01; //If it is correct, mark status.0=1
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i]; //save data in write buffer
    }
    if(ReadRecord(0x0200) == 0) //Read sector 1
    {
        status |= 0x02; //If it is correct, mark status.1=1
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i]; //save data in write buffer
    }
    if(ReadRecord(0x0400) == 0) //Read sector 2
    {
        status |= 0x04; //If it is correct, mark status.2=1
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i]; //save data in write buffer
    }

    if(status == 0) //If all sectors are wrong, write default value
    {
        for(i=0; i<64; i++)
            SaveTmp[i] = T_StringD[i]; //Read the default value
    }
    else PrintString1("上电读取3个扇区数据均正确!\r\n"); //UART1 send a string prompt

    if((status & 0x01) == 0) // If sector 0 is wrong, write default value
    {
        if(SaveRecord(0x0000) == 0)
            PrintString1("write a sector0 正确!\r\n"); // Writing record 0 sector is correct
        else
            PrintString1("write a sector0 错误!\r\n"); // Writing record 0 sector is wrong
    }
    if((status & 0x02) == 0) // If sector 1 is wrong, write default value
    {
        if(SaveRecord(0x0200) == 0)
            PrintString1("write a sector1 正确!\r\n"); // Writing record 1 sector is correct
        else
            PrintString1("write a sector1 错误!\r\n"); // Writing record 1 sector is wrong
    }
    if((status & 0x04) == 0) // If sector 2 is wrong, write default value
    {
        if(SaveRecord(0x0400) == 0)
            PrintString1("write a sector2 正确!\r\n"); // Writing record 2 sector is correct
        else
            PrintString1("write a sector2 错误!\r\n"); // Writing record 2 sector is wrong
    }
}

```

```

}

while(1)
{
    if(cmd != 0)                                // There is UART commands
    {
        if((cmd >= 'a') && (cmd <= 'z'))
            cmd -= 0x20;                        // lowercase to uppercase

        if(cmd == 'E')                          //PC requests to erase a sector
        {
            PassWord = D_PASSWORD;              //given password
            EEPROM_SectorErase(0x0000);          //Erase a sector
            PassWord = D_PASSWORD;              //given password
            EEPROM_SectorErase(0x0200);          //Erase a sector
            PassWord = D_PASSWORD;              //given password
            EEPROM_SectorErase(0x0400);          //Erase a sector
            PrintString1("扇区擦除完成!\r\n");
        }

        else if(cmd == 'W')                     //PC request to write 64 bytes data to EEPROM
        {
            for(i=0; i<64; i++)
                SaveTmp[i] = T_StringW[i];      //write value
            if(SaveRecord(0x0000) == 0)
                PrintString1("write a sector0 正确!\r\n"); // Writing record 0 sector is correct
            else
                PrintString1("write a sector0 错误!\r\n"); // Writing record 0 sector is wrong
            if(SaveRecord(0x0200) == 0)
                PrintString1("write a sector1 正确!\r\n"); // Writing record 1 sector is correct
            else
                PrintString1("write a sector1 错误!\r\n"); // Writing record 1 sector is wrong
            if(SaveRecord(0x0400) == 0)
                PrintString1("write a sector2 正确!\r\n"); // Writing record 2 sector is correct
            else
                PrintString1("write a sector2 错误!\r\n"); // Writing record 2 sector is wrong
        }

        else if(cmd == 'R')                     //PC requests to return 64 bytes of EEPROM data
        {
            if(ReadRecord(0x0000) == 0)         //read data of sector 0
            {
                PrintString1("read sector 0 的数据如下:\r\n");
                for(i=0; i<64; i++)
                {
                    TX1_write2buff(tmp[i]);      // Return data to UART
                    TX1_write2buff(0x0d);        // carriage return and line feed
                    TX1_write2buff(0x0a);
                }
            }
            else PrintString1("read sector 0 的数据错误!\r\n");

            if(ReadRecord(0x0200) == 0)         // Read data of sector 1
            {
                PrintString1("读出扇区1 的数据如下:\r\n");
                for(i=0; i<64; i++)
                {
                    TX1_write2buff(tmp[i]);      // Return data to UART
                    TX1_write2buff(0x0d);        // carriage return and line feed
                    TX1_write2buff(0x0a);
                }
            }
        }
    }
}

```

```

else PrintString1(" 读出扇区1 的数据错误!\r\n");

if(ReadRecord(0x0400) == 0)                // Read data of sector 2
{
    PrintString1(" 读出扇区2 的数据如下 :\r\n");
    for(i=0; i<64; i++)
        TX1_write2buff(tmp[i]);           // Return data to UART
    TX1_write2buff(0x0d);                 // carriage return and line feed
    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;
}
}
}

/*****/

/***** send a byte *****/
void TX1_write2buff(u8 dat)                //write send buffer
{
    B_TX1_Busy = 1;                       //Set sending busy flag
    SBUF = dat;                           //send a byte
    while(B_TX1_Busy);                    //wait for sending complish
}

//=====
// Function: void PrintString1(u8 *puts)
// Description: UART1 send string function.
// Parameters:puts: String pointer.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//=====
void PrintString1(u8 *puts)                //send a string
{
    for (; *puts != 0; puts++)            //End with stop 0
    {
        TX1_write2buff(*puts);
    }
}

//=====
// Function: void UART1_config(void)
// Description: UART1 initialization function.
// Parameters:none.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//=====
void UART1_config(void)
{
    TR1 = 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;                      // Disable Timer1 interrupt
    INT_CLKO &= ~0x02;            // Timer1 does not output high-speed clock
    TR1 = 1;                      // Start Timer1

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

    SCON = (SCON & 0x3f) | 0x40;    //UART1 mode, 0x00: synchronized shift output,
    //                                0x40: 8-bit data, variable baud rate,
    //                                0x80: 9-bit data, fixed baud rate,
    //                                0xc0: 9-bit data, variable baud rate

//    PS = 1;                      //high priority interrupt
    ES = 1;                      //enable interrupt
    REN = 1;                     //enable receiving

    B_TX1_Busy = 0;
}

//=====
// Function: void UART1_int (void) interrupt UART1_VECTOR
// Description: UART1 interrupt function
// Parameters:nine.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//=====
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        cmd = SBUF;
    }

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

```

C language code (EEPROM.c)

//Operating frequency for test is 11.0592MHz

// This program is the built-in EEPROM read and write program of STC series.

```

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

```

```

u32      PassWord;                // Password required for erasing and writing

```

```

//=====
// Function: voidISP_Disable(void)
// Description: Disable access ISP/IAP.

```

// Parameters:non.

// Return: non.

// Version: V1.0, 2012-10-22

//=====

void DisableEEPROM(void)

```
{
    ISP_CONTR = 0;                //Disable ISP/IAP operation
    IAP_TPS = 0;
    ISP_CMD = 0;                 //Remove ISP/IAP commands
    ISP_TRIG = 0;                //Prevent false triggering of ISP/IAP commands
    ISP_ADDRH = 0xff;            //Clear address high byte
    ISP_ADDRL = 0xff;            //Clear address low byte, point to non-EEPROM area to prevent
misoperation
}
```

//=====

*// Function: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)*

// Description: Read n bytes from the specified EEPROM first address and put them in the specified buffer.

// Parameters:EE_address: The first address of the EEPROM to read.

// DataAddress: The first address of the data buffer.

// number: The length of bytes to read.

// Return: non.

// Version: V1.0, 2012-10-22

//=====

void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)

```
{
    if(PassWord == D_PASSWORD)    // Only the correct password is allowed to operate the EEPROM
    {
        EA = 0;                //Disable interrupts
        ISP_CONTR = ISP_EN;     //Allow ISP/IAP operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Working frequency setting
        ISP_READ0;              //Send byte read command, when the command does not need to
be changed, there is no need to send the command again
        do
        {
            ISP_ADDRH = EE_address / 256; //Send the high byte of the address (the address needs to be re-
sent when the address needs to be changed)
            ISP_ADDRL = EE_address % 256; //Send the low byte of the address
            if(PassWord == D_PASSWORD)    // If the password is correct, trigger the operation
            {
                ISP_TRIG = 0x5A;          //Send 5AH firstly, then send A5H to the ISP/IAP trigger register,
//Do this every time
                ISP_TRIG = 0xA5;          //After sending A5H, the ISP/IAP command is triggered to start
immediately
            }                          //The CPU waits for the IAP to complete before continuing to
execute the program.
            _nop_();
            _nop_();
            _nop_();
            *DataAddress = ISP_DATA;      //Data be read
            EE_address++;
            DataAddress++;
        }while(--number);

        DisableEEPROM();
        EA = 1;                          //re-enable interrupt
    }
    PassWord = 0;                        //clear password
}
```

***** Sector Erase Function *****

=====

// Function: void EEPROM_SectorErase(u16 EE_address)

// Description: Erase the EEPROM sector at the specified address.

// Parameters:EE_address: The address of the sector EEPROM to be erased.

// Return: non.

// Version: V1.0, 2013-5-10

=====

void EEPROM_SectorErase(u16 EE_address)

```
{
    if(PassWord == D_PASSWORD)                // Only the password is correct, the EEPROM will be operated
    {
        EA = 0;                                //Disable interrupts
                                                //Only sector erase, no byte erase, 512 bytes per sector.
                                                //Any byte address in a sector is sector address.
        ISP_ADDRH = EE_address / 256;          //Send the high byte of the sector address (the address needs to be
re-sent when the address needs to be changed)
        ISP_ADDRL = EE_address % 256;          //Send the low byte of the sector address
        ISP_CONTR = ISP_EN;                    //Allow ISP/IAP operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Working frequency setting
        ISP_ERASE();                            //Send sector erase command. When the command does not need
to be changed, it is not necessary to send the command again
        if(PassWord == D_PASSWORD)            // If the password is correct, trigger the operation
        {
            ISP_TRIG = 0x5A;                   //Send 5AH firstly, then send A5H to the ISP/IAP trigger register,
                                                //Do this every time
            ISP_TRIG = 0xA5;                   //After sending A5H, the ISP/IAP command is triggered to start
immediately
        }                                     //The CPU waits for the IAP to complete before continuing to
execute the program.
        _nop_0;
        _nop_0;
        _nop_0;
        DisableEEPROM();
        EA = 1;                                //re-enable interrupt
    }
    PassWord = 0;                              //Clear password
}
```

=====

*// Function: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)*

// Description: Write the buffered n bytes into the EEPROM of the specified first address.

// Parameters:EE_address: Write the first address of the EEPROM.

// DataAddress: The first address of the buffer where the source data is written.

// number: The length of bytes written.

// Return: non.

// Version: V1.0, 2012-10-22

=====

void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)

```
{
    if(PassWord == D_PASSWORD)                // Only the password is correct, the EEPROM will be operated
    {
        EA = 0;                                //Disable interrupts

        ISP_CONTR = ISP_EN;                    //Allow ISP/IAP operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Working frequency setting
        ISP_WRITE();                            //Send byte write command. When the command does not need to
be changed, no need to send the command again
    }
```



```
do
{
    ISP_ADDRH = EE_address / 256;           //Send the high byte of the address (the address needs to be re-
sent when the address needs to be changed)
    ISP_ADDRL = EE_address % 256;           //Send the low byte of the address
    ISP_DATA = *DataAddress;                //Send data to ISP_DATA, and only need to send it again when
the data changes.
    if(PassWord == D_PASSWORD)              // If the password is correct, trigger the operation
    {
        ISP_TRIG = 0x5A;                    //Send 5AH firstly, then send A5H to the ISP/IAP trigger register,
//Do this every time
        ISP_TRIG = 0xA5;                    //After sending A5H, the ISP/IAP command is triggered to start
immediately
    }                                       //The CPU waits for the IAP to complete before continuing to
execute the program.
    _nop_();
    _nop_();
    _nop_();
    EE_address++;
    DataAddress++;
}while(--number);

    DisableEEPROM();
    EA = 1;                                //re-enable interrupt
}
PassWord = 0;                             // Clear password
}
```

17 ADC, Internal Reference Voltage

Product line	Resolution of ADC	Channels of ADC
STC8H1K08 family	10 bit	9 channels
STC8H1K28 family	10 bit	12 channels
STC8H3K64S4 family	12 bit	12 channels
STC8H3K64S2 family	12 bit	12 channels
STC8H8K64U family	12 bit	15 channels
STC8H2K64T family	12 bit	15 channels
STC8H4K64TLR family	12 bit	15 channels
STC8H4K64TLCD family	12 bit	15 channels
STC8H4K64LCD family	12 bit	15 channels

A 10-bit/12-bit high-speed Analog to Digital Converter is integrated in STC8H family of microcontrollers. The system frequency is divided by 2 and then divided again by the user-set division ratio as the clock frequency of the ADC. The range of ADC clock frequency is $SYSClk/2/1 \sim SYSClk/2/16$.

The fastest ADC speed of STC8H series: 12-bit ADC is 800K (800,000 ADC conversions per second), 10-bit ADC is 500K (500,000 ADC conversions per second).

There are two data formats for ADC conversion results: Align left and Align right. It is convenient for user program to read and reference.

Note: The 15th channel of the ADC can only be used to detect the internal reference voltage. The reference voltage value is calibrated to 1.19V at the factory. Due to the manufacturing errors and measurement errors, the actual internal reference voltage has about $\pm 1\%$ error compared to 1.19V. If you want to know the exact internal reference voltage of each chip, you can connect an accurate reference voltage and then use the 15th channel of the ADC to measure the calibration.

If the chip has ADC external reference power supply pin ADC_Vref+, it must not be floating, it must be connected to an external reference power supply or directly connected to VCC

17.1 Registers Related to ADC

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC control register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC Result High Byte	BDH									0000,0000
ADC_RESL	ADC Result Low Byte	BEH									0000,0000
ADCCFG	ADC Configuration Register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
ADCTIM	ADC Timing Control Register	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				0010,1010	

17.1.1 ADC control register

Symbol	Address	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 power supply control bit.

0: turn off the power supply of ADC.

1: turn on the power supply of ADC.

It is recommended to turn off the ADC before entering Idle mode and Power-down mode to reduce the power consumption.

Pay attention:

1. After the power supply to the internal ADC module of the MCU is turned on, wait for about 1ms, and wait for the ADC power supply inside the MCU to stabilize before allowing the ADC to work;
2. Properly lengthening the sampling time of the external signal is the charging or discharging time of the internal sampling and holding capacitor of the ADC. If the time is enough, the internal can be equal to the

external potential.

ADC_START: ADC start bit. ADC conversion will start after write 1 to this bit. It is cleared automatically by the hardware after A/D conversion completes.

0: no effect. Writing 0 to this bit will not stop the A/D conversion if the ADC has already started.

1: start the A/D conversion. It is cleared automatically by the hardware after A/D conversion completes.

ADC_FLAG: ADC conversion completion flag. It is set by the hardware after the ADC conversion has finished, and requests interrupt to CPU. It must be cleared by software.

ADC_EPWMT: enable PWM synchronous trigger ADC function.

ADC_CHS[3:0]: ADC analog channel selection bits.

(Note: The I/O port selected as the ADC input channel must be set to the PxM0/PxM1 register to set the I/O port mode to high-impedance input mode. In addition, if the MCU enters the power-down mode/clock stop mode, it still needs To enable the ADC channel, you need to set the PxIE register to close the digital input channel to prevent the external analog input signal from fluctuating high and low and causing additional power consumption)

(STC8H1K28family)

ADC_CHS[3:0]	ADC channel
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	P0.1ADC9
1010	P0.2/ADC10
1011	P0.3/ADC11
1100	No such channel
1101	No such channel
1110	No such channel
1111	Test internal 1.19V

(STC8H1K08 family)

ADC_CHS[3:0]	ADC channel
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	No such channel
0011	No such channel
0100	No such channel
0101	No such channel
0110	No such channel
0111	No such channel
1000	P3.0/ADC8
1001	P3.1ADC9
1010	P3.2/ADC10
1011	P3.3/ADC11
1100	P3.4/ADC12
1101	P3.5/ADC13
1110	P3.6/ADC14
1111	Test internal 1.19V

(STC8H3K64S4/STC8H3K64S2 family)

ADC_CHS[3:0]	ADC channel
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P1.2/ADC2

0011	No such channel
0100	No such channel
0101	No such channel
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	Test internal 1.19V

(STC8H8K64U, C8H2K64T, STC8H4K64TLR family)

ADC_CHS[3:0]	ADC channel
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P5.4/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	Test internal 1.19V

(TC8H4K64TLCD, C8H4K64LCD family)

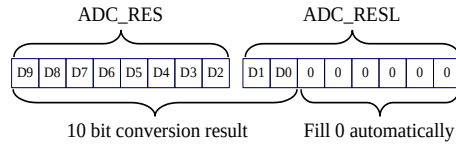
ADC_CHS[3:0]	ADC channel
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P5.4/ADC2
0011	P1.3/ADC3
0100	P1.4/ADC4
0101	P1.5/ADC5
0110	P6.2/ADC6
0111	P6.3/ADC7
1000	P0.0/ADC8
1001	P0.1/ADC9
1010	P0.2/ADC10
1011	P0.3/ADC11
1100	P0.4/ADC12
1101	P0.5/ADC13
1110	P0.6/ADC14
1111	Test internal 1.19V

17.1.2 ADC configuration register

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

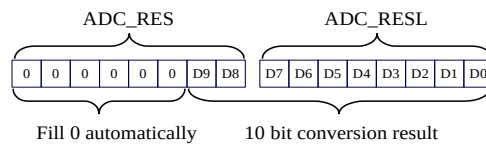
RESFMT: ADC conversion result format control bit (STC8H1K28family, C8H1K08family)

0: The conversion result aligns left. ADC_RES is used to save the upper 8 bits of the result and ADC_RESL is used to save the lower 2 bits of the result. The format is as follows:



RESFMT=0

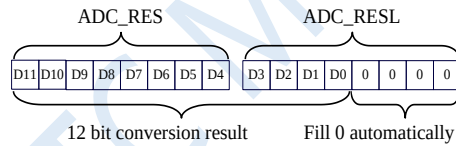
1: The conversion result aligns right. ADC_RES is used to save the upper 2 bits of the result and ADC_RESL is used to save the lower 8 bits of the result. The format is as follows:



RESFMT=1

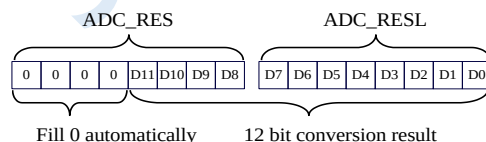
RESFMT: ADC conversion result format control bit (STC8H3K64S4 family, C8H3K64S2 family, STC8H8K64U family, STC8H2K64T family, STC8H4K64TLR family, STC8H4K64TLCD family, STC8H4K64LCD family)

0: The conversion result aligns left. ADC_RES is used to save the upper 8 bits of the result and ADC_RESL is used to save the lower 4 bits of the result. The format is as follows:



RESFMT=0

1: The conversion result aligns right. ADC_RES is used to save the upper 4 bits of the result and ADC_RESL is used to save the lower 8 bits of the result. The format is as follows:



RESFMT=1

SPEED[3:0]: ADC clock control bits { $F_{ADC} = SYSclk / (SPEED + 1)$ }

SPEED[3:0]	ADC clock frequency
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 result registers (ADC_RES, ADC_RESL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

After the A/D conversion is completed, the 10-bit/12-bit conversion result is automatically saved to ADC_RES and ADC_RESL. Please refer to the RESFMT setting in the ADC_CFG register to see the result's data format.

17.1.4 ADC timing control register

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

CSSETUP: ADC channel selection time control T_{setup}

CSSETUP	ADC number of clocks
0	1 (default)
1	2

CSHOLD[1:0]: ADC Channel selection hold time control T_{hold}

CSHOLD[1:0]	ADC number of clocks
00	1
01	2 (default)
10	3
11	4

SMPDUTY[4:0]: ADC analog signal sampling time control T_{duty} (Note: SMPDUTY must not be set less than 01010B)

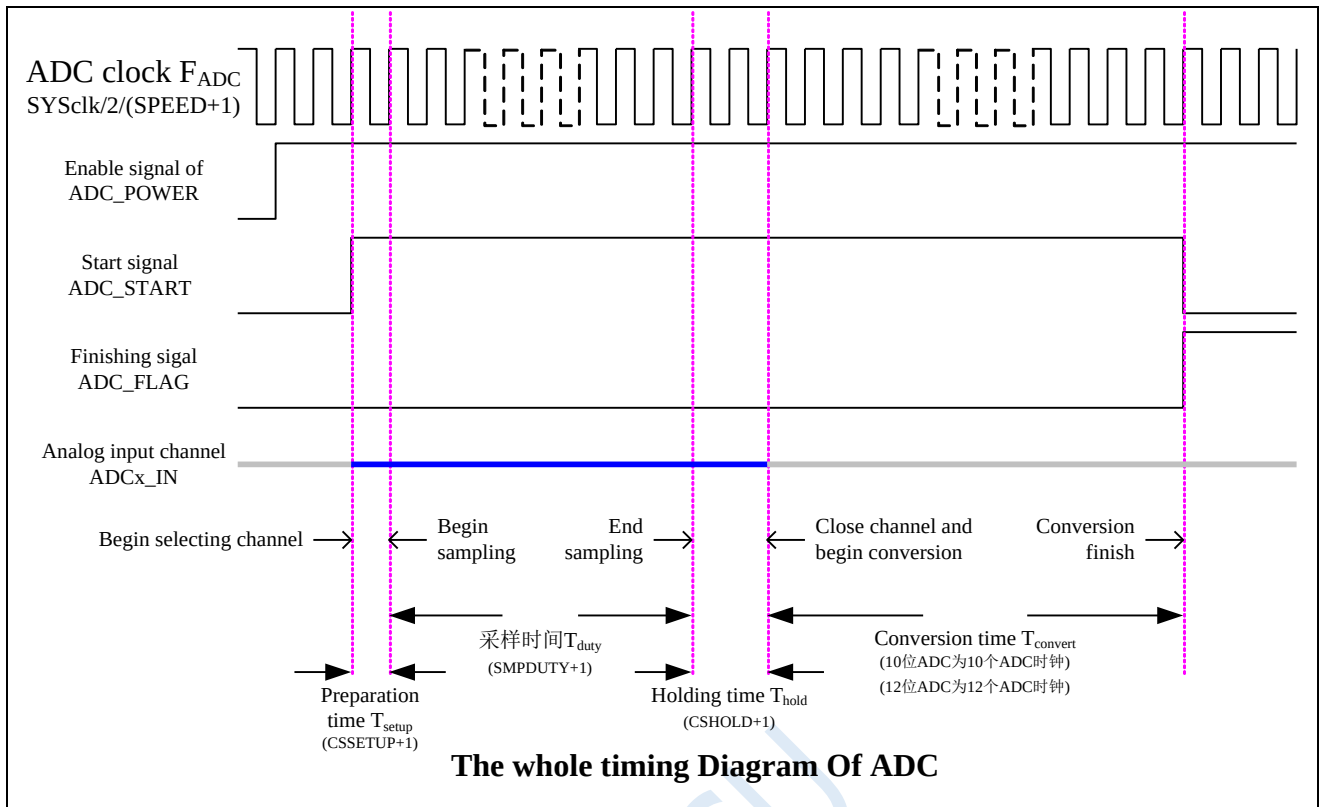
SMPDUTY[4:0]	ADC number of clocks
00000	1
00001	2
...	...
01010	11 (default)
...	...
11110	31
11111	32

ADC digital-to-analog conversion time: T_{convert}

The conversion time of 10-bit ADC is fixed at 10 ADC working clocks

The conversion time of 12-bit ADC is fixed at 12 ADC working clocks

A complete ADC conversion time is: $T_{\text{setup}} + T_{\text{duty}} + T_{\text{hold}} + T_{\text{convert}}$, as shown in the figure below



17.2 ADC related calculation formula

17.2.1 ADC speed calculation formula

The ADC conversion speed is controlled by the SPEED and ADCTIM registers in the ADCCFG register. The calculation formula of the conversion speed is as follows:

$$\text{10bit ADC conversion speed} = \frac{\text{MCU operating frequency } SYSclk}{2 \times (\text{SPEED}[3:0] + 1) \times [(\text{CSSETUP} + 1) + (\text{CSHOLD} + 1) + (\text{SMPDUTY} + 1) + 10]}$$

$$\text{12bit ADC conversion speed} = \frac{\text{MCU operating frequency } SYSclk}{2 \times (\text{SPEED}[3:0] + 1) \times [(\text{CSSETUP} + 1) + (\text{CSHOLD} + 1) + (\text{SMPDUTY} + 1) + 10]}$$

Note:

- The speed of 10-BIT ADC CANNOT BE HIGHER THAN 500KHz
- THE SPEED OF 12-BIT ADC CANNOT BE HIGHER THAN 800KHz
- The value of SMPDUTY cannot be less than 10, it is recommended to set to 15
- CSSETUP can use power-on default value 0
- CHOLD can use the power-on default value 1 (ADCTIM is recommended to be set to 3FH)

17.2.2 ADC conversion result calculation formula

$$\text{10bit ADC conversion result} = 1024 \mid \frac{\text{The input voltage } V_{in} \text{ of the ADC converted channel}}{(\text{MCU working voltage } V_{cc}) (no \text{ } ADC_{V_{ref}} +)}$$

$$\text{10bit ADC conversion result} = 1024 \mid \frac{\text{The input voltage } V_{in} \text{ of the ADC converted channel}}{(\text{ADC external reference source voltage}) (ADC_{V_{ref}} +)}$$

$$\text{12bit ADC conversion result} = 4096 \mid \frac{\text{The input voltage } V_{in} \text{ of the ADC converted channel}}{(\text{MCU working voltage } V_{cc}) (no \text{ } ADC_{V_{ref}} +)}$$

$$\text{12bit ADC conversion result} = 4096 \mid \frac{\text{The input voltage } V_{in} \text{ of the ADC converted channel}}{(\text{ADC external reference source voltage}) (ADC_{V_{ref}} +)}$$

17.2.3 Reverse calculation formula for ADC input voltage

$$\text{input voltage } V_{in} \text{ of the ADC converted channel} = \text{MCU working voltage } V_{cc} \mid \frac{\text{10bit ADC conversion result}}{1024 \mid no \text{ } ADC_{ref} +}$$

$$\text{input voltage } V_{in} \text{ of the ADC converted channel} = \text{ADC external reference source voltage} \mid \frac{\text{10bit ADC conversion result}}{1024}$$

$$\text{input voltage } V_{in} \text{ of the ADC converted channel} = \text{MCU working voltage } V_{cc} \mid \frac{\text{12bit ADC conversion result}}{4096 \mid no \text{ } ADC_{ref} +}$$

$$\text{input voltage } V_{in} \text{ of the ADC converted channel} = \text{ADC external reference source voltage} \mid \frac{\text{12bit ADC conversion result}}{4096}$$

17.2.4 Reverse working voltage calculation formula

When you need to use the ADC input voltage and ADC conversion results to reverse the working voltage, if the target chip does not have an independent ADC_V_{ref+} pin, you can directly measure and use the following formula. If the target chip has an independent ADC_V_{ref+} pin, you must connect the ADC_V_{ref+} tube. The pin is connected to the V_{cc} pin.

$$\text{MCU working voltage } V_{cc} = 1024 \mid \frac{\text{input voltage } V_{in} \text{ of the ADC converted channel}}{\text{10bit ADC conversion result}}$$

$$\text{MCU working voltage } V_{cc} = 4096 \mid \frac{\text{input voltage } V_{in} \text{ of the ADC converted channel}}{\text{12bit ADC conversion result}}$$

17.3 10 BIT ADC Static Characteristics

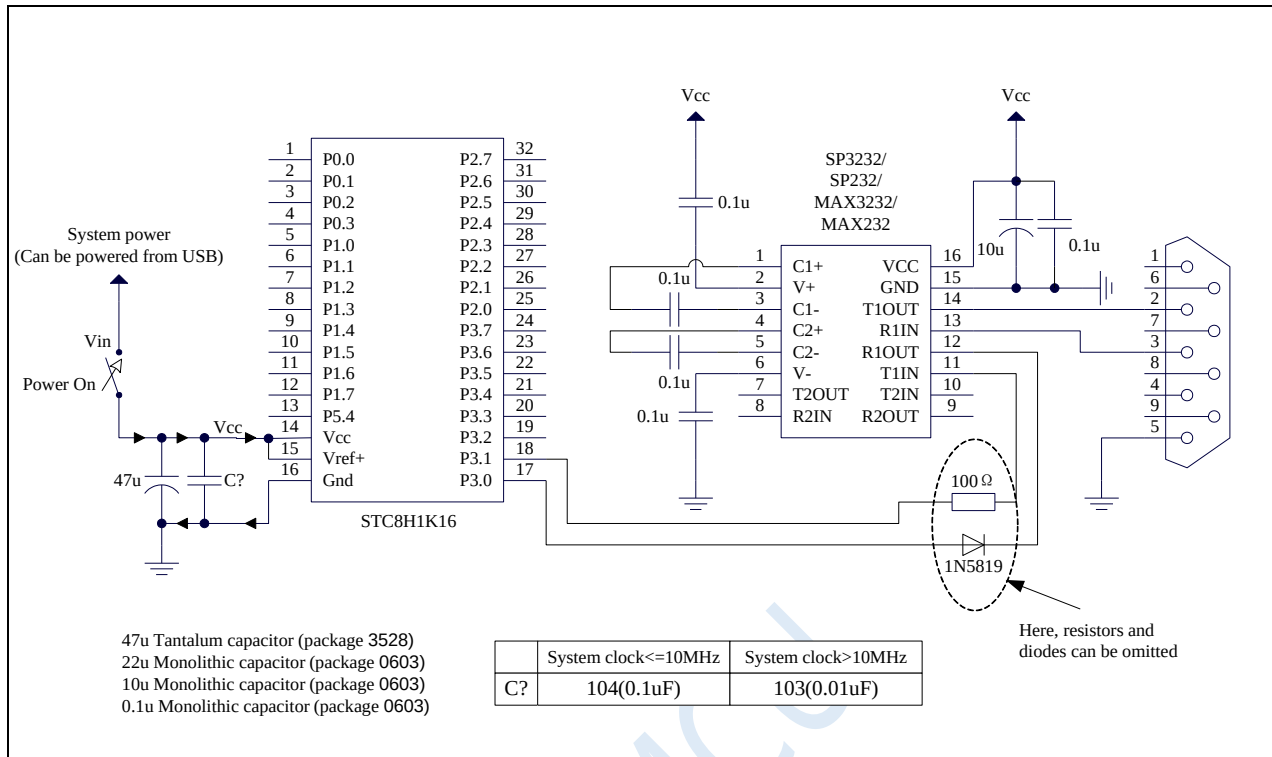
Symbol	Description	Minimum	Typical	Max	Unit
RES	Resolution	-	10	-	Bits
E _T	Overall error	-	1.3	3	LSB
E _O	Offset error	-	0.3	1	LSB
E _G	Gain error	-	0	1	LSB
E _D	Differential nonlinear error	-	0.7	1.5	LSB
E _I	Integral nonlinear error	-	1	2	LSB
R _{AIN}	Channel equivalent resistance	-	∞	-	Ohm
R _{ESD}	Antistatic resistance connected in series before the sample and hold capacitor	-	700	-	Ohm
C _{ADC}	Internal sample and hold capacitor	-	16.5	-	pF

17.4 12 BIT ADC Static Characteristics

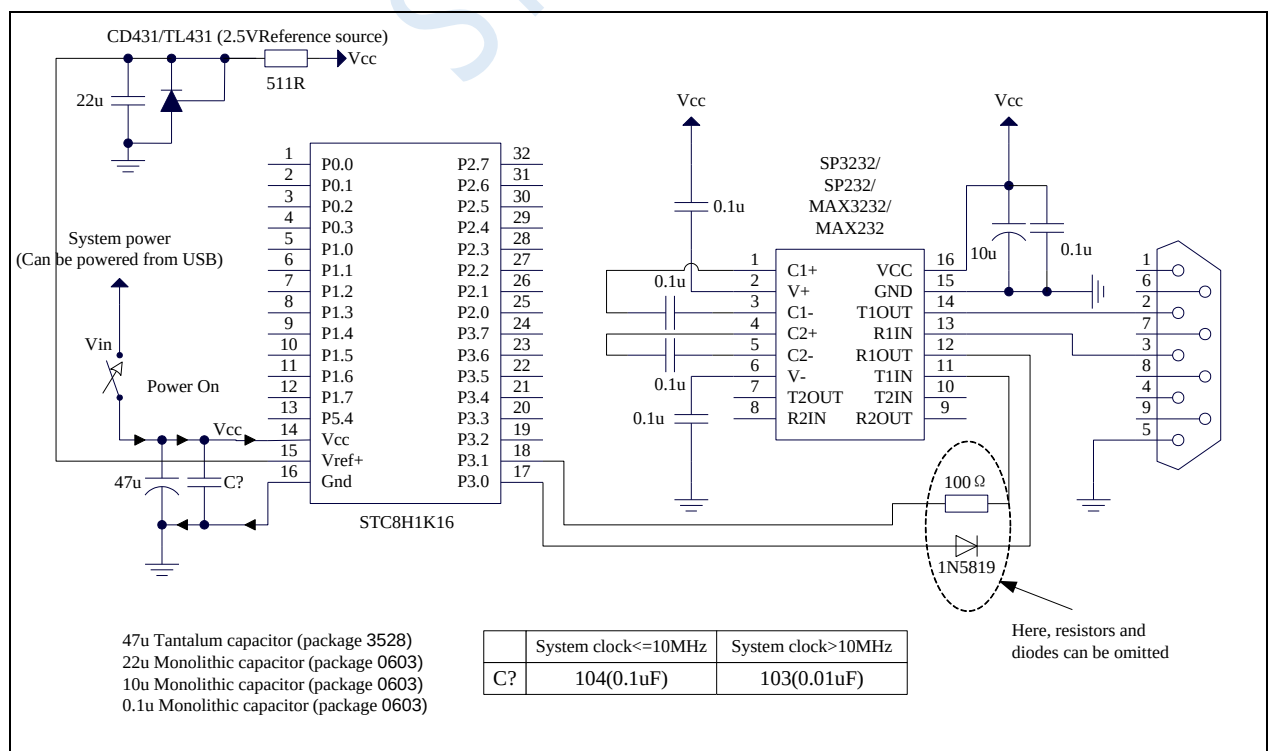
Symbol	Description	Minimum	Typical	Max	Unit
RES	Resolution	-	12	-	Bits
E _T	Overall error	-	0.5	1	LSB
E _O	Offset error	-	-0.1	1	LSB
E _G	Gain error	-	0	1	LSB
E _D	Differential nonlinear error	-	0.7	1.5	LSB
E _I	Integral nonlinear error	-	1	2	LSB
R _{AIN}	Channel equivalent resistance	-	∞	-	Ohm
R _{ESD}	Antistatic resistance connected in series before the sample and hold capacitor	-	700	-	Ohm
C _{ADC}	Internal sample and hold capacitor	-	16.5	-	pF

17.5 ADC application reference circuit diagram

17.5.1 General precision ADC reference circuit diagram



17.5.2 High-precision ADC reference circuit diagram



17.6 Example Routines

17.6.1 ADC Basic Operation (Polling Mode)

C language code

//Operating frequency for test is 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 *)0xfea8)
```

```
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;
```

```
    P1M0 = 0x00;
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;
    ADC_CONTR = 0x80;
```

//Set P1.0 as ADC input

// Set ADC internal timing

//Set the ADC clock to the system clock/2/16

//Enable ADC module

```

while (1)
{
    ADC_CONTR |= 0x40;                //Start AD conversion
    _nop_0;
    _nop_0;
    while (!(ADC_CONTR & 0x20));      //Query ADC completion flag
    ADC_CONTR &= ~0x20;              //Clear completion flag
    P2 = ADC_RES;                    //Read ADC results
}
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

```

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

```

```

P_SW2      DATA    0BAH
ADCTIM     XDATA    0FEA8H

```

```

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

```

```

ORG        0000H
LJMP       MAIN

```

```

MAIN:      ORG        0100H

```

```

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

```

```

MOV        P1M0, #00H    ;Set P1.0 as ADC input
MOV        P1M1, #01H
MOV        P_SW2, #80H
MOV        DPTR, #ADCTIM ; Set ADC internal timing

```

```

MOV      A,#3FH
MOVX     @DPTR,A
MOV      P_SW2,#00H
MOV      ADCCFG,#0FH      ;Set the ADC clock to the system clock/2/16
MOV      ADC_CONTR,#80H   ;Enable ADC module

LOOP:
ORL      ADC_CONTR,#40H    ;Start AD conversion
NOP
NOP
MOV      A,ADC_CONTR      ;Query ADC completion flag
JNB      ACC.5,$-2
ANL      ADC_CONTR,#NOT 20H ;Clear completion flag
MOV      P2,ADC_RES       ;Read ADC results

SJMP     LOOP

END

```

17.6.2 ADC Basic Operation (Interrupt Mode)

C language code

//Operating frequency for test is 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      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 ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;      //Clear interrupt flag
    P2 = ADC_RES;           //Read ADC results
    ADC_CONTR |= 0x40;      //Continue AD conversion
}

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;                //Set P1.0 as ADC input
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;              // Set ADC internal timing
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;              //Set the ADC clock to the system clock/2/16
    ADC_CONTR = 0x80;           //Enable ADC module
    EADC = 1;                   //Enable ADC interrupt
    EA = 1;
    ADC_CONTR |= 0x40;          //Start AD conversion

    while (1);
}

```

Assembly code

;Operating frequency for test is 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
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
ORG		0000H
LJMP		MAIN
ORG		002BH
LJMP		ADCISR

```

ORG          0100H

ADCISR:
ANL          ADC_CONTR,#NOT 20H      ;Clear completion flag
MOV          P2,ADC_RES                ;Read ADC results
ORL          ADC_CONTR,#40H            ;Continue AD conversion
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                ;Set P1.0 as ADC input
MOV          P1M1,#01H
MOV          P_SW2,#80H
MOV          DPTR,#ADCTIM              ; Set ADC internal timing
MOV          A,#3FH
MOVX         @DPTR,A
MOV          P_SW2,#00H
MOV          ADCCFG,#0FH              ;Set the ADC clock to the system clock/2/16
MOV          ADC_CONTR,#80H            ;Enable ADC module
SETB         EADC                      ;Enable ADC interrupt
SETB         EA
ORL          ADC_CONTR,#40H            ;Start AD conversion

SJMP         $

END

```

17.6.3 Format ADC Conversion Result

C language code

//Operating frequency for test is 11.0592MHz

```

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

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

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

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;

    P1M0 = 0x00;           //Set P1.0 as ADC input
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;         //Set ADC internal timing
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;         //Set the ADC clock to the system clock/2/16/16
    ADC_CONTR = 0x80;      //Enable ADC module
    ADC_CONTR |= 0x40;     //Start AD conversion
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20)); //Query ADC completion flag
    ADC_CONTR &= ~0x20;      //Clear completion flag

    ADCCFG = 0x00;         //Set result to align left
    ACC = ADC_RES;          //A stores the upper 8 bits of the ADC's 10-bit result
    B = ADC_RES;            //B[7: 6] stores the lower 2 bits of the 10-bit ADC result, B [5: 0] is 0

    // ADCCFG = 0x20;       //Set result to align right
    // ACC = ADC_RES;       // A [1: 0] stores the upper 2 bits of the 10-bit result of the ADC, and A [7: 2] is 0
    // B = ADC_RES;         //B stores the lower 8 bits of the ADC's 10-bit result

    while (1);
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

```

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

P_SW2       DATA      0BAH

```



```

ADCTIM      XDATA      0FEA8H

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

                ORG      0000H
                LJMP     MAIN

MAIN:         ORG      0100H

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

                MOV      P1M0, #00H                ;Set P1.0 as ADC input
                MOV      P1M1, #01H
                MOV      P_SW2, #80H
                MOV      DPTR, #ADCTIM            ; Set ADC internal timing
                MOV      A, #3FH
                MOVX     @DPTR, A
                MOV      P_SW2, #00H
                MOV      ADCCFG, #0FH            ;Set the ADC clock to the system clock/2/16
                MOV      ADC_CONTR, #80H         ;Enable ADC module

                ORL      ADC_CONTR, #40H         ;Start AD conversion
                NOP
                NOP
                MOV      A, ADC_CONTR            ;Query ADC completion flag
                JNB      ACC.5, $-2
                ANL      ADC_CONTR, #NOT 20H     ;Clear completion flag

                MOV      ADCCFG, #00H            ;Set result to align left
                MOV      A, ADC_RES              ;A stores the upper 8 bits of the ADC's 10-bit result
                MOV      B, ADC_RESL            ;B[7: 6] stores the lower 2 bits of the 10-bit ADC result, B [5: 0] is 0

;                MOV      ADCCFG, #20H            ;Set result to align right
;                MOV      A, ADC_RES ;A [3: 0] stores the upper 2 bits of the 10-bit result of the ADC, and A [7: 2] is 0
;                MOV      B, ADC_RESL            ;B stores the lower 8 bits of the ADC's 10-bit result

```

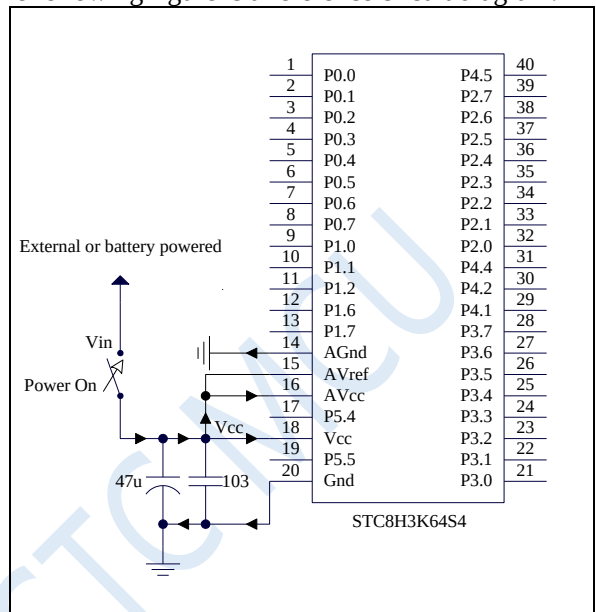
SJMP *\$*

END

17.6.4 Detect External Voltage or Battery Voltage using ADC 15th Channel

The 15th channel of ADC in the STC8H family of microcontrollers is used to measure the internal reference voltage. The internal reference voltage is stable, about 1.19V, and does not change with the chip's working voltage. So you can measure the internal reference voltage and use it to deduce the external voltage or external battery voltage through the value of the ADC.

The following figure is a reference circuit diagram:



C language code

//Operating frequency for test is 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 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;

```

```

int      *BGV;

//The internal reference voltage value is stored in idata
//The high byte is stored in idata's EFH address
//The low byte is stored in idata's F0H address
//Voltage unit is millivolt (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;

    // Set ADC internal timing

    ADCCFG = 0x2f;
    ADC_CONTR = 0x8f;

    //Set the ADC clock to the system clock/2/16
    //Enable ADC module, and select channel 15
}

```

```

int  ADCRead()

```

```

{
    int res;

    ADC_CONTR |= 0x40;

    //Start AD conversion
}

```

```

    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));           //Query ADC completion flag
    ADC_CONTR &= ~0x20;                   //Clear completion flag
    res = (ADC_RES << 8) | ADC_RES1;      //Read ADC results

    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 initialization
    UartInit();                            //UART initialization

    ES = 1;
    EA = 1;

    //  ADCRead();
    //  ADCRead();                         //Discard the first two data

    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();                 //Read data 8 times
    }
    res >>= 3;                             //take the average

    vcc = (int)(4096L * *BGV / res);        // (12-bit ADC algorithm)Calculate VREF pin voltage, i.e.
    battery voltage
    //  vcc = (int)(1024L * *BGV / res);    // (10-bit ADC algorithm)Calculate VREF pin voltage, i.e.
    battery voltage

    //Note that this voltage is in millivolts (mV)
    UartSend(vcc >> 8);                   //Output voltage value to UART
    UartSend(vcc);

    while (1);
}

```

The method above uses the 15th channel of the ADC to invert the external battery voltage. In the ADC measurement range, the external measurement voltage of the ADC is proportional to the measurement value of

the ADC, so the 15th channel of the ADC can also be used to reverse the input voltage of the external channel. Assuming that the current internal reference signal source voltage is BGV , the ADC measurement value of the internal reference signal source is res_{bg} , and the ADC measurement value of the external channel input voltage is res_x , then the external channel input voltage $V_x = BGV / res_{bg} * res_x$;

17.6.5 Using ADC as Capacitive Sensing Touch Keys

Key is one of the most commonly used parts in the circuit, and it is an important input method for the human-machine interface. We are most familiar with mechanical keys. The mechanical keys have a disadvantage of limited contact life especially for the cheap keys. And they are easy to appear poor contact and failure. Non-contact keys have no mechanical contacts, long life and easy to use.

There are various solutions for non-contact keys. Capacitive-sensing keys are low-cost solutions. Specialized ICs were used to implement capacitive-sensing keys many years ago. With the enhancement of MCU functions and the practical experience of users, MCUs were used to implement capacitive-sensing keys directly. The technology of capacitive sensing keys is mature. The most typical and reliable one is the solution using ADC.

The solution of using STC series MCUs with ADC is described in detail in this document. Any MCU with ADC function can be used to implement the scheme. The first three diagrams below are the most commonly used methods. The principles are the same. The second diagram is used.

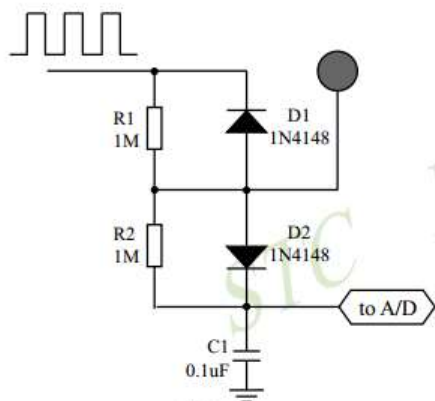


图1

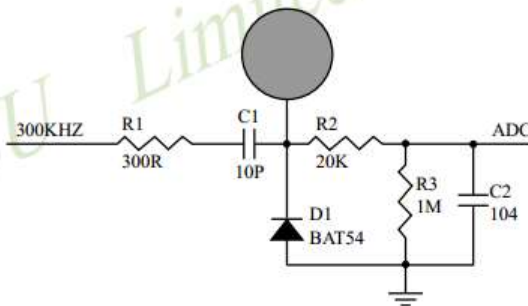


图2

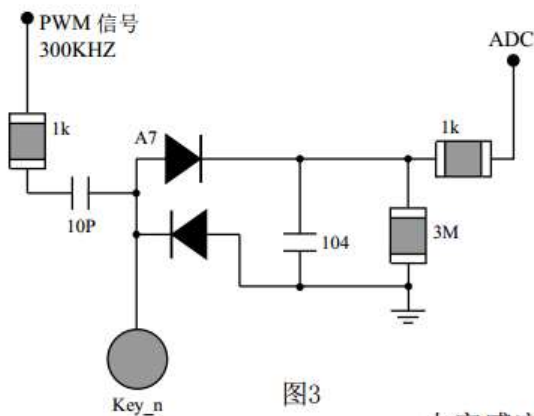


图3

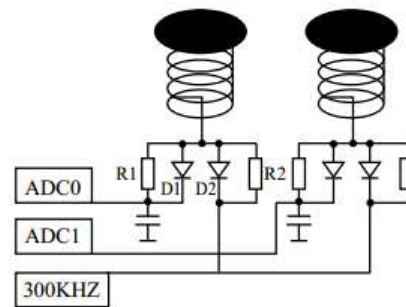
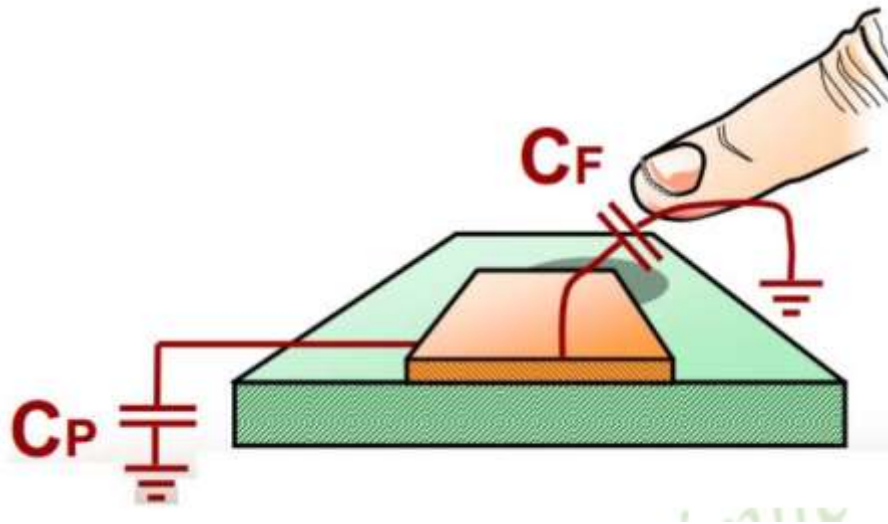


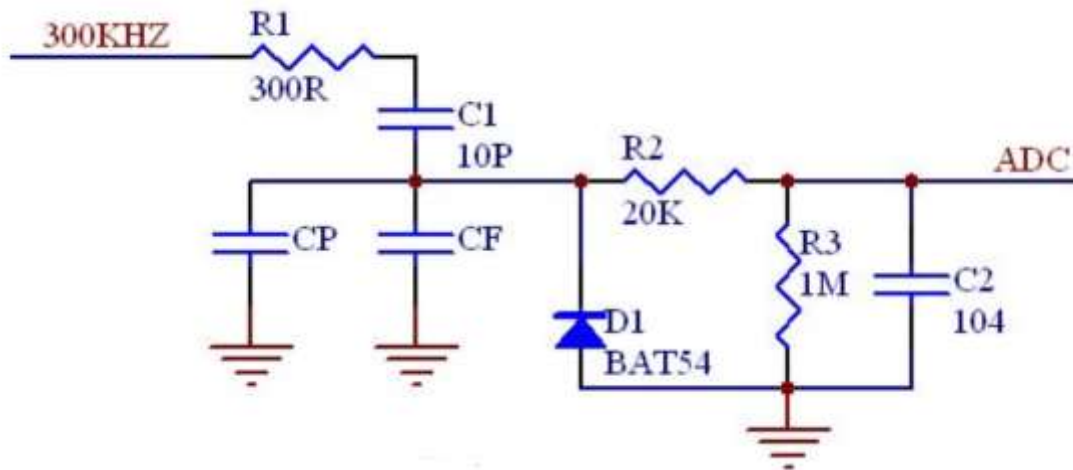
图4 加了感应弹簧

电容感应按键取样电路

In general applications, the induction spring shown in Figure 4 is used to increase the area pressed by a finger. The induction spring is equivalent to a metal plate to the ground. There is a capacitor C_P to the ground. After pressing the finger, a capacitor C_F is connected in parallel to the ground, as shown in the figure below.



The following is the description of the circuit diagram. CP is the distributed capacitance of metal plate and ground, CF is the finger capacitance, they are connected in parallel and connected with C1 to divide the input 300KHZ square wave. After being rectified by D1 and filtered by R2 and C2, the wave is sent to ADC. After pressing the finger, the voltage sent to the ADC decreases, and the program can detect the key action.



C language code

//Operating frequency for testing is 24MHz

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

```
#define MAIN_Fosc 24000000UL //Define the main clock
#define Timer0_Reload (65536UL -(MAIN_Fosc / 600000)) //Timer 0 reload value corresponds to 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;           // microcontrollers with ADC
sfr      ADC_RES   = 0xBD;           / microcontrollers with ADC
sfr      ADC_RESL   = 0xBE;           // microcontrollers with ADC
sfr      AUXR      = 0x8E;
sfr      AUXR2     = 0x8F;

#define    CHANNEL    8               //ADC channel numbers
#define    ADC_90T    (3<<5)         //ADC conversion time 90T
#define    ADC_180T   (2<<5)         //ADC conversion time 180T
#define    ADC_360T   (1<<5)         //ADC conversion time 360T
#define    ADC_540T   0               //ADC conversion time 540T
#define    ADC_FLAG   (1<<4)         //Cleared by software
#define    ADC_START   (1<<3)        //Cleared automatically

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];        //Current ADC value
u16 idata adc_prev[TOUCH_CHANNEL];    //Previous ADC value
u16 idata TouchZero[TOUCH_CHANNEL];   //ADC value of 0-point
u8 idata TouchZeroCnt[TOUCH_CHANNEL]; //Automatic tracking count for 0-point
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;                               //Initialize Timer0 to output a 300KHz clock

```

```

    TR0 = 0;
    AUXR |= 0x80; //Timer0 set as 1T mode
    AUXR2 |= 0x01; //Enable clock output
    TMOD = 0; //Timer0 set as Timer, 16 bits Auto Reload.
    TH0 = (u8)(Timer0_Reload >> 8);
    TL0 = (u8)Timer0_Reload;
    TR0 = 1;
    ADC_init(); //ADC initialization
    delay_ms(50); //Delay 50ms
    for (i=0; i<TOUCH_CHANNEL; i++) // Initialize the 0-point, the previous value and the 0-point auto-tracking count
    {
        adc_prev[i] = 1023;
        TouchZero[i] = 1023;
        TouchZeroCnt[i] = 0;
    }
    cnt_250ms = 0;
    while (1)
    {
        delay_ms(50); //Process key once every 50ms
        ShowLED();
        if (++cnt_250ms >= 5)
        {
            cnt_250ms = 0;
            AutoZero(); //Process 0-point auto-tracking every 250ms
        }
    }
}

void delay_ms(u8 ms)
{
    unsigned int i;

    do
    {
        i = MAIN_Fosc / 13000;
        while(--i);
    } while(--ms);
}

void ADC_init(void)
{
    P1M0 = 0x00; //8 channels ADC
    P1M1 = 0xff;
    ADC_CONTR = 0x80; //Enable ADC
}

u16 Get_ADC10bitResult(u8 channel)
{
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_90T | ADC_START | channel; //Trigger ADC
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    while((ADC_CONTR & ADC_FLAG) == 0); //Wait for ADC conversion complement
    ADC_CONTR = 0x80; //Clear flag
    return(((u16)ADC_RES << 2) | ((u16)ADC_RESL & 3)); //Return ADC result
}

```



```

void AutoZero(void)                                     //Call once every 250ms
// This is detected using the sum of the absolute values of the differences between two adjacent
samples.
{
    u8 i;
    u16 j,k;

    for(i=0; i<TOUCH_CHANNEL; i++)                     //Process 8 channels
    {
        j = adc[i];
        k = j - adc_prev[i];                           // Subtract previous reading
        F0 = 0;                                         //Pressed
        if(k & 0x8000) F0 = 1, k = 0 - k;               //Release, get the difference between two samples
        if(k >= 20)                                     // Big change
        {
            TouchZeroCnt[i] = 0;                       // If the change is large, clear the counter
            if(F0) TouchZero[i] = j; // If it is released, and the change is relatively large, then directly replace
        }
        else                                           // If the change is relatively small, then creep, track 0-point automatically
        {
            if(++TouchZeroCnt[i] >= 20)                 // Continuously detect small changes 20 times/4 = 5
seconds.
            {
                TouchZeroCnt[i] = 0;
                TouchZero[i] = adc_prev[i];             // Use slowly changing values as 0 points
            }
            adc_prev[i] = j;                             // Save this time's sample value
        }
    }
}

u8 check_adc(u8 index)                                 // Get touch information function, called every 50ms
// Judge key is pressed or released with hysteresis control
{
    u16 delta;

    adc[index] = 1023 - Get_ADC10bitResult(index); // Get ADC value, convert to press the key, ADC value increases
    if(adc[index] < TouchZero[index]) return 0;       // A value smaller than 0-point is considered a key release
    delta = adc[index] - TouchZero[index];
    if(delta >= 40) return 1;                          //Key pressed
    if(delta <= 20) return 0;                          //Key released
    return 2;                                           // Keep the original state
}

void ShowLED(void)
{
    u8 i;

    i = check_adc(0);
    if(i == 0) P_LED0 = 1;                             //Light off
    if(i == 1) P_LED0 = 0;                             //Light on
    i = check_adc(1);
    if(i == 0) P_LED1 = 1;                             //Light off
    if(i == 1) P_LED1 = 0;                             //Light on
    i = check_adc(2);
    if(i == 0) P_LED2 = 1;                             //Light off
    if(i == 1) P_LED2 = 0;                             //Light on
    i = check_adc(3);
    if(i == 0) P_LED3 = 1;                             //Light off

```

```

    if(i == 1) P_LED3 = 0;           //Light on
    i = check_adc(4);
    if(i == 0) P_LED4 = 1;           //Light off
    if(i == 1) P_LED4 = 0;           //Light on
    i = check_adc(5);
    if(i == 0) P_LED5 = 1;           //Light off
    if(i == 1) P_LED5 = 0;           //Light on
    i = check_adc(6);
    if(i == 0) P_LED6 = 1;           //Light off
    if(i == 1) P_LED6 = 0;           //Light on
    i = check_adc(7);
    if(i == 0) P_LED7 = 1;           //Light off
    if(i == 1) P_LED7 = 0;           //Light on
}

```

Assembly code

;Operating frequency for testing is 24MHz

Fosc_KHZ	EQU	24000	;Define the main clock KHZ
Reload	EQU	(65536 - Fosc_KHZ/600)	;Timer 0 reload value, corresponding to 300KHz
ADC_CONTR	DATA	0xBC	; microcontrollers with ADC
ADC_RES	DATA	0xBD	; microcontrollers with ADC
ADC_RESL	DATA	0xBE	; microcontrollers with ADC
AUXR	DATA	0x8E	
AUXR2	DATA	0x8F	
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	
CHANNEL	EQU	8	;ADC channel numbers
ADC_90T	EQU	(3 SHL 5)	;ADC conversion time 90T
ADC_180T	EQU	(2 SHL 5)	;ADC conversion time 180T
ADC_360T	EQU	(1 SHL 5)	;ADC conversion time 360T
ADC_540T	EQU	0	;ADC conversion time 540T
ADC_FLAG	EQU	(1 SHL 4)	;Cleared by software
ADC_START	EQU	(1 SHL 3)	;Cleared automatically
P_LED7	BIT	P2.7;	
P_LED6	BIT	P2.6;	
P_LED5	BIT	P2.5;	
P_LED4	BIT	P2.4;	
P_LED3	BIT	P2.3;	
P_LED2	BIT	P2.2;	
P_LED1	BIT	P2.1;	
P_LED0	BIT	P2.0;	
adc	EQU	30H	; Current ADC value in 30H ~ 3FH, two bytes constitute one value

```

adc_prev    EQU    40H    ; Previous ADC value in 40H ~ 4FH, two bytes constitute a value
TouchZero   EQU    50H    ; ADC 0 value in 50H~5FH, two bytes constitute a value
TouchZeroCnt EQU    60H    ; 0-point automatic tracking count in 60H~67H
cnt_250ms   DATA    68H

                ORG    0000H
                LJMP   MAIN

                ORG    0100H
MAIN:
    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    ;Initialize Timer0 to output a 300KHz clock
    CLR    TR0
    ORL    AUXR,#080H    ;Timer0 set as 1T mode
    ORL    AUXR2,#01H    ;Enable clock output
    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    ;Initialize the previous ADC value

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    ;Initialize the ADC 0-point value

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    ;Initialize the automatic tracking count value

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                ;Delay 50ms
    LCALL      F_delay_ms
    LCALL      F_ShowLED             ;Handle key value once
    DJNZ       cnt_250ms,L_MainLoop
    MOV        cnt_250ms,#5          ;Processing once 0-point automatic tracking value every 250ms
    LCALL      F_AutoZero            ; Zero tracking
    SJMP       L_MainLoop

F_ADC_init:
    MOV        P1M0,#00H             ;8 channels ADC
    MOV        P1M1,#0FFH
    MOV        ADC_CONTR,#080H       ;Enable ADC
    RET

F_Get_ADC10bitResult:
    MOV        ADC_RES,#0
    MOV        ADC_RESL,#0
    MOV        A,R7
    ORL        A,#0E8H               ;Trigger ADC
    MOV        ADC_CONTR,A
    NOP
    NOP
    NOP
    NOP

L_10bitADC_Loop1:
    MOV        A,ADC_CONTR
    JNB        ACC.4,L_10bitADC_Loop1 ;Wait for the ADC conversion complement
    MOV        ADC_CONTR,#080H       ;Clear flag
    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:                          ; Call once every 250ms
    ; This is detected using the sum of the absolute values of the differences between two adjacent samples.
    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                                ;Calculate [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, Jump
    MOV      A,#LOW (TouchZeroCnt)          ; If the change is large, clear the counter 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:                                ; If the change is relatively small, then creep, track 0-point automatically
                                           ; Continuously detect small changes 20 times/4 = 5 seconds.
    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), jump
    MOV      @R0,#0                          ;TouchZeroCnt[i]= 0;
    MOV      A,R5                            ; Use slowly changing values as 0 points
    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:                                ; Save the sampled value 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:                                ; Judge key is pressed or released, with hysteresis control
MOV      R4,7
LCALL    F_Get_ADC10bitResult                ; The ADC value returned is [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                                ;Save 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                                ;Calculate 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     R1,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), jump
    MOV     R7,#1                  ;if(delta >= 40) return 1; //Key pressed, return 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
```



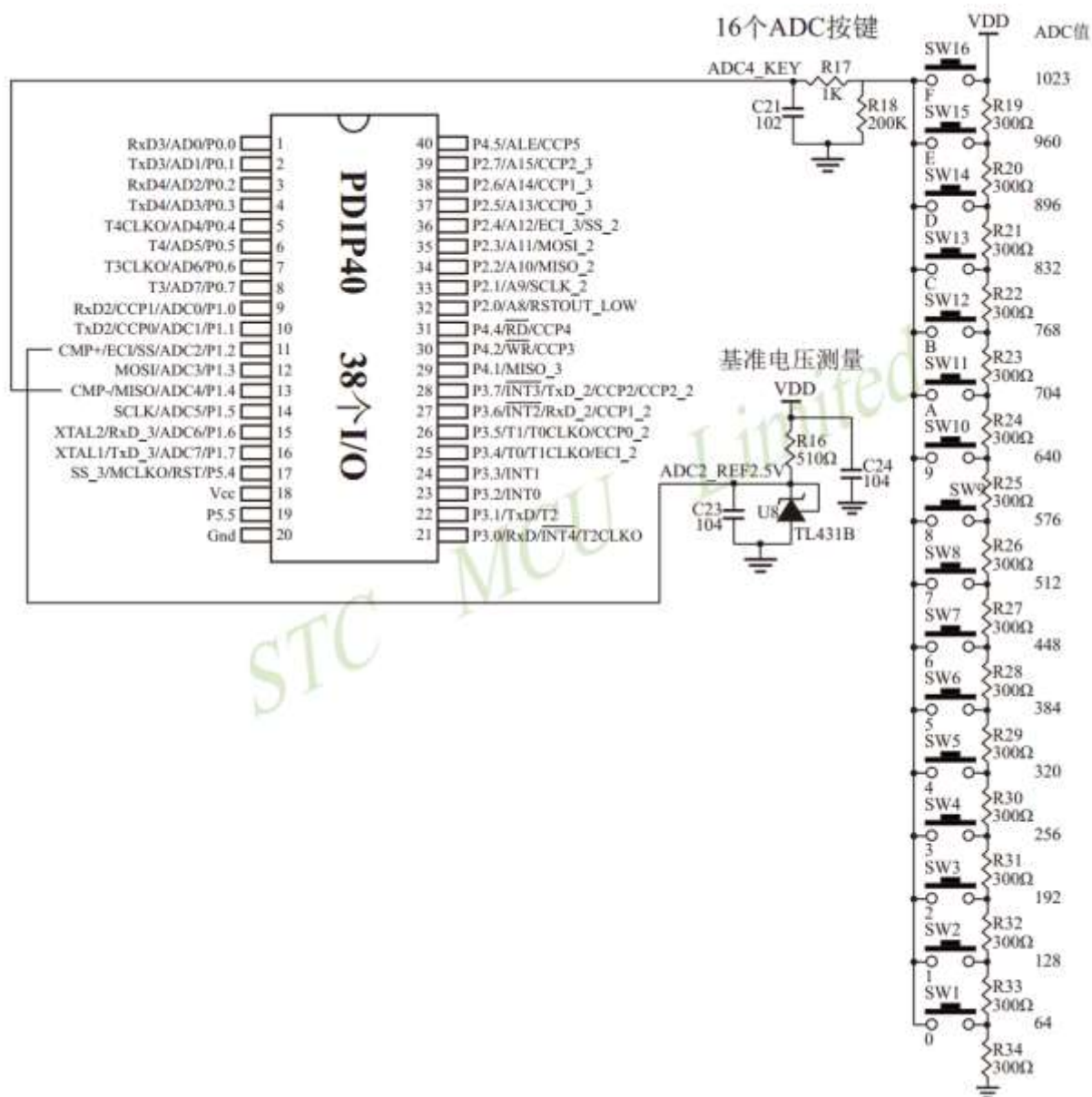
```
      CPL      C
      MOV      P_LED7,C
L_QuitCheck7:
      RET

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

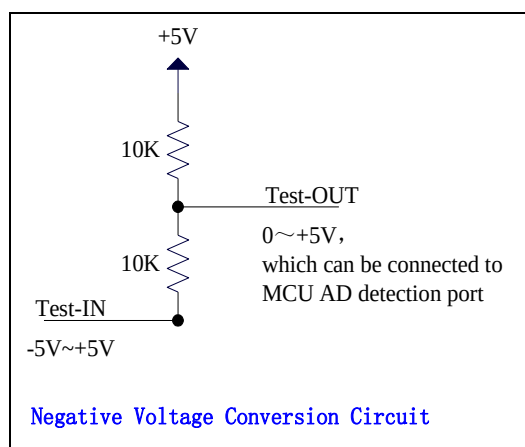
      END
```

17.6.6 Key-scan Application Circuit Diagram using ADC

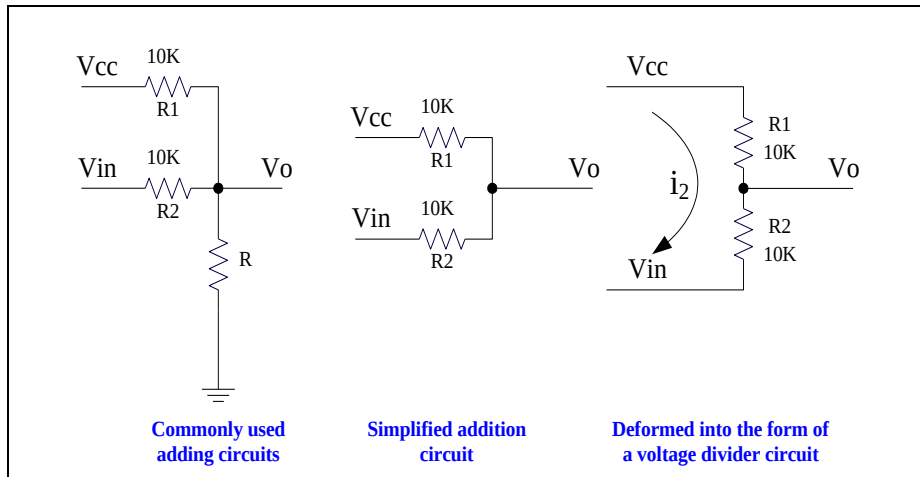
Method for reading the ADC key: Read the ADC value every 10ms or so and save the last 3 readings. If the change is relatively small, judge the key. When the key is judged be valid, a certain deviation is allowed, such as a deviation of ± 16 words.



17.6.7 Reference circuit diagram for detecting negative voltage



17.6.8 The application of common addition circuit in ADC



Refer to the voltage divider circuit to get formula 1

Formula 1: $V_o = V_{in} + i_2 * R_2$

Formula 2: $i_2 = (V_{cc} - V_{in}) / (R_1 + R_2)$ {Condition: the current flowing to $V_o \approx 0$ }

Substituting $R_1 = R_2$ into formula 2 gives formula 3

Formula 3: $i_2 = (V_{cc} - V_{in}) / 2R_2$

Substituting formula 3 into formula 1 gives formula 4

Formula 4: $V_o = (V_{cc} + V_{in}) / 2$

According to formula 4, the above circuit can be regarded as an addition circuit.

In the analog-to-digital conversion measurement of the microcontroller, the measured voltage is required to be greater than 0 and less than VCC. If the measured voltage is less than 0V, an addition circuit can be used to increase the measured voltage to above 0V. At this time, there are certain requirements for the variation range of the measured voltage:

Substituting the above conditions into formula 4, the following formula 2 can be obtained

$(V_{cc} + V_{in}) / 2 > 0$ means $V_{in} > -V_{cc}$

$(V_{cc} + V_{in}) / 2 < V_{cc}$ means $V_{in} < V_{cc}$

The above 2 formulas can be combined: $-V_{cc} < V_{in} < V_{cc}$

18 Synchronous Serial Peripheral Interface (SPI)

Product line	SPI	High speed SPI (the SPI clock is the system clock/2 if SPR=11B)
STC8H1K08 family	●	
STC8H1K28 family	●	
STC8H3K64S4 family A version	●	
STC8H3K64S4 family A version	●	
STC8H3K64S2 family B version		●
STC8H3K64S4 family B version		●
STC8H8K64U family A version	●	
STC8H8K64U family B version		●
STC8H2K64T family	●	
STC8H4K64TLR family		●
STC8H4K64TLCD family		●
STC8H4K64LCD family		●

A high-speed serial communication interface, SPI, is integrated in STC8H series of microcontrollers. SPI is a full-duplex high-speed synchronous communication bus. SPI interface integrated in the STC8H series of microcontrollers offers two operation modes: master mode and slave mode.

18.1 Registers Related to SPI

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI Control Register	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI Data Register	CFH									0000,0000

18.1.1 SPI Status register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI transfer completion flag.

When SPI completes sending / receiving 1 byte of data, the hardware will automatically set this bit and request interrupt to CPU. If the SSIG bit is set to 0, this flag will also be automatically set by hardware to indicate a mode change of device when the master / slave mode of the device changes due to changes in the SS pin level.

Note: This bit must be cleared using software writing 1 to it.

WCOL: SPI write collision flag bit.

This bit is set by hardware when the SPI is writing to the SPDAT register during data transfer.

Note: This bit must be cleared using software by writing 1 to it.

18.1.2 SPI Control Register

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

SSIG: Control bit of whether SS pin is ignored or not.

0: the SS pin decides whether the device is a master or slave.

1: the function of SS pin is ignored. MSTR decides whether the device is a master or slave.

SPEN: SPI enable bit.

0: the SPI is disabled.

1: the SPI is enabled.

DORD: Set the transmitted or received SPI data order.

0: The MSB of the data is transmitted firstly.

1: The LSB of the data is transmitted firstly.

MSTR: Master/Slave mode select bit.

To set the mastert mode:

If SSIG = 0, the SS pin must be high and set MSTR to 1.

If SSIG = 1, it only needs to set MSTR to 1 (ignoring the SS pin level).

To set the slave mode:

If SSIG = 0, the SS pin must be low (regardless of the MSTR bit).

If SSIG = 1, it only needs to set MSTR to 0 (ignoring the SS pin level).

CPOL: SPI clock polarity select bit.

0: SCLK is low when idle. The leading edge of SCLK is the rising edge and the trailing edge is the falling edge.

1: SCLK is high when idle. The leading edge of SCLK is the falling edge and the trailing edge is the rising edge.

CPHA: SPI clock phase select bit.

0: The first bit of datum is driven when SS pin is low. The datum changes on the trailing edge of SCLK and is sampled on the leading edge of SCLK. (SSIG must be 0.)

1: The datum is driven on the leading edge of SCLK, and is sampled on the trailing edge.

SPR[1:0]: SPI clock frequency select bits

SPR[1:0]	SCLK frequency	SCLK frequency of high speed SPI
00	SYSclk/4	SYSclk/4
01	SYSclk/8	SYSclk/8
10	SYSclk/16	SYSclk/16
11	SYSclk/32	SYSclk/2

18.1.3 SPI Data Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

The SPDAT holds the data to be transmitted or the data received.

18.2 SPI Communication Modes

There are three SPI communication modes: single master and single slave mode, dual devices configuration mode (any one of them can be a master or slave), single master and multiple slaves mode.

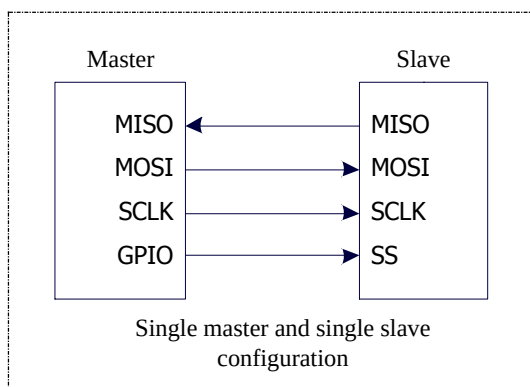
18.2.1 Single Master and Single Slave Mode

Two devices are connected, one of which is fixed as a master and the other as a slave.

Master settings: SSIG set to 1, MSTR set to 1, fixed to be master mode. The master can use any port to connect the slave SS pin, pull down the slave SS pin to enable the slave.

Slave settings: SSIG is set to 0, SS pin as the chip select signal of the slave.

Single master single slave connection configuration diagram is shown as follows:



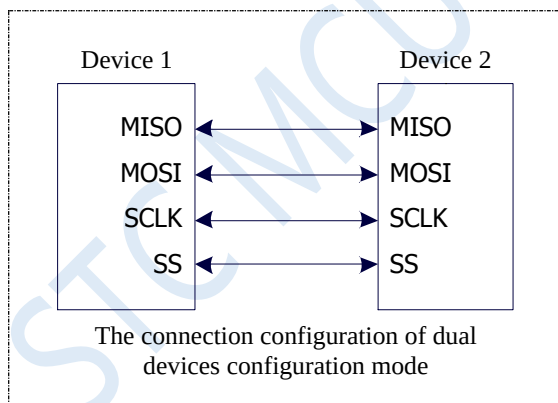
18.2.2 Dual Devices Configuration Mode

Two devices are connected, the master and the slave are not fixed.

Setting Method 1: Both devices are initialized with SSIG set to 0, MSTR set to 1, and SS pin set to bi-directional mode and output high. Now the both devices are in master mode with not ignoring SS. When one of the devices needs to initiate a transfer, set its own SS pin to output mode and output low to pull down the other device's SS pin so that the other device is forcibly set to slave mode.

Set Method 2: Both devices are initialized as slave mode with ignoring SS, where SIG is set to 1 and MSTR is set to 0. When one of the devices needs to initiate a transfer, detect the SS pin's level firstly. If SS is high, the device sets itself to master mode with ignoring SS, then starts the data transfer.

The connection configuration of dual devices configuration mode is shown as follows:



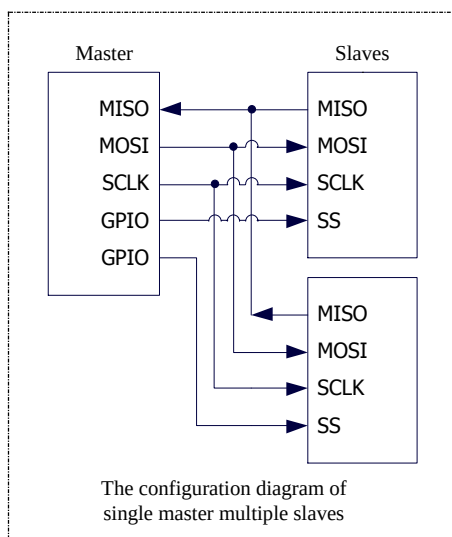
18.2.3 Single Master and Multiple Slaves Mode

Multiple devices are connected, one of which is fixed as a master and others are fixed as slaves.

Master settings: SSIG set to 1, MSTR set to 1, fixed to master mode. The master can use any port to connect with the SS pins of each slave respectively, and pull down the SS pin of one slave to enable the corresponding slave device.

Slave settings: SSIG is set to 0, SS pin is used as the chip select signal of the slave.

The configuration diagram of single master multiple slaves is as follows:



18.3 Configure SPI

Control bits			Communication port pins				Descriptions
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	input	input	input	SPI is disabled, SS/MOSI/MISO/SCLK are used as general I/O ports
1	0	0	0	output	input	input	Selected as slave
1	0	0	1	high impedance	input	input	Selected as slave , not selected.
1	0	1→0	0	output	input	input	Slave mode , master mode with notignoring SS and MSTR is 1. When SS pin is pulled low, MSTR will be automatically cleared by hardware and the operating mode will be passively set to slave mode.
1	0	1	1	input	high impedance	high impedance	Master mode , idle state
					output	output	Master mode , active state
1	1	0	x	output	input	input	Slave mode
1	1	1	x	input	output	output	Master mode

Additional Considerations for a Slave

When CPHA = 0, SSIG must be 0 (i.e. SS pin can not be ignored). The SS pin must be pulled low before each serial byte begins transfer and must be reset to high after the transfer completes. The SPDAT register can not be written while the SS pin is low, otherwise a write collision error will occur. Operation with CPHA = 0 and SSIG = 1 is undefined.

When CPHA = 1, SSIG may be set to 1 (i.e. the SS pin can be ignored). If SSIG = 0, the SS pin may remain active low (i.e., stay low all the way) for consecutive transfers. This method is suitable for fixed single master single slave system.

Additional Considerations for a Master

In SPI, transfers are always initiated by the master. If the SPI is enabled (SPEN = 1) and selected as the master, the master will initiate SPI clock generator and data transfer by writing to SPI data register, SPDAT. The data will appear on the MOSI pin a half to one SPI bit-time later after the data is written to SPDAT. The data written to the SPDAT register of the master is shifted out from the MOSI pin and sent to the MOSI pin of the slave. And, at the same time the data in SPDAT register of the selected slave is shifted out on MISO pin to the MISO pin of the master.

After one byte has been transmitted, the SPI clock generator is stopped, the transfer completion flag (SPIF) is set, and an SPI interrupt is generated if the SPI interrupt is enabled. The two shift registers for the master and slave CPUs can be considered as a 16-bit cyclic shift register. As data is shifted from the master to the slave, data is also shifted in the opposite direction simultaneously. This means that the data of the master and the slave are exchanged with each other in one shift period.

Change Mode using SS pin

If SPEN = 1, SSIG = 0 and MSTR = 1, SPI is enabled in master mode and the SS pin can be configured for input mode or quasi-bidirectional port mode. In this case, another master can drive this pin low to select the device as an SPI slave and send data to it. To avoid bus contention, the SPI system clears the slave's MSTR, forces MOSI and SCLK to be input mode, and MISO changes to output mode. The SPIF flag in SPSTAT is set, and if the SPI interrupt is enabled, an SPI interrupt will occur.

The user software must always detect the MSTR bit. If this bit is cleared by a slave selection action and the user wants to continue using the SPI as a master, the MSTR bit must be set again, otherwise it will remain in slave mode.

Write Collision

The SPI is single buffered in the transmission process and double buffered in receiving process. New data for transmission can not be written to the shift register until the previous transmission is complete. The WCOL bit will be set to indicate that a data write collision error has occurred when the data register SPDAT is written during transmission. In this case, the data currently being transmitted will continue to be transmitted, and the newly written data will be lost.

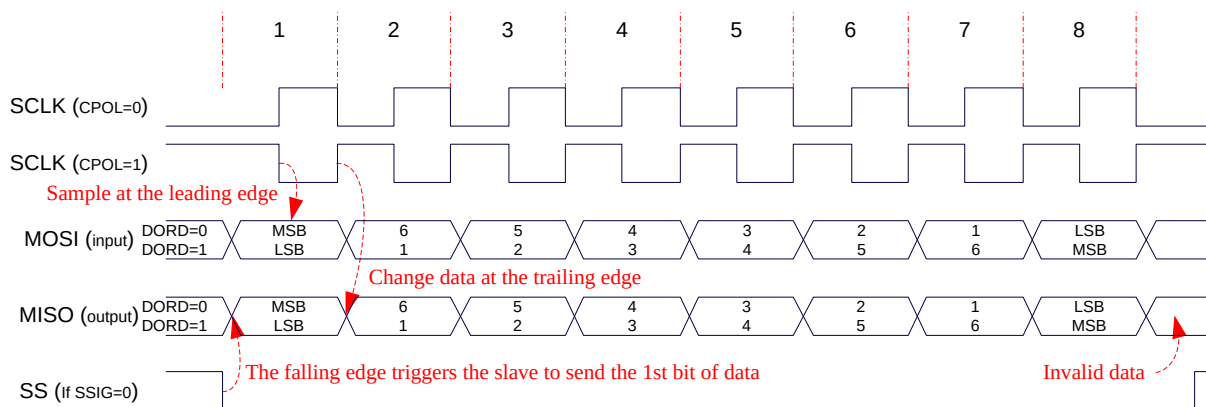
A write collision condition on the master is rare when write collision detection is performed on a master or slave because the master has full control of the data transfer. However, a write collision may occur on the slave because the slave can not control it when the master initiates the transfer.

When receiving data, the received data is transferred to a parallel read data buffer, which will release the shift register for the next data reception. However, the received data must be read from the data register before the next character is completely shifted in. Otherwise, the previous received data will be lost.

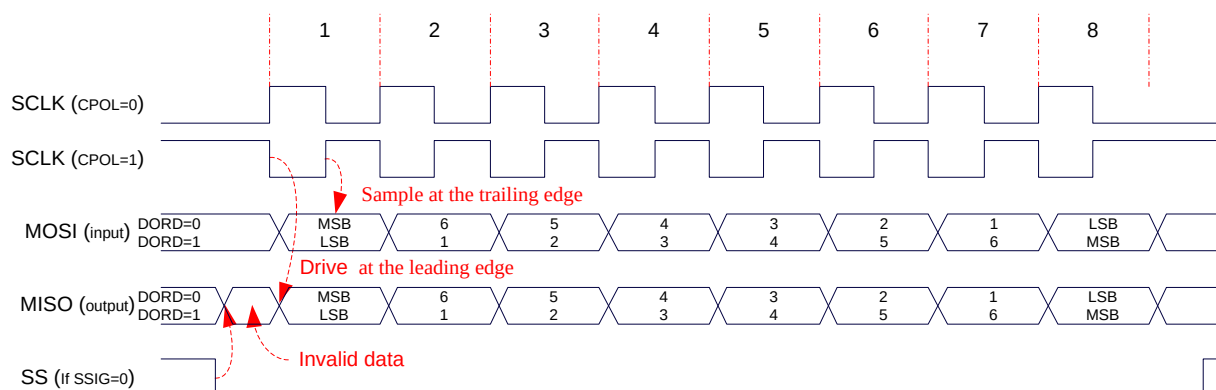
WCOL can be cleared by software by writing "1" to it.

18.4 Data Format

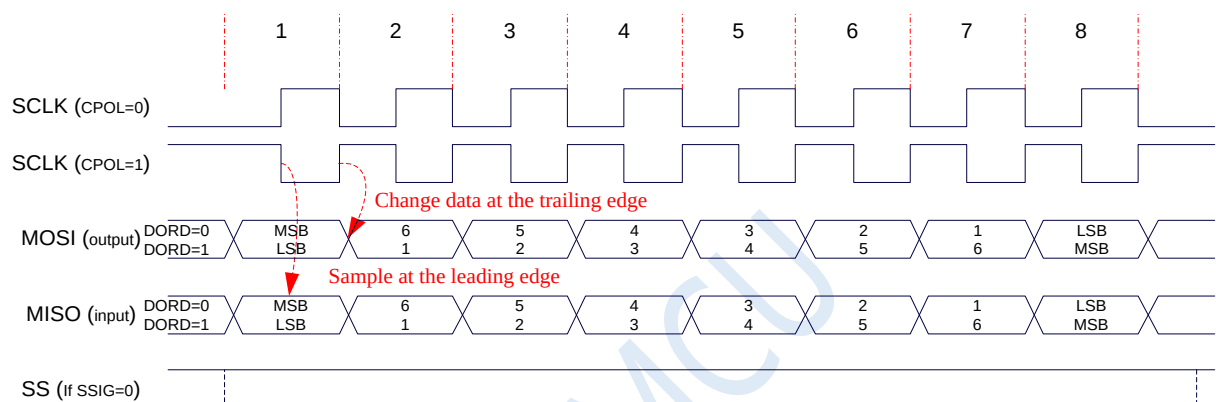
The clock phase control bit, CPHA, of the SPI allows the user to set the clock edge when the data is sampled and changed. The clock polarity bit, CPOL, allows the user to set the clock polarity. The following illustrations show the SPI communication timing under different clock phases and polarity settings.



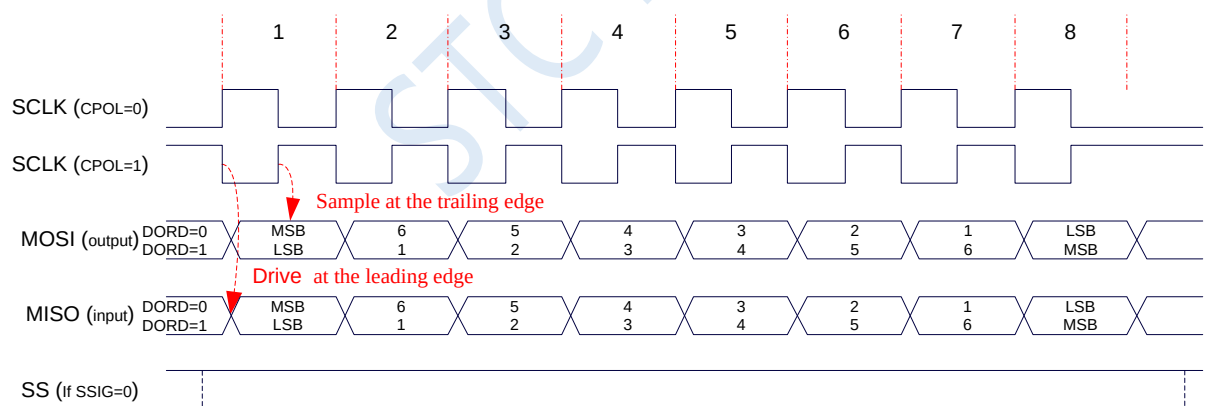
SPI slave transfer format with CPHA=0



SPI slave transfer format with CPHA=1



SPI master transfer format with CPHA=0



SPI master transfer format with CPHA=1

18.5 Example Routines

18.5.1 Master Routine of Single Master Single Slave Mode (Interrupt Mode)

C language code

//Operating frequency for test is 11.0592MHz

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

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

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

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

```
bit       busy;
```

```
void SPI_Isr() interrupt 9
```

```
{
    SPSTAT = 0xc0;           //Clear interrupt flag
    SS = 1;                  //Pull up the SS pin of the slave
    busy = 0;
    LED = !LED;              //Test port
}
```

```
void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
```

```

P5M0 = 0x00;
P5M1 = 0x00;

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

SPCTL = 0x50;           //Enable SPI master mode
SPSTAT = 0xc0;          //Clear interrupt flag
IE2 = ESPI;             //Enable SPI interrupt
EA = 1;

while (1)
{
    while (busy);
    busy = 1;
    SS = 0;              //Pull down the slave SS pin
    SPDAT = 0x5a;        //Send test data
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

SPSTAT	DATA	0CDH	
SPCTL	DATA	0CEH	
SPDAT	DATA	0CFH	
IE2	DATA	0AFH	
ESPI	EQU	02H	
BUSY	BIT	20H.0	
SS	BIT	P1.0	
LED	BIT	P1.1	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	004BH	
	LJMP	SPIISR	
	ORG	0100H	
SPIISR:	MOV	SPSTAT,#0C0H	;Clear interrupt flag
	SETB	SS	;Pull up the SS pin of the slave
	CLR	BUSY	
	CPL	LED	
	RETI		

MAIN:

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

SETB   LED
SETB   SS
CLR     BUSY

MOV     SPCTL, #50H           ;Enable SPI master mode
MOV     SPSTAT, #0C0H        ;Clear interrupt flag
MOV     IE2, #ESPI           ;Enable SPI interrupt
SETB    EA
```

LOOP:

```
JB      BUSY, $
SETB    BUSY
CLR     SS                    ;Pull down the slave SS pin
MOV     SPDAT, #5AH          ;Send test data
JMP     LOOP

END
```

18.5.2 Slave Routine of Single Master Single Slave Mode (Interrupt Mode)

C language code

//Operating frequency for test is 11.0592MHz

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

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

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

```

sfr      P3M1      = 0xb1;
sfr      P3M0      = 0xb2;
sfr      P4M1      = 0xb3;
sfr      P4M0      = 0xb4;
sfr      P5M1      = 0xc9;
sfr      P5M0      = 0xca;

sbit     LED       = P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;           //Clear interrupt flag
    SPDAT = SPDAT;          //Transmit the received data back to the master
    LED = !LED;              //Test port
}

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

    SPCTL = 0x40;            //Enable SPI slave mode
    SPSTAT = 0xc0;          //Clear interrupt flag
    IE2 = ESPI;              //Enable SPI interrupt
    EA = 1;

    while (1);
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

SPSTAT	DATA	0CDH
SPCTL	DATA	0CEH
SPDAT	DATA	0CFH
IE2	DATA	0AFH
ESPI	EQU	02H
LED	BIT	P1.1
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H

```

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

          ORG         0000H
          LJMP        MAIN
          ORG         004BH
          LJMP        SPIISR

          ORG         0100H
SPIISR:
          MOV         SPSTAT,#0C0H      ;Clear interrupt flag
          MOV         SPDAT,SPDAT      ;Transmit the received data back to the master
          CPL         LED
          RETI

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

          MOV         SPCTL,#40H      ;Enable SPI slave mode
          MOV         SPSTAT,#0C0H    ;Clear interrupt flag
          MOV         IE2,#ESPI      ;Enable SPI interrupt
          SETB        EA

          JMP         $

          END

```

18.5.3 Master Routine of Single Master Single Slave Mode (Polling Mode)

C language code

//Operating frequency for test is 11.0592MHz

```

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

```

```

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

```

```

sfr      P3M1      = 0xb1;
sfr      P3M0      = 0xb2;
sfr      P4M1      = 0xb3;
sfr      P4M0      = 0xb4;
sfr      P5M1      = 0xc9;
sfr      P5M0      = 0xca;

```

```

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

```

```

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

```

```
void main()
```

```

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

    LED = 1;
    SS = 1;

    SPCTL = 0x50;           //Enable SPI master mode
    SPSTAT = 0xc0;         //Clear interrupt flag

    while (1)
    {
        SS = 0;           //Pull down the slave SS pin
        SPDAT = 0x5a;      //Send test data
        while (!(SPSTAT & 0x80)); //Query completion flag
        SPSTAT = 0xc0;     //Clear interrupt flag
        SS = 1;           //Pull up the SS pin of the slave
        LED = !LED;       //Test port
    }
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

```

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

```

```

SS         BIT      P1.0
LED        BIT      P1.1

```

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

ORG	0000H
LJMP	MAIN

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

SETB	LED
SETB	SS

MOV	SPCTL, #50H	<i>;Enable SPI master mode</i>
MOV	SPSTAT, #0C0H	<i>;Clear interrupt flag</i>

LOOP:

CLR	SS	<i>;Pull down the slave SS pin</i>
MOV	SPDAT, #5AH	<i>;Send test data</i>
MOV	A, SPSTAT	<i>;Query completion flag</i>
JNB	ACC.7, \$-2	
MOV	SPSTAT, #0C0H	<i>;Clear interrupt flag</i>
SETB	SS	
CPL	LED	
JMP	LOOP	

END

18.5.4 Slave Routine of Single Master Single Slave Mode (Polling Mode)

C language code

//Operating frequency for test is 11.0592MHz

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

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

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

```
sbit     LED       = P1^1;
```

```
void SPI_Isr() interrupt 9
```

```
{
    SPSTAT = 0xc0;           //Clear interrupt flag
}
```

```
void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;           //Enable SPI slave mode
    SPSTAT = 0xc0;         //Clear interrupt flag

    while (1)
    {
        while (!(SPSTAT & 0x80)); //Query completion flag
        SPSTAT = 0xc0;           //Clear interrupt flag
        SPDAT = SPDAT;           //Transmit the received data back to the master
        LED = !LED;              //Test port
    }
}
```

Assembly code

;Operating frequency for test is 11.0592MHz

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

```
LED       BIT      P1.1
```

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

```
ORG       0000H
LJMP      MAIN
```

```
MAIN:     ORG      0100H
```

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

```
MOV       SPCTL, #40H    ;Enable SPI slave mode
MOV       SPSTAT, #0C0H  ;Clear interrupt flag
```

LOOP:

```
MOV       A, SPSTAT      ;Query completion flag
JNB       ACC.7, $-2
MOV       SPSTAT, #0C0H  ;Clear interrupt flag
MOV       SPDAT, SPDAT   ;Transmit the received data back to the master
CPL       LED
JMP       LOOP
```

```
END
```

18.5.5 Routine of Mutual Master-Slave Mode (Interrupt Mode)

C language code

//Operating frequency for test is 11.0592MHz

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

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

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

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

```
void SPI_Isr() interrupt 9
```

```
{
    SPSTAT = 0xc0;           //Clear interrupt flag
    if (SPCTL & 0x10)
    {
        SS = 1;             //Master mode
        SPCTL = 0x40;        //Pull up the SS pin of the slave
        //Reset to slave and standby
    }
    else
    {
        SPDAT = SPDAT;       //Slave mode
        //Transmit the received data back to the master
    }
    LED = !LED;              //Test port
}
```

```
void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
```

```

P5M0 = 0x00;
P5M1 = 0x00;

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

SPCTL = 0x40;           //Enable SPI slave mode and standby
SPSTAT = 0xc0;          //Clear interrupt flag
IE2 = ESPI;             //Enable SPI interrupt
EA = 1;

while (1)
{
    if (!KEY)            //Wait for the key to trigger
    {
        SPCTL = 0x50;    //Enable SPI master mode
        SS = 0;          //Pull down the slave SS pin
        SPDAT = 0x5a;    //Send test data
        while (!KEY);    //Wait for the keys to be released
    }
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

SPSTAT	DATA	0CDH
SPCTL	DATA	0CEH
SPDAT	DATA	0CFH
IE2	DATA	0AFH
ESPI	EQU	02H
SS	BIT	P1.0
LED	BIT	P1.1
KEY	BIT	P0.0
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
	ORG	0000H
	LJMP	MAIN
	ORG	004BH
	LJMP	SPIISR
	ORG	0100H
SPIISR:		
	PUSH	ACC
	MOV	SPSTAT,#0C0H

;Clear interrupt flag

```

        MOV     A,SPCTL
        JB      ACC.4,MASTER

SLAVE:
        MOV     SPDAT,SPDAT      ;Transmit the received data back to the master
        JMP     ISREXIT

MASTER:
        SETB    SS                ;Pull up the SS pin of the slave
        MOV     SPCTL,#40H        ;Reset to slave and standby

ISREXIT:
        CPL     LED
        POP     ACC
        RETI

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

        SETB    SS
        SETB    LED
        SETB    KEY

        MOV     SPCTL,#40H        ;Enable SPI slave mode and standby
        MOV     SPSTAT,#0C0H      ;Clear interrupt flag
        MOV     IE2,#ESPI        ;Enable SPI interrupt
        SETB    EA

LOOP:
        JB      KEY,LOOP          ;Wait for the key to trigger
        MOV     SPCTL,#50H        ;Enable SPI master mode
        CLR     SS                ;Pull down the slave SS pin
        MOV     SPDAT,#5AH        ;Send test data
        JNB     KEY,$             ;Wait for the keys to be released
        JMP     LOOP

        END

```

18.5.6 Routine of Mutual Master-Slave Mode (Polling Mode)

C language code

//Operating frequency for test is 11.0592MHz

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

```
sfr     SPSTAT    = 0xcd;
sfr     SPCTL     = 0xce;
```

```

sfr      SPDAT      = 0xcf;
sfr      IE2        = 0xaf;
#define   ESPI       0x02

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

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

```

```
void main()
```

```

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

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

    SPCTL = 0x40;           //Enable SPI slave mode and standby
    SPSTAT = 0xc0;         //Clear interrupt flag

    while (1)
    {
        if (!KEY)           //Wait for the key to trigger
        {
            SPCTL = 0x50;    //Enable SPI master mode
            SS = 0;          //Pull down the slave SS pin
            SPDAT = 0x5a;    //Send test data
            while (!KEY);    //Wait for the keys to be released
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;   //Clear interrupt flag
            if (SPCTL & 0x10)
            {
                SS = 1;      //Master mode
                             //Pull up the SS pin of the slave
                SPCTL = 0x40; //Reset to slave and standby
            }
        }
    }
}

```

```

    }
    else
    {
        SPDAT = SPDAT;           //Slave mode
        //Transmit the received data back to the master
    }
    LED = !LED;                 //Test port
}
}
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

SPSTAT	DATA	0CDH
SPCTL	DATA	0CEH
SPDAT	DATA	0CFH
IE2	DATA	0AFH
ESPI	EQU	02H
SS	BIT	P1.0
LED	BIT	P1.1
KEY	BIT	P0.0
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H
P2M1	DATA	095H
P2M0	DATA	096H
P3M1	DATA	0B1H
P3M0	DATA	0B2H
P4M1	DATA	0B3H
P4M0	DATA	0B4H
P5M1	DATA	0C9H
P5M0	DATA	0CAH
	ORG	0000H
	LJMP	MAIN
	ORG	0100H
MAIN:	MOV	SP, #5FH
	MOV	P0M0, #00H
	MOV	P0M1, #00H
	MOV	P1M0, #00H
	MOV	P1M1, #00H
	MOV	P2M0, #00H
	MOV	P2M1, #00H
	MOV	P3M0, #00H
	MOV	P3M1, #00H
	MOV	P4M0, #00H
	MOV	P4M1, #00H
	MOV	P5M0, #00H
	MOV	P5M1, #00H
	SETB	SS
	SETB	LED
	SETB	KEY

	MOV	SPCTL,#40H	;Enable SPI slave mode and standby
	MOV	SPSTAT,#0C0H	;Clear interrupt flag
LOOP:			
	JB	KEY,SKIP	;Wait for the key to trigger
	MOV	SPCTL,#50H	;Enable SPI master mode
	CLR	SS	;Pull down the slave SS pin
	MOV	SPDAT,#5AH	;Send test data
	JNB	KEY,\$	;Wait for the keys to be released
SKIP:			
	MOV	A,SPSTAT	
	JNB	ACC.7,LOOP	
	MOV	SPSTAT,#0C0H	;Clear interrupt flag
	MOV	A,SPCTL	
	JB	ACC.4,MASTER	
SLAVE:			
	MOV	SPDAT,SPDAT	;Transmit the received data back to the master
	CPL	LED	
	JMP	LOOP	
MASTER:			
	SETB	SS	;Pull up the SS pin of the slave
	MOV	SPCTL,#40H	;Reset to slave and standby
	CPL	LED	
	JMP	LOOP	
	END		

19 I2C Bus

Product line	I2C
STC8H1K08 family	●
STC8H1K28 family	●
STC8H3K64S4 family	●
STC8H3K64S2 family	●
STC8H8K64U family	●
STC8H2K64T family	●
STC8H4K64TLR family	●
STC8H4K64TLCD family	●
STC8H4K64LCD family	●

An I²C serial bus controller is integrated in the STC8H series of microcontrollers. I²C is a high-speed synchronous communication bus, which uses SCL (clock line) and SDA (data line) to carry out two-wire synchronous communication. For the port allocation of SCL and SDA, STC8H series of microcontrollers provide pin switch mode that can switch SCL and SDA to different I/O ports. Therefore, it is convenient to use a set of I²C as multiple sets of I²C buses through time sharing.

Compared with the standard I²C protocol, the following two mechanisms are ignored:

- No arbitration will be performed after the start signal (START) is sent.
- No timeout detection when the clock signal (SCL) stays at low level.

The I²C bus of the STC8H series of microcontrollers offer two modes of operation: master mode (SCL is the output port, which is used to transmit synchronous clock signal) and slave mode (SCL is the input port, which is used to receive the synchronous clock signal).

STC innovation: When the I²C serial bus controller of STC works in slave mode, the falling edge signal of SDA pin can wake up the MCU which is in power-down mode. (Note: Due to the fast I²C transmission speed, the first packet of data after the MCU wakes up is generally incorrect.)

19.1 Registers Related to I²C

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I2C Configuration Register	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I ² C Master Control Register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I ² C Master Status Register	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I ² C Slave Control Register	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I ² C Slave Status Register	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I ² C Slave Address Register	FE85H	SLADR[6:0]								MA 0000,0000
I2CTXD	I ² C Data Transmission Register	FE86H									0000,0000
I2CRXD	I ² C Data Receive Register	FE87H									0000,0000
I2CMSAUX	I ² C Master Auxiliary Control Register	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

19.2 I²C Master Mode

19.2.1 I2C Configuration Register (I2CCFG)

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

ENI2C: I²C function enable bit

0: disable I²C function

1: enable I²C function

MSSL: I²C mode selection bit

0: Salve mode

1: Master mode

MSSPEED[5:0]: I²C bus speed control bits (clocks to wait), **I2C bus speed = $F_{OSC} / 2 / (MSSPEED * 2 + 4)$**

MSSPEED[5:0]	Corresponding clocks
0	4
1	6
2	8
...	...
x	2x+4
...	...
62	128
63	130

The waiting parameter set by the MSSPEED is valid only when the I²C module is operating in the master mode. The waiting parameter is mainly used for the following signals in master mode:

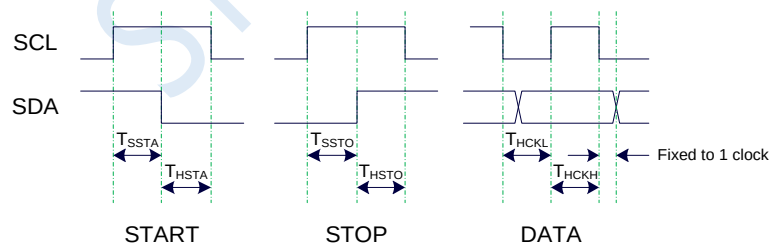
T_{SSTA}: Setup Time of START

T_{HSTA}: Hold Time of START

T_{SSTO}: Setup Time of STOP

T_{HSTO}: Hold Time of STOP

T_{HCKL}: Hold Time of SCL Low



Example 1: When MSSPEED=10, T_{SSTA}=T_{HSTA}=T_{SSTO}=T_{HSTO}=T_{HCKL}=24/F_{OSC}

Example 2: When 400K I2C bus speed is required at 24MHz operating frequency, MSSPEED=(24M / 400K / 2 - 4) / 2=13

19.2.2 I²C Master Control Register (I2CMSCR)

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

EMSI: Master mode interrupt enable control bit

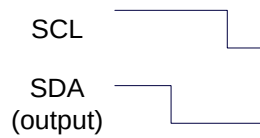
0: disable master mode interrupt

1: enable master mode interrupt

MSCMD[3:0]: master command bits

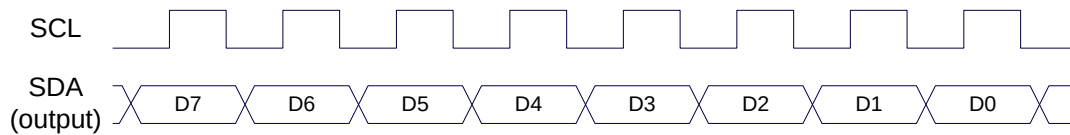
0000: Standby, no action

0001: START command. Send a START signal. If the I²C controller is in idle state currently, i.e. MSBUSY (I2CMSST.7) is 0, writing this command will make the controller enter the busy status, and the hardware will set the MSBUSY status bit automatically and start sending START signal. **If the I²C controller is busy currently, writing this command will trigger to send the START signal.** Sending the START signal waveform is shown below:



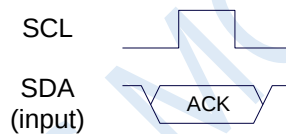
0010: Send data command.

After writing this command, the I²C bus controller will generate 8 clocks on the SCL pin and send the data in the I2CTXD register bit by bit to the SDA pin (send MSB firstly). The waveform of the transmitting data is shown in the following figure:



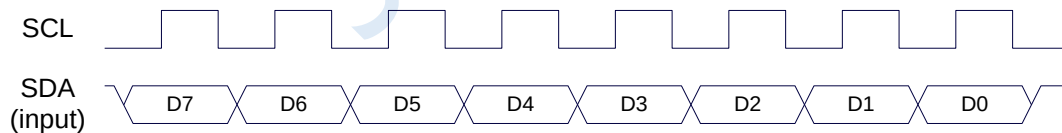
0011: Receive ACK command.

After writing this command, the I²C bus controller will generate a clock on the SCL pin and save the data bit read from SDA to MSACKI (I2CMSST.1). The waveform of the receiving ACK is shown below:



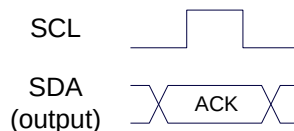
0100: Receive data command.

After writing this command, the I²C bus controller will generate 8 clocks on the SCL pin, and sequentially shift the data bit read from SDA to the I2CRXD register (receiving MSB firstly). The waveform of the receiving data is as shown in the figure below:



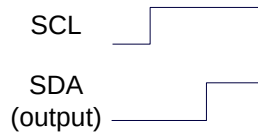
0101: Send ACK command.

After writing this command, the I²C bus controller will generate a clock on the SCL pin and send the data bit in MSACKO (I2CMSST.0) to SDA. The waveform of sending ACK is shown below:



0110: Send STOP signal.

After writing this command, the I²C bus controller starts sending STOP signal. After the signal is sent, the hardware clears the MSBUSY status bit automatically. The waveform of STOP signal is shown below:



0111: Reserved.

1000: Reserved.

1001: Start command + send data command + receive ACK command.

This command is a combination of command 0001, command 0010 and command 0011. After writing this command, the controller will execute these three commands in sequence.

1010: Send data command + receive ACK command.

This command is a combination of command 0010 and command 0011. After writing this command, the controller will execute these two commands in sequence.

1011: Receive data command + send ACK (0) command.

This command is a combination of command 0100 and command 0101. After writing this command, the controller will execute these two commands in sequence.

Note: The response signal returned by this command is fixed as ACK (0) and is not affected by the MSACKO bit.

1100: Receive data command + send NAK (1) command.

This command is a combination of command 0100 and command 0101. After writing this command, the controller will execute these two commands in sequence.

Note: The response signal returned by this command is fixed to NAK (1), and is not affected by the MSACKO bit.

19.2.3 I²C Master Auxiliary Control Register (I2CMSAUX)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA: I2C data automatic transmission enable bit in master mode.

0: disable automatic transmission

1: enable automatic transmission

If the automatic transmission function is enabled, when the MCU finishes writing to the I2CTXD data register, the I²C controller will trigger the "1010" command automatically, that is, it will send data automatically and receive the ACK signal.

19.2.4 I²C Master Status Register (I2CMSST)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY: status bit of I²C controller in master mode. (Read-only)

0: the controller is in idle state.

1: the controller is in busy state.

When the I²C controller is in master mode, the controller will enter the busy state after sending the START signal in the idle state. The busy state will be maintained until the STOP signal is successfully transmitted, and the state will return to the idle.

MSIF: master mode interrupt request bit (interrupt flag bit). When the I²C controller in the master mode executes the MSCMD command in the completion register I2CMSCR, it generates an interrupt signal. This bit is set to 1 by hardware automatically to request an interrupt to CPU. The MSIF bit must be cleared by software after responding to the interrupt.

MSACKI: In master mode, it is the ACK datum received after sending the "011" command to the MSCMD bit in I2CMSCR.

MSACKO: In master mode, it is the ACK signal ready to be transmitted. When the "101" command is sent to the MSCMD bit of I2CMSCR, the controller will read the datum of this bit automatically and send it as

ACK to SDA.

STC MCU

19.3 I²C Slave Mode

19.3.1 I²C Slave Control Register (I2CSLCR)

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

ESTAI: interrupt enable bit when receiving START signal in slave mode.

0: disable interrupt when receiving START signal in slave mode.

1: enable interrupt when receiving START signal in slave mode.

ERXI: interrupt enable bit after 1 byte datum is received in slave mode

0: disable interrupt after a datum is received in slave mode.

1: enable interrupt after 1 byte datum is received in slave mode.

ETXI: interrupt enable bit after 1 byte datum is sent in slave mode

0: disable interrupt after a datum is sent in slave mode.

1: enable interrupt after 1 byte datum is sent in slave mode.

ESTOI: interrupt enable bit after STOP signal is received in slave mode.

0: disable interrupt after STOP signal is received in slave mode.

1: enable interrupt after STOP signal is received in slave mode.

SLRST: reset slave mode

19.3.2 I²C Slave Status Register (I2CSLST)

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

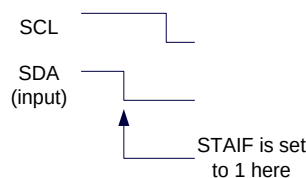
SLBUSY: status bit of I²C controller in slave mode. (Read-only)

0: the controller is in idle state.

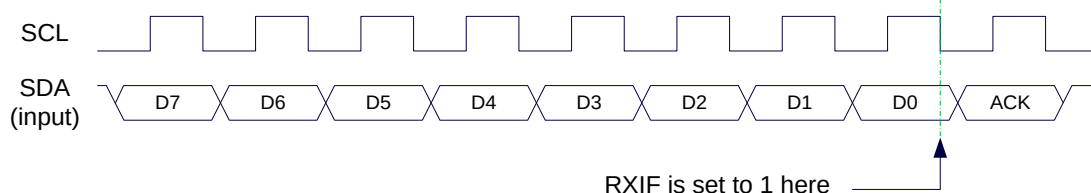
1: the controller is in busy state.

When the I²C controller is in slave mode, the controller will continue to detect the subsequent device address data when it receives the START signal from the master in idle state. If the device address matches the slave address set in the current I2CSLADR register, the controller will enter the busy state. And the busy state will be maintained until receives a STOP signal sent by the master successfully, and then the state will return to idle.

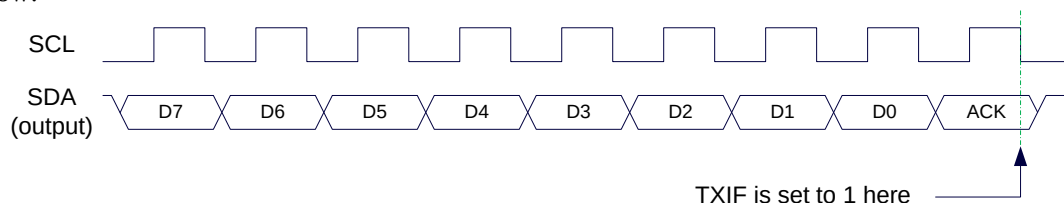
STAIF: interrupt request bit after START signal is received in slave mode. After the I²C controller in slave mode receives the START signal, this bit is set by hardware automatically and requests interrupt to CPU. The STAIF bit must be cleared by software after the interrupt is responded. The time point of STAIF being set is shown below:



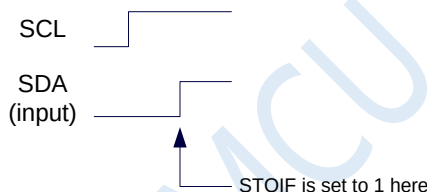
RXIF: interrupt request bit after 1-byte datum is received in slave mode. After the I²C controller in slave mode receives a 1-byte datum, this bit is set by hardware automatically at the falling edge of the 8th clock and will request interrupt to CPU. The RXIF bit must be cleared by software after the interrupt is responded. The time point of RXIF being set is shown in the figure below:



TXIF: interrupt request bit after 1-byte datum transmission is completed in slave mode. After the I²C controller in slave mode completes sending 1 byte of datum and receives a 1-bit ACK signal successfully, this bit is set by hardware automatically at the falling edge of the 9th clock and requests an interrupt to CPU. TXIF bit must be cleared by software after the interrupt is responded. The time point of TXIF being set is shown below:

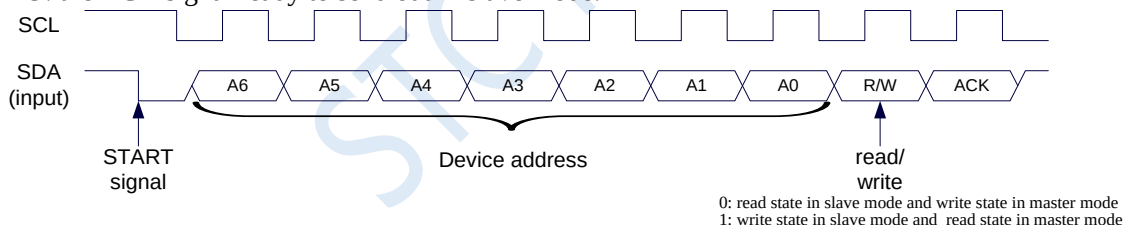


STOIF: interrupt request bit after STOP signal is received in slave mode. After the I²C controller in slave mode receives the STOP signal, this bit is set by hardware automatically and requests interrupt to CPU. The STOIF bit must be cleared by software after the interrupt is serviced. The time point of STOIF being set is shown below:



SLACKI: ACK data received in slave mode

SLACKO: the ACK signal ready to send out in slave mode.



19.3.3 I²C Slave Address Register (I2CSLADR)

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

I2CSLADR[7:1]: the slave device address

When the I²C controller is in slave mode, the controller will continue to detect the device address and read / write signals sent by the master after it receives the START signal. If the device address sent by the master matches the slave device address set in SLADR[6: 0], the controller will request an interrupt to CPU to process the I²C event. Otherwise, if the device address does not match, the I²C controller continues to monitor, wait for the next START signal, and match the next device address.

MA: Slave device address matching control bit

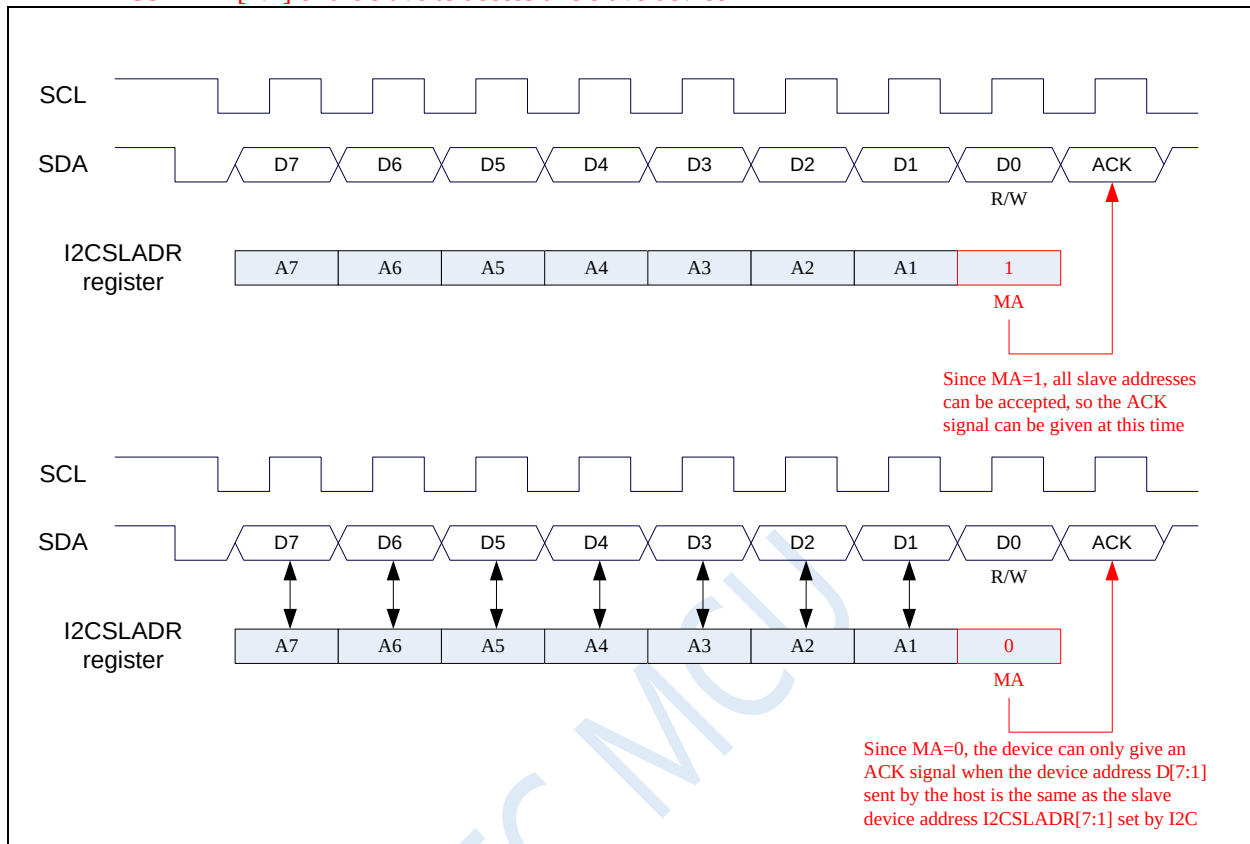
0: The device address must be the same as I2CSLADR[7:1].

1: Ignore the settings in I2CSLADR[7:1] and match all device addresses.

Note: The I2C bus protocol stipulates that a maximum of 128 I2C devices (theoretical value) can be mounted on the I2C bus, and different I2C devices are identified by different I2C slave device addresses. After the I2C master sends the start signal, the upper 7 bits of the first data (DATA0) sent are the slave device address (DATA0[7:1] is the I2C device address), and the lowest bit is the read and

write signal. When the I2C device slave address register MA (I2CSLADR.0) is 1, it means that the I2C slave can accept all device addresses. At this time, any device address sent by the host, that is, DATA0[7:1] is any value, the slave Can respond. When I2C device slave address register MA (I2CSLADR.0)

When it is 0, the device address DATA0[7:1] sent by the host must be the same as the device address I2CSLADR[7:1] of the slave to access this slave device



19.3.4 I²C data registers (I2CTXD, I2CRXD)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD is the I²C transmit data register that holds the I²C data to be transmitted.

I2CRXD is the I²C receive data register that holds the I²C data received.

19.4 Example Routines

19.4.1 I²C is Used to Access AT24C256 in Master Mode (Interrupt Mode)

C language code

//Operating frequency for test is 11.0592MHz

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

sfr      P_SW2      = 0xba;

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

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

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

bit      busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;           //Clear interrupt flag
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
{
    busy = 1;
}
```

```
    I2CMSCR = 0x81;                                //Send START command
    while (busy);
}

void SendData(char dat)
{
    I2CTXD = dat;                                    //Write data to the data buffer
    busy = 1;
    I2CMSCR = 0x82;                                    //Send a SEND command
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;                                    //Send read ACK command
    while (busy);
}

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

void SendACK()
{
    I2CMSST = 0x00;                                    //Setup the ACK signal
    busy = 1;
    I2CMSCR = 0x85;                                    //Send ACK command
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                                    //Setup the NAK signal
    busy = 1;
    I2CMSCR = 0x85;                                    //Send ACK command
    while (busy);
}

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

void Delay()
{
    int i;

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

```
        _nop_();
    }
}

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

    P_SW2 = 0x80;

    I2CCFG = 0xe0;           //Enable I2C master mode
    I2CMSST = 0x00;
    EA = 1;

    Start();    //Send start command
    SendData(0xa0);           //Send device address + write command
    RecvACK();
    SendData(0x00);           //Send storage address high byte
    RecvACK();
    SendData(0x00);           //Send storage address low byte
    RecvACK();
    SendData(0x12);           //Write test data 1
    RecvACK();
    SendData(0x78);           //Write test data 2
    RecvACK();
    Stop();           //Send stop command

    Delay();           //Waiting for the device to write data

    Start();           //Send start command
    SendData(0xa0);           //Send device address + write command
    RecvACK();
    SendData(0x00);           //Send storage address high byte
    RecvACK();
    SendData(0x00);           //Send storage address low byte
    RecvACK();
    Start();           //Send start command
    SendData(0xa1);           //Send device address + read command
    RecvACK();
    P0 = RecvData();           //Read data 1
    SendACK();
    P2 = RecvData();           //Read data 2
    SendNAK();
    Stop();           //Send stop command

    P_SW2 = 0x00;

    while (1);
}
```

}

Assembly code

;Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH	
I2CCFG	XDATA	0FE80H	
I2CMSCR	XDATA	0FE81H	
I2CMSST	XDATA	0FE82H	
I2CSLCR	XDATA	0FE83H	
I2CSLST	XDATA	0FE84H	
I2CSLADR	XDATA	0FE85H	
I2CTXD	XDATA	0FE86H	
I2CRXD	XDATA	0FE87H	
SDA	BIT	P1.4	
SCL	BIT	P1.5	
BUSY	BIT	20H.0	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	00C3H	
	LJMP	I2CISR	
	ORG	0100H	
I2CISR:			
	PUSH	ACC	
	PUSH	DPL	
	PUSH	DPH	
	MOV	DPTR,#I2CMSST	;Clear interrupt flag
	MOVX	A,@DPTR	
	ANL	A,#NOT 40H	
	MOV	DPTR,#I2CMSST	
	MOVX	@DPTR,A	
	CLR	BUSY	;Reset busy flag
	POP	DPH	
	POP	DPL	
	POP	ACC	
	RETI		
START:			
	SETB	BUSY	

```

MOV      A,#10000001B      ;Send START command
MOV      DPTR,#I2CMSCR
MOVX     @DPTR,A
JMP      WAIT

SENDDATA:
MOV      DPTR,#I2CTXD      ;Write data to the data buffer
MOVX     @DPTR,A
SETB     BUSY
MOV      A,#10000010B      ;Send a SEND command
MOV      DPTR,#I2CMSCR
MOVX     @DPTR,A
JMP      WAIT

RECVACK:
SETB     BUSY
MOV      A,#10000011B      ;Send read ACK command
MOV      DPTR,#I2CMSCR
MOVX     @DPTR,A
JMP      WAIT

RCVDATA:
SETB     BUSY
MOV      A,#10000100B      ;Send RECV command
MOV      DPTR,#I2CMSCR
MOVX     @DPTR,A
CALL     WAIT
MOV      DPTR,#I2CRXD      ;Read data from the data buffer
MOVX     A,@DPTR
RET

SENDACK:
MOV      A,#00000000B      ;Setup the ACK signal
MOV      DPTR,#I2CMSST
MOVX     @DPTR,A
SETB     BUSY
MOV      A,#10000101B      ;Send ACK command
MOV      DPTR,#I2CMSCR
MOVX     @DPTR,A
JMP      WAIT

SENDNAK:
MOV      A,#00000001B      ;Setup the NAK signal
MOV      DPTR,#I2CMSST
MOVX     @DPTR,A
SETB     BUSY
MOV      A,#10000101B      ;Send ACK command
MOV      DPTR,#I2CMSCR
MOVX     @DPTR,A
JMP      WAIT

STOP:
SETB     BUSY
MOV      A,#10000110B      ;Send STOP command
MOV      DPTR,#I2CMSCR
MOVX     @DPTR,A
JMP      WAIT

WAIT:
JB       BUSY,$            ;Wait for the command to be sent
RET

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

DELAY1:

```

```

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

```

MAIN:

```

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

MOV     P_SW2, #80H

MOV     A, #11100000B      ;Set the I2C module as master mode
MOV     DPTR, #I2CCFG
MOVX    @DPTR, A
MOV     A, #00000000B
MOV     DPTR, #I2CMSST
MOVX    @DPTR, A
SETB    EA

CALL     START              ;Send start command
MOV     A, #0A0H
CALL     SENDDATA           ;Send device address + write command
CALL     RECVACK
MOV     A, #000H            ;Send storage address high byte
CALL     SENDDATA
CALL     RECVACK
MOV     A, #000H            ;Send storage address low byte
CALL     SENDDATA
CALL     RECVACK
MOV     A, #12H             ;Write test data 1
CALL     SENDDATA
CALL     RECVACK
MOV     A, #78H             ;Write test data 2
CALL     SENDDATA
CALL     RECVACK
CALL     STOP               ;Send stop command

CALL     DELAY              ;Waiting for the device to write data

CALL     START              ;Send start command
MOV     A, #0A0H            ;Send device address + write command
CALL     SENDDATA
CALL     RECVACK
MOV     A, #000H            ;Send storage address high byte
CALL     SENDDATA

```

```

CALL    RECVACK
MOV     A,#000H                ;Send storage address low byte
CALL    SENDDATA
CALL    RECVACK
CALL    START                  ;Send start command
MOV     A,#0A1H                ;Send device address + read command
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA               ;Read data 1
MOV     P0,A
CALL    SENDACK
CALL    RECVDATA               ;Read data 2
MOV     P2,A
CALL    SENDNAK
CALL    STOP                   ;Send stop command

JMP     $

END

```

19.4.2 I²C is Used to Access AT24C256 in Master Mode AT24C256 (Polling Mode)

C language code

//Operating frequency for test is 11.0592MHz

```

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

sfr      P_SW2      = 0xba;

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

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

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

```

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

void Start()
{
    I2CMSCR = 0x01;           //Send START command
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;             //Write data to the data buffer
    I2CMSCR = 0x02;           //Send a SEND command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;           //Send read ACK command
    Wait();
}

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

void SendACK()
{
    I2CMSST = 0x00;           //Setup the ACK signal
    I2CMSCR = 0x05;           //Send ACK command
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;           //Setup the NAK signal
    I2CMSCR = 0x05;           //Send ACK command
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;           //Send STOP command
    Wait();
}

void Delay()
{
    int i;

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



```

        _nop_0;
        _nop_0;
        _nop_0;
        _nop_0;
    }
}

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

    P_SW2 = 0x80;

    I2CCFG = 0xe0;           //Enable I2C master mode
    I2CMSST = 0x00;

    Start();                 //Send start command
    SendData(0xa0);          //Send device address + write command
    RecvACK();
    SendData(0x00);         //Send storage address high byte
    RecvACK();
    SendData(0x00);         //Send storage address low byte
    RecvACK();
    SendData(0x12);         //Write test data 1
    RecvACK();
    SendData(0x78);         //Write test data 2
    RecvACK();
    Stop();                  //Send stop command

    Delay();                 //Waiting for the device to write data

    Start();                 //Send start command
    SendData(0xa0);          //Send device address + write command
    RecvACK();
    SendData(0x00);         //Send storage address high byte
    RecvACK();
    SendData(0x00);         //Send storage address low byte
    RecvACK();
    Start();                 //Send start command
    SendData(0xa1);          //Send device address + read command
    RecvACK();
    P0 = RecvData();         //Read data 1
    SendACK();
    P2 = RecvData();         //Read data 2
    SendNAK();
    Stop();                  //Send stop command

    P_SW2 = 0x00;

```

```

    while (1);
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH	
I2CCFG	XDATA	0FE80H	
I2CMSCR	XDATA	0FE81H	
I2CMSST	XDATA	0FE82H	
I2CSLCR	XDATA	0FE83H	
I2CSLST	XDATA	0FE84H	
I2CSLADR	XDATA	0FE85H	
I2CTXD	XDATA	0FE86H	
I2CRXD	XDATA	0FE87H	
SDA	BIT	P1.4	
SCL	BIT	P1.5	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0100H	
START:	MOV	A,#00000001B	;Send START command
	MOV	DPTR,#I2CMSCR	
	MOVX	@DPTR,A	
	JMP	WAIT	
SENDDATA:	MOV	DPTR,#I2CTXD	;Write data to the data buffer
	MOVX	@DPTR,A	
	MOV	A,#00000010B	;Send a SEND command
	MOV	DPTR,#I2CMSCR	
	MOVX	@DPTR,A	
	JMP	WAIT	
RECVACK:	MOV	A,#00000011B	;Send read ACK command
	MOV	DPTR,#I2CMSCR	
	MOVX	@DPTR,A	
	JMP	WAIT	
RECVDATA:	MOV	A,#00000100B	;Send RECV command
	MOV	DPTR,#I2CMSCR	
	MOVX	@DPTR,A	

```

        CALL    WAIT
        MOV     DPTR,#I2CRXD           ;Read data from the data buffer
        MOVX    A,@DPTR
        RET

SENDACK:
        MOV     A,#00000000B           ;Setup the ACK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B           ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT

SENDNAK:
        MOV     A,#00000001B           ;Setup the NAK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B           ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT

STOP:
        MOV     A,#00000110B           ;Send STOP command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT

WAIT:
        MOV     DPTR,#I2CMSST           ;Clear interrupt flag
        MOVX    A,@DPTR
        JNB     ACC.6,WAIT
        ANL     A,#NOT 40H
        MOVX    @DPTR,A
        RET

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

DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

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

```

```

MOV      P_SW2,#80H

MOV      A,#11100000B      ;Set the I2C module as master mode
MOV      DPTR,#I2CCFG
MOVX     @DPTR,A
MOV      A,#00000000B
MOV      DPTR,#I2CMSST
MOVX     @DPTR,A

CALL     START              ;Send start command
MOV      A,#0A0H
CALL     SENDDATA           ;Send device address + write command
CALL     RECVACK
MOV      A,#000H            ;Send storage address high byte
CALL     SENDDATA
CALL     RECVACK
MOV      A,#000H            ;Send storage address low byte
CALL     SENDDATA
CALL     RECVACK
MOV      A,#12H             ;Write test data 1
CALL     SENDDATA
CALL     RECVACK
MOV      A,#78H             ;Write test data 2
CALL     SENDDATA
CALL     RECVACK
CALL     STOP               ;Send stop command

CALL     DELAY              ;Waiting for the device to write data

CALL     START              ;Send start command
MOV      A,#0A0H            ;Send device address + write command
CALL     SENDDATA
CALL     RECVACK
MOV      A,#000H            ;Send storage address high byte
CALL     SENDDATA
CALL     RECVACK
MOV      A,#000H            ;Send storage address low byte
CALL     SENDDATA
CALL     RECVACK
CALL     START              ;Send start command
MOV      A,#0A1H            ;Send device address + read command
CALL     SENDDATA
CALL     RECVACK
CALL     RECVDATA           ;Read data 1
MOV      P0,A
CALL     SENDACK
CALL     RECVDATA           ;Read data 2
MOV      P2,A
CALL     SENDNAK
CALL     STOP               ;Send stop command

JMP      $

END

```

19.4.3 I²C is Used to Access PCF8563 in Master Mode

C language code

//Operating frequency for test is 11.0592MHz

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

sfr      P_SW2      = 0xba;

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

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

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

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

void Start()
{
    I2CMSCR = 0x01;           //Send START command
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;             //Write data to the data buffer
    I2CMSCR = 0x02;           //Send a SEND command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;           //Send read ACK command
    Wait();
}
```

```
}

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

void SendACK()
{
    I2CMSST = 0x00;                //Setup the ACK signal
    I2CMSCR = 0x05;                //Send ACK command
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                //Setup the NAK signal
    I2CMSCR = 0x05;                //Send ACK command
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                //Send STOP command
    Wait();
}

void Delay()
{
    int i;

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

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

    P_SW2 = 0x80;
}
```

```

I2CCFG = 0xe0;           //Enable I2C master mode
I2CMSST = 0x00;

Start();                  //Send start command
SendData(0xa2);           //Send device address + write command
RecvACK();
SendData(0x02);           //Send storage address
RecvACK();
SendData(0x00);           //Set second value
RecvACK();
SendData(0x00);           //Set minute value
RecvACK();
SendData(0x12);           //Set hour value
RecvACK();
Stop();                   //Send stop command

while (1)
{
    Start();               //Send start command
    SendData(0xa2);        //Send device address + write command
    RecvACK();
    SendData(0x02);        //Send storage address
    RecvACK();
    Start();               //Send start command
    SendData(0xa3);        //Send device address + read command
    RecvACK();
    P0 = RecvData();        //Read second value
    SendACK();
    P2 = RecvData();        //Read minute value
    SendACK();
    P3 = RecvData();        //Read hour value
    SendNAK();
    Stop();                 //Send stop command

    Delay();
}
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH
I2CCFG	XDATA	0FE80H
I2CMSCR	XDATA	0FE81H
I2CMSST	XDATA	0FE82H
I2CSLCR	XDATA	0FE83H
I2CSLST	XDATA	0FE84H
I2CSLADR	XDATA	0FE85H
I2CTXD	XDATA	0FE86H
I2CRXD	XDATA	0FE87H
SDA	BIT	P1.4
SCL	BIT	P1.5
P0M1	DATA	093H
P0M0	DATA	094H
P1M1	DATA	091H
P1M0	DATA	092H

```

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

          ORG          0000H
          LJMP         MAIN

          ORG          0100H
START:
          MOV          A,#00000001B          ;Send START command
          MOV          DPTR,#I2CMSCR
          MOVX         @DPTR,A
          JMP          WAIT

SENDDATA:
          MOV          DPTR,#I2CTXD          ;Write data to the data buffer
          MOVX         @DPTR,A
          MOV          A,#00000010B          ;Send a SEND command
          MOV          DPTR,#I2CMSCR
          MOVX         @DPTR,A
          JMP          WAIT

RECVACK:
          MOV          A,#00000011B          ;Send read ACK command
          MOV          DPTR,#I2CMSCR
          MOVX         @DPTR,A
          JMP          WAIT

RECVDATA:
          MOV          A,#00000100B          ;Send RECV command
          MOV          DPTR,#I2CMSCR
          MOVX         @DPTR,A
          CALL         WAIT
          MOV          DPTR,#I2CRXD          ;Read data from the data buffer
          MOVX         A,@DPTR
          RET

SENDACK:
          MOV          A,#00000000B          ;Setup the ACK signal
          MOV          DPTR,#I2CMSST
          MOVX         @DPTR,A
          MOV          A,#00000101B          ;Send ACK command
          MOV          DPTR,#I2CMSCR
          MOVX         @DPTR,A
          JMP          WAIT

SENDNAK:
          MOV          A,#00000001B          ;Setup the NAK signal
          MOV          DPTR,#I2CMSST
          MOVX         @DPTR,A
          MOV          A,#00000101B          ;Send ACK command
          MOV          DPTR,#I2CMSCR
          MOVX         @DPTR,A
          JMP          WAIT

STOP:
          MOV          A,#00000110B          ;Send STOP command
          MOV          DPTR,#I2CMSCR
          MOVX         @DPTR,A
          JMP          WAIT

```


WAIT:

```

MOV    DPTR,#I2CMSST    ;Clear interrupt flag
MOVX   A,@DPTR
JNB    ACC.6, WAIT
ANL    A,#NOT 40H
MOVX   @DPTR,A
RET

```

DELAY:

```

MOV    R0,#0
MOV    R1,#0

```

DELAY1:

```

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

```

MAIN:

```

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

MOV    P_SW2,#80H

MOV    A,#11100000B    ;Set the I2C module as master mode
MOV    DPTR,#I2CCFG
MOVX   @DPTR,A
MOV    A,#00000000B
MOV    DPTR,#I2CMSST
MOVX   @DPTR,A

CALL    START          ;Send start command
MOV    A,#0A2H
CALL    SENDDATA       ;Send device address + write command
CALL    RECVACK
MOV    A,#002H         ;Send storage address
CALL    SENDDATA
CALL    RECVACK
MOV    A,#00H          ;Set second value
CALL    SENDDATA
CALL    RECVACK
MOV    A,#00H          ;Set minute value
CALL    SENDDATA
CALL    RECVACK
MOV    A,#12H          ;Set hour value
CALL    SENDDATA

```

```

        CALL    RECVACK
        CALL    STOP                ;Send stop command
LOOP:
        CALL    START                ;Send start command
        MOV     A,#0A2H              ;Send device address + write command
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#002H              ;Send storage address
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START                ;Send start command
        MOV     A,#0A3H              ;Send device address + read command
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA              ;Read second value
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA              ;Read minute value
        MOV     P2,A
        CALL    SENDACK
        CALL    RECVDATA              ;Read hour value
        MOV     P3,A
        CALL    SENDNAK
        CALL    STOP                ;Send stop command

        CALL    DELAY

        JMP     LOOP

END

```

19.4.4 I²C Slave Mode (Polling Mode)

C language code

//Operating frequency for test is 11.0592MHz

```

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

sfr      P_SW2      = 0xba;

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

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

```

```

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

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

bit       isda;           //Device address flag
bit       isma;           //Storage address flag
unsigned char  addr;
unsigned char pdata      buffer[256];

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

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;           //Handle the START event
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;           //Handle the RECV event
        if (isda)
        {
            isda = 0;               //Handle the RECV event (RECV DEVICE ADDR)
        }
        else if (isma)
        {
            isma = 0;               //Handle the RECV event (RECV MEMORY ADDR)
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD; //Handle the RECV event (RECV DATA)
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;           //Handle the SEND event
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;           //Stop receiving data when receiving NAK
        }
        else
        {
            I2CTXD = buffer[++addr]; //Continue reading data when receiving ACK
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;           //Handle the STOP event
        isda = 1;
        isma = 1;
    }
}

```

```

    _pop_(P_SW2);
}

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

    P_SW2 = 0x80;

    I2CCFG = 0x81;           //Enable I2C slave mode
    I2CSLADR = 0x5a;         //Set the slave device address to 5A
                             //That is, I2CSLADR[7:1]=010_1101B, MA=0B.
                             //Since MA is 0, the device address sent by the host must be
                             // the same as I2CSLADR[7:1] to access this I2C slave
    device.                  //If the host needs to write data, it will send
    5AH(0101_1010B)
    I2CSLST = 0x00;
    I2CSLCR = 0x78;         //Enable interrupt of slave mode
    EA = 1;

    isda = 1;               //User variable initialization
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH	
I2CCFG	XDATA	0FE80H	
I2CMSCR	XDATA	0FE81H	
I2CMSST	XDATA	0FE82H	
I2CSLCR	XDATA	0FE83H	
I2CSLST	XDATA	0FE84H	
I2CSLADR	XDATA	0FE85H	
I2CTXD	XDATA	0FE86H	
I2CRXD	XDATA	0FE87H	
SDA	BIT	P1.4	
SCL	BIT	P1.5	
ISDA	BIT	20H.0	;Device address flag
ISMA	BIT	20H.1	;Storage address flag

ADDR	DATA	21H	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	00C3H	
	LJMP	I2CISR	
	ORG	0100H	
I2CISR:	PUSH	ACC	
	PUSH	PSW	
	PUSH	DPL	
	PUSH	DPH	
	MOV	DPTR,#I2CSLST	<i>;Detect slave status</i>
	MOVX	A,@DPTR	
	JB	ACC.6,STARTIF	
	JB	ACC.5,RXIF	
	JB	ACC.4,TXIF	
	JB	ACC.3,STOPIF	
ISREXIT:	POP	DPH	
	POP	DPL	
	POP	PSW	
	POP	ACC	
	RETI		
STARTIF:	ANL	A,#NOT 40H	<i>;Handle the START event</i>
	MOVX	@DPTR,A	
	JMP	ISREXIT	
RXIF:	ANL	A,#NOT 20H	<i>;Handle the RECV event</i>
	MOVX	@DPTR,A	
	MOV	DPTR,#I2CRXD	
	MOVX	A,@DPTR	
	JBC	ISDA,RXDA	
	JBC	ISMA,RXMA	
	MOV	R0,ADDR	<i>;Handle the RECV event (RECV DATA)</i>
	MOVX	@R0,A	
	INC	ADDR	
	JMP	ISREXIT	
RXDA:	JMP	ISREXIT	<i>;Handle the RECV event (RECV DEVICE ADDR)</i>
RXMA:	MOV	ADDR,A	<i>;Handle the RECV event (RECV MEMORY ADDR)</i>
	MOV	R0,A	

```

MOVX    A,@R0
MOV     DPTR,#I2CTXD
MOVX    @DPTR,A
JMP     ISREXIT

TXIF:
ANL     A,#NOT 10H                ;Handle the SEND event
MOVX    @DPTR,A
JB      ACC.1,RXNAK
INC     ADDR
MOV     R0,ADDR
MOVX    A,@R0
MOV     DPTR,#I2CTXD
MOVX    @DPTR,A
JMP     ISREXIT

RXNAK:
MOVX    A,#0FFH
MOV     DPTR,#I2CTXD
MOVX    @DPTR,A
JMP     ISREXIT

STOPIF:
ANL     A,#NOT 08H                ;Handle the STOP event
MOVX    @DPTR,A
SETB    ISDA
SETB    ISMA
JMP     ISREXIT

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

MOV     P_SW2,#80H

MOV     A,#10000001B                ;Enable I2C slave mode
MOV     DPTR,#I2CCFG
MOVX    @DPTR,A
MOV     A,#01011010B                ;Set the slave device address to 5A
                                        ;That is, I2CSLADR[7:1]=010_1101B, MA=0B.
                                        ;Since MA is 0, the device address sent by the host must be
                                        ; the same as I2CSLADR[7:1] to access this I2C slave
device.
                                        ;If the host needs to write data, it will send
5AH(0101_1010B)

MOV     DPTR,#I2CSLADR
MOVX    @DPTR,A
MOV     A,#00000000B
MOV     DPTR,#I2CSLST
MOVX    @DPTR,A

```

```

MOV      A,#01111000B           ;Enable interrupt of slave mode
MOV      DPTR,#I2CSLCR
MOVX     @DPTR,A

SETB     ISDA                   ;User variable initialization
SETB     ISMA
CLR      A
MOV      ADDR,A
MOV      R0,A
MOVX     A,@R0
MOV      DPTR,#I2CTXD
MOVX     @DPTR,A

SETB     EA

SJMP     $

END

```

19.4.5 I²C Slave Mode (Polling Mode)

C language code

//Operating frequency for test is 11.0592MHz

```

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

sfr      P_SW2      = 0xba;

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

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

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

bit      isda;           //Device address flag
bit      isma;           //Storage address flag
unsigned char      addr;

```

```
unsigned char pdata      buffer[256];
```

```
void main()
```

```
{
```

```
    P0M0 = 0x00;
```

```
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
```

```
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
```

```
    P4M1 = 0x00;
```

```
    P5M0 = 0x00;
```

```
    P5M1 = 0x00;
```

```
    P_SW2 = 0x80;
```

```
    I2CCFG = 0x81; //Enable I2C slave mode
```

```
    I2CSLADR = 0x5a;
```

```
    //Set the slave device address to 5A
```

```
    //That is, I2CSLADR[7:1]=010_1101B, MA=0B.
```

```
    //Since MA is 0, the device address sent by the host must be
```

```
    // the same as I2CSLADR[7:1] to access this I2C slave
```

```
device.
```

```
    //If the host needs to write data, it will send
```

```
5AH(0101_1010B)
```

```
    //If the host needs to read data, it will send 5BH
```

```
(0101_1011B)
```

```
    I2CSLST = 0x00;
```

```
    I2CSLCR = 0x00;
```

```
    //Disable interrupt of slave mode
```

```
    isda = 1;
```

```
    //User variable initialization
```

```
    isma = 1;
```

```
    addr = 0;
```

```
    I2CTXD = buffer[addr];
```

```
while (1)
```

```
{
```

```
    if (I2CSLST & 0x40)
```

```
    {
```

```
        I2CSLST &= ~0x40;
```

```
    //Handle the START event
```

```
    }
```

```
    else if (I2CSLST & 0x20)
```

```
    {
```

```
        I2CSLST &= ~0x20;
```

```
    //Handle the RECV event
```

```
        if (isda)
```

```
        {
```

```
            isda = 0;
```

```
    //Handle the RECV event (RECV DEVICE ADDR)
```

```
        }
```

```
        else if (isma)
```

```
        {
```

```
            isma = 0;
```

```
    //Handle the RECV event (RECV MEMORY ADDR)
```

```
            addr = I2CRXD;
```

```
            I2CTXD = buffer[addr];
```

```
        }
```

```
        else
```

```
        {
```

```
            buffer[addr++] = I2CRXD;
```

```
    //Handle the RECV event (RECV DATA)
```



```

    }
}
else if (I2CSLST & 0x10)
{
    I2CSLST &= ~0x10;           //Handle the SEND event
    if (I2CSLST & 0x02)
    {
        I2CTXD = 0xff;          //Stop receiving data when receiving NAK
    }
    else
    {
        I2CTXD = buffer[++addr]; //Continue reading data when receiving ACK
    }
}
else if (I2CSLST & 0x08)
{
    I2CSLST &= ~0x08;           //Handle the STOP event
    isda = 1;
    isma = 1;
}
}
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH	
I2CCFG	XDATA	0FE80H	
I2CMSCR	XDATA	0FE81H	
I2CMSST	XDATA	0FE82H	
I2CSLCR	XDATA	0FE83H	
I2CSLST	XDATA	0FE84H	
I2CSLADR	XDATA	0FE85H	
I2CTXD	XDATA	0FE86H	
I2CRXD	XDATA	0FE87H	
SDA	BIT	P1.4	
SCL	BIT	P1.5	
ISDA	BIT	20H.0	;Device address flag
ISMA	BIT	20H.1	;Storage address flag
ADDR	DATA	21H	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	

```

MAIN:      ORG          0100H

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

MOV        P_SW2, #80H

MOV        A, #10000001B          ;Enable I2C slave mode
MOV        DPTR, #I2CCFG
MOVX       @DPTR, A
MOV        A, #01011010B          ;Set the slave device address to 5A
                                          ;/That is, I2CSLADR[7:1]=010_1101B, MA=0B.
                                          ;Since MA is 0, the device address sent by the host must be
                                          ; the same as I2CSLADR[7:1] to access this I2C slave
device.
5AH(0101_1010B)
(0101_1011B)
                                          ;If the host needs to write data, it will send
                                          ;If the host needs to read data, it will send 5BH

MOV        DPTR, #I2CSLADR
MOVX       @DPTR, A
MOV        A, #00000000B
MOV        DPTR, #I2CSLST
MOVX       @DPTR, A
MOV        A, #00000000B          ;Disable interrupt of slave mode
MOV        DPTR, #I2CSLCR
MOVX       @DPTR, A

SETB       ISDA                  ;User variable initialization
SETB       ISMA
CLR        A
MOV        ADDR, A
MOV        R0, A
MOVX       A, @R0
MOV        DPTR, #I2CTXD
MOVX       @DPTR, A

LOOP:      MOV        DPTR, #I2CSLST          ;Detect slave status
MOVX       A, @DPTR
JB         ACC.6, STARTIF
JB         ACC.5, RXIF
JB         ACC.4, TXIF
JB         ACC.3, STOPIF
JMP        LOOP

STARTIF:

```

```

        ANL      A,#NOT 40H                ;Handle the START event
        MOVX     @DPTR,A
        JMP      LOOP

RXIF:
        ANL      A,#NOT 20H                ;Handle the RECV event
        MOVX     @DPTR,A
        MOV      DPTR,#I2CRXD
        MOVX     A,@DPTR
        JBC      ISDA,RXDA
        JBC      ISMA,RXMA
        MOV      R0,ADDR                    ;Handle the RECV event (RECV DATA)
        MOVX     @R0,A
        INC      ADDR
        JMP      LOOP

RXDA:
        JMP      LOOP                    ;Handle the RECV event (RECV DEVICE ADDR)

RXMA:
        MOV      ADDR,A                    ;Handle the RECV event (RECV MEMORY ADDR)
        MOV      R0,A
        MOVX     A,@R0
        MOV      DPTR,#I2CTXD
        MOVX     @DPTR,A
        JMP      LOOP

TXIF:
        ANL      A,#NOT 10H                ;Handle the SEND event
        MOVX     @DPTR,A
        JB       ACC.1,RXNAK
        INC      ADDR
        MOV      R0,ADDR
        MOVX     A,@R0
        MOV      DPTR,#I2CTXD
        MOVX     @DPTR,A
        JMP      LOOP

RXNAK:
        MOVX     A,#0FFH
        MOV      DPTR,#I2CTXD
        MOVX     @DPTR,A
        JMP      LOOP

STOPIF:
        ANL      A,#NOT 08H                ;Handle the STOP event
        MOVX     @DPTR,A
        SETB     ISDA
        SETB     ISMA
        JMP      LOOP

END

```

19.4.6 Master Codes for testing I²C Slave Mode

C language code

//Operating frequency for test is 11.0592MHz

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

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

```

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

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

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

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

void Start()
{
    I2CMSCR = 0x01;           //Send START command
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;             //Write data to the data buffer
    I2CMSCR = 0x02;           //Send a SEND command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;           //Send read ACK command
    Wait();
}

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

void SendACK()
{

```

```

    I2CMSST = 0x00;                                //Setup the ACK signal
    I2CMSCR = 0x05;                                //Send ACK command
    Wait();
}
void SendNAK()
{
    I2CMSST = 0x01;                                //Setup the NAK signal
    I2CMSCR = 0x05;                                //Send ACK command
    Wait();
}
void Stop()
{
    I2CMSCR = 0x06;                                //Send STOP command
    Wait();
}
void Delay()
{
    int i;

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

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

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                                //Enable I2C master mode
    I2CMSST = 0x00;

    Start();                                        //Send start command
    SendData(0x5a);                                //Send device address (010_1101B) + write command (0b)
    RecvACK();
    SendData(0x00);                                //Send storage address
    RecvACK();
    SendData(0x12);                                //Write test data 1
    RecvACK();
    SendData(0x78);                                //Write test data 2
    RecvACK();
    Stop();                                        //Send stop command
}

```

```

    Start();                                     //Send start command
    SendData(0x5a);                             //Send device address (010_1101B) + write command (0b)
    RecvACK();
    SendData(0x00);                             //Send storage address high byte
    RecvACK();
    Start();                                     //Send start command
    SendData(0x5b);                             //Send device address (010_1101B) + read command (1b)
    RecvACK();
    P0 = RecvData();                             //Read data 1
    SendACK();
    P2 = RecvData();                             //Read data 2
    SendNAK();
    Stop();                                     //Send stop command

    P_SW2 = 0x00;

    while (1);
}

```

Assembly code

;Operating frequency for test is 11.0592MHz

P_SW2	DATA	0BAH	
I2CCFG	XDATA	0FE80H	
I2CMSCR	XDATA	0FE81H	
I2CMSST	XDATA	0FE82H	
I2CSLCR	XDATA	0FE83H	
I2CSLST	XDATA	0FE84H	
I2CSLADR	XDATA	0FE85H	
I2CTXD	XDATA	0FE86H	
I2CRXD	XDATA	0FE87H	
SDA	BIT	P1.4	
SCL	BIT	P1.5	
P0M1	DATA	093H	
P0M0	DATA	094H	
P1M1	DATA	091H	
P1M0	DATA	092H	
P2M1	DATA	095H	
P2M0	DATA	096H	
P3M1	DATA	0B1H	
P3M0	DATA	0B2H	
P4M1	DATA	0B3H	
P4M0	DATA	0B4H	
P5M1	DATA	0C9H	
P5M0	DATA	0CAH	
	ORG	0000H	
	LJMP	MAIN	
	ORG	0100H	
START:	MOV	A,#00000001B	;Send START command
	MOV	DPTR,#I2CMSCR	
	MOVX	@DPTR,A	
	JMP	WAIT	
SENDDATA:			

```

        MOV     DPTR,#I2CTXD           ;Write data to the data buffer
        MOVX    @DPTR,A
        MOV     A,#00000010B          ;Send a SEND command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT

RECVACK:
        MOV     A,#00000011B          ;Send read ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT

RECVDATA:
        MOV     A,#00000100B          ;Send RECV command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        CALL    WAIT
        MOV     DPTR,#I2CRXD          ;Read data from the data buffer
        MOVX    A,@DPTR
        RET

SENDACK:
        MOV     A,#00000000B          ;Setup the ACK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B          ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT

SENDNAK:
        MOV     A,#00000001B          ;Setup the NAK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B          ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT

STOP:
        MOV     A,#00000110B          ;Send STOP command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT

WAIT:
        MOV     DPTR,#I2CMSST          ;Clear interrupt flag
        MOVX    A,@DPTR
        JNB     ACC.6,WAIT
        ANL     A,#NOT 40H
        MOVX    @DPTR,A
        RET

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

DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

```

MAIN:

```

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

MOV      P_SW2, #80H

MOV      A, #11100000B           ;Set the I2C module as master mode
MOV      DPTR, #I2CCFG
MOVX     @DPTR, A
MOV      A, #00000000B
MOV      DPTR, #I2CMSST
MOVX     @DPTR, A

CALL     START                   ;Send start command
MOV      A, #5AH                 ;Slave address is 5A
CALL     SENDDATA                ;Send device address (010_1101B) + write command (0b)
CALL     RECVACK
MOV      A, #000H                ;Send storage address
CALL     SENDDATA
CALL     RECVACK
MOV      A, #12H                 ;Write test data 1
CALL     SENDDATA
CALL     RECVACK
MOV      A, #78H                 ;Write test data 2
CALL     SENDDATA
CALL     RECVACK
CALL     STOP                    ;Send stop command

CALL     DELAY                   ;Waiting for the device to write data

CALL     START                   ;Send start command
MOV      A, #5AH                 ;Send device address (010_1101B) + write command (0b)
CALL     SENDDATA
CALL     RECVACK
MOV      A, #000H                ;Send storage address
CALL     SENDDATA
CALL     RECVACK
CALL     START                   ;Send start command
MOV      A, #5BH                 ;Send device address (010_1101B) + read command (1b)
CALL     SENDDATA
CALL     RECVACK
CALL     RECVDATA                ;Read data 1
MOV      P0, A
CALL     SENDACK
CALL     RECVDATA                ;Read data 2
MOV      P2, A
CALL     SENDNAK

```

CALL *STOP* ;Send stop command

JMP \$

END

20 16-bit advanced PWM timer, support quadrature encoder

Product line	Advanced PWM
STC8H1K08 family	●
STC8H1K28 family	●
STC8H3K64S4 family	●
STC8H3K64S2 family	●
STC8H8K64U family	●
STC8H2K64T family	●
STC8H4K64TLR family	●
STC8H4K64TLCD family	●
STC8H4K64LCD family	●

The STC8H series of microcomputers integrate 8-channel 16-bit advanced PWM timers, which are divided into two groups of PWMs with different periods, named PWMA and PWMB (the previous data sheet used to be named PWM1 and PWM2, but it is easy to match the chip pin name for confusion, it is changed to PWMA and PWMB), which can be set separately. The first group of PWM/PWMA can be configured as 4 groups of complementary/symmetrical/dead-zone controlled PWM or capture external signals, and the second group of PWM/PWMB can be configured as 4 channels of PWM output or capture external signals.

The clock frequency of the first set of PWM/PWMA can be the system clock divided by the registers [PWMA_PSCRH](#) and [PWMA_PSCRL](#). The frequency division value can be any value between 1~65535. The clock frequency of the second group of PWM/PWMB can be the system clock divided by the [PWMB_PSCRH](#) [PWMB_PSCRL](#) register, and the divided value can be any value between 1 and 65535. The clock frequencies of the two sets of PWM can be set independently.

The first set of PWM timers/PWMA has 4 channels (PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N), each channel can independently realize PWM output (complementary symmetrical PWM output with dead zone can be set), capture and compare functions; the second group of PWM timer/PWMB has 4 channels (PWM5, PWM6, PWM7, PWM8), and each channel can also independently implement PWM output, capture and compare functions. The only difference between the two sets of PWM timers is that the first set can output complementary symmetrical PWM with dead zone, while the second set can only output single-ended PWM, and other functions are exactly the same. The following description of advanced PWM timers only takes the first group as an example.

When using the first set of PWM timers to output PWM waveforms, the PWM1P/PWM2P/PWM3P/PWM4P output can be individually enabled, or the PWM1N/PWM2N/PWM3N/PWM4N output can be individually enabled. For example: if PWM1P output is enabled separately, PWM1N can no longer output independently, unless PWM1P and PWM1N form a set of complementary symmetrical output. The 4 outputs of PWMA can be set independently, for example: PWM1P and PWM2N output can be individually enabled, or PWM2N and PWM3N output can be individually enabled. If you need to use the first set of PWM timers for capture function or pulse width measurement, the input signal can only be input from the positive end of each channel, that is, only PWM1P/PWM2P/PWM3P/PWM4P have the capture function and pulse width measurement function.

When the two sets of advanced PWM timers capture external signals, you can select rising edge capture or

falling edge capture. If you need to capture the rising and falling edges at the same time, you can connect the input signal to two PWMs at the same time, and enable one of them to capture the rising edge and the other to capture the falling edge. **What's more powerful is that when the external input signal is connected to two PWMs at the same time, the period value and duty ratio value of the signal can be captured at the same time.**

Comparison of STC three hardware PWMs:

Compatible with traditional 8051 PCA/CCP/PWM: It can output PWM waveforms, capture external input signals and output high-speed pulses. It can output 6-bit/7-bit/8-bit/10-bit PWM waveform externally. The frequency of the 6-bit PWM waveform is the PCA module clock source frequency/64; the frequency of the 7-bit PWM waveform is the PCA module clock source frequency/128; 8-bit The frequency of the PWM waveform is the PCA module clock source frequency/256; the frequency of the 10-bit PWM waveform is the PCA module clock source frequency/1024. Capture the external input signal, you can capture the rising edge, the falling edge, or both the rising edge and the falling edge.

15-bit enhanced PWM of STC8G series: It can only output PWM waveform externally, without input capture function. The frequency and duty cycle of external output PWM can be set arbitrarily. Through software intervention, multi-channel complementary/symmetrical/with dead zone PWM waveform can be realized. It has external abnormality detection function and real-time trigger ADC conversion function.

The 16-bit advanced PWM timer of the STC8H series: It is the PWM with the strongest STC function at present, and can output PWM waveforms of any frequency and any duty cycle. It can output complementary/symmetrical/with dead zone PWM waveform without software intervention. It can capture the external input signal, can capture the rising edge, the falling edge or the rising edge and the falling edge at the same time, when measuring the external waveform, the period value and the duty ratio value of the waveform can be measured at the same time. There are quadrature encoding function, external anomaly detection function and real-time trigger ADC conversion function.

In the following description, PWMA represents the first group of PWM timers, and PWMB represents the second group of PWM timers

The first group of advanced PWM timer/PWMB internal signal description

TI1: External clock input signal 1 (PWM1P pin signal or the signal after the exclusive OR of PWM1P/PWM2P/PWM3P)

TI1F: TI1 signal after IC1F digital filtering

TI1FP: TI1F signal after CC1P/CC2P edge detector

TI1F_ED: TI1F edge signal

TI1FP1: TI1F signal after CC1P edge detector

TI1FP2: TI1F signal after CC2P edge detector

IC1: Capture input signal of channel 1 selected by CC1S

OC1REF: output reference waveform of channel 1 (middle waveform)

OC1: Main output signal of channel 1 (OC1REF signal after CC1P polarity processing)

OC1N: Complementary output signal of channel 1 (OC1REF signal after CC1NP polarity processing)

TI2: External clock input signal 2 (PWM2P pin signal)

TI2F: TI2 signal after IC2F digital filtering

TI2F_ED: TI2F edge signal

TI2FP: TI2F signal after CC1P/CC2P edge detector

TI2FP1: TI2F signal after CC1P edge detector

TI2FP2: TI2F signal after CC2P edge detector

IC2: Capture input signal of channel 2 selected by CC2S

OC2REF: output reference waveform of channel 2 (middle waveform)

OC2: Main output signal of channel 2 (OC2REF signal after CC2P polarity processing)

OC2N: Complementary output signal of channel 2 (OC2REF signal after CC2NP polarity processing)

TI3: External clock input signal 3 (PWM3P pin signal)

TI3F: TI3 signal after IC3F digital filtering

TI3F_ED: TI3F edge signal

TI3FP: TI3F signal after CC3P/CC4P edge detector

TI3FP3: TI3F signal after CC3P edge detector

TI3FP4: TI3F signal after CC4P edge detector

IC3: Capture input signal of channel 3 selected by CC3S

OC3REF: output reference waveform of channel 3 (middle waveform)

OC3: Main output signal of channel 3 (OC3REF signal after CC3P polarity processing)

OC3N: Complementary output signal of channel 3 (OC3REF signal after CC3NP polarity processing)

TI4: External clock input signal 4 (PWM4P pin signal)

TI4F: TI4 signal after IC4F digital filtering

TI4F_ED: TI4F edge signal

TI4FP: TI4F signal after CC3P/CC4P edge detector

TI4FP3: TI4F signal after CC3P edge detector

TI4FP4: TI4F signal after CC4P edge detector

IC4: The capture input signal of channel 4 selected by CC4S

OC4REF: output reference waveform of channel 4 (middle waveform)

OC4: Main output signal of channel 4 (OC4REF signal after CC4P polarity processing)

OC4N: Complementary output signal of channel 4 (OC4REF signal after CC4NP polarity processing)

ITR1: Internal trigger input signal 1

ITR2: Internal trigger input signal 2

TRC: fixed to TI1_ED

TRGI: Trigger input signal after TS multiplexer

TRGO: Trigger output signal after MMS multiplexer

ETR: External trigger input signal (PWMETI1 pin signal)

ETRP: ETR signal after passing ETP edge detector and ETPS divider

ETRF: ETRP signal after ETF digital filtering

BRK: Brake input signal (PWMFLT)

CK_PSC: prescaler clock, PWMA_PSCR prescaler input clock

CK_CNT: PWMA_PSCR prescaler output clock, PWM timer clock

The second group of advanced PWM timer/PWMB internal signal description

TI5: External clock input signal 5 (PWM5 pin signal or signal after the exclusive OR of PWM5/PWM6/PWM7)

TI5F: TI5 signal after IC5F digital filtering

TI5FP: TI5F signal after CC5P/CC6P edge detector

TI5F_ED: TI5F edge signal

TI5FP5: TI5F signal after CC5P edge detector

TI5FP6: TI5F signal after CC6P edge detector

IC5: The capture input signal of channel 5 selected by CC5S

OC5REF: output reference waveform of channel 5 (middle waveform)

OC5: Main output signal of channel 5 (OC5REF signal after CC5P polarity processing)

TI6: External clock input signal 6 (PWM6 pin signal)

TI6F: TI6 signal after IC6F digital filtering

TI6F_ED: TI6F edge signal

TI6FP: TI6F signal after CC5P/CC6P edge detector

TI6FP5: TI6F signal after CC5P edge detector

TI6FP6: TI6F signal after CC6P edge detector

IC6: The capture input signal of channel 6 selected by CC6S

OC6REF: output reference waveform of channel 6 (middle waveform)

OC6: Main output signal of channel 6 (OC6REF signal after CC6P polarity processing)

TI7: External clock input signal 7 (PWM7 pin signal)

TI7F: TI7 signal after IC7F digital filtering

TI7F_ED: TI7F edge signal

TI7FP: TI7F signal after CC7P/CC8P edge detector

TI7FP7: TI7F signal after CC7P edge detector

TI7FP8: TI7F signal after CC8P edge detector

IC7: Capture input signal of channel 7 selected by CC7S

OC7REF: output reference waveform of channel 7 (middle waveform)

OC7: Main output signal of channel 7 (OC7REF signal after CC7P polarity processing)

TI8: External clock input signal 8 (PWM8 pin signal)

TI8F: TI8 signal after IC8F digital filtering

TI8F_ED: TI8F edge signal

TI8FP: TI8F signal after CC7P/CC8P edge detector

TI8FP7: TI8F signal after CC7P edge detector

TI8FP8: TI8F signal after CC8P edge detector

IC8: The capture input signal of channel 8 selected by CC8S

OC8REF: output reference waveform of channel 8 (middle waveform)

OC8: Main output signal of channel 8 (OC8REF signal after CC8P polarity processing)

20.1 Introduction

PWMA consists of a 16-bit auto-load counter, which is driven by a programmable prescaler.

PWMA is suitable for many different purposes:

- Basic timing
- Measure the pulse width of the input signal (input capture)
- Generate output waveforms (output compare, PWM and single pulse mode)
- Corresponding to interrupts of different events (capture, compare, overflow, brake, trigger)
- Synchronize with PWMB or external signals (external clock, reset signal, trigger and enable signal)

PWMA is widely used in a variety of control applications, including those that require mid-aligned mode PWM, which supports complementary output and dead time control. The clock source of PWMA can be an internal clock or an external signal, which can be selected through the configuration register.

20.2 Main features

Features of PWMA include:

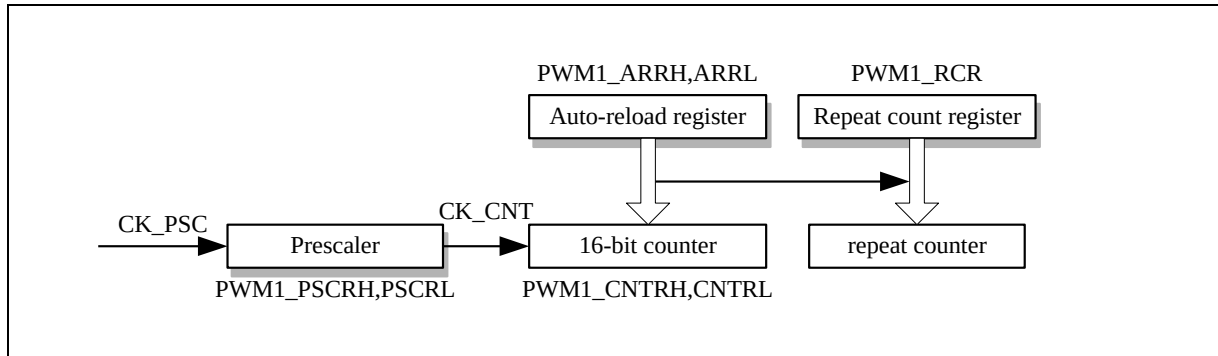
- 16-bit up, down, up/down auto load counter
- Allows to update the repeat counter of the timer register after a specified number of counter cycles
- 16-bit programmable (can be modified in real time) prescaler, the division coefficient of the counter clock frequency is any value between 1~65535
- Synchronization circuit, used to control timer and timer interconnection with external signal
- Up to 4 independent channels can be configured as:
 - Input capture
 - Output comparison
 - PWM output (edge or center alignment mode)
 - Six-step PWM output
 - Single pulse mode output
 - Supports complementary output on 4 channels with programmable dead time
- The brake input signal (PWMFLT) can put the timer output signal in a reset state or a certain state
- External trigger input pin (PWMETI)
- Events that generate interrupts include:
 - Update: Counter overflow/downflow, counter initialization (by software or internal/external trigger)
 - Trigger event (counter start, stop, initialization or count by internal/external trigger)
 - Input capture, measure pulse width
- External Interrupt
- Output comparison
- Brake signal input

20.3 Time base unit

The time base unit of PWMA includes:

- 16-bit up/down counter
- 16-bit automatic reload register
- Repeat counter
- Prescaler

PWMA time base unit



The 16-bit counter, prescaler, auto-reload register and repeat counter register can all be read and written by software. The auto-reload register is composed of preload register and shadow register.

The automatic reload register can be written in two modes:

- Auto preloading is enabled (ARPE bit of PWMA_CR1 register is 1). In this mode, the data written into the auto-reload register will be saved in the preload register and transferred to the shadow register at the next update event (UEV).
- Auto preloading is disabled (ARPE bit of PWMA_CR1 register is 0). In this mode, the data written to the auto-reload register will be written to the shadow register immediately.

Conditions for generating update events:

- The counter overflows up or down.
- The software sets the UG bit in the PWMA_EGR register.
- The clock/trigger controller generates a trigger event.

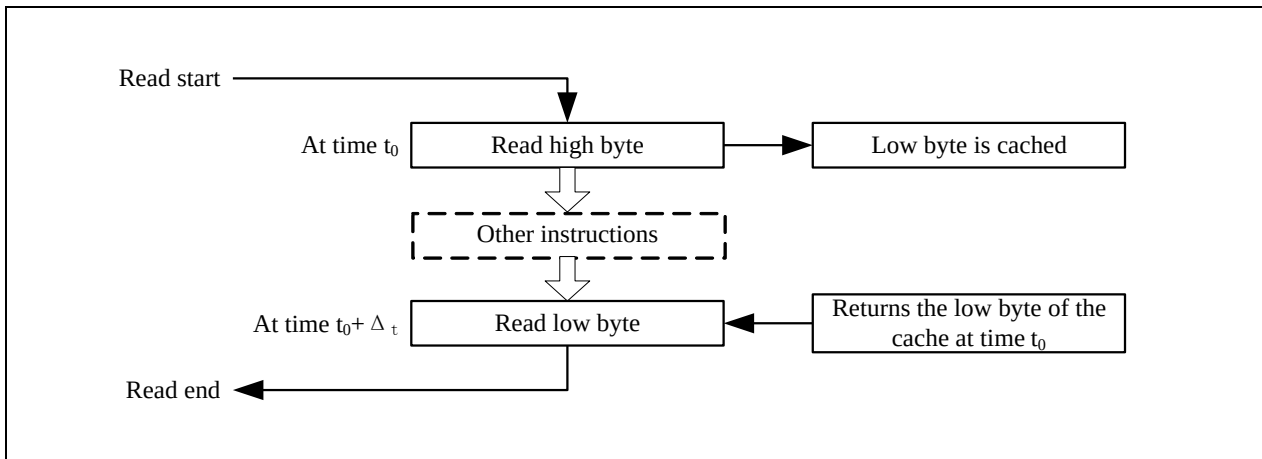
When the preload is enabled (ARPE=1), if an update event occurs, the value in the preload register (PWMA_ARR) will be written into the shadow register, and the value in the PWMA_PSCR register will be written into the prescaler. Setting the UDIS bit in the PWMA_CR1 register will disable the update event (UEV). The output of the prescaler CK_CNT drives the counter, and CK_CNT is only valid when the counter enable bit (CEN) of the PWMA_CR1 register is set.

Note: The actual counter does not start counting until one clock cycle after the CEN bit is enabled.

20.3.1 Reading and writing 16-bit counter

There is no buffer for writing counter operations, and PWMA_CNTRH and PWMA_CNTRL registers can be written at any time. Therefore, in order to avoid writing wrong values, it is generally recommended not to write new values while the counter is running.

The operation of the read counter has an 8-bit buffer. The user must first read the high byte of the timer. After the user reads the high byte, the low byte will be automatically buffered, and the buffered data will be kept until the 16-bit data read operation is completed.



20.3.2 Writing operation of 16-bit PWMA_ARR register

The value in the preload register will be written into the 16-bit PWMA_ARR register. This operation is completed by two instructions, each of which writes 1 byte. The high byte must be written first, followed by the low byte.

The shadow register is locked when the high byte is written, and remains until the low byte is written.

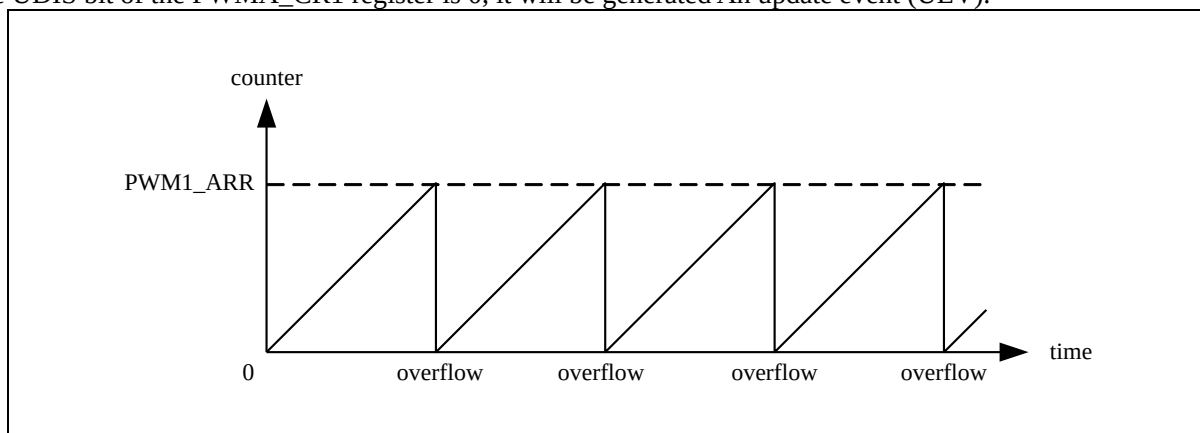
20.3.3 Prescaler

The prescaler of PWMA is based on a 16-bit counter controlled by a 16-bit register (PWMA_PSCR). Since this control register has a buffer, it can be changed at runtime. The prescaler can divide the counter clock frequency by any value between 1 and 65536. The value of the prescaler is written by the preload register, and the shadow register holding the current value is loaded when the low byte is written. Since two separate write operations are required to write 16-bit registers, it is necessary to ensure that the high byte is written first. The new prescaler value is used when the next update event arrives. The read operation of the PWMA_PSCR register is completed through the preload register.

Frequency calculation formula of counter: $f_{CK_{CNT}} = f_{CK_{PSC}} \frac{1}{PSCR[15:0] + 1}$

20.3.4 Up counting mode

In the up-counting mode, the counter counts from 0 to the user-defined comparison value (the value of the PWMA_ARR register), and then restarts counting from 0 and generates a counter overflow event. At this time, if the UDIS bit of the PWMA_CR1 register is 0, it will be generated An update event (UEV).



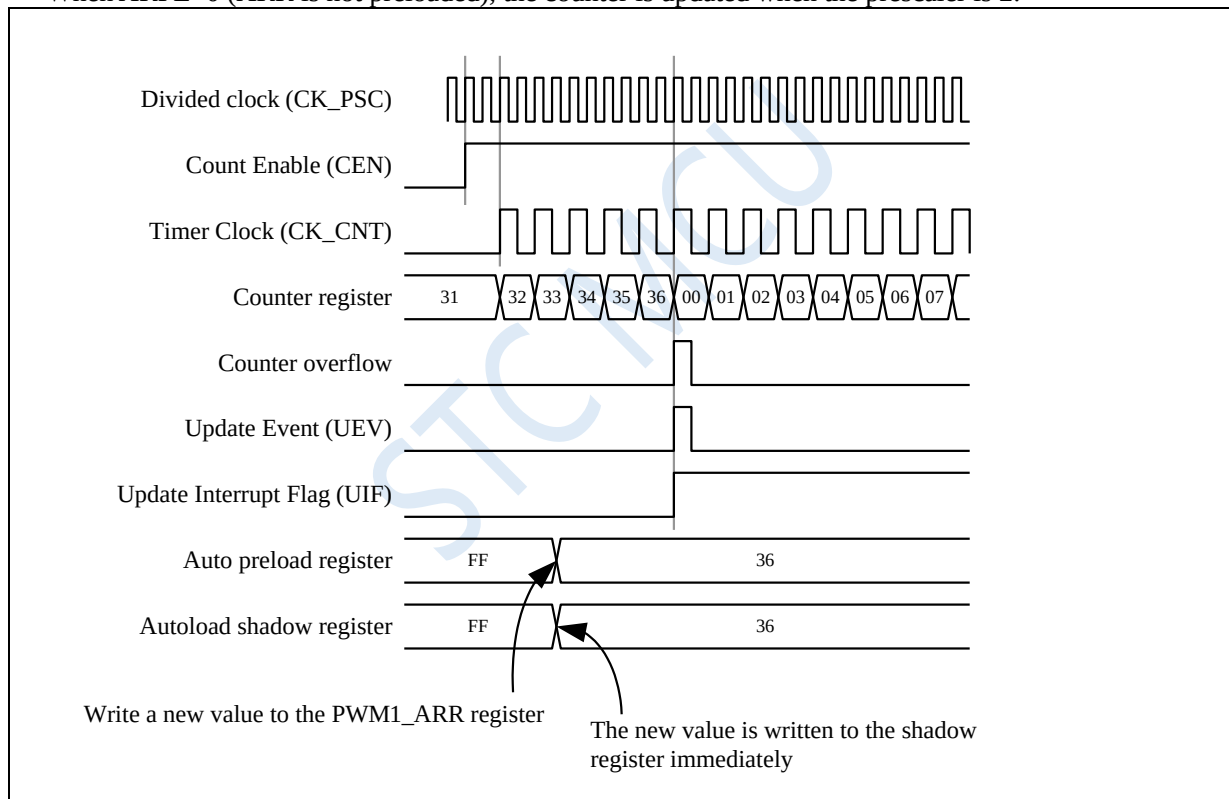
An update event can also be generated by software or by using the trigger controller to set the UG bit of the PWMA_EGR register. Use software to set the UDIS bit of the PWMA_CR1 register to disable the update event, which can avoid updating the shadow register when updating the preload register. Until the UDIS bit is cleared, no update event will be generated. But when an update event should occur, the counter will still be cleared, and the count of the prescaler will also be cleared (but the value of the prescaler will not change). In addition, if the URS bit in the PWMA_CR1 register (select update request) is set, setting the UG bit will generate an update event UEV, but the hardware does not set the UIF flag (that is, no interrupt request is generated). This is to avoid the update and capture interrupts when the counter is cleared in capture mode.

When an update event occurs, all registers are updated, and the hardware simultaneously sets the update flag (the UIF bit of the PWMA_SR register) according to the URS bit:

- The autoloading shadow register is reset to the value of the preload register (PWMA_ARR).
- The buffer of the prescaler is set into the value of the preload register (PWMA_PSC).

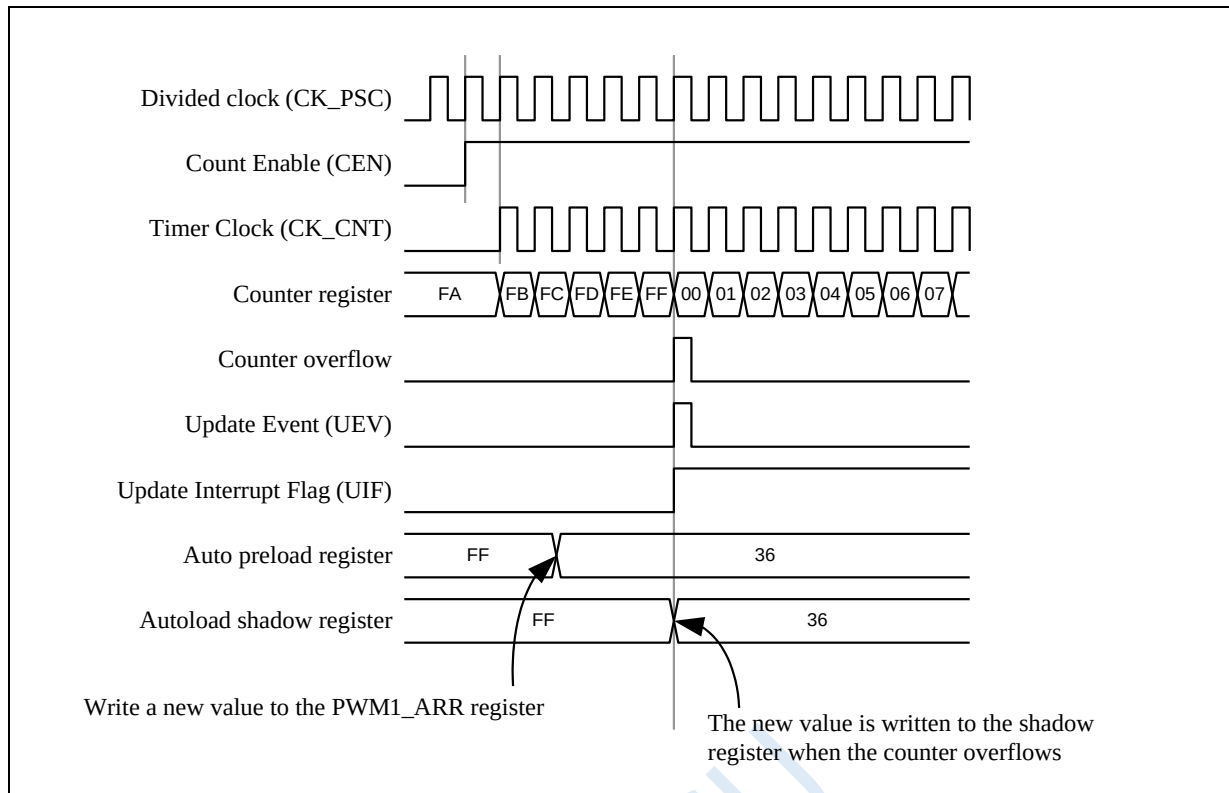
The following figure gives some examples to illustrate the actions of the counter at different clock frequencies when PWMA_ARR=0x36. The prescaler in the figure is 2, so the counter clock (CK_CNT) frequency is half of the prescaler clock (CK_PSC) frequency. In the figure, the automatic loading function is disabled (ARPE=0), so when the counter reaches 0x36, the counter overflows, the shadow register is updated immediately, and an update event is generated at the same time.

When ARPE=0 (ARR is not preloaded), the counter is updated when the prescaler is 2:



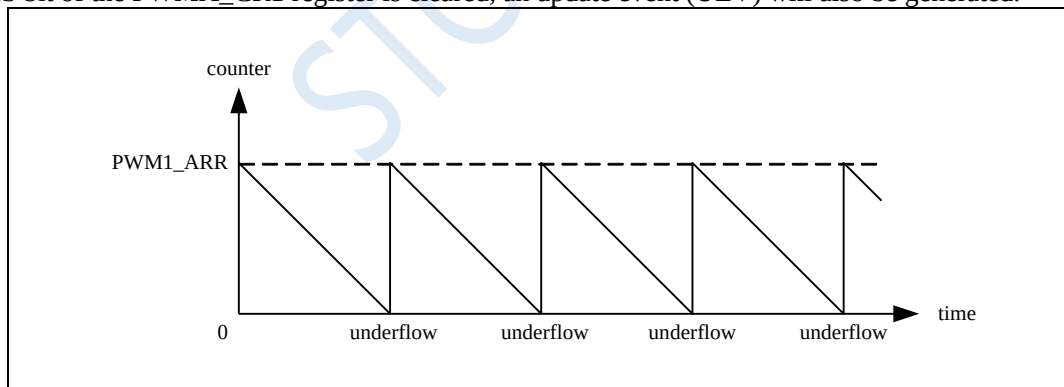
The prescaler in the figure below is 1, so the frequency of CK_CNT is the same as CK_PSC. In the figure, auto reload is enabled (ARPE=1), so an overflow occurs when the counter reaches 0xFF. 0x36 will be written on overflow and an update event will be generated at the same time.

When ARPE=1 (PWMA_ARR preload), the counter update when the prescale is 1:



20.3.5 Down counting mode

In the down mode, the counter starts counting down from the auto-loaded value (PWMA_ARR register value) to 0, and then restarts counting from the auto-loaded value, and a counter overflow event is generated. If the UDIS bit of the PWMA_CR1 register is cleared, an update event (UEV) will also be generated.



An update event can also be generated by software or by using the trigger controller to set the UG bit of the PWMA_EGR register. The UEV event can be disabled by setting the UDIS bit in the PWMA_CR1 register. This avoids updating the shadow register when updating the preload register. Therefore, no update event will be generated before the UDIS bit is cleared. However, the counter will still start counting from the current autoloader value, and the counter of the prescaler will restart from 0 (but the prescaler cannot be modified). In addition, if the URS bit (select update request) in the PWMA_CR1 register is set, setting the UG bit will generate an update event UEV without setting the UIF flag (so no interrupt is generated). This is to avoid the occurrence of a capture event and clearing the counter, Simultaneously generate update and capture interrupts.

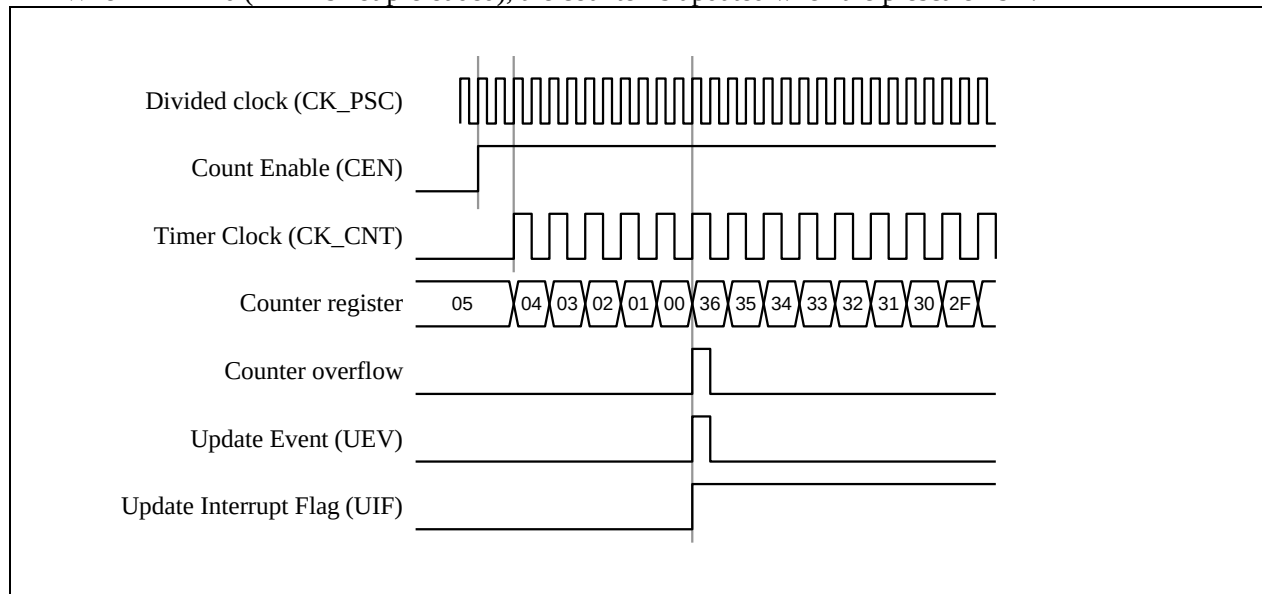
When an update event occurs, all registers are updated, and the hardware sets the update flag (the UIF bit of the PWMA_SR register) according to the URS bit at the same time:

- The autoloader shadow register is reset to the value of the preload register (PWMA_ARR).
- The buffer of the prescaler is set to the value of the preload register (PWMA_PSC).

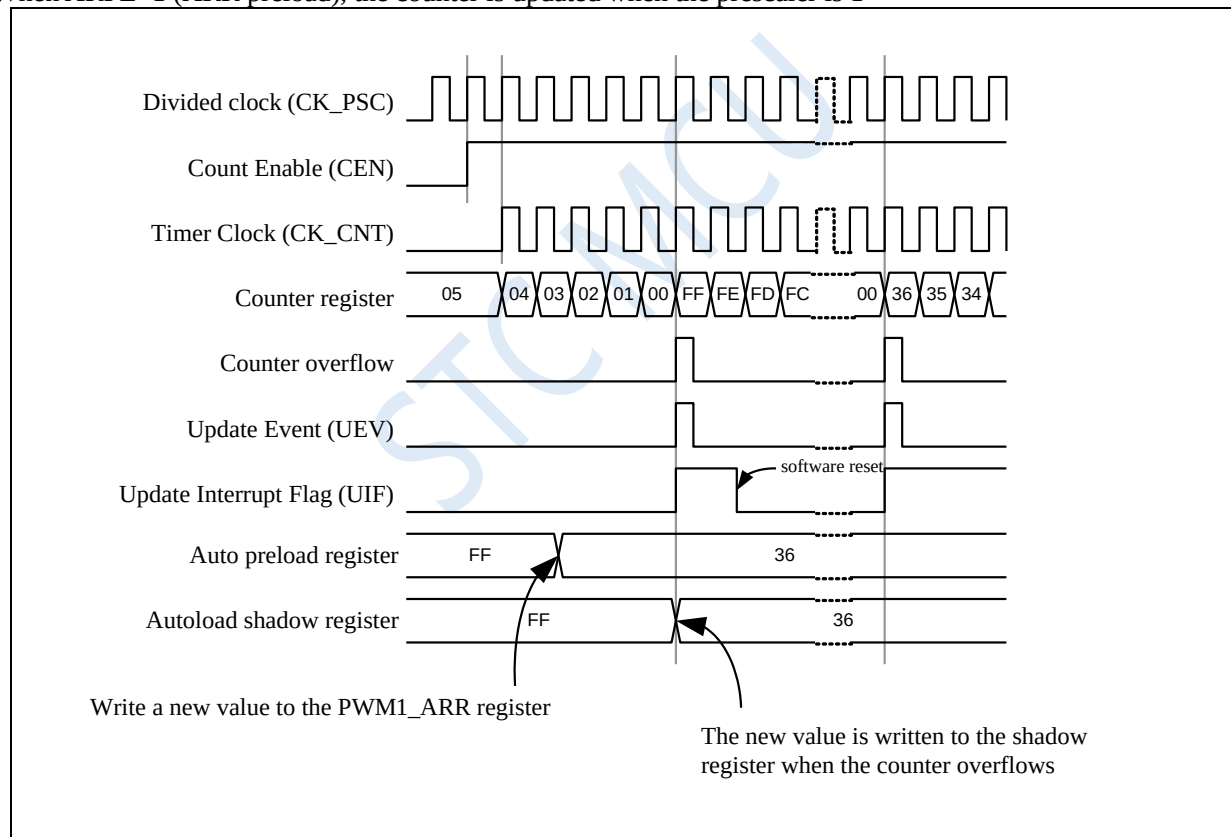
The following are some charts of the counter at different clock frequencies when PWMA_ARR=0x36. The following figure depicts that in the down-counting mode, the new value is written in the next cycle when

preloading is disabled.

When ARPE=0 (ARR is not preloaded), the counter is updated when the prescaler is 2:

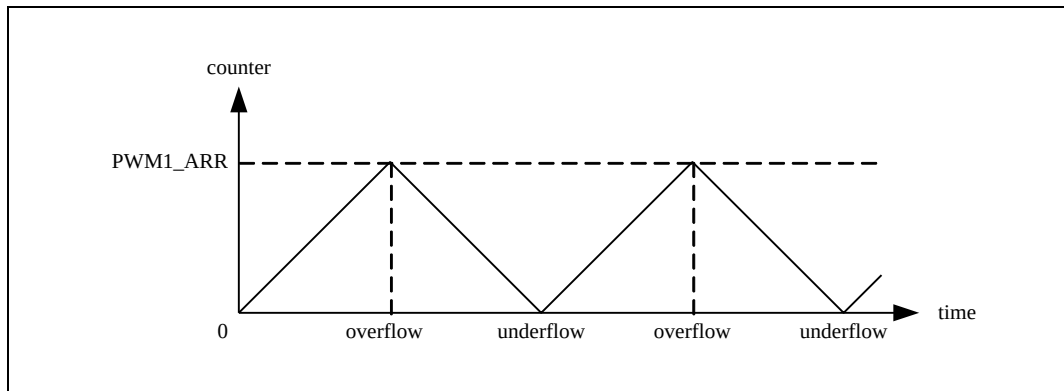


When ARPE=1 (ARR preload), the counter is updated when the prescaler is 1



20.3.6 Center alignment mode (up/down count)

In the center-aligned mode, the counter starts counting from 0 to the value of the PWMA_ARR register, generates a counter overflow event, then counts down from the value of the PWMA_ARR register to 0 and generates a counter underflow event; then restarts counting from 0. In this mode, the DIR direction bit in PWMA_CR1 cannot be written. It is updated by hardware and indicates the current counting direction.



If the timer has a repetition counter, an update event (UEV) will be generated after the up and down overflow of the specified number of times (the value of PWMA_RCR) is repeated. Otherwise, each upward and downward overflow will generate an update event. An update event can also be generated by software or by using the trigger controller to set the UG bit of the PWMA_EGR register. At this time, the counter starts counting from 0 again, and the prescaler also starts counting from 0 again. The UEV event can be disabled by setting the UDIS bit in the PWMA_CR1 register. This avoids updating the shadow register when updating the preload register. Therefore, no update event will be generated before the UDIS bit is cleared to 0. However, the counter will continue to count up or down according to the current auto-reload value. If the timer has a repeat counter, since the repeat register is not double-buffered, the new repeat value will take effect immediately, so be careful when modifying it. In addition, if the URS bit (select update request) in the PWMA_CR1 register is set, setting the UG bit will generate an update event UEV but does not set the UIF flag (so no interrupt is generated). This is to avoid the occurrence of a capture event and clear the counter, Simultaneously generate update and capture interrupts.

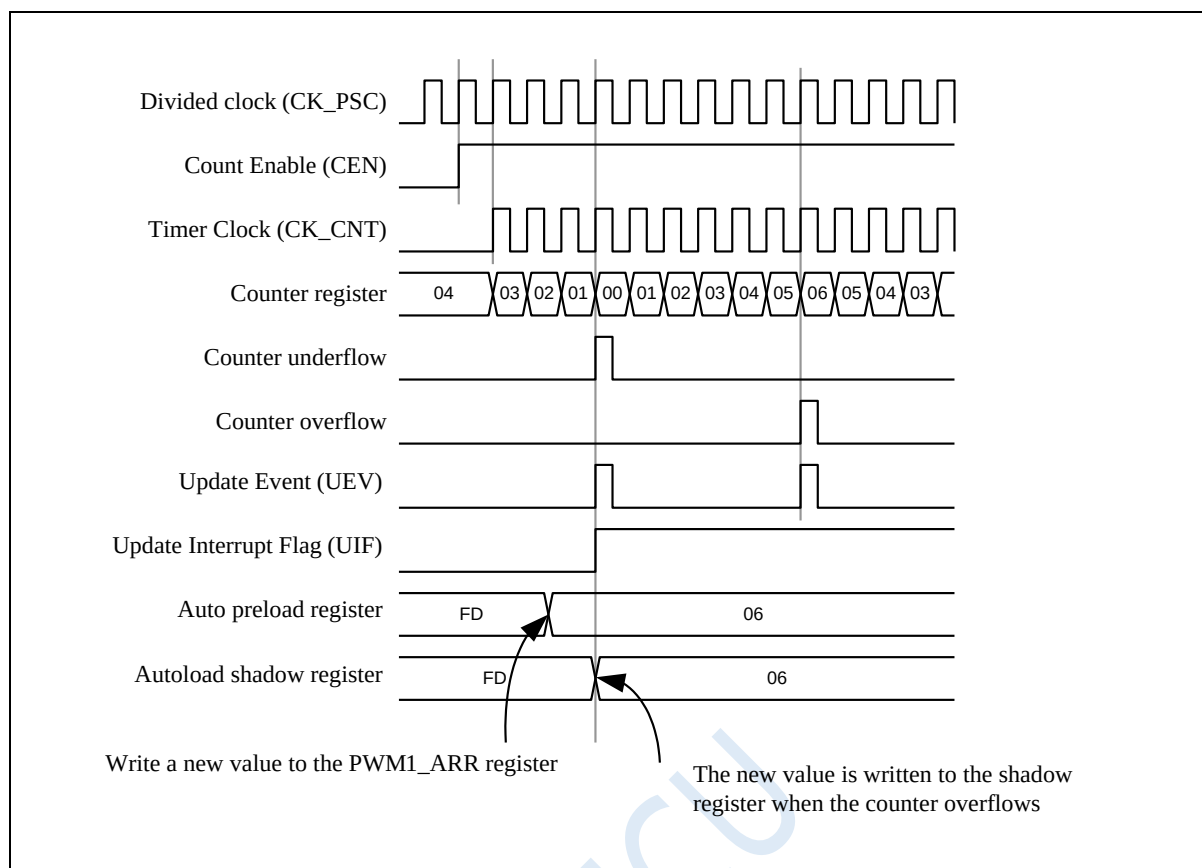
When an update event occurs, all registers are updated, and the hardware updates the flag bit according to the URS bit (the UIF bit in the PWMA_SR register):

- The register of the prescaler is loaded with the preloaded value (PWMA_PSCR).
- The current autoloader register is updated to the preload value (PWMA_ARR).

It should be noted that if an update occurs due to a counter overflow, the auto-reload register will be updated before the counter is reloaded, so the next cycle is the expected value (the counter is loaded with the new value).

The following are some examples of counter operations at different clock frequencies:

The internal clock frequency division factor is 1, PWMA_ARR=0x6, ARPE=1



Tips for using center alignment mode:

- When the center alignment mode is activated, the counter will count according to the original up/down configuration. In other words, the DIR bit in the PWMA_CR1 register will determine whether the counter counts up or down. In addition, the software cannot modify the value of DIR bit and CMS bit at the same time.
- It is not recommended to write the counter value when the counter is counting in the center-aligned mode. This will cause unpredictable consequences. Specifically:
 - When a value larger than the auto-load value is written to the counter (PWMA_CNT > PWMA_ARR), the counting direction of the counter does not change. For example, the counter has overflowed, but the counter is still counting up.
 - Write 0 or the value of PWMA_ARR to the counter, but the update event does not occur.
- The safe way to use the counter in center-aligned mode is to use software (set the UG bit of the PWMA_EGR register) to generate an update event before starting the counter, and not to modify the counter value when the counter is counting.

20.3.7 Repeat counter

The time base unit explains how the update event (UEV) is generated when the counter overflows/downflows, but in fact it can only be generated when the value of the repeat counter reaches 0. This feature is very useful for generating PWM signals.

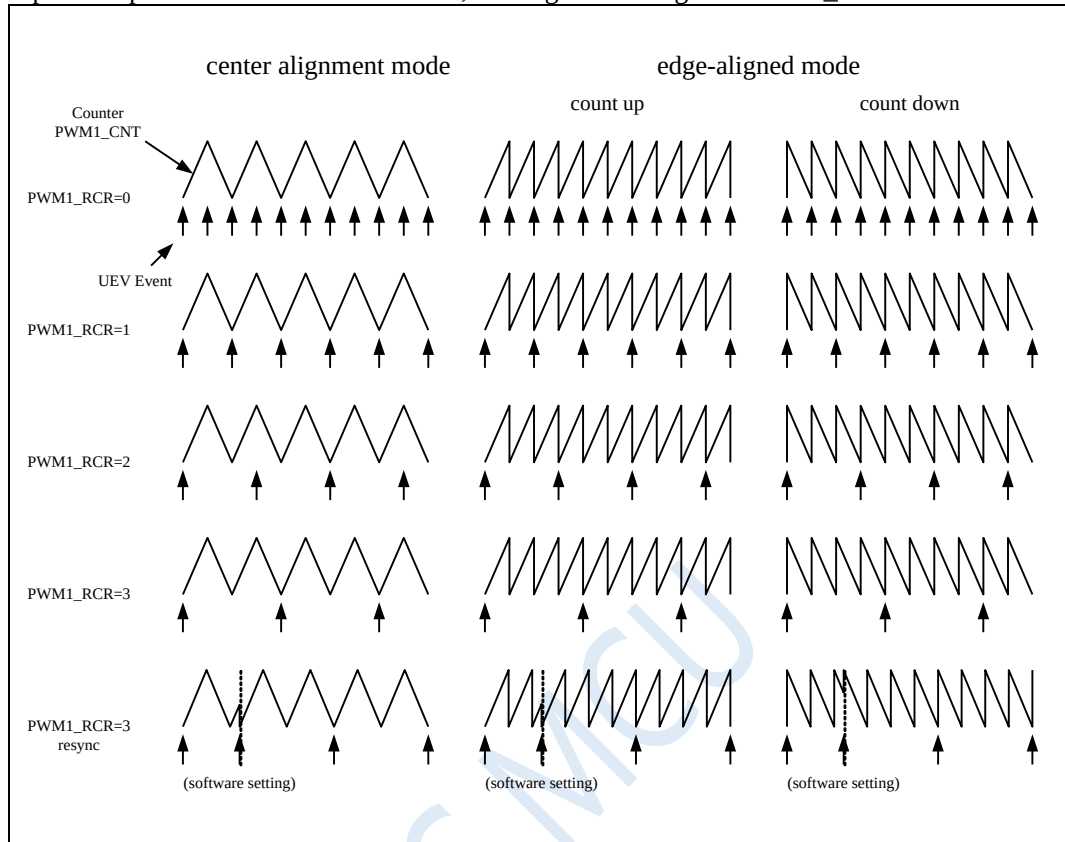
This means that every N counts overflow or underflow, the data is transferred from the preload register to the shadow register (PWMA_ARR auto-reload register, PWMA_PSCR preload register, and capture/compare register in compare mode PWMA_CCRx), N is the value in the PWMA_RCR repeat count register.

The repeat counter is decremented when any of the following conditions are met:

- Each time the counter overflows in the up-counting mode
- Every time the counter overflows in down counting mode
- At each overflow and each underflow in the center-aligned mode. Although this limits the maximum PWM cycle period to 128, it can update the duty cycle twice in each PWM cycle. In the center-aligned mode, because the waveform is symmetrical, if the compare register is refreshed only once in each PWM cycle, the maximum resolution is $2 \cdot t_{CK\_PSC}$.

The repetition counter is automatically loaded, and the repetition rate is defined by the value of the PWMA_RCR register. When the update event is generated by software or by the hardware clock/trigger controller, no matter what the value of the repeat counter is, the update event occurs immediately, and the content in the PWMA_RCR register is reloaded into the repeat counter.

Examples of update rates in different modes, and register settings of PWMA_RCR



20.4 Clock/Trigger Controller

The clock/trigger controller allows users to select the counter clock source, input trigger signal and output signal.

20.4.1 Prescaler Clock (CK_PSC)

The prescaler clock (CK_PSC) of the time base unit can be provided by the following sources:

- Internal clock (fMASTER)
- External clock mode 1: External clock input (TlX)
- External clock mode 2: External trigger input ETR
- Internal trigger input (ITRx): Use TRGO of one PWM as the prescaler clock of another PWM.

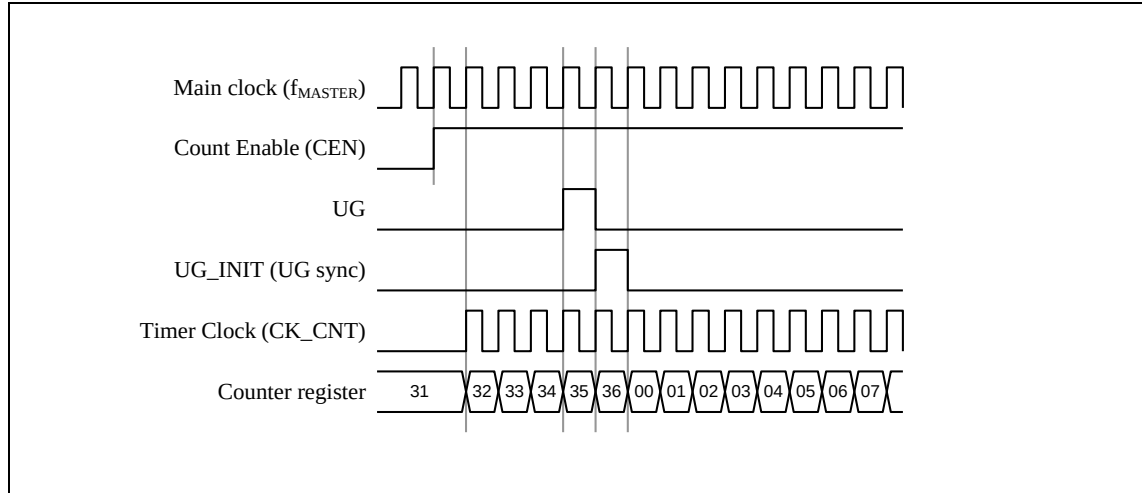
20.4.2 Internal clock source (fMASTER)

If the clock/trigger mode controller and external trigger input are disabled at the same time (SMS=000 in the PWMA_SMCR register, ECE=0 in the PWMA_ETR register), the CEN, DIR and UG bits are the actual control bits and can only be modified by software (The UG bit is still automatically cleared). Once the CEN bit is written as 1, the prescaler clock is provided by the internal clock.

The following figure describes the operation of the control circuit and up-counter in normal mode without

prescaler.

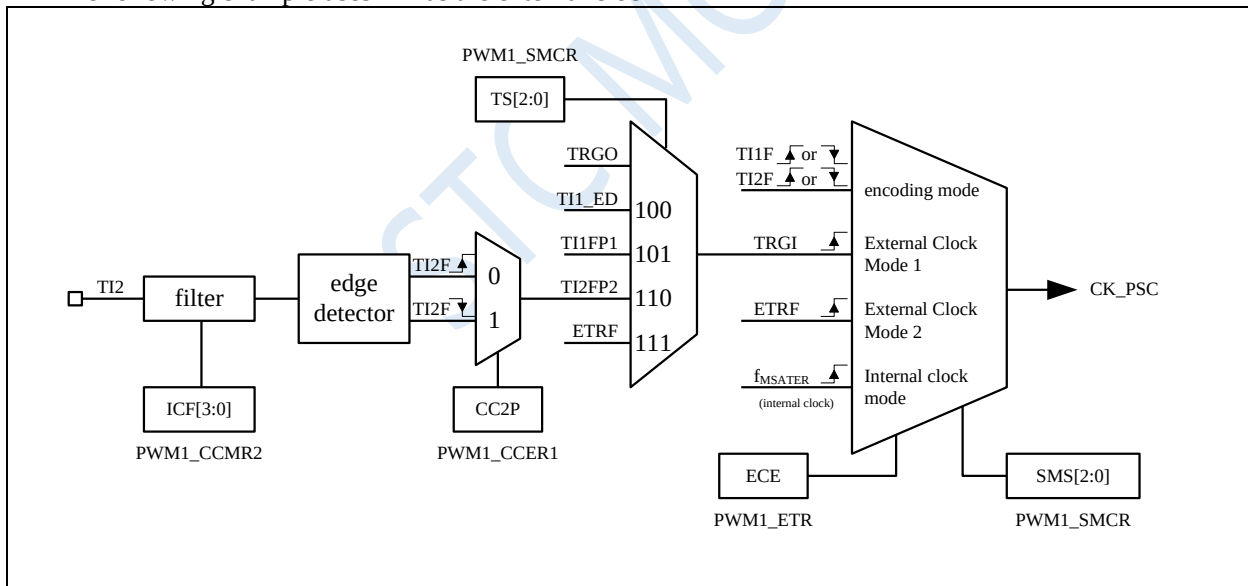
Control circuit in normal mode, f_{MASTER} division factor is 1



20.4.3 External clock source mode 1

When SMS=111 in PWMA_SMCR register, this mode is selected. Then select the signal source of TRGI through TS of PWMA_SMCR register. The counter can count on every rising or falling edge of the selected input.

The following example uses TI2 as the external clock



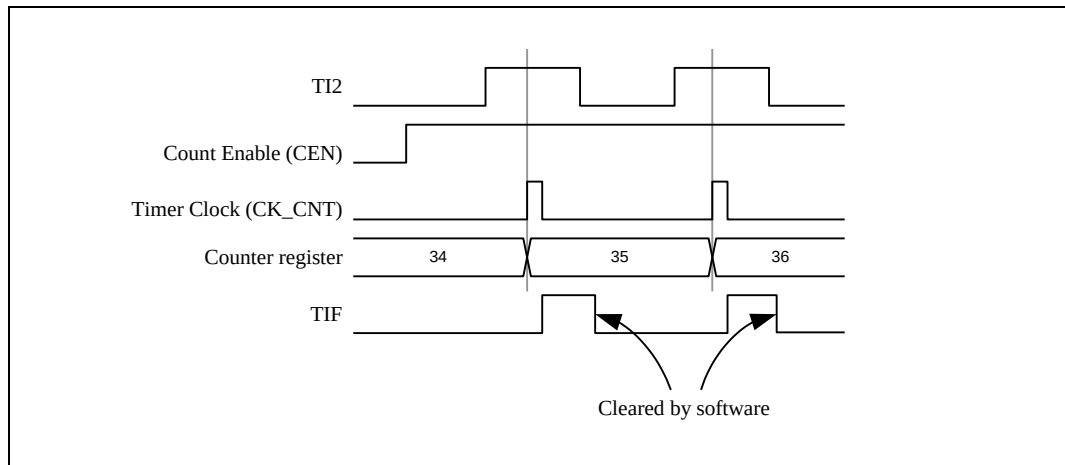
For example, to configure the up counter to count on the rising edge of the TI2 input, use the following steps:

1. Configure CC2S=01 in PWMA_CCMR2 register, use channel 2 to detect the rising edge of TI2 input
2. Configure the IC2F[3:0] bits of the PWMA_CCMR2 register to select the input filter bandwidth
3. Configure CC2P=0 in PWMA_CCER1 register, select the rising edge polarity
4. Configure SMS=111 in the PWMA_SMCR register, configure the counter to use external clock mode 1
5. Configure TS=110 in PWMA_SMCR register and select TI2 as input source
6. Set CEN=1 in PWMA_CR1 register to start the counter

When the rising edge occurs at TI2, the counter counts once, and the trigger flag (TIF bit of the PWMA_SR1 register) is set to 1, if the interrupt is enabled (configured in the PWMA_IER register), an interrupt request will be generated.

The delay between the rising edge of TI2 and the actual counter clock depends on the resynchronization circuit at the input of TI2.

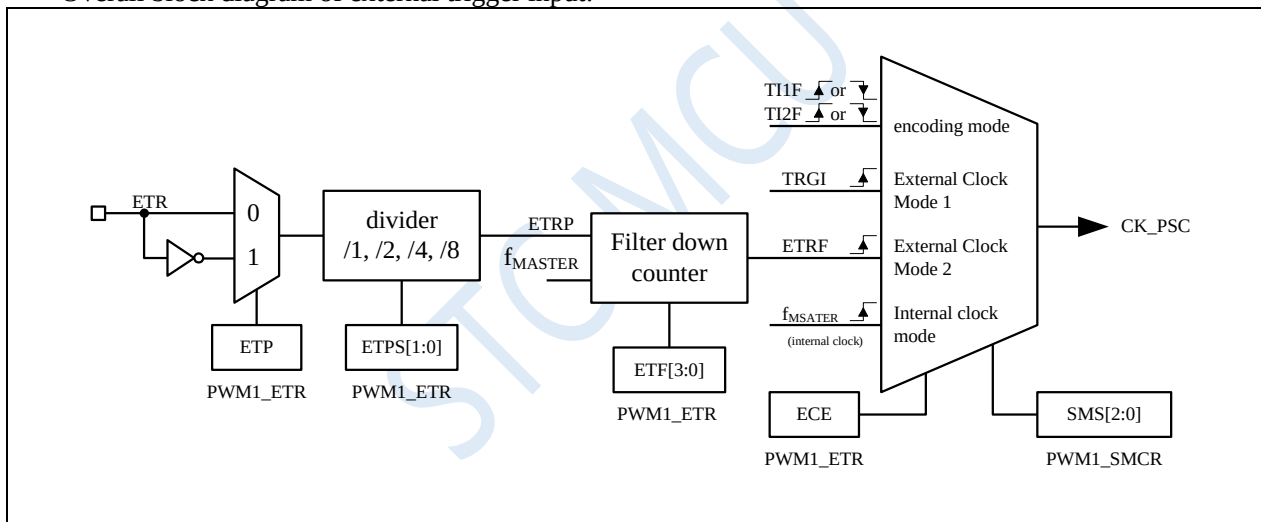
Control circuit in external clock mode 1



20.4.4 External clock source mode 2

The counter can count on every rising or falling edge of the external trigger input ETR signal. Write 1 to the ECE bit of the PWMA_ETR register to select this mode. (When SMS=111 in PWMA_SMCR register and TS=111 in PWMA_SMCR register, this mode can also be selected)

Overall block diagram of external trigger input:

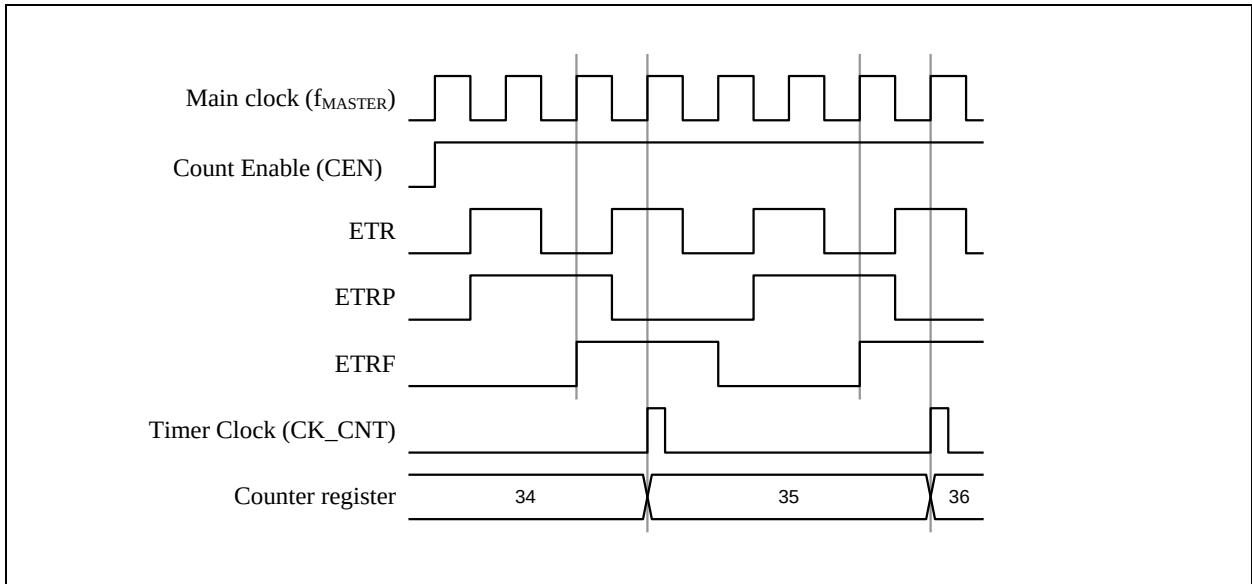


For example, to configure the counter to count up every 2 rising edges of the ETR signal, use the following steps:

1. No filter is needed in this example, configure ETF[3:0]=0000 in PWMA_ETR register
2. Set the prescaler, configure ETPS[1:0]=01 in the PWMA_ETR register
3. Select the rising edge detection of ETR, configure ETP=0 in the PWMA_ETR register
4. Turn on external clock mode 2, configure ECE=1 in the PWMA_ETR register
5. Start the counter, write CEN=1 in the PWMA_CR1 register

The counter counts once every 2 ETR rising edges.

Control circuit in external clock mode 2



20.4.5 Trigger synchronization

The PWMA counter uses three modes to synchronize with the external trigger signal:

- Standard trigger mode
- Reset trigger mode
- Gated trigger mode

Standard trigger mode

The enable of the counter (CEN) depends on the event on the selected input.

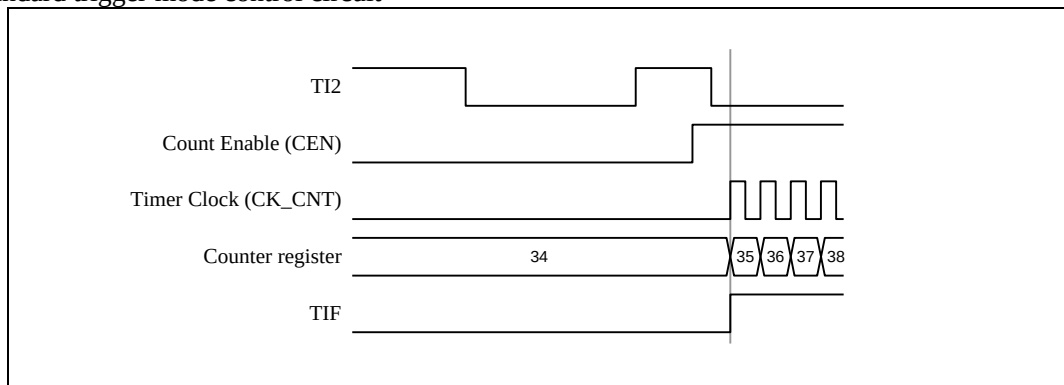
In the following example, the counter starts counting up on the rising edge of the TI2 input:

1. Configure CC2P=0 in the PWMA_CCER1 register and select the rising edge of TI2 as the trigger condition.
2. Configure SMS=110 in the PWMA_SMCR register and select the counter as trigger mode. Configure the PWMA_SMCR register

TS=110, select TI2 as the input source.

When TI2 has a rising edge, the counter starts to count under the internal clock drive and the TIF flag is set at the same time. The delay between the rising edge of TI2 and the counter starting to count depends on the resynchronization circuit at the TI2 input.

Standard trigger mode control circuit



Reset trigger mode

When a trigger input event occurs, the counter and its prescaler can be reinitialized. At the same time, if the URS bit of the PWMA_CR1 register is low, an update event UEV is also generated, and then all preload registers (PWMA_ARR, PWMA_CCRx) will be updated.

In the following example, the rising edge of the TI1 input causes the up counter to be cleared:

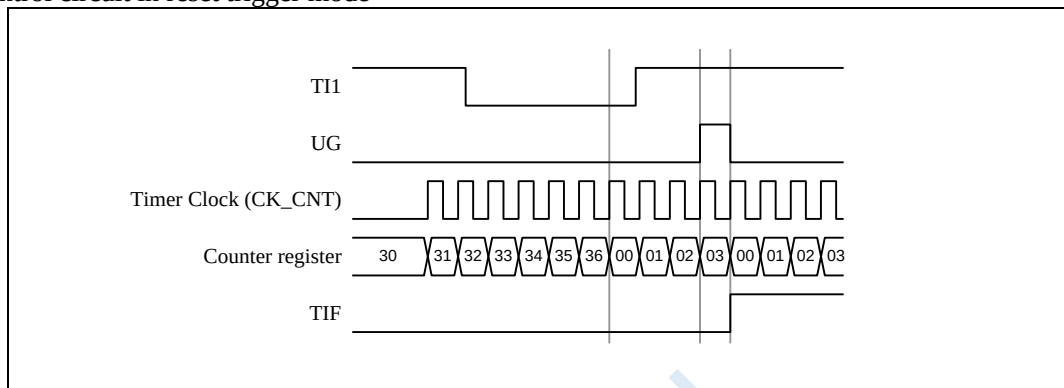
1. Configure CC1P=0 in the PWMA_CCER1 register to select the polarity of TI1 (only detect the rising edge of TI1).

2. Configure SMS=100 in the PWMA_SMCR register and select the timer as the reset trigger mode. Configure TS=101 in PWMA_SMCR register and select TI1 as input source.
3. Configure CEN=1 in the PWMA_CR1 register to start the counter.

The counter starts to count according to the internal clock, and then counts normally until TI1 has a rising edge. At this time, the counter is cleared and then restarts counting from 0. At the same time, the trigger flag (TIF bit of PWMA_SR1 register) is set. If the interrupt is enabled (TIE bit of PWMA_IER register), an interrupt request is generated.

The following figure shows the action when the auto reload register PWMA_ARR=0x36. The delay between the rising edge of TI1 and the actual reset of the counter depends on the resynchronization circuit at the input of TI1.

Control circuit in reset trigger mode



Gated trigger mode

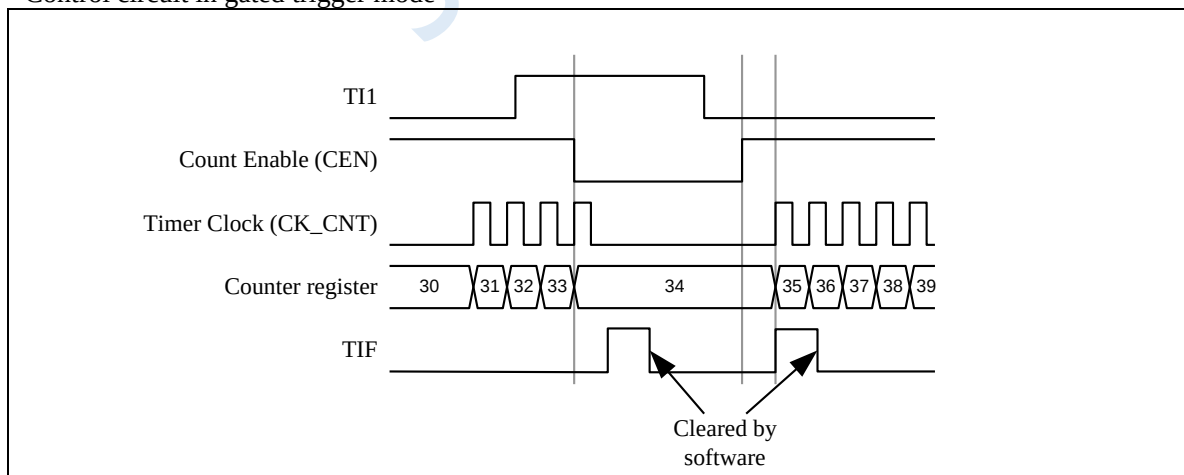
The counter is enabled by the level of the selected input signal.

In the following example, the counter only counts up when TI1 is low:

1. Configure CC1P=1 in the PWMA_CCER1 register to determine the polarity of TI1 (only detect the low level on TI1).
2. Configure SMS=101 in the PWMA_SMCR register, select the timer as gated trigger mode, configure TS=101 in the PWMA_SMCR register, and select TI1 as the input source.
3. Configure CEN=1 in the PWMA_CR1 register to start the counter (in gating mode, if CEN=0, the counter cannot be started, regardless of the trigger input level).

As long as TI1 is low, the counter starts counting according to the internal clock, and stops counting once TI1 goes high. The TIF flag will be set when the counter starts or stops. The delay between the rising edge of TI1 and the actual stop of the counter depends on the resynchronization circuit at the input of TI1.

Control circuit in gated trigger mode



External clock mode 2 Joint trigger mode

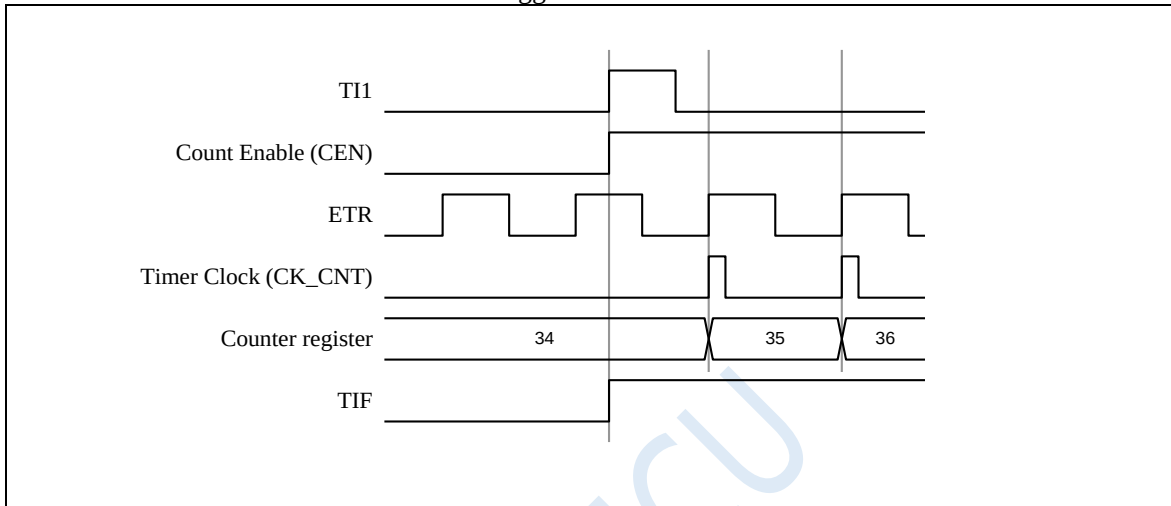
External clock mode 2 can be used with another input signal trigger mode. For example, the ETR signal is used as the input of an external clock, and another input signal can be used as the trigger input (supports standard trigger mode, reset trigger mode and gated trigger mode). Note that ETR cannot be configured as TRGI through the TS bit of the PWMA_SMCR register.

In the following example, once a rising edge occurs on TI1, the counter counts up once on each rising edge of ETR:

1. Configure the external trigger input circuit through the PWMA_ETR register. Configure ETPS=00 to disable prescaler, configure ETP=0 to monitor the rising edge of the ETR signal, configure ECE=1 to enable external clock mode 2.
2. Configure CC1P=0 in the PWMA_CCER1 register to select the rising edge trigger of TI1.
3. Configure SMS=110 in the PWMA_SMCR register to select the timer as trigger mode. Configure TS=101 in PWMA_SMCR register to select TI1 as the input source.

When a rising edge occurs on TI1, the TIF flag is set and the counter starts counting on the rising edge of ETR. The delay between the rising edge of the TI1 signal and the actual counter clock depends on the resynchronization circuit at the TI1 input. The delay between the rising edge of the ETR signal and the actual clock of the counter depends on the resynchronization circuit at the ETRP input.

External clock mode 2 + control circuit in trigger mode



20.4.6 Synchronize with PWMB

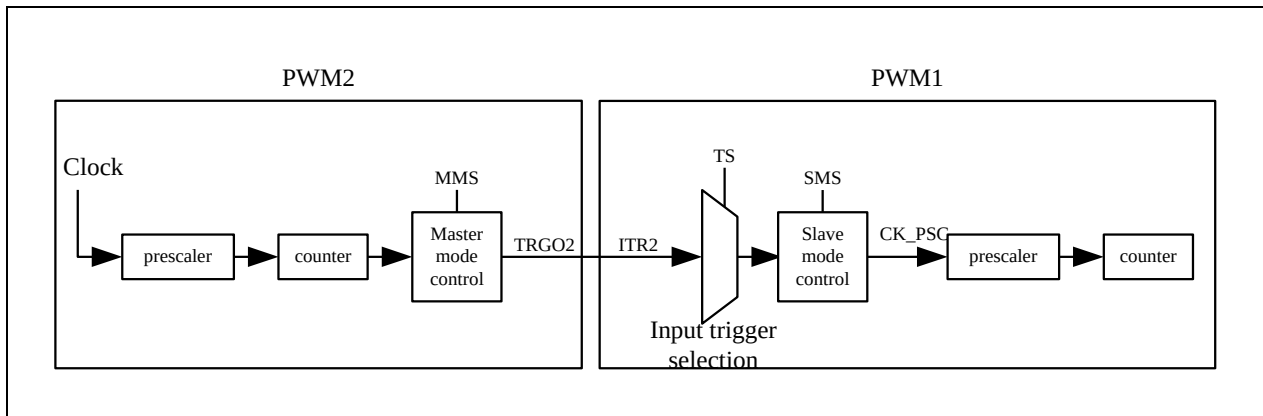
In the chip, the timers are interconnected internally for synchronization or linking of the timers. When a timer is configured as master mode, it can output trigger signal (TRGO) to those timers configured as slave mode to complete reset operation, start operation, stop operation or as the driving clock of those timers.

Use the TRGO of PWMB as the prescaler clock of PWMA

For example, users can configure PWMB as the prescaler clock of PWMA, and the following configuration is required:

1. Configure PWMB as the main mode, so that a periodic trigger signal is output at each update event (UEV). Configure MMS=010 in the PWMB_CR2 register so that TRGO can output a rising edge during each update event.
2. The TRGO signal output by PWMB is linked to PWMA. PWMA needs to be configured to trigger slave mode, using ITR2 as the input trigger signal. The above operations can be achieved by configuring TS=010 in the PWMA_SMCR register.
3. Configure SMS=111 in the PWMA_SMCR register to set the clock/trigger controller to external clock mode 1. This operation will cause the rising edge of the periodic trigger signal TRGO output by PWMB to drive the PWMA clock.
4. Finally, set the CEN bit of PWMB (in the PWMB_CR1 register) to enable two PWMs.

Example of master trigger slave mode



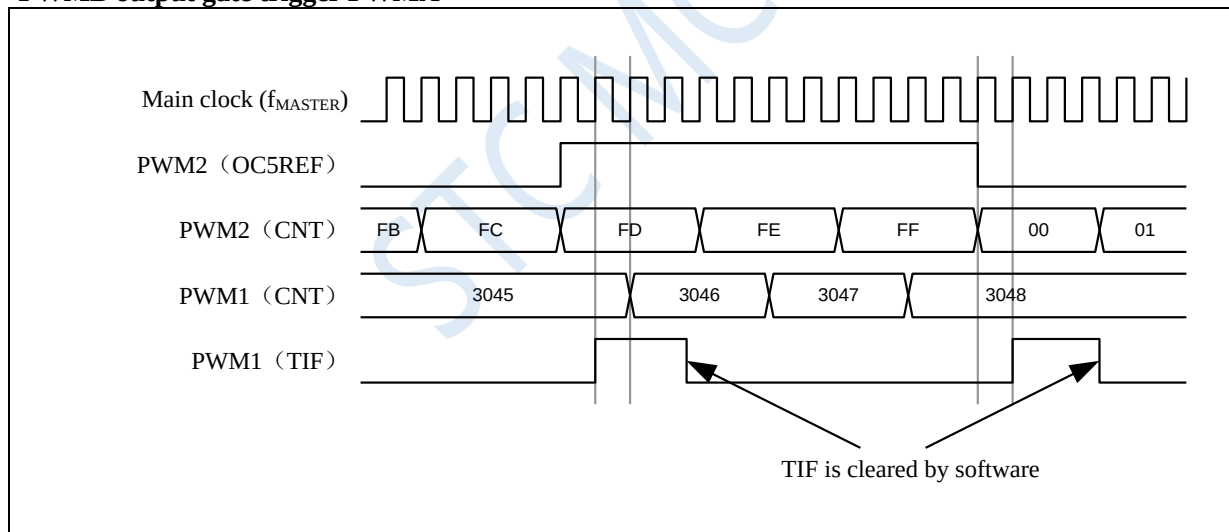
Use PWMB to enable PWMA

In this example, we use the compare output of PWMB to enable PWMA. PWMA counts according to its own driving clock only when the OC1REF signal of PWMB is high. Both PWMs use f_{MASTER} divided by 4 as the clock ($f_{CK\_CNT} = f_{MASTER}/4$).

1. Configure PWMB as the main mode, and output the comparison output signal (OC5REF) as the trigger signal. (Configure MMS=100 in PWMB_CR2 register).
2. Configure the waveform of the OC5REF signal of PWMB (PWMB_CCMR1 register).
3. Configure PWMA to use the output of PWMB as its own trigger input signal (configure TS=010 in PWMA_SMCR register).
4. Configure PWMA as the gated trigger mode (configure SMS=101 in the PWMA_SMCR register).
5. Set the CEN bit (PWMA_CR1 register) to enable PWMA.
6. Set the CEN bit (PWMB_CR1 register) to enable PWMB.

Note: The clocks of the two PWMs are not synchronized, but only affect the enable signal of PWMA.

PWMB output gate trigger PWMA

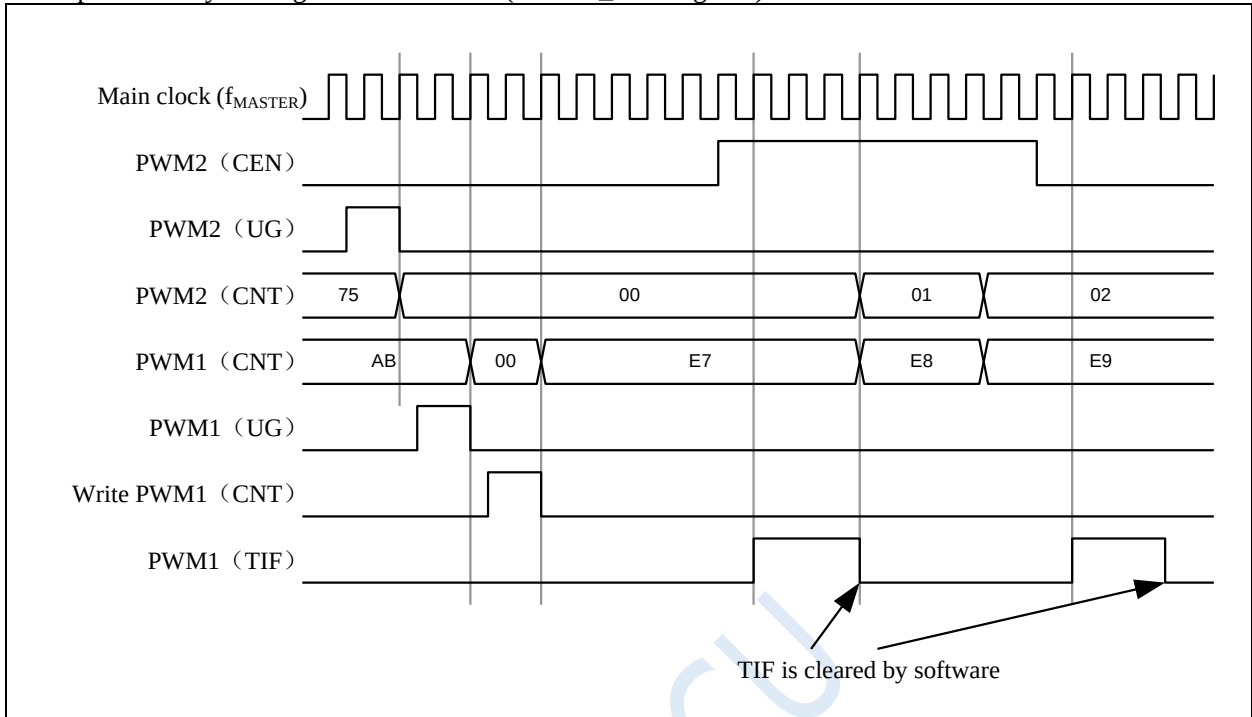


In the figure above, the PWMA counter and prescaler are not initialized before starting, so they start counting from the existing value. If the two timers are reset before starting PWMB, the user can write the desired value to the PWMA counter to start counting from the specified value. The reset operation of PWMA can be realized by software writing the UG bit of PWMA_EGR register.

In the following example, we synchronize PWMB and PWMA. PWMB is the main mode and starts counting from 0. PWMA is the trigger slave mode and starts counting from 0xE7. The two PWMs use the same frequency division factor. When the CEN bit in the PWMB_CR1 register is cleared, PWMB is disabled and PWMA stops counting.

1. Configure PWMB as the main mode, and output the comparison output signal (OC5REF) as the trigger signal. (Configure MMS=100 in PWMB_CR2 register).
2. Configure the waveform of the OC5REF signal of PWMB (PWMB_CCMR1 register).
3. Configure PWMA to use the output of PWMB as its own trigger input signal (configure TS=010 in PWMA_SMCR register).
4. Configure PWMA as the gated trigger mode (configure SMS=101 in the PWMA_SMCR register).
5. Reset PWMB by writing 1 to the UG bit (PWMB_EGR register).
6. Reset PWMA by writing 1 to the UG bit (PWMA_EGR register).

7. Write 0xE7 into the counter of PWMA (PWMA_CNTRL) to initialize PWMA.
8. Enable PWMA by writing 1 to the CEN bit (PWMA_CR1 register).
9. Start PWMB by writing 1 to the CEN bit (PWMB_CR1 register).
10. Stop PWMB by writing 0 to the CEN bit (PWMB_CR1 register).



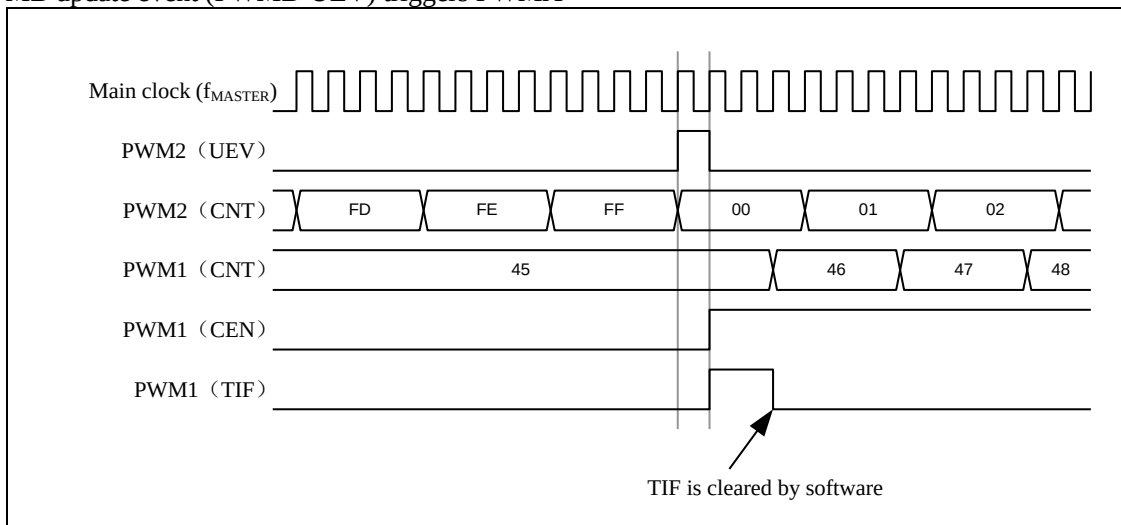
Use PWMB to start PWMA

In this example, we use the update event of PWMB to start PWMA.

When the PWMB update event occurs, PWMA starts counting from its existing value according to PWMA's own drive clock (it can be a non-zero value). After PWMA receives the trigger signal, it automatically enables the CEN bit and starts counting until the user writes 0 to the CEN bit of the PWMA_CR1 register. Both PWMs use f_{MASTER} divided by 4 as the driving clock ($f_{CK\_CNT} = f_{MASTER}/4$).

1. Configure PWMB as the main mode and output the update signal (UEV). (Configure MMS=010 in PWMB_CR2 register).
2. Configure the period of PWMB (PWMB_ARR register).
3. Configure PWMA to use the output of PWMB as the input trigger signal (configure TS=010 in PWMA_SMCR register).
4. Configure PWMA as trigger mode (configure SMS=110 in PWMA_SMCR register).
5. Set the CEN bit (PWMB_CR1 register) to start PWMB.

PWMB update event (PWMB-UEV) triggers PWMA



As in the previous example, the user can also initialize them before starting the counter.

Trigger two PWMs synchronously with external signals

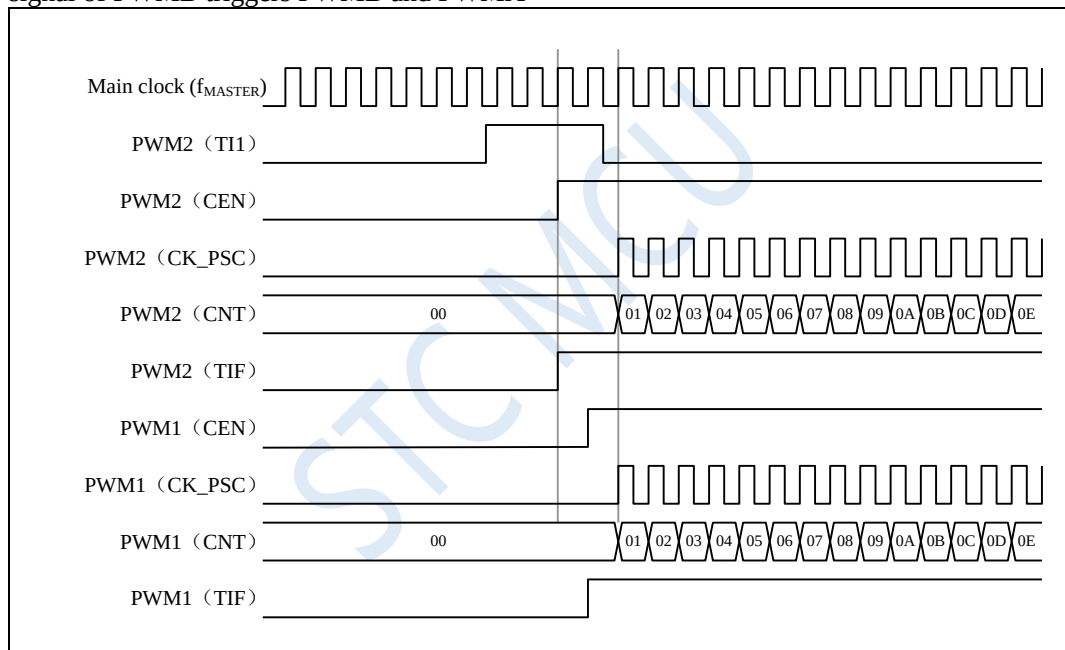
In this example, the rising edge of TI1 is used to enable PWMB and enable PWMA at the same time. In order to maintain timer alignment, PWMB needs to be configured in master/slave mode (slave mode for TI1 signal and master mode for PWMA).

1. Configure PWMB as the main mode, and use the output enable signal as the trigger of PWMA (configure MMS=001 in the PWMB_CR2 register).
2. Configure PWMB as slave mode, and use TI1 signal as the input trigger signal (configure TS=100 in PWMB_SMCR register).
3. Configure the trigger mode of PWMB (configure SMS=110 in PWMB_SMCR register).
4. Configure PWMB as master/slave mode (configure MSM=1 in PWMB_SMCR register).
5. Configure PWMA to use the output of PWMB as the input trigger signal (configure TS=010 in the PWMA_SMCR register).
6. Configure the trigger mode of PWMA (configure SMS=110 in the PWMA_SMCR register).

When a rising edge occurs on TI1, the two timers start counting synchronously, and the TIF bit is set.

Note: In this example, both timers are initialized (setting the UG bit) before starting, so they both start counting from 0, but the user can also insert an offset by modifying the counter register (PWMA_CNT), In this case, a delay will be inserted between the CK_PSC signal and the CNT_EN signal of PWMB.

The TI1 signal of PWMB triggers PWMB and PWMA

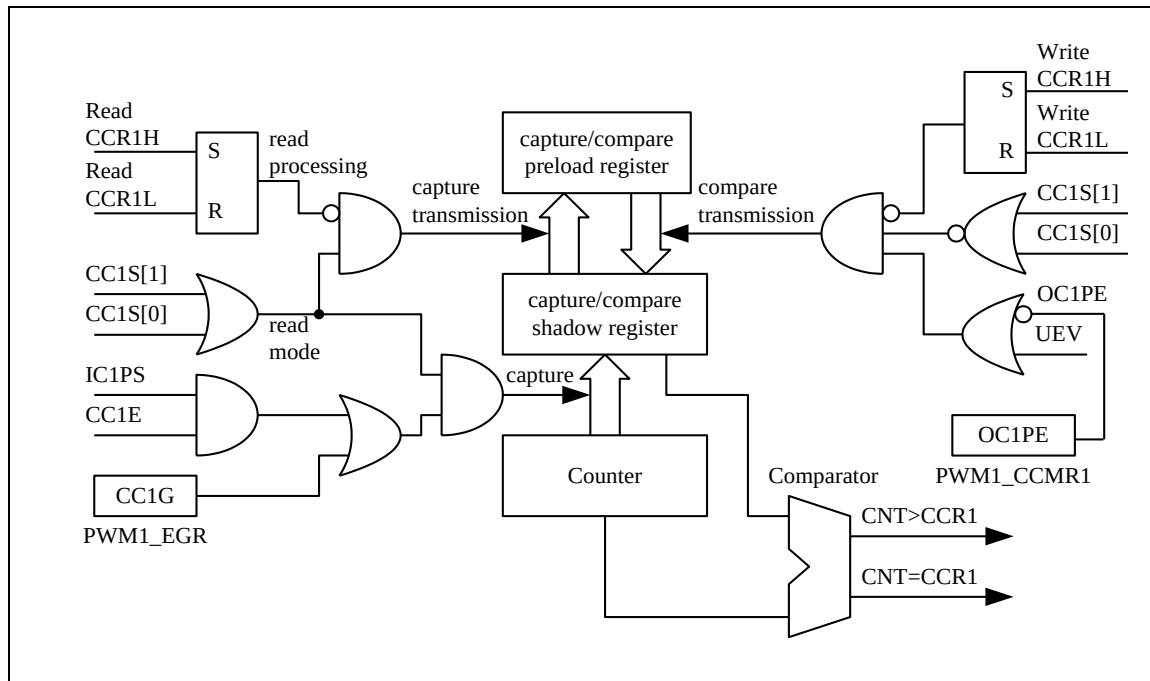


20.5 Capture/Compare Channel

PWM1P, PWM2P, PWM3P, PWM4P can be used as input capture, PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N can output comparison. This function can be realized by configuring the CCiS channel selection bit of the capture/compare channel mode register (PWMA_CCMRi), where i represents the number of channels from 1 to 4.

Each capture/compare channel is built around a capture/compare register (including shadow registers), including the input part of the capture (digital filtering, multiplexing and prescaler) and the output part (comparator and Output control).

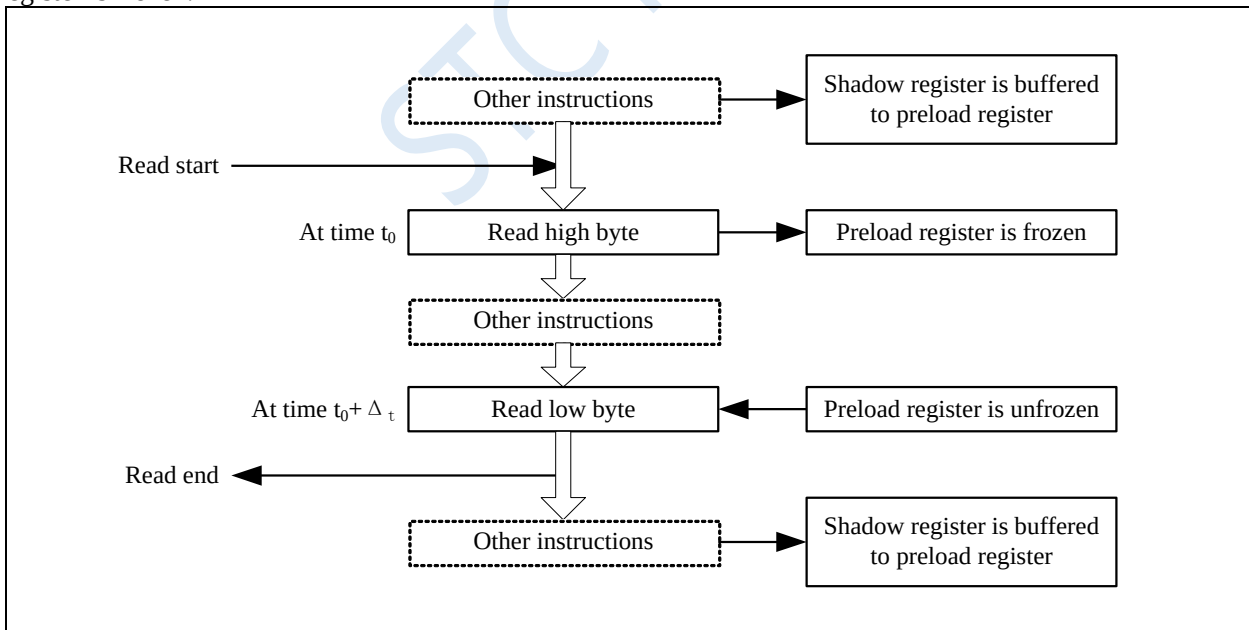
The main circuit of capture/compare channel 1 (other channels are similar to this)



The capture/compare module consists of a preload register and a shadow register. The read and write process only operates the preload register. In capture mode, the capture occurs on the shadow register and then copied to the preload register. In the compare mode, the contents of the preload register are copied to the shadow register, and then the contents of the shadow register are compared with the counter.

When the channel is configured in output mode, the PWM1_CCRi register can be accessed at any time.

When the channel is configured as input mode, the read operation of the PWM1_CCRi register is similar to the read operation of the counter. When the capture occurs, the contents of the counter are captured to the PWM1_CCRi shadow register, and then copied to the preload register. During the read operation, the preload register is frozen.



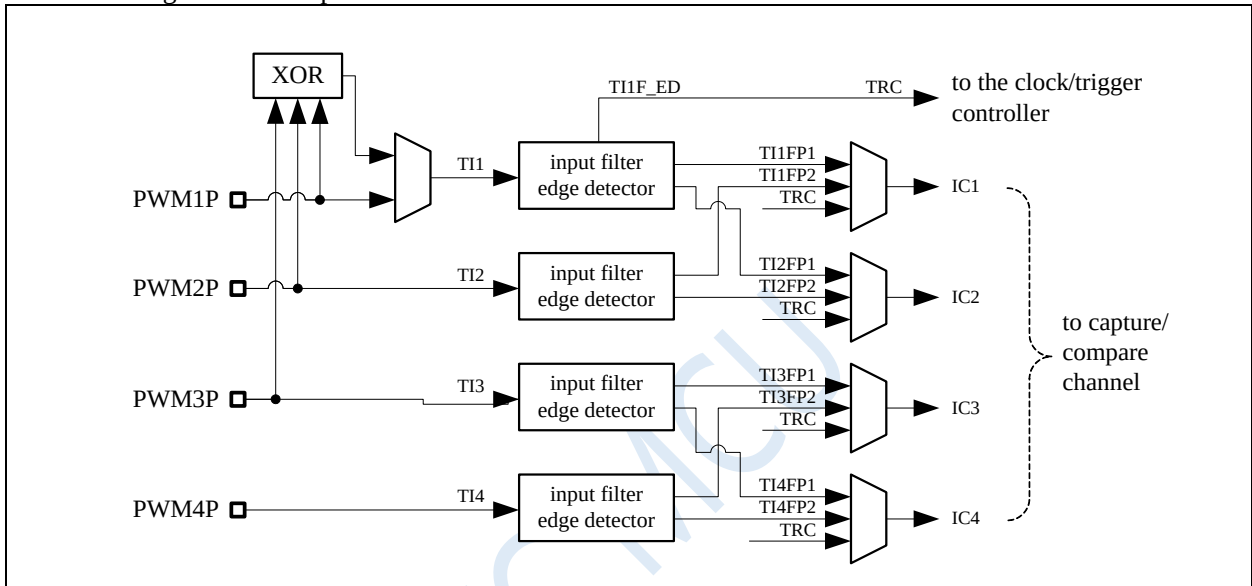
The figure above describes the read operation flow of the 16-bit CCRi register. The data buffered will remain unchanged until the end of the read flow. After the entire reading process is over, if only the PWM1_CCRiL register is read, the low bit of the counter value is returned. If the upper data is read after reading the lower data, the same lower data will not be returned.

20.5.1 Writing process of 16-bit PWMA_CCRi register

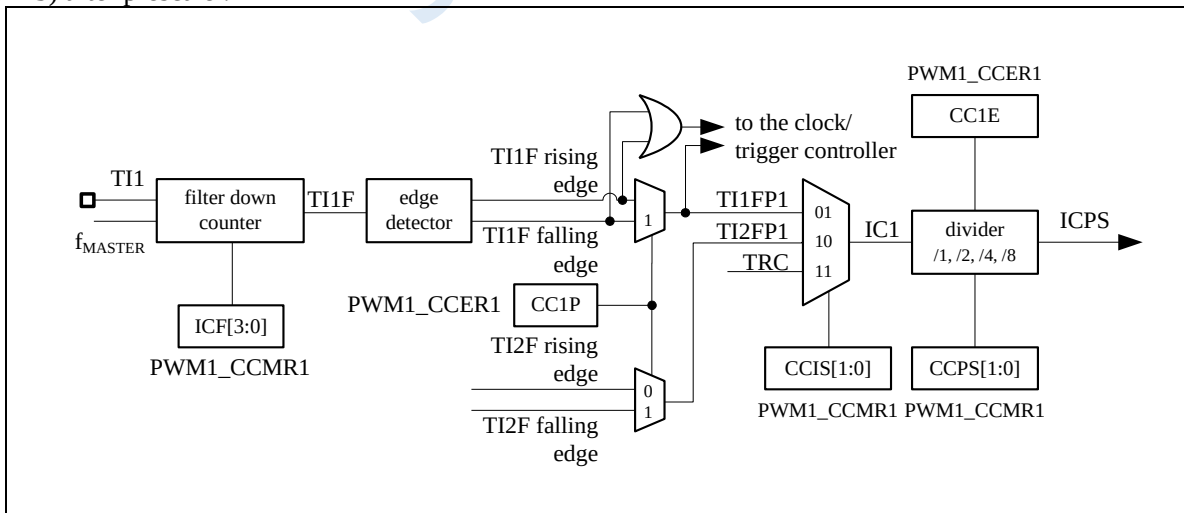
The writing operation of the 16-bit PWMA_CCRi register is completed through the preload register. Two instructions must be used to complete the entire process, one instruction corresponds to one byte. The upper byte must be written first. When writing the upper byte, the shadow register update is prohibited until the write operation of the lower byte is completed.

20.5.2 Input Module

Block diagram of the input module



As shown in the figure, the input part samples the corresponding TIx input signal and generates a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx), which can be used as an input trigger for the trigger mode controller or as a capture control. This signal enters the capture register (ICxPS) after prescaler.



20.5.3 Input Capture Mode

In the input capture mode, when the corresponding edge on the ICi signal is detected, the current value of

the counter is latched into the capture/compare register (PWMA_CCRx). When a capture event occurs, the corresponding CCiIF flag (PWMA_SR register) is set. If the CCiIE bit of the PWMA_IER register is set, that is, the interrupt is enabled, an interrupt request will be generated. If the CCiIF flag is already high when the capture event occurs, the repeated capture flag CCiOF (PWMA_SR2 register) is set. Writing CCiIF=0 or reading the captured data stored in the PWMA_CCRiL register can clear CCiIF. Write CCiOF=0 to clear CCiOF.

Capture on rising edge of PWM input signal

The following example shows how to capture the counter value to the PWMA_CCR1 register at the rising edge of TI1 input. The steps are as follows:

1. Select a valid input terminal and set CC1S=01 in the PWMA_CCMR1 register. At this time, the channel is configured as an input, and the PWMA_CCR1 register becomes read-only.
2. According to the characteristics of the input signal TIi, the filter time of the corresponding input filter can be set by configuring the IC1F bit in the PWMA_CCMR1 register. Assuming that the input signal jitters within a maximum of 5 clock cycles, we need to configure the filter bandwidth to be longer than 5 clock cycles; therefore, we can sample 8 times continuously to confirm the last real edge transition at TI1, that is, in the PWMA_CCMR1 register Write IC1F=0011 in the middle. At this time, only if 8 consecutively sampled TI1 signals valid (sampling frequency is f_{MASTER}).
3. Select the effective conversion edge of the TI1 channel and write CC1P=0 (rising edge) in the PWMA_CCER1 register.
4. Configure the input prescaler. In this example, we want the capture to occur at every valid level transition moment, so the prescaler is disabled (write IC1PS=00 in the PWMA_CCMR1 register).
5. Set CC1E of the PWMA_CCER1 register to 1, allowing the value of the counter to be captured into the capture register.
6. If necessary, enable related interrupt requests by setting the CC1IE bit in the PWMA_IER register.

When an input capture occurs:

- When a valid level conversion occurs, the value of the counter is transferred to the PWMA_CCR1 register.
- The CC1IF flag is set. When at least 2 consecutive captures have occurred and CC1IF has not been cleared, CC1OF is also set.
- If the CC1IE bit is set, an interrupt will be generated.

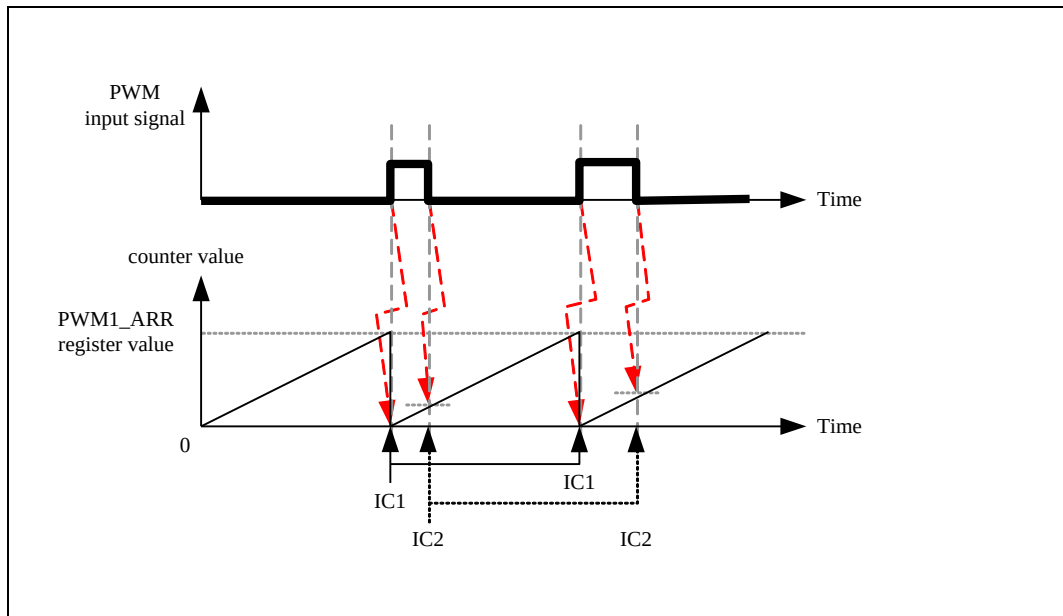
In order to handle the capture overflow event (CC1OF bit), it is recommended to read the data before reading the repeated capture flag. This is to avoid losing the repeated capture information that may be generated after the capture overflow flag is read and before the data is read.

Note: Setting the corresponding CCiG bit in the PWMA_EGR register can generate an input capture interrupt by software.

PWM input signal measurement

This mode is a special case of the input capture mode, except for the following differences, the operation is the same as the input capture mode:

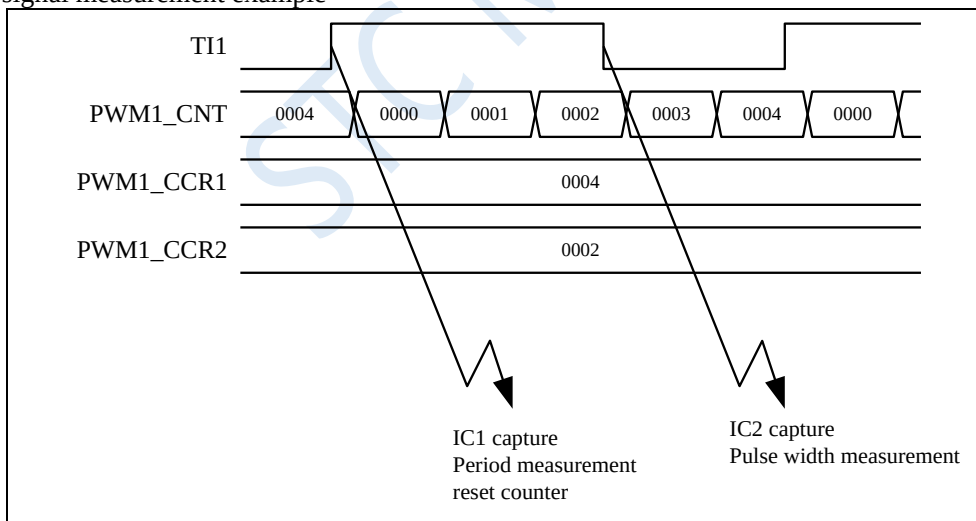
- Two ICi signals are mapped to the same TIi input.
- The valid edges of the two ICI signals have opposite polarities.
- One of the TIiFP signals is used as the trigger input signal, and the trigger mode controller is configured to reset the trigger mode.



For example, you can measure the period (PWMA_CCR1 register) and duty cycle (PWMA_CCR2 register) of the PWM signal input on TI1 in the following way.

1. Select the effective input of PWMA_CCR1: set CC1S=01 in the PWMA_CCMR1 register (select TI1FP1).
2. Select the valid polarity of TI1FP1: set CC1P=0 (valid at rising edge).
3. Select the valid input of PWMA_CCR2: set CC2S=10 in the PWMA_CCMR2 register (select TI1FP2).
4. Select the valid polarity of TI1FP2 (capture data to PWMA_CCR2): set CC2P=1 (falling edge valid).
5. Select a valid trigger input signal: set TS=101 in the PWMA_SMCR register (select TI1FP1).
6. Configure the trigger mode controller to reset trigger mode: set SMS=100 in PWMA_SMCR.
7. Enable capture: set CC1E=1 and CC2E=1 in the PWMA_CCER1 register.

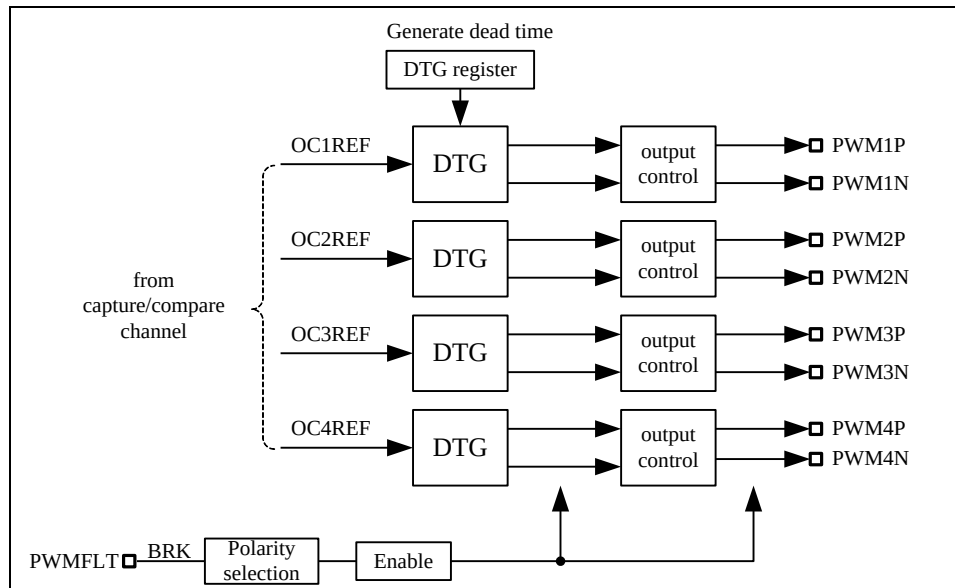
PWM input signal measurement example



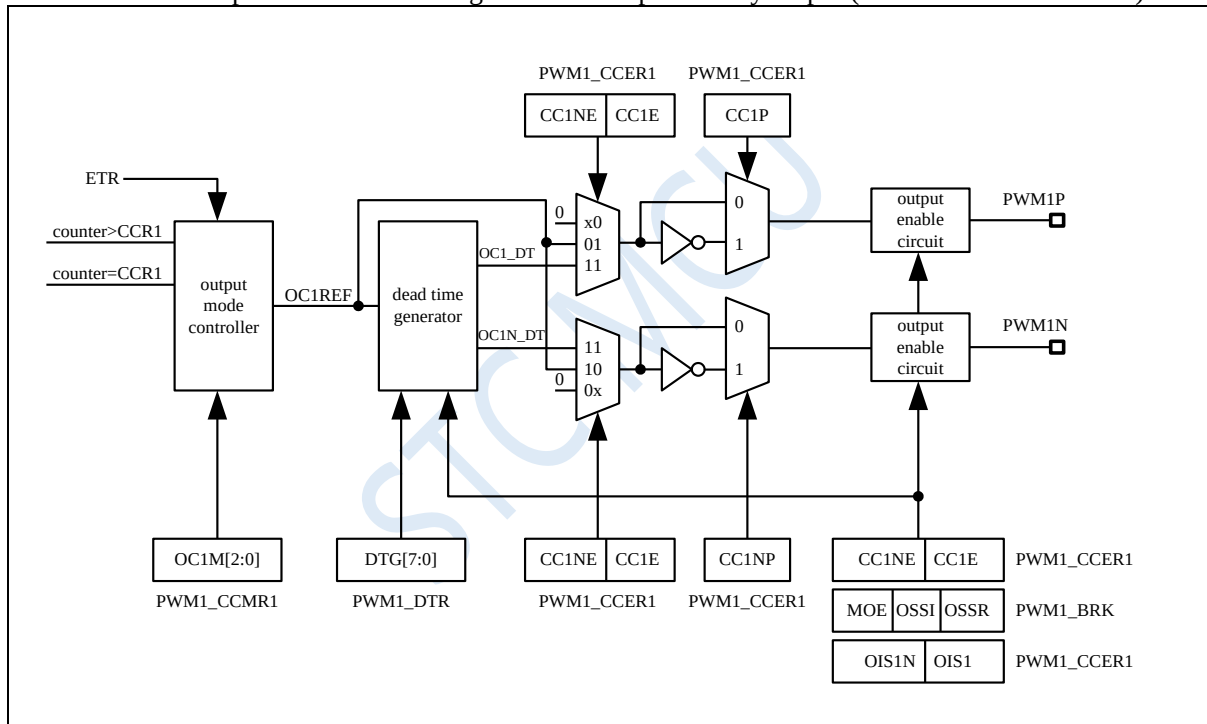
20.5.4 Output Module

The output module will generate an intermediate waveform used as a reference, called OCIREF (high effective). The processing of the brake function and polarity is at the end of the module.

Output module block diagram



Channel 1 detailed output module block diagram with complementary output (similar to other channels)



20.5.5 Forced output mode

In the output mode, the output comparison signal can be directly forced to a high or low state by software, instead of relying on the comparison result between the output comparison register and the counter.

Set OCIM=101 in the PWMA_CCMRi register to force the OCiREF signal to be low.

Set OCIM=100 in PWMA_CCMRi register to force the OCiREF signal to be low.

Whether the output of OCi/OCiN is high or low depends on the CCiP/CCiNP polarity flag.

In this mode, the comparison between the PWMA_CCRi shadow register and the counter is still in progress, the corresponding flag will be modified, and the corresponding interrupt will still be generated.

20.5.6 Output Compare Mode

This mode is used to control an output waveform or indicate that a given period of time has been reached.

When the counter matches the content of the capture/compare register, the following operations are performed:

- According to different output comparison modes, the corresponding OCI output signal:
 - Keep unchanged (OCiM=000)
 - Set to effective level (OCiM=001)
 - Set to invalid level (OCiM=010)
 - Flip (OCiM=011)
- Set the flag bit in the interrupt status register (CCiIF bit in the PWMA_SR1 register).
- If the corresponding interrupt enable bit (CCiIE bit in the PWMA_IER register) is set, an interrupt is generated.

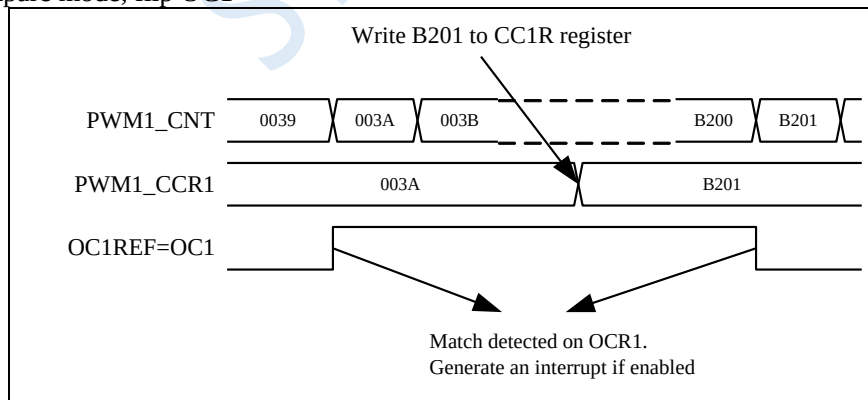
The OPiM bit of the PWMA_CCMRi register is used to select the output compare mode, and the CCiP bit of the PWMA_CCMRi register is used to select the valid and invalid level polarity. The OPiPE bit of the PWMA_CCMRi register is used to select whether the PWMA_CCRi register needs to use the preload register. In the output compare mode, the update event UEV has no effect on the OCiREF and OCI outputs. The time accuracy is one counting period of the counter. The output compare mode can also be used to output a single pulse.

Configuration steps of output comparison mode:

1. Select the counter clock (internal, external or prescaler).
2. Write the corresponding data into the PWMA_ARR and PWMA_CCRi registers.
3. To generate an interrupt request, set the CCiIE bit.
4. Steps to select output mode:
 1. Set OCiM=011, when the counter matches CCRi, flip the output of OCiM pin
 2. Set OPiPE = 0, disable preload register
 3. Set CCiP = 0, select high level as active level
 4. Set CCiE = 1, enable output
 5. Set the CEN bit of the PWMA_CR1 register to start the counter

The PWMA_CCRi register can be updated by software at any time to control the output waveform, provided that the preload register is not used (OCiPE=0), otherwise the shadow register of PWMA_CCRi can only be updated when the next update event occurs.

Output compare mode, flip OC1



20.5.7 PWM Mode

Pulse width modulation (PWM) mode can generate a signal whose frequency is determined by the PWMA_ARR register and the duty cycle is determined by the PWMA_CCRi register.

Write 110 (PWM mode 1) or 111 (PWM mode 2) in the OCiM bit in the PWMA_CCMRi register to independently set each OCI output channel to generate a PWM. The OPiPE bit of the PWMA_CCMRi register must be set to enable the corresponding preload register, and the ARPE bit of the PWMA_CR1 register can also be set to enable the preload register for auto-reload (in up-counting mode or central symmetric mode).

Since the preload register can only be transferred to the shadow register when an update event occurs, all registers must be initialized by setting the UG bit of the PWMA_EGR register before the counter starts counting.

The polarity of OC_i can be set by software in the CCiP bit in the PWMA_CCER_i register, and it can be set as active high or active low. The output enable of OC_i is controlled by the combination of CCiE, MOE, OIS_i, OSSR and OSS_i bits in the PWMA_CCER_i and PWMA_BKR registers.

In PWM mode (mode 1 or mode 2), PWMA_CNT and PWMA_CCR_i are always being compared (according to the counting direction of the counter) to determine whether it meets $PWMA\_CNT \leq PWMA\_CCR_i$ or $PWMA\_CNT \geq PWMA\_CCR_i$.

According to the state of the CMS bit field in the PWMA_CR1 register, the timer can generate edge-aligned PWM signals or center-aligned PWM signals.

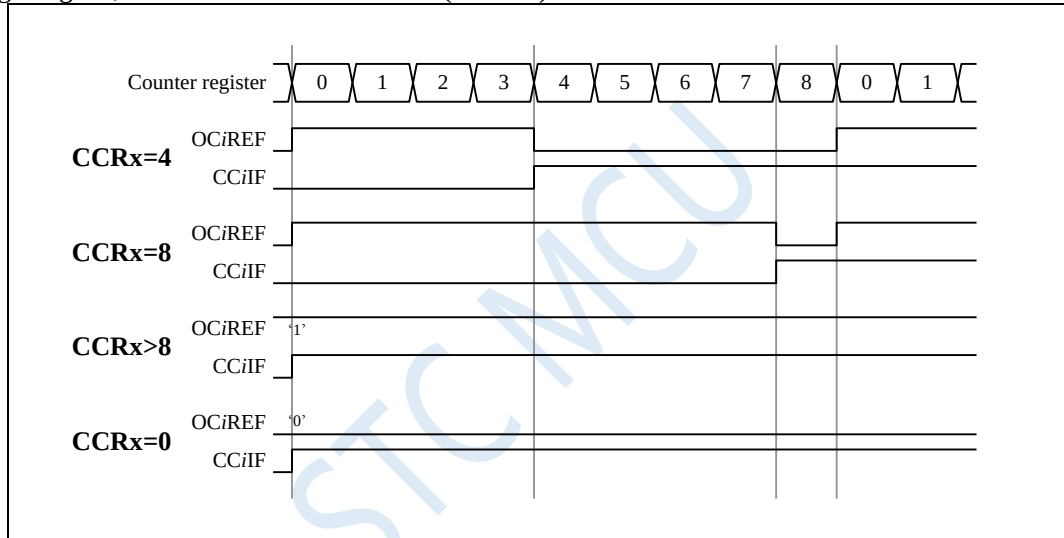
PWM edge alignment mode

Up counting configuration

When the DIR bit in the PWMA_CR1 register is 0, count up is executed.

The following is an example of PWM mode 1. When $PWMA\_CNT < PWMA\_CCR_i$, the PWM reference signal OCiREF is high, otherwise it is low. If the comparison value in PWMA_CCR_i is greater than the auto-reload value (PWMA_ARR), then OCiREF remains high. If the comparison value is 0, then OCiREF remains low.

Edge aligned, waveform of PWM mode 1 (ARR=8)



Countdown configuration

When the DIR bit of the PWMA_CR1 register is 1, the down count is executed.

In PWM mode 1, when $PWMA\_CNT > PWMA\_CCR_i$, the reference signal OCiREF is low, otherwise it is high. If the comparison value in PWMA_CCR_i is greater than the auto-reload value in PWMA_ARR, then OCiREF remains high. In this mode, a PWM waveform with a duty cycle of 0% cannot be generated.

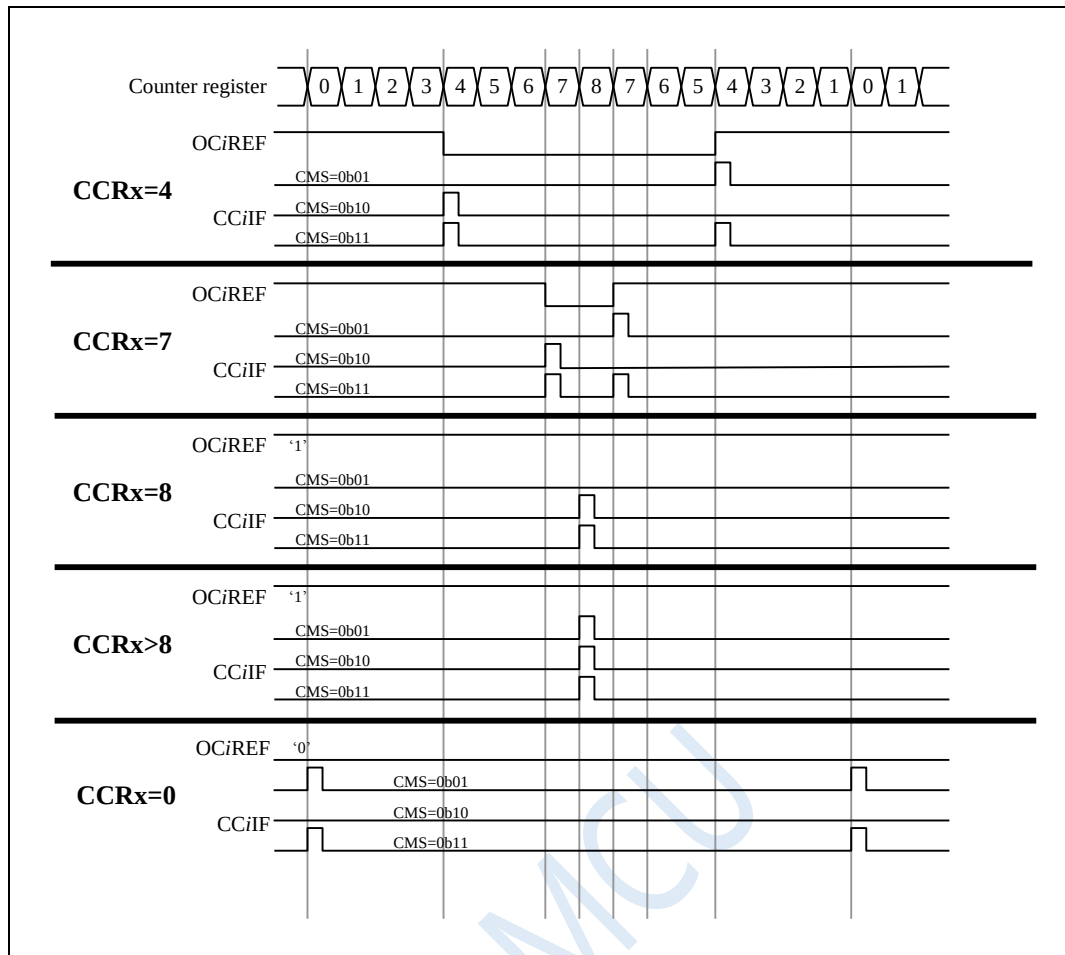
PWM center aligned mode

When the CMS bit in the PWMA_CR1 register is not '00', it is the center-aligned mode (all other configurations have the same effect on the OCiREF/OC_i signal).

According to different CMS bit settings, the compare flag can be set when the counter counts up, counts down, or counts up and down. The counting direction bit (DIR) in the PWMA_CR1 register is updated by hardware. Do not modify it by software.

Some examples of center-aligned PWM waveforms are given below:

- PWMA_ARR=8
- PWM mode 1
- The flag bit is set in the following three situations:
 - Only when the counter counts down (CMS=01)
 - Only when the counter is counting up (CMS=10)
 - When the counter is counting up and down (CMS=11)
- Center-aligned PWM waveform (ARR=8)



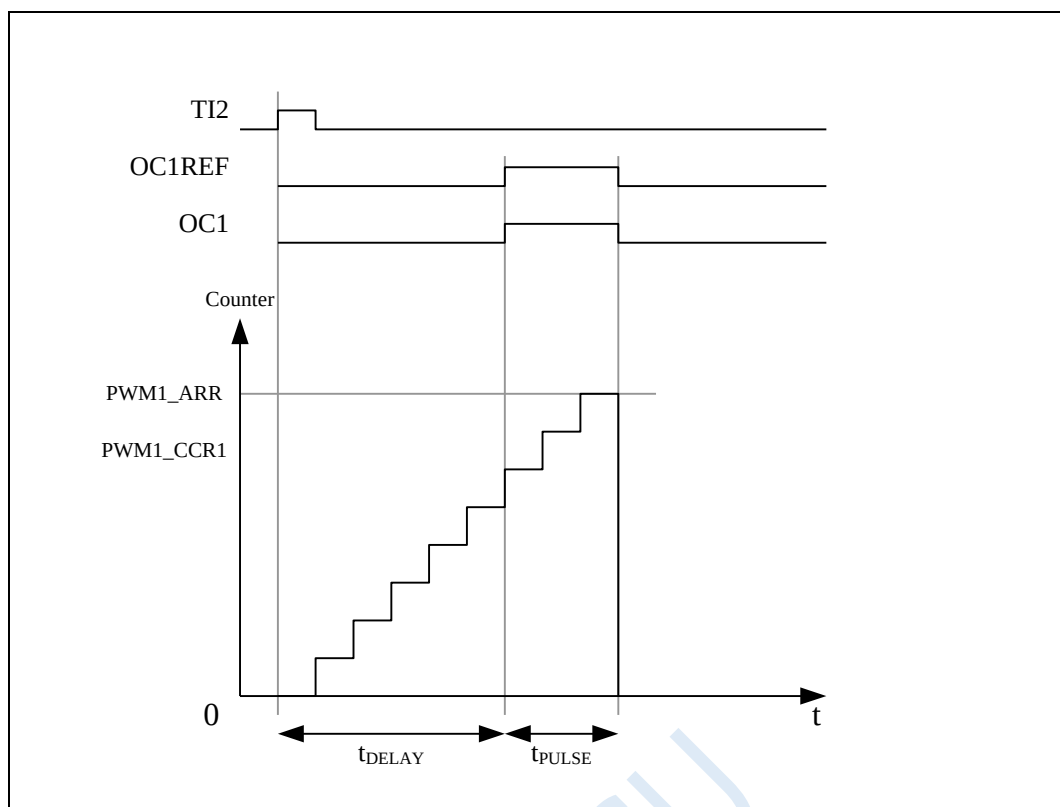
Single pulse mode

Single pulse mode (OPM) is a special case of many of the aforementioned modes. This mode allows the counter to respond to a stimulus and generate a pulse with a controllable pulse width after a programmable delay.

The counter can be started by the clock/trigger controller, and the waveform can be generated in output comparison mode or PWM mode. Setting the OPM bit of the PWMA_CR1 register will select the single pulse mode, and the counter will automatically stop at the next update event UEV. Only when the comparison value is different from the initial value of the counter can a pulse be generated. Before starting (when the timer is waiting to be triggered), the following must be configured:

- Up counting mode: counter $CNT < CCRi \leq ARR$,
- Counting down mode: counter $CNT > CCRi$.

Single pulse mode legend



For example, delay t_{DELAY} after detecting a rising edge on the TI2 input pin, and generate a positive pulse of t_{PULSE} width on OC1: (assuming IC2 is used as the trigger source for triggering channel 1)

- Set CC2S=01 in PWMA_CCMR2 register to map IC2 to TI2.
- Set CC2P=0 in PWMA_CCER1 register to enable IC2 to detect the rising edge.
- Set TS=110 in the PWMA_SMCR register to make IC2 the trigger source (TRGI) of the clock/trigger controller.
- Set SMS=110 (trigger mode) in PWMA_SMCR register, IC2 is used to start the counter. The OPM waveform is determined by the value written in the compare register (the clock frequency and counter prescaler should be considered).
- t_{DELAY} is defined by the value in the PWM1_CCR1 register.
- t_{PULSE} is defined by the difference between the autoloading value and the comparison value (PWM1_ARR – PWM1_CCR1).
- Suppose that a waveform from 0 to 1 is to be generated when a comparison match occurs, and a waveform from 1 to 0 is generated when the counter reaches the preload value. First, set the OCIM of the PWMA_CCMR1 register to 111 and enter PWM mode 2. According to You need to selectively set OC1PE=1 in the PWMA_CCMR1 register, set the ARPE in the PWMA_CR1 register, enable the preload register, then fill in the comparison value in the PWM1_CCR1 register, fill in the autoloading value in the PWM1_ARR register, and set the UG bit to generate An update event, and then wait for an external trigger event on TI2.

In this example, the DIR and CMS bits in the PWMA_CR1 register should be set low.

Because only one pulse is needed, set OPM=1 in the PWMA_CR1 register to stop counting at the next update event (when the counter rolls over from the auto-load value to 0).

OCx fast enable (special case)

In the single pulse mode, the edge detection of the TIi input pin will set the CEN bit to start the counter, and then the comparison operation between the counter and the comparison value produces a single pulse output. But these operations require certain clock cycles, so it limits the minimum delay t_{DELAY} that can be obtained.

If you want to output the waveform with the minimum delay, you can set the OCiFE bit in the PWMA_CCMRi register. At this time, OCiREF (and OCx) is forced to respond directly to the excitation without relying on the comparison result. The output waveform is the same as the waveform when the comparison matches. OCiFE only works when the channel is configured in PWMA and PWMB mode.

Complementary output and dead zone insertion

PWMA can output two complementary signals, and can manage the instantaneous turn-off and turn-on of the output. This period is usually called the dead zone. The user should base on the connected output devices and their characteristics (delay of level conversion, power supply). Switch delay, etc.) to adjust the dead time.

Configure the CCiP and CCiNP bits in the PWMA_CCERi register to independently select the polarity (main output OCi or complementary output OCiN) for each output. The complementary signals OCi and OCiN are controlled by the following control bit combinations: CCiE and CCiNE bits in PWMA_CCERi register, MOE, OISi, OISiN, OSSi and OSSR bits in PWMA_BKR register. In particular, the dead zone control is activated when transitioning to the IDLE state (MOE drops to 0).

Setting the CCiE and CCiNE bits at the same time will insert the dead zone. If there is a brake circuit, the MOE bit should also be set. Each channel has an 8-bit dead zone generator.

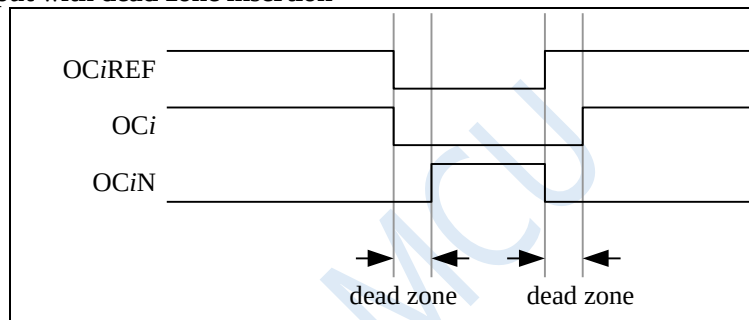
If OSi and OSiN are high effective:

- The OCi output signal is the same as OCiREF, except that its rising edge has a delay relative to the rising edge of OCiREF.
- The output signal of OCiN is opposite to OCiREF, except that its rising edge has a delay relative to the falling edge of OCiREF.

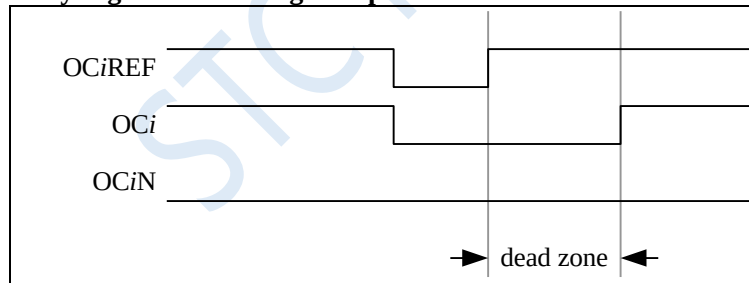
If the delay is greater than the currently effective output width (OCi or OCiN), the corresponding pulse will not be generated.

The following figures show the relationship between the output signal of the dead zone generator and the current reference signal OSiREF. (Assuming CCiP=0, CCiNP=0, MOE=1, CCiE=1 and CCiNE=1)

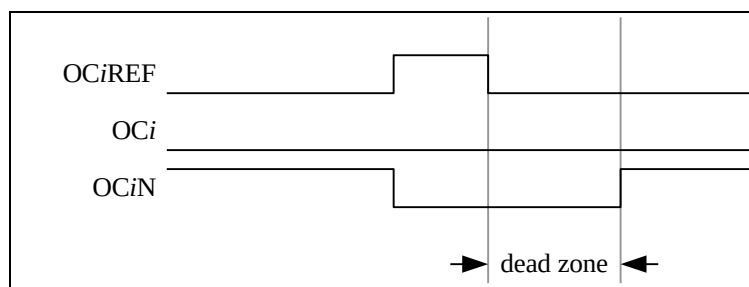
Complementary output with dead zone insertion



Dead zone waveform delay is greater than negative pulse



Dead zone waveform delay is greater than positive pulse



The dead time delay of each channel is the same, which is programmed and configured by the DTG bit in the PWMA_DTR register.

Redirect OCiREF to OCi or OCiN

In the output mode (forced output, output compare or PWM output), by configuring the CCiE and CCiNE bits of the PWMA_CCERi register, OCiREF can be redirected to the output of OCi or OCiN.

This function can send a special waveform (such as PWM or static active level) on an output when the complementary output is at an invalid level. Another function is to make the two outputs be at the inactive level at the same time, or at the active level at the same time (this is still the complementary output with dead zone).

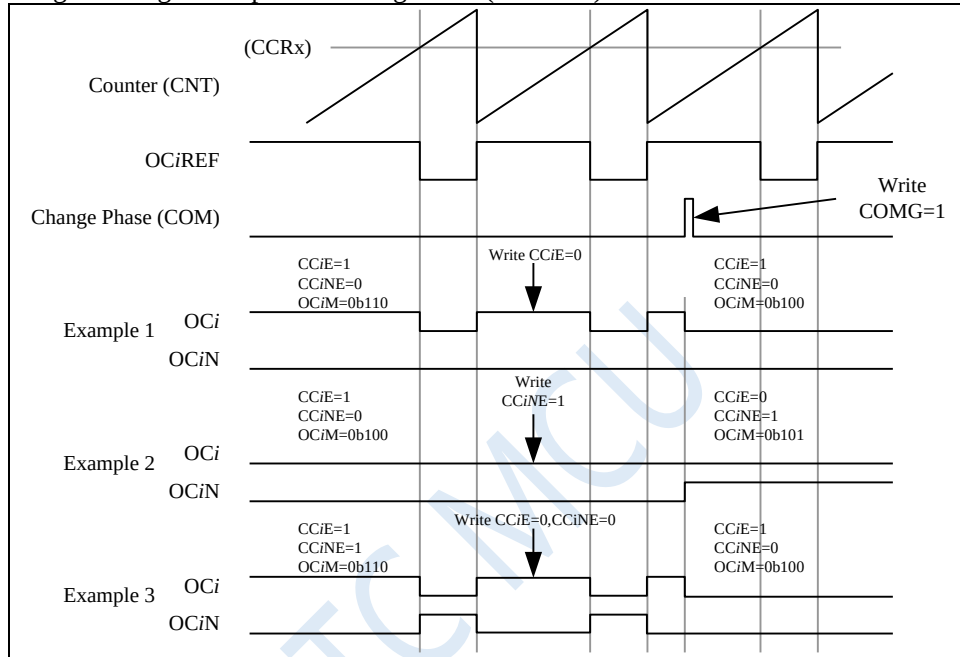
Note: When only OCiN is enabled (CCiE=0, CCiNE=1), it will not be inverted, and it will be effective immediately when OCiREF goes high. For example, if CCiNP=0, then OSiN=OCiREF. On the other hand, when both OCI and OCIN are enabled (CCiE=CCiNE=1), OCI is valid when OCiREF is high; on the contrary, when OCiREF is low, OCIN becomes valid.

Six-step PWM output for motor control

When complementary output is required on a channel, the preload bits are OPiM, CCiE and CCiNE. When a COM commutation event occurs, these preload bits are transferred to the shadow register bits. In this way, you can set the next step configuration in advance, and modify the configuration of all channels at the same time. COM can be generated by software by setting the COMG bit in the PWMA_EGR register, or by hardware on the rising edge of TRGI.

The following figure shows the output of OCx and OCxN in three different configurations when a COM event occurs.

Example of generating six-step PWM using COM (OSSR=1)



20.5.8 Using the brake function (PWMFLT)

The brake function is often used in motor control. When using the brake function, according to the corresponding control bits (MOE, OSSI and OSSR bits in the PWMA_BKR register), the output enable signal and invalid level will be modified.

After the system is reset, the brake circuit is disabled and the MOE bit is low. Setting the BKE bit in the PWMA_BKR register can enable the brake function. The polarity of the brake input signal can be selected by configuring the BKP bit in the same register. BKE and BKP can be modified at the same time.

The falling edge of MOE can be asynchronous with respect to the clock module, so a resynchronization circuit is set between the actual signal (acting on the output) and the synchronization control bit (in the PWMA_BKR register). This resynchronization circuit creates a delay between the asynchronous signal and the synchronous signal. In particular, if you write MOE=1 when it is low, you must insert a delay (null instruction) before reading it to read the correct value. This is because asynchronous signals are written and synchronous signals are read.

When a brake occurs (the selected level appears at the brake input), the following actions are taken:

The MOE bit is cleared asynchronously, putting the output in an inactive state, an idle state or a reset state (selected by the OSSI bit). This feature is still valid when the MCU's oscillator is turned off.

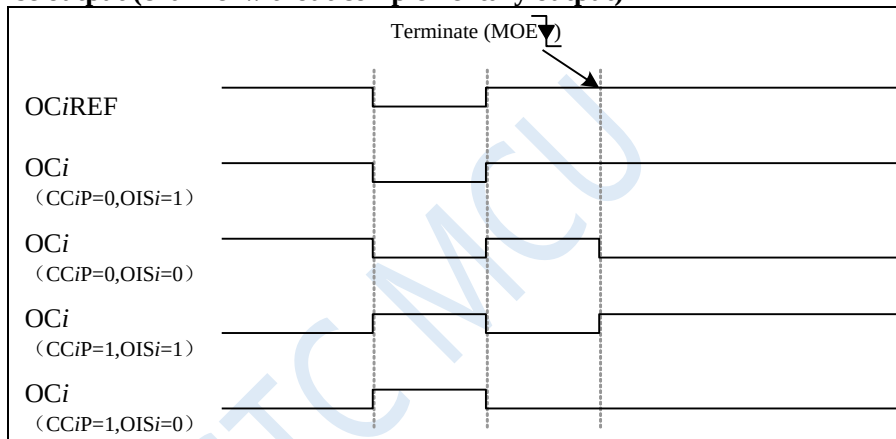
- Once MOE=0, each output channel outputs the level set by the OISi bit of the PWMA_OISR register. If OSSI=0, the timer no longer controls the output enable signal, otherwise the output enable signal is always high.
- When using complementary output:
 - The output is first placed in a reset state, that is, an invalid state (depending on polarity). This is an asynchronous operation. This function is valid even when the timer does not have a clock.

- If the timer clock still exists, the dead-band generator will take effect again and drive the output port according to the level indicated by the OISi and OISiN bits after the dead-band. Even in this case, OSi and OSiN cannot be driven to a valid level at the same time. Note: Because of the resynchronization of the MOE, the dead time is longer than usual (about 2 clock cycles).
- If the BIE bit of the PWMA_IER register is set, when the brake status flag (the BIF bit in the PWMA_SR1 register) is 1, an interrupt is generated.
- If the AOE bit in the PWMA_BKR register is set, the MOE bit will be automatically set in the next update event UEV. For example, this can be used for waveform control, otherwise, MOE will always remain low until it is set to 1 again. This feature can be used in safety, you can connect the brake input to the power-driven alarm output, thermal sensor or other safety devices.

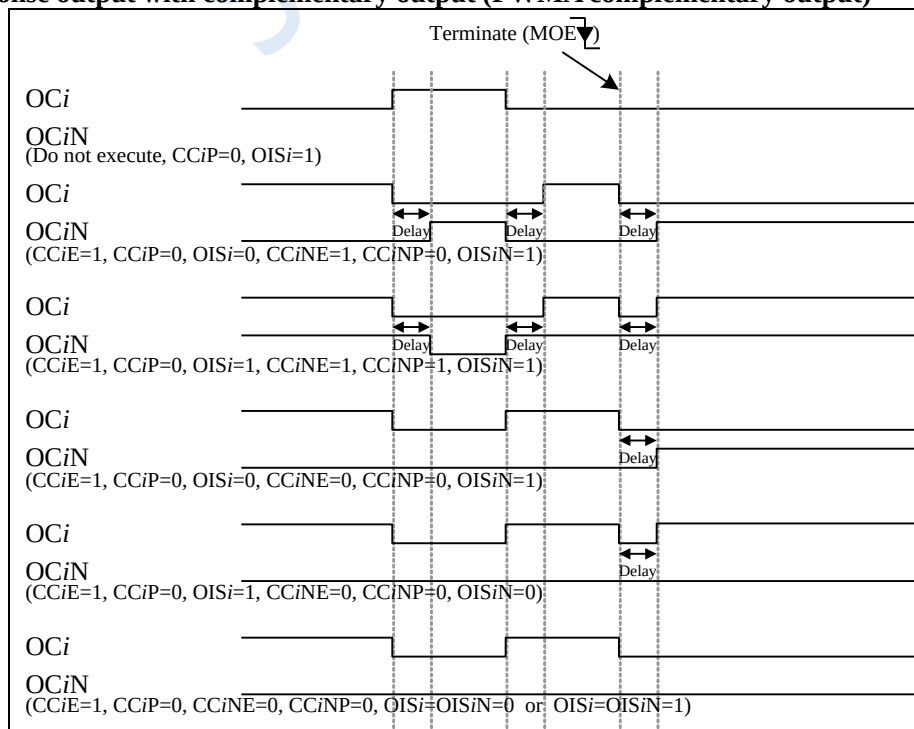
Note: The brake input is level effective. Therefore, when the brake input is valid, MOE cannot be set at the same time (automatically or through software). At the same time, the status flag BIF cannot be cleared.

The brake is generated by the BRK input, its effective polarity is programmable, and it is enabled or disabled by the BKE bit of the PWMA_BKR register. In addition to brake input and output management, write protection is also implemented in the brake circuit to ensure the safety of the application. It allows the user to freeze several configuration parameters (OCi polarity and state when disabled, OCIM configuration, brake enable and polarity). The user can select one of three levels of protection through the LOCK bit of the PWMA_BKR register. The LOCK bit field can only be modified once after the MCU is reset.

Brake response output (channel without complementary output)



Brake response output with complementary output (PWMA complementary output)



20.5.9 Clear the OCiREF signal when an external event occurs

For a given channel, a high level at the ETRF input terminal (set the corresponding OCiCE bit in the PWMA_CCMRi register to '1') can pull the OCiREF signal low, and the OCiREF signal will remain low until the next update event UEV occurs. This function can only be used in output comparison mode and PWM mode, but not in forced mode.

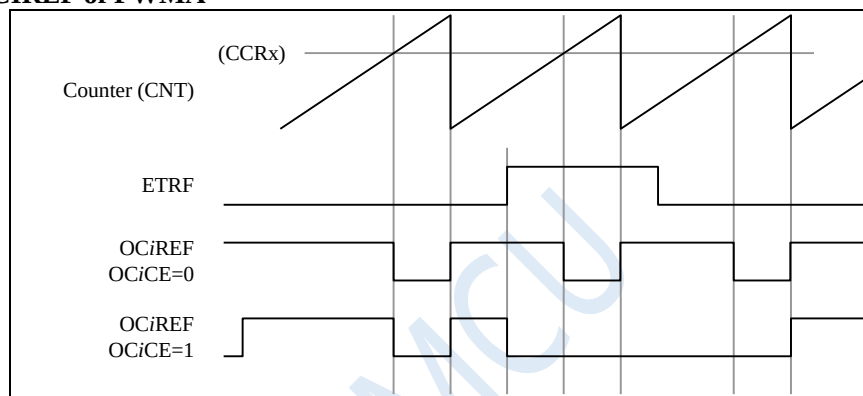
For example, the OCiREF signal can be connected to the output of a comparator to control the current. At this time, ETR must be configured as follows:

1. The external trigger prescaler must be turned off: ETPS[1:0]=00 in the PWMA_ETR register.
2. The external clock mode 2: ECE=0 in the PWMA_ETR register must be disabled.
3. External trigger polarity (ETP) and external trigger filter (ETF) can be configured as required.

The following figure shows the action of the OCiREF signal corresponding to different OCiCE values when the ETRF input goes high.

In this example, the timer PWMA is placed in PWM mode.

ETR clear OCiREF of PWMA



20.5.10 Encoder interface mode

The encoder interface mode is generally used for motor control.

The method to select the encoder interface mode is:

- If the counter only counts on the edge of TI2, set SMS=001 in the PWMA_SMCR register;
- If counting only on the edge of TI1, set SMS=010;
- If the counter counts on both TI1 and TI2 edges, set SMS=011.

By setting the CC1P and CC2P bits in the PWMA_CCER1 register, the TI1 and TI2 polarity can be selected; if necessary, the input filter can also be programmed.

Two inputs TI1 and TI2 are used as the interface of incremental encoder. Assuming that the counter has been started (CEN=1 in the PWMA_CR1 register), the counter counts every time TI1FP1 or TI2FP2 has a valid transition. TI1FP1 and TI2FP2 are the signals of TI1 and TI2 after passing the input filter and polarity control. If there is no filtering and polarity conversion, then TI1FP1=TI1, TI2FP2=TI2. According to the jump sequence of the two input signals, count pulses and direction signals are generated. According to the transition sequence of the two input signals, the counter counts up or down, and the hardware sets the DIR bit of the PWMA_CR1 register accordingly. Regardless of whether the counter counts on TI1, TI2, or both TI1 and TI2, a transition on either input (TI1 or TI2) will recalculate the DIR bit.

The encoder interface mode is basically equivalent to using an external clock with direction selection. This means that the counter only counts continuously from 0 to the autoload value of the PWMA_ARR register (according to the direction, either 0 to ARR or ARR to 0). Therefore, PWMA_ARR must be configured before counting. In this mode, the capturer, comparator, prescaler, repeat counter, trigger output feature, etc. still work as usual. Encoder mode and external clock mode 2 are not compatible, so they cannot be operated at the same time.

In the encoder interface mode, the counter is automatically modified according to the speed and direction of the incremental encoder, so the content of the counter always indicates the position of the encoder, and the counting direction corresponds to the direction of rotation of the connected sensor.

The following table lists all possible combinations (assuming that TI1 and TI2 do not change at the same time).

The relationship between counting direction and encoder signal

Active edge	Relative signal level (TI1FP1 corresponds to TI2, TI2FP2 corresponds to TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Count only at TI1	High	count down	count up	does not count	does not count
	Low	count up	count down	does not count	does not count
Count only at TI2	High	does not count	does not count	count up	count down
	Low	does not count	does not count	count down	count up
Count at TI1 and TI2	High	count down	count up	count up	count down
	Low	count up	count down	count down	count up

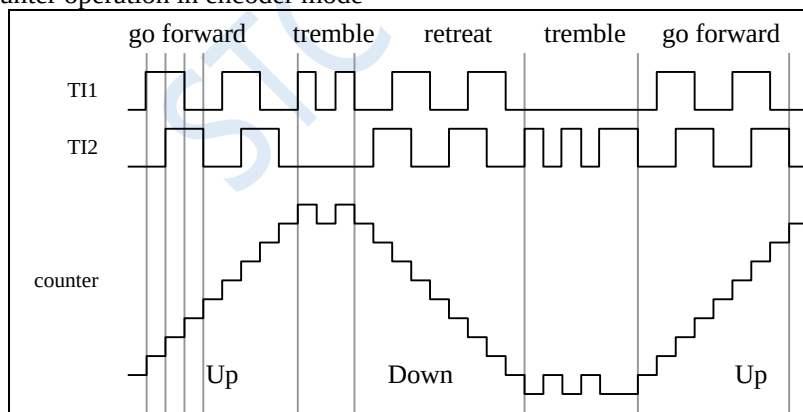
An external incremental encoder can be directly connected to the MCU without external interface logic. However, a comparator is generally used to edit

The differential output of the encoder is converted into a digital signal, which greatly increases the ability to resist noise interference. The third signal output by the encoder represents the mechanical zero point, which can be connected to an external interrupt input and trigger a counter reset.

The following is an example of counter operation, showing the count signal generation and direction control. It also shows how input jitter is suppressed when both edges are selected; jitter may occur when the sensor is close to a transition point. In this example, we assume the following configuration:

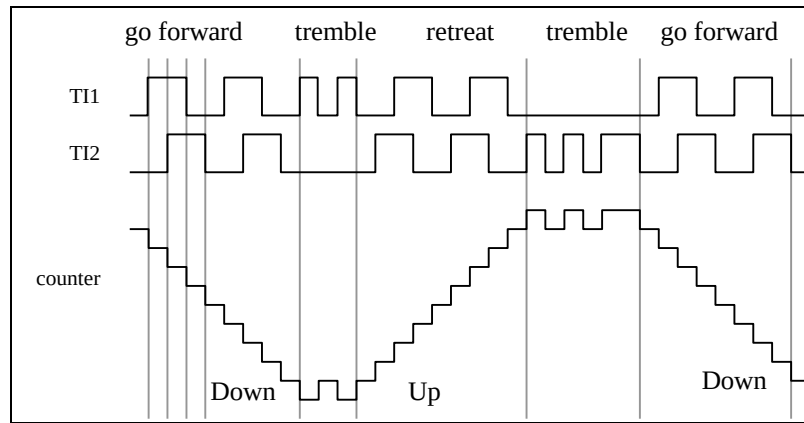
- CC1S=01 (PWMA_CCMR1 register, IC1FP1 is mapped to TI1)
- CC2S=01 (PWMA_CCMR2 register, IC2FP2 is mapped to TI2)
- CC1P=0 (PWMA_CCER1 register, IC1 is not inverted, IC1=TI1)
- CC2P=0 (PWMA_CCER1 register, IC2 is not inverted, IC2=TI2)
- SMS=011 (PWMA_SMCR register, all inputs are valid on rising and falling edges).
- CEN=1 (PWMA_CR1 register, counter enable)

Example of counter operation in encoder mode



The following figure shows the operation example of the counter when the polarity of IC1 is reversed (CC1P=1, other configurations are the same as the above example)

IC1 inverted encoder interface mode example



When the timer is configured in the encoder interface mode, it provides information about the current position of the sensor. Use another timer configured in capture mode to measure the interval between two encoder events to obtain dynamic information (speed, acceleration, deceleration). The encoder output indicating the mechanical zero point can be used for this purpose. According to the interval between two events, the counter can be read at a certain time interval. If possible, you can latch the value of the counter into the third input capture register (the capture signal must be periodic and can be generated by another timer).

20.6 Interrupt

PWMA/PWMB each have 8 interrupt request sources:

- The brake is interrupted
- Trigger interrupt
- COM event interrupt
- Input capture/output compare 4 interrupt
- Input capture/output compare 3 interrupt
- Input capture/output compare 2 interrupt
- Input capture/output compare 1 interrupt
- Update event interrupt (such as: counter overflow, underflow and initialization)

In order to use the interrupt feature, for each interrupt channel used, set the corresponding interrupt enable bit in the PWM_IER/PWMB_IER register: BIE, TIE, COMIE, CCiIE, UIE bit. By setting the corresponding bits in the PWMA_EGR/PWMB_EGR register, the above-mentioned interrupt sources can also be generated by software.

20.7 PWMA/PWMB registers description

20.7.1 Output Enable Registers (PWMx_ENO)

Symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ENO	FEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P
PWMB_ENO	FEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P

ENO8P: PWM8 output control bit

0: Disable PWM8 output

1: Enable PWM8 output

ENO7P: PWM7 output control bit

0: Disable PWM7 output

1: Enable PWM7 output

ENO6P: PWM6 output control bit

0: Disable PWM6 output

1: Enable PWM6 output
 ENO5P: PWM5 output control bit
 0: Disable PWM5 output
 1: Enable PWM5 output
 ENO4N: PWM4N output control bit
 0: Disable PWM4N output
 1: Enable PWM4N output
 ENO4P: PWM4P output control bit
 0: Disable PWM4P output
 1: Enable PWM4P output
 ENO3N: PWM3N output control bit
 0: Disable PWM3N output
 1: Enable PWM3N output
 ENO3P: PWM3P output control bit
 0: Disable PWM3P output
 1: Enable PWM3P output
 ENO2N: PWM2N output control bit
 0: Disable PWM2N output
 1: Enable PWM2N output
 ENO2P: PWM2P output control bit
 0: Disable PWM2P output
 1: Enable PWM2P output
 ENO1N: PWM1N output control bit
 0: Disable PWM1N output
 1: Enable PWM1N output
 ENO1P: PWM1P output control bit
 0: Disable PWM1P output
 1: Enable PWM1P output

20.7.2 Output Additional Enable Registers (PWMx_IOAUX)

Symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX	FEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P
PWMB_IOAUX	FEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P

AUX8P: PWM8 output additional control bit
 0: PWM8 output is directly controlled by ENO8P
 1: The output of PWM8 is controlled by ENO8P and PWMB_BKR
 AUX7P: PWM7 output additional control bit
 0: The output of PWM7 is directly controlled by ENO7P
 1: The output of PWM7 is controlled by ENO7P and PWMB_BKR
 AUX6P: PWM6 output additional control bit
 0: The output of PWM6 is directly controlled by ENO6P
 1: The output of PWM6 is jointly controlled by ENO6P and PWMB_BKR
 AUX5P: PWM5 output additional control bit
 0: The output of PWM5 is directly controlled by ENO5P
 1: The output of PWM5 is controlled by ENO5P and PWMB_BKR
 AUX4N: PWM4N output additional control bit
 0: The output of PWM4N is directly controlled by ENO4N
 1: The output of PWM4N is controlled by ENO4N and PWMA_BKR
 AUX4P: PWM4P output additional control bit
 0: The output of PWM4P is directly controlled by ENO4P
 1: The output of PWM4P is jointly controlled by ENO4P and PWMA_BKR
 AUX3N: PWM3N output additional control bit
 0: The output of PWM3N is directly controlled by ENO3N
 1: The output of PWM3N is controlled by ENO3N and PWMA_BKR
 AUX3P: PWM3P output additional control bit

- 0: The output of PWM3P is directly controlled by ENO3P
 1: The output of PWM3P is jointly controlled by ENO3P and PWMA_BKR
- AUX2N: PWM2N output additional control bit
 0: PWM2N output is directly controlled by ENO2N
 1: The output of PWM2N is controlled by ENO2N and PWMA_BKR
- AUX2P: PWM2P output additional control bit
 0: PWM2P output is directly controlled by ENO2P
 1: The output of PWM2P is jointly controlled by ENO2P and PWMA_BKR
- AUX1N: PWM1N output additional control bit
 0: PWM1N output is directly controlled by ENO1N
 1: The output of PWM1N is jointly controlled by ENO1N and PWMA_BKR
- AUX1P: PWM1P output additional control bit
 0: The output of PWM1P is directly controlled by ENO1P
 1: The output of PWM1P is jointly controlled by ENO1P and PWMA_BKR

20.7.3 Control Registers 1 (PWMx_CR1)

Symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR1	FEC0H	ARPEA	CMSA[1:0]		DIRA	OPMA	URSA	UDISA	CENA
PWMB_CR1	FEE0H	ARPEB	CMSB[1:0]		DIRB	OPMB	URSB	UDISB	CENB

- ARPEn: Automatic preloading permission bit (n=A,B)
 0: PWMn_ARR register is not buffered, it can be written directly
 1: PWMn_ARR register is buffered by the preload buffer
- CMSn[1:0]: select alignment mode (n=A,B)

CMSn[1:0]	Alignment mode	Description
00	Edge alignment mode	The counter counts up or down according to the direction bit (DIR)
01	Center alignment mode 1	The counter counts up and down alternately. The output compare interrupt flag bit of the channel configured as an output is only set when the counter counts down.
10	Center alignment mode2	The counter counts up and down alternately. The output compare interrupt flag bit of the channel configured as an output is only set when the counter is counting up.
11	Center alignment mode3	The counter counts up and down alternately. The output compare interrupt flag bit of the channel configured as output is set to 1 when the counter is counting up and down.

Note 1: When the counter is turned on (CEN=1), it is not allowed to switch from edge-aligned mode to center-aligned mode.

Note 2: In the center-aligned mode, the encoder mode (SMS=001, 010, 011) must be disabled.

DIRn: counting direction of the counter (n= A, B)

- 0: The counter counts up;
 1: The counter counts down.

Note: When the counter is configured in center aligned mode or encoder mode, this bit is read-only.

OPMn: Single pulse mode (n= A, B)

- 0: When an update event occurs, the counter does not stop;
 1: When the next update event occurs, the CEN bit is cleared and the counter stops.

URSn: Update request source (n= A, B)

- 0: If UDIS allows the generation of update events, any of the following events will generate an update interrupt:
- Register is updated (counter overflow/underflow)
 - Software setting UG bit
 - Updates generated by the clock/trigger controller

1: If UDIS allows the generation of update events, the update interrupt will only be generated when the following events occur, and UIF is set to 1:

- Register is updated (counter overflow/underflow)

UDISn: Update is prohibited (n= A, B)

0: Once the following events occur, an update (UEV) event occurs:

- Counter overflow/underflow
- Generate software update events
- Hardware reset generated by the clock/trigger mode controller The cached registers are loaded with their preload values.

1: No update event is generated, and the shadow registers (ARR, PSC, CCRx) maintain their values. If the UG bit is set or the clock/trigger controller issues a hardware reset, the counter and prescaler are reinitialized.

CENn: Allow counter (n= A, B)

0: disable the counter;

1: Enable the counter.

Note: The external clock, gating mode and encoder mode can only work after the CEN bit is set by the software. However, the trigger mode can automatically set the CEN bit by hardware.

20.7.4 Control Registers 2 (PWMx_CR2), and trigger ADC in real time

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR2	FEC1H	TI1S	MMSA[2:0]			-	COMSA	-	CCPCA
PWMB_CR2	FEE1H	TI5S	MMSB[2:0]			-	COMSB	-	CCPCB

TI1S: TI1 selection of the first group of PWM/PWMA

0: PWM1P input pin is connected to TI1 (digital filter input);

1: PWM1P, PWM2P and PWM3P pins are connected to TI1 of the first group of PWM after exclusive OR.

TI5S: TI5 selection of the second set of PWM/PWMB

0: PWM5 input pin is connected to TI5 (digital filter input);

1: PWM5, PWM6 and PWM7 pins are connected to TI5 of the second group of PWM after exclusive OR.

MMSA[2:0]: Main mode selection

MMSA[2:0]	Master mode	Description
000	Reset	The UG bit of the PWMA_EGR register is used as a trigger output (TRGO). If the trigger input (the clock/trigger controller is configured in reset mode) generates a reset, the signal on TRGO will have a delay relative to the actual reset
001	Enable	The counter enable signal is used as a trigger output (TRGO). It is used to start the ADC in order to control the enabling of the ADC within a period of time. The counter enable signal is generated by the logical OR of the CEN control bit and the trigger input signal in the gating mode. Unless the master/slave mode is selected, there will be a delay on TRGO when the counter enable signal is controlled by the trigger input. Note: When you need to use PWM to trigger ADC conversion, you need to set ADC_POWER, ADC_CHS and ADC_EPWMT in ADC_CONTR register first. When PWM generates TRGO internal signal, the system will automatically set ADC_START to start AD conversion. For detailed usage, please refer to the sample program. "Use PWM CEN to start PWMA timer and trigger ADC in real time"
010	Update	Update event is selected as trigger output (TRGO)
011	Comparison pulse	Once a capture or a comparison is successful, when the CC1IF flag is set to 1, the trigger output sends a positive pulse (TRGO)
100	Compare	OC1REF signal is used as trigger output (TRGO)
101	Compare	OC2REF signal is used as trigger output (TRGO)

110	Compare	OC3REF signal is used as trigger output (TRGO)
111	Compare	OC4REF signal is used as trigger output (TRGO)

MMSB[2:0]: Main mode selection

MMSB[2:0]	Master mode	Description
000	Reset	The UG bit of the PWMB_EGR register is used as a trigger output (TRGO). If the trigger input (the clock/trigger controller is configured in reset mode) generates a reset, the signal on TRGO will have a delay relative to the actual reset.
001	Enable	The counter enable signal is used as a trigger output (TRGO). It is used to start multiple PWMs in order to control to enable slave PWM for a period of time. The counter enable signal is generated by the logical OR of the CEN control bit and the trigger input signal in the gating mode. Unless the master/slave mode is selected, there will be a delay on TRGO when the counter enable signal is controlled by the trigger input.
010	Enable	Update event is selected as trigger output (TRGO)
011	Update	Once a capture occurs or a comparison is successful, when the CC5IF flag is set to 1, the trigger output sends a positive pulse (TRGO)
100	Comparison pulse	OC5REF signal is used as trigger output (TRGO)
101	Compare	OC6REF signal is used as trigger output (TRGO)
110	Compare	OC7REF signal is used as trigger output (TRGO)
111	Compare	OC8REF signal is used as trigger output (TRGO)

Note: Only the TRGO of the first set of PWM can be used to trigger the start of ADC

Note: Only the TRGO of the second group of PWM can be used for the ITR2 of the first group of PWM

COMSn: Update control selection of capture/compare control bit (n=A, B)

0: When CCPCn=1, these control bits are only updated when the COMG bit is set to 1.

1: When CCPCn=1, these control bits will be updated only when the COMG bit is 1 or TRGI has a rising edge

CCPCn: capture/compare preload control bit (n=A,B)

0: CCIE, CCINE, CCiP, CCiNP and OCIM bits are not preloaded

1: CCIE, CCINE, CCiP, CCiNP and OCIM bits are pre-loaded; after setting this bit, they will only be updated after setting the COMG bit.

Note: This bit only works on channels with complementary outputs.

20.7.5 Slave Mode Control Registers (PWMx_SMCR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SMCR	FEC2H	MSMA	TSA[2:0]			-	SMSA[2:0]		
PWMB_SMCR	FEE2H	MSMB	TSB[2:0]			-	SMSB[2:0]		

MSMn: Master/Slave mode (n= A, B)

0: No effect

1: The event on the trigger input (TRGI) is delayed to allow perfect synchronization between PWMn and its slave PWM (via TRGO)

TSA[2:0]: trigger source selection

TSA[2:0]	Trigger source
000	-
001	-
010	Internal trigger ITR2
011	-
100	TI1 edge detector (TI1F_ED)
101	Filtered timer input 1 (TI1FP1)

110	Filtered timer input 2 (TI2FP2)
111	External trigger input (ETRF)

TSB[2:0]: trigger source selection

TSB[2:0]	Trigger source
000	-
001	-
010	-
011	-
100	TI5 edge detector (TI5F_ED)
101	Filtered timer input 1 (TI5FP5)
110	Filtered timer input 2 (TI5FP6)
111	External trigger input (ETRF)

Note: These bits can only be changed when SMS=000, to avoid false edge detection when changing.

SMSA[2:0]: clock/trigger/slave mode selection

SMSA[2:0]	function	Description
000	Internal clock mode	If CEN=1, the prescaler is directly driven by the internal clock
001	Encoder mode 1	According to the level of TI1FP1, the counter counts up/down on the edge of TI2FP2
010	Encoder mode 2	According to the level of TI2FP2, the counter counts up/down on the edge of TI1FP1
011	Encoder mode 3	According to the level of another input, the counter counts up/down on the edge of TI1FP1 and TI2FP2
100	Reset mode	Reinitialize the counter on the rising edge of the selected trigger input (TRGI) and generate a signal to update the register
101	Gating mode	When the trigger input (TRGI) is high, the counter clock is turned on. Once the trigger input goes low, the counter stops (but does not reset). The start and stop of the counter are controlled
110	Trigger mode	The counter is started (but not reset) on the rising edge of the trigger input TRGI, only the start of the counter is controlled
111	External clock mode 1	The rising edge of the selected trigger input (TRGI) drives the counter. Note: If TI1F_ED is selected as the trigger input (TS=100), do not use the gated mode. This is because TI1F_ED only outputs a pulse every time TI1F changes, but the gate control mode is to check the level of the trigger input

SMSB[2:0]: clock/trigger/slave mode selection

SMSB[2:0]	function	Description
000	Internal clock mode	If CEN=1, the prescaler is directly driven by the internal clock
001	Encoder mode 1	According to the level of TI5FP5, the counter counts up/down on the edge of TI6FP6
010	Encoder mode 2	According to the level of TI6FP6, the counter counts up/down on the edge of TI5FP5
011	Encoder mode 3	According to the level of another input, the counter counts up/down on the edge of TI5FP5 and TI6FP6
100	Reset mode	Reinitialize the counter on the rising edge of the selected trigger input (TRGI) and generate a signal to update the register
101	Gating mode	When the trigger input (TRGI) is high, the counter clock is turned on. Once the trigger input goes low, the counter stops (but does not reset). The start and stop of the counter are controlled
110	Trigger mode	The counter is started (but not reset) on the rising edge of the trigger input TRGI, only the start of the counter is controlled
111	External clock mode 1	The rising edge of the selected trigger input (TRGI) drives the counter. Note: If TI5F_ED is selected as the trigger input (TS=100), do not use the gated mode. This is because TI5F_ED only outputs a pulse every time TI5F changes, but the gate control mode is to check the level of the trigger input.

20.7.6 External Trigger Registers (PWMx_ETR)

Symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETR	FEC3H	ETP1	ECEA	ETPSA[1:0]		ETFA[3:0]			
PWMB_ETR	FEE3H	ETP2	ECEB	ETPSB[1:0]		ETFB[3:0]			

ETPn: Polarity of external trigger ETR (n= A, B)

0: high level or rising edge valid

1: Low level or falling edge valid

ECEn: External clock enable (n= A, B)

0: Disable external clock mode 2

1: Enable external clock mode 2, the counter clock is the valid edge of ETRF.

Note 1: The effect of setting ECE to 1 is the same as selecting external clock mode 1 connecting TRGI to ETRF (in the PWMn_SMCR register, SMS=111, TS=111).

Note 2: External clock mode 2 can be used simultaneously with the following modes: trigger standard mode; trigger reset mode; trigger gated mode. However, at this time, TRGI must not be connected to ETRF (in the PWMn_SMCR register, TS cannot be 111).

Note 3: External clock mode 1 and external clock mode 2 are enabled at the same time, and the external clock input is ETRF.

ETPSn: The frequency of the external trigger signal ETRP of the external trigger prescaler cannot exceed fMASTER/4. You can use a prescaler to reduce low ETRP frequency, it is very useful when ETRP frequency is high: (n=A,B)

00: prescaler is off

01: ETRP frequency/2

02: ETRP frequency/4

03: ETRP frequency/8

ETFn[3:0]: External trigger filter selection, this bit field defines the ETRP sampling frequency and digital filter length. (N=A,B)

ETFn[3:0]	Number of clocks	ETF[3:0]	Number of clocks
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

20.7.7 Interrupt Enable Registers (PWMx_IER)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	FEC4H	BIEA	TIEA	COMIEA	CC4IE	CC3IE	CC2IE	CC1IE	UIEA
PWMB_IER	FEE4H	BIEB	TIEB	COMIEB	CC8IE	CC7IE	CC6IE	CC5IE	UIEB

BIEn: Allow brake interruption (n= A, B)

0: Prohibit brake interruption;

1: Allow brake interruption.

- TIE: trigger interrupt enable (n= A, B)
 0: Prohibit triggering interrupts;
 1: Enable trigger interrupt.
- COMIE: Allow COM interruption (n= A, B)
 0: Disable COM interrupt;
 1: Enable COM interrupt.
- CCnIE: Allow capture/compare n interrupts (n=1,2,3,4,5,6,7,8)
 0: Disable capture/compare n interrupt;
 1: Allow capture/compare n interrupts.
- UIEn: Allow update interruption (n= A, B)
 0: Update interrupt is prohibited;
 1: Allow update interruption.

20.7.8 Status Registers 1 (PWMx_SR1)

symbol	sddress	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	FEC5H	BIFA	TIFA	COMIFA	CC4IF	CC3IF	CC2IF	CC1IF	UIFA
PWMB_SR1	FEE5H	BIFB	TIFB	COMIFB	CC8IF	CC7IF	CC6IF	CC5IF	UIFB

BIFn: Brake interruption mark. Once the brake input is valid, the bit is set by the hardware. If the brake input is invalid, this bit can be cleared by software. (N=A,B)

0: No brake event occurs

1: Valid level detected on brake input

TIFn: Trigger interrupt mark. The bit is set by hardware when a trigger event occurs. Cleared by software. (N=A,B)

0: No trigger event is generated

1: Trigger interrupt waiting for response

COMIFn: COM interrupt flag. Once a COM event occurs, this bit is set by hardware. Cleared by software. (N=A,B)

0: No COM event is generated

1: COM interrupt waiting for response

CC8IF: capture/compare 8 interrupt flag, refer to CC1IF description

CC7IF: Capture/Compare 7 interrupt flag, refer to CC1IF description

CC6IF: Capture/compare 6 interrupt flag, refer to CC1IF description

CC5IF: Capture/compare 5 interrupt flag, refer to CC1IF description

CC4IF: capture/compare 4 interrupt flag, refer to CC1IF description

CC3IF: Capture/Compare 3 interrupt flag, refer to CC1IF description

CC2IF: Capture/Compare 2 interrupt flag, refer to CC1IF description

CC1IF: Capture/Compare 1 interrupt flag.

If channel CC1 is configured as output mode:

This bit is set by hardware when the counter value matches the comparison value, except in the center symmetric mode. It is cleared by software.

0: No match occurred;

1: The value of PWMA_CNT matches the value of PWMA_CCR1.

Note: In the center symmetry mode, when the counter value is 0, count up, when the counter value is ARR, count down (it counts up from 0 to ARR-1, and then counts down from ARR to 1). Therefore, for all SMS bit values, these two values are not marked. However, if CCR1>ARR, when CNT reaches the ARR value, CC1IF is set.

If channel CC1 is configured as input mode:

This bit is set by hardware when a capture event occurs, and it is cleared by software or cleared by reading PWMA_CCR1L.

0: No input capture is generated

1: The counter value has been captured to PWMA_CCR1

UIFn: Update interrupt flag This bit is set by hardware when an update event is generated. It is cleared by software. (N=A,B)

0: No update event is generated

1: Update event waiting for response. This bit is set by hardware when the register is updated

- If UDIS=0 in the PWMn_CR1 register, when the counter overflows or underflows
- If UDIS=0 and URS=0 in the PWMn_CR1 register, when the UG bit of the PWMn_EGR register is set by the software to reinitialize the counter CNT
- If UDIS=0 and URS=0 in the PWMn_CR1 register, when the counter CNT is reinitialized by a trigger event

20.7.9 Status Registers 2 (PWMx_SR2)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-
PWMB_SR2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

CC8OF: capture/compare 8 repeat capture flag. See CC1OF description.

CC7OF: Capture/Compare 7 repeat capture flag. See CC1OF description.

CC6OF: Capture/Compare 6 Repeat Capture Mark. See CC1OF description.

CC5OF: Capture/Compare 5 repeat capture flag. See CC1OF description.

CC4OF: Capture/Compare 4 repeat capture flag. See CC1OF description.

CC3OF: Capture/Compare 3 repeat capture flag. See CC1OF description.

CC2OF: Capture/Compare 2 repeat capture flag. See CC1OF description.

CC1OF: Capture/Compare 1 repeat capture flag. This flag can be set by hardware only when the corresponding channel is configured as input capture. Write 0 to clear this bit.

0: No repeated capture is generated;

1: When the counter value is captured to the PWMA_CCR1 register, the state of CC1IF is already 1.

20.7.10 Event Generation Registers (PWMx_EGR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_EGR	FEC7H	BGA	TGA	COMGA	CC4G	CC3G	CC2G	CC1G	UGA
PWMB_EGR	FEE7H	BGB	TGB	COMGB	CC8G	CC7G	CC6G	CC5G	UGB

BGn: A brake event is generated. This bit is set by software to generate a brake event and is automatically cleared by hardware (n = A, B)

0: No action

1: Generate a brake event. At this moment, MOE=0, BIF=1, if the corresponding interrupt is turned on (BIE=1), the corresponding interrupt will be generated

TGn: Generate a trigger event. This bit is set by software to generate a trigger event and is automatically cleared by hardware (n = A, B)

0: No action

1: TIF=1, if the corresponding interrupt is turned on (TIE=1), the corresponding interrupt will be generated

COMGn: Capture/compare events and generate control updates. This bit is set by software and cleared by hardware automatically (n= A, B)

0: No action

1: CCPC=1, allow to update CCIE, CCINE, CCiP, CCiNP, OCIM bits.

Note: This bit is only valid for channels with complementary outputs

CC8G: Generate capture/compare 8 events. Refer to CC1G description

CC7G: Generate capture/compare 7 events. Refer to CC1G description

CC6G: Generate capture/compare 6 events. Refer to CC1G description

CC5G: Generate capture/compare 5 events. Refer to CC1G description

CC4G: Generate capture/compare 4 events. Refer to CC1G description

CC3G: Generate capture/compare 3 events. Refer to CC1G description

CC2G: Generate capture/compare 2 event. Refer to CC1G description

CC1G: Generate capture/compare 1 event. Generate capture/compare 1 event. This bit is set by software to generate a capture/compare event and is automatically cleared by hardware.

0: No action;

1: Generate a capture/compare event on channel CC1.

If channel CC1 is configured as output: set CC1IF=1, if the corresponding interrupt is turned on, the corresponding interrupt will be generated.

If channel CC1 is configured as input: the current counter value is captured to the PWMA_CCR1 register, set CC1IF=1, if the corresponding interrupt is enabled, the corresponding interrupt will be generated. If CC1IF is already 1, set CC1OF=1.

UGn: Update event generated. This bit is set by software and cleared by hardware automatically. (N=A,B)

0: No action;

1: Reinitialize the counter and generate an update event.

Note that the counter of the prescaler is also cleared to 0 (but the prescaler coefficient remains unchanged).

If in center symmetric mode or DIR=0 (counting up), the counter will be cleared; if DIR=1 (counting down), the counter will take the value of PWMn_ARR..

20.7.11 Capture/Compare Mode Register 1 (PWMx_CCMR1)

The channel can be used to capture input mode or compare output mode, and the direction of the channel is defined by the corresponding CCnS bit. The functions of other bits of this register are different in input and output modes. OCxx describes the function of the channel in output mode, and ICxx describes the function of the channel in input mode. Therefore, it must be noted that the function of the same bit in output mode and input mode is different.

Channel is configured to compare output mode

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	FEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
PWMB_CCMR1	FEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]	

OCnCE: Output compare n clear enable. This bit is used to enable the use of external events on the PWMETI pin to clear the output signal of channel n (OCnREF) (n=1,5)

0: OCnREF is not affected by ETRF input;

1: Once the ETRF input high level is detected, OCnREF=0.

OCnM[2:0]: Output compare n mode. These 3 bits define the action of the output reference signal OCnREF, and OCnREF determines the value of OCn. OCnREF is active high, and the active level of OCn depends on the CCnP bit. (N=1,5)

OCnM[2:0]	mode	Description
000	freeze	The comparison between PWMn_CCR1 and PWMn_CNT has no effect on OCnREF
001	Set channel n when matching Output is valid level	When PWMn_CCR1=PWMn_CNT, OCnREF output high
010	Set channel n when matching Output is invalid level	When PWMn_CCR1=PWMn_CNT, OCnREF output low
011	Flip	When PWMn_CCR1=PWMn_CNT, flip OCnREF
100	Forced to invalid level	Force OCnREF to low
101	Forced to active level	Force OCnREF to be high
110	PWM mode 1	When counting up, when PWMn_CNT<PWMn_CCR1, OCnREF outputs high, otherwise OCnREF outputs low When counting down, when PWMn_CNT>PWMn_CCR1 OCnREF output low, otherwise OCnREF output high
111	PWM mode 2	When counting up, when PWMn_CNT<PWMn_CCR1OCnREF outputs low, otherwise OCnREF outputs high When counting down, when PWMn_CNT>PWMn_CCR1 OCnREF output high, otherwise OCnREF output low

Note 1: Once the LOCK level is set to 3 (LOCK bit in the PWMn_BKR register) and CCnS=00 (the channel is configured as an output), this bit cannot be modified.

Note 2: In PWM mode 1 or PWM mode 2, the OCnREF level only changes when the comparison result is changed or when switching from freeze mode to PWM mode in output comparison mode.

Note 3: On channels with complementary outputs, these bits are pre-loaded. If the CCPC of the PWMn_CR2 register is 1, the OCM bit will take a new value from the preload bit only when a COM event occurs.

OCnPE: output compare n preload enable (n=1,5)

0: Disable the preload function of the PWMn_CCR1 register, and the PWMn_CCR1 register can be written at any time, and the newly written value will take effect immediately.

1: Enable the preload function of the PWMn_CCR1 register. Read and write operations only operate on the preload register. The preload value of PWMn_CCR1 is loaded into the current register when the update event arrives.

Note 1: Once the LOCK level is set to 3 (LOCK bit in the PWMn_BKR register) and CCnS=00 (the channel is configured as an output), this bit cannot be modified.

Note 2: In order to operate correctly, the preload function must be enabled in the PWM mode. But in single pulse mode (OPM=1 in PWMn_CR1 register), it is not necessary.

OCnFE: Output compare n fast enable. This bit is used to speed up the response of the CC output to the trigger input event. (N=1,5)

0: According to the value of the counter and CCRn, CCn operates normally, even if the trigger is turned on. When the flip-flop input has a valid edge, the minimum delay for activating the CCn output is 5 clock cycles.

1: The effect of the valid edge input to the trigger is like a comparison match. Therefore, OC is set to the comparison level regardless of the comparison result. The delay between the valid edge of the sampling flip-flop and the CC1 output is shortened to 3 clock cycles. OCFE only works when the channel is configured in PWMA or PWMB mode.

CC1S[1:0]: Capture/compare 1 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC5S[1:0]	direction	Input pin
00	Output	
01	Input	IC1 is mapped on TI1FP1
10	Input	IC1 is mapped on TI2FP1
11	Input	IC1 is mapped on TRC. This mode only works when the internal trigger input is selected (selected by the TS bit of the PWMA_SMCR register)

CC5S[1:0]: Capture/compare 5 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC5S[1:0]	direction	Input pin
00	Output	
01	Input	IC5 is mapped on TI5FP5
10	Input	IC5 is mapped on TI6FP5
11	Input	IC5 is mapped on TRC. This mode only works when the internal trigger input is selected (selected by the TS bit of the PWM5_SMCR register)

Note: CC1S is writable only when the channel is closed (CC1E=0 in PWMA_CCER1 register).

Note: CC5S is writable only when the channel is closed (CC5E=0 in PWM5_CCER1 register).

Channel is configured to capture input mode

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	FEC8H	IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
PWMB_CCMR1	FEE8H	IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]	

ICnF[3:0]: Input capture n filter selection, this bit field defines the sampling frequency of TIn and the length of the digital filter. (N=1,5)

ICnF[3:0]	Number of clocks	ICnF[3:0]	Number of clocks
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80

0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

Note: Even for channels with complementary outputs, this bit field is not preloaded, and the value of CCPC (PWMn_CR2 register) will not be considered

ICnPSC[1:0]: Input/capture n prescaler. These two bits define the prescaler coefficient of CCn input (IC1). (N=1,5)

00: No prescaler, every edge detected on the capture input port triggers a capture

01: Trigger a capture every 2 events

10: Trigger a capture every 4 events

11: Trigger a capture every 8 events

CC1S[1:0]: Capture/compare 1 selection.

These two bits define the direction of the channel (input/output), and the selection of input pins

CC1S[1:0]	direction	input pin
00	Output	
01	Input	IC1 is mapped on TI1FP1
10	Input	IC1 is mapped on TI2FP1
11	Input	IC1 is mapped on TRC. This mode only works when the internal trigger input is selected (selected by the TS bit of the PWMA_SMCR register)

CC5S[1:0]: Capture/compare 5 selection.

These two bits define the direction of the channel (input/output), and the selection of input pins

CC5S[1:0]	direction	input pin
00	Output	
01	Input	IC5 is mapped on TI5FP5
10	Input	IC5 is mapped on TI6FP5
11	Input	IC5 is mapped on TRC. This mode only works when the internal trigger input is selected (selected by the TS bit of the PWM5_SMCR register)

Note: CC1S is writable only when the channel is closed (CC1E=0 in PWMA_CCER1 register).

Note: CC5S is writable only when the channel is closed (CC5E=0 in PWM5_CCER1 register).

20.7.12 Capture/compare mode registers 2 (PWMx_CCMR2)

Channel is configured to compare output mode

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	FEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	
PWMB_CCMR2	FEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]	

OCnCE: Output compare n clear enable. This bit is used to enable the use of external events on the PWMETI pin to clear the output signal of channel n (OCnREF) (n=2,6)

0: OCnREF is not affected by ETRF input;

1: Once the ETRF input high level is detected, OCnREF=0.

OCnM[2:0]: Output compare 2 mode, refer to OC1M. (N=2,6)

OCnPE: Output compare 2 preload enable, refer to OP1PE. (N=2,6)

CC2S[1:0]: Capture/compare 2 selection.

These two bits define the direction of the channel (input/output), and the selection of input pins

CC2S[1:0]	direction	Input pin
00	Output	
01	Input	IC2 is mapped on TI2FP2
10	Input	IC2 is mapped on TI1FP2
11	Input	IC2 is mapped on TRC

CC6S[1:0]: Capture/compare 6 selection.

These two bits define the direction of the channel (input/output), and the selection of input pins

CC6S[1:0]	direction	Input pin
00	Output	
01	Input	IC6 is mapped on TI6FP6
10	Input	IC6 is mapped on TI5FP6
11	Input	IC6 is mapped on TRC

Channel is configured to capture input mode

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	FEC9H	IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]	
PWMB_CCMR2	FEE9H	IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]	

ICnF[3:0]: Input capture n filter selection, refer to IC1F. (N=2,6)

ICnPSC[1:0]: Input/capture n prescaler, refer to IC1PSC. (N=2,6)

CC2S[1:0]: Capture/compare 2 selection.

These two bits define the direction of the channel (input/output), and the selection of input pins

CC2S[1:0]	direction	Input pin
00	Output	
01	Input	IC2 is mapped on TI2FP2
10	Input	IC2 is mapped on TI1FP2
11	Input	IC2 is mapped on TRC。

CC6S[1:0]: Capture/compare 6 selection.

These two bits define the direction of the channel (input/output), and the selection of input pins

CC6S[1:0]	direction	Input pin
00	Output	
01	Input	IC6 is mapped on TI6FP6
10	Input	IC6 is mapped on TI5FP6
11	Input	IC6 is mapped on TRC

20.7.13 Capture/Compare Mode Registers 3 (PWMx_CCMR3)

Channel is configured to compare output mode

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	FECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
PWMB_CCMR3	FEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]	

OCnCE: Output compare n clear enable. This bit is used to enable the use of external events on the PWMETI pin to clear the output signal of channel n (OCnREF) (n=3,7)

0: OCnREF is not affected by ETRF input;

1: Once the ETRF input high level is detected, OCnREF=0.

OCnM[2:0]: Output compare 3 mode, refer to OC1M. (N=3,7) OCnPE: Output compare 3 preload enable, refer to OP1PE. (N=3,7)

CC3S[1:0]: Capture/compare 3 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC3S[1:0]	direction	Input pin
00	Output	
01	Input	IC3 is mapped on TI3FP3
10	Input	IC3 is mapped on TI4FP3
11	Input	IC3 is mapped on TRC

CC7S[1:0]: Capture/compare 7 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC7S[1:0]	direction	Input pin
00	Output	
01	Input	IC7 is mapped on TI7FP7
10	Input	IC7 is mapped on TI8FP7
11	Input	IC7 is mapped on TRC

Channel is configured to capture input mode

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	FECAH	IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
PWMB_CCMR3	FEEAH	IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]	

ICnF[3:0]: Input capture n filter selection, refer to IC1F. (N=3,7)

ICnPSC[1:0]: Input/capture n prescaler, refer to IC1PSC. (N=3,7)

CC3S[1:0]: Capture/compare 3 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC3S[1:0]	direction	Input pin
00	Output	
01	Input	IC3 is mapped on TI3FP3
10	Input	IC3 is mapped on TI4FP3
11	Input	IC3 is mapped on TRC

CC7S[1:0]: Capture/compare 7 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC7S[1:0]	direction	Input pin
00	Output	
01	Input	IC7 is mapped on TI7FP7
10	Input	IC7 is mapped on TI8FP7
11	Input	IC7 is mapped on TRC

20.7.14 Capture/Compare Mode Registers 4 (PWMx_CCMR4)

Channel is configured to compare output mode

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	FECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	
PWMB_CCMR4	FEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]	

OCnCE: Output compare n clear enable. This bit is used to enable the use of external events on the PWMETI pin to clear the output signal of channel n (OCnREF) (n=4,8)

0: OCnREF is not affected by ETRF input;

1: Once the ETRF input high level is detected, OCnREF=0.

OCnM[2:0]: Output compare n mode, refer to OC1M. (N=4,8) OCnPE: output compare n preload enable, refer to OP1PE. (N=4,8)

CC4S[1:0]: Capture/compare 4 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC4S[1:0]	direction	Input pin
00	Output	
01	Input	IC4 is mapped on TI4FP4
10	Input	IC4 is mapped on TI3FP4
11	Input	IC4 is mapped on TRC。

CC8S[1:0]: Capture/compare 8 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC8S[1:0]	direction	Input pin
00	Output	
01	Input	IC8 is mapped on TI8FP8
10	Input	IC8 is mapped on TI7FP8
11	Input	IC8 is mapped on TRC。

Channel is configured to capture input mode

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	FECBH	IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]	
PWMB_CCMR4	FEEBH	IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]	

ICnF[3:0]: Input capture n filter selection, refer to IC1F. (N=4,8)

ICnPSC[1:0]: Input/capture n prescaler, refer to IC1PSC. (N=4,8)

CC4S[1:0]: Capture/compare 4 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC4S[1:0]	direction	Input pin
00	Output	
01	Input	IC4 is mapped on TI4FP4
10	Input	IC4 is mapped on TI3FP4
11	Input	IC4 is mapped on TRC。

CC8S[1:0]: Capture/compare 8 selection. These two bits define the direction of the channel (input/output), and the selection of input pins

CC8S[1:0]	direction	Input pin
00	Output	
01	Input	IC8 is mapped on TI8FP8
10	Input	IC8 is mapped on TI7FP8
11	Input	IC8 is mapped on TRC。

20.7.15 Capture/Compare Enable Registers 1 (PWMx_CCER1)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER1	FECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
PWMB_CCER1	FEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E

CC6P: OC6 input capture/compare output polarity. Reference CC1P

CC6E: OC6 input capture/compare output enable. Reference CC1E

CC5P: OC5 input capture/compare output polarity. Reference CC1P

CC5E: OC5 input capture/compare output enable. Reference CC1E

CC2NP: OC2N compare output polarity. Reference CC1NP

CC2NE: OC2N compare output enable. Reference CC1NE

CC2P: OC2 input capture/compare output polarity. Reference CC1P

CC2E: OC2 input capture/compare output enable. Reference CC1E

CC1NP: OC1N compare output polarity

0: High level is active;

1: Low level is active.

Note 1: Once the LOCK level (LOCK bit in the PWMA_BKR register) is set to 3 or 2 and CC1S=00 (the channel is configured as an output), this bit cannot be modified.

Note 2: For channels with complementary outputs, this bit is pre-loaded. If CCPC=1 (PWMA_CR2 register), only when a COM event occurs, the CC1NP bit will take a new value from the preload bit.

CC1NE: OC1N compare output enable

0: Turn off the comparison output.

1: Turn on the comparison output. The output level depends on the value of the MOE, OSSI, OSSR, OIS1, OIS1N and CC1E bits.

Note: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA_CR2 register), only when a COM event occurs, the CC1NE bit will take a new value from the preload bit.

CC1P: OC1 input capture/compare output polarity

CC1 channel is configured as output:

0: high level is active

1: low level is active

CC1 channel is configured as input or capture:

0: Capture occurs on the rising edge of TI1F or TI2F;

1: Capture occurs on the falling edge of TI1F or TI2F.

CC1E: OC1 input capture/compare output enable

0: Turn off input capture/compare output;

1: Turn on input capture/compare output.

Note 1: Once the LOCK level (LOCK bit in PWMA_BKR register) is set to 3 or 2, this bit cannot be modified.

Note 2: For channels with complementary outputs, this bit is pre-loaded. If CCPC=1 (PWMA_CR2 register), the CC1P bit will take a new value from the preload bit only when a COM event occurs.

Control bits of complementary output channels OC_i and OC_{iN} with brake function

Control bit					output status	
MOE	OSSI	OSSR	CCiE	CCiNE	OC _i output status	OC _{iN} output status
1	X	0	0	0	Output prohibited	Output prohibited
		0	0	1	Output prohibited	OCiREF with polarity
		0	1	0	OCiREF with polarity	Output prohibited
		0	1	1	OCiREF with polarity and dead zone	Reverse OCiREF with polarity and dead zone
		1	0	0	Output prohibited	Output prohibited

		1	0	1	Disabled (The output is enabled and invalid level) OCi=CCiP	OCiREF with polarity
		1	1	0	OCiREF with polarity	Disabled (The output is enabled and invalid level) OCi=CCiP
		1	1	1	OCiREF with polarity and dead zone	Reverse OCiREF with polarity and dead zone
0	0	X	X	X	Output prohibited	
	1				Closed state (output enabled and at an invalid level) asynchronously: OCi=CCiP, OCIN=CCiNP; Then, if the clock exists: OCi=OISi, OCIN=OISiN after a dead time, assuming that OISi and OISiN are not both Corresponds to the effective level of OCi and OCIN.	

Note: The status of the external I/O pins connected to the complementary OCi and OCiN channels depends on the status of the OCi and OCiN channels and the GPIO register.

20.7.16 Capture/Compare Enable Registers 2 (PWMx_CCER2)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER2	FECDDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E
PWMB_CCER2	FEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E

CC8P: OC8 input capture/compare output polarity. Reference CC1P

CC8E: OC8 input capture/compare output enable. Reference CC1E

CC7P: OC7 input capture/compare output polarity. Reference CC1P

CC7E: OC7 input capture/compare output enable. Reference CC1E

CC4NP: OC4N compare output polarity. Reference CC1NP

CC4NE: OC4N compare output enable. Reference CC1NE

CC4P: OC4 input capture/compare output polarity. Reference CC1P

CC4E: OC4 input capture/compare output enable. Reference CC1E

CC3NP: OC3N compare output polarity. Reference CC1NP

CC3NE: OC3N compare output enable. Reference CC1NE

CC3P: OC3 input capture/compare output polarity. Reference CC1P

CC3E: OC3 input capture/compare output enable. Reference CC1E

20.7.17 Counters upper 8 bits (PWMx_CNTRH)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRH	FECEH	CNT1[15:8]							
PWMB_CNTRH	FEFEH	CNT2[15:8]							

CNTn[15:8]: the high 8-bit value of the counter (n= A, B)

20.7.18 Counters low 8 bits (PWMx_CNTRL)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRL	FECFH	CNT1[7:0]							
PWMB_CNTRL	FEFFH	CNT2[7:0]							

CNTn[7:0]: the high 8-bit value of the counter (n= A, B)

20.7.19 Prescalers high 8 bits (PWMx_PSCRH), output frequency calculation formula

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRH	FED0H	PSC1[15:8]							
PWMB_PSCRH	FEF0H	PSC2[15:8]							

PSCn[15:8]: The high 8-bit value of the prescaler. (N=A,B)

The prescaler is used to divide the frequency of CK_PSC. The counter clock frequency (fCK_CNT) is equal to fCK_PSC/(PSCR[15:0]+1).

PSCR contains the value loaded into the current prescaler register when an update event occurs (update events include the counter being cleared by the UG bit of TIM_EGR or cleared by the slave controller working in reset mode). This means that in order for the new value to take effect, an update event must be generated.

PWM output frequency calculation formula

The output frequency calculation formula of the PWMA and PWMB two groups of PWM is the same, and each group can set a different frequency.

Alignment mode	PWM output frequency calculation formula
Edge alignment	$\text{PWM output frequency} = \frac{\text{System clock SYSclk}}{(\text{PWMx\_PSCR} + 1) \times (\text{PWMx\_ARR} + 1)}$
Center aligned	$\text{PWM output frequency} = \frac{\text{System clock SYSclk}}{(\text{PWMx\_PSCR} + 1) \times \text{PWMx\_ARR} \times 2}$

20.7.20 Prescalers low 8 bits (PWMx_PSCRL)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRL	FED1H	PSC1[7:0]							
PWMB_PSCRL	FEF1H	PSC2[7:0]							

PSCn[7:0]: The lower 8-bit value of the prescaler. (N=A,B)

20.7.21 The upper 8 bits of the auto-reload registers (PWMx_ARRH)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRH	FED2H	ARR1[15:8]							
PWMB_ARRH	FEF2H	ARR2[15:8]							

ARRn[15:8]: automatically reload the high 8-bit value (n= A, B)

ARR contains the value to be loaded into the actual auto-reload register. When the value of auto reload is 0, the counter does not work.

20.7.22 The lower 8 bits of the auto-reload registers (PWMx_ARRL)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRL	FED3H	ARR1[7:0]							
PWMB_ARRL	FEF3H	ARR2[7:0]							

ARRn[7:0]: Automatic reloading of the lower 8-bit value (n= A, B)

20.7.23 Repeat Counter Registers (PWMx_RCR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_RCR	FED4H	REP1[7:0]							
PWMB_RCR	FEF4H	REP2[7:0]							

REPn[7:0]: Repeat counter value (n= A, B)

After enabling the preload function, these bits allow the user to set the update rate of the compare register (that is, periodically transfer from the preload register to the current register); if the update interrupt is allowed, it will also affect the rate of the update interrupt. Every time the down counter REP_CNT reaches 0, an update event is generated and the counter REP_CNT restarts counting from the REP value. Since REP_CNT only reloads the REP value when the period update event U_RC occurs, the new value written to the PWMn_RCR register will only take effect when the next period update event occurs. This means that in the PWM mode, (REP+1) corresponds to:

- In edge-aligned mode, the number of PWM cycles;
- In the center symmetric mode, the number of PWM half cycles;

20.7.24 Capture/Compare Registers 1/5 High 8 bits (PWMx_CCR1H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	FED5H	CCR1[15:8]							
PWMB_CCR5H	FEF5H	CCR5[15:8]							

CCRn[15:8]: Capture/compare the high 8-bit value of n (n=1,5)

If the CCn channel is configured as output: CCRn contains the loaded current comparison value (preload value).

If the preload function is not selected in the PWMn_CCMR1 register (OCnPE bit), the written value will be transferred to the current register immediately. Otherwise, only when an update event occurs, the preload value will be transferred to the current capture/compare n register. The current comparison value is compared with the value of the counter PWMn_CNT, and an output signal is generated on the OCn port.

If the CCn channel is configured as an input: CCRn contains the counter value when the last input capture event occurred (this register is read-only at this time).

20.7.25 Capture/compare registers 1/5 lower 8 bits (PWMx_CCR1L)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1L	FED6H	CCR1[7:0]							
PWMB_CCR5L	FEF6H	CCR5[7:0]							

CCRn[7:0]: Capture/compare the low 8-bit value of n (n=1,5)

20.7.26 Capture/Compare Registers 2/6 High 8-bit (PWMx_CCR2H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2H	FED7H	CCR2[15:8]							
PWMB_CCR6H	FEF7H	CCR6[15:8]							

CCRN[15:8]: Capture/compare the high 8-bit value of n (n=2,6)

20.7.27 Capture/compare registers 2/6 lower 8 bits (PWMx_CCR2L)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2L	FED8H	CCR2[7:0]							
PWMB_CCR6L	FEF8H	CCR6[7:0]							

CCRN[7:0]: Capture/compare the low 8-bit value of n (n=2,6)

20.7.28 Capture/Compare Registers 3/7 High 8 bits (PWMx_CCR3H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3H	FED9H	CCR3[15:8]							
PWMB_CCR7H	FEF9H	CCR7[15:8]							

CCRN[15:8]: Capture/compare the high 8-bit value of n (n=3,7)

20.7.29 Capture/compare registers 3/7 lower 8 bits (PWMx_CCR3L)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3L	FEDA H	CCR3[7:0]							
PWMB_CCR7L	FEFA H	CCR7[7:0]							

CCRN[7:0]: Capture/compare the low 8-bit value of n (n=3,7)

20.7.30 Capture/compare registers 4/8 high 8 bits (PWMx_CCR4H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4H	FEDBH	CCR4[15:8]							
PWMB_CCR8H	FEFBH	CCR8[15:8]							

CCRN[15:8]: Capture/compare the high 8-bit value of n (n=4,8)

20.7.31 Capture/compare registers 4/8 lower 8 bits (PWMx_CCR4L)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4L	FEDCH	CCR4[7:0]							
PWMB_CCR8L	FEFCH	CCR8[7:0]							

CCRn[7:0]: Capture/compare the low 8-bit value of n (n=4,8)

20.7.32 Brake Registers (PWMx_BKR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_BRK	FEDDH	MOEA	AOEA	BKPA	BKEA	OSSRA	OSSIA	LOCKA[1:0]	
PWMB_BRK	FEFDH	MOEB	AOEB	BKPB	BKEB	OSSRB	OSSIB	LOCKB[1:0]	

MOEn: Main output enable. Once the brake input is valid, this bit is asynchronously cleared by hardware. According to the setting value of the AOE bit, this bit can be set by software or be set automatically. It is only valid for channels configured as output. (N=A,B)

0: Disable OC and OCN output or force to idle state

1: If the corresponding enable bit (CCiE bit of PWMn_CCERX register) is set, OC and OCN output are enabled.

AOEn: automatic output enable (n= A, B)

0: MOE can only be set by software;

1: MOE can be set to 1 by the software or automatically set to 1 in the next update event (if the brake input is invalid).

Note: Once the LOCK level (LOCK bit in the PWMn_BKR register) is set to 1, this bit cannot be modified

BKPn: Brake input polarity (n= A, B)

0: The brake input is active at low level

1: The brake input is active at high level

Note: Once the LOCK level (LOCK bit in the PWMn_BKR register) is set to 1, this bit cannot be modified

BKEEn: Brake function enable (n= A, B)

0: Forbid brake input (BRK)

1: Turn on the brake input (BRK)

Note: Once the LOCK level (LOCK bit in PWMn_BKR register) is set to 1, this bit cannot be modified.

OSSRn: "Off state" selection in operating mode. This bit is valid when MOE=1 and the channel is set to output (n= A, B)

0: When PWM is not working, disable OC/OCN output (OC/OCN enable output signal=0);

1: When PWM is not working, once CCiE=1 or CCiNE=1, first turn on OC/OCN and output an invalid level, and then set OC/OCN enable output signal=1.

Note: Once the LOCK level (LOCK bit in PWMn_BKR register) is set to 2, this bit cannot be modified.

OSSIIn: "Off state" selection in idle mode. This bit is valid when MOE=0 and the channel is set to output. (N=A,B)

0: When PWM is not working, disable OC/OCN output (OC/OCN enable output signal=0);

1: When the PWM is not working, once CCiE=1 or CCiNE=1, OC/OCN outputs its idle level first, and then OC/OCN

Enable output signal=1.

Note: Once the LOCK level (LOCK bit in PWMn_BKR register) is set to 2, this bit cannot be modified.

LOCKn[1:0]: Lock settings. This bit provides write protection measures to prevent software errors (n= A, B)

LOCKn[1:0]	Protection level	Protect content
00	No protection	Register is not write protected
01	Lock level 1	Cannot write to the BKE, BKP, and AOE bits of the PWMn_BKR register, OISI bit of PWMn_OISR register
10	Lock level 2	You cannot write to the bits in lock level 1, Also cannot write CC polarity bit and OSSR/OSSI bit
11	Lock level 3	You cannot write to the bits in lock level 2, Can not write CC control bit

Note: Because the BKE, BKP, AOE, OSSR, OSSI bits can be locked (depending on the LOCK bit), they must be set when writing to the PWMn_BKR register for the first time.

20.7.33 Dead Zone Registers (PWMx_DTR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DTR	FEDEH	DTGA[7:0]							
PWMB_DTR	FEFEH	DTGB[7:0]							

DTGn[7:0]: Dead zone generator setting. (N=A,B)

These bits define the duration of the dead zone inserted between complementary outputs. (TCK_PSC is the clock pulse of PWMn)

DTGn[7:5]	Dead time
000	$DTGn[7:0] * t_{CK\_PSC}$
001	
010	
011	
100	$(64 + DTGn[6:0]) * 2 * t_{CK\_PSC}$
101	
110	$(32 + DTGn[5:0]) * 8 * t_{CK\_PSC}$
111	$(32 + DTGn[4:0]) * 16 * t_{CK\_PSC}$

20.7.34 Output Idle Status Registers (PWMx_OISR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_OISR	FEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1
PWMB_OISR	FEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5

OIS8: OC8 output level in idle state

OIS7: OC7 output level in idle state

OIS6: OC6 output level in idle state

OIS5: OC5 output level in idle state

OIS4N: OC4N output level in idle state

OIS4: idle state OC4 output level

OIS3N: OC3N output level in idle state

OIS3: OC3 output level in idle state

OIS2N: OC2N output level in idle state

OIS2: OC2 output level in idle state

OIS1N: OC1N output level in idle state

0: When MOE=0, after a dead time, OC1N=0;

1: When MOE=0, after a dead time, OC1N=1.

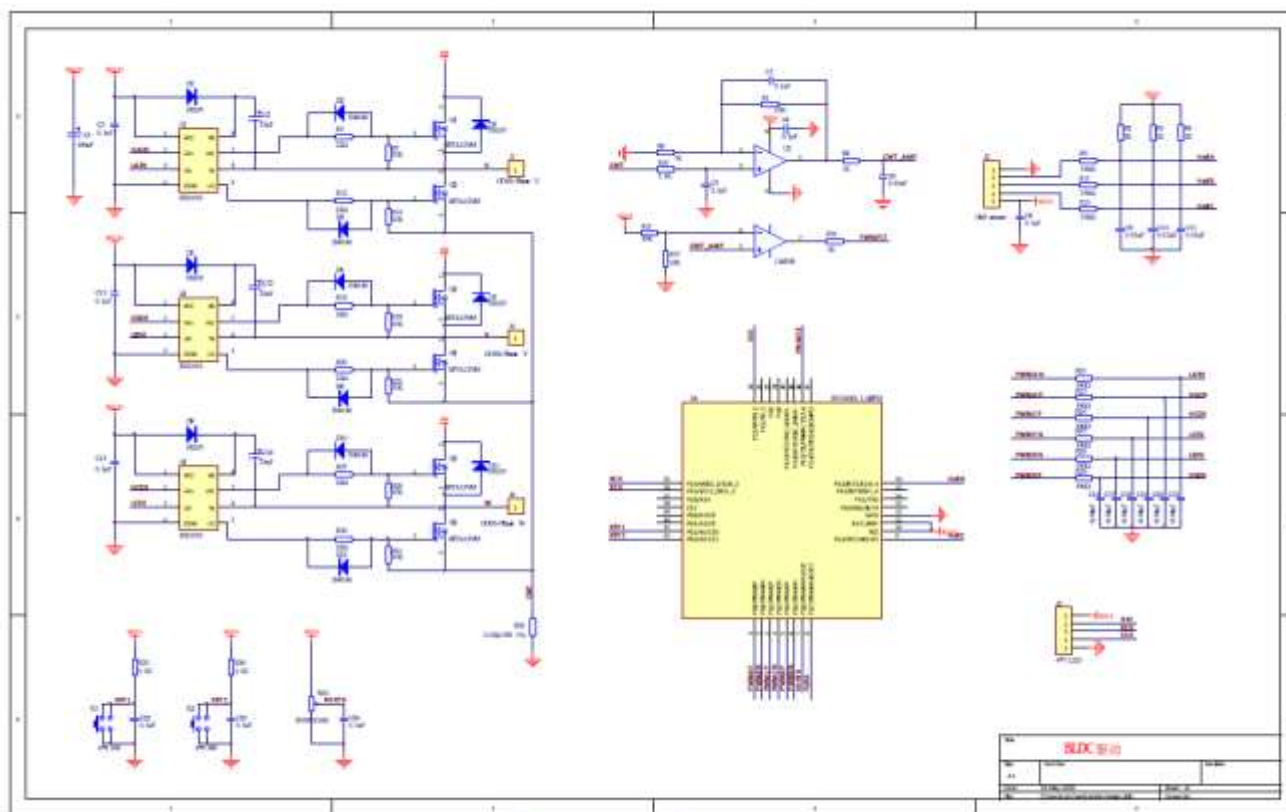
OIS1: OC1 output level in idle state

0: When MOE=0, if OC1N is enabled, after a dead zone, OC1=0;

1: When MOE=0, if OC1N is enabled, after a dead zone, OC1=1.

20.8 Sample Routines

20.8.1 Six-step BLDC brushless DC motor drive (with HALL)



C language code

//Operating frequency for test is 11.0592MHz

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

```
typedef unsigned char u8;
typedef unsigned int u16;
```

```
typedef struct TIM1_struct
{
```

```
volatile unsigned char CR1;
volatile unsigned char CR2;
volatile unsigned char SMCR;
volatile unsigned char ETR;
volatile unsigned char IER;
volatile unsigned char SR1;
volatile unsigned char SR2;
volatile unsigned char EGR;
volatile unsigned char CCMR1;
volatile unsigned char CCMR2;
volatile unsigned char CCMR3;
```

```
/*!< control register 1 */
/*!< control register 2 */
/*!< Synchro mode control register */
/*!< external trigger register */
/*!< interrupt enable register*/
/*!< status register 1 */
/*!< status register 2 */
/*!< event generation register */
/*!< CC mode register 1 */
/*!< CC mode register 2 */
/*!< CC mode register 3 */
```

```

volatile unsigned char CCMR4;
volatile unsigned char CCER1;
volatile unsigned char CCER2;
volatile unsigned char CNTRH;
volatile unsigned char CNTRL;
volatile unsigned char PSCRH;
volatile unsigned char PSCRL;
volatile unsigned char ARRH;
volatile unsigned char ARRL;
volatile unsigned char RCR;
volatile unsigned char CCR1H;
volatile unsigned char CCR1L;
volatile unsigned char CCR2H;
volatile unsigned char CCR2L;
volatile unsigned char CCR3H;
volatile unsigned char CCR3L;
volatile unsigned char CCR4H;
volatile unsigned char CCR4L;
volatile unsigned char BKR;
volatile unsigned char DTR;
volatile unsigned char OISR;
}TIM1_TypeDef;

#define TIM1_BaseAddress 0xFEC0
#define TIM2_BaseAddress 0xFEE0

#define TIM1 ((TIM1_TypeDef xdata*) TIM1_BaseAddress)
#define TIM2 ((TIM1_TypeDef xdata*) TIM2_BaseAddress)

#define PWMA_ETRPS (*(unsigned char volatile xdata *) 0xFEB0)
#define PWMA_ENO (*(unsigned char volatile xdata *) 0xFEB1)
#define PWMA_PS (*(unsigned char volatile xdata *) 0xFEB2)
#define PWMB_ENO (*(unsigned char volatile xdata *) 0xFEB5)
#define PWMB_PS (*(unsigned char volatile xdata *) 0xFEB6)

sfr ADC_CONTR = 0xbc;
sfr ADC_RES = 0xbd;
sfr ADC_RESL = 0xbe;
sfr ADCCFG = 0xde;
sfr CMPCR1 = 0xe6;
sfr CMPCR2 = 0xe7;

sfr P0M0 = 0x94;
sfr P0M1 = 0x93;
sfr P1M0 = 0x92;
sfr P1M1 = 0x91;
sfr P2M0 = 0x96;
sfr P2M1 = 0x95;
sfr P3M0 = 0xb2;
sfr P3M1 = 0xb1;
sfr P5M0 = 0xca;
sfr P5M1 = 0xc9;
sfr P5 = 0xc8;
sfr P_SW2 = 0xba;

sbit P00 = P0^0;
sbit P01 = P0^1;
sbit P02 = P0^2;
/*!< CC mode register 4 */
/*!< CC enable register 1 */
/*!< CC enable register 2 */
/*!< counter high */
/*!< counter low */
/*!< prescaler high */
/*!< prescaler low */
/*!< auto-reload register high */
/*!< auto-reload register low */
/*!< Repetition Counter register */
/*!< capture/compare register 1 high */
/*!< capture/compare register 1 low */
/*!< capture/compare register 2 high */
/*!< capture/compare register 2 low */
/*!< capture/compare register 3 high */
/*!< capture/compare register 3 low */
/*!< capture/compare register 3 high */
/*!< capture/compare register 3 low */
/*!< Break Register */
/*!< dead-time register */
/*!< Output idle register */

```

```

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    P50      =    P5^0;
sbit    P51      =    P5^1;
sbit    P52      =    P5^2;
sbit    P53      =    P5^3;
sbit    P54      =    P5^4;
sbit    P55      =    P5^5;

```

```

#define    TRUE          1
#define    FALSE         0

```

```

#define    RV09_CH       6

```

```

#define    TIM1_Period    ((u16)0x0180)
#define    TIM1_STPulse   ((u16)342)

```

```

#define    START          0x1A
#define    RUN             0x1B
#define    STOP           0x1C
#define    IDLE           0x1D

```

```

#define    TIM1_OCMODE_MASK    ((u8)0x70)
#define    TIM1_OCCE_ENABLE    ((u8)0x80)
#define    TIM1_OCCE_DISABLE   ((u8)0x00)
#define    TIM1_OCMODE_TIMING   ((u8)0x00)
#define    TIM1_OCMODE_ACTIVE   ((u8)0x10)

```

```

#define TIM1_OCMODE_INACTIVE ((u8)0x20)
#define TIM1_OCMODE_TOGGLE ((u8)0x30)
#define TIM1_FORCE_INACTIVE ((u8)0x40)
#define TIM1_FORCE_ACTIVE ((u8)0x50)
#define TIM1_OCMODE_PWMA ((u8)0x60)
#define TIM1_OCMODE_PWMB ((u8)0x70)
#define CC1_POLARITY_HIGH ((u8)0x02)
#define CC1N_POLARITY_HIGH ((u8)0x08)
#define CC2_POLARITY_HIGH ((u8)0x20)
#define CC2N_POLARITY_HIGH ((u8)0x80)
#define CC1_POLARITY_LOW ((u8)~0x02)
#define CC1N_POLARITY_LOW ((u8)~0x08)
#define CC2_POLARITY_LOW ((u8)~0x20)
#define CC2N_POLARITY_LOW ((u8)~0x80)
#define CC1_OCENABLE ((u8)0x01)
#define CC1N_OCENABLE ((u8)0x04)
#define CC2_OCENABLE ((u8)0x10)
#define CC2N_OCENABLE ((u8)0x40)
#define CC1_OCDISABLE ((u8)~0x01)
#define CC1N_OCDISABLE ((u8)~0x04)
#define CC2_OCDISABLE ((u8)~0x10)
#define CC2N_OCDISABLE ((u8)~0x40)
#define CC3_POLARITY_HIGH ((u8)0x02)
#define CC3N_POLARITY_HIGH ((u8)0x08)
#define CC4_POLARITY_HIGH ((u8)0x20)
#define CC4N_POLARITY_HIGH ((u8)0x80)
#define CC3_POLARITY_LOW ((u8)~0x02)
#define CC3N_POLARITY_LOW ((u8)~0x08)
#define CC4_POLARITY_LOW ((u8)~0x20)
#define CC4N_POLARITY_LOW ((u8)~0x80)
#define CC3_OCENABLE ((u8)0x01)
#define CC3N_OCENABLE ((u8)0x04)
#define CC4_OCENABLE ((u8)0x10)
#define CC4N_OCENABLE ((u8)0x40)
#define CC3_OCDISABLE ((u8)~0x01)
#define CC3N_OCDISABLE ((u8)~0x04)
#define CC4_OCDISABLE ((u8)~0x10)
#define CC4N_OCDISABLE ((u8)~0x40)

```

```
void LED_OUT(u8 X);
```

```
//LED single-byte serial shift function
```

```

unsigned char code LED_0F[] =
{
    0xC0,0xF9,0xA4,0xB0,
    0x99,0x92,0x82,0xF8,
    0x80,0x90,0x8C,0xBF,
    0xC6,0xA1,0x86,0xFF,
    0xbf
};

```

```

#define DIO          P23
#define RCLK         P24
#define SCLK         P25

```

```

// Serial data input
// Clock pulse signal - valid on rising edge
//input signal - valid on rising edge

```

```

void DelayXus(unsigned char delayTime);
void DelayXms(unsigned char delayTime);
unsigned int ADC_Convert(u8 ch);
void PWM_Init(void);

```

```

void SPEED_ADJ();
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP();
unsigned char KEY_detect();
void LED4_Display (unsigned int dat,unsigned char num);

unsigned char Display_num=1;
unsigned int Display_dat=0;
unsigned int Motor_speed;
unsigned char Motor_sta = IDLE;
unsigned char BRK_occur=0;
unsigned int TIM2_CAP1_v=0;
unsigned int CAP1_avg=0;
unsigned char CAP1_cnt=0;
unsigned long CAP1_sum=0;

void main(void)
{
    P_SW2 = 0x80;

    P1 = 0x00;
    P0M1 = 0x0C;
    P0M0 = 0x01;
    P1M1 = 0xc0;
    P1M0 = 0x3F;
    P2M1 = 0x00;
    P2M0 = 0x38;
    P3M1 = 0x28;
    P3M0 = 0x00;

    ET0=1;
    TR0=1;

    ADCCFG = 0x0f;
    ADC_CONTR = 0x80;

    PWMA_ENO = 0x3F;           //PWMA output enable
    PWMB_ENO = 0x00;           //PWMB output enable
    PWMA_PS = 0x00;            //PWMA pin selection
    PWMB_PS = 0xd5;            //PWMB pin selection

    /*****
output compare mode PWMx_duty = [CCRx/(ARR + 1)]*100
*****/
    /*****PWMB connect with hall sensor*****/
    ////////// time base unit //////////
    TIM2-> PSCRL = 15;
    TIM2-> ARR = 0xff;          //Auto-reload registers, counter overflow points
    TIM2-> ARRL = 0xff;
    TIM2-> CCR4H = 0x00;
    TIM2-> CCR4L = 0x05;

    //////////Channel configuration////////
    TIM2-> CCMR1 = 0x43;         //Channel Mode Configuration
    TIM2-> CCMR2 = 0x41;
    TIM2-> CCMR3 = 0x41;
    TIM2-> CCMR4 = 0x70;

```

```

TIM2-> CCER1 = 0x11;
TIM2-> CCER2 = 0x11;

//////////Mode Configuration//////////
TIM2-> CR2 = 0xf0;
TIM2-> CR1 = 0x81;
TIM2-> SMCR = 0x44;

//////////Enable & Interrupt Configuration//////////
TIM2-> BKR = 0x80;           //main output enable
TIM2-> IER = 0x02;          //enable interrupt

/*****PWMA Controls motor to change phase *****/
//////////time base unit//////////
TIM1-> PSCRH = 0x00;         //Prescaler register
TIM1-> PSCRL = 0x00;
TIM1-> ARRH = (u8)(TIM1_Period >> 8);
TIM1-> ARRL = (u8)(TIM1_Period);

//////////Channel configuration//////////
TIM1-> CCMR1 = 0x70;         //Channel Mode Configuration
TIM1-> CCMR2 = 0x70;
TIM1-> CCMR3 = 0x70;
TIM1-> CCER1 = 0x11;        //Config channel output enable and polarity
TIM1-> CCER2 = 0x01;        //Config channel output enable and polarity
TIM1-> OISR = 0xAA;         //Configure the output level of each channel when MOE=0

//////////Mode Configuration//////////
TIM1-> CR1 = 0xA0;
TIM1-> CR2 = 0x24;
TIM1-> SMCR = 0x20;

//////////Enable & Interrupt Configuration//////////
TIM1-> BKR = 0x1c;
TIM1-> CR1 |= 0x01;         //enable counter

EA = 1;
while (1)
{
    P22=~P22;
    Display_dat = Motor_speed; //Motor_speed

    switch(Motor_sta)
    {
        case START:
            MOTOR_START();
            Motor_sta = RUN;
            break;
        case RUN:
            SPEED_ADJ();
            if((KEY_detect() == 2)|| (BRK_occur == TRUE))
                Motor_sta = STOP;
            break;
        case STOP:
            MOTOR_STOP();
            Motor_sta = IDLE;
            break;
    }
}

```

```

        case IDLE:
            if(KEY_detect()==1)
                Motor_sta = START;
            BRK_occur = FALSE;
            Motor_speed = 0;
            CAP1_avg = 0;
            CAP1_cnt = 0;
            CAP1_sum = 0;
            break;
        }
    }
}

void TIM0_ISR() interrupt 1
{
    TH0=0xf0;
    if(Display_num>8)
        Display_num=1;
    LED4_Display(Display_dat,Display_num);
    Display_num=(Display_num<<1);
}

void PWMA_ISR() interrupt 26
{
    if((TIM1->SR1 & 0x20))
    {
        switch(RD_HALL())
        {
            case 3:
                TIM1->CCMR3 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR3 |= TIM1_FORCE_INACTIVE;
                TIM1->CCMR1 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR1 |= TIM1_OCMODE_PWMB;
                break;
            case 2:
                TIM1->CCER1 &= CC2N_POLARITY_LOW;
                TIM1->CCER2 |= CC3N_POLARITY_HIGH;
                break;
            case 6:
                TIM1->CCMR1 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR1 |= TIM1_FORCE_INACTIVE;
                TIM1->CCMR2 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR2 |= TIM1_OCMODE_PWMB;
                break;
            case 4:
                TIM1->CCER1 |= CC1N_POLARITY_HIGH;
                TIM1->CCER2 &= CC3N_POLARITY_LOW;
                break;
            case 5:
                TIM1->CCMR2 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR2 |= TIM1_FORCE_INACTIVE;
                TIM1->CCMR3 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR3 |= TIM1_OCMODE_PWMB;
                break;
            case 1:
                TIM1->CCER1 &= CC1N_POLARITY_LOW;
                TIM1->CCER1 |= CC2N_POLARITY_HIGH;
                break;
        }
    }
}

```

```

    }

    CAP1_sum += TIM2_CAP1_v;
    CAP1_cnt++;
    if(CAP1_cnt==128)
    {
        CAP1_cnt=0;
        CAP1_avg = (CAP1_sum>>7);
        CAP1_sum = 0;
        Motor_speed = 5000000/CAP1_avg;
    }

    TIM1->SR1 &=~0x20;                //clear
}
if((TIM1->SR1 & 0x80))                //BRK
{
    BRK_occur = TRUE;
    TIM1->SR1 &=~0x80;                //clear
}
}

void PWMB_ISR() interrupt 27
{
    if((TIM2->SR1 & 0x02))
    {
        TIM2_CAP1_v = TIM2->CCR1H;
        TIM2_CAP1_v = (TIM2_CAP1_v<<8) + TIM2->CCR1L;
        TIM2->SR1 &=~0x02;
    }
}

void DelayXus(unsigned char delayTime)
{
    int i = 0;
    while( delayTime--)
    {
        for( i = 0 ; i < 1 ; i++);
    }
}

void DelayXms( unsigned char delayTime )
{
    int i = 0;
    while( delayTime--)
    {
        for( i = 0 ; i < 2 ; i++)
        {
            DelayXus(100);
        }
    }
}

unsigned int ADC_Convert(u8 ch)
{
    u16 res=0;

    ADC_CONTR &= ~0x0f;
    ADC_CONTR |= ch;

```



```

    ADC_CONTR |= 0x40;
    DelayXus(1);
    while (!(ADC_CONTR & 0x20));
    ADC_CONTR &= ~0x20;

    res = ADC_RES;
    res = (res<<2)+(ADC_RES<>>6);
    return res;
}

void SPEED_ADJ()
{
    u16 ADC_result;

    ADC_result = (ADC_Convert(RV09_CH)/3);
    TIM1->CCR1H = (u8)(ADC_result >> 8);           //Counter comparison value
    TIM1->CCR1L = (u8)(ADC_result);
    TIM1->CCR2H = (u8)(ADC_result >> 8);
    TIM1->CCR2L = (u8)(ADC_result);
    TIM1->CCR3H = (u8)(ADC_result >> 8);
    TIM1->CCR3L = (u8)(ADC_result);
}

unsigned char RD_HALL()
{
    unsigned char Hall_sta = 0;

    (P17)? (Hall_sta|=0x01) : (Hall_sta&=~0x01);
    (P54)? (Hall_sta|=0x02) : (Hall_sta&=~0x02);
    (P33)? (Hall_sta|=0x04) : (Hall_sta&=~0x04);

    return Hall_sta;
}

void MOTOR_START()
{
    u16 temp;
    u16 ADC_result;

    TIM1->CCR1H = (u8)(TIM1_STPulse >> 8);           //Counter comparison value
    TIM1->CCR1L = (u8)(TIM1_STPulse);
    TIM1->CCR2H = (u8)(TIM1_STPulse >> 8);
    TIM1->CCR2L = (u8)(TIM1_STPulse);
    TIM1->CCR3H = (u8)(TIM1_STPulse >> 8);
    TIM1->CCR3L = (u8)(TIM1_STPulse);
    TIM1->BKR |= 0x80;                               //enable main output, equivalent to a master switch
    TIM1->IER |= 0xA0;                               //enable interrupt

    switch(RD_HALL())
    {
    case 1:
        TIM1->CCER1 &= CC1N_POLARITY_LOW;
        TIM1->CCER1 |= CC2N_POLARITY_HIGH;
        TIM1->CCER2 &= CC3N_POLARITY_LOW;
        TIM1->CCMR3 &= ~TIM1_OCMODE_MASK;
        TIM1->CCMR3 |= TIM1_FORCE_INACTIVE;
        TIM1->CCMR2 &= ~TIM1_OCMODE_MASK;
        TIM1->CCMR2 |= TIM1_FORCE_INACTIVE;

```

```

TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR1 |= TIM1_OCMODE_PWMB;
break;
case 3:
TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR1 |= TIM1_OCMODE_PWMB;
TIM1-> CCER1 &= CC1N_POLARITY_LOW;
TIM1-> CCER1 &= CC2N_POLARITY_LOW;
TIM1-> CCER2 |= CC3N_POLARITY_HIGH;
break;
case 2:
TIM1-> CCER1 &= CC1N_POLARITY_LOW;
TIM1-> CCER1 &= CC2N_POLARITY_LOW;
TIM1-> CCER2 |= CC3N_POLARITY_HIGH;
TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR2 |= TIM1_OCMODE_PWMB;
TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
break;
case 6:
TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR2 |= TIM1_OCMODE_PWMB;
TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
TIM1-> CCER1 |= CC1N_POLARITY_HIGH;
TIM1-> CCER1 &= CC2N_POLARITY_LOW;
TIM1-> CCER2 &= CC3N_POLARITY_LOW;
break;
case 4:
TIM1-> CCER1 |= CC1N_POLARITY_HIGH;
TIM1-> CCER1 &= CC2N_POLARITY_LOW;
TIM1-> CCER2 &= CC3N_POLARITY_LOW;
TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR3 |= TIM1_OCMODE_PWMB;
break;
case 5:
TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
TIM1-> CCMR3 |= TIM1_OCMODE_PWMB;
TIM1-> CCER1 &= CC1N_POLARITY_LOW;
TIM1-> CCER1 |= CC2N_POLARITY_HIGH;
TIM1-> CCER2 &= CC3N_POLARITY_LOW;
break;

```

```

}
ADC_result = (ADC_Convert(RV09_CH)/3);

for(temp = TIM1_STPulse; temp > ADC_result; temp--)
{
    TIM1-> CCR1H = (u8)(temp >> 8);           //Counter comparison value
    TIM1-> CCR1L = (u8)(temp);
    TIM1-> CCR2H = (u8)(temp >> 8);
    TIM1-> CCR2L = (u8)(temp);
    TIM1-> CCR3H = (u8)(temp >> 8);
    TIM1-> CCR3L = (u8)(temp);
    DelayXms(10);
}
}

void MOTOR_STOP()
{
    TIM1-> BKR &= ~0x80;
    TIM1-> IER &= ~0xA0;
}

void LED4_Display (u16 dat,u8 num)
{
    switch(num)
    {
    case 0x01:
        LED_OUT(LED_0F[(dat/1)%10]);
        LED_OUT(0x01);
        RCLK = 0;
        RCLK = 1;
        break;
    case 0x02:
        LED_OUT(LED_0F[(dat/10)%10]);
        LED_OUT(0x02);
        RCLK = 0;
        RCLK = 1;
        break;
    case 0x04:
        LED_OUT(LED_0F[(dat/100)%10]);
        LED_OUT(0x04);
        RCLK = 0;
        RCLK = 1;
        break;
    case 0x08:
        LED_OUT(LED_0F[(dat/1000)%10]);
        LED_OUT(0x08);
        RCLK = 0;
        RCLK = 1;
        break;
    }
}

void LED_OUT(u8 X)
{
    u8 i;

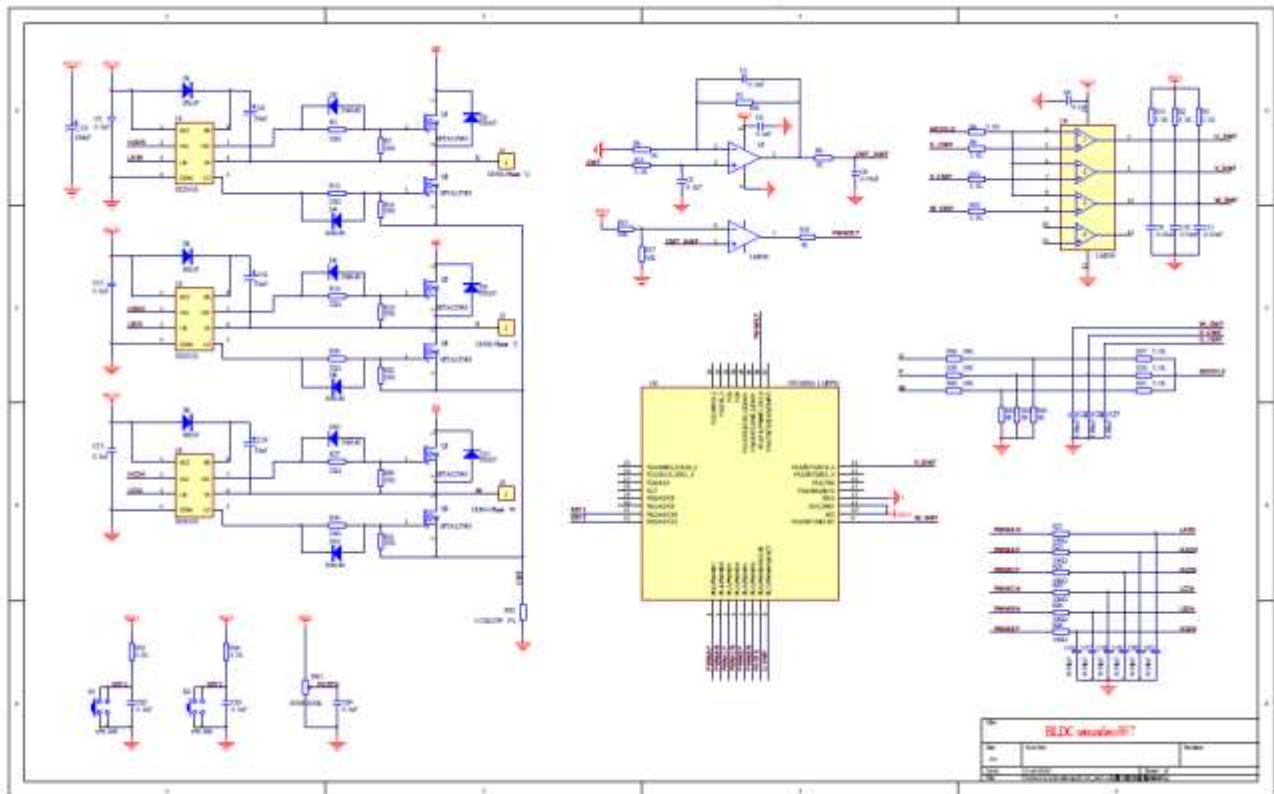
    for(i=8;i>=1;i--)
    {

```

```
        if (X&0x80) DIO=1;
        else DIO=0;
        X<<=1;
        SCLK = 0;
        SCLK = 1;
    }
}
```

```
unsigned char KEY_detect()
{
    if(!P02)
    {
        DelayXms(10);
        if(!P02)
        {
            return 1;
        }
        else return 0;
    }
    else if(!P03)
    {
        DelayXms(10);
        if(!P03)
        {
            return 2;
        }
        else return 0;
    }
    else return 0;
}
```

20.8.2 BLDC brushless DC motor drive (without HALL)



C language code

//Operating frequency for test is 11.0592MHz
 //This routine implements the following functions: Control the operation of the Hallless motor through 3 sets of PWM channels
 //This example is only applicable to the demonstration of 57BL02 motor under 24V no-load condition

```
#include "reg51.h"
#include "intrins.h"
#include "reg51.h"
typedef unsigned char u8;
typedef unsigned int u16;

typedef struct TIM1_struct
{
    volatile unsigned char CR1; /*!< control register 1 */
    volatile unsigned char CR2; /*!< control register 2 */
    volatile unsigned char SMCR; /*!< Synchro mode control register */
    volatile unsigned char ETR; /*!< external trigger register */
    volatile unsigned char IER; /*!< interrupt enable register*/
    volatile unsigned char SR1; /*!< status register 1 */
    volatile unsigned char SR2; /*!< status register 2 */
    volatile unsigned char EGR; /*!< event generation register */
    volatile unsigned char CCMR1; /*!< CC mode register 1 */
    volatile unsigned char CCMR2; /*!< CC mode register 2 */
    volatile unsigned char CCMR3; /*!< CC mode register 3 */
    volatile unsigned char CCMR4; /*!< CC mode register 4 */
    volatile unsigned char CCER1; /*!< CC enable register 1 */
    volatile unsigned char CCER2; /*!< CC enable register 2 */
    volatile unsigned char CNTRH; /*!< counter high */
    volatile unsigned char CNTRL; /*!< counter low */
    volatile unsigned char PSCRH; /*!< prescaler high */
    volatile unsigned char PSCRL; /*!< prescaler low */
    volatile unsigned char ARRH; /*!< auto-reload register high */
    volatile unsigned char ARRL; /*!< auto-reload register low */
    volatile unsigned char RCR; /*!< Repetition Counter register */
}
```

```

volatile unsigned char CCR1H; /*!< capture/compare register 1 high */
volatile unsigned char CCR1L; /*!< capture/compare register 1 low */
volatile unsigned char CCR2H; /*!< capture/compare register 2 high */
volatile unsigned char CCR2L; /*!< capture/compare register 2 low */
volatile unsigned char CCR3H; /*!< capture/compare register 3 high */
volatile unsigned char CCR3L; /*!< capture/compare register 3 low */
volatile unsigned char CCR4H; /*!< capture/compare register 3 high */
volatile unsigned char CCR4L; /*!< capture/compare register 3 low */
volatile unsigned char BKR; /*!< Break Register */
volatile unsigned char DTR; /*!< dead-time register */
volatile unsigned char OISR; /*!< Output idle register */
}TIM1_TypeDef;

```

```

#define TIM1_BaseAddress 0xFEC0
#define TIM2_BaseAddress 0xFEE0
#define TIM1 ((TIM1_TypeDef xdata*) TIM1_BaseAddress)
#define TIM2 ((TIM1_TypeDef xdata*) TIM2_BaseAddress)
#define PWMA_ETRPS (*(unsigned char volatile xdata *) 0xFEB0)
#define PWMA_ENO (*(unsigned char volatile xdata *) 0xFEB1)
#define PWMA_PS (*(unsigned char volatile xdata *) 0xFEB2)
#define PWMB_ENO (*(unsigned char volatile xdata *) 0xFEB5)
#define PWMB_PS (*(unsigned char volatile xdata *) 0xFEB6)

```

```

sfr ADC_CONTR = 0xbc;
sfr ADC_RES = 0xbd;
sfr ADC_RESL = 0xbe;
sfr ADCCFG = 0xde;
sfr CMPCR1 = 0xe6;
sfr CMPCR2 = 0xe7;
sfr AUXR = 0x8e;
sfr P0M0 = 0x94;
sfr P0M1 = 0x93;
sfr P1M0 = 0x92;
sfr P1M1 = 0x91;
sfr P2M0 = 0x96;
sfr P2M1 = 0x95;
sfr P3M0 = 0xb2;
sfr P3M1 = 0xb1;
sfr P5M0 = 0xca;
sfr P5M1 = 0xc9;
sfr P5 = 0xc8;
sfr P_SW2 = 0xba;

```

```

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    P50      =    P5^0;
sbit    P51      =    P5^1;
sbit    P52      =    P5^2;
sbit    P53      =    P5^3;
sbit    P54      =    P5^4;
sbit    P55      =    P5^5;

#define    TRUE          1
#define    FALSE         0

#define    RV09_CH        6

#define    TIM1_Period    ((u16)280)
#define    TIM1_STPulse   ((u16)245)

#define    START          0x1A
#define    RUN            0x1B
#define    STOP           0x1C
#define    IDLE           0x1D

#define    TIM1_OCMODE_MASK    ((u8)0x70)
#define    TIM1_OCCE_ENABLE    ((u8)0x80)
#define    TIM1_OCCE_DISABLE    ((u8)0x00)
#define    TIM1_OCMODE_TIMING    ((u8)0x00)
#define    TIM1_OCMODE_ACTIVE    ((u8)0x10)
#define    TIM1_OCMODE_INACTIVE    ((u8)0x20)
#define    TIM1_OCMODE_TOGGLE    ((u8)0x30)
#define    TIM1_FORCE_INACTIVE    ((u8)0x40)
#define    TIM1_FORCE_ACTIVE    ((u8)0x50)
#define    TIM1_OCMODE_PWMA    ((u8)0x60)
#define    TIM1_OCMODE_PWMB    ((u8)0x70)
#define    CC1_POLARITY_HIGH    ((u8)0x02)
#define    CC1N_POLARITY_HIGH    ((u8)0x08)
#define    CC2_POLARITY_HIGH    ((u8)0x20)
#define    CC2N_POLARITY_HIGH    ((u8)0x80)
#define    CC1_POLARITY_LOW    ((u8)~0x02)
#define    CC1N_POLARITY_LOW    ((u8)~0x08)
#define    CC2_POLARITY_LOW    ((u8)~0x20)
#define    CC2N_POLARITY_LOW    ((u8)~0x80)
#define    CC1_OCENABLE    ((u8)0x01)
#define    CC1N_OCENABLE    ((u8)0x04)

```

```

#define CC2_OCENABLE          ((u8)0x10)
#define CC2N_OCENABLE         ((u8)0x40)
#define CC1_OCDISABLE         ((u8)~0x01)
#define CC1N_OCDISABLE        ((u8)~0x04)
#define CC2_OCDISABLE         ((u8)~0x10)
#define CC2N_OCDISABLE        ((u8)~0x40)
#define CC3_POLARITY_HIGH     ((u8)0x02)
#define CC3N_POLARITY_HIGH    ((u8)0x08)
#define CC4_POLARITY_HIGH     ((u8)0x20)
#define CC4N_POLARITY_HIGH    ((u8)0x80)
#define CC3_POLARITY_LOW      ((u8)~0x02)
#define CC3N_POLARITY_LOW     ((u8)~0x08)
#define CC4_POLARITY_LOW      ((u8)~0x20)
#define CC4N_POLARITY_LOW     ((u8)~0x80)
#define CC3_OCENABLE          ((u8)0x01)
#define CC3N_OCENABLE         ((u8)0x04)
#define CC4_OCENABLE          ((u8)0x10)
#define CC4N_OCENABLE         ((u8)0x40)
#define CC3_OCDISABLE         ((u8)~0x01)
#define CC3N_OCDISABLE        ((u8)~0x04)
#define CC4_OCDISABLE         ((u8)~0x10)
#define CC4N_OCDISABLE        ((u8)~0x40)

```

```

void UART_INIT();
void DelayXus(unsigned char delayTime);
void DelayXms( unsigned char delayTime);
unsigned int ADC_Convert(u8 ch);
void PWM_Init(void);
void SPEED_ADJ();
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP();
unsigned char KEY_detect();

```

```

unsigned char Timer0_cnt=0xb0;
unsigned int HA=0;
unsigned int Motor_speed;
unsigned char Motor_sta = IDLE;
unsigned char BRK_occur=0;
unsigned int TIM2_CAP1_v=0;
unsigned int CAP1_avg=0;
unsigned char CAP1_cnt=0;
unsigned long CAP1_sum=0;

```

```

void main(void)
{
    unsigned int temp=0;
    unsigned int ADC_result=0;

    P_SW2= 0x80;
    P1 = 0x00;
    P0M1 = 0x0C;
    P0M0 = 0x01;
    P1M1 = 0xc0;
    P1M0 = 0x3F;
    P2M1 = 0x00;
    P2M0 = 0x38;
    P3M1 = 0x88;
    P3M0 = 0x02;

```



```

ET0=1;
TR0=0;
ADCCFG = 0x0f;
ADC_CONTR = 0x80;
PWMA_ENO = 0x3F;           //PWMA output enable
PWMB_ENO = 0x00;           //PWMB output enable
PWMA_PS = 0x00;            //PWMA pin selection
PWMB_PS = 0xD5;            //PWMB pin selection
/*****
output compare mode PWMx_duty = [CCRx/(ARR + 1)]*100
*****/
/*****PWMB BMF input *****/
//////// time base unit //////////
TIM2-> PSCRL = 15;
TIM2-> ARR = 0xff;          // Auto-reload register, counter overflow point
TIM2-> ARRL = 0xff;
TIM2-> CCR4H = 0x00;
TIM2-> CCR4L = 0x05;
//////// Channel configuration //////////
TIM2-> CCMR1 = 0xf3;        //Channel Mode Configuration
TIM2-> CCMR2 = 0xf1;
TIM2-> CCMR3 = 0xf1;
TIM2-> CCMR4 = 0x70;
TIM2-> CCER1 = 0x11;
TIM2-> CCER2 = 0x11;
//////// Mode Configuration //////////
TIM2-> CR2 = 0xf0;
TIM2-> CR1 = 0x81;
TIM2-> SMCR = 0x44;
//////// Enable & Interrupt Configuration //////////
TIM2-> BKR = 0x80;          //enable output
TIM2-> IER = 0x02;          //enable interrupt
/*****PWMA Controls motor to change phase *****/
//////// time base unit //////////
TIM1-> PSCRH = 0x00;        //Prescaler register
TIM1-> PSCRL = 0x00;
TIM1-> ARR = (u8)(TIM1_Period >> 8);
TIM1-> ARRL = (u8)(TIM1_Period);
//////// Channel configuration //////////
TIM1-> CCMR1 = 0x70;        //Channel Mode Configuration
TIM1-> CCMR2 = 0x70;
TIM1-> CCMR3 = 0x70;
TIM1-> CCER1 = 0x11;        //Config channel output enable and polarity
TIM1-> CCER2 = 0x01;        //Config channel output enable and polarity
TIM1-> OISR = 0xAA;         //Configure the output level of each channel when MOE=0
//////// Mode Configuration //////////
TIM1-> CR1 = 0xA0;
TIM1-> CR2 = 0x24;
TIM1-> SMCR = 0x20;
TIM1-> BKR = 0x0c;
//////// Enable & Interrupt Configuration //////////
TIM1-> CR1 |= 0x01;         //enable counter
EA = 1;

UART_INIT();

while (1)
{
    switch(Motor_sta)
    {
        case START:
            MOTOR_START();
            Motor_sta = RUN;
            for(temp = TIM1_STPulse; temp > ADC_result; temp--) // Open loop start
            {
                ADC_result = (ADC_Convert(RV09_CH)/4);
            }
        }
    }
}

```

```

        TIM1->CCR1H = (u8)(temp >> 8);
        TIM1->CCR1L = (u8)(temp);
        TIM1->CCR2H = (u8)(temp >> 8);
        TIM1->CCR2L = (u8)(temp);
        TIM1->CCR3H = (u8)(temp >> 8);
        TIM1->CCR3L = (u8)(temp);
        DelayXms(10);
    }
    break;
case RUN:
    SPEED_ADJ();           // Motor speed adjustment
    if((BRK_occur == TRUE))
        Motor_sta = STOP;
    break;
case STOP:
    MOTOR_STOP();
    Motor_sta = IDLE;
    break;
case IDLE:
    if(KEY_detect()==1)
        Motor_sta = START;           //Start motor
    BRK_occur = FALSE;
    Motor_speed = 0;
    CAP1_avg = 0;
    CAP1_cnt = 0;
    CAP1_sum = 0;
    break;
}
}
}
void TIM0_ISR() interrupt 1
{
    if(Motor_sta == START)
    {
        if(Timer0_cnt<0xe0) Timer0_cnt++;
        TH0=Timer0_cnt;

        switch(HA%6)
        {
            case 0:
                TIM1->CCMR3 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR3 |= TIM1_FORCE_INACTIVE;
                TIM1->CCMR1 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR1 |= TIM1_OCMODE_PWMB;
                break;
            case 1:
                TIM1->CCER1 &= CC2N_POLARITY_LOW;
                TIM1->CCER2 |= CC3N_POLARITY_HIGH;
                break;
            case 2:
                TIM1->CCMR1 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR1 |= TIM1_FORCE_INACTIVE;
                TIM1->CCMR2 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR2 |= TIM1_OCMODE_PWMB;
                break;
            case 3:
                TIM1->CCER1 |= CC1N_POLARITY_HIGH;
                TIM1->CCER2 &= CC3N_POLARITY_LOW;
                break;
            case 4:
                TIM1->CCMR2 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR2 |= TIM1_FORCE_INACTIVE;
                TIM1->CCMR3 &= ~TIM1_OCMODE_MASK;
                TIM1->CCMR3 |= TIM1_OCMODE_PWMB;
                break;
            case 5:
                TIM1->CCER1 &= CC1N_POLARITY_LOW;

```

```

        TIM1-> CCER1 |= CC2N_POLARITY_HIGH;
        break;
    }
    HA++;
}
if(Motor_sta == RUN)
{
    TR0=0;
    switch(RD_HALL())
    {
        case 3:
            TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
            TIM1-> CCMR3 |= TIM1_FORCE_INACTIVE;
            TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
            TIM1-> CCMR1 |= TIM1_OCMODE_PWMB;
            break;
        case 1:
            TIM1-> CCER1 &= CC2N_POLARITY_LOW;
            TIM1-> CCER2 |= CC3N_POLARITY_HIGH;
            break;
        case 5:
            TIM1-> CCMR1 &= ~TIM1_OCMODE_MASK;
            TIM1-> CCMR1 |= TIM1_FORCE_INACTIVE;
            TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
            TIM1-> CCMR2 |= TIM1_OCMODE_PWMB;
            break;
        case 4:
            TIM1-> CCER1 |= CC1N_POLARITY_HIGH;
            TIM1-> CCER2 &= CC3N_POLARITY_LOW;
            break;
        case 6:
            TIM1-> CCMR2 &= ~TIM1_OCMODE_MASK;
            TIM1-> CCMR2 |= TIM1_FORCE_INACTIVE;
            TIM1-> CCMR3 &= ~TIM1_OCMODE_MASK;
            TIM1-> CCMR3 |= TIM1_OCMODE_PWMB;
            break;
        case 2:
            TIM1-> CCER1 &= CC1N_POLARITY_LOW;
            TIM1-> CCER1 |= CC2N_POLARITY_HIGH;
            break;
    }
}
}
void PWMA_ISR() interrupt 26
{
    if((TIM1->SR1 & 0x20))
    {
        P00=0;
        CAP1_sum += TIM2_CAP1_v;
        CAP1_cnt++;
        if(CAP1_cnt==128)
        {
            CAP1_cnt=0;
            CAP1_avg = (CAP1_sum>>7);
            CAP1_sum = 0;
            Motor_speed = 5000000/CAP1_avg;
        }
        TIM1->SR1 &= ~0x20; //clear
    }
    if((TIM1->SR1 & 0x80)) //BRK
    {
        BRK_occur = TRUE;
        TIM1->SR1 &= ~0x80; //clear
    }
}

void PWMB_ISR() interrupt 27

```

```

{
    unsigned char ccr_tmp=0;

    if((TIM2->SR1 & 0X02))
    {
        ccr_tmp = TIM2->CCR1H;
        if(ccr_tmp>1) // software filtering
        {
            TIM2_CAP1_v = ccr_tmp;
            TIM2_CAP1_v = (TIM2_CAP1_v<<8) + TIM2->CCR1L;
            if(Motor_sta == RUN) //phase change delay timing
            {
                TR0=1;
                TH0 = 256-(TIM2_CAP1_v>>9);
            }
        }
        TIM2->SR1 &=~0X02;
    }
}

void UART_INIT()
{
    SCON = 0x50; //8-bit, variable baud rate
    AUXR = 0x40; // Timer 1 is in 1T mode
    TMOD = 0x20; // Timer 1 works in Mode 0 (16-bit auto-reload)
    TL1 = 254;
    TH1 = 254;
    // ET1 = 0;
    TR1 = 1;
}

void DelayXus(unsigned char delayTime)
{
    int i = 0;
    while( delayTime--)
    {
        for( i = 0 ; i < 1 ; i++);
    }
}

void DelayXms( unsigned char delayTime )
{
    int i = 0;
    while( delayTime--)
    {
        for( i = 0 ; i < 2 ; i++)
        {
            DelayXus(100);
        }
    }
}

unsigned int ADC_Convert(u8 ch)
{
    u16 res=0;

    ADC_CONTR &= ~0x0f;
    ADC_CONTR |= ch;
    ADC_CONTR |= 0x40;
    DelayXus(1);
    while (!(ADC_CONTR & 0x20));
    ADC_CONTR &= ~0x20;

    res = ADC_RES;
    res = (res<<2)+(ADC_RES>>6);

    if (res < 360) res=360;
    if (res > 900) res=900;
    return res;
}

void SPEED_ADJ()

```

```

{
    u16 ADC_result;

    ADC_result = (ADC_Convert(RV09_CH)/4);           // Speed knob ADC sampling
    TIM1->CCR1H = (u8)(ADC_result >> 8);             //Counter comparison value
    TIM1->CCR1L = (u8)(ADC_result);
    TIM1->CCR2H = (u8)(ADC_result >> 8);
    TIM1->CCR2L = (u8)(ADC_result);
    TIM1->CCR3H = (u8)(ADC_result >> 8);
    TIM1->CCR3L = (u8)(ADC_result);
}

unsigned char RD_HALL()                             // read hall sensor
{
    unsigned char Hall_sta = 0;

    DelayXus(40);
    (P17)? (Hall_sta|=0x01) : (Hall_sta&=~0x01);
    (P54)? (Hall_sta|=0x02) : (Hall_sta&=~0x02);
    (P33)? (Hall_sta|=0x04) : (Hall_sta&=~0x04);

    return Hall_sta;
}

void MOTOR_START()
{
    TIM1->CCR1H = (u8)(TIM1_STPulse >> 8);           //Counter comparison value
    TIM1->CCR1L = (u8)(TIM1_STPulse);
    TIM1->CCR2H = (u8)(TIM1_STPulse >> 8);
    TIM1->CCR2L = (u8)(TIM1_STPulse);
    TIM1->CCR3H = (u8)(TIM1_STPulse >> 8);
    TIM1->CCR3L = (u8)(TIM1_STPulse);
    TIM1->BKR |= 0x80;                               //enable main output, equivalent to a master switch
    TIM1->IER = 0x00;                                //enable interrupt
    TR0 = 1;

    while (HA < 6*20);

    TIM1->IER = 0xa0;                                //enable interrupt
}

void MOTOR_STOP()
{
    TIM1->BKR &= ~0x80;
    TIM1->IER &= ~0x20;
}

unsigned char KEY_detect()
{
    if(!P37)
    {
        DelayXms(10);
        if(!P37)
        {
            return 1;
        }
        else return 0;
    }
    else if(!P03)
    {
        DelayXms(10);
        if(!P03)
        {
            return 2;
        }
        else return 0;
    }
    else return 0;
}

```

20.8.3 Quadrature encoder mode

C language code

//Operating frequency for test is 11.0592MHz

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

```
typedef struct TIM1_struct
{
```

```
volatile unsigned char CR1;
```

```
    volatile unsigned char CR2;
```

```
    volatile unsigned char SMCR;
```

```
    volatile unsigned char ETR;
```

```
    volatile unsigned char IER;
```

```
    volatile unsigned char SR1;
```

```
    volatile unsigned char SR2;
```

```
    volatile unsigned char EGR;
```

```
    volatile unsigned char CCMR1;
```

```
    volatile unsigned char CCMR2;
```

```
    volatile unsigned char CCMR3;
```

```
    volatile unsigned char CCMR4;
```

```
    volatile unsigned char CCER1;
```

```
    volatile unsigned char CCER2;
```

```
    volatile unsigned char CNTRH;
```

```
    volatile unsigned char CNTRL;
```

```
    volatile unsigned char PSCRH;
```

```
    volatile unsigned char PSCRL;
```

```
    volatile unsigned char ARRH;
```

```
    volatile unsigned char ARRL;
```

```
    volatile unsigned char RCR;
```

```
    volatile unsigned char CCR1H;
```

```
    volatile unsigned char CCR1L;
```

```
    volatile unsigned char CCR2H;
```

```
    volatile unsigned char CCR2L;
```

```
    volatile unsigned char CCR3H;
```

```
    volatile unsigned char CCR3L;
```

```
    volatile unsigned char CCR4H;
```

```
    volatile unsigned char CCR4L;
```

```
    volatile unsigned char BKR;
```

```
    volatile unsigned char DTR;
```

```
    volatile unsigned char OISR;
```

```
}TIM1_TypeDef;
```

```
#define TIM1_BaseAddress 0xFEC0
```

```
#define TIM1 ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
```

```
#define PWMA_ENO (*(unsigned char volatile xdata *)0xFEB1)
```

```
#define PWMA_PS (*(unsigned char volatile xdata *)0xFEB2)
```

```
sfr P0M0 = 0x94;
```

```
sfr P0M1 = 0x93;
```

```
sfr P1M0 = 0x92;
```

```
sfr P1M1 = 0x91;
```

```
/*!< control register 1 */
```

```
/*!< control register 2 */
```

```
/*!< Synchro mode control register */
```

```
/*!< external trigger register */
```

```
/*!< interrupt enable register*/
```

```
/*!< status register 1 */
```

```
/*!< status register 2 */
```

```
/*!< event generation register */
```

```
/*!< CC mode register 1 */
```

```
/*!< CC mode register 2 */
```

```
/*!< CC mode register 3 */
```

```
/*!< CC mode register 4 */
```

```
/*!< CC enable register 1 */
```

```
/*!< CC enable register 2 */
```

```
/*!< counter high */
```

```
/*!< counter low */
```

```
/*!< prescaler high */
```

```
/*!< prescaler low */
```

```
/*!< auto-reload register high */
```

```
/*!< auto-reload register low */
```

```
/*!< Repetition Counter register */
```

```
/*!< capture/compare register 1 high */
```

```
/*!< capture/compare register 1 low */
```

```
/*!< capture/compare register 2 high */
```

```
/*!< capture/compare register 2 low */
```

```
/*!< capture/compare register 3 high */
```

```
/*!< capture/compare register 3 low */
```

```
/*!< capture/compare register 3 high */
```

```
/*!< capture/compare register 3 low */
```

```
/*!< Break Register */
```

```
/*!< dead-time register */
```

```
/*!< Output idle register */
```

```

sfr      P_SW2      = 0xba;

sbit     P03        = P0^3;

unsigned char cnt_H, cnt_L;

void main(void)
{
    P_SW2 = 0x80;

    PIM1 = 0x0f;
    PIM0 = 0x00;

    PWMA_ENO = 0x00; // The pin configured as TRGI needs to turn off the
corresponding bit of ENO and configure it as input //00:PWM at P1
    PWMA_PS = 0x00; // Prescaler register

    TIM1->PSCRH = 0x00; // Prescaler register
    TIM1->PSCRL = 0x00;

    TIM1->CCMR1 = 0x21; // Channel mode is configured as input, connected to encoder,
filter 4 clocks
    TIM1->CCMR2 = 0x21; // Channel mode is configured as input, connected to encoder,
filter 4 clocks

    TIM1->SMCR = 0x03; // encoder mode 3

    TIM1->CCER1 = 0x55; // Configure Channel Enable and Polarity
    TIM1->CCER2 = 0x55; // Configure Channel Enable and Polarity

    TIM1->IER = 0x02; // enable interrupt

    TIM1->CR1 |= 0x01; // enable counter

    EA = 1;

    while (1);
}

/***** PWM interrupt to read encoder count value *****/
void PWMA_ISR() interrupt 26
{
    if (TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        cnt_H = TIM1->CCR1H;
        cnt_L = TIM1->CCR1L;
        TIM1->SR1 &= ~0X02;
    }
}

```

20.8.4 Single pulse mode (trigger control pulse output)

C language code

//Operating frequency for test is 11.0592MHz

```

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

typedef struct TIM1_struct
{
    volatile unsigned char CR1;          /*!< control register 1 */
    volatile unsigned char CR2;          /*!< control register 2 */
    volatile unsigned char SMCR;         /*!< Synchro mode control register */
    volatile unsigned char ETR;          /*!< external trigger register */
    volatile unsigned char IER;          /*!< interrupt enable register*/
    volatile unsigned char SR1;          /*!< status register 1 */
    volatile unsigned char SR2;          /*!< status register 2 */
    volatile unsigned char EGR;          /*!< event generation register */
    volatile unsigned char CCMR1;        /*!< CC mode register 1 */
    volatile unsigned char CCMR2;        /*!< CC mode register 2 */
    volatile unsigned char CCMR3;        /*!< CC mode register 3 */
    volatile unsigned char CCMR4;        /*!< CC mode register 4 */
    volatile unsigned char CCER1;        /*!< CC enable register 1 */
    volatile unsigned char CCER2;        /*!< CC enable register 2 */
    volatile unsigned char CNTRH;        /*!< counter high */
    volatile unsigned char CNTRL;        /*!< counter low */
    volatile unsigned char PSCRH;        /*!< prescaler high */
    volatile unsigned char PSCRL;        /*!< prescaler low */
    volatile unsigned char ARRH;         /*!< auto-reload register high */
    volatile unsigned char ARRL;         /*!< auto-reload register low */
    volatile unsigned char RCR;          /*!< Repetition Counter register */
    volatile unsigned char CCR1H;        /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;        /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;        /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;        /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;        /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;        /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;        /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;        /*!< capture/compare register 3 low */
    volatile unsigned char BKR;          /*!< Break Register */
    volatile unsigned char DTR;          /*!< dead-time register */
    volatile unsigned char OISR;         /*!< Output idle register */
}TIM1_TypeDef;

#define TIM1_BaseAddress 0xFEC0

#define TIM1 ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define PWMA_ENO (*(unsigned char volatile xdata *)0xFEB1)
#define PWMA_PS (*(unsigned char volatile xdata *)0xFEB2)

sfr P0M0 = 0x94;
sfr P0M1 = 0x93;
sfr P1M0 = 0x92;
sfr P1M1 = 0x91;
sfr P_SW2 = 0xba;

sbit P03 = P0^3;

void main(void)
{
    P_SW2 = 0x80;

```



```

P0M1 = 0x00;
P0M0 = 0xFF;
P1M1 = 0x0c;
P1M0 = 0xF3;

PWMA_ENO = 0xF3;           //IO outputs PWM
PWMA_PS = 0x00;           //00:PWM at P1

/*****
PWMx_duty = [CCRx/(ARR + 1)]*100
*****/

// The pin configured as TRGI needs to turn off the corresponding bit of ENO and configure it as input
TIM1->PSCRH = 0x00;        // Prescaler register
TIM1->PSCRL = 0x00;
TIM1->DTR = 0x00;          // Dead time configuration

TIM1->CCMR1 = 0x68;        // Channel Mode Configuration
TIM1->CCMR2 = 0x01;        // Configured as an input channel
TIM1->CCMR3 = 0x68;
TIM1->CCMR4 = 0x68;

TIM1->SMCR = 0x66;

TIM1->ARRH = 0x08;         // Auto-reload register, counter overflow point
TIM1->ARRL = 0x00;

TIM1->CCR1H = 0x04;        // Counter comparison value
TIM1->CCR1L = 0x00;
TIM1->CCR2H = 0x02;
TIM1->CCR2L = 0x00;
TIM1->CCR3H = 0x01;
TIM1->CCR3L = 0x00;
TIM1->CCR4H = 0x01;
TIM1->CCR4L = 0x00;

TIM1->CCER1 = 0x55;        // Configure Channel Output Enable and Polarity
TIM1->CCER2 = 0x55;        // Configure Channel Output Enable and Polarity

TIM1->BKR = 0x80;         // Main output enable, equivalent to main switch
TIM1->IER = 0x02;         // enable interrupt
TIM1->CR1 = 0x08;         // Single pulse mode
TIM1->CR1 |= 0x01;        // enable counter

EA = 1;
while (1);
}

void PWMA_ISR() interrupt 26
{
    if (TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &= ~0X02;
    }
}

```

20.8.5 Gating mode (input level enable counter)

C language code

//Operating frequency for test is 11.0592MHz

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

```
typedef struct TIM1_struct
```

```
{
    volatile unsigned char CR1;          /*!< control register 1 */
    volatile unsigned char CR2;          /*!< control register 2 */
    volatile unsigned char SMCR;         /*!< Synchro mode control register */
    volatile unsigned char ETR;          /*!< external trigger register */
    volatile unsigned char IER;          /*!< interrupt enable register*/
    volatile unsigned char SR1;          /*!< status register 1 */
    volatile unsigned char SR2;          /*!< status register 2 */
    volatile unsigned char EGR;          /*!< event generation register */
    volatile unsigned char CCMR1;        /*!< CC mode register 1 */
    volatile unsigned char CCMR2;        /*!< CC mode register 2 */
    volatile unsigned char CCMR3;        /*!< CC mode register 3 */
    volatile unsigned char CCMR4;        /*!< CC mode register 4 */
    volatile unsigned char CCER1;        /*!< CC enable register 1 */
    volatile unsigned char CCER2;        /*!< CC enable register 2 */
    volatile unsigned char CNTRH;        /*!< counter high */
    volatile unsigned char CNTRL;        /*!< counter low */
    volatile unsigned char PSCRH;        /*!< prescaler high */
    volatile unsigned char PSCRL;        /*!< prescaler low */
    volatile unsigned char ARRH;         /*!< auto-reload register high */
    volatile unsigned char ARRL;         /*!< auto-reload register low */
    volatile unsigned char RCR;          /*!< Repetition Counter register */
    volatile unsigned char CCR1H;        /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;        /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;        /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;        /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;        /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;        /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;        /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;        /*!< capture/compare register 3 low */
    volatile unsigned char BKR;          /*!< Break Register */
    volatile unsigned char DTR;          /*!< dead-time register */
    volatile unsigned char OISR;         /*!< Output idle register */
}
```

```
}TIM1_TypeDef;
```

```
#define TIM1_BaseAddress 0xFEC0
```

```
#define TIM1 ((TIM1_TypeDef)xdata*)TIM1_BaseAddress)
```

```
#define PWMA_ENO (*(unsigned char volatile xdata *)0xFEB1)
```

```
#define PWMA_PS (*(unsigned char volatile xdata *)0xFEB2)
```

```
sfr P0M0 = 0x94;
sfr P0M1 = 0x93;
sfr P1M0 = 0x92;
sfr P1M1 = 0x91;
sfr P3M0 = 0xb2;
sfr P3M1 = 0xb1;
```

```

sfr      P_SW2      = 0xba;

sbit     P03        = P0^3;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;
    P3M1 = 0x04;
    P3M0 = 0x00;

    PWMA_ENO = 0xFF;           //IO outputs PWM
    PWMA_PS = 0x00;           //00:PWM at P1

    /*****
    PWMx_duty = [CCRx/(ARR + 1)]*100
    *****/

    // The pin configured as TRGI needs to turn off the corresponding bit of ENO and configure it as input
    TIM1->PSCRH = 0x00;        // Prescaler register
    TIM1->PSCRL = 0x00;
    TIM1->DTR = 0x00;          // Dead time configuration

    TIM1->CCMR1 = 0x68;         // Channel Mode Configuration
    TIM1->CCMR2 = 0x68;         // Configured as an input channel
    TIM1->CCMR3 = 0x68;
    TIM1->CCMR4 = 0x68;

    TIM1->SMCR = 0x75;          // Gated trigger mode ETRF input

    TIM1->ARRH = 0x08;          // Auto-reload register, counter overflow point
    TIM1->ARRL = 0x00;

    TIM1->CCR1H = 0x04;         // Counter comparison value
    TIM1->CCR1L = 0x00;         //
    TIM1->CCR2H = 0x02;         //
    TIM1->CCR2L = 0x00;         //
    TIM1->CCR3H = 0x01;         //
    TIM1->CCR3L = 0x00;         //
    TIM1->CCR4H = 0x01;         //
    TIM1->CCR4L = 0x00;         //

    TIM1->CCER1 = 0x55;         // Configure Channel Output Enable and Polarity
    TIM1->CCER2 = 0x55;         // Configure Channel Output Enable and Polarity

    TIM1->BKR = 0x80;           // Main output enable, equivalent to main switch
    TIM1->IER = 0x02;           // enable interrupt

    TIM1->CR1 |= 0x01;          // enable counter

    EA = 1;
    while (1) ;
}

void PWMA_ISR() interrupt 26

```

```

{
    if(TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &=~0X02;
    }
}

```

20.8.6 External clock mode

C language code

//Operating frequency for test is 11.0592MHz

```

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

```

```
typedef struct TIM1_struct
```

```

{
    volatile unsigned char CR1;          /*!< control register 1 */
    volatile unsigned char CR2;          /*!< control register 2 */
    volatile unsigned char SMCR;         /*!< Synchro mode control register */
    volatile unsigned char ETR;          /*!< external trigger register */
    volatile unsigned char IER;          /*!< interrupt enable register*/
    volatile unsigned char SR1;          /*!< status register 1 */
    volatile unsigned char SR2;          /*!< status register 2 */
    volatile unsigned char EGR;          /*!< event generation register */
    volatile unsigned char CCMR1;        /*!< CC mode register 1 */
    volatile unsigned char CCMR2;        /*!< CC mode register 2 */
    volatile unsigned char CCMR3;        /*!< CC mode register 3 */
    volatile unsigned char CCMR4;        /*!< CC mode register 4 */
    volatile unsigned char CCER1;        /*!< CC enable register 1 */
    volatile unsigned char CCER2;        /*!< CC enable register 2 */
    volatile unsigned char CNTRH;        /*!< counter high */
    volatile unsigned char CNTRL;        /*!< counter low */
    volatile unsigned char PSCRH;        /*!< prescaler high */
    volatile unsigned char PSCLR;        /*!< prescaler low */
    volatile unsigned char ARRH;         /*!< auto-reload register high */
    volatile unsigned char ARRL;         /*!< auto-reload register low */
    volatile unsigned char RCR;          /*!< Repetition Counter register */
    volatile unsigned char CCR1H;        /*!< capture/compare register 1 high */
    volatile unsigned char CCR1L;        /*!< capture/compare register 1 low */
    volatile unsigned char CCR2H;        /*!< capture/compare register 2 high */
    volatile unsigned char CCR2L;        /*!< capture/compare register 2 low */
    volatile unsigned char CCR3H;        /*!< capture/compare register 3 high */
    volatile unsigned char CCR3L;        /*!< capture/compare register 3 low */
    volatile unsigned char CCR4H;        /*!< capture/compare register 3 high */
    volatile unsigned char CCR4L;        /*!< capture/compare register 3 low */
    volatile unsigned char BKR;          /*!< Break Register */
    volatile unsigned char DTR;          /*!< dead-time register */
    volatile unsigned char OISR;         /*!< Output idle register */
}TIM1_TypeDef;

```

```
#define TIM1_BaseAddress 0xFEC0
```

```

#define TIM1 ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define PWMA_ENO (*(unsigned char volatile xdata *)0xFEB1)
#define PWMA_PS (*(unsigned char volatile xdata *)0xFEB2)

sfr P0M0 = 0x94;
sfr P0M1 = 0x93;
sfr P1M0 = 0x92;
sfr P1M1 = 0x91;
sfr P3M0 = 0xb2;
sfr P3M1 = 0xb1;
sfr P_SW2 = 0xba;

sbit P03 = P0^3;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;
    P3M1 = 0x04;
    P3M0 = 0x00;

    PWMA_ENO = 0xFF; //IO outputs PWM
    PWMA_PS = 0x00; //00:PWM at P1

    /**
    PWMx_duty = [CCRx/(ARR + 1)]*100
    */
    // The pin configured as TRGI needs to turn off the corresponding bit of ENO and configure it as input
    TIM1->PSCRH = 0x00; // Prescaler register
    TIM1->PSCRL = 0x00;
    TIM1->DTR = 0x00; // Dead time configuration

    TIM1->CCMR1 = 0x68; // Channel Mode Configuration
    TIM1->CCMR2 = 0x68; // Configured as an input channel
    TIM1->CCMR3 = 0x68;
    TIM1->CCMR4 = 0x68;

    TIM1->SMCR = 0x77; //ETRF input

    TIM1->ARRH = 0x08; // Auto-reload register, counter overflow point
    TIM1->ARRL = 0x00;

    TIM1->CCR1H = 0x04; // Counter comparison value
    TIM1->CCR1L = 0x00;
    TIM1->CCR2H = 0x02;
    TIM1->CCR2L = 0x00;
    TIM1->CCR3H = 0x01;
    TIM1->CCR3L = 0x00;
    TIM1->CCR4H = 0x01;
    TIM1->CCR4L = 0x00;

    TIM1->CCER1 = 0x55; // Configure Channel Output Enable and Polarity
    TIM1->CCER2 = 0x55; // Configure Channel Output Enable and Polarity

```

```

    TIM1->BKR = 0x80;           // enable main output, equivalent to main switch
    TIM1->IER = 0x02;           // enable interrupt
    TIM1->CR1 |= 0x01;          // enable counter

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &= ~0X02;
    }
}

```

20.8.7 Input capture mode to measure the pulse period (capture rising edge to rising edge or falling edge to falling edge)

C language code

//Operating frequency for test is 11.0592MHz

```

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

```

```

typedef struct TIM1_struct
{

```

volatile unsigned char CR1;	/*!< control register 1 */
volatile unsigned char CR2;	/*!< control register 2 */
volatile unsigned char SMCR;	/*!< Synchro mode control register */
volatile unsigned char ETR;	/*!< external trigger register */
volatile unsigned char IER;	/*!< interrupt enable register*/
volatile unsigned char SR1;	/*!< status register 1 */
volatile unsigned char SR2;	/*!< status register 2 */
volatile unsigned char EGR;	/*!< event generation register */
volatile unsigned char CCMR1;	/*!< CC mode register 1 */
volatile unsigned char CCMR2;	/*!< CC mode register 2 */
volatile unsigned char CCMR3;	/*!< CC mode register 3 */
volatile unsigned char CCMR4;	/*!< CC mode register 4 */
volatile unsigned char CCER1;	/*!< CC enable register 1 */
volatile unsigned char CCER2;	/*!< CC enable register 2 */
volatile unsigned char CNTRH;	/*!< counter high */
volatile unsigned char CNTRL;	/*!< counter low */
volatile unsigned char PSCRH;	/*!< prescaler high */
volatile unsigned char PSCRL;	/*!< prescaler low */
volatile unsigned char ARRH;	/*!< auto-reload register high */
volatile unsigned char ARRL;	/*!< auto-reload register low */
volatile unsigned char RCR;	/*!< Repetition Counter register */
volatile unsigned char CCR1H;	/*!< capture/compare register 1 high */
volatile unsigned char CCR1L;	/*!< capture/compare register 1 low */
volatile unsigned char CCR2H;	/*!< capture/compare register 2 high */
volatile unsigned char CCR2L;	/*!< capture/compare register 2 low */

```

volatile unsigned char CCR3H;          /*!< capture/compare register 3 high */
volatile unsigned char CCR3L;          /*!< capture/compare register 3 low */
volatile unsigned char CCR4H;          /*!< capture/compare register 3 high */
volatile unsigned char CCR4L;          /*!< capture/compare register 3 low */
volatile unsigned char BKR;            /*!< Break Register */
volatile unsigned char DTR;            /*!< dead-time register */
volatile unsigned char OISR;           /*!< Output idle register */
}TIM1_TypeDef;

#define TIM1_BaseAddress 0xFEC0

#define TIM1 ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define PWMA_ENO (*(unsigned char volatile xdata *)0xFEB1)
#define PWMA_PS (*(unsigned char volatile xdata *)0xFEB2)

sfr P0M0 = 0x94;
sfr P0M1 = 0x93;
sfr P1M0 = 0x92;
sfr P1M1 = 0x91;
sfr P3M0 = 0xb2;
sfr P3M1 = 0xb1;
sfr P_SW2 = 0xba;

sbit P03 = P0^3;

int cap;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x0c;
    P1M0 = 0xF3;

    PWMA_ENO = 0xF3;          //IO outputs PWM
    PWMA_PS = 0x00;          //00:PWM at P1

    /* The pin configured as TRGI needs to turn off the corresponding bit of ENO and configure it as input */
    TIM1->PSCRH = 0x00;      // Prescaler register
    TIM1->PSCRL = 0x00;
    TIM1->DTR = 0x00;        // Dead time configuration

    TIM1->CCMR1 = 0x68;      // Channel Mode Configuration
    TIM1->CCMR2 = 0x01;      // Configured as an input channel
    TIM1->CCMR3 = 0x68;
    TIM1->CCMR4 = 0x68;

    TIM1->SMCR = 0x66;

    TIM1->CCER1 = 0x55;      // Configure Channel Output Enable and Polarity
    TIM1->CCER2 = 0x55;      // Configure Channel Output Enable and Polarity

    TIM1->IER = 0x04;        // enable interrupt

    TIM1->CR1 |= 0x01;        // enable counter

```

```

    EA = 1;
    while (1);
}

/* Channel 2 input, capture data is read through TIM1->CCR2H / TIM1->CCR2L */
void PWMA_ISR() interrupt 26
{
    if(TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &=~0X02;
    }
    if(TIM1->SR1 & 0X04)
    {
        P03 = ~P03;
        cap = TIM1-> CCR2H;           //read CCR2H
        cap = (cap << 8) + TIM1-> CCR2L; //read CCR2L
        TIM1->SR1 &=~0X04;
    }
}

```

20.8.8 Input capture mode to measure the pulse high level width (capture rising edge to falling edge)

C language code

//Operating frequency for test is 11.0592MHz

```

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

sfr      P_SW2      = 0xba;
sfr      P1M0       = 0x92;
sfr      P1M1       = 0x91;
sfr      P3M0       = 0xb2;
sfr      P3M1       = 0xb1;
sfr      P5M0       = 0xca;
sfr      P5M1       = 0xc9;

#define PWMA_CR1      (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_IER      (*(unsigned char volatile xdata *)0xfec4)
#define PWMA_SR1      (*(unsigned char volatile xdata *)0xfec5)
#define PWMA_CCMR1    (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCMR2    (*(unsigned char volatile xdata *)0xfec9)
#define PWMA_CCER1    (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCR1     (*(unsigned int volatile xdata *)0xfed5)
#define PWMA_CCR2     (*(unsigned int volatile xdata *)0xfed7)

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
}

```



```

P5M0 = 0x00;
P5M1 = 0x00;

P_SW2 = 0x80;

// ( CC1 captures the rising edge of TI1, CC2 captures the falling
edge of TI1)
PWMA_CCER1 = 0x00;
PWMA_CCMR1 = 0x01;
PWMA_CCMR2 = 0x02;
PWMA_CCER1 = 0x11;
PWMA_CCER1 |= 0x00;
PWMA_CCER1 |= 0x20;
PWMA_CR1 = 0x01;

PWMA_IER = 0x04;
EA = 1;

while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04;

        cnt = PWMA_CCR2 - PWMA_CCR1;
        // The difference is the high level width
    }
}

```

20.8.9 Input capture mode to measure the pulse low-level width (capture falling edge to rising edge)

C language code

//Operating frequency for test is 11.0592MHz

```

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

sfr    P_SW2      = 0xba;
sfr    P1M0       = 0x92;
sfr    P1M1       = 0x91;
sfr    P3M0       = 0xb2;
sfr    P3M1       = 0xb1;
sfr    P5M0       = 0xca;
sfr    P5M1       = 0xc9;

#define PWMA_CR1    (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_IER    (*(unsigned char volatile xdata *)0xfec4)
#define PWMA_SR1    (*(unsigned char volatile xdata *)0xfec5)

```

```

#define PWMA_CCMR1    (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCMR2    (*(unsigned char volatile xdata *)0xfec9)
#define PWMA_CCER1    (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_CCR1     (*(unsigned int volatile xdata *)0xfed5)
#define PWMA_CCR2     (*(unsigned int volatile xdata *)0xfed7)

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    // ( CC1 captures the rising edge of TI1, CC2 captures the falling
    edge of TI1)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;           // CC1 is in the input mode and is mapped to TI1FP1
    PWMA_CCMR2 = 0x02;           // CC2 is in the input mode and is mapped to TI1FP2
    PWMA_CCER1 = 0x11;           // Enable capture function on CC1/CC2
    PWMA_CCR1 |= 0x00;           // Set the capture polarity to the rising edge of CC1
    PWMA_CCR1 |= 0x20;           // Set capture polarity to falling edge of CC2
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x02;             // Enable CC1 capture interrupt
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;

        cnt = PWMA_CCR1 - PWMA_CCR2;           // The difference is the low level width
    }
}

```

20.8.10 Simultaneous measurement of pulse period and duty cycle in input capture mode

Note: Only PWM1P, PWM2P, PWM5, PWM6 can measure the period and duty cycle at the same time

C language code

//Operating frequency for test is 11.0592MHz

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

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

```
sfr      P_SW2      = 0xba;
sfr      P1M0       = 0x92;
sfr      P1M1       = 0x91;
sfr      P3M0       = 0xb2;
sfr      P3M1       = 0xb1;
sfr      P5M0       = 0xca;
sfr      P5M1       = 0xc9;
```

```
#define PWMA_CR1      (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_SMCR      (*(unsigned char volatile xdata *)0xfec2)
#define PWMA_IER      (*(unsigned char volatile xdata *)0xfec4)
#define PWMA_SR1      (*(unsigned char volatile xdata *)0xfec5)
#define PWMA_CCMR1     (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCMR2     (*(unsigned char volatile xdata *)0xfec9)
#define PWMA_CCER1     (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_CCR1      (*(unsigned int volatile xdata *)0xfed5)
#define PWMA_CCR2      (*(unsigned int volatile xdata *)0xfed7)
```

```
void main()
```

```
{
    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    edge of TI1)

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;
    PWMA_CCMR2 = 0x02;
    PWMA_CCER1 = 0x11;
    PWMA_CCER1 |= 0x00;
    PWMA_CCER1 |= 0x20;
    PWMA_SMCR = 0x54;
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x06;
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;

        cnt = PWMA_CCR1;

        edge of TI1)
        // ( CC1 captures the rising edge of TI1, CC2 captures the falling
        // CC1 captures the period, CC2 captures the high level width
        // CC1 is in the input mode and is mapped to TI1FP1
        // CC2 is in the input mode and is mapped to TI1FP2
        // Enable capture function on CC1/CC2
        // Set the capture polarity to the rising edge of CC1
        // Set capture polarity to falling edge of CC2
        // TS=TI1FP1, SMS=TI1, rising edge reset mode

        // Enable CC1/CC2 capture interrupt

        // CC1 captures the period
```

```

}
if (PWMA_SR1 & 0x04)
{
    PWMA_SR1 &= ~0x04;

    cnt = PWMA_CCR2;           //CC2 captures duty cycle (high width)
}
}

```

20.8.11 PWM complementary output with dead zone control

C language code

//Operating frequency for test is 11.0592MHz

```

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

```

```

typedef struct TIM1_struct
{

```

```

    volatile unsigned char CR1;
    volatile unsigned char CR2;
    volatile unsigned char SMCR;
    volatile unsigned char ETR;
    volatile unsigned char IER;
    volatile unsigned char SR1;
    volatile unsigned char SR2;
    volatile unsigned char EGR;
    volatile unsigned char CCMR1;
    volatile unsigned char CCMR2;
    volatile unsigned char CCMR3;
    volatile unsigned char CCMR4;
    volatile unsigned char CCER1;
    volatile unsigned char CCER2;
    volatile unsigned char CNTRH;
    volatile unsigned char CNTRL;
    volatile unsigned char PSCRH;
    volatile unsigned char PSCRL;
    volatile unsigned char ARRH;
    volatile unsigned char ARRL;
    volatile unsigned char RCR;
    volatile unsigned char CCR1H;
    volatile unsigned char CCR1L;
    volatile unsigned char CCR2H;
    volatile unsigned char CCR2L;
    volatile unsigned char CCR3H;
    volatile unsigned char CCR3L;
    volatile unsigned char CCR4H;
    volatile unsigned char CCR4L;
    volatile unsigned char BKR;
    volatile unsigned char DTR;
    volatile unsigned char OISR;

```

```

}TIM1_TypeDef;

```

```

/*!< control register 1 */
/*!< control register 2 */
/*!< Synchro mode control register */
/*!< external trigger register */
/*!< interrupt enable register*/
/*!< status register 1 */
/*!< status register 2 */
/*!< event generation register */
/*!< CC mode register 1 */
/*!< CC mode register 2 */
/*!< CC mode register 3 */
/*!< CC mode register 4 */
/*!< CC enable register 1 */
/*!< CC enable register 2 */
/*!< counter high */
/*!< counter low */
/*!< prescaler high */
/*!< prescaler low */
/*!< auto-reload register high */
/*!< auto-reload register low */
/*!< Repetition Counter register */
/*!< capture/compare register 1 high */
/*!< capture/compare register 1 low */
/*!< capture/compare register 2 high */
/*!< capture/compare register 2 low */
/*!< capture/compare register 3 high */
/*!< capture/compare register 3 low */
/*!< capture/compare register 3 high */
/*!< capture/compare register 3 low */
/*!< Break Register */
/*!< dead-time register */
/*!< Output idle register */

```

```

#define    TIM1_BaseAddress    0xFEC0

#define    TIM1                ((TIM1_TypeDef xdata*)TIM1_BaseAddress)
#define    PWMA_ENO            (*(unsigned char volatile xdata *)0xFEB1)
#define    PWMA_PS             (*(unsigned char volatile xdata *)0xFEB2)

sfr        P0M0                =    0x94;
sfr        P0M1                =    0x93;
sfr        P1M0                =    0x92;
sfr        P1M1                =    0x91;
sfr        P3M0                =    0xb2;
sfr        P3M1                =    0xb1;
sfr        P_SW2               =    0xba;

sbit       P03                 =    P0^3;

void main(void)
{
    P_SW2 = 0x80;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;

    PWMA_ENO = 0xFF;           //IO outputs PWM
    PWMA_PS = 0x00;           //00:PWM at P1

    /*****
    PWMx_duty = [CCRx/(ARR + 1)]*100
    *****/

    TIM1-> PSCRH = 0x00;       // Prescaler register
    TIM1-> PSCRL = 0x00;
    TIM1-> DTR = 0x00;         // Dead time configuration

    TIM1-> CCMR1 = 0x68;       // Channel Mode Configuration
    TIM1-> CCMR2 = 0x68;
    TIM1-> CCMR3 = 0x68;
    TIM1-> CCMR4 = 0x68;

    TIM1-> ARRH = 0x08;        // Auto-reload register, counter overflow point
    TIM1-> ARRL = 0x00;

    TIM1-> CCR1H = 0x04;       // Counter comparison value
    TIM1-> CCR1L = 0x00;
    TIM1-> CCR2H = 0x02;
    TIM1-> CCR2L = 0x00;
    TIM1-> CCR3H = 0x01;
    TIM1-> CCR3L = 0x00;
    TIM1-> CCR4H = 0x01;
    TIM1-> CCR4L = 0x00;

    TIM1-> CCER1 = 0x55;       // Configure Channel Output Enable and Polarity
    TIM1-> CCER2 = 0x55;       // Configure Channel Output Enable and Polarity

    TIM1-> BKR = 0x80;         // enable main output, equivalent to main switch
    TIM1-> IER = 0x02;         // enable interrupt
    TIM1-> CR1 = 0x01;         // enable counter

```

```

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(TIM1->SR1 & 0X02)
    {
        P03 = ~P03;
        TIM1->SR1 &=~0X02;
    }
}

```

20.8.12 PWM port as external interrupt (falling edge interrupt or rising edge interrupt)

C language code

//Operating frequency for test is 11.0592MHz

```

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

#define PWMA_CR1      (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_IER      (*(unsigned char volatile xdata *)0xfec4)
#define PWMA_SR1      (*(unsigned char volatile xdata *)0xfec5)
#define PWMA_CCMR1    (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCER1    (*(unsigned char volatile xdata *)0xfec9)

sfr P0M0 = 0x94;
sfr P0M1 = 0x93;
sfr P1M0 = 0x92;
sfr P1M1 = 0x91;
sfr P3M0 = 0xb2;
sfr P3M1 = 0xb1;

sfr P_SW2 = 0xba;

sbit P37 = P3^7;

void main(void)
{
    P_SW2 = 0x80;

    P1M1 = 0x00;
    P1M0 = 0x00;
    P3M1 = 0x00;
    P3M0 = 0x00;

    P_SW2 = 0x80;

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;
}

```

//(Capture PWM1P rising/falling edge)

//CC1 is in the input mode and is mapped to TI1FP1

```

    PWMA_CCER1 = 0x01;                // Enable capture function on CC1
    PWMA_CCER1 |= 0x00;                // Set the capture polarity to the rising edge of CC1
//    PWMA_CCER1 |= 0x02;                // Set the capture polarity to the falling edge of CC1
    PWMA_CR1 = 0x01;
    PWMA_IER = 0x02;
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P37 = ~P37;
        PWMA_SR1 &= ~0X02;
    }
}

```

20.8.13 Output waveforms with any period and any duty cycle

C language code

//Operating frequency for test is 11.0592MHz

```

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

sfr      P_SW2      =    0xba;

#define    PWMA_CCER1    (*(unsigned char volatile xdata *)0xfecc)
#define    PWMA_CCMR1    (*(unsigned char volatile xdata *)0xfec8)
#define    PWMA_ENO      (*(unsigned char volatile xdata *)0xfeb1)
#define    PWMA_BKR      (*(unsigned char volatile xdata *)0xfedd)
#define    PWMA_CCR1     (*(unsigned int volatile xdata *)0xfed5)
#define    PWMA_ARR      (*(unsigned int volatile xdata *)0xfed2)
#define    PWMA_CR1      (*(unsigned char volatile xdata *)0xfec0)

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;

P_SW2 = 0x80;
PWMA_CCER1 = 0x00;      // CCERx must be cleared to close the channel before writing to CCMRx
PWMA_CCMR1 = 0x60;      // Set CC1 to PWMA output mode
PWMA_CCER1 = 0x01;      // Enable CC1 channel
PWMA_CCR1 = 100;        // Set duty cycle time
PWMA_ARR = 500;         // Set period time
PWMA_ENO = 0x01;        // Enable PWM1P port output
PWMA_BKR = 0x80;        // enable main output
PWMA_CR1 = 0x01;        // start timing

while (1);
}

```

20.8.14 Use PWM CEN to start PWMA timer and trigger ADC in real time

C language code

//Operating frequency for test is 11.0592MHz

```

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

#define PWMA_CR1      (*(unsigned char volatile xdata *)0xfec0)
#define PWMA_CR2      (*(unsigned char volatile xdata *)0xfec1)
#define PWMA_IER      (*(unsigned char volatile xdata *)0xfec4)
#define PWMA_SR1      (*(unsigned char volatile xdata *)0xfec5)
#define PWMA_CCMR1    (*(unsigned char volatile xdata *)0xfec8)
#define PWMA_CCER1    (*(unsigned char volatile xdata *)0xfec9)
#define PWMA_ARR      (*(unsigned int volatile xdata *)0xfed2)

sfr P0M0 = 0x94;
sfr P0M1 = 0x93;
sfr P1M0 = 0x92;
sfr P1M1 = 0x91;
sfr P3M0 = 0xb2;
sfr P3M1 = 0xb1;

sfr P_SW2 = 0xba;

sfr ADC_CONTR = 0xbc;

```



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

sbit     EADC           =  IE^5;

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

void main()
{
    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
    P3M1 = 0x00;

    P_SW2 |= 0x80;

    ADC_CONTR = ADC_POWER | ADC_EPWMT | 0;           // Select P1.0 as ADC input channel
    delay();                                         // Wait for ADC power supply to stabilize
    EADC = 1;

    PWMA_CR2 = 0x10;                               // The CEN signal is TRGO, which can be used to trigger the ADC
    PWMA_ARR = 5000;
    PWMA_IER = 0x01;
    PWMA_CR1 = 0x01;                               // Set CEN to start the PWMA timer and trigger the ADC in real time
    EA = 1;

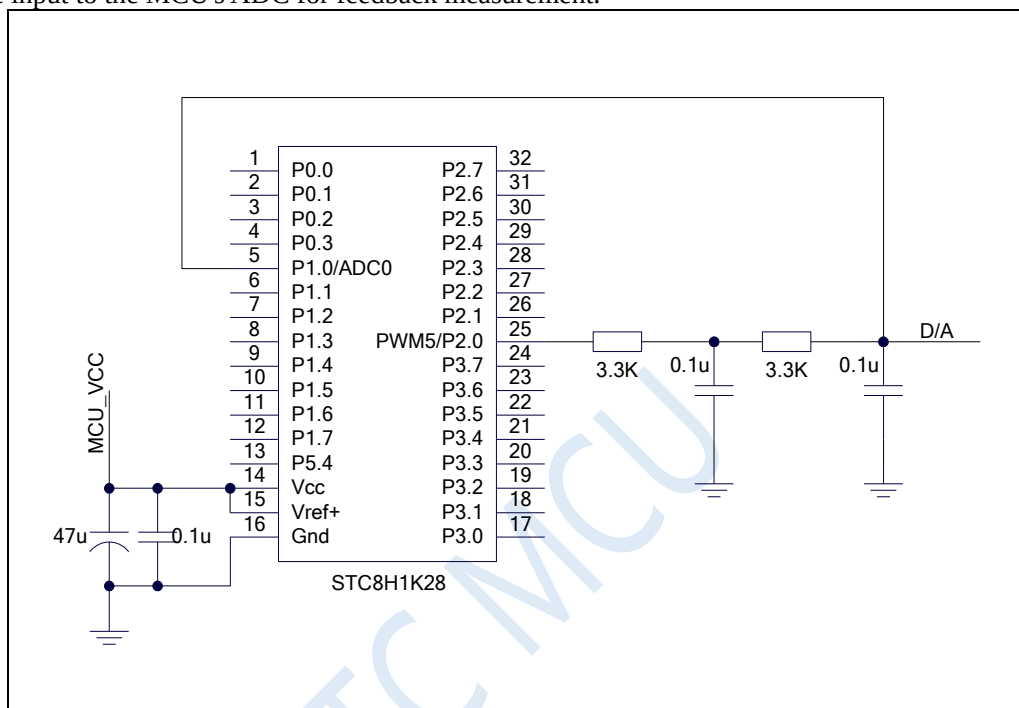
    while (1);
}

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

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &= ~0x01;
    }
}
```

20.8.15 Reference circuit diagram for implementing 16-bit DAC using PWM

The advanced PWM timer of STC8H series MCU can output 16-bit PWM waveform, and then after two-stage low-pass filtering, 16-bit DAC signal can be generated. The DAC signal can be changed by adjusting the high-level duty cycle of the PWM waveform. The application circuit diagram is shown in the figure below. The output DAC signal can be input to the MCU's ADC for feedback measurement.



20.8.16 Using PWM to realize complementary SPWM

Advanced PWM timers PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N Each channel can realize PWM output independently, or two-by-two complementary symmetrical output. Demonstrate the use of PWM1P and PWM1N to generate complementary SPWM. The main clock is 24MHZ, the PWM clock is 1T, the PWM period is 2400, the dead zone is 12 clocks (0.5us), and the sine wave table uses 200 points.

Output sine wave frequency = $24000000/2400/200 = 50$ HZ.

This program is just a demonstration program of SPWM. Users can modify the PWM period and the number and amplitude of the sine wave through the above calculation method. The output frequency of this program is fixed. If frequency conversion is required, please design your own frequency conversion scheme.

C language code

```
//Operating frequency for test is 24MHz
```

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

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

```
#define MAIN_Fosc 24000000L
```

```
// define the main clock
```

```
typedef unsigned char u8;
```

```
typedef unsigned int u16;
```

```

typedef unsigned long u32;

sfr TH2 = 0xD6;
sfr TL2 = 0xD7;
sfr IE2 = 0xAF;
sfr INT_CLKO = 0x8F;
sfr AUXR = 0x8E;
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;

```

```

/*****user-defined macro *****/

```

```

#define PWMA_ENO (*(unsigned char volatile xdata *) 0xFEB1)
#define PWMA_PS (*(unsigned char volatile xdata *) 0xFEB2)
#define PWMB_ENO (*(unsigned char volatile xdata *) 0xFEB5)
#define PWMB_PS (*(unsigned char volatile xdata *) 0xFEB6)

#define PWMA_CR1 (*(unsigned char volatile xdata *) 0xFEC0)
#define PWMA_CR2 (*(unsigned char volatile xdata *) 0xFEC1)
#define PWMA_SMCR (*(unsigned char volatile xdata *) 0xFEC2)
#define PWMA_ETR (*(unsigned char volatile xdata *) 0xFEC3)
#define PWMA_IER (*(unsigned char volatile xdata *) 0xFEC4)
#define PWMA_SR1 (*(unsigned char volatile xdata *) 0xFEC5)
#define PWMA_SR2 (*(unsigned char volatile xdata *) 0xFEC6)
#define PWMA_EGR (*(unsigned char volatile xdata *) 0xFEC7)
#define PWMA_CCMR1 (*(unsigned char volatile xdata *) 0xFEC8)
#define PWMA_CCMR2 (*(unsigned char volatile xdata *) 0xFEC9)
#define PWMA_CCMR3 (*(unsigned char volatile xdata *) 0xFECA)
#define PWMA_CCMR4 (*(unsigned char volatile xdata *) 0xFECB)
#define PWMA_CCER1 (*(unsigned char volatile xdata *) 0xFECC)
#define PWMA_CCER2 (*(unsigned char volatile xdata *) 0xFECD)
#define PWMA_CNTRH (*(unsigned char volatile xdata *) 0xFECE)
#define PWMA_CNTRL (*(unsigned char volatile xdata *) 0xFECF)
#define PWMA_PSCRH (*(unsigned char volatile xdata *) 0xFED0)
#define PWMA_PSCRL (*(unsigned char volatile xdata *) 0xFED1)
#define PWMA_ARRH (*(unsigned char volatile xdata *) 0xFED2)
#define PWMA_ARRL (*(unsigned char volatile xdata *) 0xFED3)

```

```

#define PWMA_RCR      (*(unsigned char volatile xdata *) 0xFED4)
#define PWMA_CCR1H    (*(unsigned char volatile xdata *) 0xFED5)
#define PWMA_CCR1L    (*(unsigned char volatile xdata *) 0xFED6)
#define PWMA_CCR2H    (*(unsigned char volatile xdata *) 0xFED7)
#define PWMA_CCR2L    (*(unsigned char volatile xdata *) 0xFED8)
#define PWMA_CCR3H    (*(unsigned char volatile xdata *) 0xFED9)
#define PWMA_CCR3L    (*(unsigned char volatile xdata *) 0xFEDA)
#define PWMA_CCR4H    (*(unsigned char volatile xdata *) 0xFEDB)
#define PWMA_CCR4L    (*(unsigned char volatile xdata *) 0xFEDC)
#define PWMA_BKR      (*(unsigned char volatile xdata *) 0xFEDD)
#define PWMA_DTR      (*(unsigned char volatile xdata *) 0xFEDE)
#define PWMA_OISR     (*(unsigned char volatile xdata *) 0xFEDF)

```

```

/*****

```

```

#define PWMA_1        0x00          //P:P1.0 N:P1.1
#define PWMA_2        0x01          //P:P2.0 N:P2.1
#define PWMA_3        0x02          //P:P6.0 N:P6.1

#define PWMB_1        0x00          //P:P1.2/P5.4 N:P1.3
#define PWMB_2        0x04          //P:P2.2 N:P2.3
#define PWMB_3        0x08          //P:P6.2 N:P6.3

#define PWM3_1        0x00          //P:P1.4 N:P1.5
#define PWM3_2        0x10          //P:P2.4 N:P2.5
#define PWM3_3        0x20          //P:P6.4 N:P6.5

#define PWM4_1        0x00          //P:P1.6 N:P1.7
#define PWM4_2        0x40          //P:P2.6 N:P2.7
#define PWM4_3        0x80          //P:P6.6 N:P6.7
#define PWM4_4        0xC0          //P:P3.4 N:P3.3

#define ENO1P         0x01
#define ENO1N         0x02
#define ENO2P         0x04
#define ENO2N         0x08
#define ENO3P         0x10
#define ENO3N         0x20
#define ENO4P         0x40
#define ENO4N         0x80

```

```

/***** local variable declaration *****/

```

```

unsigned int code T_SinTable[]=
{
    1220, 1256, 1292, 1328, 1364, 1400, 1435, 1471,
    1506, 1541, 1575, 1610, 1643, 1677, 1710, 1742,
    1774, 1805, 1836, 1866, 1896, 1925, 1953, 1981,
    2007, 2033, 2058, 2083, 2106, 2129, 2150, 2171,
    2191, 2210, 2228, 2245, 2261, 2275, 2289, 2302,
    2314, 2324, 2334, 2342, 2350, 2356, 2361, 2365,
    2368, 2369, 2370, 2369, 2368, 2365, 2361, 2356,
    2350, 2342, 2334, 2324, 2314, 2302, 2289, 2275,
    2261, 2245, 2228, 2210, 2191, 2171, 2150, 2129,
    2106, 2083, 2058, 2033, 2007, 1981, 1953, 1925,
    1896, 1866, 1836, 1805, 1774, 1742, 1710, 1677,
    1643, 1610, 1575, 1541, 1506, 1471, 1435, 1400,
    1364, 1328, 1292, 1256, 1220, 1184, 1148, 1112,

```

```

1076, 1040, 1005, 969, 934, 899, 865, 830,
797, 763, 730, 698, 666, 635, 604, 574,
544, 515, 487, 459, 433, 407, 382, 357,
334, 311, 290, 269, 249, 230, 212, 195,
179, 165, 151, 138, 126, 116, 106, 98,
90, 84, 79, 75, 72, 71, 70, 71,
72, 75, 79, 84, 90, 98, 106, 116,
126, 138, 151, 165, 179, 195, 212, 230,
249, 269, 290, 311, 334, 357, 382, 407,
433, 459, 487, 515, 544, 574, 604, 635,
666, 698, 730, 763, 797, 830, 865, 899,
934, 969, 1005, 1040, 1076, 1112, 1148, 1184,
};

```

```

u16 PWMA_Duty;
u8 PWM_Index;

```

```
// SPWM lookup table index
```

```

/***** main function *****/

```

```
void main(void)
```

```

{
    P0M1 = 0; P0M0 = 0; //set as quasi-bidirectional port
    P1M1 = 0; P1M0 = 0; //set as quasi-bidirectional port
    P2M1 = 0; P2M0 = 0; //set as quasi-bidirectional port
    P3M1 = 0; P3M0 = 0; //set as quasi-bidirectional port
    P4M1 = 0; P4M0 = 0; //set as quasi-bidirectional port
    P5M1 = 0; P5M0 = 0; //set as quasi-bidirectional port
    P6M1 = 0; P6M0 = 0; //set as quasi-bidirectional port
    P7M1 = 0; P7M0 = 0; //set as quasi-bidirectional port

    PWMA_Duty = 1220;

    P_SW2 |= 0x80;

    PWMA_CCER1 = 0x00; // CCxE must be cleared before writing to CCMRx to close the
channel
    PWMA_CCER2 = 0x00;
    PWMA_CCMR1 = 0x60; // Channel Mode Configuration
    // PWMA_CCMR2 = 0x60;
    // PWMA_CCMR3 = 0x60;
    // PWMA_CCMR4 = 0x60;
    PWMA_CCER1 = 0x05; // Configure Channel Output Enable and Polarity
    // PWMA_CCER2 = 0x55;

    PWMA_ARRH = 0x09; // Set period time
    PWMA_ARRL = 0x60;

    PWMA_CCR1H = (u8)(PWMA_Duty >> 8); // Set duty cycle time
    PWMA_CCR1L = (u8)(PWMA_Duty);

    PWMA_DTR = 0x0C; // Set dead time

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P; //enable output
    PWMA_ENO |= ENO1N; //enable output
    // PWMA_ENO |= ENO2P; //enable output
    // PWMA_ENO |= ENO2N; //enable output
    // PWMA_ENO |= ENO3P; //enable output
    // PWMA_ENO |= ENO3N; //enable output

```

```

// PWMA_ENO |= ENO4P; //enable output
// PWMA_ENO |= ENO4N; //enable output

PWMA_PS = 0x00; // Advanced PWM Channel Output Pin Selection Bits
PWMA_PS |= PWMA_3; // Select PWMA_3 channel
// PWMA_PS |= PWMB_3; // Select PWMB_3 channel
// PWMA_PS |= PWM3_3; // Select PWM3_3 channel
// PWMA_PS |= PWM4_3; // Select PWM4_3 channel

PWMA_BKR = 0x80; // enable main output
PWMA_IER = 0x01; // enable interrupt
PWMA_CR1 |= 0x01; //start timing

P_SW2 &= 0x7f;

EA = 1;

while (1)
{
}

}

/***** interrupt function *****/
void PWMA_ISR() interrupt 26
{
    P_SW2 |= 0x80;
    if (PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &= ~0x01;
        PWMA_Duty = T_SinTable[PWM_Index];
        if (++PWM_Index >= 200)
            PWM_Index = 0;

        PWMA_CCR1H = (u8)(PWMA_Duty >> 8); // Set duty cycle time
        PWMA_CCR1L = (u8)(PWMA_Duty);
    }
    PWMA_SR1 = 0;
    P_SW2 &= 0x7f;
}

```

20.8.17 Advanced PWM Output - Frequency Adjustable - Pulse Count

C language code

//Operating frequency for test is 24MHz

```

/***** Function Description *****/
This routine is based on STC8H8K64U as the main control chip to write and test on the experiment box 8. STC8H series chips can be used as a general reference.
Advanced PWM timer realizes high-speed PWM pulse output.
The period/duty cycle is adjustable, and the number of pulses is counted through compare/capture interrupts.
Demonstrate output through P6 port, output PWM every 10ms, stop output after counting 10 pulses.
The timer adjusts the PWM period every 1ms.
When downloading, select the clock 24MHZ (users can modify the frequency by themselves).
*****/

```

```

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

#define MAIN_Fosc      24000000L

typedef unsigned char  u8;
typedef unsigned int   u16;
typedef unsigned long  u32;

sfr TH2      = 0xD6;
sfr TL2      = 0xD7;
sfr IE2      = 0xAF;
sfr INT_CLKO = 0x8F;
sfr AUXR     = 0x8E;
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 P0      = 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;

```

```

/***** user-defined macro *****/

```

```

#define    Timer0_Reload    (65536UL -(MAIN_Fosc / 1000))    // Timer0 interrupt frequency, 1000 times/second

#define    PWMA_ENO    (*(unsigned char volatile xdata *) 0xFEB1)
#define    PWMA_PS    (*(unsigned char volatile xdata *) 0xFEB2)
#define    PWMB_ENO    (*(unsigned char volatile xdata *) 0xFEB5)
#define    PWMB_PS    (*(unsigned char volatile xdata *) 0xFEB6)

#define    PWMA_CR1    (*(unsigned char volatile xdata *) 0xFEC0)
#define    PWMA_CR2    (*(unsigned char volatile xdata *) 0xFEC1)
#define    PWMA_SMCR    (*(unsigned char volatile xdata *) 0xFEC2)
#define    PWMA_ETR    (*(unsigned char volatile xdata *) 0xFEC3)
#define    PWMA_IER    (*(unsigned char volatile xdata *) 0xFEC4)
#define    PWMA_SR1    (*(unsigned char volatile xdata *) 0xFEC5)
#define    PWMA_SR2    (*(unsigned char volatile xdata *) 0xFEC6)
#define    PWMA_EGR    (*(unsigned char volatile xdata *) 0xFEC7)
#define    PWMA_CCMR1    (*(unsigned char volatile xdata *) 0xFEC8)
#define    PWMA_CCMR2    (*(unsigned char volatile xdata *) 0xFEC9)
#define    PWMA_CCMR3    (*(unsigned char volatile xdata *) 0xFECA)
#define    PWMA_CCMR4    (*(unsigned char volatile xdata *) 0xFECB)
#define    PWMA_CCER1    (*(unsigned char volatile xdata *) 0xFECC)
#define    PWMA_CCER2    (*(unsigned char volatile xdata *) 0xFECD)
#define    PWMA_CNTR    (*(unsigned int volatile xdata *) 0xFECE)
#define    PWMA_CNTRH    (*(unsigned char volatile xdata *) 0xFECE)
#define    PWMA_CNTRL    (*(unsigned char volatile xdata *) 0xFECF)
#define    PWMA_PSCRH    (*(unsigned char volatile xdata *) 0xFED0)
#define    PWMA_PSCRL    (*(unsigned char volatile xdata *) 0xFED1)
#define    PWMA_ARR    (*(unsigned int volatile xdata *) 0xFED2)

```



```

#define PWMA_ARRH      (*(unsigned char volatile xdata *) 0xFED2)
#define PWMA_ARRL      (*(unsigned char volatile xdata *) 0xFED3)
#define PWMA_RCR        (*(unsigned char volatile xdata *) 0xFED4)
#define PWMA_CCR1       (*(unsigned int  volatile xdata *) 0xFED5)
#define PWMA_CCR1H      (*(unsigned char volatile xdata *) 0xFED5)
#define PWMA_CCR1L      (*(unsigned char volatile xdata *) 0xFED6)
#define PWMA_CCR2       (*(unsigned int  volatile xdata *) 0xFED7)
#define PWMA_CCR2H      (*(unsigned char volatile xdata *) 0xFED7)
#define PWMA_CCR2L      (*(unsigned char volatile xdata *) 0xFED8)
#define PWMA_CCR3       (*(unsigned int  volatile xdata *) 0xFED9)
#define PWMA_CCR3H      (*(unsigned char volatile xdata *) 0xFED9)
#define PWMA_CCR3L      (*(unsigned char volatile xdata *) 0xFEDA)
#define PWMA_CCR4       (*(unsigned int  volatile xdata *) 0xFEDB)
#define PWMA_CCR4H      (*(unsigned char volatile xdata *) 0xFEDB)
#define PWMA_CCR4L      (*(unsigned char volatile xdata *) 0xFEDC)
#define PWMA_BKR        (*(unsigned char volatile xdata *) 0xFEDD)
#define PWMA_DTR        (*(unsigned char volatile xdata *) 0xFEDE)
#define PWMA_OISR       (*(unsigned char volatile xdata *) 0xFEDF)

```

```

/*****

```

```

#define PWM1_1          0x00          //P:P1.0 N:P1.1
#define PWM1_2          0x01          //P:P2.0 N:P2.1
#define PWM1_3          0x02          //P:P6.0 N:P6.1

#define PWM2_1          0x00          //P:P1.2/P5.4 N:P1.3
#define PWM2_2          0x04          //P:P2.2 N:P2.3
#define PWM2_3          0x08          //P:P6.2 N:P6.3

#define PWM3_1          0x00          //P:P1.4 N:P1.5
#define PWM3_2          0x10          //P:P2.4 N:P2.5
#define PWM3_3          0x20          //P:P6.4 N:P6.5

#define PWM4_1          0x00          //P:P1.6 N:P1.7
#define PWM4_2          0x40          //P:P2.6 N:P2.7
#define PWM4_3          0x80          //P:P6.6 N:P6.7
#define PWM4_4          0xC0          //P:P3.4 N:P3.3

#define ENO1P           0x01
#define ENO1N           0x02
#define ENO2P           0x04
#define ENO2N           0x08
#define ENO3P           0x10
#define ENO3N           0x20
#define ENO4P           0x40
#define ENO4N           0x80

```

```

/***** local variable declaration *****/

```

```

bit B_1ms;              //1ms flag
bit PWM1_Flag;

u16 Period;
u8 Counter;
u8 msSecond;

void UpdatePwm(void);
void TxPulse(void);

```

```

/***** main function *****/

```

```

void main(void)

```

```

{
    P0M1 = 0x00; P0M0 = 0x00;           //set as quasi-bidirectional port
    P1M1 = 0x00; P1M0 = 0x00;           //set as quasi-bidirectional port
    P2M1 = 0x00; P2M0 = 0x00;           //set as quasi-bidirectional port
    P3M1 = 0x00; P3M0 = 0x00;           //set as quasi-bidirectional port
    P4M1 = 0x00; P4M0 = 0x00;           //set as quasi-bidirectional port
    P5M1 = 0x00; P5M0 = 0x00;           //set as quasi-bidirectional port
    P6M1 = 0x00; P6M0 = 0x00;           //set as quasi-bidirectional port
    P7M1 = 0x00; P7M0 = 0x00;           //set as quasi-bidirectional port

    PWM1_Flag = 0;
    Counter = 0;
    Period = 0x1000;

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

    P_SW2 |= 0x80;                       // Enable XFR access

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P;                  // enable output

    PWMA_PS = 0x00;                     // Advanced PWM Channel Output Pin Selection Bits
    PWMA_PS |= PWM1_3;                  // Select PWM1_3 channel

    UpdatePwm();
    PWMA_BKR = 0x80;                    // enable main output
    PWMA_CR1 |= 0x01;                   // start timing

    P40 = 0;                            // Powering the LED
    EA = 1;                             //Enable CPU interrupt

    while (1)
    {
        if(B_1ms)
        {
            B_1ms = 0;
            msSecond++;
            if(msSecond >= 10)
            {
                msSecond = 0;
                TxPulse();               //10ms Start a PWM output
            }
        }
    }
}

```

```

/***** send pulse function *****/

```

```

void TxPulse(void)

```

```

{
    PWMA_CCER1 = 0x00;                  // CCxE must be cleared before writing to CCMRx to close the
channel

```

```

    PWMA_CCMR1 = 0x60;           // Set PWM1 Mode 1 Output
    PWMA_CCER1 = 0x01;           // Enable CC1E channel, active high
    PWMA_SR1 = 0;                //clear flag
    PWMA_CNTR = 0;               //clear counter
    PWMA_IER = 0x02;             // Enable capture/compare 1 interrupt
}

/***** Timer0 1ms interrupt function *****/
void timer0(void) interrupt 1
{
    B_1ms = 1;
    if(PWM1_Flag)
    {
        Period++;                // Period increment
        if(Period >= 0x1000) PWM1_Flag = 0;
    }
    else
    {
        Period--;                // period decrement
        if(Period <= 0x0100) PWM1_Flag = 1;
    }
    UpdatePwm();                // Set period, duty cycle
}

/***** PWM interrupt function *****/
void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        PWMA_SR1 &= ~0X02;       //clear flag

        Counter++;
        if(Counter >= 10)         // Turn off the PWM counter after counting 10 pulses
        {
            Counter = 0;
            PWMA_CCER1 = 0x00;     // CCxE must be cleared before writing to CCMRx to close the
channel
            PWMA_CCMR1 = 0x40;     // Set PWM1 to force inactive level
            PWMA_CCER1 = 0x01;     // Enable CC1E channel, active high
            PWMA_IER = 0x00;       //disable interrupt
        }
    }
}

//=====
// function: UpdatePwm(void)
// description: 更新 PWM 周期占空比.
// parameters: none.
// return: none.
// version: V1.0, 2012-11-22
//=====
void UpdatePwm(void)
{
    PWMA_ARR = Period;
    PWMA_CCR1 = (Period >> 1);    // Set duty cycle time: Period/2
}

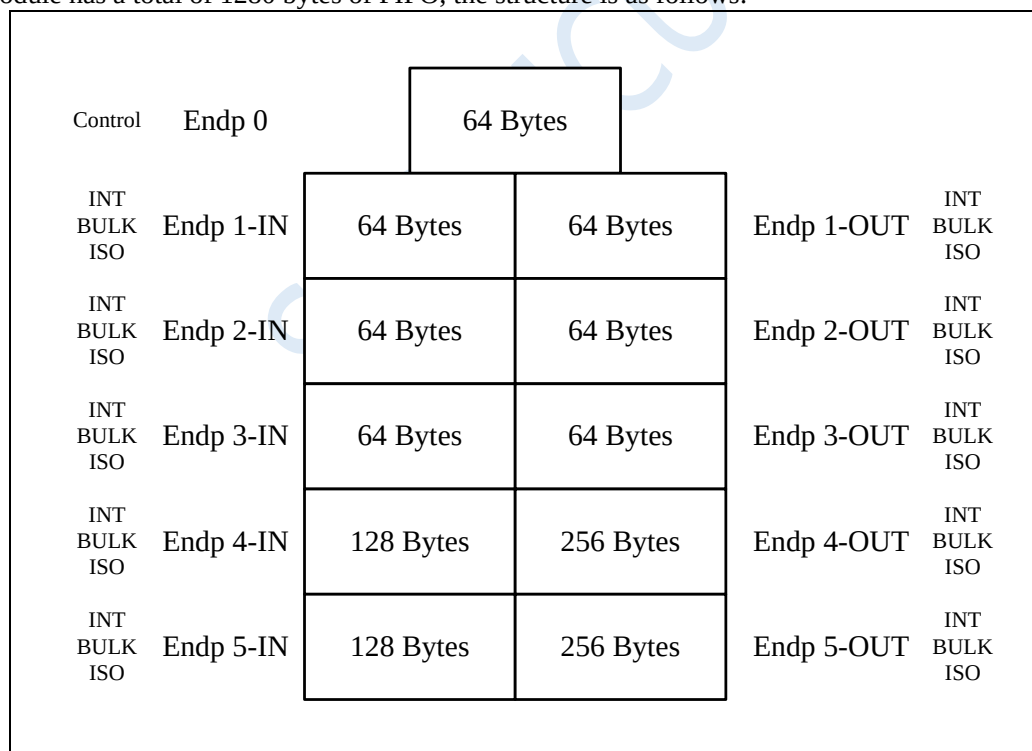
```

21 USB Universal Serial Bus

Product line	USB
STC8H1K08 family	
STC8H1K28 family	
STC8H3K64S4 family	
STC8H3K64S2 family	
STC8H8K64U family	●
STC8H2K64T family	
STC8H4K64TLR family	
STC8H4K64TLCD family	
STC8H4K64LCD family	

STC8H series microcomputer integrates USB2.0/USB1.1 compatible full-speed USB, 6 bidirectional endpoints, supports 4 endpoint transmission modes (control transmission, interrupt transmission, batch transmission and synchronous transmission), each endpoint has a 64-byte buffer Area.

The USB module has a total of 1280 bytes of FIFO, the structure is as follows:



21.1 USB related registers

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	

USBCLK	USB clock control register	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]		0010,0000
USB DAT	USB Data register	ECH									0000,0000
USBCON	USB Control register	F4H	ENUSB	USBRST	PS2M	PUEN	PDEN	DFREC	DP	DM	0000,0000
USBADR	USB Address register	FCH	BUSY	AUTORD	UADR[5:0]						0000,0000

21.1.1 USB Control Register (USBCON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
USBCON	DFH	ENUSB	USBRST	PS2M	PUEN	PDEN	DFREC	DP	DM

ENUSB: USB function and USB clock control bit

0: Turn off USB function and USB clock

1: Enable USB function and USB clock

ENRST: USB reset setting control bit

0: Turn off USB reset settings

1: Enable USB reset

PS2M: PS2 mode function control bit 0: disable PS2 mode function 1: enable PS2 mode function

PUEN: 1.5K pull-up resistor control bit on DP/DM port

0: Disable the pull-up resistor

1: Enable pull-up resistor

PDEN: 500K pull-down resistor control bit on DP/DM port

0: Disable pull-down resistor

1: Enable pull-down resistor

DFREC: Differential receive status bit (read only)

0: The current DP/DM differential status is "0"

1: The current DP/DM differential status is "1"

DP: D+ port status (read only when PS2 is 0, read and write when PS2 is 1)

0: The current D+ is logic 0 level

1: The current D+ is logic 1 level

DM: D-port status (read only when PS2 is 0, read and write when PS2 is 1)

0: The current D- is logic 0 level

1: The current D- is logic 1 level

21.1.2 USB clock control register (USBCLK)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
USBCLK	DEH	ENCKM	PCKI[1:0]	CRE	TST_USB	TST_PHY	PHYTST[1:0]		

ENCKM: PLL frequency multiplication control

0: Disable PLL frequency multiplication

1: Enable PLL frequency multiplication

PCKI[1:0]: PLL clock selection

PCKI[1:0]	PLL clock source
00	6M
01	12M(default)
10	24M
11	IRC/2

CRE: Clock chasing control bit

0: Clock chasing prohibited

1: Enable clock tracking

TST_USB: USB test mode

0: Disable USB test mode

1: Enable USB test mode

TST_PHY: PHY test mode

0: Disable PHY test mode

1: Enable PHY test mode

PHYTST[1:0]: USB PHY test

PHYTST[1:0]	Mode	DP	DM
00	Mode 0: Normal	x	x
01	Mode 1: Force to “1”	1	0
10	Mode 2: Force to “0”	0	1
11	Mode 3: Force to single-ended “0”	0	0

21.1.3 USB Indirect Address Address Register (USBADDR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
USBADDR	EEH	BUSY	AUTORD	UADR[5:0]					

BUSY: USB register read busy flag bit

write 0: meaningless

Write 1: Start the read operation of USB indirect register, the address is set by USBADDR

Read 0: The data in USBDATA register is valid

Reading 1: The data in the USBDATA register is invalid, and the USB is reading the indirect register

AUTORD: USB register automatic reading flag, used for USB FIFO block reading and writing

0: Every time the indirect USB register is read, the BUSY flag must be written first

Write 1: When the software reads USBDATA, the next USB indirect register reading will start automatically (USBADDR remains unchanged)

UADR[5:0]: USB indirect register address

21.1.4 USB indirect address data register (USBDATA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
USBDATA	EFH	UDAT[7:0]							

UDAT[7:0]: used to read and write USB registers indirectly

21.2 USB Controller Registers (SIE)

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
30H	UTRKCTL	UTRKSTS						
28H								
20H	FIFO0	FIFO1	FIFO2	FIFO3	FIFO4	FIFO5		
18H								
10H	INMAXP	CSR0 INCSR1	INCSR2	OUTMAXP	OUTCSR1	OUTCSR2	COUNT0 OUTCOUNT1	OUTCOUNT2
08H		INTROUT1E		INTRUSBE	FRAME1	FRAME2	INDEX	
00H	FADDR	POWER	INTRIN1	-	INTROUT1	-	INTRUSB	INTRIN1E

Symbol	Description	Addre	Bit Address and Symbol	Reset

		ss	B7	B6	B5	B4	B3	B2	B1	B0	Value
FADDR	USB function address register	00H	UPDATE	UADDR[6:0]							0000,0000
POWER	USB Power Management Register	01H	ISOUD	-	-	-	USBRST	USBRSU	USBSUS	ENSUS	0xxx,0000
INTRIN1	USB endpoint IN interrupt flag	02H	-	-	EP5INIF	EP4INIF	EP3INIF	EP2INIF	EP1INIF	EP0IF	xx00,0000
INTROUT1	USB endpoint OUT interrupt flag	04H	-	-	EP5OUTIF	EP4OUTIF	EP3OUTIF	EP2OUTIF	EP1OUTIF	-	xx00,000x
INTRUSB	USB power interrupt flag	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF	xxxx,0000
INTRIN1E	USB endpoint IN interrupt enable bit	07H	-	-	EP5INIE	EP4INIE	EP3INIE	EP2INIE	EP1INIE	EP0IE	xx11,1111
INTROUT1E	USB endpoint OUT interrupt enable bit	09H	-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-	xx11,111x
INTRUSBE	USB power interruption enable bit	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE	xxxx,0110
FRAME1	USB data frame number low byte	0CH	FRAME[7:0]								0000,0000
FRAME2	USB data frame number high byte	0DH	-	-	-	-	-	FRAME[10:8]			xxxx,x000
INDEX	USB Endpoint Index Register	0EH	-	-	-	-	-	INDEX[2:0]			xxxx,x000
INMAXP	Maximum packet size of IN endpoint	10H	INMAXP[7:0]								0000,0000
CSR0	Endpoint 0 control status register	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY	0000,0000
INCSR1	IN endpoint control status register 1	11H	CLRDT	STSTL	SDSTL	FLUSH	-	UNDRUN	FIFONE	IPRDY	0000,x000
INCSR2	IN endpoint control status register 2	12H	AUTOSET	ISO	MODE	ENDMA	FCDT	-	-	-	0010,0xxx
OUTMAXP	Maximum packet size of OUT endpoint	13H	OUTMAXP[7:0]								0000,0000
OUTCSR1	OUT endpoint control status register 1	14H	CLRDT	STSTL	SDSTL	FLUSH	DATERR	OVRRUN	FIFOFUL	OPRDY	0000,0000
OUTCSR2	OUT endpoint control status register 2	15H	AUTOCLR	ISO	ENDMA	DMAMD	-	-	-	-	0000,xxxx
COUNT0	OUT length of endpoint 0	16H	-	OUTCNT0[6:0]							x000,0000
OUTCOUNT1	USB endpoint OUT length low byte	16H	OUTCNT[7:0]								0000,0000
OUTCOUNT2	USB endpoint OUT length high byte	17H	-	-	-	-	-	OUTCNT[10:8]			xxxx,x000
FIFO0	FIFO access register of endpoint 0	20H	FIFO0[7:0]								0000,0000
FIFO1	FIFO access register of endpoint 1	21H	FIFO1[7:0]								0000,0000
FIFO2	FIFO access register of endpoint 2	22H	FIFO2[7:0]								0000,0000
FIFO3	FIFO access register of endpoint 3	23H	FIFO3[7:0]								0000,0000
FIFO4	FIFO access register of endpoint 4	24H	FIFO4[7:0]								0000,0000
FIFO5	FIFO access register of endpoint 5	25H	FIFO5[7:0]								0000,0000
UTRKCTL	USB tracking control register	30H	FTM1	FTM0	INTV[1:0]		ENST5	RES[2:0]			1011,1011
UTRKSTS	USB trace status register	31H	INTVCNT[3:0]				STS[1:0]		TST_UTRK	UTRK_RDY	1111,00x0

21.2.1 USB Function Address Register (FADDR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
FADDR	00H	UPDATE	UADDR[6:0]						

UPDATE: Update the USB function address

0: The last UADDR address has taken effect

1: The last UADDR address has not yet taken effect

UADDR[6:0]: Save the 7-bit function address of the USB

21.2.2 USB Power Control Register (POWER)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
POWER	01H	ISOUD	-	-	-	USBRST	USBRSU	USBSUS	ENSUS

ISOUD (ISO Update): ISO update

0: When the software writes "1" to IPRDY, the USB will send data packets after receiving the next IN token

1: When the software writes "1" to IPRDY, the USB sends a data packet after receiving the SOF token. If the IN token is received before the SOF token, the USB sends a data packet with a length of 0

USBRST (USB Reset): USB reset control bit

Writing "1" to this bit can force an asynchronous USB reset. Read this bit to get the reset status information on the current bus

0: No reset signal is detected on the bus

1: A reset signal is detected on the bus

USBRSU (USB Resume): USB recovery control bit

To force a resume signal to be generated on the bus in software mode to remotely wake up the USB device from the suspend mode. When USB is in suspend mode (USBSUS=1), writing "1" to this bit will force a resume signal to be generated on the USB bus. The software should write "0" to this bit after 10-15ms to end the resume signal. After the software writes "0" to USBRSU, a USB recovery interrupt will be generated, and the hardware will automatically clear USBSUS to "0"

USBSUS (USB Suspend): USB suspend control bit

When USB enters suspend mode, this bit is set to "1" by hardware. When the recovery signal is forced to be generated on the bus by software or the recovery signal is detected on the bus and the INTRUSB register is read, the hardware automatically clears this bit to "0".

ENSUS (Enable Suspend Detection): enable USB suspend detection

0: Suspend detection is disabled, USB will ignore the suspend signal on the bus

1: Enable suspend detection, when the suspend signal on the bus is detected, the USB will enter the suspend mode

21.2.3 USB endpoint IN interrupt flag Register (INTRIN1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INTRIN1	02H	-	-	EP5INIF	EP4INIF	EP3INIF	EP2INIF	EP1INIF	EP0IF

EP5INIF: IN interrupt flag bit of endpoint 5

0: IN interrupt of endpoint 5 is invalid

1: IN interrupt of endpoint 5 is valid

EP4INIF: IN interrupt flag bit of endpoint 4

0: IN interrupt of endpoint 4 is invalid

1: IN interrupt of endpoint 4 is valid

EP3INIF: Endpoint 3 IN interrupt flag bit

0: IN interrupt of endpoint 3 is invalid

1: The IN interrupt of endpoint 3 is valid

EP2INIF: Endpoint 2 IN interrupt flag bit

0: IN interrupt of endpoint 2 is invalid

1: IN interrupt of endpoint 2 is valid

EP1INIF: IN interrupt flag bit of endpoint 1

0: IN interrupt of endpoint 1 is invalid

1: IN interrupt of endpoint 1 is valid

EP0IF: IN/OUT interrupt flag bit of endpoint 0

0: IN/OUT interrupt of endpoint 0 is invalid

1: IN/OUT interrupt of endpoint 0 is valid

After the software reads the INTRIN1 register, the hardware will automatically clear all interrupt flags in INTRIN1

21.2.4 USB endpoint OUT interrupt flag Register (INTROUT1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INTROUT1	04H	-	-	EP5OUTIF	EP4OUTIF	EP3OUTIF	EP2OUTIF	EP1OUTIF	-

EP5OUTIF: Endpoint 5 OUT interrupt flag bit

0: The OUT interrupt of endpoint 5 is invalid

1: The OUT interrupt of endpoint 5 is valid

EP4OUTIF: Endpoint 4 OUT interrupt flag bit

0: The OUT interrupt of endpoint 4 is invalid

1: The OUT interrupt of endpoint 4 is valid

EP3OUTIF: Endpoint 3 OUT interrupt flag bit

0: The OUT interrupt of endpoint 3 is invalid

1: The OUT interrupt of endpoint 3 is valid

EP2OUTIF: Endpoint 2 OUT interrupt flag bit

0: The OUT interrupt of endpoint 2 is invalid

1: The OUT interrupt of endpoint 2 is valid

EP1OUTIF: Endpoint 1 OUT interrupt flag bit

0: The OUT interrupt of endpoint 1 is invalid

1: The OUT interrupt of endpoint 1 is valid

After the software reads the INTROUT1 register, the hardware will automatically clear all interrupt flags in INTROUT1

21.2.5 USB power interruption flag (INTRUSB)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INTRUSB	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF

SOFIF: USB frame start signal interrupt flag

0: USB frame start signal interrupt is invalid

1: USB frame start signal interrupt is valid

RSTIF: USB reset signal interrupt flag

0: USB reset signal interrupt is invalid

1: USB reset signal interrupt is valid

RSUIF: USB recovery signal interruption flag

0: USB recovery signal interruption is invalid

1: USB recovery signal interruption is valid

SUSIF: USB suspend signal interrupt flag

0: USB suspend signal interrupt is invalid

1: USB suspend signal interrupt is valid

After the software reads the INTRUSB register, the hardware will automatically clear all interrupt flags in the INTRUSB

21.2.6 USB endpoint IN interrupt enable register (INTRIN1E)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INTRIN1E	07H	-	-	EP5INIE	EP4INIE	EP3INIE	EP2INIE	EP1INIE	EP0IE

EP5INIE: IN interrupt control bit of endpoint 5

0: Disable the IN interrupt of endpoint 5

1: Enable IN interrupt of endpoint 5

EP4INIE: IN interrupt control bit of endpoint 4

0: Disable the IN interrupt of endpoint 4

1: Enable IN interrupt of endpoint 4

EP3INIE: IN interrupt control bit of endpoint 3

0: Disable the IN interrupt of endpoint 3

1: Enable IN interrupt of endpoint 3

EP2INIE: IN interrupt control bit of endpoint 2

0: Disable the IN interrupt of endpoint 2

1: Enable IN interrupt of endpoint 2

EP1INIE: IN interrupt control bit of endpoint 1

0: Disable the IN interrupt of endpoint 1

1: Enable IN interrupt of endpoint 1

EP0IE: IN/OUT interrupt control bit of endpoint 0

0: Disable the IN/OUT interrupt of endpoint 0

1: Enable IN/OUT interrupt of endpoint 0

21.2.7 USB endpoint OUT interrupt enable register (INTROUT1E)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INTROUT1E	09H	-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-

EP5OUTIE: OUT interrupt control bit of endpoint 5

0: Disable the OUT interrupt of endpoint 5

1: Enable OUT interrupt of endpoint 5

EP4OUTIE: OUT interrupt control bit of endpoint 4

0: Disable the OUT interrupt of endpoint 4

1: Enable OUT interrupt of endpoint 4

EP3OUTIE: Endpoint 3 OUT interrupt control bit

0: Disable the OUT interrupt of endpoint 3

1: Enable OUT interrupt of endpoint 3

EP2OUTIE: Endpoint 2 OUT interrupt control bit

0: Disable the OUT interrupt of endpoint 2

1: Enable OUT interrupt of endpoint 2

EP1OUTIE: OUT interrupt control bit of endpoint 1

0: Disable the OUT interrupt of endpoint 1

1: Enable OUT interrupt of endpoint 1

21.2.8 USB power interruption enable register (INTRUSB)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INTRUSB	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE

SOFIE: USB frame start signal interrupt control bit

0: Disable USB frame start signal interrupt

1: Allow USB frame start signal interrupt

RSTIE: USB reset signal interrupt control bit

0: Disable USB reset signal interrupt

1: Allow USB reset signal interrupt

RSUIE: USB recovery signal interrupt control bit

0: Disable USB recovery signal interrupt

1: Allow USB recovery signal interruption

SUSIE: USB suspend signal interrupt control bit

0: Disable USB suspend signal interrupt

1: Allow USB suspend signal interrupt

21.2.9 USB data frame number registers (FRAMEn)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
FRAME1	0CH	FRAME[7:0]							
FRAME2	0DH	-	-	-	-	-	FRAME[10:8]		

FRAME[10:0]: used to save the 11-bit frame number of the last received data frame

21.2.10 USB Endpoint Index Register (INDEX)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INDEX	0EH	-	-	-	-	-	INDEX[2:0]		

INDEX[2:0]: select USB endpoint

INDEX[2:0]	Target endpoint
000	endpoint 0
001	endpoint 1
010	endpoint 2
011	endpoint 3
100	endpoint 4
101	endpoint 5

21.2.11 Maximum packet size of IN endpoint (INMAXP)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INMAXP	10H	INMAXP[7:0]							

INMAXP[7:0]: Set the maximum data packet size of the USB IN endpoint

When you need to get/set this information, you must first use INDEX to select the target endpoint 0~5

21.2.12 USB Endpoint 0 Control Status Register (CSR0)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CSR0	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY

SSUEND (Serviced Setup End)

SETUP End event processing completed flag. After processing the SETUP end event (SUEND), the software needs to set the SSUEND flag bit, and the hardware will automatically clear the SUEND bit to “0” when it detects that SSUEND is written to “1”.

SOPRDY (Serviced OPRDY)

OPRDY event processing complete flag. After processing the data packet received from endpoint 0, the software needs to set the SOPRDY flag bit, and the hardware will automatically clear the OPRDY bit to "0" when it detects that SOPRDY is written to "1".

SDSTL (Send Stall)

When receiving wrong conditions or unsupported requests, you can write "1" to this bit to end the current data transmission. When the STALL signal is sent, the hardware automatically clears this bit to "0".

SUEND (Setup End)

Installation package end flag. When a control transfer ends before the software writes "1" to the DATAEND bit, the hardware will set this read-only bit to "1". When the software writes a "1" to SSUEND, the hardware clears this bit to "0".

DATEND (Data End)

End of data. The software should write "1" to this bit in the following situations:

1. After sending the last data packet, the firmware writes "1" to IPRDY;
2. After sending a zero-length data packet, the firmware writes "1" to IPRDY;
3. After receiving the last data packet, the firmware writes "1" to SOPRDY;

This bit will be automatically cleared to "0" by hardware

STSTL (Sent Stall)

STALL signal transmission completed flag. After sending the STALL signal, the hardware will set this bit to "1".

This bit must be cleared to "0" by software.

IPRDY (IN Packet Ready)

IN Data packet preparation complete flag. The software should "1" the bit after packing a data to be sent into the FIFO of endpoint 0.

When one of the following conditions occurs, the hardware clears this bit to "0":

1. When the data packet has been sent;
2. When the data packet is covered by a SETUP packet;
3. When the data packet is covered by an OUT packet;

OPRDY (OUT Packet Ready)

OUT packet preparation complete flag. When an OUT packet is received, the hardware will set the read-only bit to "1" and generate an interrupt. This bit is cleared to "0" only when the software writes "1" to the SOPRDY bit.

When you need to get/set this information, you must first use INDEX to select the target endpoint 0

21.2.13 IN endpoint control status register 1 (INCSR1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INCSR1	11H	CLRDT	STSTL	SDSTL	FLUSH	-	UNDRUN	FIFONE	IPRDY

CLRDT (Clear Data Toggle): Reset the IN data toggle bit.

When the IN endpoint needs to reset the data switch bit to "0" due to reconfiguration or STALL, the software needs to write "1" to this data bit.

STSTL (Sent Stall): STALL signal transmission completion flag.

After the STALL signal is sent, the hardware will set this bit to "1" (the FIFO is cleared at this time, and the IPRDY bit is cleared to "0"). This flag must be cleared to "0" by software.

SDSTL (Send Stall): STALL signal sending request bit.

Software should write "1" to this bit to generate STALL signal as a response to an IN token. Software should write "0" to this bit to end the STALL signal. This bit has no effect on ISO mode.

FLUSH (FIFO Flush): Clear the next data packet of the FIFO of the IN endpoint.

Writing "1" to this bit will clear the next data packet to be sent from the IN endpoint FIFO. The FIFO pointer is reset and the IPRDY bit is cleared to "0". If the FIFO contains multiple data packets, the software must write "1" to FLUSH for each data packet. When the FIFO is cleared, the hardware will clear the FLUSH bit to "0".

UNDRUN (Data Underrun): insufficient data.

The function of this bit depends on the mode of the IN endpoint:

ISO mode: When IPRDY is "0" and a zero-length data packet is sent after receiving an IN token, this bit is set to "1".

Interrupt/Batch mode: When NAK is used as a response to an IN token, this bit is set to "1".

This bit must be cleared to "0" by software.

FIFONE (FIFO Not Empty): FIFO not empty flag of IN endpoint

0: FIFO of IN endpoint is empty

1: The FIFO of the IN endpoint contains one or more data packets

IPRDY (IN Packet Ready): IN packet preparation completion flag.

The software should "1" the bit after packing a data to be sent into the endpoint's FIFO. When one of the following conditions occurs, the hardware clears this bit to "0":

1. When the data packet has been sent;

2. Automatic setting is enabled (AUTOSET = '1') and the FIFO data packet of endpoint IN reaches the value set by INMAXP;

3. If the endpoint is in synchronous mode and ISOUD is "1", the read value of IPRDY is always 0 before receiving the next SOF.

When you need to get/set this information, you must first use INDEX to select the target endpoint 1~5

21.2.14 IN endpoint control status register 2 (INCSR2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INCSR2	12H	AUTOSET	ISO	MODE	ENDMA	FCDT	-	-	-

AUTOSET: Automatically set the IPRDY flag control bit.

0: prohibit automatic setting of IPRDY flag

1: Enable automatic setting of IPRDY (the data loaded into the IN FIFO must reach the value set by INMAXP, otherwise the IPRDY flag must be manually set)

ISO: Synchronous transmission enable.

0: The endpoint is configured for bulk/interrupt transmission

1: The endpoint is configured for synchronous transmission

MODE: Endpoint direction selection bit.

0: Select the endpoint direction as OUT

1: Select the end direction as IN

ENDMA: DMA control of IN endpoint

0: Disable DMA request of IN endpoint

1: Enable DMA request for IN endpoint

FCDT: Force DATA0/DATA1 data switching settings.

0: Endpoint data is only switched after sending a data packet and receiving ACK.

1: Endpoint data is forced to switch after sending a data packet, regardless of whether it receives ACK.

When you need to get/set this information, you must first use INDEX to select the target endpoint 1~5

21.2.15 Maximum packet size of OUT endpoint (OUTMAXP)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
OUTMAXP	13H	OUTMAXP[7:0]							

OUTMAXP[7:0]: Set the maximum data packet size of the USB OUT endpoint

When you need to get/set this information, you must first use INDEX to select the target endpoint 1~5

21.2.16 OUT endpoint control status register 1 (OUTCSR1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR1	14H	CLRDT	STSTL	SDSTL	FLUSH	DATERR	OVRRUN	FIFOFUL	OPRDY

CLRDT (Clear Data Toggle): Reset the OUT data toggle bit.

When the OUT endpoint needs to reset the data switch bit to "0" due to reconfiguration or STALL, the software needs to write "1" to this data bit.

STSTL (Sent Stall): STALL signal transmission completion flag.

When the STALL signal is sent, the hardware will set this bit to "1". This flag must be cleared to "0" by software.
SDSTL (Send Stall): STALL signal sending request bit.

The software should write "1" to this bit to generate STALL signal as a response to an OUT token. Software should write "0" to this bit to end the STALL signal. This bit has no effect on ISO mode.

FLUSH (FIFO Flush): Clear the next data packet of the FIFO of the OUT endpoint.

Writing "1" to this bit will clear the next data packet from the OUT endpoint FIFO. The FIFO pointer is reset and the OPRDY bit is cleared to "0". If the FIFO contains multiple data packets, the software must write "1" to FLUSH for each data packet. When the FIFO is empty,

The hardware clears the FLUSH bit to "0".

DATAERR (Data Error): data error.

In ISO mode, this bit is set to '1' by hardware if the received packet has a CRC or bit stuffing error. When software clears OPRDY, this bit is cleared to '0'. This bit is only valid in ISO mode.

OVRUN (Data Overrun): Data overflow.

When an input packet cannot be loaded into the OUT endpoint FIFO, this bit is set to "1" by hardware. This bit is only valid in ISO mode. This bit must be cleared to "0" by software.

0: No data overflow

1: Since the flag was cleared for the last time, data packets have been lost due to full FIFO

FIFOFUL (FIFO Full): FIFO data full flag of OUT endpoint.

0: The FIFO of the OUT endpoint is not full

1: FIFO of OUT endpoint is full

OPRDY (OUT Packet Ready): OUT packet receiving completion flag.

The hardware sets this bit to "1" when there are data packets available. Software should clear this bit to '0' after unloading each data packet from the OUT endpoint FIFO.

When you need to get/set this information, you must first use INDEX to select the target endpoint 1~5

21.2.17 OUT endpoint control status register 2 (OUTCSR2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR2	15H	AUTOCLR	ISO	~~ENDMA~~	~~DMAMD~~	-	-	-	-

AUTOCLR: Automatically clear the OPRDY flag control bit.

0: Prohibit automatic clearing of OPRDY flag

1: Enable automatic clearing of OPRDY (the data downloaded from OUT FIFO must reach the value set by OUTMAXP, otherwise the OPRDY flag must be cleared manually)

ISO: Synchronous transmission enable.

0: The endpoint is configured for bulk/interrupt transmission

1: The endpoint is configured for synchronous transmission

~~ENDMA: DMA control of OUT endpoint~~

~~0: Disable DMA request of OUT endpoint~~

~~1: Enable DMA request for OUT endpoint~~

~~DMAMD: Set the DMA mode of the OUT endpoint~~

When you need to get/set this information, you must first use INDEX to select the target endpoint 1~5

21.2.18 OUT length of USB endpoint 0 (COUNT0)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
COUNT0	16H	-	OUTCNT0[6:0]						

OUTCNT0[6:0]: OUT byte length of endpoint 0

COUNT0 is dedicated to save the data length of the last OUT packet received by endpoint 0 (because the longest packet of endpoint 0 can only be 64 bytes, only 7 bits are needed). This length value is only valid when the OPRDY bit of endpoint 0 is "1".

When you need to obtain this length information, you must first use INDEX to select the target endpoint 0

21.2.19 OUT length of USB endpoint (OUTCOUNTn))

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
OUTCOUNT1	16H	OUTCNT[7:0]							
OUTCOUNT2	17H	-	-	-	-	-	OUTCNT[10:8]		

OUTCNT[10:0]: OUT byte length of the endpoint

OUTCOUNT1 and OUTCOUNT2 are combined to form an 11-bit number, which saves the data length of the last OUT packet, which is suitable for endpoints 1~5. This length value is only valid when the OPRDY bit of endpoint 1~5 is "1".

When you need to obtain this length information, you must first use INDEX to select the target endpoint 1~5

21.2.20 FIFO Data Access Registers (FIFO_n) of USB Endpoint

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
FIFO0	20H	FIFO0[7:0]							
FIFO1	21H	FIFO1[7:0]							
FIFO2	22H	FIFO2[7:0]							
FIFO3	23H	FIFO3[7:0]							
FIFO4	24H	FIFO4[7:0]							
FIFO5	25H	FIFO5[7:0]							

FIFO_n[7:0]: Indirect access register of IN/OUT data of each endpoint of USB

21.3 USB Product Development Considerations

Each USB product must have its own unique VID&PID combination in order to be correctly recognized by the computer. If the VID&PID combinations corresponding to two different USB products are the same, the computer may not recognize the USB product abnormally, so that the USB product cannot be used normally. To avoid this situation, both VID and PID need to be uniformly planned and assigned through formal ways.

At present, STC has obtained the VID number 13503 (hexadecimal: 34BF) of STC's dedicated USB device through the USB-IF organization. When customers use STC's USB chip to develop their own USB products, if you have obtained your own VID through other ways, the corresponding PID can be planned by yourself. If your USB product needs to use the official VID of STC, you must apply to STC for the PID of the product.

21.4 Sample Routines

21.4.1 Example of HID man-machine interface equipment

C Language Code

//Operating frequency for test is 11.0592MHz

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

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

```
typedef unsigned char  BYTE;
typedef unsigned int   WORD;
typedef unsigned long   DWORD;
```

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

```
sfr      USBCLK        =    0xDC;
```

```
sfr      USBDAT        =    0xEC;
```

```
sfr      USBCON        =    0xF4;
```

```
sfr      USBADR        =    0xFC;
```

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

```
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;
```

```
#define IRC48MCR        (*(unsigned char volatile xdata *)0xfe07)
```

```
#define FADDR           0
```

```
#define POWER           1
```

```
#define INTRIN1         2
```

```
#define EP5INIF         0x20
```

```
#define EP4INIF         0x10
```

```
#define EP3INIF         0x08
```

```
#define EP2INIF         0x04
```

```
#define EP1INIF         0x02
```

```
#define EP0IF           0x01
```

```
#define INTROUT1        4
```

```
#define EP5OUTIF        0x20
```

```
#define EP4OUTIF        0x10
```

```
#define EP3OUTIF        0x08
```

```
#define EP2OUTIF        0x04
```



```
#define EP1OUTIF      0x02
#define INTRUSB      6
#define SOFIF        0x08
#define RSTIF        0x04
#define RSUIF        0x02
#define SUSIF        0x01
#define INTRIN1E     7
#define EP5INIE      0x20
#define EP4INIE      0x10
#define EP3INIE      0x08
#define EP2INIE      0x04
#define EP1INIE      0x02
#define EP0IE        0x01
#define INTROUT1E    9
#define EP5OUTIE     0x20
#define EP4OUTIE     0x10
#define EP3OUTIE     0x08
#define EP2OUTIE     0x04
#define EP1OUTIE     0x02
#define INTRUSBE     11
#define SOFIE        0x08
#define RSTIE        0x04
#define RSUIE        0x02
#define SUSIE        0x01
#define FRAME1       12
#define FRAME2       13
#define INDEX        14
#define INMAXP       16
#define CSR0         17
#define SSUEND       0x80
#define SOPRDY       0x40
#define SDSTL        0x20
#define SUEND        0x10
#define DATEND       0x08
#define STSTL        0x04
#define IPRDY        0x02
#define OPRDY        0x01
#define INCSR1       17
#define INCLRDT      0x40
#define INSTSTL      0x20
#define INSDSTL      0x10
#define INFLUSH      0x08
#define INUNDRUN     0x04
#define INFIFONE     0x02
#define INIPRDY      0x01
#define INCSR2       18
#define INAUTOSET    0x80
#define INISO        0x40
#define INMODEIN     0x20
#define INMODEOUT    0x00
#define INENDMA      0x10
#define INFCDT       0x08
#define OUTMAXP      19
#define OUTCSR1      20
#define OUTCLRDT     0x80
#define OUTSTSTL     0x40
#define OUTSDSTL     0x20
#define OUTFLUSH     0x10
```

```

#define OUTDATERR      0x08
#define OUTOVERRUN     0x04
#define OUTFIFOFUL     0x02
#define OUTOPRDY       0x01
#define OUTCSR2        21
#define OUTAUTOCLR     0x80
#define OUTISO         0x40
#define OUTENDMA       0x20
#define OUTDMAMD       0x10
#define COUNT0        22
#define OUTCOUNT1    22
#define OUTCOUNT2    23
#define FIFO0         32
#define FIFO1         33
#define FIFO2         34
#define FIFO3         35
#define FIFO4         36
#define FIFO5         37
#define UTRKCTL       48
#define UTRKSTS       49

#define EPIDLE        0
#define EPSTATUS      1
#define EPDATAIN      2
#define EPDATAOUT     3
#define EPSTALL       -1

#define GET_STATUS    0x00
#define CLEAR_FEATURE 0x01
#define SET_FEATURE   0x03
#define SET_ADDRESS   0x05
#define GET_DESCRIPTOR 0x06
#define SET_DESCRIPTOR 0x07
#define GET_CONFIG    0x08
#define SET_CONFIG    0x09
#define GET_INTERFACE 0x0A
#define SET_INTERFACE 0x0B
#define SYNCH_FRAME   0x0C

#define GET_REPORT     0x01
#define GET_IDLE       0x02
#define GET_PROTOCOL   0x03
#define SET_REPORT     0x09
#define SET_IDLE       0x0A
#define SET_PROTOCOL   0x0B

#define DESC_DEVICE    0x01
#define DESC_CONFIG    0x02
#define DESC_STRING    0x03
#define DESC_HIDREPORT 0x22

#define STANDARD_REQUEST 0x00
#define CLASS_REQUEST   0x20
#define VENDOR_REQUEST  0x40
#define REQUEST_MASK    0x60

```

```

typedef struct
{

```

```

    BYTE    bmRequestType;
    BYTE    bRequest;
    BYTE    wValueL;
    BYTE    wValueH;
    BYTE    wIndexL;
    BYTE    wIndexH;
    BYTE    wLengthL;
    BYTE    wLengthH;
}SETUP;

typedef struct
{
    BYTE    bStage;
    WORD    wResidue;
    BYTE    *pData;
}EP0STAGE;

void UsbInit();
BYTE ReadReg(BYTE addr);
void WriteReg(BYTE addr, BYTE dat);
BYTE ReadFifo(BYTE fifo, BYTE *pdat);
void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt);

char code DEVICEDESC[18];
char code CONFIGDESC[41];
char code HIDREPORTDESC[27];
char code LANGIDDESC[4];
char code MANUFACTDESC[8];
char code PRODUCTDESC[30];

SETUP Setup;
EP0STAGE Ep0Stage;
BYTE xdata HidFeature[64];
BYTE xdata HidInput[64];
BYTE xdata HidOutput[64];

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;

    UsbInit();

    IE2 = 0x80;
    EA = 1;

    while (1);
}

```

BYTE ReadReg(BYTE addr)

```
{  
    BYTE dat;  
  
    while (USBADR & 0x80);  
    USBADR = addr | 0x80;  
    while (USBADR & 0x80);  
    dat = USBDAT;  
    return dat;  
}
```

void WriteReg(BYTE addr, BYTE dat)

```
{  
    while (USBADR & 0x80);  
    USBADR = addr & 0x7f;  
    USBDAT = dat;  
}
```

*BYTE ReadFifo(BYTE fifo, BYTE *pdat)*

```
{  
    BYTE cnt;  
    BYTE ret;  
  
    ret = cnt = ReadReg(COUNT0);  
    while (cnt--)  
    {  
        *pdat++ = ReadReg(fifo);  
    }  
    return ret;  
}
```

*void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt)*

```
{  
    while (cnt--)  
    {  
        WriteReg(fifo, *pdat++);  
    }  
}
```

void UsbInit()

```
{  
    P3M0 = 0x00;  
    P3M1 = 0x03;  
  
    P_SW2 |= 0x80;  
    IRC48MCR = 0x80;  
    while (!(IRC48MCR & 0x01));  
    P_SW2 &= ~0x80;  
    USBCLK = 0x00;  
    USBCON = 0x90;  
  
    WriteReg(FADDR, 0x00);  
    WriteReg(POWER, 0x08);  
    WriteReg(INTRIN1E, 0x3f);  
    WriteReg(INTROUT1E, 0x3f);  
    WriteReg(INTRUSBE, 0x00);  
    WriteReg(POWER, 0x01);  
}
```

```

    Ep0Stage.bStage = EPIDLE;
}

void usb_isr() interrupt 25
{
    BYTE intrusb;
    BYTE intrin;
    BYTE introut;
    BYTE csr;
    BYTE cnt;
    WORD len;

    intrusb = ReadReg(INTRUSB);
    intrin = ReadReg(INTRIN1);
    introut = ReadReg(INTROUT1);

    if (intrusb & RSTIF)
    {
        WriteReg(INDEX, 1);
        WriteReg(INCSR1, INCLRDT);
        WriteReg(INDEX, 1);
        WriteReg(OUTCSR1, OUTCLRDT);
        Ep0Stage.bStage = EPIDLE;
    }

    if (intrin & EP0IF)
    {
        WriteReg(INDEX, 0);
        csr = ReadReg(CSR0);
        if (csr & STSTL)
        {
            WriteReg(CSR0, csr & ~STSTL);
            Ep0Stage.bStage = EPIDLE;
        }
        if (csr & SUEND)
        {
            WriteReg(CSR0, csr | SSUEND);
        }

        switch (Ep0Stage.bStage)
        {
        case EPIDLE:
            if (csr & OPRDY)
            {
                Ep0Stage.bStage = EPSTATUS;
                ReadFifo(FIFO0, (BYTE *)&Setup);
                ((BYTE *)&Ep0Stage.wResidue)[0] = Setup.wLengthH;
                ((BYTE *)&Ep0Stage.wResidue)[1] = Setup.wLengthL;
                switch (Setup.bmRequestType & REQUEST_MASK)
                {
                case STANDARD_REQUEST:
                    switch (Setup.bRequest)
                    {
                    case SET_ADDRESS:
                        WriteReg(FADDR, Setup.wValueL);
                        break;
                    case SET_CONFIG:

```

```

    WriteReg(INDEX, 1);
    WriteReg(INCSR2, INMODEIN);
    WriteReg(INMAXP, 8);
    WriteReg(INDEX, 1);
    WriteReg(INCSR2, INMODEOUT);
    WriteReg(OUTMAXP, 8);
    WriteReg(INDEX, 0);
    break;
case GET_DESCRIPTOR:
    Ep0Stage.bStage = EPDATAIN;
    switch (Setup.wValueH)
    {
    case DESC_DEVICE:
        Ep0Stage.pData = DEVIDDESC;
        len = sizeof(DEVIDDESC);
        break;
    case DESC_CONFIG:
        Ep0Stage.pData = CONFIGDESC;
        len = sizeof(CONFIGDESC);
        break;
    case DESC_STRING:
        switch (Setup.wValueL)
        {
        case 0:
            Ep0Stage.pData = LANGIDDESC;
            len = sizeof(LANGIDDESC);
            break;
        case 1:
            Ep0Stage.pData = MANUFACTDESC;
            len = sizeof(MANUFACTDESC);
            break;
        case 2:
            Ep0Stage.pData = PRODUCTDESC;
            len = sizeof(PRODUCTDESC);
            break;
        default:
            Ep0Stage.bStage = EPSTALL;
            break;
        }
        break;
    case DESC_HIDREPORT:
        Ep0Stage.pData = HIDREPORTDESC;
        len = sizeof(HIDREPORTDESC);
        break;
    default:
        Ep0Stage.bStage = EPSTALL;
        break;
    }
    if (len < Ep0Stage.wResidue)
    {
        Ep0Stage.wResidue = len;
    }
    break;
default:
    Ep0Stage.bStage = EPSTALL;
    break;
}
break;

```

```

    case CLASS_REQUEST:
        switch (Setup.bRequest)
        {
            case GET_REPORT:
                Ep0Stage.pData = HidFreature;
                Ep0Stage.bStage = EPDATAIN;
                break;
            case SET_REPORT:
                Ep0Stage.pData = HidFreature;
                Ep0Stage.bStage = EPDATAOUT;
                break;
            case SET_IDLE:
                break;
            case GET_IDLE:
            case GET_PROTOCOL:
            case SET_PROTOCOL:
            default:
                Ep0Stage.bStage = EPSTALL;
                break;
        }
        break;
    default:
        Ep0Stage.bStage = EPSTALL;
        break;
}

switch (Ep0Stage.bStage)
{
    case EPDATAIN:
        WriteReg(CSR0, SOPRDY);
        goto L_Ep0SendData;
        break;
    case EPDATAOUT:
        WriteReg(CSR0, SOPRDY);
        break;
    case EPSTATUS:
        WriteReg(CSR0, SOPRDY | DATEND);
        Ep0Stage.bStage = EPIDLE;
        break;
    case EPSTALL:
        WriteReg(CSR0, SOPRDY | SDSTL);
        Ep0Stage.bStage = EPIDLE;
        break;
}
}
break;
case EPDATAIN:
    if (!(csr & IPRDY))
    {
L_Ep0SendData:
        cnt = Ep0Stage.wResidue > 64 ? 64 : Ep0Stage.wResidue;
        WriteFifo(FIFO0, Ep0Stage.pData, cnt);
        Ep0Stage.wResidue -= cnt;
        Ep0Stage.pData += cnt;
        if (Ep0Stage.wResidue == 0)
        {
            WriteReg(CSR0, IPRDY | DATEND);
            Ep0Stage.bStage = EPIDLE;

```

```

        }
        else
        {
            WriteReg(CSR0, IPRDY);
        }
    }
    break;
case EPDATAOUT:
    if (csr & OPRDY)
    {
        cnt = ReadFifo(FIFO0, Ep0Stage.pData);
        Ep0Stage.wResidue -= cnt;
        Ep0Stage.pData += cnt;
        if (Ep0Stage.wResidue == 0)
        {
            WriteReg(CSR0, SOPRDY | DATEND);
            Ep0Stage.bStage = EPIDLE;
        }
        else
        {
            WriteReg(CSR0, SOPRDY);
        }
    }
    break;
}

if (intrin & EP1INIF)
{
    WriteReg(INDEX, 1);
    csr = ReadReg(INCSR1);
    if (csr & INSTSTL)
    {
        WriteReg(INCSR1, INCLRDT);
    }
    if (csr & INUNDRUN)
    {
        WriteReg(INCSR1, 0);
    }
}

if (introut & EP1OUTIF)
{
    WriteReg(INDEX, 1);
    csr = ReadReg(OUTCSR1);
    if (csr & OUTSTSTL)
    {
        WriteReg(OUTCSR1, OUTCLRDT);
    }
    if (csr & OUTOPRDY)
    {
        ReadFifo(FIFO1, HidOutput);
        WriteReg(OUTCSR1, 0);

        WriteReg(INDEX, 1);
        WriteFifo(FIFO1, HidOutput, 64);
        WriteReg(INCSR1, INIPRDY);
    }
}

```



```

    }
}

char code DEVICEDESC[18] =
{
    0x12,                //bLength(18);
    0x01,                //bDescriptorType(Device);
    0x00,0x02,           //bcdUSB(2.00);
    0x00,                //bDeviceClass(0);
    0x00,                //bDeviceSubClass(0);
    0x00,                //bDeviceProtocol(0);
    0x40,                //bMaxPacketSize0(64);
    0xbf,0x34,           //idVendor(34bf);
    0x01,0xf0,           //idProduct(f001);
    0x00,0x01,           //bcdDevice(1.00);
    0x01,                //iManufacturer(1);
    0x02,                //iProduct(2);
    0x00,                //iSerialNumber(0);
    0x01,                //bNumConfigurations(1);
};

char code CONFIGDESC[41] =
{
    0x09,                //bLength(9);
    0x02,                //bDescriptorType(Configuration);
    0x29,0x00,           //wTotalLength(41);
    0x01,                //bNumInterfaces(1);
    0x01,                //bConfigurationValue(1);
    0x00,                //iConfiguration(0);
    0x80,                //bmAttributes(BUSPower);
    0x32,                //MaxPower(100mA);

    0x09,                //bLength(9);
    0x04,                //bDescriptorType(Interface);
    0x00,                //bInterfaceNumber(0);
    0x00,                //bAlternateSetting(0);
    0x02,                //bNumEndpoints(2);
    0x03,                //bInterfaceClass(HID);
    0x00,                //bInterfaceSubClass(0);
    0x00,                //bInterfaceProtocol(0);
    0x00,                //iInterface(0);

    0x09,                //bLength(9);
    0x21,                //bDescriptorType(HID);
    0x01,0x01,           //bcdHID(1.01);
    0x00,                //bCountryCode(0);
    0x01,                //bNumDescriptors(1);
    0x22,                //bDescriptorType(HID Report);
    0x1b,0x00,           //wDescriptorLength(27);

    0x07,                //bLength(7);
    0x05,                //bDescriptorType(Endpoint);
    0x81,                //bEndpointAddress(EndPoint1 as IN);
    0x03,                //bmAttributes(Interrupt);
    0x40,0x00,           //wMaxPacketSize(64);
    0x01,                //bInterval(10ms);

    0x07,                //bLength(7);

```

```

        0x05,                                //bDescriptorType(Endpoint);
        0x01,                                //bEndpointAddress(EndPoint1 as OUT);
        0x03,                                //bmAttributes(Interrupt);
        0x40,0x00,                            //wMaxPacketSize(64);
        0x01,                                //bInterval(10ms);
};

char code HIDREPORTDESC[27] =
{
    0x05,0x0c,                                //USAGE_PAGE(Consumer);
    0x09,0x01,                                //USAGE(Consumer Control);
    0xa1,0x01,                                //COLLECTION(Application);
    0x15,0x00,                                // LOGICAL_MINIMUM(0);
    0x25,0xff,                                // LOGICAL_MAXIMUM(255);
    0x75,0x08,                                // REPORT_SIZE(8);
    0x95,0x40,                                // REPORT_COUNT(64);
    0x09,0x01,                                // USAGE(Consumer Control);
    0xb1,0x02,                                // FEATURE(Data,Variable);
    0x09,0x01,                                // USAGE(Consumer Control);
    0x81,0x02,                                // INPUT(Data,Variable);
    0x09,0x01,                                // USAGE(Consumer Control);
    0x91,0x02,                                // OUTPUT(Data,Variable);
    0xc0,                                //END_COLLECTION;
};

char code LANGIDDESC[4] =
{
    0x04,0x03,
    0x09,0x04,
};

char code MANUFACTDESC[8] =
{
    0x08,0x03,
    'S',0,
    'T',0,
    'C',0,
};

char code PRODUCTDESC[30] =
{
    0x1e,0x03,
    'S',0,
    'T',0,
    'C',0,
    ' ',0,
    'U',0,
    'S',0,
    'B',0,
    ' ',0,
    'D',0,
    'e',0,
    'v',0,
    'i',0,
    'c',0,
    'e',0,
};

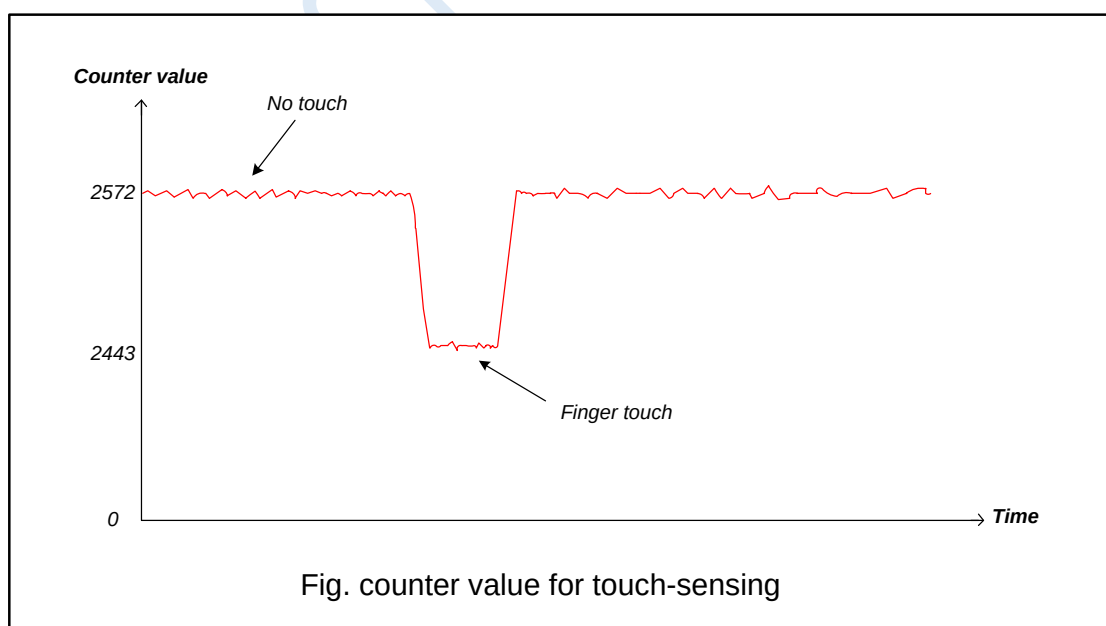
```

22 Touch Key Controller

Product line	Touch key
STC8H1K08 family	
STC8H1K28 family	
STC8H3K64S4 family	
STC8H3K64S2 family	
STC8H8K64U family	
STC8H2K64T family	●
STC8H4K64TLR family	●
STC8H4K64TLCD family	●
STC8H4K64LCD family	

A touch key controller or touch sensor unit (TSU in short) is integrated in the STC8H series of microcontrollers, which can connect up to 16 keys. It can detect the small capacitance changes caused by a finger touching the key electrode and quantify the capacitance as a 16-bit number. In principle, the TSU module is similar to a 16-bit ADC. The only difference is that the ADC detects and quantifies the simulated voltage or current, while the TSU detects and quantifies the capacitor. For touch-key sensing, a fixed capacitor of about 10nF to 47nF needs to be added for application as a reference. If the temperature of the environment changes rapidly and intensely, a low-temperature drift capacitor should be used to maintain the TSU output data no too much change, so as to avoid software misjudgment.

A description of continuous sensing of a certain key is shown in the following figure. The counter output value is near 2572 if there is no finger touch. The counter output becomes near 2443 if there is a finger touch. The difference of counter output value with or without finger touch is about 5%.



The TSU module can connect up to 16 keys. Each touch key port can be independently enabled through the two 8-bit registers TSCHEN1 and TSCHEN2. The I/O port not used as touch key function can still maintain its original GPIO or special functions such as LED driver. The frequency of the switched capacitor circuit is selected through SCR [2: 0]. It is recommended not to exceed 12.5MHz to avoid large errors. There are four reference voltage

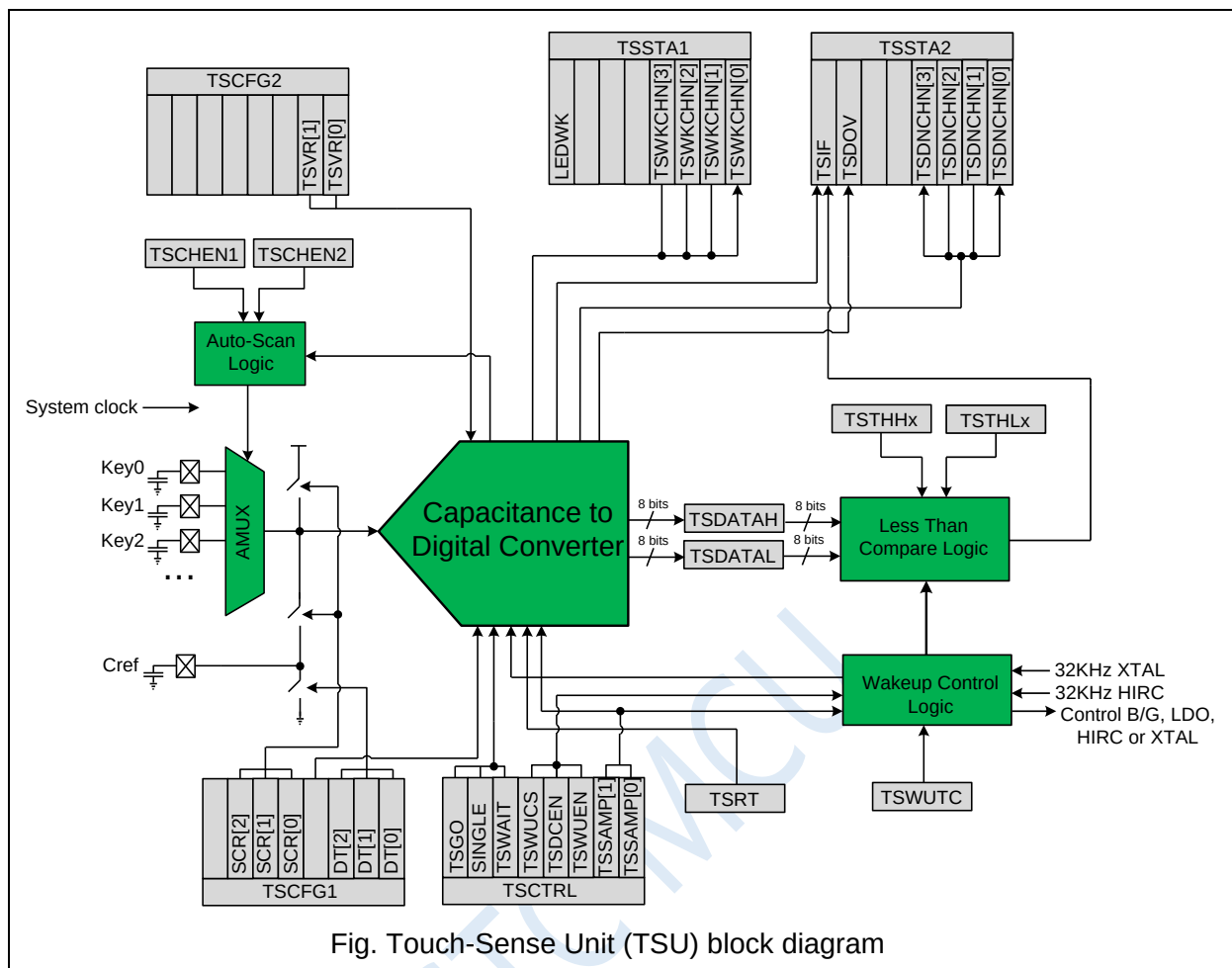
segments of the internal comparator of the TSU module for choosing, which is selected through TSVR [1:0]. Changing the reference voltage will change the touch sensing time and sensitivity.

The touch key scan can be configured to scan continuously or to stop after one round of the enabled key. This function is controlled by the SINGLE bit. The configuration register TSSAMP [1: 0] allows the TSU module to scan four times continuously to one channel to sample and calculate the average value of the data. To do this has the effect of hardware filtering. The WAIT bit allows the TSU module to enter the wait mode when the TSIF flag is 1. TSU will continue to perform key scans until the software clears the WAIT bit to 0, which helps the heavy CPU to have enough time to process other things. The value of the external capacitor Cref is recommended to be in the range of 10nF to 47nF. The time from the initial value of Cref discharging to zero must be sufficient. This can be adjusted flexibly through DT [2: 0].

When a key scan is completed, the output value of the 16-bit counter will be written to TSDATAH and TSDATAL. The flag bit TSIF is set to 1 by hardware, and the scanned touch channel number will be written to TSDNCHN [3: 0]. If the module's external interrupt controller is enabled, TSIF can request interrupt to the CPU. The software may read the TSDNCHN [3: 0] register content to determine which touch channel requested the TSIF interrupt. You can get the status of the TSU module being scanned and the number of keys being scanned in real time by reading TSWKCHN [3: 0] and TSGO using software. If the 16-bit counter overflows, the TSDOV flag will be set to 1 by hardware.

The TSU module can use I/O port in time-sharing multiplexing with the LED driver circuit. It is necessary to enable the LED driving circuit together and time-multiplex the I/O port when the content of the TSRT register is not zero, which means that TSGO enables TSU. Therefore, the relevant registers of the TSU module and the LED driving circuit must be configured in order firstly before TSGO is enabled. In TSU / LED time-sharing multi-tasking mode, in order to maintain the fixed frame rate of the LED, if the time allocated to the key scan has arrived but the key being scanned has not been completed, the incomplete key will be scanned to start a new key scan.

22.1 Internal Structure Diagram of Touch Key Controller



22.2 Wake-up from Low Power Mode using Touch Key

The TSU module has a dedicated timing and control circuit, which can accept an external 32KHz crystal or internal 32KHz RC oscillation circuit as a clock source, and wake up the TSU module at regular intervals to perform key scans, and realize the low-power touch wake-up function by duty control. There is a dedicated 16-bit threshold register {TSTHHx, TSTHLx} in each touch channel. If the wake-up enable bit TSWUEN and the digital comparator enable bit TSDCEN are set, MCU will enter the power-down state, and the entire chip enters a low-power state. In touch wake-up mode, the TSU module can repeatedly and regularly self-wake for key scans to wake up the CPU. If the data (or average result) of the key scan is less than the set threshold, the hardware will set TSIF to 1 and wake up the CPU out of power-down state. There is a hardware averaging circuit inside the TSU module, which can average the maximum of four consecutive scans of the same channel. The TSSAMP [1: 0] register is used to configure the number of samples. {TSDATAH, TSDATAL} stores average result.

22.3 Operation Steps When Touch Key Function is Used only

1. Turn on the TSU power switch TSPD = 0, select the channel to be scanned, and the registers are TSCHEN1 and TSCHEN2.
2. Set the TSRT content to 0x00, which means that the LED driver time-sharing multi-tasking function is not enabled.
3. Configure the switching frequency SCR [2: 0] and discharge time DT [2: 0] according to the value of Cref and the touch key capacitor, and select the internal comparator reference voltage TSVR [1: 0] according to the required scan time and sensitivity.

4. Configure the SINGLE bit to determine whether the scan is stopped automatically or continuously. Configure TSSAMP [1: 0] to allow a channel to be resampled up to four times. If the CPU task is heavy, configure TSWAIT to use the TSIF status to delay scanning of the next channel.
5. Configure TSDCEN to enable internal digital comparison function if necessary.
6. Set TSGO=1 to start the touch key scanning. You can read TSWKCHN [3: 0] to know which channel is currently being scanned using software. After finish scanning a channel, the hardware will set TSIF to 1, and the completed channel number will be written into TSDNCHN [3: 0]. If an overflow occurs, TSOV will also be set to 1. You should read these registers to decide what to do next in your software. TSIF and TSOV can only be set by hardware and cleared by software.
7. If SINGLE = 1, the hardware will clear TSGO automatically and end scanning after one round of scanning, otherwise TSGO will remain at 1 and continue the new scanning.
8. If you want to stop the touch key scanning, you can set TSGO to 0 at any time. If you want to reduce power consumption, you must set TSPD to 1.

22.4 Operation Steps for Wake-up from Low Power Mode using Touch Key

1. Select the channel to be scanned, the registers are TSCHEN1 and TSCHEN2.
2. Be sure to set the TSRT content to 0x00. At this time, you cannot turn on the LED driver time-sharing multitasking function.
3. Configure switching frequency SCR [2: 0], discharge time DT [2: 0] and select internal comparator reference voltage TSVR [1: 0].
4. Set the SINGLE bit to 0 for continuous scanning and set TSWAIT to 0.
5. Configure TSWUCS to determine which clock source is used to wake up the controller, external 32KHz crystal or internal 32KHz IRC.
6. Configure TSWUTC to determine how often TSU needs to work and enter the power saving mode automatically after work.
7. Configure TSSAMP [1: 0] to determine the number of scan samples for each channel, configure TSDCEN = 1 to enable the internal digital comparison function.
8. Configure the wake-up threshold {TSTHHx, TSTHLx} for each channel, which will be compared with the average of the scan results.
9. Enable TSWUEN = 1, set TSPD = 1 to turn off the analog power of the TSU module, enable TSIF to wake up the CPU, then let the MCU enter the power-down state. Once the MCU enters the power-down state, the wake-up controller inside the TSU starts to work, and controls the power switch, key scan, data comparison, and so on of the TSU module periodically.
10. If the data result is lower than the set threshold, the hardware will set TSIF to 1, write the key number in TSDNCHN [3: 0], and the CPU will be woken up, the low power wake-up process ends.
11. After the CPU is awakened, in addition to directly reading TSDNCHN [3: 0] to determine which key is touched, you can also perform a key scan in the normal working mode to confirm whether the wake-up is caused by noise interference.

22.5 Registers Related to Touch Key Controller

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
TSCHEN1	Touch Key Enable Register 1	FB40H	TKEN7	TKEN6	TKEN5	TKEN4	TKEN3	TKEN2	TKEN1	TKEN0	0000,0000
TSCHEN2	Touch Key Enable Register 2	FB41H	TKEN15	TKEN14	TKEN13	TKEN12	TKEN11	TKEN10	TKEN9	TKEN8	0000,0000
TSCFG1	Touch Key Configuration Register 1	FB42H	-	SCR[2:0]			-	DT[2:0]			0000,0000
TSCFG2	Touch Key Configuration Register 2	FB43H	-	-	-	-	-	-	TSVR[1:0]		0000,0000
TSWUTC	Touch key power-down mode wake-up time control register	FB44H									0000,0000
TSCTRL	Touch Key Control Register	FB45H	TSGO	SINGLE	TSWAIT	TSWUCS	TSDCEN	TSWUEN	TSSAMP[1:0]		0000,0000
TSSTA1	Touch Key Status Register 1	FB46H	LEDWK	-	-	-	TSWKCHN[3:0]			0000,0000	
TSSTA2	Touch Key Status Register 2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]			0000,0000	

TSRT	Touch Key Time Control Register	FB48H		0000,0000
TSDATH	Touch Key Data High Byte	FB49H		0000,0000
TSDATL	Touch Key Data Low Byte	FB4AH		0000,0000
TSTH00H	Touch Key0 Threshold High Byte	FB50H		0000,0000
TSTH00L	Touch Key0 Threshold Low Byte	FB51H		0000,0000
TSTH01H	Touch Key1 Threshold High Byte	FB52H		0000,0000
TSTH01L	Touch Key1 Threshold Low Byte	FB53H		0000,0000
TSTH02H	Touch Key2 Threshold High Byte	FB54H		0000,0000
TSTH02L	Touch Key2 Threshold Low Byte	FB55H		0000,0000
TSTH03H	Touch Key3 Threshold High Byte	FB56H		0000,0000
TSTH03L	Touch Key3 Threshold Low Byte	FB57H		0000,0000
TSTH04H	Touch Key4 Threshold High Byte	FB58H		0000,0000
TSTH04L	Touch Key4 Threshold Low Byte	FB59H		0000,0000
TSTH05H	Touch Key5 Threshold High Byte	FB5AH		0000,0000
TSTH05L	Touch Key5 Threshold Low Byte	FB5BH		0000,0000
TSTH06H	Touch Key6 Threshold High Byte	FB5CH		0000,0000
TSTH06L	Touch Key6 Threshold Low Byte	FB5DH		0000,0000
TSTH07H	Touch Key7 Threshold High Byte	FB5EH		0000,0000
TSTH07L	Touch Key7 Threshold Low Byte	FB5FH		0000,0000
TSTH08H	Touch Key8 Threshold High Byte	FB60H		0000,0000
TSTH08L	Touch Key8 Threshold Low Byte	FB61H		0000,0000
TSTH09H	Touch Key9 Threshold High Byte	FB62H		0000,0000
TSTH09L	Touch Key9 Threshold Low Byte	FB63H		0000,0000
TSTH10H	Touch Key10 Threshold High Byte	FB64H		0000,0000
TSTH10L	Touch Key10 Threshold Low Byte	FB65H		0000,0000
TSTH11H	Touch Key11 Threshold High Byte	FB66H		0000,0000
TSTH11L	Touch Key11 Threshold Low Byte	FB67H		0000,0000
TSTH12H	Touch Key12 Threshold High Byte	FB68H		0000,0000
TSTH12L	Touch Key12 Threshold Low Byte	FB69H		0000,0000
TSTH13H	Touch Key13 Threshold High Byte	FB6AH		0000,0000
TSTH13L	Touch Key13 Threshold Low Byte	FB6BH		0000,0000
TSTH14H	Touch Key14 Threshold High Byte	FB6CH		0000,0000
TSTH14L	Touch Key14 Threshold Low Byte	FB6DH		0000,0000
TSTH15H	Touch Key15 Threshold High Byte	FB6EH		0000,0000
TSTH15L	Touch Key15 Threshold Low Byte	FB6FH		0000,0000

22.5.1 Touch Key Enable Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSCHEN1	FB40H	TKEN7	TKEN6	TKEN5	TKEN4	TKEN3	TKEN2	TKEN1	TKEN0
TSCHEN2	FB41H	TKEN15	TKEN14	TKEN13	TKEN12	TKEN11	TKEN10	TKEN9	TKEN8

TKENn: Touch key enable bit (n=0~15)

0: Corresponding TKn pin is GPIO

1: Corresponding TKn pin is a touch key

22.5.2 Touch Key Configuration Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSCFG1	FB42H	-	SCR[2:0]			-	DT[2:0]		
TSCFG2	FB43H	-	-	-	-	-	-	TSVR[1:0]	

SCR: Configure the switching capacitor working frequency inside the touch key controller (the higher the frequency, the shorter the charging time)

$$\text{Working frequency of the switching capacitor} = \frac{\text{System working frequency}}{2 * (\text{SCR}[2:0] + 1)}$$

DT[2:0]: Configure the initial discharge time of Cref inside the touch key controller to ground

DT[2:0]	discharge time
000	125 system clocks
001	250 system clocks
010	500 system clocks
011	1000 system clocks
100	2000 system clocks
101	2500 system clocks
110	5000 system clocks
111	7500 system clocks

TSCV[1:0]: Configure the reference voltage inside the touch key controller

TSCV[1:0]	reference voltage
00	1/4 AVCC
01	1/2 AVCC
10	5/8 AVCC
11	3/4 AVCC

22.5.3 Touch key power-down mode wake-up time control register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSWUTC	FB44H								

TSWUTC register is used to configure how often the touch key controller is woken up.

$$\text{Wake-up frequency} = \frac{F_{32K}}{32 * 8 * \text{TSWUTC}[7:0]}$$

For example: if a external 32.768KHz crystal is used and TSWUTC = 0x80,

Then the wake-up frequency of the touch key controller is $32768 / (32 * 8 * 0x80) = 1\text{Hz}$, that is to wake up once every 1 second.

Note: If the wake-up frequency is set too fast, the wake-up time is not enough to complete a round of key scan, and the touch key controller will scan continuously and cannot enter the power saving mode.

22.5.4 Touch Key Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSCTRL	FB45H	TSGO	SINGLE	TSWAIT	TSWUCS	TSDCEN	TSWUEN	TSSAMP[1:0]	

TSGO: Touch key controller start control bit

0: Touch key controller does not work.

1: Touch key controller start to work.

SINGLE: Single scan mode control bit

0: Repeat scan mode

1: Single scan mode. If TSGO=1, the hardware will clear TSGO to 0 automatically to terminate the scan after completing a round of key scan.

Note: IF TSGO = 1 and TSRT is not 0, it means that the touch button controller and LED driver share GPIO and time-sharing and multiplexing. At this time, the SINGLE control bit is invalid.

TSWAIT: Touch key controller waiting for control

0: touch key controller repeats scanning automatically

1: After completing one round of scanning, the TSIF is set to 1 by the hardware. At this time, the touch key controller will pause scanning until the TSIF flag is cleared to 0 and start the next round of scanning.

TSWUCS: clock source selection of touch key controller in low power mode

0: The clock source of the touch key controller in the low power mode is the internal 32K IRC.

1: The clock source of touch key controller in low power mode is external 32K crystal.

TSDCEN: 16-bit digital comparator inside the touch key controller control bit

0: disable 16-bit digital comparator inside the touch key controller

1: enable 16-bit digital comparator inside the touch key controller

Note: If the internal digital comparator of TSU is enabled, the TSIF will be set to 1 only when the touch sensing data result {TSDATAH, TSDATAL} is less than the threshold {TSTHHx, TSTHLx} set for the corresponding channel. This function is used to wake up the CPU by touch in low power mode.

TSWUEN: touch key controller low power wake up enable bit

0: disable touch key controller low power wake up function

1: enable touch key controller low power wake up function. After enabled, when the MCU enters the power-down state, it enters the touch-key low-power wake-up MCU mode immediately. In this mode, the low-power timing control circuit inside the touch key controller will enable the TSU periodically to perform key touch scanning. The duty control is used to maintain extremely low average current. TSWUEN is only effective when the MCU enters power-down mode and TSIF is 0.

Note: There are two 32K oscillators in this chip, one is an external 32K crystal, and the other is an internal IRC32K oscillator. In the case of non-STOP-mode, the internal IRC32K enable mechanism is simply using XFR: IRC32KCR [7], the external X32K enable mechanism is simply using XFR: X2KCR [7]. In the case of STOP-mode, the internal IRC32K enable mechanism is that in addition to XFR: IRC32KCR [7] must be set to 1, SFR: ENWKT or XFR: TSWUEN must also be set to 1, the external X32K enable mechanism is that in addition to XFR: X32KCR [7] must be set to 1, XFR: TSWUEN must also be set to 1. The key point to emphasize is that TSWUEN also plays the role of enabling or disabling the 32K oscillator in the case of STOP-mode.

TSSAMP[1:0]: single touch channel repeats scan times setting bits

TSSAMP [1:0]	Repeat scan times
00	once
01	2 times
10	3 times
11	4 times

Note: The interrupt flag TSIF will only be set when the number of scans of the same key reaches the configuration of TSSAMP. At this time, the average value of the results is written in {TSDATAH, TSDATAL}. However, if any overflow occurs, the hardware will set TSDOV to 1. With TSWKEN enabled, the average value must be less than the threshold to cause the interrupt flag TSIF to be set to 1 to wake the CPU.

22.5.5 Touch Key Status Register 1

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSSTA1	FB46H	LEDWK	-	-	-	TSWKCHN[3:0]			

LEDWK: working state of touch key controller and LED driver in time-sharing operation

0: The LED driver is in waiting state and the touch key controller is in working state.

1: The LED driver is in working state and the touch key controller is in waiting state.

TSWKCHN [3:0]: touch channel scan status

TSWKCHN [3:0]	Touch channel scan status
0000	Touch channel 0 is being scanned
0001	Touch channel 1 is being scanned
0010	Touch channel 2 is being scanned

0011	Touch channel 3 is being scanned
0100	Touch channel 4 is being scanned
0101	Touch channel 5 is being scanned
0110	Touch channel 6 is being scanned
0111	Touch channel 7 is being scanned
1000	Touch channel 8 is being scanned
1001	Touch channel 9 is being scanned
1010	Touch channel 10 is being scanned
1011	Touch channel 11 is being scanned
1100	Touch channel 12 is being scanned
1101	Touch channel 13 is being scanned
1110	Touch channel 14 is being scanned
1111	Touch channel 15 is being scanned

22.5.6 Touch Key Status Register 2

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSSTA2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]			

TSIF: touch key channel scanning completion flag (cleared by writing 1 using software)

0: Scan has not completed.

1: When the number of scans set by TSSAMP is completed, TSIF is set by the hardware, and TSIF can request an interrupt to the CPU. If it is in the low-power wake-up mode, the scan data result value is lower than the set threshold at the same time, then TSIF will be set to 1.

Note: TSIF can only be set by hardware. Software cannot set TSIF to 1. It is important to note that writing 1 to TSIF in software clears TSIF to 0, and writing 0 to TSIF in software has no effect. If TSWAIT = 1 and TSIF is 1, the TSU is in the pause and wait state. The next key scan will be continued after the CPU is busy finished and clears the TSIF to 0.

TSDOV: Key scan data overflow flag (cleared by writing 1 using software)

0: The key scan data does not overflow, the scan data is less than or equal to 0xFFFF.

1: The key scan data overflows, and the scan data is greater than 0xFFFF. At this time, extreme software configurations (such as TSVR) or system hardware must be adjusted to avoid overflow. TSDOV can only be set to 1 by hardware. It is cleared by software writing 1 to it. There is no effect on it if software writing 0 to it.

TSDNCHN [3:0]: Touch channel completion status

TSDNCHN [3:0]	Touch channel completion status
0000	Scanning of touch channel 0 is completed
0001	Scanning of touch channel 1 is completed
0010	Scanning of touch channel 2 is completed
0011	Scanning of touch channel 3 is completed
0100	Scanning of touch channel 4 is completed
0101	Scanning of touch channel 5 is completed
0110	Scanning of touch channel 6 is completed
0111	Scanning of touch channel 7 is completed
1000	Scanning of touch channel 8 is completed
1001	Scanning of touch channel 9 is completed
1010	Scanning of touch channel 10 is completed
1011	Scanning of touch channel 11 is completed
1100	Scanning of touch channel 12 is completed
1101	Scanning of touch channel 13 is completed

1110	Scanning of touch channel 14 is completed
1111	Scanning of touch channel 15 is completed

22.5.7 Touch Key Time Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSRT	FB48H								

The TSRT register is used to configure the touch key controller and the LED driver to work in time-sharing.

If TSRT is not 00, the touch keycontroller and LED driver are in the time-sharing mode. The length of working time for the touch key controller is $TSRT * T_{LED}$. (Please refer to the LED driver description section for T_{LED})

22.5.8 Touch key data registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSDATH	FB49H	TSDAT[15:8]							
TSDATL	FB4AH	TSDAT[7:0]							

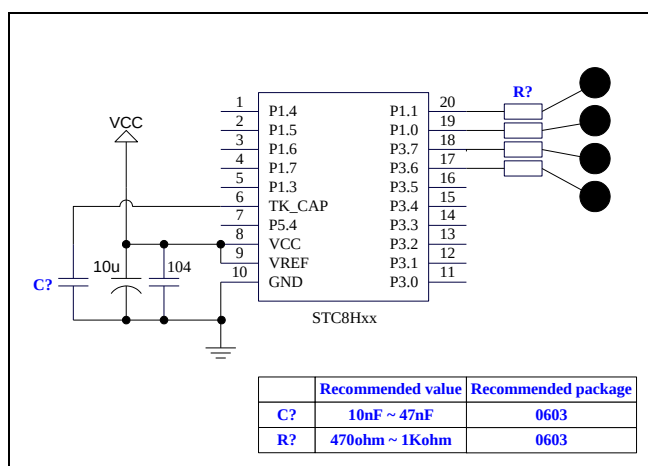
TSDAT[15:0]: Data scanned by touch key

22.5.9 Touch key threshold register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TSTH00H	FB50H	TSTH00[15:8]							
TSTH00L	FB51H	TSTH00[7:0]							
TSTH01H	FB52H	TSTH01[15:8]							
TSTH01L	FB53H	TSTH01[7:0]							
TSTH02H	FB54H	TSTH02[15:8]							
TSTH02L	FB55H	TSTH02[7:0]							
TSTH03H	FB56H	TSTH03[15:8]							
TSTH03L	FB57H	TSTH03[7:0]							
TSTH04H	FB58H	TSTH04[15:8]							
TSTH04L	FB59H	TSTH04[7:0]							
TSTH05H	FB5AH	TSTH05[15:8]							
TSTH05L	FB5BH	TSTH05[7:0]							
TSTH06H	FB5CH	TSTH06[15:8]							
TSTH06L	FB5DH	TSTH06[7:0]							
TSTH07H	FB5EH	TSTH07[15:8]							
TSTH07L	FB5FH	TSTH07[7:0]							
TSTH08H	FB60H	TSTH08[15:8]							
TSTH08L	FB61H	TSTH08[7:0]							
TSTH09H	FB62H	TSTH09[15:8]							
TSTH09L	FB63H	TSTH09[7:0]							
TSTH10H	FB64H	TSTH10[15:8]							
TSTH10L	FB65H	TSTH10[7:0]							
TSTH11H	FB66H	TSTH11[15:8]							
TSTH11L	FB67H	TSTH11[7:0]							
TSTH12H	FB68H	TSTH12[15:8]							
TSTH12L	FB69H	TSTH12[7:0]							
TSTH13H	FB6AH	TSTH13[15:8]							
TSTH13L	FB6BH	TSTH13[7:0]							
TSTH14H	FB6CH	TSTH14[15:8]							
TSTH14L	FB6DH	TSTH14[7:0]							
TSTH15H	FB6EH	TSTH15[15:8]							
TSTH15L	FB6FH	TSTH15[7:0]							

TSTHn[15:0]: Touch key scan data threshold. After the digital comparator is enabled, TSIF will be set to 1 by hardware only when the scan data is below this threshold.

22.6 Basic Reference Circuit and Precautions



Note: In the reference circuit diagram, C? is the sensitivity adjustment capacitor for touch keys, and R? is the ESD protection resistor. In PCB layout, C? and R? must be as close as possible to the IC pins.

22.7 Example Routines

22.7.1 Introduction of touch key configuration software

1. Serial port settings

"COM": select the serial port number;

"Baud rate": Set the baud rate of serial communication, the default is '115200, N, 8, 1'. If the main frequency of the MCU system is set below 5.5296MHz, the baud rate needs to be reduced for normal communication, and 9600 is recommended. The MCU code needs to modify the definition of "Baudrate" synchronously.

"Open": After setting the serial port parameters, click this button to open the serial port, and the button name becomes "Close"; click it again to close the serial port, and the button name becomes "Open".

STC触摸按键配置

COM: COM4 打开 波特率: 115200 主频: 22.1184MHz 设置 获取 发送配置 接收配置 退出

触摸按键选择(TSCHEN)

☐ TK00 ☐ TK01 ☐ TK02 ☐ TK03 ☐ TK04 ☐ TK05 ☐ TK06 ☐ TK07

差值:

☐ TK08 ☐ TK09 ☐ TK10 ☐ TK11 ☐ TK12 ☐ TK13 ☐ TK14 ☐ TK15

差值:

配置参数(TSCFG)

开关电容工作频率: 系统工作频率/(2*(7+1))

放电时间: 7500个系统时钟

内部参考电压: 5/8 AVcc

触摸控制(TSCTRL)

☒ 触摸启动

扫描模式: 重复扫描模式 等待控制: 自动重复扫描

休眠时钟: 内部32K IRC 数字比较器: 关闭内部16位比较器

休眠唤醒: 关闭触控唤醒功能 重复扫描: 单通道重复扫描1次

自定义

触摸信号持续检测时间: 20 ms

唤醒时间控制(TSWUTC) 唤醒频率=F32K/(32*8*TSWUTC)

唤醒时间间隔(0~255): 0

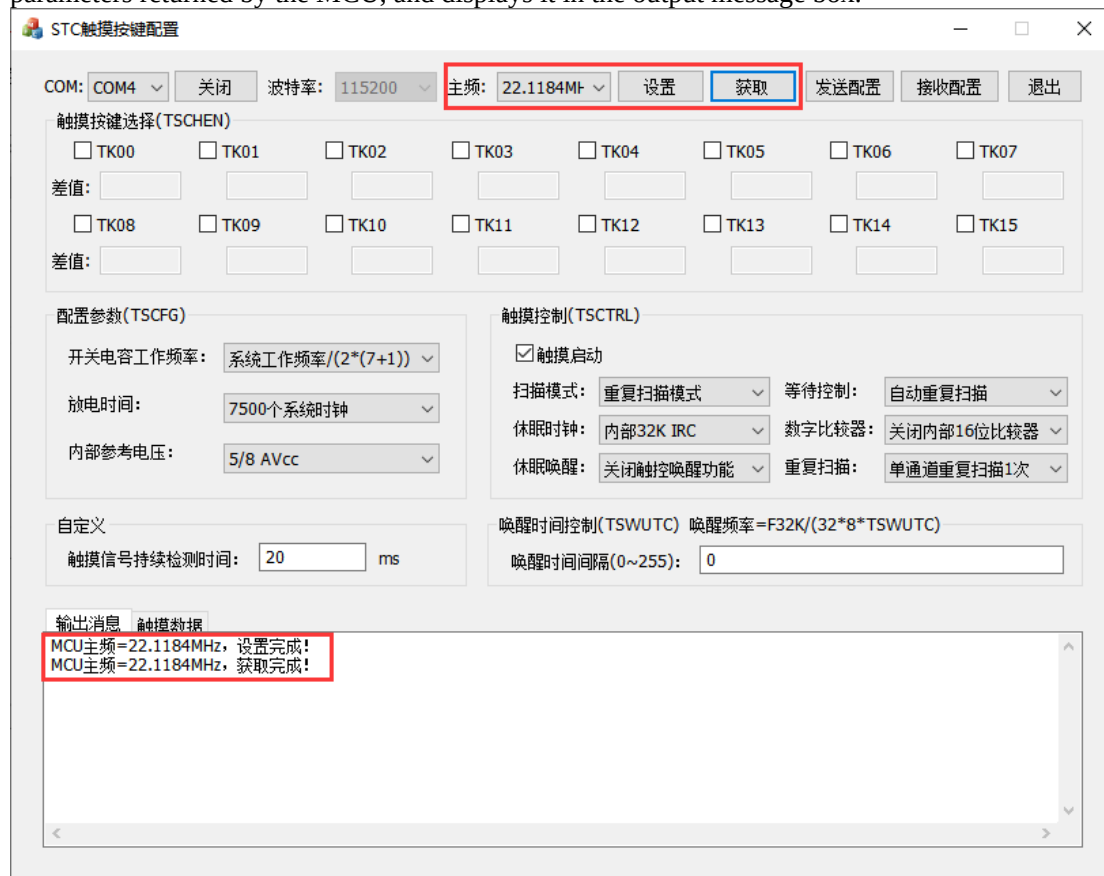
输出消息 触摸数据

2. MCU main frequency setting

"Frequency": drop-down box to select the MCU main frequency to be set.

"Set": The button sends the main frequency setting command to the MCU, and at the same time outputs the message box to display the message of setting the main frequency.

"Get": The button sends the main frequency acquisition command to the MCU, receives the main frequency parameters returned by the MCU, and displays it in the output message box.



3. Touch Configuration Settings

"Touch key selection": Check the channel that needs to be set as the touch key function, "Difference" is the key value difference before and after the touch key is pressed, and the MCU uses half of the difference as the threshold to judge whether the touch key is pressed.

"Configuration Parameters", "Touch Control": Set the corresponding register parameters through the drop-down box and check box.

"Custom": You can set the continuous detection time of the touch signal and the wake-up time interval.

"Send Configuration": Click this button to send the touch key configuration parameters to the MCU, and at the same time output the message box to display the touch key register parameter configuration message.

"Receive configuration": Click this button to send the obtained touch key configuration to the MCU, receive the configuration parameters returned by the MCU, and display the obtained touch key register parameter configuration message in the output message box.

STC 触摸按键配置

COM: COM4 关闭 波特率: 115200 主频: 22.1184MHz 设置 获取 发送配置 接收配置 退出

2 发送配置参数指令

触摸按键选择(TSCHEN)

☒ TK00 ☒ TK01 ☐ TK02 ☐ TK03 ☐ TK04 ☐ TK05 ☐ TK06 ☐ TK07

差值: 940 890

☐ TK08 ☐ TK09 ☐ TK10 ☐ TK11 ☐ TK12 ☐ TK13 ☐ TK14 ☒ TK15

差值: 1854

配置参数(TSCFG)

开关电容工作频率: 系统工作频率/(2*(7+1))

放电时间: 7500个系统时钟

内部参考电压: 5/8 AV_{CC}

自定义

触摸信号持续检测时间: 20 ms

触摸控制(TSCCTRL)

1 设置触摸按键参数

☒ 触摸启动

扫描模式: 重复扫描模式 等待控制: 自动重复扫描

休眠时钟: 内部32K IRC 数字比较器: 关闭内部16位比较器

休眠唤醒: 关闭触键唤醒功能 重复扫描: 单通道重复扫描1次

唤醒时间控制(TSWUTC) 唤醒频率=F32K/(32*8*TSWUTC)

唤醒时间间隔(0~255): 0

输出消息 触摸数据

TSCHEN1 = 0x03; TSCHEN2 = 0x80; TSCFG1 = 0x7F; TSCFG2 = 0x02; TSCCTRL = 0x80; TSRT = 0x00; TSWUTC = 0; 检测时间 = 20

TK00差值 = 940; TK01差值 = 890; TK02差值 = 0; TK03差值 = 0; TK04差值 = 0; TK05差值 = 0; TK06差值 = 0; TK07差值 = 0;

TK08差值 = 0; TK09差值 = 0; TK10差值 = 0; TK11差值 = 0; TK12差值 = 0; TK13差值 = 0; TK14差值 = 0; TK15差值 = 1854;

设置完成!

TSCHEN1 = 0x03; TSCHEN2 = 0x80; TSCFG1 = 0x7F; TSCFG2 = 0x02; TSCCTRL = 0x80; TSRT = 0x00; TSWUTC = 0; 检测时间 = 20

TK00差值 = 940; TK01差值 = 890; TK02差值 = 800; TK03差值 = 800; TK04差值 = 800; TK05差值 = 800; TK06差值 = 800; TK07差值 = 800;

TK08差值 = 800; TK09差值 = 800; TK10差值 = 800; TK11差值 = 800; TK12差值 = 800; TK13差值 = 800; TK14差值 = 800; TK15差值 = 1854;

获取完成!

3 输出配置参数消息

23 LED Driver

Product line	LED driver
STC8H1K08 family	
STC8H1K28 family	
STC8H3K64S4 family	
STC8H3K64S2 family	
STC8H8K64U family	
STC8H2K64T family	●
STC8H4K64TLR family	●
STC8H4K64TLCD family	
STC8H4K64LCD family	

An LED driver is integrated in STC8H series of microcontrollers.

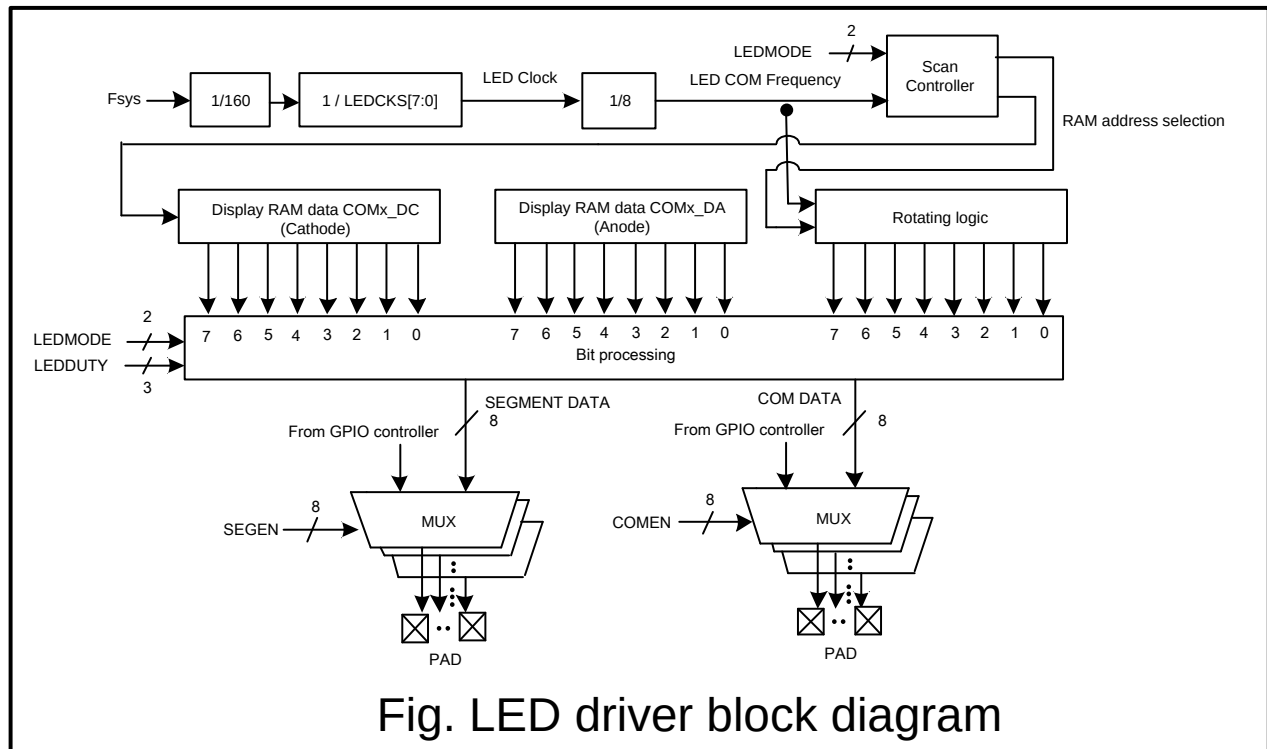
The LED driver circuit includes a timing controller, 8 COM output pins and 8 SEGMENT output pins. Each pin has a corresponding register enable bit, which can independently control whether the pin is enabled or not. Pins that are not enabled can be used as pins for GPIO or other functions.

The LED driver supports three modes: common cathode, common anode, common cathode / common anode. At the same time, it can select 1/8 ~ 8/8 duty/cycle to adjust the gray scale. So only software is needed to adjust the LED and digital LED brightness.

After power-on reset, the enable bit LEDON is 0 and the LED driver is turned off. Set LEDON to 1 to enable the LED driver. If LEDMODE= 00, the driver works in the common cathode mode. At this time, the selected COM outputs a low level, the selected SEGMENTS to light LED output high level. Therefore, the forward bias of the LED between the two points of SEGMENT and COM turns on and lights up. Similarly, if LEDMODE = 01, the driver works in the common anode mode. At this time, the selected COM outputs a high level, and the selected SEGMENTS to light the LED output low level. The forward bias of the LED lights up. If LEDMODE = 10, the driver works in the common cathode / common anode time-sharing drive mode. The COM level is low and high-level time-sharing. The principle of LED lighting is the same as common cathode and common anode.

In the common cathode mode and the common anode mode, the display RAM address is independent. The location of display RAM address in the common cathode / common anode time-sharing mode is also read independently.

23.1 Internal Structure Diagram of LED Driver



23.2 Registers Related to LED Driver

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
COMEN	COM Enable Register	FB00H	C7EN	C6EN	C5EN	C4EN	C3EN	C2EN	C1EN	C0EN	0000,0000
SEGENL	SEG Enable Register	FB01H	S7EN	S6EN	S5EN	S4EN	S3EN	S2EN	S1EN	S0EN	0000,0000
SEGENH	SEG Enable Register	FB02H	S15EN	S14EN	S13EN	S12EN	S11EN	S10EN	S9EN	S8EN	0000,0000
LEDCTRL	LED Control Register	FB03H	LEDON	-	LEDMODE[1:0]		-	LEDDUTY[2:0]			0000,0000
LEDCKS	LED Clock Divide Register	FB04H									0000,0001
COM0_DA_L	Common Anode Mode Disply	FB10H									0000,0000
COM1_DA_L	Common Anode Mode Disply	FB11H									0000,0000
COM2_DA_L	Common Anode Mode Disply	FB12H									0000,0000
COM3_DA_L	Common Anode Mode Disply	FB13H									0000,0000
COM4_DA_L	Common Anode Mode Disply	FB14H									0000,0000
COM5_DA_L	Common Anode Mode Disply	FB15H									0000,0000
COM6_DA_L	Common Anode Mode Disply	FB16H									0000,0000
COM7_DA_H	Common Anode Mode Disply	FB17H									0000,0000
COM0_DA_H	Common Anode Mode Disply	FB18H									0000,0000
COM1_DA_H	Common Anode Mode Disply	FB19H									0000,0000
COM2_DA_H	Common Anode Mode Disply	FB1AH									0000,0000
COM3_DA_H	Common Anode Mode Disply	FB1BH									0000,0000
COM4_DA_H	Common Anode Mode Disply	FB1CH									0000,0000
COM5_DA_H	Common Anode Mode Disply	FB1DH									0000,0000
COM6_DA_H	Common Anode Mode Disply	FB1EH									0000,0000
COM7_DA_H	Common Anode Mode Disply	FB1FH									0000,0000
COM0_DC_L	Common Cathode Mode Disply	FB20H									0000,0000
COM1_DC_L	Common Cathode Mode Disply	FB21H									0000,0000
COM2_DC_L	Common Cathode Mode Disply	FB22H									0000,0000
COM3_DC_L	Common Cathode Mode Disply	FB23H									0000,0000
COM4_DC_L	Common Cathode Mode Disply	FB24H									0000,0000
COM5_DC_L	Common Cathode Mode Disply	FB25H									0000,0000

COM6_DC_L	Common Cathode Mode Disply	FB26H		0000,0000
COM7_DC_L	Common Cathode Mode Disply	FB27H		0000,0000
COM0_DC_H	Common Cathode Mode Disply	FB28H		0000,0000
COM1_DC_H	Common Cathode Mode Disply	FB29H		0000,0000
COM2_DC_H	Common Cathode Mode Disply	FB2AH		0000,0000
COM3_DC_H	Common Cathode Mode Disply	FB2BH		0000,0000
COM4_DC_H	Common Cathode Mode Disply	FB2CH		0000,0000
COM5_DC_H	Common Cathode Mode Disply	FB2DH		0000,0000
COM6_DC_H	Common Cathode Mode Disply	FB2EH		0000,0000
COM7_DC_H	Common Cathode Mode Disply	FB2FH		0000,0000

23.2.1 COM Enable Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
COMEN	FB00H	C7EN	C6EN	C5EN	C4EN	C3EN	C2EN	C1EN	C0EN

CnEN: COMn enable control bit (n=0~7)

0: disable COMn, keep GPIO function

1: enable COMn, the corresponding I/O outputs the driving waveform of COM when LEDON = 1.

23.2.2 SEG Enable Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SEGENL	FB01H	S7EN	S6EN	S5EN	S4EN	S3EN	S2EN	S1EN	S0EN
SEGENH	FB02H	S15EN	S14EN	S13EN	S12EN	S11EN	S10EN	S9EN	S8EN

SnEN: SEGn enable control bit (n=0~7)

0: disable SEGn, keep GPIO function

1: enable SEGn, the corresponding I/O outputs the driving waveform of SEG when LEDON = 1.

23.2.3 LED Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LEDCTRL	FB03H	LEDON	-	LEDMODE[1:0]		-	LEDDUTY[2:0]		

LEDON: LED driver enable control bit

0: disable LED driver

1: enable LED driver.

LEDMODE[1:0]: LED drive mode

LEDMODE[1:0]	Drive mode
00	Common cathod mode
01	Common anode mode
10	Common cathod / common anode mode
11	Reserved

LEDDUTY[2:0]: LED grayscale adjustment

LEDDUTY[2:0]	LED Duty/cycle	LED brightness
000	8/8	100%
001	7/8	87.5%
010	6/8	75%
011	5/8	62.5%
100	4/8	50%

101	3/8	37.5%
110	2/8	25%
111	1/8	12.5%

23.2.4 LED Clock Divide Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LEDCKS	FB04H								

LEDCKS: LED clock division control

$$\text{LED working frequency} = \frac{\text{SYSclk}}{160 * \text{LEDCKS}[7:0]}$$

If the value of the register LEDCKS is set too large, it will cause the LED to flicker. Generally, if the LED refresh frequency is greater than or equal to 75Hz, there will be no obvious flicker.

$$\frac{\text{SYSclk}}{160 * 8 * \text{Ncom} * \text{LEDCKS}[7:0]} \geq 75\text{HZ}$$

NCOM: The number of COMs enabled, if it is a common cathode/common anode mode, it is twice the number of COMs

For example: if the operating frequency of the single-chip microcomputer is 11.0592MHz, the operating mode of the LED is common cathode/common anode mode, COMEN is set to 0FFH, even if 8 COMs are available, $11059200/160/8/16 / \text{LEDCKS} \geq 75$, $\text{LEDCKS} \leq 7.2$, so LEDCKS is recommended to be set to 7.

23.2.5 LED data registers of common anode mode

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
COM0_DA_L	FB10H								
COM1_DA_L	FB11H								
COM2_DA_L	FB12H								
COM3_DA_L	FB13H								
COM4_DA_L	FB14H								
COM5_DA_L	FB15H								
COM6_DA_L	FB16H								
COM7_DA_L	FB17H								
COM0_DA_H	FB18H								
COM1_DA_H	FB19H								
COM2_DA_H	FB1AH								
COM3_DA_H	FB1BH								
COM4_DA_H	FB1CH								
COM5_DA_H	FB1DH								
COM6_DA_H	FB1EH								
COM7_DA_H	FB1FH								

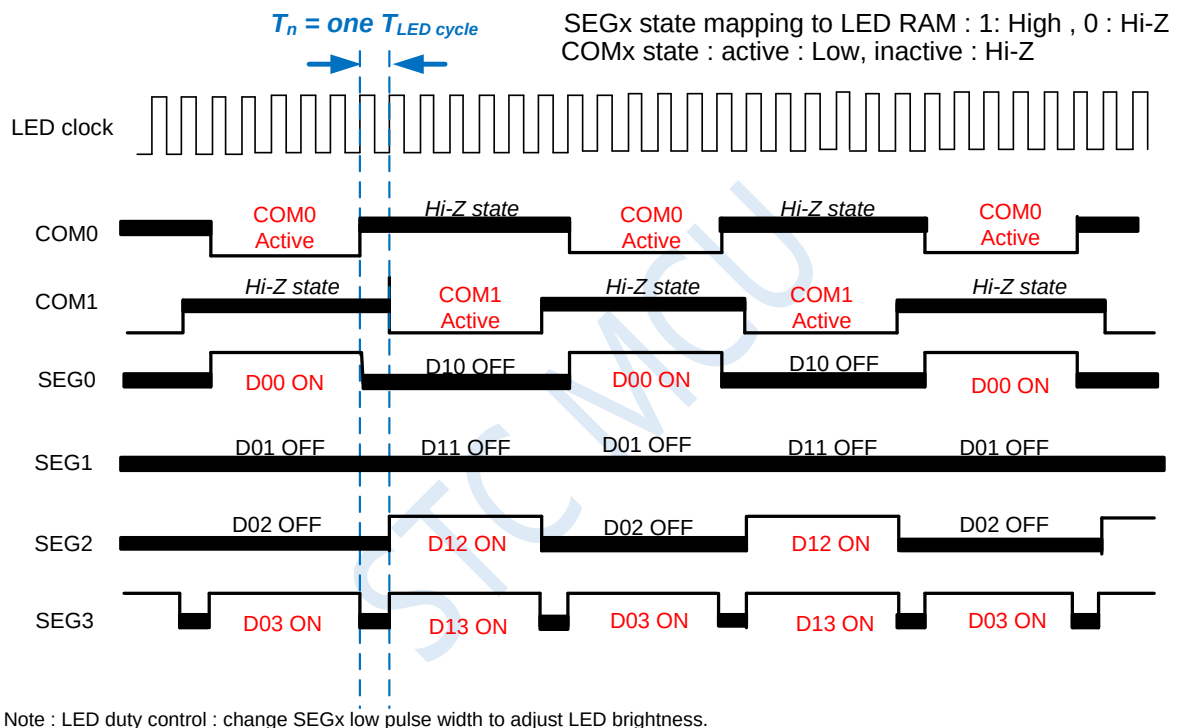
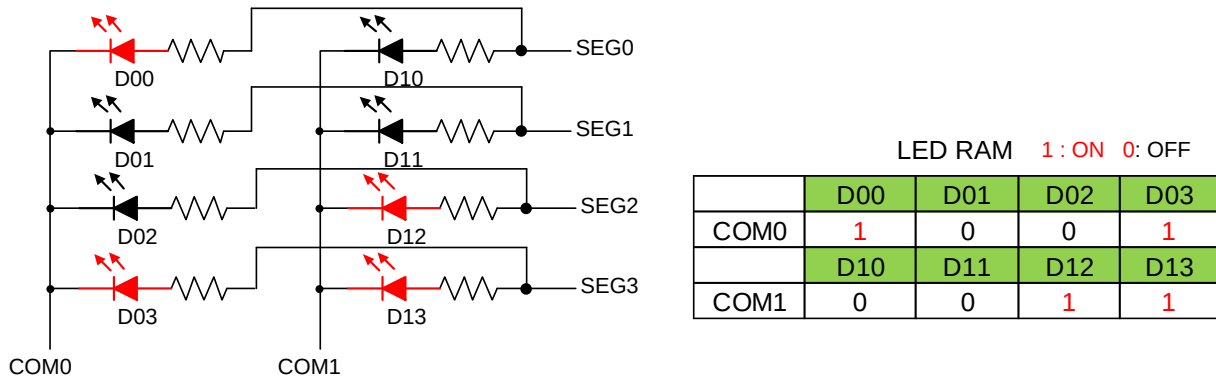
23.2.6 LED data registers of common cathod mode

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0

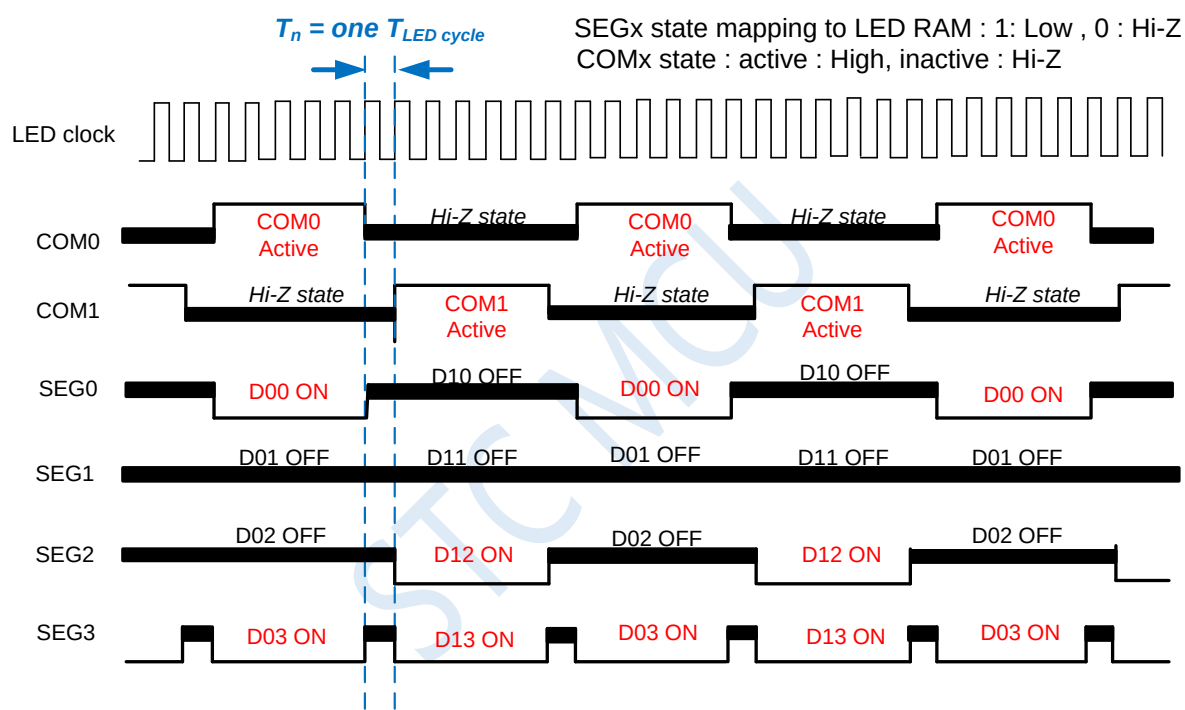
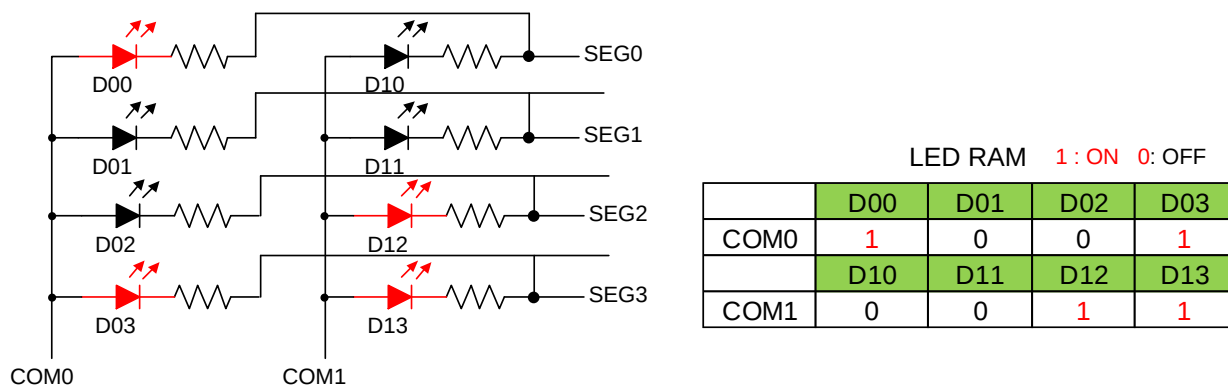
COM0_DC_L	FB20H	
COM1_DC_L	FB21H	
COM2_DC_L	FB22H	
COM3_DC_L	FB23H	
COM4_DC_L	FB24H	
COM5_DC_L	FB25H	
COM6_DC_L	FB26H	
COM7_DC_L	FB27H	
COM0_DC_H	FB28H	
COM1_DC_H	FB29H	
COM2_DC_H	FB2AH	
COM3_DC_H	FB2BH	
COM4_DC_H	FB2CH	
COM5_DC_H	FB2DH	
COM6_DC_H	FB2EH	
COM7_DC_H	FB2FH	

STC MCU

23.3 LED Common Cathod Mode (LEDMODE = 00)



23.4 LED Common Anode Mode (LEDMODE = 01)

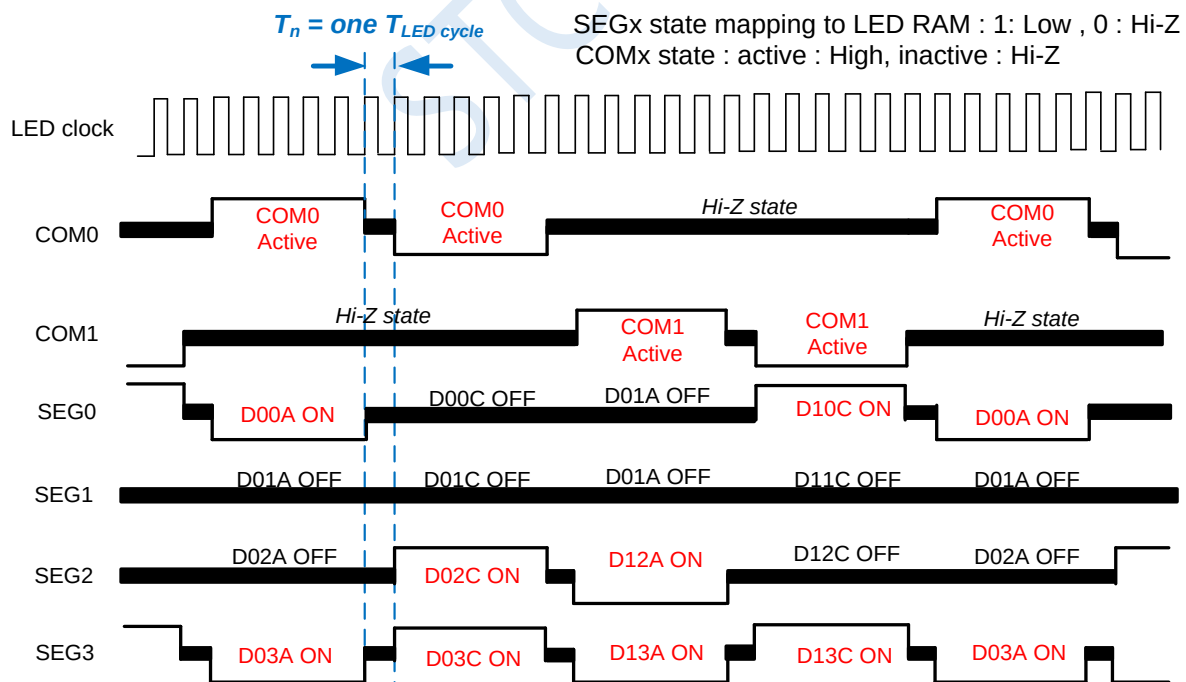
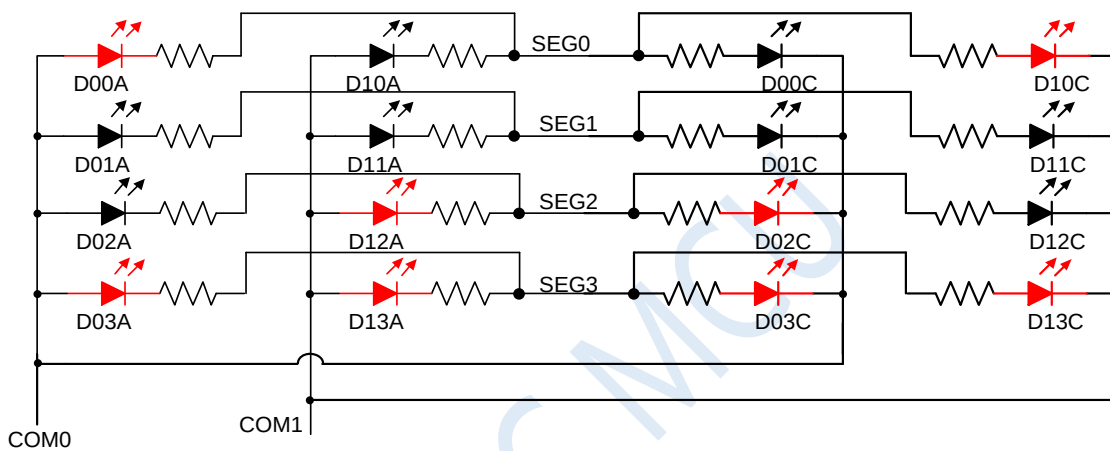


Note : LED duty control : change SEGx low pulse width to adjust LED brightness.

23.5 LED Common Cathode/ Common Anode Mode (LEDMODE = 10)

LED RAM 1: ON 0: OFF

	D00A	D01A	D02A	D03A
COM0	1	0	0	1
	D00C	D01C	D02C	D03C
COM0	0	0	1	1
	D10A	D11A	D12A	D13A
COM1	0	0	1	1
	D10C	D11C	D12C	D13C
COM1	1	0	0	1



Note : LED duty control : change SEGx low pulse width to adjust LED brightness.

23.6 Touch Keys and LED Driver Share I/O

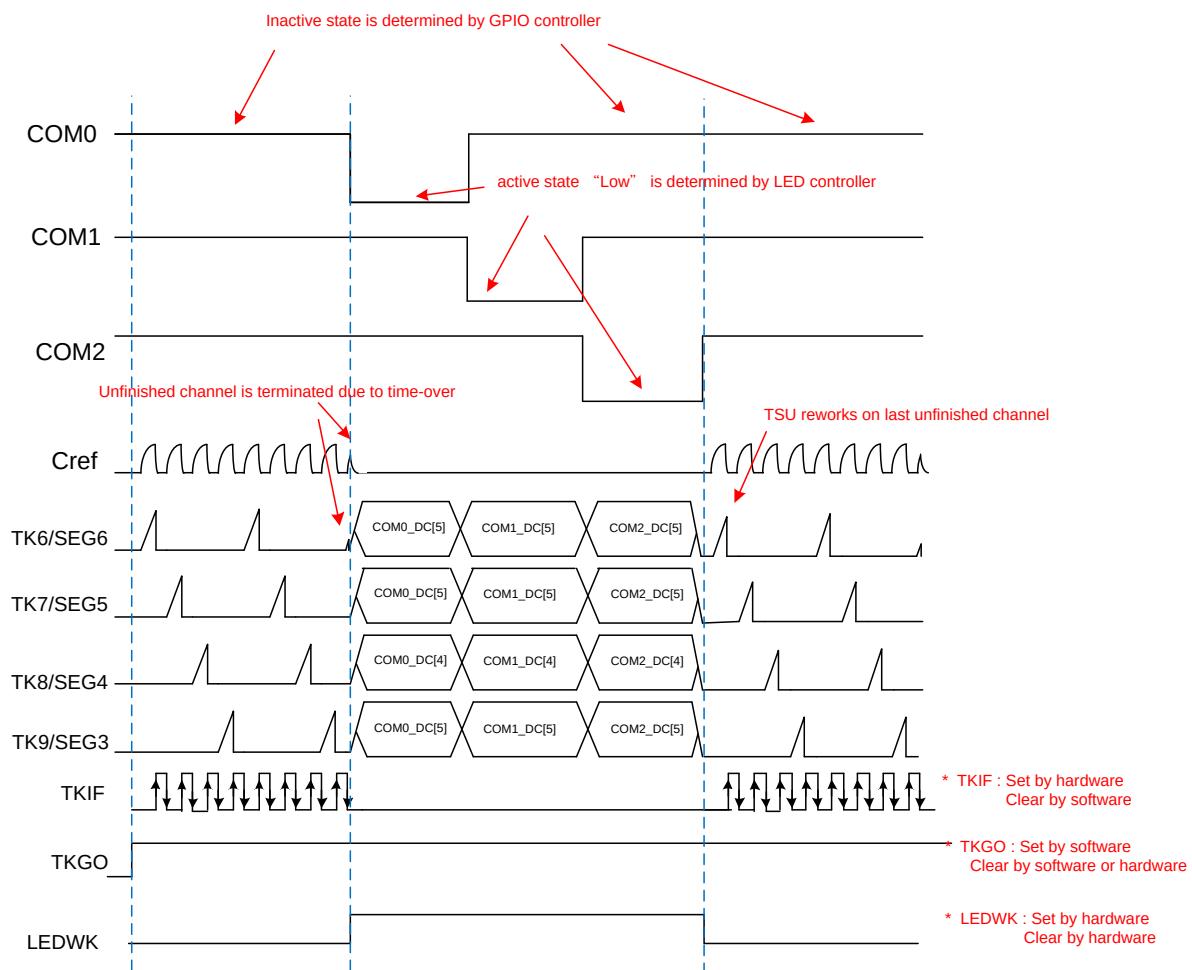


Fig. Touch/LED share timing diagram

(Common Cathode, only COM 0~2 and TK6~TK9 are enabled)

Steps:

1. Select the touch key channel to be scanned. The registers are TSCHEN1 and TSCHEN2.
2. Configure switching frequency SCR [2: 0], discharge time DT [2: 0] and select internal comparator reference voltage TSVR [1: 0].
3. Configure TSSAMP [1: 0] to determine the number of repeated scans of the same channel. If the CPU task is heavy, configure TSWAIT to use the TSIF state to delay the next channel scan.
4. If necessary, configure TSDCEN to enable the internal digital comparison function.
5. Set the TSRT content. If the TSRT content is not 0x00, the LED driver time sharing multitasking function is not enabled.
6. Configure the SEGEN and COMEN registers.
7. Configure LEDCKS to determine the time length of each COM action. This needs to be considered with the TKRT register to calculate the LED refresh rate.
8. Configure LEDMODE [1: 0] and LEDDUTY according to the operating mode and brightness of the LED required.
9. Write data to the data registers COMx_DC and COM_DA of the LED.
10. Set TSGO=1, the touch key controller starts scanning.
 - a) You can read TSWKCHN [3: 0] using software to know which channel is currently being scanned. After a channel is scanned, the hardware will set TSIF to 1 and the completed channel number will

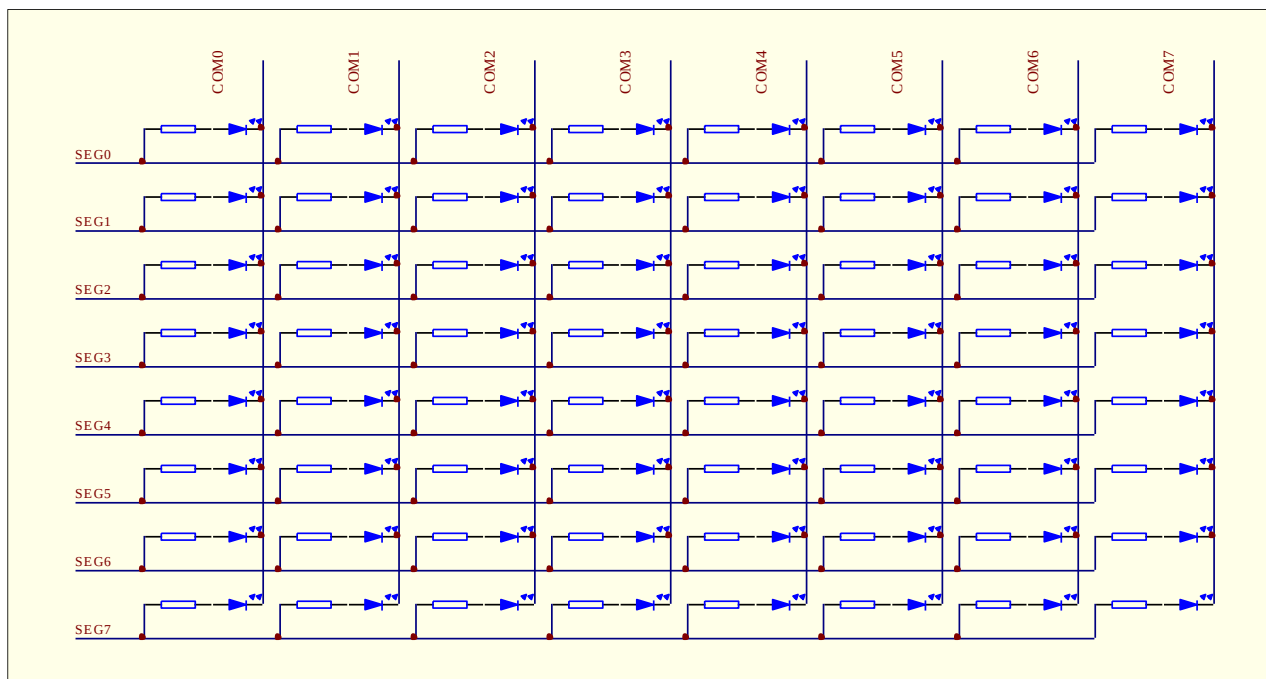
be written into TSDNCHN [3: 0]. If an overflow occurs, TSOV will also be set to 1. Software should read these registers to decide what to do next. TSIF and TSOV can only be set by hardware and cleared by software.

- b) When switching to the LED working time, the LEDWK bit is read as 1, which is used to determine whether the touch key controller is working or the LED driver is working.
 - c) Software continuously updates LED data register according to actual needs.
11. If you want to terminate the touch key and LED time-sharing multi-tasking mode, you need to write TSGO = 0, then the multi-task mode is terminated, the touch key and LED are not working. And the control right of the I/O port returns to the GPIO controller.

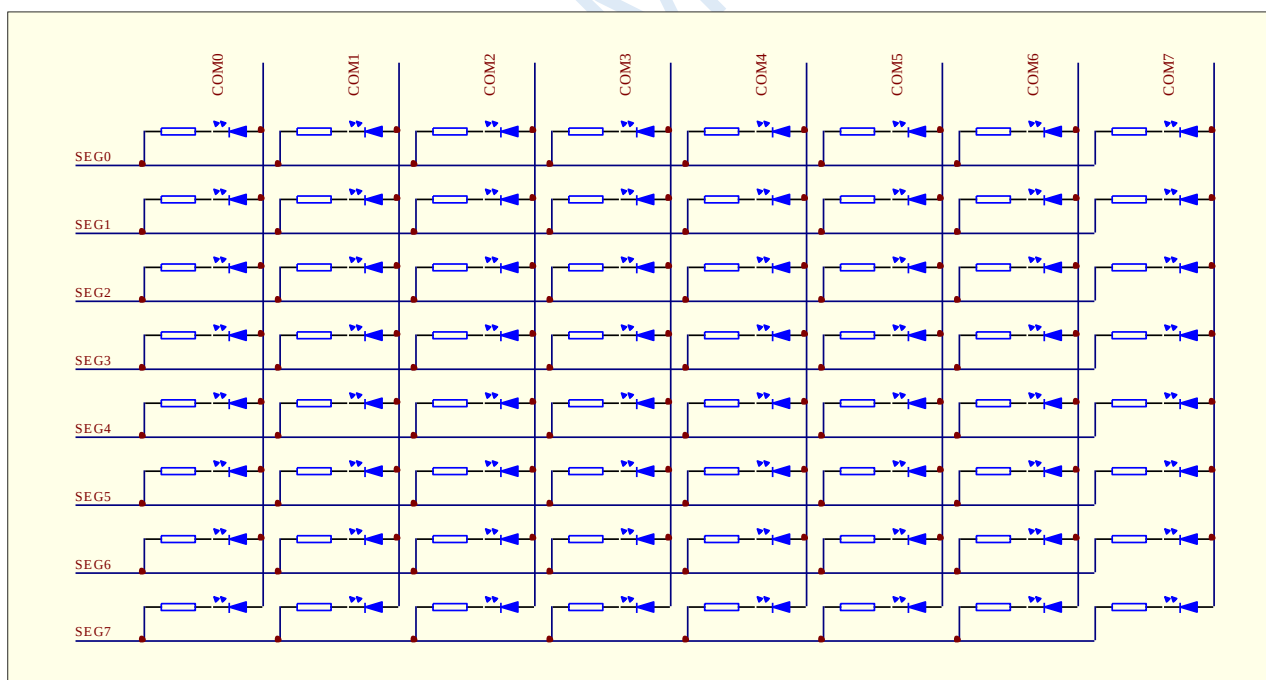
Note:

1. The SINGLE control bit is invalid in touch key and LED time-sharing multi-tasking mode. Only software writing TSGO can control the module on and off.
2. When the touch key scan time is terminated and turned to LED action time, the last key is almost incomplete. At this time, the hardware processing will not produce TSIF, and not update the registers related to the touch data, but the hardware will remember this channel number. The incomplete channel will be re-scanned and start a new round of scanning after the LED action period ends and turn to the touch key scanning time.
3. The circuit diagram of LED and touch multiplexing is as above. It should be noted that the LED's light-emitting color is different, and the equivalent capacitance of the LED will be different. The larger the capacitance, the more unfavorable the touch button, and the zero sensitivity will decrease. Generally speaking, the capacitance value of the red LED may be 35pF, but the yellow light will be as high as 100pF. At this time, if you want to increase the sensitivity of the touch button, you can connect a 1N4148 diode in series. The capacitance of 1N4148 is only 4pF, and 1N4148 string a 100pF yellow LED, the parasitic capacitance of the key will be slightly smaller than 4pF.

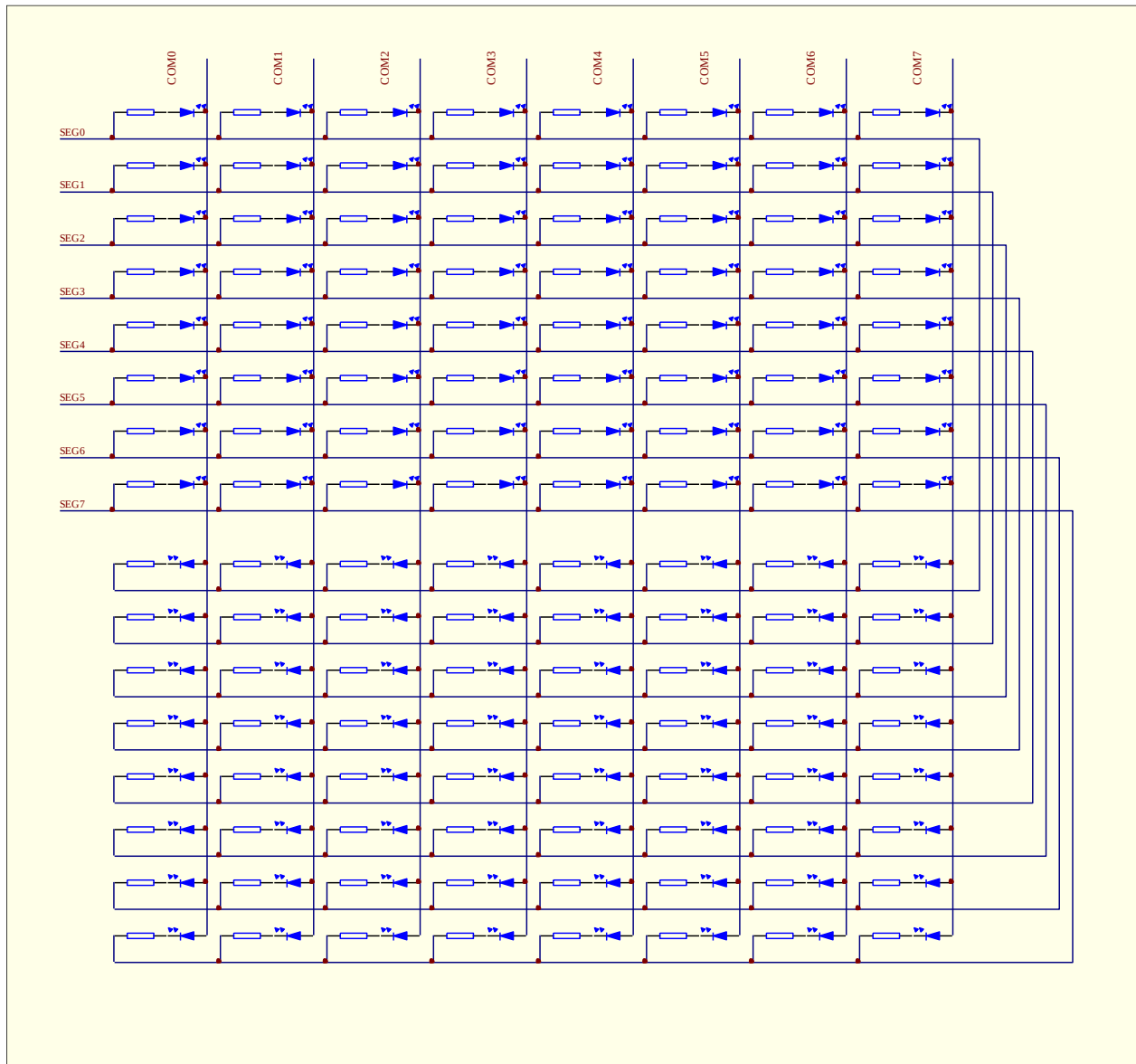
23.7 Reference Circuit of Common Cathode Mode



23.8 Reference Circuit of Common Anode Mode

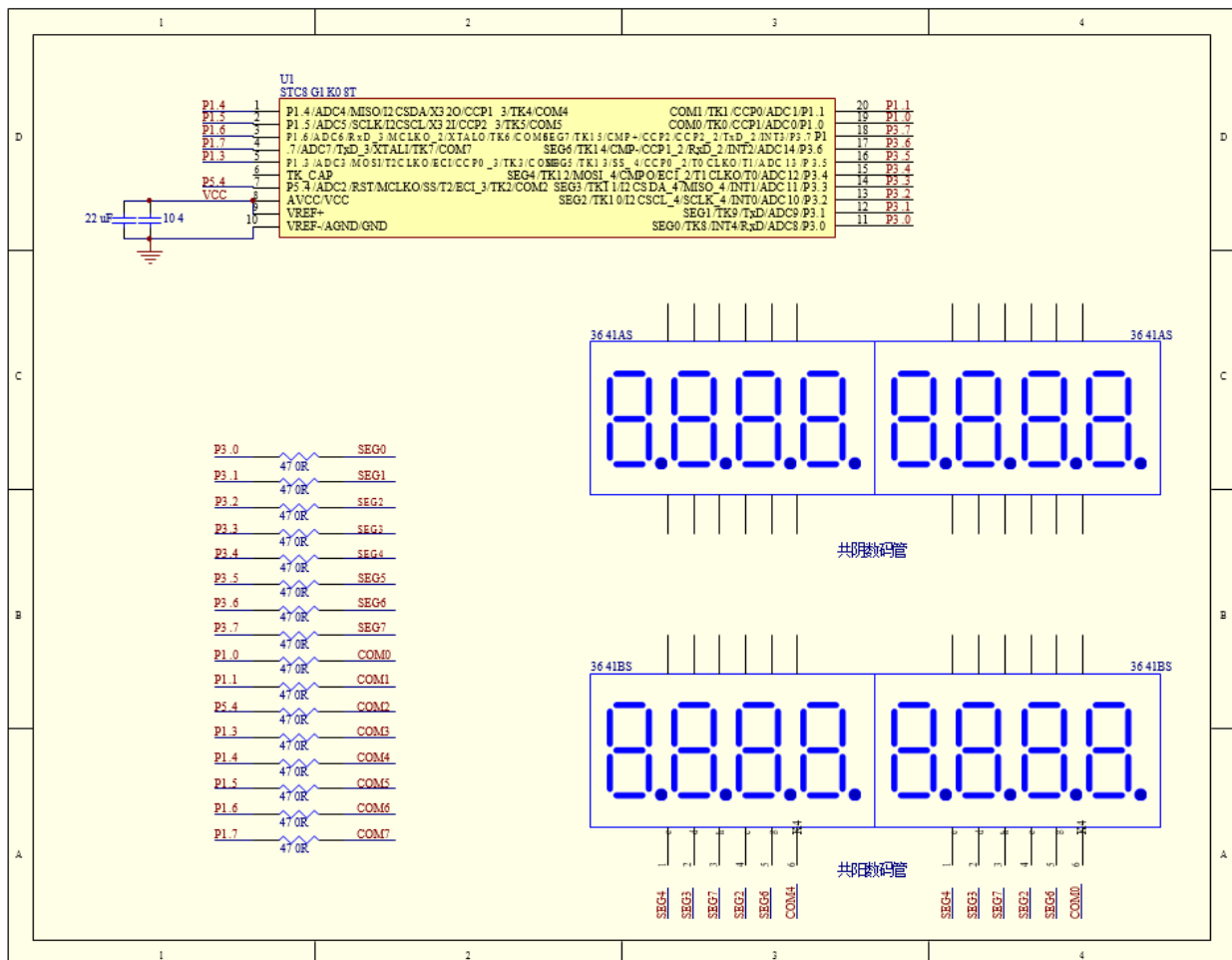


23.9 Reference Circuit of Common Cathode/Common Anode Mode



23.10 Example Routines

23.10.1 Common cathode/common anode mode drives 16 7-segment digital tubes



C language code

//Operating frequency for test is 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;

```

```

#define COMEN      (*(unsigned char volatile xdata *)0xfb00)
#define SEGENL     (*(unsigned char volatile xdata *)0xfb01)
#define LEDCTR     (*(unsigned char volatile xdata *)0xfb03)
#define LEDCKS     (*(unsigned char volatile xdata *)0xfb04)
#define COM0_DA    (*(unsigned char volatile xdata *)0xfb10)
#define COM1_DA    (*(unsigned char volatile xdata *)0xfb11)
#define COM2_DA    (*(unsigned char volatile xdata *)0xfb12)
#define COM3_DA    (*(unsigned char volatile xdata *)0xfb13)
#define COM4_DA    (*(unsigned char volatile xdata *)0xfb14)
#define COM5_DA    (*(unsigned char volatile xdata *)0xfb15)
#define COM6_DA    (*(unsigned char volatile xdata *)0xfb16)
#define COM7_DA    (*(unsigned char volatile xdata *)0xfb17)
#define COM0_DC    (*(unsigned char volatile xdata *)0xfb20)
#define COM1_DC    (*(unsigned char volatile xdata *)0xfb21)
#define COM2_DC    (*(unsigned char volatile xdata *)0xfb22)
#define COM3_DC    (*(unsigned char volatile xdata *)0xfb23)
#define COM4_DC    (*(unsigned char volatile xdata *)0xfb24)
#define COM5_DC    (*(unsigned char volatile xdata *)0xfb25)
#define COM6_DC    (*(unsigned char volatile xdata *)0xfb26)
#define COM7_DC    (*(unsigned char volatile xdata *)0xfb27)

```

```
char code PATTERN[16] =
```

```

{
    0x3f,      //0
    0x06,      //1
    0x5b,      //2
    0x4f,      //3
    0x66,      //4
    0x6d,      //5
    0x7d,      //6
    0x27,      //7
    0x7f,      //8
    0x6f,      //9
    0x77,      //A
    0x7c,      //b
    0x39,      //C
    0x5E,      //d
    0x79,      //E
    0x71,      //F

```

```
};
```

```
void main()
```

```

{
    P1M0 = 0xff;
    P1M1 = 0x00;
    P3M0 = 0xff;
    P3M1 = 0x00;
    P5M0 = 0x10;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    COMEN = 0xff;           //enable COM0~COM7
    SEGENL = 0xff;         //enable SEG0~SEG7

```

```
LEDCTRL = 0x20;                                     // Common cathode/common anode mode for LED driver
LEDCKS = 7;                                           // Set the LED refresh rate

COM0_DA = PATTERN[0];                                // Set the LED display content
COM1_DA = PATTERN[1];
COM2_DA = PATTERN[2];
COM3_DA = PATTERN[3];
COM4_DA = PATTERN[4];
COM5_DA = PATTERN[5];
COM6_DA = PATTERN[6];
COM7_DA = PATTERN[7];

COM0_DC = PATTERN[8];
COM1_DC = PATTERN[9];
COM2_DC = PATTERN[10];
COM3_DC = PATTERN[11];
COM4_DC = PATTERN[12];
COM5_DC = PATTERN[13];
COM6_DC = PATTERN[14];
COM7_DC = PATTERN[15];

LEDCTRL |= 0x80;                                     // Start LED driver

P_SW2 &= ~0x80;

while (1);
}
```

24 RTC real time clock

Product line	RTC
STC8H1K08 family	
STC8H1K28 family	
STC8H3K64S4 family	
STC8H3K64S2 family	
STC8H8K64U family A version	
STC8H8K64U family B version	●
STC8H2K64T family	●
STC8H4K64TLR family	●
STC8H4K64TLCD family	●
STC8H4K64LCD family	●

The STC8H2K64T series single-chip microcomputer integrates a real-time clock control circuit, which mainly has the following characteristics:

- Low power consumption: RTC module working current is as low as 10uA
- Long time span: support from 2000 to 2099, and automatically judge leap years
- Alarm clock: support a group of alarm clock settings
- Support multiple interrupts: alarm interrupt, day interrupt, hour interrupt, minute interrupt, second interrupt, 1/2 second interrupt, 1/8 second interrupt, 1/32 second interrupt
- Support power-down wake-up

24.1 RTC related registers

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
RTCCR	RTC Control register	FE60H	-	-	-	-	-	-	-	RUNRTC	0000,0000
RTCCFG	RTC Configuration register	FE61H	-	-	-	-	-	-	RTCKKS	SETRTC	0000,0000
RTCEN	RTC Interrupt enable register	FE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I	0000,0000
RTCIF	RTC Interrupt request register	FE63H	ALAIF	DAYIF	HOURLF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF	0000,0000
ALAHOUR	RTC alarm hour value	FE64H	-	-	-						0000,0000
ALAMIN	RTC alarm minute value	FE65H	-	-						0000,0000	
ALASEC	RTC alarm second value	FE66H	-	-						0000,0000	
ALASSEC	RTC alarm 1/128 second value	FE67H	-								0000,0000
INIYEAR	RTC year initialization	FE68H	-								0000,0000
INIMONTH	RTC month initialization	FE69H	-	-	-	-					0000,0000
INIDAY	RTC day initialization	FE6AH	-	-	-						0000,0000
INIHOUR	RTC hour initialization	FE6BH	-	-	-						0000,0000
INIMIN	RTC minute initialization	FE6CH	-	-						0000,0000	
INISEC	RTC second initialization	FE6DH	-	-						0000,0000	
INISSEC	RTC 1/128 second	FE6EH	-								0000,0000

	initialization								
YEAR	RTC year count value	FE70H	-						0000,0000
MONTH	RTC month count value	FE71H	-	-	-	-			0000,0000
DAY	RTC day count value	FE72H	-	-	-				0000,0000
HOURL	RTC hour count value	FE73H	-	-	-				0000,0000
MIN	RTC minute count value	FE74H	-	-					0000,0000
SEC	RTC second count value	FE75H	-	-					0000,0000
SSEC	RTC 1/128 second count value	FE76H	-						0000,0000

24.1.1 RTC control register (RTCCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
RTCCR	FE60H	-	-	-	-	-	-	-	RUNRTC

RUNRTC: RTC module control bit

0: Turn off RTC, RTC stops counting

1: Enable RTC and start RTC counting

24.1.2 RTC Configuration Register (RTCCFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
RTCCFG	FE61H	-	-	-	-	-	-	RTCKS	SETRTC

RTCKS: RTC clock source selection

0: select external 32.768KHz clock source (The software needs to start the external 32K crystal oscillator firstly)

1: Select internal 32K clock source (The software needs to start the internal 32K crystal oscillator firstly)

SETRTC: Set the initial value of RTC

0: meaningless

1: Trigger RTC register initialization. When SETRTC is set to 1, the hardware will automatically copy the values in the registers INIYEAR, INIMONTH, INIDAY, INIHOUR, INIMIN, INISEC, INISSEC to the registers YEAR, MONTH, DAY, HOUR, MIN, SEC, SSEC. After the initial completion, the hardware will automatically clear the SETRTC bit to 0.

24.1.3 RTC Interrupt Enable Register (RTCIEN)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
RTCIEN	FE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I

EALAI: Alarm interrupt enable bit

0: Turn off the alarm interrupt

1: Enable alarm interrupt

EDAYI: One day (24 hours) interrupt enable bit

0: Turn off one-day interrupt

1: Enable one-day interrupt

EHOURI: One hour (60 minutes) interrupt enable bit

0: Turn off hourly interrupt

1: Enable hourly interrupt

EMINI: One minute (60 seconds) interrupt enable bit

0: Turn off hourly interrupt

1: Enable hourly interrupt
 ESECI: One second interrupt enable bit
 0: Turn off the second interrupt
 1: Enable second interrupt
 ESEC2I: 1/2 second interrupt enable bit
 0: Turn off the 1/2 second interrupt
 1: Enable 1/2 second interrupt
 ESEC8I: 1/8 second interrupt enable bit
 0: Turn off 1/8 second interrupt
 1: Enable 1/8 second interrupt
 ESEC32I: 1/32 second interrupt enable bit
 0: Disable 1/32 second interrupt
 1: Enable 1/32 second interrupt

24.1.4 RTC Interrupt Request Register (RTCIF)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
RTCIF	FE63H	ALAIF	DAYIF	HOURLIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF

ALAIF: Alarm interrupt request bit. Need software to clear, software write 1 is invalid.

DAYIF: One day (24 hours) interrupt request bit. Need software to clear, software write 1 is invalid.

HOURLIF: One hour (60 minutes) interrupt request bit. Need software to clear, software write 1 is invalid.

MINIF: One minute (60 seconds) interrupt request bit. Need software to clear, software write 1 is invalid.

SECIF: One-second interrupt request bit. Need software to clear, software write 1 is invalid.

SEC2IF: 1/2 second interrupt request bit. Need software to clear, software write 1 is invalid.

SEC8IF: 1/8 second interrupt request bit. Need software to clear, software write 1 is invalid.

SEC32IF: 1/32 second interrupt request bit. Need software to clear, software write 1 is invalid.

24.1.5 RTC Alarm Setting Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
ALAHOUR	FE64H	-	-	-					
ALAMIN	FE65H	-	-						
ALASEC	FE66H	-	-						
ALASSEC	FE67H	-							

ALAHOUR: Set the hour value of the daily alarm.

Note: The value set is not BCD code, but HEX code. For example, if you need to set the hour value from 20 to ALAHOUR, you need to use the following code to set

```
MOV DPTR,#ALAHOUR
```

```
MOV A,#14H
```

```
MOVX @@DPTR,A
```

ALAMIN: Set the minute value of the daily alarm. The digital code is the same as ALAHOUR.

ALASEC: Set the second value of the daily alarm. The digital code is the same as ALAHOUR.

ALASSEC: Set the 1/128 second value of the daily alarm. The digital code is the same as ALAHOUR.

24.1.6 RTC real-time clock initial value setting registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INIYEAR	FE68H	-							
INIMONTH	FE69H								

INIDAY	FE6AH				
INIHOUR	FE6BH	-	-	-	
INIMIN	FE6CH	-	-		
INISEC	FE6DH	-	-		
INISSEC	FE6EH	-			

INIYEAR: Set the year value of the current real-time time. The valid range is 00~99. Corresponding to 2000~2099

Note: The set value is not BCD code, but HEX code. For example, if you need to set 20 to INIYEAR, you need to use the following code to set

```
MOV DPTR,#INIYEAR
```

```
MOV A,#14H
```

```
MOVX @@DPTR,A
```

INIMONTH: Set the monthly value of the current real-time time. The valid range is 1-12. The number code is the same as INIYEAR.

INIDAY: Set the daily value of the current real-time time. The valid range is 1~31. The number code is the same as INIYEAR.

INIHOUR: Set the hour value of the current real-time time. The valid range is 00~23. The number code is the same as INIYEAR.

INIMIN: Set the minute value of the current real-time time. The valid range is 00~59. The number code is the same as INIYEAR.

INISEC: Set the second value of the current real-time time. The valid range is 00~59. The number code is the same as INIYEAR.

INISSEC: Set 1/128 second value of current real time. The valid range is 00~127. The number code is the same as INIYEAR.

After the user has set the above initial value register, the user also needs to write 1 to the SETRTC bit (RTCCFG.0) to trigger the hardware to load the initial value into the RTC real-time counter

Also note: the hardware will not check the validity of the initialized data. When setting the initial value, the user must ensure the validity of the data and cannot exceed its valid range.

24.1.7 RTC real-time clock count registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
YEAR	FE70H	-							
MONTH	FE71H								
DAY	FE72H								
HOUR	FE73H	-	-	-					
MIN	FE74H	-	-						
SEC	FE75H	-	-						
SSEC	FE76H	-							

YEAR: The year value of the current real time. Note: The value of the register is not BCD code, but HEX code

MONTH: The monthly value of the current real-time time. The number code is the same as YEAR.

DAY: The daily value of the current real time. The number code is the same as YEAR.

HOUR: The hour value of the current real time. The number code is the same as YEAR.

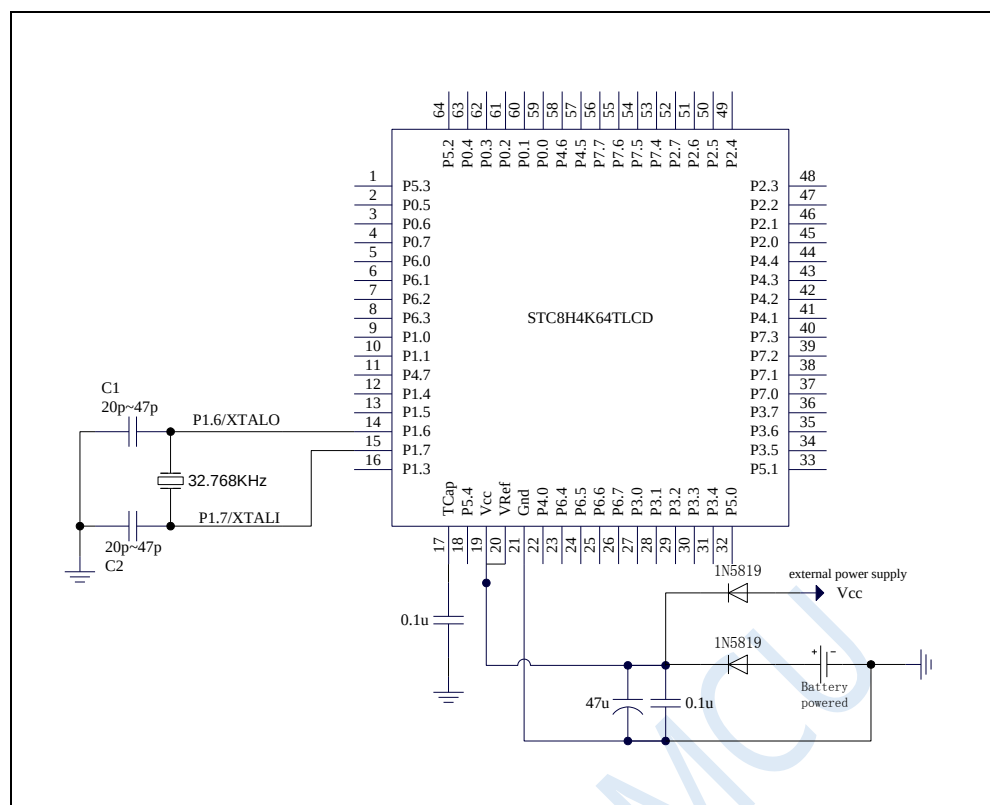
MIN: The minute value of the current real time. The number code is the same as YEAR.

SEC: The second value of the current real time. The number code is the same as YEAR.

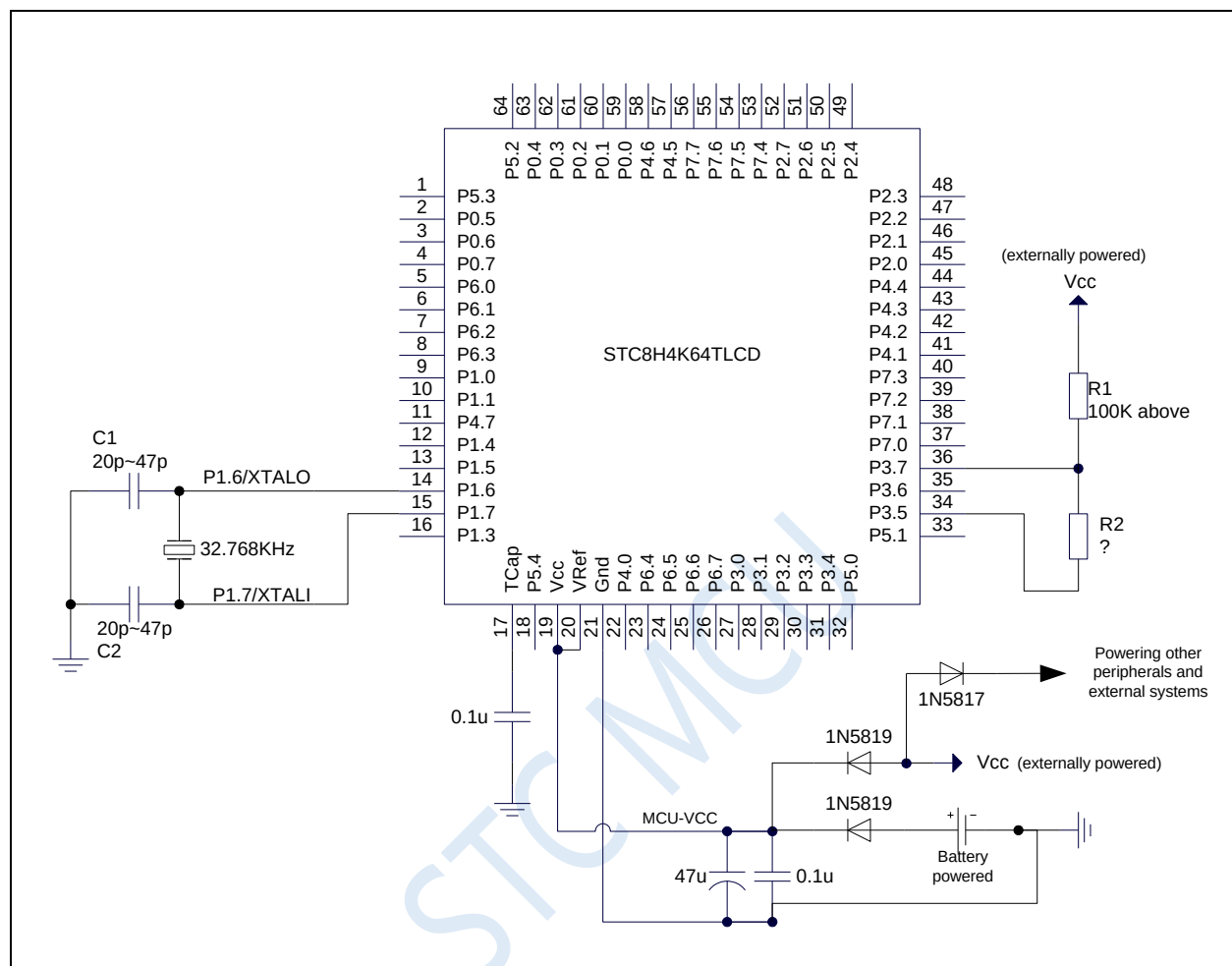
SSEC: 1/128 second value of current real time. The number code is the same as YEAR.

Note: YEAR, MONTH, DAY, HOUR, MIN, SEC, and SSEC are all read-only registers. If you need to write to these registers, you must use the registers INIYEAR, INIMONTH, INIDAT, INIHOU, INIMIN, INISEC, INISSEC and SETRTC are implemented.

24.2 RTC reference circuit diagram (without VBAT pin)



24.3 RTC practical circuit diagram



24.4 Example Routines

24.4.1 Serial port printing RTC clock example

C language code

//Operating frequency for test is 22.1184MHz. It is necessary to load the C language code file and the following assembly code file into the same project for use

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

sfr      TH2      = 0xD6;
sfr      TL2      = 0xD7;
sfr      AUXR     = 0x8E;
sfr      P_SW2    = 0xBA;

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;

#define RTCCR      (*(unsigned char volatile xdata *)0xfe60)
#define RTCCFG     (*(unsigned char volatile xdata *)0xfe61)
#define RTCIEN     (*(unsigned char volatile xdata *)0xfe62)
#define RTCIF      (*(unsigned char volatile xdata *)0xfe63)
#define ALAHOUR    (*(unsigned char volatile xdata *)0xfe64)
#define ALAMIN     (*(unsigned char volatile xdata *)0xfe65)
#define ALASEC     (*(unsigned char volatile xdata *)0xfe66)
#define ALASSEC    (*(unsigned char volatile xdata *)0xfe67)
#define INIYEAR    (*(unsigned char volatile xdata *)0xfe68)
#define INIMONTH   (*(unsigned char volatile xdata *)0xfe69)
#define INIDAY     (*(unsigned char volatile xdata *)0xfe6a)
#define INIhour    (*(unsigned char volatile xdata *)0xfe6b)
#define INIMIN     (*(unsigned char volatile xdata *)0xfe6c)
#define INISEC     (*(unsigned char volatile xdata *)0xfe6d)
#define INISSEC    (*(unsigned char volatile xdata *)0xfe6e)
#define YEAR      (*(unsigned char volatile xdata *)0xfe70)
#define MONTH     (*(unsigned char volatile xdata *)0xfe71)
#define DAY       (*(unsigned char volatile xdata *)0xfe72)
#define HOUR      (*(unsigned char volatile xdata *)0xfe73)
#define MIN       (*(unsigned char volatile xdata *)0xfe74)
#define SEC       (*(unsigned char volatile xdata *)0xfe75)
#define SSEC      (*(unsigned char volatile xdata *)0xfe76)

#define MAIN_Fosc  22118400L
#define Baudrate   115200L
```

```

#define    TM                (65536 -(MAIN_Fosc/Baudrate/4))

bit        B1S_Flag;

void RTC_config(void);

void UartInit(void)
{
    SCON = (SCON & 0x3f) | 0x40;
    TL2 = TM;
    TH2 = TM>>8;
    AUXR |= 0x15;
}

void UartPutc(unsigned char dat)
{
    SBUF = dat;
    while(TI==0);
    TI = 0;
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

void RTC_Isr() interrupt 13
{
    char store;

    store = P_SW2;
    P_SW2 |= 0x80;    // Enable XFR access

    if(RTCIF & 0x08)    // Determine whether it is the second interrupt
    {
        RTCIF &= ~0x08;    // clear interrupt flag
        B1S_Flag = 1;
    }

    P_SW2 = store;
}

void main(void)
{
    P0M1 = 0; P0M0 = 0; //set as quasi-bidirectional port
    P1M1 = 0; P1M0 = 0; //set as quasi-bidirectional port
    P2M1 = 0; P2M0 = 0; //set as quasi-bidirectional port
    P3M1 = 0; P3M0 = 0; //set as quasi-bidirectional port
    P4M1 = 0; P4M0 = 0; //set as quasi-bidirectional port
    P5M1 = 0; P5M0 = 0; //set as quasi-bidirectional port

    UartInit();
    RTC_config();
    EA = 1;
    printf("RTC Test Programme!\r\n");    // UART sends a string

    while (1)

```

```

{
    if(B1S_Flag)
    {
        B1S_Flag = 0;
        P_SW2 |= 0x80;                // Enable XFR access

        printf("Year=%02d ", YEAR);
        printf("Month=%02d ", MONTH);
        printf("Day=%02d ", DAY);
        printf("Hour=%02d ", HOUR);
        printf("Minute=%02d ", MIN);
        printf("Second=%02d ", SEC);
        printf("\r\n");

        P_SW2 &= ~0x80;                //Disable XFR access
    }
}

void RTC_config(void)
{
    P_SW2 |= 0x80;                    //Enable XFR access

    //Select Internal 32K
    IRC32KCR = 0x80;                  //Start the internal 32K oscillator
    while (!(IRC32KCR & 0x01));        //Wait for the clock to stabilize
    RTCCFG |= 0x02;                    //Select Internal 32K as RTC clock source

// //Select external 32K
// X32KCR = 0xc0;                     //Start the external 32K crystal oscillator
// while (!(X32KCR & 0x01));           //Wait for the clock to stabilize
// RTCCFG &= ~0x02;                   //Select external 32K as RTC clock source

    INIYEAR = 21;                      //Y:2021
    INIMONTH = 12;                     //M:12
    INIDAY = 31;                       //D:31
    INIHOUR = 23;                      //H:23
    INIMIN = 59;                       //M:59
    INISEC = 50;                       //S:50
    INISSEC = 0;                       //S/128:0
    RTCCFG |= 0x01;                    //Trigger RTC register initialization

    RTCIF = 0;                         //clear interrupt flag
    RTCIEN = 0x08;                     //Enable RTC second interrupt
    RTCCR = 0x01;                      // Enable RTC
    P_SW2 &= ~0x80;                    //Disable XFR access
}

```

Assembly code

; Save the following code as an ASM format file and load it into the project together, for example: isr.asm

```

CSEG AT 0123H
JMP 006BH
END

```

25 LCD driver (Traditional segment/stroke LCD driver)

Product line	LCD driver
STC8H1K08 family	
STC8H1K28 family	
STC8H3K64S4 family	
STC8H3K64S2 family	
STC8H8K64U family	
STC8H2K64T family	
STC8H4K64TLR family	
STC8H4K64TLCD family	●
STC8H4K64LCD family	●

Some microcontrollers of the STC8H series integrate an LCD driver, which can be used to drive the LCD. It can drive up to 4COM*40SEG dot matrix LCD.

Ports corresponding to the COM line:

COM7	COM6	COM5	COM4	COM3	COM2	COM1	COM0
-	-	-	-	P3.6	P3.5	P5.1	P5.0

Ports corresponding to SEG lines:

SEG39	SEG38	SEG37	SEG36	SEG35	SEG34	SEG33	SEG32
P6.3	P6.2	P6.1	P6.0	P7.0	P7.1	P7.2	P7.3
SEG31	SEG30	SEG29	SEG28	SEG27	SEG26	SEG25	SEG24
P3.7	P4.1	P4.2	P4.3	P4.4	P2.0	P2.1	P2.2
SEG23	SEG22	SEG21	SEG20	SEG19	SEG18	SEG17	SEG16
P2.3	P2.4	P2.5	P2.6	P2.7	P4.5	P4.6	P0.0
SEG15	SEG14	SEG13	SEG12	SEG11	SEG10	SEG9	SEG8
P0.1	P0.2	P0.3	P0.4	P5.2	P5.3	P0.5	P0.6
SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0
P0.7	P1.0	P1.1	P4.7	P1.2/P7.4	P1.3/P7.5	P1.4/P7.6	P1.5/P7.7

25.1 LCD function pins switch

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCD CFG2	FB81H	-	-	-	-	SEG3PS	SEG2PS	SEG1PS	SEG0PS

SEG3PS: SEG3 control pin selection bit

SEG3PS	SEG3
0	P1.2
1	P7.4

SEG2PS: SEG2 control pin selection bit

SEG2PS	SEG2
0	P1.3
1	P7.5

SEG1PS: SEG1 control pin selection bit

SEG1PS	SEG1
0	P1.4

1	P7.6
SEG0PS: SEG0 control pin selection bit	
SEG0PS	SEG0
0	P1.5
1	P7.7

25.2 LCD related registers

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
LCDCFG	LCD configuration register	FB80H	CKSEL	-	-	-	VRLPSEL	VLCDSEL[2:0]			0xxx,0000
LCDCFG2	LCD configuration register 2	FB81H	-	-	-	-	SEG3PS	SEG2PS	SEG1PS	SEG0PS	xxxx,0000
DBLEN	Dead time length configuration	FB82H	-	-	-	-	-	DBLEN[2:0]			xxxx,x000
COMLENL	COM time length configuration low bits	FB83H	COMLEN[7:0]								0000,0000
COMLENM	COM time length configuration medium bits	FB84H	COMLEN[15:8]								0000,0000
COMLENH	COM time length configuration high bits	FB85H	-	-	-	-	COMLEN[19:16]			xxxx,0000	
BLINKRATE	Flicker Rate Configuration Register	FB86H	BLANKRATE[7:0]								1000,0000
LCDCR	LCD control register	FB87H	-	-	-	-	-	ACTMODE[1:0]		ENLCD	xxxx,x000
COMON	COM Line Enable Register	FB88H	-	-	-	-	COM3	COM2	COM1	COM0	xxxx,0000
SEGON1	SEG Line Enable Register 1	FB8AH	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0	0000,0000
SEGON2	SEG Line Enable Register 2	FB8BH	SEG15	SEG14	SEG13	SEG12	SEG11	SEG10	SEG9	SEG8	0000,0000
SEGON3	SEG Line Enable Register 3	FB8CH	SEG23	SEG22	SEG21	SEG20	SEG19	SEG18	SEG17	SEG16	0000,0000
SEGON4	SEG Line Enable Register 4	FB8DH	SEG31	SEG30	SEG29	SEG28	SEG27	SEG26	SEG25	SEG24	0000,0000
SEGON5	SEG Line Enable Register 5	FB8EH	SEG39	SEG38	SEG37	SEG36	SEG35	SEG34	SEG33	SEG32	0000,0000
C0SEGV0	C0SEG7_0 Data Register	FB90H	C0S7	C0S6	C0S5	C0S4	C0S3	C0S2	C0S1	C0S0	0000,0000
C0SEGV1	C0SEG15_8 Data Register	FB91H	C0S15	C0S14	C0S13	C0S12	C0S11	C0S10	C0S9	C0S8	0000,0000
C0SEGV2	C0SEG23_16 Data Register	FB92H	C0S23	C0S22	C0S21	C0S20	C0S19	C0S18	C0S17	C0S16	0000,0000
C0SEGV3	C0SEG31_24 Data Register	FB93H	C0S31	C0S30	C0S29	C0S28	C0S27	C0S26	C0S25	C0S24	0000,0000
C0SEGV4	C0SEG39_32 Data Register	FB94H	C0S39	C0S38	C0S37	C0S36	C0S35	C0S34	C0S33	C0S32	0000,0000
C1SEGV0	C1SEG7_0 Data Register	FB98H	C1S7	C1S6	C1S5	C1S4	C1S3	C1S2	C1S1	C1S0	0000,0000
C1SEGV1	C1SEG15_8 Data Register	FB99H	C1S15	C1S14	C1S13	C1S12	C1S11	C1S10	C1S9	C1S8	0000,0000
C1SEGV2	C1SEG23_16 Data Register	FB9AH	C1S23	C1S22	C1S21	C1S20	C1S19	C1S18	C1S17	C1S16	0000,0000
C1SEGV3	C1SEG31_24 Data Register	FB9BH	C1S31	C1S30	C1S29	C1S28	C1S27	C1S26	C1S25	C1S24	0000,0000
C1SEGV4	C1SEG39_32 Data Register	FB9CH	C1S39	C1S38	C1S37	C1S36	C1S35	C1S34	C1S33	C1S32	0000,0000
C2SEGV0	C2SEG7_0 Data Register	FBA0H	C2S7	C2S6	C2S5	C2S4	C2S3	C2S2	C2S1	C2S0	0000,0000
C2SEGV1	C2SEG15_8 Data Register	FBA1H	C2S15	C2S14	C2S13	C2S12	C2S11	C2S10	C2S9	C2S8	0000,0000
C2SEGV2	C2SEG23_16 Data Register	FBA2H	C2S23	C2S22	C2S21	C2S20	C2S19	C2S18	C2S17	C2S16	0000,0000
C2SEGV3	C2SEG31_24 Data Register	FBA3H	C2S31	C2S30	C2S29	C2S28	C2S27	C2S26	C2S25	C2S24	0000,0000
C2SEGV4	C2SEG39_32 Data Register	FBA4H	C2S39	C2S38	C2S37	C2S36	C2S35	C2S34	C2S33	C2S32	0000,0000
C3SEGV0	C3SEG7_0 Data Register	FBA8H	C3S7	C3S6	C3S5	C3S4	C3S3	C3S2	C3S1	C3S0	0000,0000
C3SEGV1	C3SEG15_8 Data Register	FBA9H	C3S15	C3S14	C3S13	C3S12	C3S11	C3S10	C3S9	C3S8	0000,0000
C3SEGV2	C3SEG23_16 Data Register	FBAAH	C3S23	C3S22	C3S21	C3S20	C3S19	C3S18	C3S17	C3S16	0000,0000
C3SEGV3	C3SEG31_24 Data Register	FBABH	C3S31	C3S30	C3S29	C3S28	C3S27	C3S26	C3S25	C3S24	0000,0000
C3SEGV4	C3SEG39_32 Data Register	FBACH	C3S39	C3S38	C3S37	C3S36	C3S35	C3S34	C3S33	C3S32	0000,0000

25.2.1 LCD configuration register (LCDCFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCDCFG	FB80H	CKSEL	-	-	-	VRLPSEL	VLCDSEL[2:0]		

CKSEL: LCD clock source selection

0: Select CPU clock as LCD clock source

1: Select external 32K crystal oscillator as LCD clock source

VRLPSEL: not used temporarily, it is recommended to set it to 0

VLCDSEL: VLCD voltage select bit

VLCDSEL[2:0]	VLCD
000	0.65*VCC
001	0.70*VCC
010	0.75*VCC
011	0.80*VCC
100	0.85*VCC
101	0.90*VCC
110	0.95*VCC
111	1.00*VCC

For example:

If the VCC is 3.3V and the LCD is 3.0V powered, VLCDSEL can be set to 101B, that is, the VLCD voltage can be set to $3.3V \times 0.90 = 2.97V$;

If the VCC is 5.0V and the LCD is 3.0V powered, VLCDSEL can be set to 000B, that is, set the VLCD voltage to $5.0V \times 0.65 = 3.25V$;

25.2.2 LCD configuration register 2 (LCDCFG2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCDCFG2	FB81H	-	-	-	-	SEG3PS	SEG2PS	SEG1PS	SEG0PS

SEG3PS: SEG3 control pin selection

SEG3PS	SEG3
0	P1.2
1	P7.4

SEG2PS: SEG2 control pin selection

SEG2PS	SEG2
0	P1.3
1	P7.5

SEG1PS: SEG1 control pin selection

SEG1PS	SEG1
0	P1.4
1	P7.6

SEG0PS: SEG0 control pin selection

SEG0PS	SEG0
0	P1.5
1	P7.7

25.2.3 Dead Time Length Configuration Register (DBLEN)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DBLEN	FB82H	-	-	-	-	-	DBLEN[2:0]		

DBLEN[2:0]: Set the length of dead time when LCD display

25.2.4 COM Time Length Configuration Registers (COMLENx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
COMLENL	FB83H	COMLEN[7:0]							
COMLENM	FB84H	COMLEN[15:8]							
COMLENH	FB85H	-	-	-	-	COMLEN[19:16]			

COMLEN[19:0]: Set the COM time length for LCD display

25.2.5 Flicker Rate Configuration Register (BLANKRATE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
BLANKRATE	FB86H	BLANKRATE[7:0]							

BLANKRATE[7:0]: Set the blinking rate of LCD in blinking mode

25.2.6 LCD Control Register (LCDCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LDCR	FB87H	-	-	-	-	-	ACTMODE[1:0]		ENLCD

ACTMODE[1:0]: Set LCD display mode

ACTMODE[1:0]	Mode
00	normal display mode
01	Always light mode
10	Always dark mode
11	Blink mode

ENLCD: LCD module enable control bit

0: disable LCD module

1: enable LCD module

25.2.7 COM Line Enable Register (COMON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
COMON	FB88H	-	-	-	-	COM3	COM2	COM1	COM0

COMn: COMn enable control bit (n=0~3)

0: disable COMn(The corresponding port is normal IO)

1: enable COMn

25.2.8 SEG Line Enable Register (SEGONx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SEGON1	FB8AH	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0
SEGON2	FB8BH	SEG15	SEG14	SEG13	SEG12	SEG11	SEG10	SEG9	SEG8
SEGON3	FB8CH	SEG23	SEG22	SEG21	SEG20	SEG19	SEG18	SEG17	SEG16
SEGON4	FB8DH	SEG31	SEG30	SEG29	SEG28	SEG27	SEG26	SEG25	SEG24
SEGON5	FB8EH	SEG39	SEG38	SEG37	SEG36	SEG35	SEG34	SEG33	SEG32

SEGn: SEGn enable control bit (n=0~39)

0: disable SEGn (The corresponding port is normal IO)

1: enable SEGn

25.2.9 LCD Data Registers (CxSEGVx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
C0SEGV0	FB90H	C0S7	C0S6	C0S5	C0S4	C0S3	C0S2	C0S1	C0S0
C0SEGV1	FB91H	C0S15	C0S14	C0S13	C0S12	C0S11	C0S10	C0S9	C0S8
C0SEGV2	FB92H	C0S23	C0S22	C0S21	C0S20	C0S19	C0S18	C0S17	C0S16

C0SEGV3	FB93H	C0S31	C0S30	C0S29	C0S28	C0S27	C0S26	C0S25	C0S24
C0SEGV4	FB94H	C0S39	C0S38	C0S37	C0S36	C0S35	C0S34	C0S33	C0S32
C1SEGV0	FB98H	C1S7	C1S6	C1S5	C1S4	C1S3	C1S2	C1S1	C1S0
C1SEGV1	FB99H	C1S15	C1S14	C1S13	C1S12	C1S11	C1S10	C1S9	C1S8
C1SEGV2	FB9AH	C1S23	C1S22	C1S21	C1S20	C1S19	C1S18	C1S17	C1S16
C1SEGV3	FB9BH	C1S31	C1S30	C1S29	C1S28	C1S27	C1S26	C1S25	C1S24
C1SEGV4	FB9CH	C1S39	C1S38	C1S37	C1S36	C1S35	C1S34	C1S33	C1S32
C2SEGV0	FBA0H	C2S7	C2S6	C2S5	C2S4	C2S3	C2S2	C2S1	C2S0
C2SEGV1	FBA1H	C2S15	C2S14	C2S13	C2S12	C2S11	C2S10	C2S9	C2S8
C2SEGV2	FBA2H	C2S23	C2S22	C2S21	C2S20	C2S19	C2S18	C2S17	C2S16
C2SEGV3	FBA3H	C2S31	C2S30	C2S29	C2S28	C2S27	C2S26	C2S25	C2S24
C2SEGV4	FBA4H	C2S39	C2S38	C2S37	C2S36	C2S35	C2S34	C2S33	C2S32
C3SEGV0	FBA8H	C3S7	C3S6	C3S5	C3S4	C3S3	C3S2	C3S1	C3S0
C3SEGV1	FBA9H	C3S15	C3S14	C3S13	C3S12	C3S11	C3S10	C3S9	C3S8
C3SEGV2	FBAAH	C3S23	C3S22	C3S21	C3S20	C3S19	C3S18	C3S17	C3S16
C3SEGV3	FBABH	C3S31	C3S30	C3S29	C3S28	C3S27	C3S26	C3S25	C3S24
C3SEGV4	FBACH	C3S39	C3S38	C3S37	C3S36	C3S35	C3S34	C3S33	C3S32

CmSn: Display data of COMm-SEgN in LCD matrix (m=0~3, n=0~39)

25.3 LCD display related configuration

25.3.1 Configure LCD refresh rate (frame rate)

The configuration of the refresh rate of the LCD refers to setting the time to scan all the dots of the entire LCD. Generally, the refresh rate of the LCD is configured to be 55 to 60 Hz, and the display effect is the best.

In the LCD module of STC8H, the refresh rate is mainly set by DBLEN, COMLENL, COMLENM, COMLENH and COMON registers.

$$\text{LCD refresh rate} = \frac{\text{LCD Clock frequency}}{(\text{DBLEN}[2:0] + \text{COMLEN}[19:0] + 1) * \text{number of COM}} \quad (\text{Hz})$$

For example: CPU operating frequency is 24MHz, select CPU clock as LCD clock source, enable COM0~COM3, then DBLEN can be set to 2, COMLEN can be set to 99997 (ie COMLENH=0x01, COMLENM=0x86, COMLENL=0x9D), thus The LCD refresh rate can be obtained as $24000000 / (4 * (2 + 99997 + 1)) = 60\text{Hz}$

25.3.2 Configure LCD Flicker Rate

When the display mode of the LCD is configured as blinking mode, the blinking rate of the LCD is set by the BLANKRATE register.

$$\text{LCD blink rate} = \frac{\text{LCD refresh rate}}{\text{BLANKRATE}[7:0] * 2} \quad (\text{Hz})$$

26 LCM interface (8/16-bit color screen module I8080/M6800 interface)

Product line	LCM
STC8H1K08 family	
STC8H1K28 family	
STC8H3K64S4 family	
STC8H3K64S2 family	
STC8H8K64U family A version	
STC8H8K64U family B version	●
STC8H2K64T family	
STC8H4K64TLR family	●
STC8H4K64TLCD family	●
STC8H4K64LCD family	●

Some microcontrollers of the STC8H series integrate an LCM interface controller, which can be used to drive the current popular LCD modules. It can drive I8080 interface and M6800 interface color screen, support 8-bit and 16-bit data width.

26.1 LCM interface function pin switch

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

LCMIFCPS[1:0]: LCM interface control pin selection bit

LCMIFCPS [1:0]	RS	RD signal of I8080 E signal of M6800	WR signal of I8080 RW signal of M6800
00	P4.5	P4.4	P4.2
01	P4.5	P3.7	P3.6
10	P4.0	P4.4	P4.2
11	P4.0	P3.7	P3.6

LCMIFDPS[1:0]: 8-bit data LCM interface data pin selection bit

LCMIFDPS [1:0]	D16_D8	Data byte DAT[7:0]
00	0	P2[7:0]
01	0	P6[7:0]
10	0	P2[7:0]
11	0	P6[7:0]

LCMIFDPS[1:0]: 16-bit data LCM interface data pin selection bit

LCMIFDPS [1:0]	D16_D8	High byte DAT[15:8]	Low byte DAT[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	{P0[7:4],P4[7],P4[6],P4[3],P4[1]}
11	1	P6[7:0]	P7[7:0]

26.2 Registers Related to LCM

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
LCMIFCFG	LCM Interface Configuration Register	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFCFG2	LCM Interface Configuration Register 2	FE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000
LCMIFCR	LCM Interface Control Register	FE52H	ENLCMIF	-	-	-	-	CMD[2:0]			0xxx,x000
LCMIFSTA	LCM Interface Status Register	FE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
LCMIDDATL	LCM Interface low byte data	FE54H	LCMIFDAT[7:0]								0000,0000
LCMIDDATH	LCM Interface high byte data	FE55H	LCMIFDAT[15:8]								0000,0000

● LCM Interface Configuration Register (LCMIFCFG)

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

LCMIFIE: LCM interrupt enable control bit

0: disable LCM interface interrupt

1: enable LCM interface interrupt

LCMIFIP[1:0]: LCM interface interrupt priority control bits

LCMIFIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

LCMIFDPS[1:0]: LCM interface data pin selection bit

LCMIFDPS [1:0]	D16_D8	High byte DAT[15:8]	Low byte DAT[7:0]
00	0	N/A	P2[7:0]
01	0	N/A	P6[7:0]

10	0	N/A	P2[7:0]
11	0	N/A	P6[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	{P0[7:4],P4[7],P4[6],P4[3],P4[1]}
11	1	P6[7:0]	P7[7:0]

D16_D8: LCM interface data width control bit

0: 8-bit data width

1: 16-bit data width

M68_I80: LCM interface mode selection bit

0: I8080 mode

1: M6800 mode

26.2.2 LCM Interface Configuration Register 2 (LCMIFCFG2)

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

LCMIFCPS[1:0]: LCM interface control pin selection bit

LCMIFCPS [1:0]	RS	RD signal of I8080 E signal of M6800	WR signal of I8080 RW signal of M6800
00	P4.5	P4.4	P4.2
01	P4.5	P3.7	P3.6
10	P4.0	P4.4	P4.2
11	P4.0	P3.7	P3.6

SETUPT[2:0]: Data setup time control bit for LCM interface communication (see timing diagrams in subsequent chapters for details)

HOLDT[1:0]: Data hold time control bit for LCM interface communication (see the timing diagram in the subsequent chapters for details)

26.2.3 LCM Interface Control Register (LCMIFCR)

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

ENLCMIF: LCM interface enable control bit

0: Disable LCM interface function

1: Enable LCM interface function

CMD[2:0]: LCM interface trigger command

CMD[2:0]	trigger command
100	Write command
101	Write data
110	Read command/status
111	Read data

26.2.4 LCM Interface Status Register (LCMIFSTA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFSTA	FE53H	-	-	-	-	-	-	-	LCMIFIF

LCMIFIF: LCM interface interrupt request flag, needs to be cleared by software

26.2.5 LCM interface Data Registers (LCMIFDATL, LCMIFDATH)

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

LCMIFDAT: LCM interface data register

When the data width is 8 bits, only the LCMDATL data is valid.

When the data width is 16 bits, it is combined into 16-bit data by LCMDATL and LCMDATH.

STC MCU

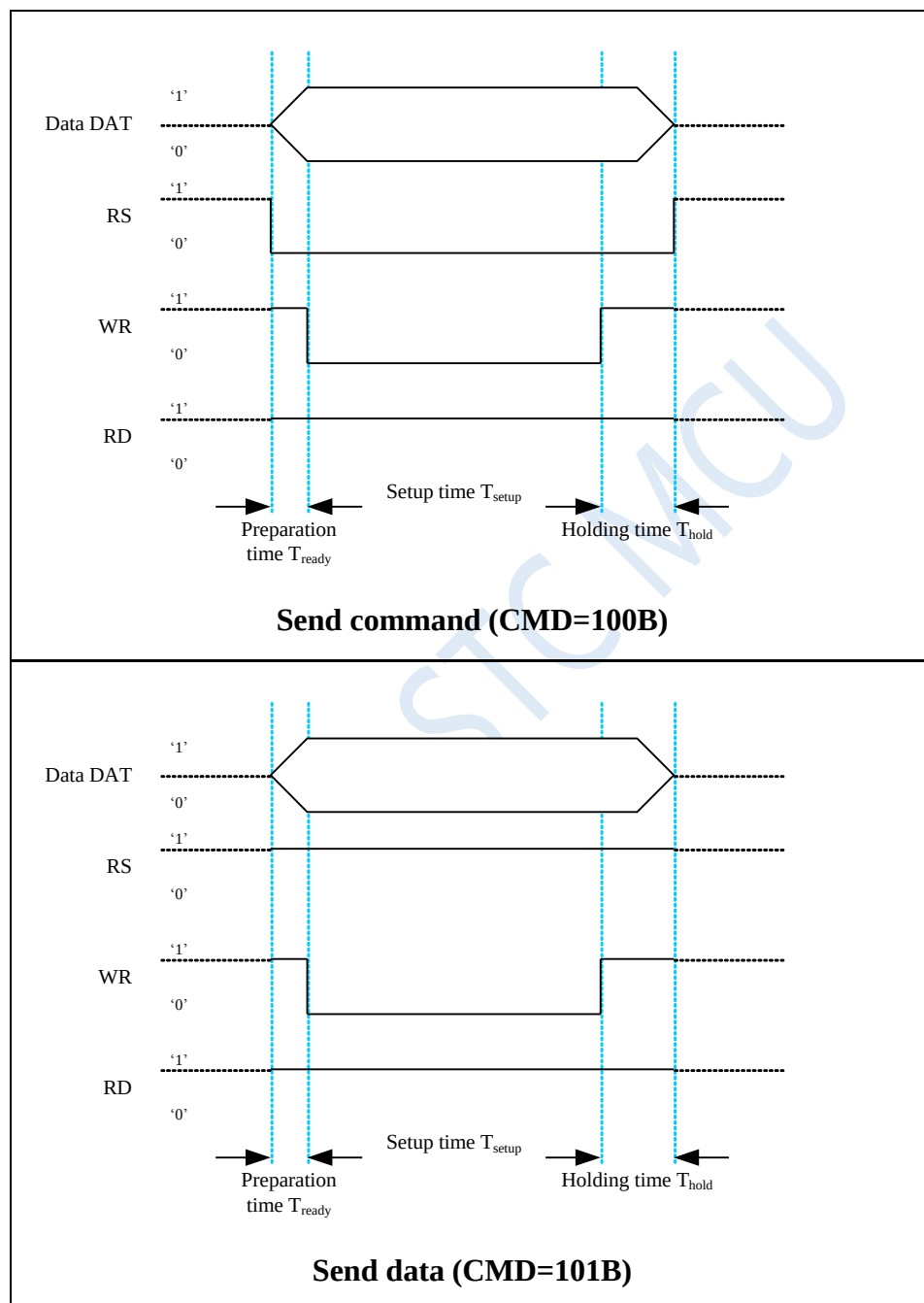
26.3 I8080/M6800 mode LCM interface timing diagram

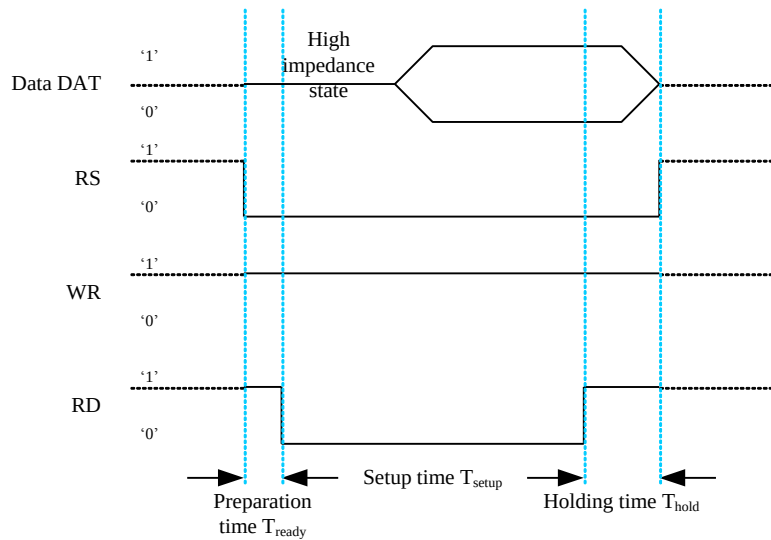
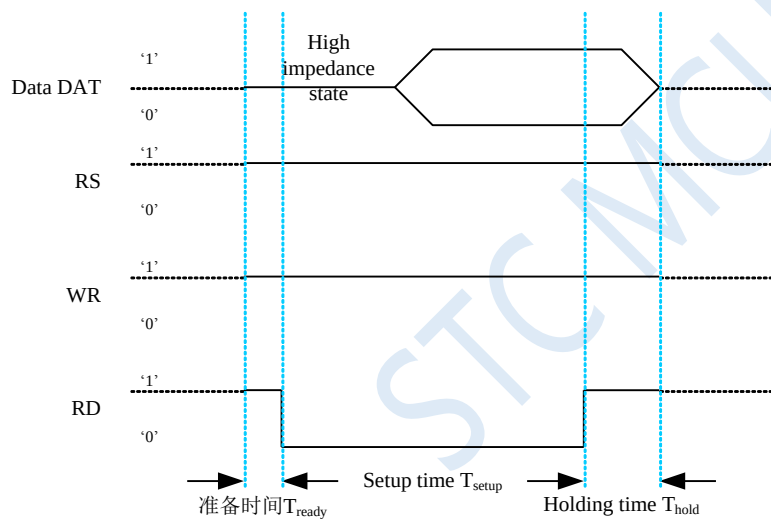
Note: $T_{\text{ready}} = 1$ system clock

$T_{\text{setup}} = (\text{SETUPT} + 1)$ system clocks

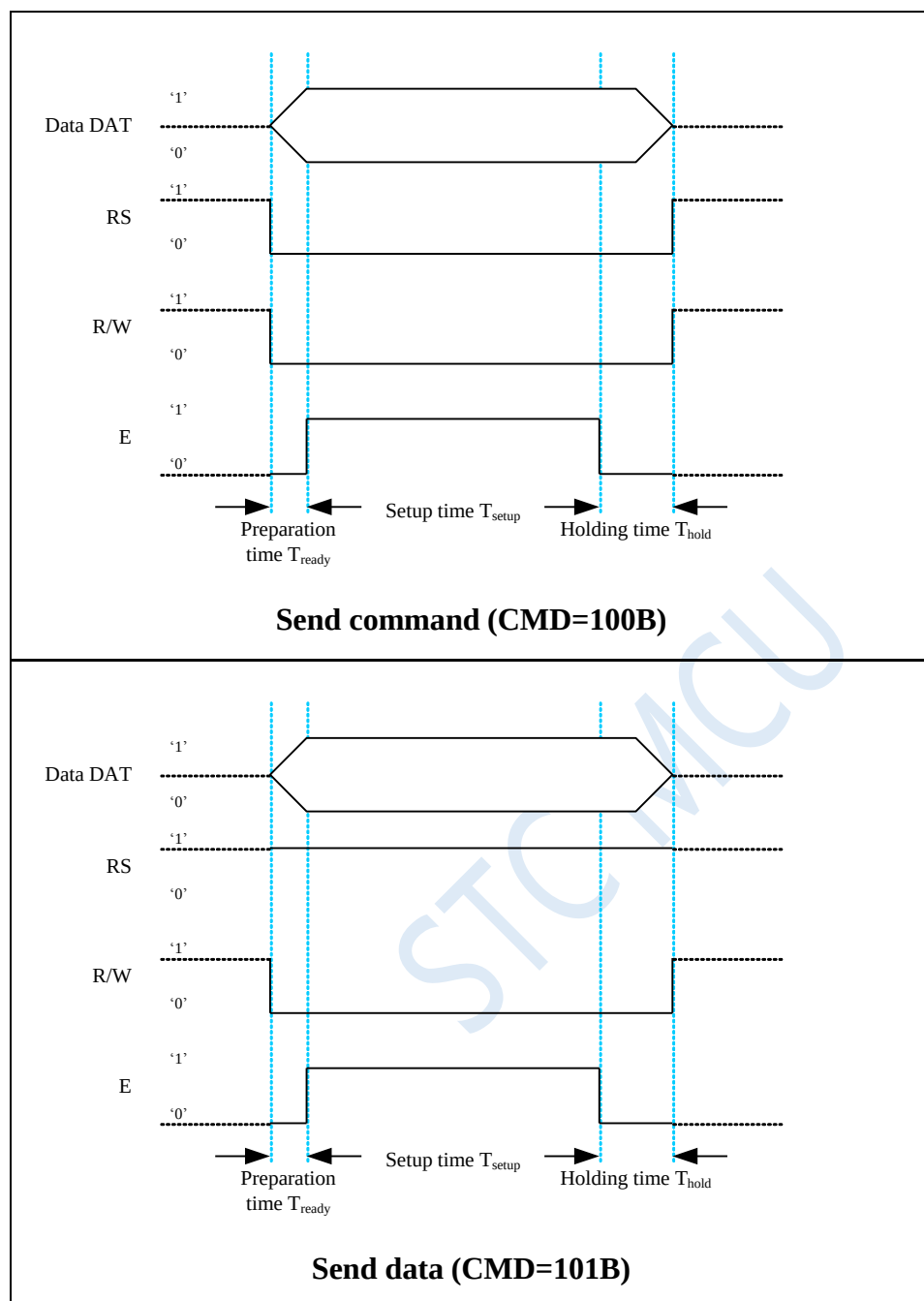
$T_{\text{hold}} = (\text{HOLDT} + 1)$ system clocks

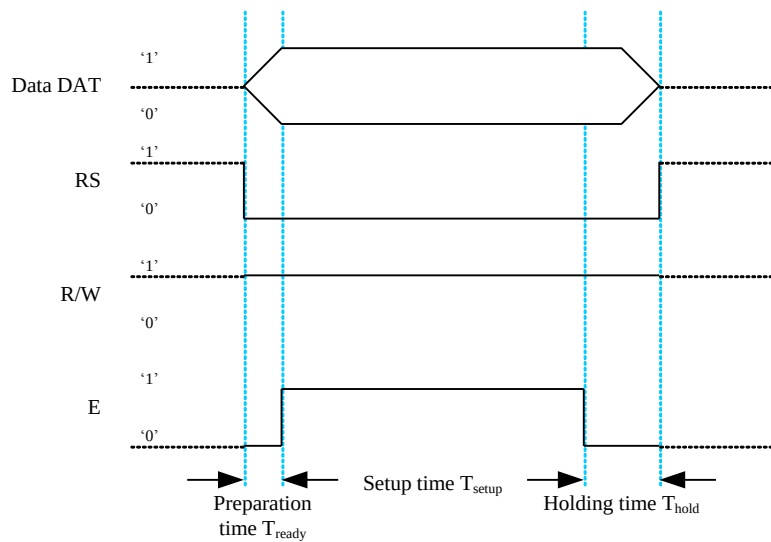
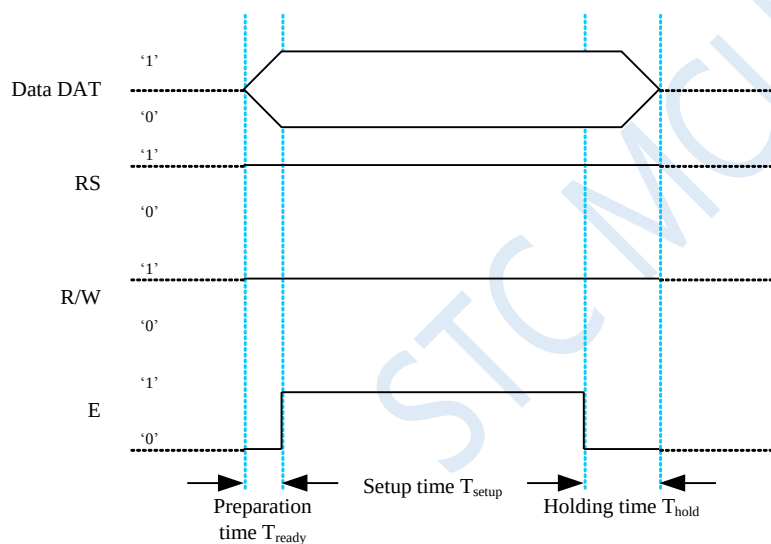
26.3.1 I8080 mode



**Read command (CMD=110B)****Read data (CMD=111B)**

26.3.2 M6800 mode



**Read command (CMD=110B)****Read data (CMD=111B)**

27 DMA (batch data transfer)

Product line	DMA
STC8H1K08 family	
STC8H1K28 family	
STC8H3K64S4 family	
STC8H3K64S2 family	
STC8H8K64U family A version	
STC8H8K64U family B version	●
STC8H2K64T family	
STC8H4K64TLR family	●
STC8H4K64TLCD family	●
STC8H4K64LCD family	●

Some microcomputers of the STC8H series support the function of batch data storage, that is, traditional DMA.

The following DMA operations are supported:

- M2M_DMA: read and write data from XRAM memory to XRAM memory
- ADC_DMA: automatically scan the enabled ADC channels and automatically store the converted ADC data into XRAM
- SPI_DMA: automatically exchange data between XRAM data and SPI peripherals
- UR1T_DMA: automatically send the data in XRAM through UART1
- UR1R_DMA: automatically store the data received from UART1 into XRAM
- UR2T_DMA: automatically send the data in XRAM through UART2
- UR2R_DMA: automatically store the data received from UART2 into XRAM
- UR3T_DMA: automatically send the data in XRAM through UART3
- UR3R_DMA: automatically store the data received from UART3 into XRAM
- UR4T_DMA: automatically send the data in XRAM through UART 4
- UR4R_DMA: automatically store the data received from UART4 into XRAM
- LCM_DMA: automatically exchange data between the data in XRAM and the LCM device

The maximum size of each DMA data transfer is 256 bytes.

Each DMA read and write operation to XRAM can be set to 4-level access priority, and the hardware will automatically perform the access arbitration of the XRAM bus, which will not affect CPU access to XRAM. Under the same priority, the access order of different DMAs to XRAM is as follows: SPI_DMA, UR1R_DMA, UR1T_DMA, UR2R_DMA, UR2T_DMA, UR3R_DMA, UR3T_DMA, UR4R_DMA, UR4T_DMA, LCM_DMA, M2M_DMA, ADC_DMA

27.1 Registers Related to DMA

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_M2M_CFG	M2M_DMA Configuration Register	FA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_CR	M2M_DMA Control Register	FA01H	ENM2M	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA Status Register	FA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_M2M_AMT	M2M_DMA Total Bytes Need to be Transferred	FA03H									0000,0000
DMA_M2M_DONE	M2M_DMA Transfer Completed Bytes	FA04H									0000,0000
DMA_M2M_TXAH	M2M_DMA Send High Address	FA05H									0000,0000
DMA_M2M_TXAL	M2M_DMA Send Low Address	FA06H									0000,0000
DMA_M2M_RXAH	M2M_DMA Receive High Address	FA07H									0000,0000
DMA_M2M_RXAL	M2M_DMA Receive Low Address	FA08H									0000,0000
DMA_ADC_CFG	ADC_DMA Configuration Register	FA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_CR	ADC_DMA Control Register	FA11H	ENADC	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_ADC_STA	ADC_DMA Status Register	FA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_ADC_RXAH	ADC_DMA Receive High Address	FA17H									0000,0000
DMA_ADC_RXAL	ADC_DMA Receive Low Address	FA18H									0000,0000

DMA_ADC_CFG2	ADC_DMA Configuration Register2	FA19H	-	-	-	-	CVTIMESEL[3:0]				xxxx,0000
DMA_ADC_CHSW0	ADC_DMA Channel Enable	FA1AH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	1000,0000
DMA_ADC_CHSW1	ADC_DMA Channel Enable	FA1BH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0	0000,0001
DMA_SPI_CFG	SPI_DMA Configuration Register	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_CR	SPI_DMA Control Register	FA21H	ENSP1	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO	000x,xxx0
DMA_SPI_STA	SPI_DMA Status Register	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_SPI_AMT	SPI_DMA Total Bytes Need to be Transferred	FA23H									0000,0000
DMA_SPI_DONE	SPI_DMA Transfer Completed Bytes	FA24H									0000,0000
DMA_SPI_TXAH	SPI_DMA Send High Address	FA25H									0000,0000
DMA_SPI_TXAL	SPI_DMA Send Low Address	FA26H									0000,0000
DMA_SPI_RXAH	SPI_DMA Receive High Address	FA27H									0000,0000
DMA_SPI_RXAL	SPI_DMA Receive Low Address	FA28H									0000,0000
DMA_SPI_CFG2	SPI_DMA Configuration Register2	FA29H	-	-	-	-	-	WRPSS	SSS[1:0]		xxxx,x000
DMA_UR1T_CFG	UR1T_DMA Configuration Register	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_CR	UR1T_DMA Control Register	FA31H	ENUR1T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA Status Register	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1T_AMT	UR1T_DMA Total Bytes Need to be Transferred	FA33H									0000,0000
DMA_UR1T_DONE	UR1T_DMA Transfer Completed Bytes	FA34H									0000,0000
DMA_UR1T_TXAH	UR1T_DMA Send High Address	FA35H									0000,0000
DMA_UR1T_TXAL	UR1T_DMA Send Low Address	FA36H									0000,0000
DMA_UR1R_CFG	UR1R_DMA Configuration Register	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_CR	UR1R_DMA Control Register	FA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR1R_STA	UR1R_DMA Status Register	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,x0x0
DMA_UR1R_AMT	UR1R_DMA Total Bytes Need to be Transferred	FA3BH									0000,0000
DMA_UR1R_DONE	UR1R_DMA Transfer Completed Bytes	FA3CH									0000,0000
DMA_UR1R_TXAH	UR1R_DMA Send High Address	FA3DH									0000,0000
DMA_UR1R_TXAL	UR1R_DMA Send Low Address	FA3EH									0000,0000
DMA_UR2T_CFG	UR2T_DMA Configuration Register	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_CR	UR2T_DMA Control Register	FA41H	ENUR2T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA Status Register	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA Total Bytes Need to be Transferred	FA43H									0000,0000
DMA_UR2T_DONE	UR2T_DMA Transfer Completed Bytes	FA44H									0000,0000
DMA_UR2T_TXAH	UR2T_DMA Send High Address	FA45H									0000,0000
DMA_UR2T_TXAL	UR2T_DMA Send Low Address	FA46H									0000,0000
DMA_UR2R_CFG	UR2R_DMA Configuration Register	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_CR	UR2R_DMA Control Register	FA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR2R_STA	UR2R_DMA Status Register	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,x0x0
DMA_UR2R_AMT	UR2R_DMA Total Bytes Need to be Transferred	FA4BH									0000,0000
DMA_UR2R_DONE	UR2R_DMA Transfer Completed Bytes	FA4CH									0000,0000
DMA_UR2R_TXAH	UR2R_DMA Send High Address	FA4DH									0000,0000
DMA_UR2R_TXAL	UR2R_DMA Send Low Address	FA4EH									0000,0000
DMA_UR3T_CFG	UR3T_DMA Configuration Register	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_CR	UR3T_DMA Control Register	FA51H	ENUR3T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA Status Register	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA Total Bytes Need to be Transferred	FA53H									0000,0000
DMA_UR3T_DONE	UR3T_DMA Transfer Completed Bytes	FA54H									0000,0000
DMA_UR3T_TXAH	UR3T_DMA Send High Address	FA55H									0000,0000
DMA_UR3T_TXAL	UR3T_DMA Send Low Address	FA56H									0000,0000
DMA_UR3R_CFG	UR3R_DMA Configuration Register	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000
DMA_UR3R_CR	UR3R_DMA Control Register	FA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR3R_STA	UR3R_DMA Status Register	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,x0x0
DMA_UR3R_AMT	UR3R_DMA Total Bytes Need to be Transferred	FA5BH									0000,0000
DMA_UR3R_DONE	UR3R_DMA Transfer Completed Bytes	FA5CH									0000,0000
DMA_UR3R_TXAH	UR3R_DMA Send High Address	FA5DH									0000,0000
DMA_UR3R_TXAL	UR3R_DMA Send Low Address	FA5EH									0000,0000
DMA_UR4T_CFG	UR4T_DMA Configuration Register	FA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_CR	UR4T_DMA Control Register	FA61H	ENUR4T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA Status Register	FA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA Total Bytes Need to be Transferred	FA63H									0000,0000

DMA_UR4T_DONE	UR4T_DMA Transfer Completed Bytes	FA64H									0000,0000
DMA_UR4T_TXAH	UR4T_DMA Send High Address	FA65H									0000,0000
DMA_UR4T_TXAL	UR4T_DMA Send Low Address	FA66H									0000,0000
DMA_UR4R_CFG	UR4R_DMA Configuration Register	FA68H	UR4RIE	-	-	-	-	UR4RIP[1:0]	UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_CR	UR4R_DMA Control Register	FA69H	ENUR4R	-	TRIG	-	-	-	-	CLRIFO	0x0x,xxx0
DMA_UR4R_STA	UR4R_DMA Status Register	FA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_UR4R_AMT	UR4R_DMA Total Bytes Need to be Transferred	FA6BH									0000,0000
DMA_UR4R_DONE	UR4R_DMA Transfer Completed Bytes	FA6CH									0000,0000
DMA_UR4R_TXAH	UR4R_DMA Send High Address	FA6DH									0000,0000
DMA_UR4R_TXAL	UR4R_DMA Send Low Address	FA6EH									0000,0000
DMA_LCM_CFG	LCM_DMA Configuration Register	FA70H	LCMIE	-	-	-	-	LCMIP[1:0]	LCMPTY[1:0]		0xxx,0000
DMA_LCM_CR	LCM_DMA Control Register	FA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	-	0000,0xxx
DMA_LCM_STA	LCM_DMA Status Register	FA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_LCM_AMT	LCM_DMA Total Bytes Need to be Transferred	FA73H									0000,0000
DMA_LCM_DONE	LCM_DMA Transfer Completed Bytes	FA74H									0000,0000
DMA_LCM_TXAH	LCM_DMA Send High Address	FA75H									0000,0000
DMA_LCM_TXAL	LCM_DMA Send Low Address	FA76H									0000,0000
DMA_LCM_RXAH	LCM_DMA Receive High Address	FA77H									0000,0000
DMA_LCM_RXAL	LCM_DMA Receive Low Address	FA78H									0000,0000

27.2 Data read and write between memory and memory (M2M_DMA)

27.2.1 M2M_DMA Configuration Register (DMA_M2M_CFG)

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

M2MIE: M2M_DMA interrupt enable control bit

0: Disable M2M_DMA interrupt

1: Enable M2M_DMA interrupt

TXACO: M2M_DMA source address (read address) changes direction

0: The address is automatically incremented after the data read is completed

1: The address is automatically decremented after the data read is completed

RXACO: M2M_DMA target address (write address) changed direction

0: The address is automatically incremented after data writing is completed

1: The address is automatically decremented after data writing is completed

M2MIP[1:0]: M2M_DMA interrupt priority control bits

M2MIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

M2MPTY[1:0]: M2M_DMA Data bus access priority control bits

M2MPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.2.2 M2M_DMA Control Register (DMA_M2M_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CR	FA01H	ENM2M	TRIG	-	-	-	-	-	-

ENM2M: M2M_DMA function enable control bit

0: Disable M2M_DMA function

1: Enable M2M_DMA function

TRIG: M2M_DMA data read and write trigger control bit

0: Write 0 is invalid

1: Write 1 to start M2M_DMA operation.

27.2.3 M2M_DMA Status Register (DMA_M2M_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	FA02H	-	-	-	-	-	-	-	M2MIF

M2MIF: M2M_DMA interrupt request flag bit. When the M2M_DMA operation is completed, the hardware automatically sets M2MIF to 1, and if the M2M_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

27.2.4 M2M_DMA transfer total byte register (DMA_M2M_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_AMT	FA03H								

DMA_M2M_AMT: Set the number of bytes that need to read and write data.

Note: The actual number of bytes read and written is (DMA_M2M_AMT+1), that is, when DMA_M2M_AMT is set to 0, 1 byte is read and written, and when DMA_M2M_AMT is set to 255, 256 bytes are read and written.

27.2.5 M2M_DMA transfer complete byte register (DMA_M2M_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_DONE	FA04H								

DMA_M2M_DONE: The number of bytes that have been read and written currently.

27.2.6 M2M_DMA Send Address Registers (DMA_M2M_TXAx)

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

DMA_M2M_TXA: Set the source address when reading and writing data. Data will be read from this address when the M2M_DMA operation is performed.

27.2.7 M2M_DMA Receive Address Registers (DMA_M2M_RXAx)

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

DMA_M2M_RXA: Set the target address when reading and writing data. Data will be written from this address when the M2M_DMA operation is performed.

27.3 ADC Automatic Data Storage (ADC_DMA)

27.3.1 ADC_DMA Configuration Register (DMA_ADC_CFG)

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

ADCIE: ADC_DMA interrupt enable control bit

0: Disable ADC_DMA interrupt

1: Enable ADC_DMA interrupt

ADCIP[1:0]: ADC_DMA interrupt priority control bits

ADCIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

ADCPTY[1:0]: ADC_DMA Data bus access priority control bits

ADCPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.3.2 ADC_DMA Control Register (DMA_ADC_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CR	FA11H	ENADC	TRIG	-	-	-	-	-	-

ENADC: ADC_DMA function enable control bit

0: Disable ADC_DMA function

1: Enable ADC_DMA function

TRIG: ADC_DMA operation trigger control bit

0: Write 0 is invalid

1: Write 1 to start ADC_DMA operation.

27.3.3 ADC_DMA Status Register (DMA_ADC_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_STA	FA12H	-	-	-	-	-	-	-	ADCIF

ADCIF: ADC_DMA interrupt request flag bit. After ADC_DMA completes scanning all enabled ADC channels, the hardware automatically sets ADCIF to 1. If the ADC_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

27.3.4 ADC_DMA Receive Address Registers (DMA_ADC_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_RXAH	FA17H	ADDR[15:8]							
DMA_ADC_RXAL	FA18H	ADDR[7:0]							

DMA_ADC_RXA: Set the storage address of ADC conversion data during ADC_DMA operation.

27.3.5 ADC_DMA Configuration Register 2 (DMA_ADC_CFG2)

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

CVTIMESEL[3:0]: Set the number of ADC conversions for each ADC channel during ADC_DMA operation.

CVTIMESEL[3:0]	Number of ADC conversions
0xxx	1
1000	2
1001	4
1010	8
1011	16
1100	32
1101	64
1110	128
1111	256

27.3.6 ADC_DMA Channel Enable Registers (DMA_ADC_CHSWx)

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

CHn: The ADC channel to be scanned automatically when setting ADC_DMA operation. Channel scanning always starts from the lower-numbered channel.

27.3.7 Data storage format of ADC_DMA

Note: ADC conversion speed and conversion result alignment are set by ADC related registers.

XRAM[DMA_ADC_RXA+0] = high byte of the 1st ADC conversion result of the 1st enabled channel;
 XRAM[DMA_ADC_RXA+1] = low byte of the 1st ADC conversion result of the 1st enabled channel;
 XRAM[DMA_ADC_RXA+2] = high byte of the 2nd ADC conversion result of the 1st enabled channel;
 XRAM[DMA_ADC_RXA+3] = low byte of the 2nd ADC conversion result of the 1st enabled channel;
 ...
 XRAM[DMA_ADC_RXA+2n-2] = high byte of the nth ADC conversion result of the 1st enabled channel;
 XRAM[DMA_ADC_RXA+2n-1] = low byte of the nth ADC conversion result of the 1st enabled channel;
 XRAM[DMA_ADC_RXA+2n] = ADC channel number of 1st channel;
 XRAM[DMA_ADC_RXA+2n+1] = remainder after the average value of the n ADC conversion results of the 1st channel;
 XRAM[DMA_ADC_RXA+2n+2] = high byte of the average value of the n ADC conversion results of the 1st channel;
 XRAM[DMA_ADC_RXA+2n+3] = low byte of the average value of the n ADC conversion results of the 1st channel;

 XRAM[DMA_ADC_RXA+(2n+3)+0] = high byte of the 1st ADC conversion result of the 2nd enabled channel;
 XRAM[DMA_ADC_RXA+(2n+3)+1] = low byte of the 1st ADC conversion result of the 2nd enabled channel;
 XRAM[DMA_ADC_RXA+(2n+3)+2] = high byte of the 2nd ADC conversion result of the 2nd enabled channel;
 XRAM[DMA_ADC_RXA+(2n+3)+3] = low byte of the 2nd ADC conversion result of the 2nd enabled channel;
 ...
 XRAM[DMA_ADC_RXA+(2n+3)+2n-2] = high byte of the nth ADC conversion result of the enabled 2nd channel;
 XRAM[DMA_ADC_RXA+(2n+3)+2n-1] = low byte of the nth ADC conversion result of the enabled 2nd channel;
 XRAM[DMA_ADC_RXA+(2n+3)+2n] = ADC channel number of 2nd channel;
 XRAM[DMA_ADC_RXA+(2n+3)+2n+1] = remainder after the average value of the n ADC conversion results of 2nd channel;
 XRAM[DMA_ADC_RXA+(2n+3)+2n+2] = high byte of the average value of n ADC conversion results of 2nd channel;
 XRAM[DMA_ADC_RXA+(2n+3)+2n+3] = low byte of the average value of n ADC conversion results of 2nd channel;
 ...
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+0] = high byte of the 1st ADC conversion result of the enabled mth channel;
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+1] = low byte of the 1st ADC conversion result of the enabled mth channel;
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2] = high byte of the 2nd ADC conversion result of the enabled mth channel;
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+3] = low byte of the 2nd ADC conversion result of the enabled mth channel;
 ...
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n-2] = high byte of the nth ADC conversion result of the enabled mth channel;
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n-1] = low byte of the nth ADC conversion result of the enabled mth channel;
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n] = ADC channel number of the mth channel;
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n+1] = remainder after the average value of the n ADC conversion results of the mth channel;
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n+2] = high byte of the average value of the n ADC conversion results of the mth channel;
 XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n+3] = low byte of the average value of the n ADC conversion results of the mth channel;

The form is as follows:

ADC channel	Offset address	Data
1 st channel	0	high byte of the 1st ADC conversion result of the 1st enabled channel
	1	low byte of the 1st ADC conversion result of the 1st enabled channel
	2	high byte of the 2nd ADC conversion result of the 1st enabled channel
	3	low byte of the 2nd ADC conversion result of the 1st enabled channel
	...	...
	2n-2	high byte of the nth ADC conversion result of the 1st enabled channel
	2n-1	low byte of the nth ADC conversion result of the 1st enabled channel
	2n	ADC channel number of 1st channel
	2n+1	remainder after the average value of the n ADC conversion results of the 1st channel
	2n+2	high byte of the average value of the n ADC conversion results of the 1st channel
	2n+3	low byte of the average value of the n ADC conversion results of the 1st channel
2 nd channel	(2n+3) + 0	high byte of the 1st ADC conversion result of the 2nd enabled channel
	(2n+3) + 1	low byte of the 1st ADC conversion result of the 2nd enabled channel
	(2n+3) + 2	high byte of the 2nd ADC conversion result of the 2nd enabled channel
	(2n+3) + 3	low byte of the 2nd ADC conversion result of the 2nd enabled channel
	...	...
	(2n+3) + 2n-2	high byte of the nth ADC conversion result of the enabled 2nd channel
	(2n+3) + 2n-1	low byte of the nth ADC conversion result of the enabled 2nd channel
	(2n+3) + 2n	ADC channel number of 2nd channel
	(2n+3) + 2n+1	remainder after the average value of the n ADC conversion results of 2nd channel
	(2n+3) + 2n+2	high byte of the average value of n ADC conversion results of 2nd channel
	(2n+3) + 2n+3	low byte of the average value of n ADC conversion results of 2nd channel
	...	...
mth channel	(m-1)(2n+3) + 0	high byte of the 1st ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 1	low byte of the 1st ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 2	high byte of the 2nd ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 3	low byte of the 2nd ADC conversion result of the enabled mth channel
	...	...
	(m-1)(2n+3) + 2n-2	high byte of the nth ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 2n-1	low byte of the nth ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 2n	ADC channel number of the mth channel
	(m-1)(2n+3) + 2n+1	remainder after the average value of the n ADC conversion results of the mth channel
	(m-1)(2n+3) + 2n+2	high byte of the average value of the n ADC conversion results of the mth channel
	(m-1)(2n+3) + 2n+3	low byte of the average value of the n ADC conversion results of the mth channel

27.4 Data exchange between SPI and memory (SPI_DMA)

27.4.1 SPI_DMA Configuration Register (DMA_SPI_CFG)

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

SPIIE: SPI_DMA interrupt enable control bit

0: Disable SPI_DMA interrupt

1: Enable SPI_DMA interrupt

ACT_TX: SPI_DMA transmit data control bit

0: Disable SPI_DMA to send data. In master mode, SPI only sends clock to SCLK port, but does not read data from XRAM, nor send data to MOSI port; in slave mode, SPI does not read data from XRAM, nor send data to MISO port.

1: Enable SPI_DMA to send data. In master mode, SPI sends clock to SCLK port, and reads data from XRAM and sends data to MOSI port; in slave mode, SPI reads data from XRAM and sends data to MISO port.

ACT_RX: SPI_DMA receive data control bit

0: Disable SPI_DMA to receive data. In master mode, SPI only sends clock to SCLK port, but does not read data from MISO port, nor write data to XRAM; in slave mode, SPI does not read data from MOSI port, nor write data to XRAM.

1: Enable SPI_DMA to receive data. In master mode, SPI sends clock to SCLK port, and reads data from MISO port and writes data to XRAM; in slave mode, SPI reads data from MOSI port and writes XRAM.

SPIIP[1:0]: SPI_DMA interrupt priority control bits

SPIIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

SPIPTY[1:0]: SPI_DMA Data bus access priority control bits

SPIPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.4.2 SPI_DMA Control Register (DMA_SPI_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CR	FA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRIFO

ENSPI: SPI_DMA function enable control bit

0: Disable SPI_DMA function

1: Enable SPI_DMA function

TRIG_M: SPI_DMA master mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start SPI_DMA master mode operation.

TRIG_S: SPI_DMA slave mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start SPI_DMA slave mode operation.

CLRIFO: Clear SPI_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the SPI_DMA operation, clear the built-in FIFO of SPI_DMA firstly.

27.4.3 SPI_DMA Status Register (DMA_SPI_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_STA	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF

SPIIF: SPI_DMA interrupt request flag bit. After the SPI_DMA data exchange is completed, the hardware automatically sets SPIIF to 1. If the SPI_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

RXLOSS: SPI_DMA receive data discard flag. During the SPI_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the SPI_DMA and the data received by the SPI_DMA is automatically discarded, the hardware automatically sets RXLOSS to 1. The flag bit needs to be cleared by software.

TXOVW: SPI_DMA data coverage flag. During the data transfer process of SPI_DMA, when the host mode SPI writes the SPDAT register to trigger the SPI data transfer again, the data transfer will fail, and the hardware will

automatically set TXOVW to 1. The flag bit needs to be cleared by software.

27.4.4 SPI_DMA transfer total byte register (DMA_SPI_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_AMT	FA23H								

DMA_SPI_AMT: Set the number of bytes that need to read and write data.

Note: The actual number of bytes read and written is (DMA_SPI_AMT+1), that is, when DMA_SPI_AMT is set to 0, 1 byte is transferred, and when DMA_SPI_AMT is set to 255, 256 bytes are transferred.

27.4.5 SPI_DMA transfer complete byte register (DMA_SPI_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_DONE	FA24H								

DMA_SPI_DONE: The number of bytes that have been read and written currently.

27.4.6 SPI_DMA Send Address Registers (DMA_SPI_TXAx)

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

DMA_SPI_TXA: Set the source address when reading and writing data. Data will be read from this address when the SPI_DMA operation is performed.

27.4.7 SPI_DMA Receive Address Registers (DMA_SPI_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_RXAH	FA27H	ADDR[15:8]							
DMA_SPI_RXAL	FA28H	ADDR[7:0]							

DMA_SPI_RXA: Set the target address when reading and writing data. Data will be written from this address when the SPI_DMA operation is performed.

27.4.8 SPI_DMA Configuration Register 2 (DMA_SPI_CFG2)

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

WRPSS: Enable SS pin control bit during SPI_DMA process

0: During the SPI_DMA transfer process, the SS pin is not automatically controlled

1: During the SPI_DMA transfer process, the SS pin is automatically pulled down, and the original state is automatically restored after the transfer is completed.

SSS[1:0]: During the SPI_DMA process, control the SS selection bit automatically

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

27.5 Data exchange between UART1 and memory (UR1T_DMA, UR1R_DMA)

27.5.1 UR1T_DMA Configuration Register (DMA_UR1T_CFG)

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

UR1TIE: UR1T_DMA interrupt enable control bit

0: Disable UR1T_DMA interrupt

1: Enable UR1T_DMA interrupt

UR1TIP[1:0]: UR1T_DMA interrupt priority control bits

UR1TIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR1TPTY[1:0]: UR1T_DMA Data bus access priority control bits

UR1TPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.5.2 UR1T_DMA Control Register (DMA_UR1T_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CR	FA31H	ENUR1T	TRIG	-	-	-	-	-	-

ENUR1T: UR1T_DMA function enable control bit

0: Disable UR1T_DMA function

1: Enable UR1T_DMA function

TRIG: UR1T_DMA UART1 transmit trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR1T_DMA automatically sending data.

27.5.3 UR1T_DMA Status Register (DMA_UR1T_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_STA	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF

UR1TIF: UR1T_DMA interrupt request flag bit. When the UR1T_DMA data transmission is completed, the hardware automatically sets UR1TIF to 1, and if the UR1T_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

TXOVW: UR1T_DMA data coverage flag. When UR1T_DMA is in the process of data transmission, and the UART writes the SBUF register to trigger the UART to send data again, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

27.5.4 UR1T_DMA transfer total byte register (DMA_UR1T_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_AMT	FA33H								

DMA_UR1T_AMT: Set the number of bytes of data that needs to be automatically sent.

Note: The actual number of bytes is (DMA_UR1T_AMT+1), that is, when DMA_UR1T_AMT is set to 0, 1

byte is transferred, and when DMA_UR1T_AMT is set to 255, 256 bytes are transferred.

27.5.5 UR1T_DMA transfer complete byte register (DMA_UR1T_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_DONE	FA34H								

DMA_UR1T_DONE: The number of bytes that have been sent so far.

27.5.6 UR1T_DMA Send Address Registers (DMA_UR1T_TXAx)

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

DMA_UR1T_TXA: Set the source address of automatic data transmission. Data is read from this address when performing a UR1T_DMA operation.

27.5.7 UR1R_DMA Configuration Register (DMA_UR1R_CFG)

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

UR1RIE: UR1R_DMA interrupt enable control bit

0: Disable UR1R_DMA interrupt

1: Enable UR1R_DMA interrupt

UR1RIP[1:0]: UR1R_DMA interrupt priority control bits

UR1RIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR1RPTY[1:0]: UR1R_DMA Data bus access priority control bits

UR1RPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.5.8 UR1R_DMA Control Register (DMA_UR1R_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CR	FA39H	ENUR1R	-	TRIG	-	-	-	-	CLRIFO

ENUR1R: UR1R_DMA function enable control bit

0: Disable UR1R_DMA function

1: Enable UR1R_DMA function

TRIG: UR1R_DMA UART1 receive trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR1R_DMA receiving data automatically

CLRIFO: Clear UR1R_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the UR1R_DMA operation, clear the built-in FIFO of the UR1R_DMA firstly

27.5.9 UR1R_DMA Status Register (DMA_UR1R_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_STA	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF

UR1RIF: UR1R_DMA interrupt request flag bit. When UR1R_DMA receives data, the hardware will automatically set UR1RIF to 1. If the UR1R_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR1R_DMA receive data discard flag. During the UR1R_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the UR1R_DMA and the data received by the UR1R_DMA is automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

27.5.10 UR1R_DMA transfer total byte register (DMA_UR1R_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_AMT	FA3BH								

DMA_UR1R_AMT: Set the number of data bytes that need to automatically receive.

Note: The actual number of bytes is (DMA_UR1R_AMT+1), that is, when DMA_UR1R_AMT is set to 0, 1 byte is transferred, and when DMA_UR1R_AMT is set to 255, 256 bytes are transferred.

27.5.11 UR1R_DMA transfer complete byte register (DMA_UR1R_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_DONE	FA3CH								

DMA_UR1R_DONE: The number of bytes that have been received currently.

27.5.12 UR1R_DMA Receive Address Registers (DMA_UR1T_RXAx)

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

DMA_UR1R_RXA: Set the target address for automatically receiving data. Data will be written from this address when performing a UR1R_DMA operation.

27.6 Data exchange between UART2 and memory (UR2T_DMA, UR2R_DMA)

27.6.1 UR2T_DMA Configuration Register (DMA_UR2T_CFG)

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

UR2TIE: UR2T_DMA interrupt enable control bit

0: Disable UR2T_DMA interrupt

1: Enable UR2T_DMA interrupt

UR2TIP[1:0]: UR2T_DMA interrupt priority control bits

UR2TIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR2TPTY[1:0]: UR2T_DMA Data bus access priority control bits

UR2TPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.6.2 UR2T_DMA Control Register (DMA_UR2T_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CR	FA41H	ENUR2T	TRIG	-	-	-	-	-	-

ENUR2T: UR2T_DMA function enable control bit

0: Disable UR2T_DMA function

1: Enable UR2T_DMA function

TRIG: UR2T_DMA UART1 transmit trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR2T_DMA automatically sending data.

27.6.3 UR2T_DMA Status Register (DMA_UR2T_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_STA	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF

UR2TIF: UR2T_DMA interrupt request flag bit. When the UR2T_DMA data transmission is completed, the hardware automatically sets UR2TIF to 1, and if the UR2T_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

TXOVW: UR2T_DMA data coverage flag. When UR2T_DMA is in the process of data transmission, and the UART writes the SBUF register to trigger the UART to send data again, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

27.6.4 UR2T_DMA transfer total byte register (DMA_UR2T_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_AMT	FA43H								

DMA_UR2T_AMT: Set the number of bytes of data that needs to be automatically sent.

Note: The actual number of bytes is (DMA_UR2T_AMT+1), that is, when DMA_UR2T_AMT is set to 0, 1

byte is transferred, and when DMA_UR2T_AMT is set to 255, 256 bytes are transferred.

27.6.5 UR2T_DMA transfer complete byte register (DMA_UR2T_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_DONE	FA44H								

DMA_UR2T_DONE: The number of bytes that have been sent so far.

27.6.6 UR2T_DMA Send Address Registers (DMA_UR2T_TXAx)

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

DMA_UR2T_TXA: Set the source address of automatic data transmission. Data is read from this address when performing a UR2T_DMA operation.

27.6.7 UR2R_DMA Configuration Register (DMA_UR2R_CFG)

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

UR2RIE: UR2R_DMA interrupt enable control bit

0: Disable UR2R_DMA interrupt

1: Enable UR2R_DMA interrupt

UR2RIP[1:0]: UR2R_DMA interrupt priority control bits

UR2RIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR2RPTY[1:0]: UR2R_DMA Data bus access priority control bits

UR2RPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.6.8 UR2R_DMA Control Register (DMA_UR2R_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CR	FA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO

ENUR2R: UR2R_DMA function enable control bit

0: Disable UR2R_DMA function

1: Enable UR2R_DMA function

TRIG: UR2R_DMA UART1 receive trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR2R_DMA receiving data automatically

CLRFIFO: Clear UR2R_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the UR2R_DMA operation, clear the built-in FIFO of the UR2R_DMA firstly

27.6.9 UR2R_DMA Status Register (DMA_UR2R_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0

DMA_UR2R_STA	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF

UR2RIF: UR2R_DMA interrupt request flag bit. When UR2R_DMA receives data, the hardware will automatically set UR2RIF to 1. If the UR2R_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR2R_DMA receive data discard flag. During the UR2R_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the UR2R_DMA and the data received by the UR2R_DMA is automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

27.6.10 UR2R_DMA transfer total byte register (DMA_UR2R_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_AMT	FA4BH								

DMA_UR2R_AMT: Set the number of data bytes that need to automatically receive.

Note: The actual number of bytes is (DMA_UR2R_AMT+1), that is, when DMA_UR2R_AMT is set to 0, 1 byte is transferred, and when DMA_UR2R_AMT is set to 255, 256 bytes are transferred.

27.6.11 UR2R_DMA transfer complete byte register (DMA_UR2R_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_DONE	FA4CH								

DMA_UR2R_DONE: The number of bytes that have been received currently.

27.6.12 UR2R_DMA Receive Address Registers (DMA_UR2T_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_RXAH	FA4DH								
DMA_UR2R_RXAL	FA4EH								

DMA_UR2R_RXA: Set the target address for automatically receiving data. Data will be written from this address when performing a UR2R_DMA operation.

27.7 Data exchange between UART3 and memory (UR3T_DMA, UR3R_DMA)

27.7.1 UR3T_DMA Configuration Register (DMA_UR3T_CFG)

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

UR3TIE: UR3T_DMA interrupt enable control bit

0: Disable UR3T_DMA interrupt

1: Enable UR3T_DMA interrupt

UR3TIP[1:0]: UR3T_DMA interrupt priority control bits

UR3TIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR3TPTY[1:0]: UR3T_DMA Data bus access priority control bits

UR3TPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.7.2 UR3T_DMA Control Register (DMA_UR3T_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CR	FA51H	ENUR3T	TRIG	-	-	-	-	-	-

ENUR3T: UR3T_DMA function enable control bit

0: Disable UR3T_DMA function

1: Enable UR3T_DMA function

TRIG: UR3T_DMA UART1 transmit trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR3T_DMA automatically sending data.

27.7.3 UR3T_DMA Status Register (DMA_UR3T_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_STA	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF

UR3TIF: UR3T_DMA interrupt request flag bit. When the UR3T_DMA data transmission is completed, the hardware automatically sets UR3TIF to 1, and if the UR3T_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

TXOVW: UR3T_DMA data coverage flag. When UR3T_DMA is in the process of data transmission, and the UART writes the SBUF register to trigger the UART to send data again, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

27.7.4 UR3T_DMA transfer total byte register (DMA_UR3T_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_AMT	FA53H								

DMA_UR3T_AMT: Set the number of bytes of data that needs to be automatically sent.

Note: The actual number of bytes is (DMA_UR3T_AMT+1), that is, when DMA_UR3T_AMT is set to 0, 1

byte is transferred, and when DMA_UR3T_AMT is set to 255, 256 bytes are transferred.

27.7.5 UR3T_DMA transfer complete byte register (DMA_UR3T_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_DONE	FA54H								

DMA_UR3T_DONE: The number of bytes that have been sent so far.

27.7.6 UR3T_DMA Send Address Registers (DMA_UR3T_TXAx)

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

DMA_UR3T_TXA: Set the source address of automatic data transmission. Data is read from this address when performing a UR3T_DMA operation.

27.7.7 UR3R_DMA Configuration Register (DMA_UR3R_CFG)

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

UR3RIE: UR3R_DMA interrupt enable control bit

0: Disable UR3R_DMA interrupt

1: Enable UR3R_DMA interrupt

UR3RIP[1:0]: UR3R_DMA interrupt priority control bits

UR3RIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR3RPTY[1:0]: UR3R_DMA Data bus access priority control bits

UR3RPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.7.8 UR3R_DMA Control Register (DMA_UR3R_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CR	FA59H	ENUR3R	-	TRIG	-	-	-	-	CLRIFO

ENUR3R: UR3R_DMA function enable control bit

0: Disable UR3R_DMA function

1: Enable UR3R_DMA function

TRIG: UR3R_DMA UART1 receive trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR3R_DMA receiving data automatically

CLRIFO: Clear UR3R_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the UR3R_DMA operation, clear the built-in FIFO of the UR3R_DMA firstly

27.7.9 UR3R_DMA Status Register (DMA_UR3R_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_STA	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF

UR3RIF: UR3R_DMA interrupt request flag bit. When UR3R_DMA receives data, the hardware will automatically set UR3RIF to 1. If the UR3R_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR3R_DMA receive data discard flag. During the UR3R_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the UR3R_DMA and the data received by the UR3R_DMA is automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

27.7.10 UR3R_DMA transfer total byte register (DMA_UR3R_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_AMT	FA5BH								

DMA_UR3R_AMT: Set the number of data bytes that need to automatically receive.

Note: The actual number of bytes is (DMA_UR3R_AMT+1), that is, when DMA_UR3R_AMT is set to 0, 1 byte is transferred, and when DMA_UR3R_AMT is set to 255, 256 bytes are transferred.

27.7.11 UR3R_DMA transfer complete byte register (DMA_UR3R_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_DONE	FA5CH								

DMA_UR3R_DONE: The number of bytes that have been received currently.

27.7.12 UR3R_DMA Receive Address Registers (DMA_UR3T_RXAx)

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

DMA_UR3R_RXA: Set the target address for automatically receiving data. Data will be written from this address when performing a UR3R_DMA operation.

27.8 Data exchange between UART4 and memory (UR4T_DMA, UR4R_DMA)

27.8.1 UR4T_DMA Configuration Register (DMA_UR4T_CFG)

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

UR4TIE: UR4T_DMA interrupt enable control bit

0: Disable UR4T_DMA interrupt

1: Enable UR4T_DMA interrupt

UR4TIP[1:0]: UR4T_DMA interrupt priority control bits

UR4TIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR4TPTY[1:0]: UR4T_DMA Data bus access priority control bits

UR4TPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.8.2 UR4T_DMA Control Register (DMA_UR4T_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CR	FA51H	ENUR4T	TRIG	-	-	-	-	-	-

ENUR4T: UR4T_DMA function enable control bit

0: Disable UR4T_DMA function

1: Enable UR4T_DMA function

TRIG: UR4T_DMA UART1 transmit trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR4T_DMA automatically sending data.

27.8.3 UR4T_DMA Status Register (DMA_UR4T_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_STA	FA52H	-	-	-	-	-	TXOVW	-	UR4TIF

UR4TIF: UR4T_DMA interrupt request flag bit. When the UR4T_DMA data transmission is completed, the hardware automatically sets UR4TIF to 1, and if the UR4T_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

TXOVW: UR4T_DMA data coverage flag. When UR4T_DMA is in the process of data transmission, and the UART writes the SBUF register to trigger the UART to send data again, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

27.8.4 UR4T_DMA transfer total byte register (DMA_UR4T_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_AMT	FA53H								

DMA_UR4T_AMT: Set the number of bytes of data that needs to be automatically sent.

Note: The actual number of bytes is (DMA_UR4T_AMT+1), that is, when DMA_UR4T_AMT is set to 0, 1

byte is transferred, and when DMA_UR4T_AMT is set to 255, 256 bytes are transferred.

27.8.5 UR4T_DMA transfer complete byte register (DMA_UR4T_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_DONE	FA54H								

DMA_UR4T_DONE: The number of bytes that have been sent so far.

27.8.6 UR4T_DMA Send Address Registers (DMA_UR4T_TXAx)

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

DMA_UR4T_TXA: Set the source address of automatic data transmission. Data is read from this address when performing a UR4T_DMA operation.

27.8.7 UR4R_DMA Configuration Register (DMA_UR4R_CFG)

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

UR4RIE: UR4R_DMA interrupt enable control bit

0: Disable UR4R_DMA interrupt

1: Enable UR4R_DMA interrupt

UR4RIP[1:0]: UR4R_DMA interrupt priority control bits

UR4RIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR4RPTY[1:0]: UR4R_DMA Data bus access priority control bits

UR4RPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.8.8 UR4R_DMA Control Register (DMA_UR4R_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CR	FA59H	ENUR4R	-	TRIG	-	-	-	-	CLRIFO

ENUR4R: UR4R_DMA function enable control bit

0: Disable UR4R_DMA function

1: Enable UR4R_DMA function

TRIG: UR4R_DMA UART1 receive trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR4R_DMA receiving data automatically

CLRIFO: Clear UR4R_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the UR4R_DMA operation, clear the built-in FIFO of the UR4R_DMA firstly

27.8.9 UR4R_DMA Status Register (DMA_UR4R_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_STA	FA5AH	-	-	-	-	-	-	RXLOSS	UR4RIF

UR4RIF: UR4R_DMA interrupt request flag bit. When UR4R_DMA receives data, the hardware will automatically set UR4RIF to 1. If the UR4R_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR4R_DMA receive data discard flag. During the UR4R_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the UR4R_DMA and the data received by the UR4R_DMA is automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

27.8.10 UR4R_DMA transfer total byte register (DMA_UR4R_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_AMT	FA5BH								

DMA_UR4R_AMT: Set the number of data bytes that need to automatically receive.

Note: The actual number of bytes is (DMA_UR4R_AMT+1), that is, when DMA_UR4R_AMT is set to 0, 1 byte is transferred, and when DMA_UR4R_AMT is set to 255, 256 bytes are transferred.

27.8.11 UR4R_DMA transfer complete byte register (DMA_UR4R_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_DONE	FA5CH								

DMA_UR4R_DONE: The number of bytes that have been received currently.

27.8.12 UR4R_DMA Receive Address Registers (DMA_UR4T_RXAx)

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

DMA_UR4R_RXA: Set the target address for automatically receiving data. Data will be written from this address when performing a UR4R_DMA operation.

27.9 Data exchange between LCM and memory (LCM_DMA)

27.9.1 LCM_DMA Configuration Register (DMA_LCM_CFG)

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

LCMIE: LCM_DMA interrupt enable control bit

0: Disable LCM_DMA interrupt

1: Enable LCM_DMA interrupt

LCMIP [1:0]: LCM_DMA interrupt priority control bits

LCMIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

LCMPTY[1:0]: LCM_DMA Data bus access priority control bits

LCMPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

27.9.2 LCM_DMA Control Register (DMA_LCM_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CR	FA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	CLRIFO

ENLCM: LCM_DMA function enable control bit

0: Disable LCM_DMA function

1: Enable LCM_DMA function

TRIGWC: LCM_DMA send command mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start LCM_DMA send command mode operation

TRIGWD: LCM_DMA send data mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start LCM_DMA send data mode operation

TRIGRC: LCM_DMA read command mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start LCM_DMA read command mode operation

TRIGRD: LCM_DMA read data mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start LCM_DMA read data mode operation

27.9.3 LCM_DMA Status Register (DMA_LCM_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_STA	FA72H	-	-	-	-	-	-	TXOVW	LCMIF

LCMIF: LCM_DMA interrupt request flag bit. After the LCM_DMA data exchange is completed, the hardware automatically sets LCMIF to 1. If the LCM_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software

TXOVW: LCM_DMA data coverage flag. When LCM_DMA is in the process of data transmission, and LCMIF writes the LCMIFDATL and LCMIDDATH registers, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software

27.9.4 LCM_DMA transfer total byte register (DMA_LCM_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_AMT	FA73H								

DMA_LCM_AMT: Set the number of bytes that need to be read and written.

Note: The actual number of bytes read and written is (DMA_LCM_AMT+1), that is, when DMA_LCM_AMT is set to 0, 1 byte is transferred, and when DMA_LCM_AMT is set to 255, 256 bytes are transferred.

27.9.5 LCM_DMA transfer complete byte register (DMA_LCM_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_DONE	FA74H								

DMA_LCM_DONE: The number of bytes that have been transferred so far.

27.9.6 LCM_DMA Send Address Registers (DMA_LCM_TXAx)

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

DMA_LCM_TXA: Set the source address of automatic data transmission. Data is read from this address when performing an LCM_DMA operation.

27.9.7 LCM_DMA Receive Address Registers (DMA_LCM_RXAx)

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

DMA_LCM_RXA: Set the target address for data transfer. Data is written from this address when performing an LCM_DMA operation.

27.10 Example Routines

27.10.1 UART1 interrupt mode and computer transceiver test - DMA receive timeout interrupt

C language code

//Operating frequency for test is 11.0592MHz

****** Function Description ******

UART1 works in full-duplex interrupt mode to send and receive data. PC sends data to the MCU, and the MCU will automatically store the received data in the DMA space. When the content received at one time is full of the set DMA space, the data in the storage space are output through the DMA automatic sending function of UART 1. Use UART receive interrupt to judge timeout, if no new data is received and timeout, it means that a string of data has been received, then outputs the received content, and clear the DMA space. If a timer is used as baud rate generator, it is recommended to use 1T mode (unless 12T is used for low baud rates), and select a clock frequency that is divisible by the baud rate to improve accuracy.

When downloading, select the clock 22.1184MHz (users can modify the frequency by themselves).

******/*

#include "stdio.h"

#include "stc8h.h"

#define MAIN_Fosc 22118400L // Define the main clock (accurately calculate 115200 baud rate)

#define Baudrate1 115200L

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

#define DMA_AMT_LEN 255 //Set total bytes to be transferred(0~255) : DMA_AMT_LEN+1

bit B_1ms; //1ms flag

bit DMATxFlag;

bit DMARxFlag;

bit BusyFlag;

u8 Rx_cnt;

u8 RX1_TimeOut;

u8 xdata DMABuffer[256];

void UART1_config(u8 brt);

void DMA_Config(void);

void UartPutc(unsigned char dat)

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

char putchar(char c)

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

```

void main(void)
{
    u16 i;

    P0M1 = 0x00; P0M0 = 0x00;           //set as quasi-bidirectional port
    P1M1 = 0x00; P1M0 = 0x00;           //set as quasi-bidirectional port
    P2M1 = 0x00; P2M0 = 0x00;           //set as quasi-bidirectional port
    P3M1 = 0x00; P3M0 = 0x00;           //set as quasi-bidirectional port
    P4M1 = 0x00; P4M0 = 0x00;           //set as quasi-bidirectional port
    P5M1 = 0x00; P5M0 = 0x00;           //set as quasi-bidirectional port
    P6M1 = 0x00; P6M0 = 0x00;           //set as quasi-bidirectional port
    P7M1 = 0x00; P7M0 = 0x00;           //set as quasi-bidirectional port

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

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

    UART1_config(1);                     //Use Timer1 as baud rate generator.
    DMA_Config();
    EA = 1;                             //enable CPU interrupt

    printf("UART1 DMA Timeout Programme!\r\n"); //UART1 sends a string
    DMATxFlag = 0;
    DMARxFlag = 0;

    while (1)
    {
        if((DMATxFlag) && (DMARxFlag)) //Judging the send completion flag and the reception completion
        {
            Rx_cnt = 0;
            RX1_TimeOut = 0;
            printf("\r\nUART1 DMA FULL!\r\n"); //UART1 sends a string
            DMATxFlag = 0;
            DMA_UR1T_CR = 0xc0;              //bit7 1:Enable UART1_DMA,
            //bit6 1:Start UART1_DMA automatic transmission

            DMARxFlag = 0;
            DMA_UR1R_CR = 0xa1;              //bit7 1:Enable UART1_DMA,
            //bit5 1:Start UART1_DMA automatic reception,
            //bit0 1:clear FIFO

        }

        if(B_1ms)                          //reach 1ms
        {
            B_1ms = 0;
            if(RX1_TimeOut > 0)              //timeout count
            {
                if(--RX1_TimeOut == 0)
                {
                    DMA_UR1R_CR = 0x00;      //Disable UART1_DMA
                }
            }
        }
    }
}

```

```

        printf("\r\nUART1 Timeout!\r\n");    //UART1 sends a string

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

        Rx_cnt = 0;
        DMA_UR1R_CR = 0xa1;                //bit7 1:Enable UART1_DMA,
                                           //bit5 1:Start UART1_DMA automatic reception,
                                           //bit0 1:clear FIFO
    }
    }
}

void DMA_Config(void)
{
    P_SW2 = 0x80;
    DMA_UR1T_CFG = 0x80;                  //bit7 1:Enable Interrupt
    DMA_UR1T_STA = 0x00;
    DMA_UR1T_AMT = DMA_AMT_LEN;           //Set total bytes to be transferred: n+1
    DMA_UR1T_TXA = DMABuffer;
    DMA_UR1T_CR = 0xc0;                  //bit7 1:Enable UART1_DMA,
                                           //bit6 1:Start UART1_DMA automatic transmission

    DMA_UR1R_CFG = 0x80;                  //bit7 1:Enable Interrupt
    DMA_UR1R_STA = 0x00;
    DMA_UR1R_AMT = DMA_AMT_LEN;           //Set total bytes to be transferred: n+1
    DMA_UR1R_RXA = DMABuffer;
    DMA_UR1R_CR = 0xa1;                  //bit7 1:Enable UART1_DMA,
                                           //bit5 1:Start UART1_DMA automatic reception,
                                           //bit0 1:clear FIFO
}

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

void UART1_config(u8 brt)
                                           //select baud rate:
                                           //2: Use Timer2 as baud rate generator,
                                           //Other values: Use Timer1 as baud rate generator.
{
    /***** Use Timer2 as baud rate generator *****/
    if(brt == 2)
    {
        AUXR |= 0x01;                  //S1 BRT Use Timer2;
        SetTimer2Baudrate(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /***** Use Timer1 as baud rate generator *****/
    else

```

```

{
    TR1 = 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)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
    TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
    ET1 = 0;                 //Disable interrupts
    INTCLKO &= ~0x02;        //Does not output clock
    TR1 = 1;
}
/*****/

SCON = (SCON & 0x3f) | 0x40; //UART1 mode:
                                //0x00: Synchronous shift output,
                                //0x40: 8-bit data, variable baud rate,
                                //0x80: 9-bit data, fixed baud rate,
                                //0xc0: 9-bit data, variable baud rate

// PS = 1;                    //high priority interrupt
ES = 1;                       //enable interrupt
REN = 1;                      //enable to receive
P_SW1 &= 0x3f;
P_SW1 |= 0x00;               //UART1 switch to:
                                //0x00: P3.0 P3.1,
                                //0x40: P3.6 P3.7,
                                //0x80: P1.6 P1.7,
                                //0xc0: P4.3 P4.4

RX1_TimeOut = 0;
}

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        Rx_cnt++;
        if(Rx_cnt >= DMA_AMT_LEN) Rx_cnt = 0;
        RX1_TimeOut = 5; //If no new data is received within 5ms, it is determined that a
// string of data has been received.
    }

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

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

void UART1_DMA_Interrupt(void) interrupt 13
{
    if (DMA_UR1T_STA & 0x01) //send completed

```



```

{
    DMA_UR1T_STA &= ~0x01;
    DMATxFlag = 1;
}
if (DMA_UR1T_STA & 0x04)                //data coverage
{
    DMA_UR1T_STA &= ~0x04;
}

if (DMA_UR1R_STA & 0x01)                //Receive complete
{
    DMA_UR1R_STA &= ~0x01;
    DMARxFlag = 1;
}
if (DMA_UR1R_STA & 0x02)                //data is discarded
{
    DMA_UR1R_STA &= ~0x02;
}
}

```

//File: ISR.ASM

//Interrupts with interrupt numbers greater than 31 require interrupt entry address remapping processing

```

CSEG AT 012BH                ;P0INT_VECTOR
JMP P0INT_ISR
CSEG AT 0133H                ;P1INT_VECTOR
JMP P1INT_ISR
CSEG AT 013BH                ;P2INT_VECTOR
JMP P2INT_ISR
CSEG AT 0143H                ;P3INT_VECTOR
JMP P3INT_ISR
CSEG AT 014BH                ;P4INT_VECTOR
JMP P4INT_ISR
CSEG AT 0153H                ;P5INT_VECTOR
JMP P5INT_ISR
CSEG AT 015BH                ;P6INT_VECTOR
JMP P6INT_ISR
CSEG AT 0163H                ;P7INT_VECTOR
JMP P7INT_ISR
CSEG AT 016BH                ;P8INT_VECTOR
JMP P8INT_ISR
CSEG AT 0173H                ;P9INT_VECTOR
JMP P9INT_ISR
CSEG AT 017BH                ;M2MDMA_VECTOR
JMP M2MDMA_ISR
CSEG AT 0183H                ;ADCDMA_VECTOR
JMP ADCDMA_ISR
CSEG AT 018BH                ;SPIDMA_VECTOR
JMP SPIDMA_ISR
CSEG AT 0193H                ;U1TXDMA_VECTOR
JMP U1TXDMA_ISR
CSEG AT 019BH                ;U1RXDMA_VECTOR
JMP U1RXDMA_ISR
CSEG AT 01A3H                ;U2TXDMA_VECTOR
JMP U2TXDMA_ISR
CSEG AT 01ABH                ;U2RXDMA_VECTOR
JMP U2RXDMA_ISR
CSEG AT 01B3H                ;U3TXDMA_VECTOR

```

```

        JMP      U3TXDMA_ISR
        CSEG    AT    01BBH                      ;U3RXDMA_VECTOR
        JMP      U3RXDMA_ISR
        CSEG    AT    01C3H                      ;U4TXDMA_VECTOR
        JMP      U4TXDMA_ISR
        CSEG    AT    01CBH                      ;U4RXDMA_VECTOR
        JMP      U4RXDMA_ISR
        CSEG    AT    01D3H                      ;LCMDMA_VECTOR
        JMP      LCMDMA_ISR
        CSEG    AT    01DBH                      ;LCMIF_VECTOR
        JMP      LCMIF_ISR

```

```

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

```

```

        JMP      006BH

```

```

        END

```

27.10.2 UART1 interrupt mode and computer transceiver test - DMA data check

C language code

// Operating frequency for test is 11.0592MHz

/****** Function Description *****

UART1 works in full-duplex interrupt mode to send and receive data. PC sends data to the MCU, and the MCU will automatically store the received data in the DMA space. The last two bytes of the data packet are used as the check digit, and the routine performs the check using the crc16_ccitt algorithm. When the DMA space is full of the content of the set size, the valid data is checked and calculated, and then compared with the last two check digits. The data in the storage space is output through the DMA automatic sending function of UART1. If a timer is used as baud rate generator, it is recommended to use 1T mode (unless 12T is used for low baud rates), and select a clock frequency

that is divisible by the baud rate to improve accuracy.

When downloading, select the clock 22.1184MHz (users can modify the frequency by themselves).

*****/

```
#include "stdio.h"
#include "stc8h.h"
#include "crc16.h"

#define MAIN_Fosc      22118400L           // Define the main clock (accurately calculate 115200 baud rate)
#define Baudrate1      115200L

#define DMA_AMT_LEN    255                ///Set total bytes to be transferred(0~255) : DMA_AMT_LEN+1

bit    DMATxFlag;
bit    DMARxFlag;

u8      xdata DMABuffer[256];

void UART1_config(u8 brt);
void DMA_Config(void);

void UartPutc(unsigned char dat)
{
    SBUF = dat;
    while(TI == 0);
    TI = 0;
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

/*****CRC calculation function*****/
u16 crc16_ccitt(u8 *pbuf, u16 len)
{
    unsigned short code crc16_ccitt_table[256] =
    {
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
        0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
        0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
        0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
        0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
        0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
        0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
        0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
        0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
        0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
        0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
        0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
        0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
        0xFF9F, 0xEFBE, 0xDFDD, 0-CFFC, 0-BF1B, 0-AF3A, 0x9F59, 0x8F78,
        0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
        0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
        0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
```

```

    0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
    0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
    0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
    0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
    0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
    0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
    0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
    0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
    0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};

u16 crc16 = 0x0000;
u16 crc_h8, crc_l8;

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

return crc16;
}

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

    P0M1 = 0x00; P0M0 = 0x00; //set as quasi-bidirectional port
    P1M1 = 0x00; P1M0 = 0x00; //set as quasi-bidirectional port
    P2M1 = 0x00; P2M0 = 0x00; //set as quasi-bidirectional port
    P3M1 = 0x00; P3M0 = 0x00; //set as quasi-bidirectional port
    P4M1 = 0x00; P4M0 = 0x00; //set as quasi-bidirectional port
    P5M1 = 0x00; P5M0 = 0x00; //set as quasi-bidirectional port
    P6M1 = 0x00; P6M0 = 0x00; //set as quasi-bidirectional port
    P7M1 = 0x00; P7M0 = 0x00; //set as quasi-bidirectional port

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

    P_SW2 = 0x80;
    DMA_UR1T_STA = 0x00;
    UART1_config(1);
    printf("UART1 DMA CRC Programme!\r\n");

    DMA_Config();
    EA = 1; //enable CPU interrupt

    DMATxFlag = 0;
    DMARxFlag = 0;

```

```

while (1)
{
    if((DMATxFlag) && (DMARxFlag))
    {
        CheckSum = crc16_ccitt(DMABuffer,DMA_AMT_LEN-1);
        if(((u8)CheckSum == DMABuffer[DMA_AMT_LEN-1]) &&
            ((u8)(CheckSum>>8) == DMABuffer[DMA_AMT_LEN]))
        {
            printf("\r\nOK! CheckSum = %04x\r\n",CheckSum);
        }
        else
        {
            printf("\r\nERROR! CheckSum = %04x\r\n",CheckSum);
        }
        DMATxFlag = 0;
        DMA_UR1T_CR = 0xc0;                //bit7 1:Enable UART1_DMA,
                                           //bit6 1:Start UART1_DMA automatic transmission

        DMARxFlag = 0;
        DMA_UR1R_CR = 0xa1;                //bit7 1:Enable UART1_DMA,
                                           //bit5 1:Start UART1_DMA automatic reception,
                                           //bit0 1:clear FIFO
    }
}

void DMA_Config(void)
{
    P_SW2 = 0x80;
    DMA_UR1T_CFG = 0x80;                  //bit7 1:Enable Interrupt
    DMA_UR1T_STA = 0x00;
    DMA_UR1T_AMT = DMA_AMT_LEN;           // Set total bytes to be transferred: n+1
    DMA_UR1T_TXA = DMABuffer;
    DMA_UR1T_CR = 0xc0;                  //bit7 1:Enable UART1_DMA,
                                           //bit6 1:Start UART1_DMA automatic transmission

    DMA_UR1R_CFG = 0x80;                  //bit7 1:Enable Interrupt
    DMA_UR1R_STA = 0x00;
    DMA_UR1R_AMT = DMA_AMT_LEN;           // Set total bytes to be transferred: n+1
    DMA_UR1R_RXA = DMABuffer;
    DMA_UR1R_CR = 0xa1;                  //bit7 1:Enable UART1_DMA,
                                           //bit5 1:Start UART1_DMA automatic reception, bit0 1:clear
}

FIFO
}

void SetTimer2Baudrate(u16 dat)
//select baud rate:
//2: Use Timer2 as baud rate generator,
//Other values: Use Timer1 as baud rate generator.

{
    AUXR &= ~(1<<4);                    //Timer stop
    AUXR &= ~(1<<3);                    //Timer2 set As Timer
    AUXR |= (1<<2);                      //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2 &= ~(1<<2);                    //Disable interrupts
    AUXR |= (1<<4);                    //Timer run enable
}

void UART1_config(u8 brt)
//select baud rate:

```

```

//2: Use Timer2 as baud rate generator
//Other values: Use Timer1 as baud rate generator.

{
    /***** Use Timer2 as baud rate generator *****/
    if(brt == 2)
    {
        AUXR |= 0x01; //S1 BRT Use Timer2;
        SetTimer2Baudrate(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /***** Use Timer1 as baud rate generator *****/
    else
    {
        TR1 = 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)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
        TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
        ET1 = 0; //Disable interrupts
        INTCLKO &= ~0x02; //Does not output clock
        TR1 = 1;
    }
    /*****/

    SCON = (SCON & 0x3f) | 0x40; //UART1 mode,
    //0x00: Synchronous shift output,
    //0x40: 8-bit data, variable baud rate,
    //0x80: 9-bit data, fixed baud rate,
    //0xc0: 9-bit data, variable baud rate

    // PS = 1; //high priority interrupt
    // ES = 1; //enable interrupt
    REN = 1; //enable to receive
    P_SW1 &= 0x3f;
    P_SW1 |= 0x00;
}

void UART1_DMA_Interruption(void) interrupt 13
{
    if (DMA_UR1T_STA & 0x01) //send completed
    {
        DMA_UR1T_STA &= ~0x01;
        DMATxFlag = 1;
    }
    if (DMA_UR1T_STA & 0x04) //data coverage
    {
        DMA_UR1T_STA &= ~0x04;
    }

    if (DMA_UR1R_STA & 0x01) //Receive complete
    {
        DMA_UR1R_STA &= ~0x01;
        DMARxFlag = 1;
    }
    if (DMA_UR1R_STA & 0x02) //data is discarded
    {
        DMA_UR1R_STA &= ~0x02;
    }
}

```

```

    }
}

```

//File: ISR.ASM

//Interrupts with interrupt numbers greater than 31 require interrupt entry address remapping processing

```

    CSEG    AT    012BH                      ;P0INT_VECTOR
    JMP     P0INT_ISR
    CSEG    AT    0133H                      ;P1INT_VECTOR
    JMP     P1INT_ISR
    CSEG    AT    013BH                      ;P2INT_VECTOR
    JMP     P2INT_ISR
    CSEG    AT    0143H                      ;P3INT_VECTOR
    JMP     P3INT_ISR
    CSEG    AT    014BH                      ;P4INT_VECTOR
    JMP     P4INT_ISR
    CSEG    AT    0153H                      ;P5INT_VECTOR
    JMP     P5INT_ISR
    CSEG    AT    015BH                      ;P6INT_VECTOR
    JMP     P6INT_ISR
    CSEG    AT    0163H                      ;P7INT_VECTOR
    JMP     P7INT_ISR
    CSEG    AT    016BH                      ;P8INT_VECTOR
    JMP     P8INT_ISR
    CSEG    AT    0173H                      ;P9INT_VECTOR
    JMP     P9INT_ISR
    CSEG    AT    017BH                      ;M2MDMA_VECTOR
    JMP     M2MDMA_ISR
    CSEG    AT    0183H                      ;ADCDMA_VECTOR
    JMP     ADCDMA_ISR
    CSEG    AT    018BH                      ;SPIDMA_VECTOR
    JMP     SPIDMA_ISR
    CSEG    AT    0193H                      ;U1TXDMA_VECTOR
    JMP     U1TXDMA_ISR
    CSEG    AT    019BH                      ;U1RXDMA_VECTOR
    JMP     U1RXDMA_ISR
    CSEG    AT    01A3H                      ;U2TXDMA_VECTOR
    JMP     U2TXDMA_ISR
    CSEG    AT    01ABH                      ;U2RXDMA_VECTOR
    JMP     U2RXDMA_ISR
    CSEG    AT    01B3H                      ;U3TXDMA_VECTOR
    JMP     U3TXDMA_ISR
    CSEG    AT    01BBH                      ;U3RXDMA_VECTOR
    JMP     U3RXDMA_ISR
    CSEG    AT    01C3H                      ;U4TXDMA_VECTOR
    JMP     U4TXDMA_ISR
    CSEG    AT    01CBH                      ;U4RXDMA_VECTOR
    JMP     U4RXDMA_ISR
    CSEG    AT    01D3H                      ;LCMDMA_VECTOR
    JMP     LCMDMA_ISR
    CSEG    AT    01DBH                      ;LCMIF_VECTOR
    JMP     LCMIF_ISR

```

P0INT_ISR:

P1INT_ISR:

P2INT_ISR:

P3INT_ISR:

P4INT_ISR:

P5INT_ISR:
 P6INT_ISR:
 P7INT_ISR:
 P8INT_ISR:
 P9INT_ISR:
 M2MDMA_ISR:
 ADCDMA_ISR:
 SPIDMA_ISR:
 U1TXDMA_ISR:
 U1RXDMA_ISR:
 U2TXDMA_ISR:
 U2RXDMA_ISR:
 U3TXDMA_ISR:
 U3RXDMA_ISR:
 U4TXDMA_ISR:
 U4RXDMA_ISR:
 LCMDMA_ISR:
 LCMIF_ISR:

JMP 006BH

END

Code testing method

According to the predefined DMA packet length (for example: 256 bytes), send a packet of data (254 bytes) through the serial port tool, and add a 2-byte CCITT-CRC16 check code at the end:



After the MCU receives the entire packet of data (256 bytes), it performs CRC16 check on the first 254 bytes of data, and the obtained check code is compared with the last two bytes. If the values are equal, print "OK!" and calculate the check code, and then output the content read from the DMA space.



If the checksum values are not equal, print "ERROR!" and the calculated checksum.

28 Enhanced Dual Data Pointer

Two 16-bit data pointers are integrated in STC8H series of microcontrollers. The data pointers can be increased or decreased automatically by the program control, and they can be switched automatically.

28.1 Related special function registers

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	Data pointer low byte register	82H									0000,0000
DPH	Data pointer high byte register	83H									0000,0000
DPL1	2nd Data pointer low byte	E4H									0000,0000
DPH1	2nd Data pointer high byte	E5H									0000,0000
DPS	DPTR Selection Register	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR Timing control register	AEH									0000,0000

28.1.1 1st 16-bit Data Pointer Registers (DPTR0)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL is Data pointer 0 low byte.

DPH is Data pointer 0 high byte.

The combination of DPL and DPH is the first 16-bit data pointer register DPTR0.

28.1.2 2nd 16-bit Data Pointer Registers (DPTR1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1 is Data pointer 1 low byte.

DPH1 is Data pointer 1 low byte.

The combination of DPL1 and DPH1 is the second 16-bit data pointer register DPTR1.

28.1.3 DPTR control register

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

ID1: DPTR1 auto-increment or auto-decrement mode control bit

0: DPTR1 auto-increment mode

1: DPTR1 auto-decrement mode

ID0: DPTR0 auto-increment or auto-decrement mode control bit

0: DPTR0 auto-increment mode

1: DPTR0 auto-decrement mode

TSL: DPTR0/DPTR1 auto-switch control bit (invert SEL automatically)

0: DPTR0/DPTR1 auto switch is disabled.

1: DPTR0/DPTR1 auto switch is enabled.

If the TSL bit is set, the SEL bit will be inverted automatically after the relevant instruction is executed.

Instructions related to TSL include:

MOV DPTR,#data16

```

INC      DPTR
MOVC     A,@A+DPTR
MOVX     A,@DPTR
MOVX     @DPTR,A

```

AU1/AU0: Enable DPTR1 / DPTR0 Automatic increment / decrement control bit

0: disable Automatic increment / decrement function

1: enable Automatic increment / decrement function

Note: In write-protect mode, AU0 and AU1 can not be enabled individually. AU0 will be enabled automatically if AU1 is enabled. If AU0 is enabled alone, there is no effect to AU1. To enable AU1 or AU0 independently, the TA register must be used to trigger the DPS protection mechanism. For more detail, please refer to the description of the TA register. In addition, DPTR0 / DPTR1 will be incremented / decremented automatically only after executing the following three instructions.

```

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

```

SEL: DPTR register select bit.

0: Default. DPTR0 is selected as current Data pointer.

1: DPTR1 is selected as current Data pointer.

The selection of current DPTR using SEL is valid for the following instructions,

```

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

```

28.1.4 Data Pointer control register (TA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

AU1 and AU0 in the DPS register is write-protected by TA register. Since the program can not write AU1 and AU0 separately, TA register must be used to trigger enabling AU1 or AU0 independently. TA is a write-only register.

The following steps must be executed if you need to enable AU1 or AU0 separately.

```

CLR      EA          ; disable interrupt (it is necessary to disable interrupt)
MOV      TA,#0AAH    ; write the trigger command sequence 1
                        ; any other instructions can not be here
MOV      TA,#55H     ; write the trigger command sequence 2
                        ; any other instructions can not be here
MOV      DPS,#xxH    ; Write-protection is temporarily disabled,
                        ; and any value can be written to the DPS
                        ; DSP enters the write-protected mode again
SETB     EA          ; enable interrupt if necessary

```

28.2 Example Routines

28.2.1 Example Routine 1

Copy 4 bytes of data stored in program space 1000H to 1003H in reverse to 0100H to 0103H of the extended RAM, that is,

C:1000H → X:0103H

C:1001H → X:0102H

C:1002H → X:0101H

C:1003H → X:0100H

Assembly code

;Operating frequency for test is 11.0592MHz

```

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

                ORG      0000H
                LJMP     MAIN

MAIN:          ORG      0100H

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

                MOV      DPS, #00100000B    ; enable TSL and select DPTR0
                MOV      DPTR, #1000H      ; write 1000H to DPTR0, and then select DPTR1 as current DPTR
                MOV      DPTR, #0103H      ; write 0103H to DPTR1
                MOV      DPS, #10111000B    ; set DPTR1 in auto decreasing mode,
                                           ; DPTR0 in auto increasing mode, enable TSL, AU0 and AU1,
                                           ; select DPTR0 as the current DPTR

                MOV      R7, #4             ; set the counter of copies

COPY_NEXT:    CLR      A                   ;
                MOVC     A, @A+DPTR         ; Read data from the program space indicated by DPTR0,
                                           ; When done, DPTR0 increments automatically and select DPTR1 as the current DPTR

```

```

MOVX    @DPTR,A           ; write the content of ACC to XDARA indicated by DPTR1,
                          ; When done, DPTR1 decrements automatically and select DPTR0 as the current DPTR
DJNZ    R7,COPY_NEXT      ;

SJMP     $

END

```

28.2.2 Example Routine 2

Send the data stored in the extended RAM 0100H to 0103H to P0 port successively.

Assembly code

;Operating frequency for test is 11.0592MHz

```

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

        ORG      0000H
        LJMP     MAIN

MAIN:    ORG      0100H

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

        CLR      EA           ; disable interrupt
        MOV      TA, #0AAH    ; Write DPS write-protection trigger command 1
        MOV      TA, #55H     ; Write DPS write-protection trigger command 2
        MOV      DPS, #00001000B ; set DPTR0 in increasing mode, enable AU0 independently,
                                ; and select DPTR0

        SETB     EA           ; enable interrupt
        MOV      DPTR, #0100H ; write 0100H to DPTR0
        MOVX     A, @DPTR     ; Read data from XRAM indicated by DPTR0,
                                ; and then DPTR0 increments automatically
        MOV      P0, A        ; output the datum to Port0

```

<i>MOVX</i>	<i>A,@DPTR</i>	<i>; Read data from XRAM indicated by DPTR0,</i> <i>;and then DPTR0 increments automatically</i>
<i>MOV</i>	<i>P0,A</i>	<i>; output the datum to Port0</i>
<i>MOVX</i>	<i>A,@DPTR</i>	<i>; Read data from XRAM indicated by DPTR0,</i> <i>;and then DPTR0 increments automatically</i>
<i>MOV</i>	<i>P0,A</i>	<i>; output the datum to Port0</i>
<i>MOVX</i>	<i>A,@DPTR</i>	<i>; Read data from XRAM indicated by DPTR0,</i> <i>;and then DPTR0 increments automatically</i>
<i>MOV</i>	<i>P0,A</i>	<i>; output the datum to Port0</i>
<i>SJMP</i>	<i>\$</i>	
<i>END</i>		

STC MCU

29 MDU16 Hardware 16-bit Multiplier and Divider

Product line	MDU16
STC8H1K08 family	
STC8H1K28 family	
STC8H3K64S4 family	●
STC8H3K64S2 family	●
STC8H8K64U family	●
STC8H2K64T family	●
STC8H4K64TLR family	●
STC8H4K64TLCD family	●
STC8H4K64LCD family	●

A 16-bit hardware multiply / divide unit MDU16 is integrated in some microcontrollers of the STC8H series.

The following data operations are supported:

- Data standardization (need 3-20 clocks of computing time)
- Logic left shift (need 3~18 clocks of operation time)
- Logic shift right (need 3~18 clocks of operation time)
- 16 bits multiplied by 16 bits (it takes 10 clocks of operation time)
- 16 bits divided by 16 bits (need 9 clocks of operation time)
- 32 bits divided by 16 bits (requires 17 clocks of operation time)

All operations are based on unsigned integer data types.

29.1 Registers Related to MDU16

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
MD3	MDU Data Register	FCF0H	MD3[7:0]								0000,0000
MD2	MDU Data Register	FCF1H	MD2[7:0]								0000,0000
MD1	MDU Data Register	FCF2H	MD1[7:0]								0000,0000
MD0	MDU Data Register	FCF3H	MD0[7:0]								0000,0000
MD5	MDU Data Register	FCF4H	MD5[7:0]								0000,0000
MD4	MDU Data Register	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU Mode Control Register	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU Operation Control Register	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000

29.1.1 Operand 1 Data Registers

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

29.1.2 Operand 2 Data Registers

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

32-bit division by 16-bit division:

Dividend: {MD3,MD2,MD1,MD0}

Divisor: {MD5,MD4}

Quotient: {MD3,MD2,MD1,MD0}

Remainder: {MD5,MD4}

16-bit division by 16-bit division:

Dividend: {MD1,MD0}

Divisor: {MD5,MD4}

Quotient: {MD1,MD0}

Remainder: {MD5,MD4}

16-bit multiplication by 16-bit multiplication:

Multiplicand: {MD1,MD0}

Multiplier: {MD5,MD4}

Product: {MD3,MD2,MD1,MD0}

32-bit logical shift left / logical shift right

Operand: {MD3,MD2,MD1,MD0}

32-bit data normalization:

Operand: {MD3,MD2,MD1,MD0}

29.1.3 MDU Mode Control Register (ARCON)

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

MODE[2:0]: MDU mode selection

MODE[2:0]	Mode	Clocks	Description
1	Logical right shift	3~18	Shift the data in {MD3, MD2, MD1, MD0} right by SC [4: 0] bits, MD3 high-order complement 0
2	Logical left shift	3~18	Shift the data in {MD3, MD2, MD1, MD0} SC [4: 0] bits to the left, low-order complement of MD0
3	Data normalization	3~20	Perform logical left shift on the data in {MD3, MD2, MD1, MD0}, shift out all the high-order 0s of the data, make the highest bit of MD3 be 1, and the number of logical left shifts is recorded in SC [4: 0].
4	16-bit multiplication	10	$\{MD1,MD0\} \times \{MD5,MD4\} = \{MD3,MD2,MD1,MD0\}$
5	16-bit division	9	$\{MD1,MD0\} \div \{MD5,MD4\} = \{MD1,MD0\} \cdots \{MD5,MD4\}$
6	32-bit division	17	$\{MD3,MD2,MD1,MD0\} \div \{MD5,MD4\} = \{MD3,MD2,MD1,MD0\} \cdots \{MD5,MD4\}$
Others	Invalid		

SC[4:0]: Data shift bits

When the MDU is in shift mode, SC is used to set the number of bits for left/right shift

When MDU is in data normalization mode, SC is the actual number of bits moved by the data after data normalization

29.1.4 MDU Operation Control Register (OPCON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
OPCON	FCF7H	-	MDOV	-	-	-	-	RST	ENOP

MDOV: MDU Overflow flag (read-only flag)

MDOV is set by hardware automatically in the following situations:

1. When the divisor is 0;
2. When the product of multiplication is greater than 0FFFFH;

When software writes OPCON.0 (EN) or writes ARCON, MDOV is cleared by the hardware automatically.

RST: Software resets the MDU multiplication and division unit. Writing 1 to it will trigger a software reset. It is cleared by the hardware automatically after the MDU reset is complete.

Note: The value of the ARCON register is cleared when software resets the MDU multiply and divide unit.

ENOP: MDU enable bit.

Writing 1 to this bit will trigger the MDU module to start calculation. When the MDU calculation is completed, ENOP is cleared to 0 by the hardware automatically. After setting ENOP to 1, the software can query ENOP cyclically. When ENOP changes from 1 to 0, the calculation is completed.

STC MCU

29.2 Example Routines

C Language code

;Operating frequency for test is 11.0592MHz

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

#define MD3U32      (*(unsigned long volatile xdata *)0x0000)
#define MD3U16      (*(unsigned int volatile xdata *)0x0000)
#define MD1U16      (*(unsigned int volatile xdata *)0x0002)
#define MD5U16      (*(unsigned int volatile xdata *)0x0004)

#define MD3          (*(unsigned char volatile xdata *)0x0000)
#define MD2          (*(unsigned char volatile xdata *)0x0001)
#define MD1          (*(unsigned char volatile xdata *)0x0002)
#define MD0          (*(unsigned char volatile xdata *)0x0003)
#define MD5          (*(unsigned char volatile xdata *)0x0004)
#define MD4          (*(unsigned char volatile xdata *)0x0005)
#define ARCON        (*(unsigned char volatile xdata *)0x0006)
#define OPCON        (*(unsigned char volatile xdata *)0x0007)

sfr      P_SW2      = 0x80;
/////////////////////////////////////////////////
//16 bits by 16 bits
/////////////////////////////////////////////////
unsigned long res;
unsigned int dat1, dat2;
P_SW2 |= 0x80;           // Access the extension register xsfr
MD1U16 = dat1;           //dat1 User given
MD5U16 = dat2;           //dat2 User given
ARCON = 4 << 5;          //16 bits*16 bits, multiplication mode
OPCON = 1;               // Start calculation
while((OPCON & 1) != 0); // Wait for the calculation to complete
res = MD3U32;            //32-bit result
/////////////////////////////////////////////////
//32 bits divided by 16 bits
/////////////////////////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned int dat2;
P_SW2 |= 0x80;           // Access the extension register xsfr
MD3U32 = dat1;           //dat1 User given
MD5U16 = dat2;           //dat2 User given
ARCON = 6 << 5;          //32-bit/16-bit, division mode
OPCON = 1;               // Start calculation
while((OPCON & 1) != 0); // Wait for the calculation to complete
res = MD3U32;            //32-bit quotient, 16-bit remainder in MD5U16

unsigned long res;
unsigned long dat1;
unsigned char num;
MD3U32 = dat1;           // The number of bits to shift, User given
ARCON = (2 << 5) + num;  //dat1 User given
//ARCON = (1 << 5) + num; //32-bit left shift mode
OPCON = 1;               //32-bit right shift mode
while((OPCON & 1) != 0); // Start calculation
res = MD3U32;            // Wait for the calculation to complete
//32-bit result
```

Appendix A STC Emulator User Guide

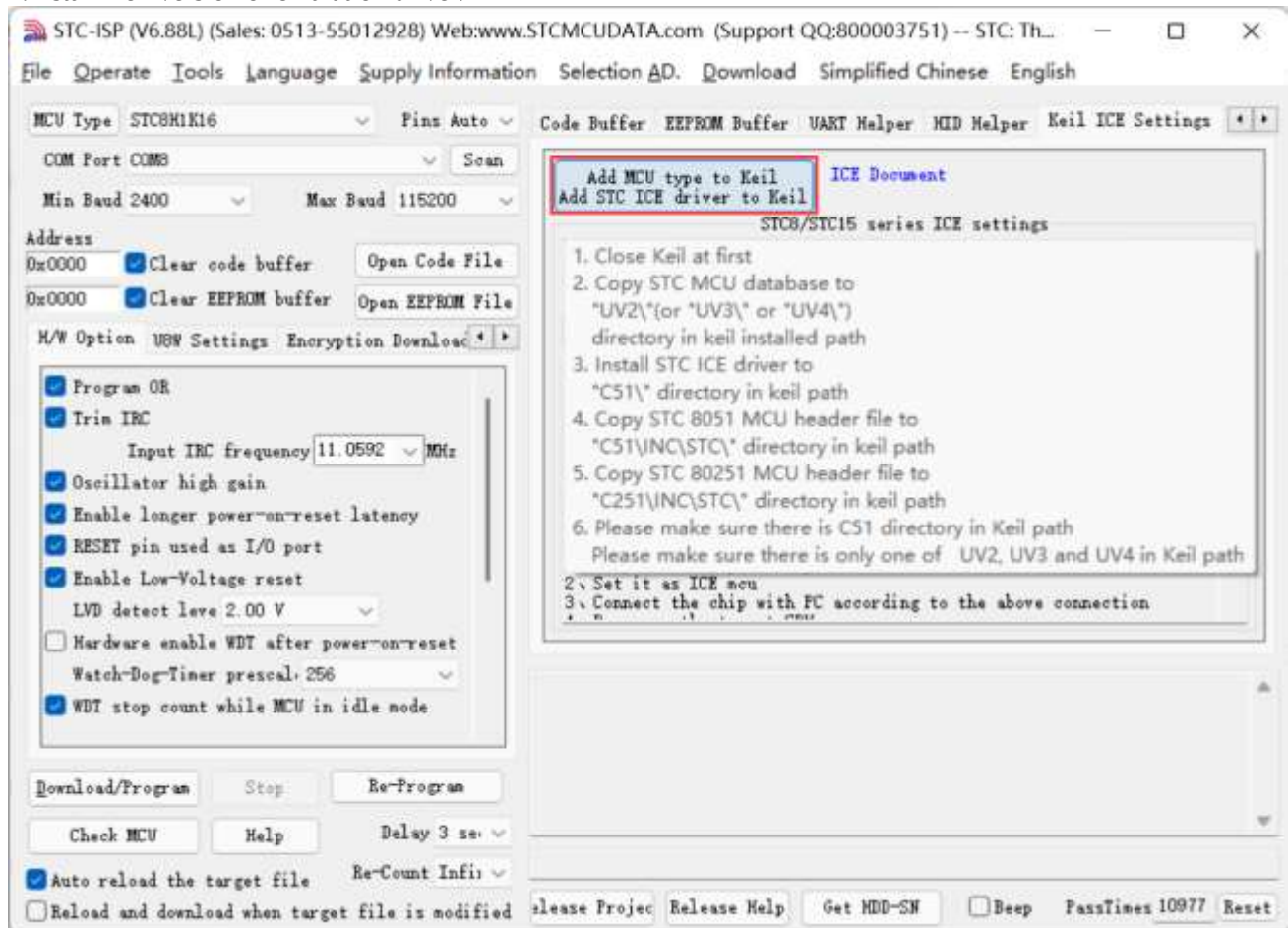
A: What kind of compiler/assembler should be used for STC MCU?

Q: Any old 8051 compiler/assembler can support it. Keil C51 is popular now

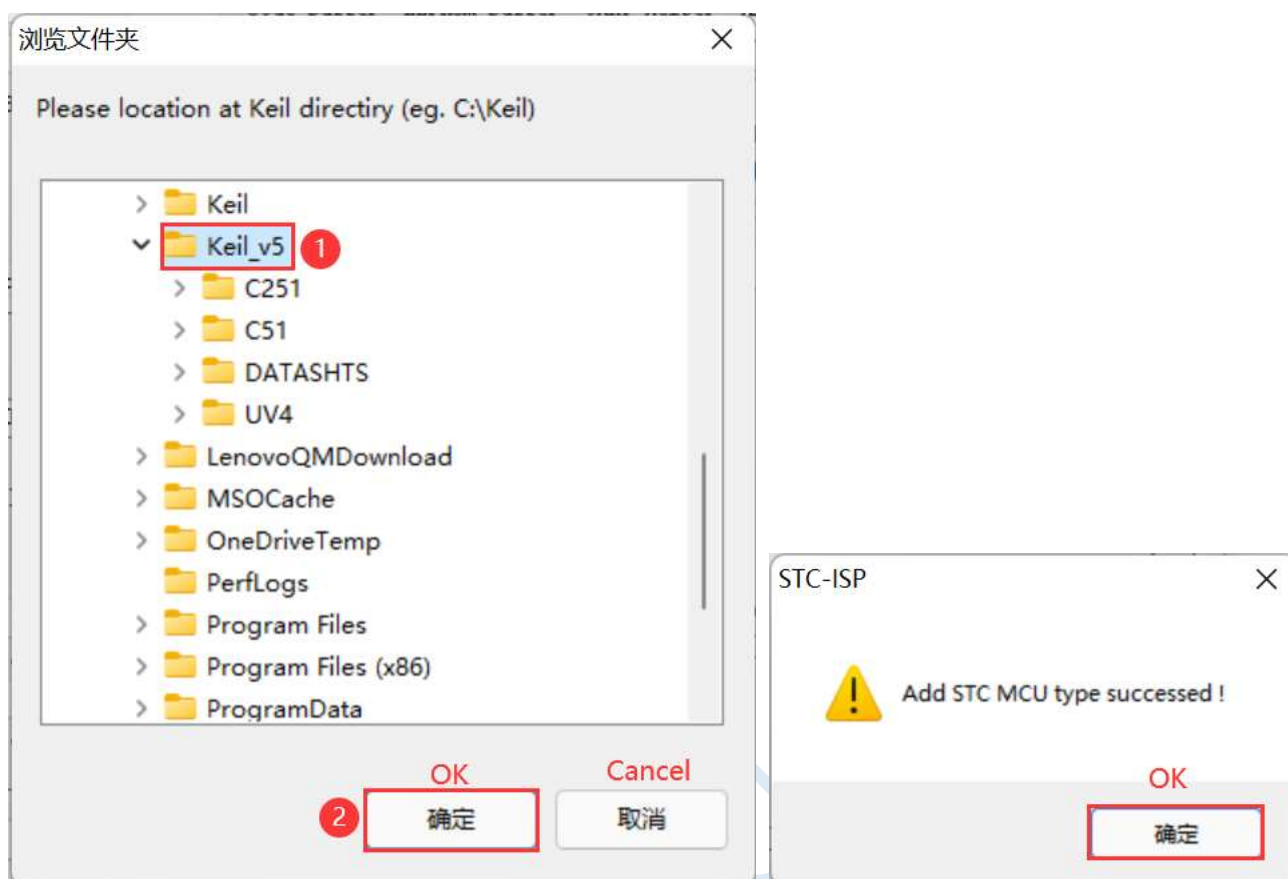
A: How to include header files in Keil environment

Q: After installing the driver and header files according to the steps shown below, select the STC corresponding MCU model when creating a new project, and directly use "#include <stc8h.h>" in the source file to complete the inclusion of the header file. If you select Intel's 8052/87C52/87C54/87C58 or Philips P87C52/P87C54/P87C58 compile, the header file contains <reg51.h>, but the new STC special function register needs to be declared by the user.

1. Install Keil version of emulation driver.

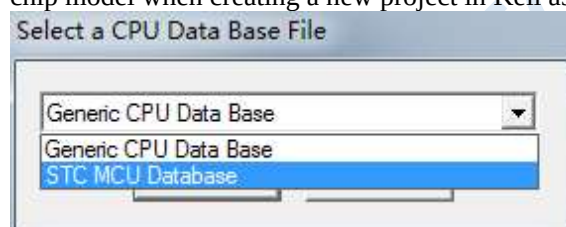


As shown above, select the "Keil Simulation Settings" page firstly, click "Add MCU model to Keil", and in the following directory selection window that appears, navigate to the installation directory of Keil (usually "C:\Keil\"). After press "OK" button, the prompt message shown on the right in the following figure appears, indicating that the installation was successful. The STC Monitor51 emulation driver STCMON51.DLL will also be installed when adding the header file. The installation directory of the driver and header file is shown above.

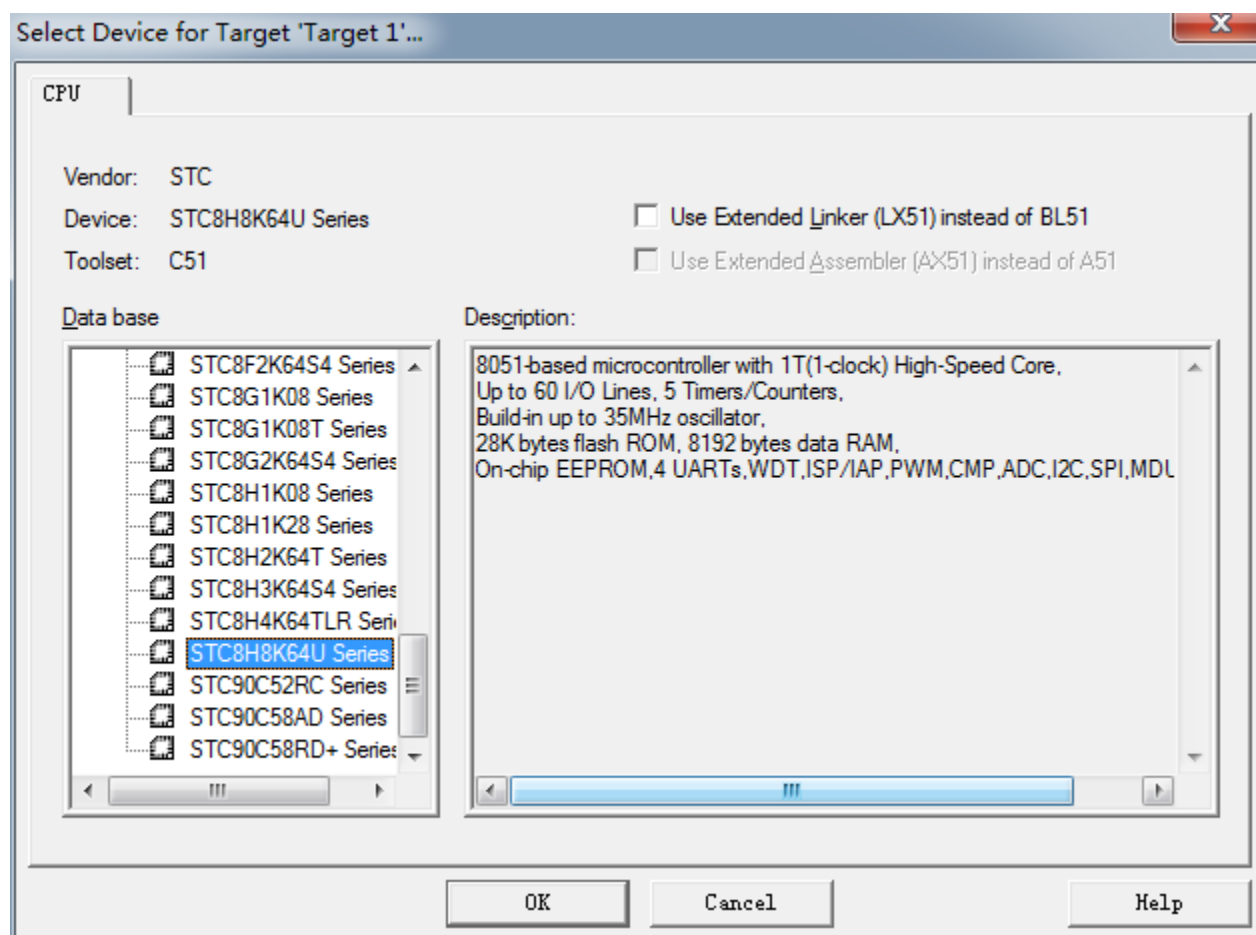


2. Create a project in Keil

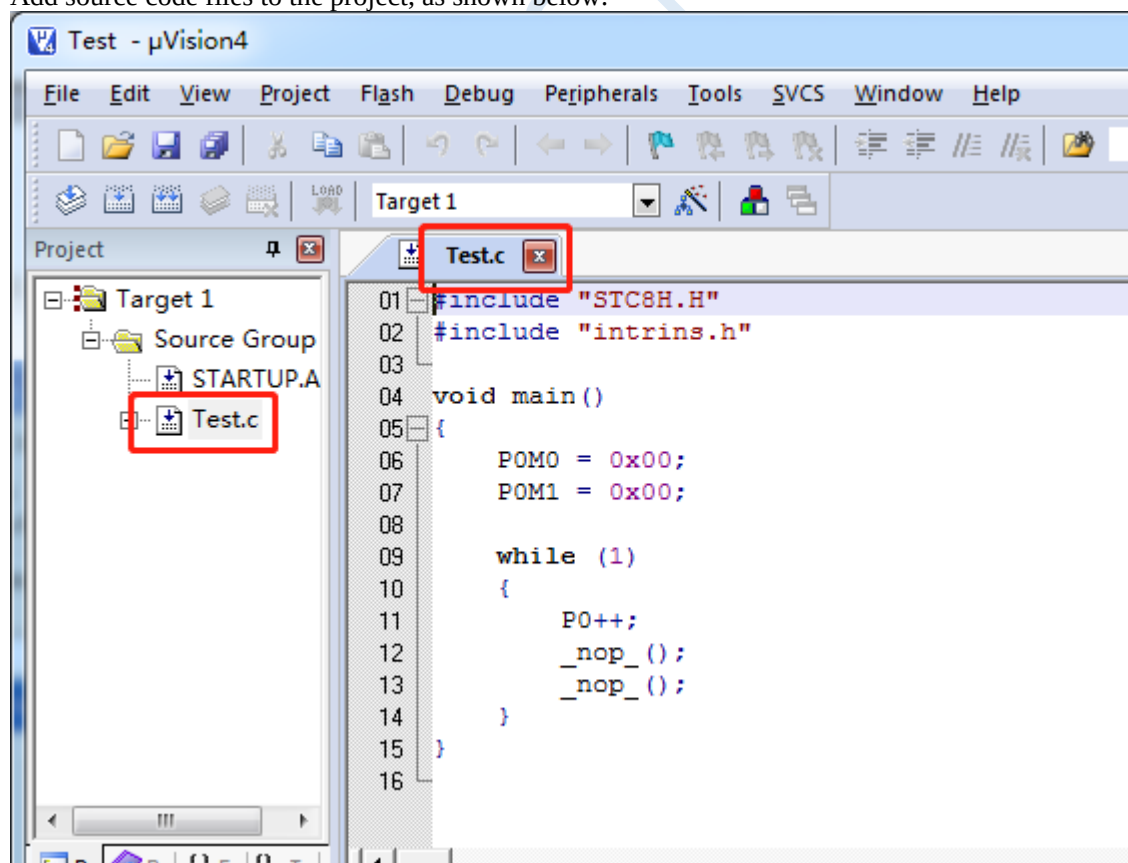
If the driver installation is successful in the first step, there will be an option of "STC MCU Database" in selecting the chip model when creating a new project in Keil as shown below.



Then select the responding MCU model from the list. Here we select the model of "STC8A8K64S4A12" and click "OK" to complete the selection.



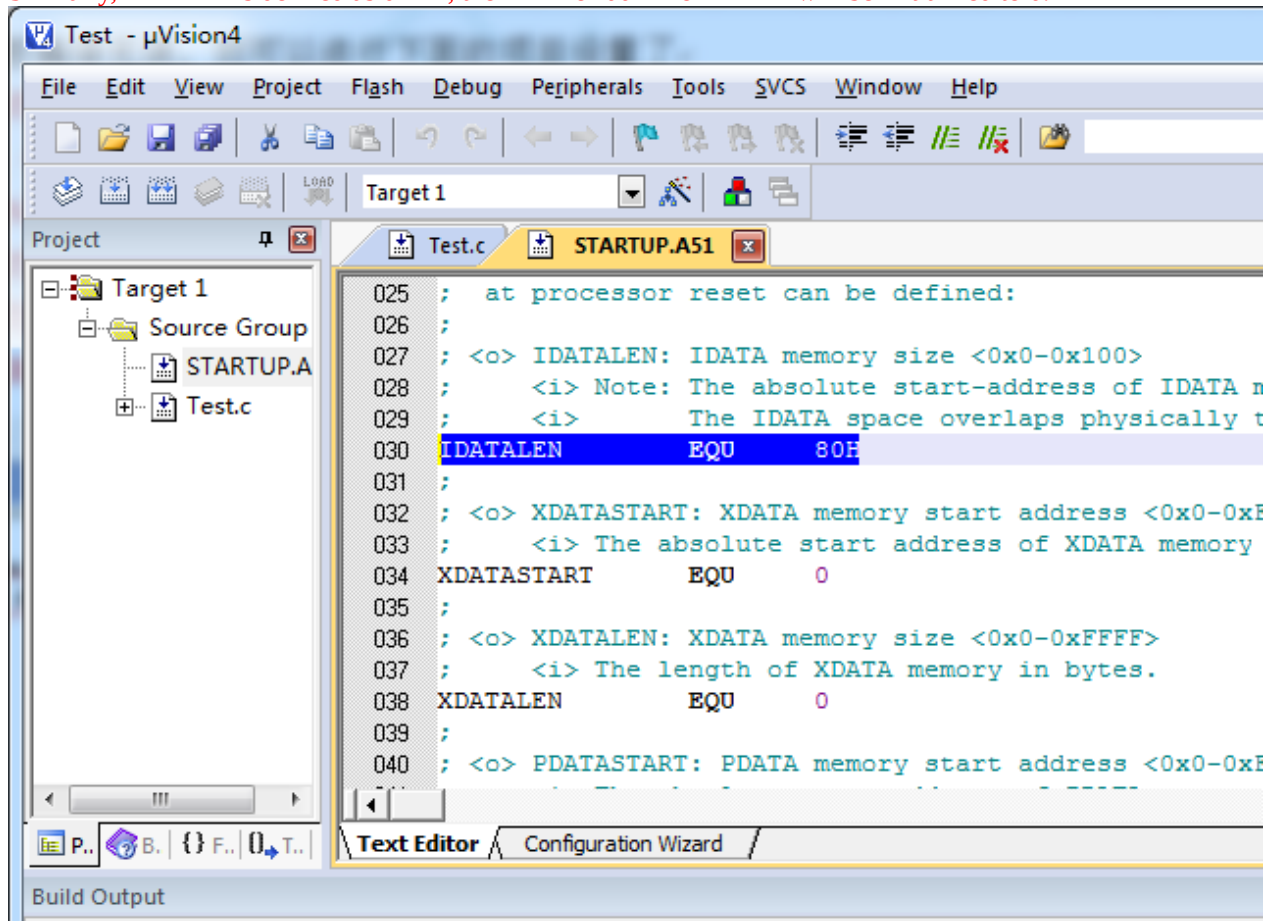
Add source code files to the project, as shown below:



Save the project. If there is no error while compiling the project, you can set the following projects.

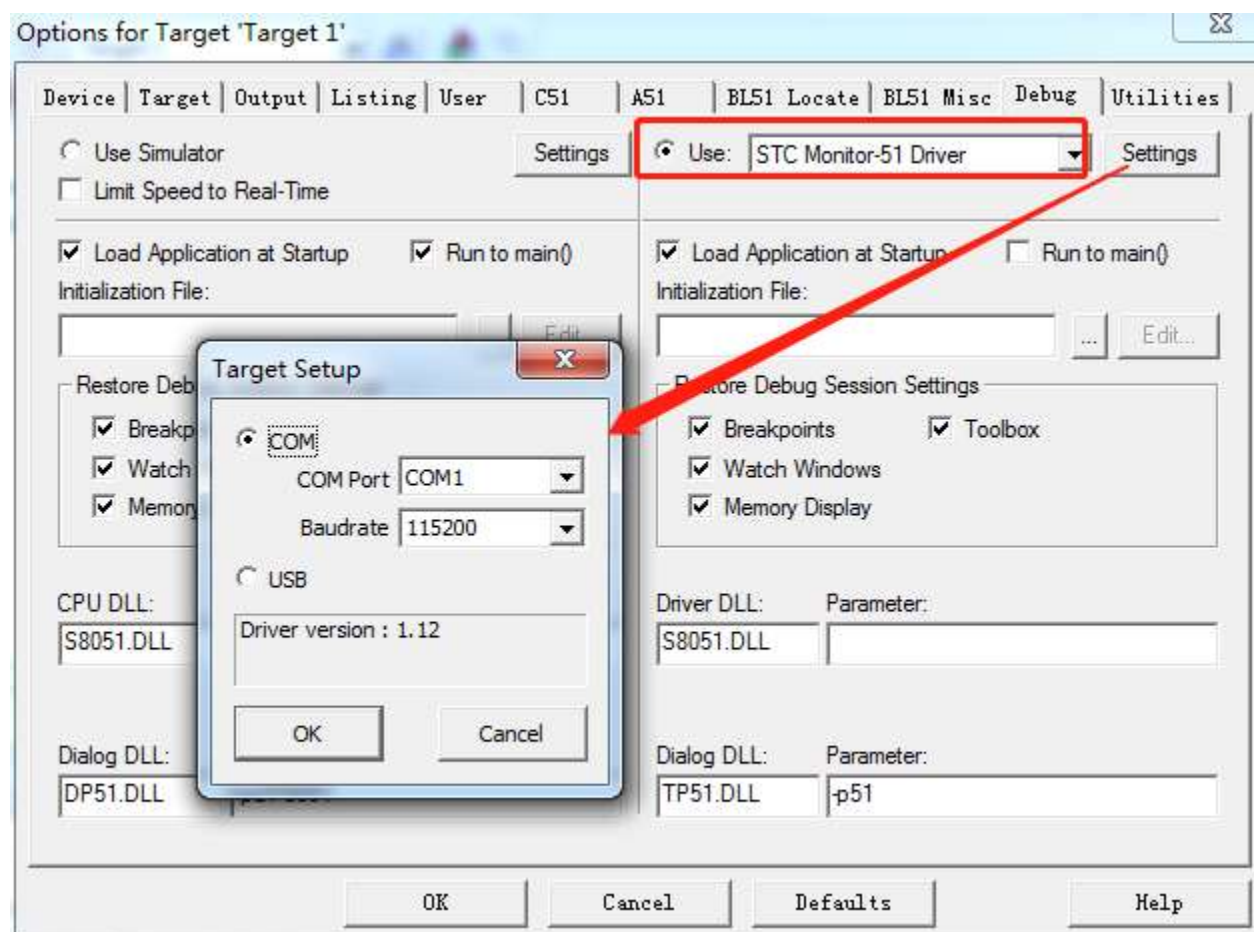
One additional note:

When a C language project is created and a startup file "STARTUP.A51" is added to the project, there is a macro definition named "IDATALEN", which is a macro used to define the size of the IDATA. The default value is 128, which is 80H in hexadecimal, and it is also the size of IDATA in the startup file that needs to be initialized to 0. Therefore if IDATA is defined as 80H, the code in STARTUP.A51 will initialize the RAM of 00-7F of IDATA to 0. Similarly, if IDATA is defined as 0FFH, the RAM of 00-FF of IDATA will be initialized to 0.



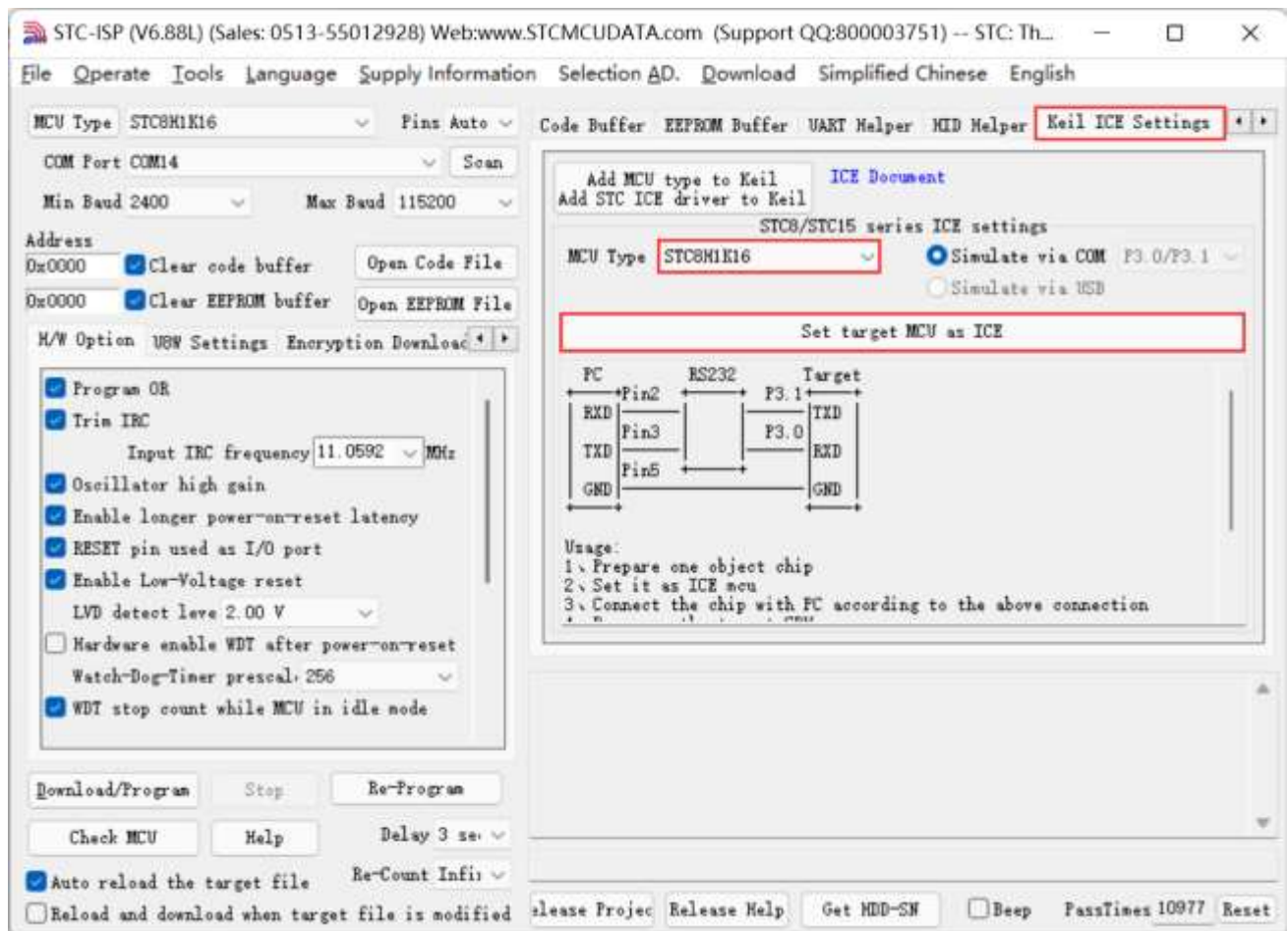
The IDATA size of the STC8 series of microcontrollers is 256 bytes (DATA of 00-7F and IDATA of 80H-FFH). Because the last 17 bytes of RAM have the ID number and related test parameters, if you need to use this part of the data in the program, you must not define IDATALEN as 256.

3. Project settings, select STC simulation driver.



As shown above, enter the project setting page firstly, select the "Debug" setting page, select the hardware emulation "Use..." on the right, and select "STC Monitor-51 Driver" in the emulation driver drop-down list. And then click the "Settings" button to enter the following setting screen. Set the port number and baud rate of the serial port. The baud rate is generally 115200. Then complete the setup.

4. Create simulation chip



Prepare a STC8H series chip, and connect it to the serial port of the computer through the download board. Then select the correct chip model as shown above, and enter the "Keil simulation settings" page, click the button of the corresponding model. After the program downloading completes, the simulator is ready for use.

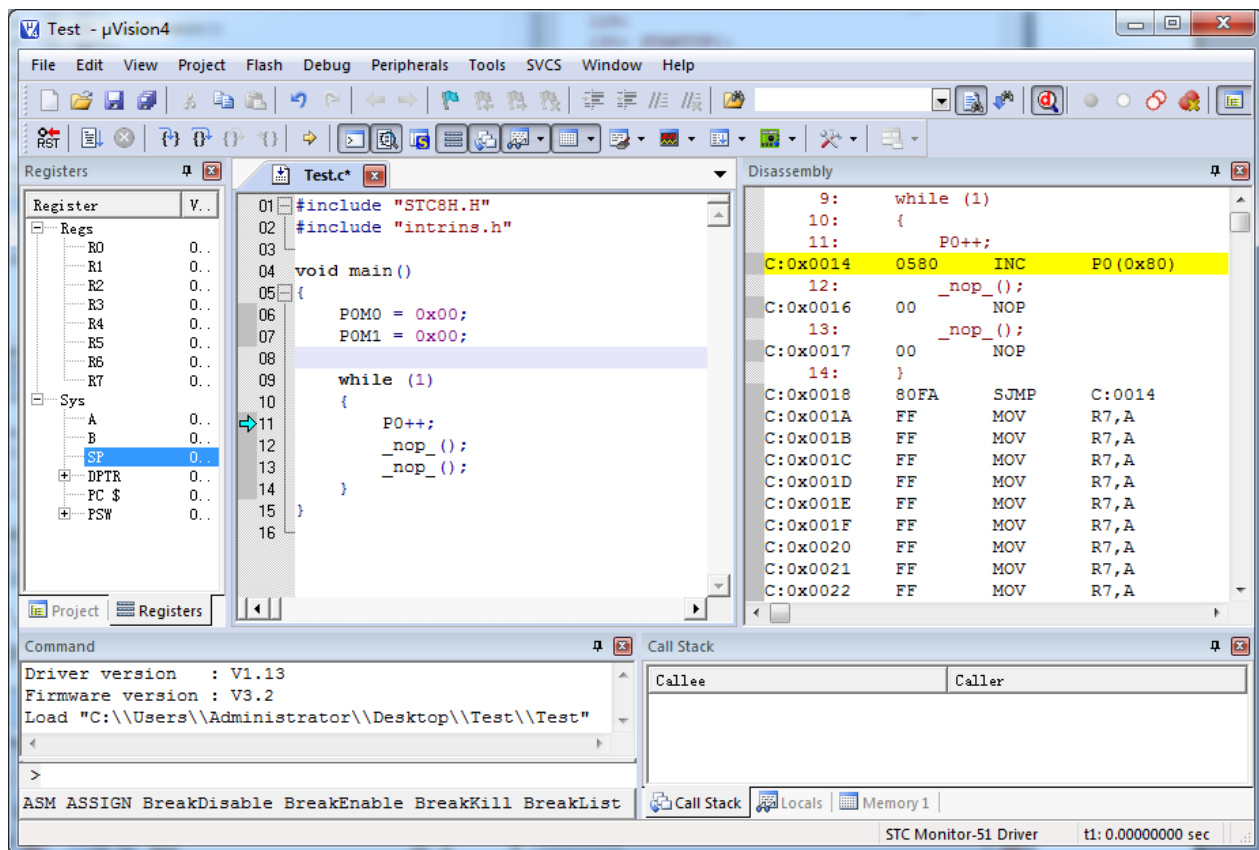
5. Start simulation

Connect the completed simulation chip to the computer through the serial port.

After compiling the project we created before without errors, press "Ctrl + F5" to start debugging.

If the hardware connection is correct, you will enter a debugging interface similar to the following, and display the current simulation driver version number and the current simulation monitoring code firmware version number in the command output window.

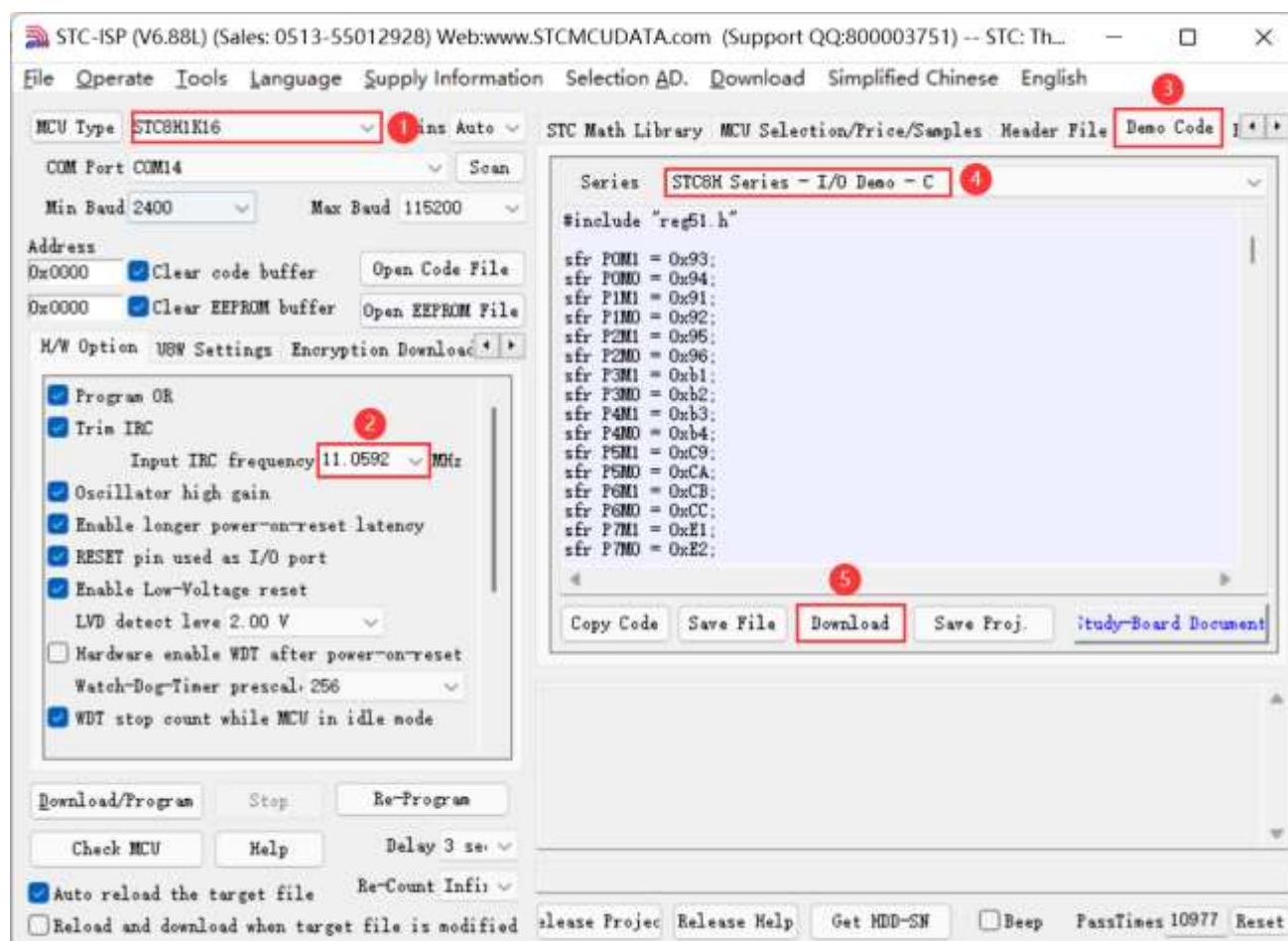
The current maximum number of breakpoints is 20 (in theory, any number of breakpoints can be set, but too many breakpoints will affect the speed of debugging).



Simulation considerations:

1. The simulation monitoring program occupies the two ports P3.0/P3.1, but does not occupy the serial port 1. The user can switch the serial port 1 to P3.6/P3.7 or P1.6/P1.7 to be used.
2. The simulation monitoring program occupies the last 768 bytes of the internal extended RAM (XDATA), and the user cannot write to XDATA in this area.

Appendix B How to Test I/O Ports



Test I/O port steps:

1. Select the microcontroller model
2. Set the operating frequency of the test program
3. Open the "Example Program" page
4. Select the "I/O port test" program of STC8G or STC8H series
5. Click "Download Hex Directly" on the page

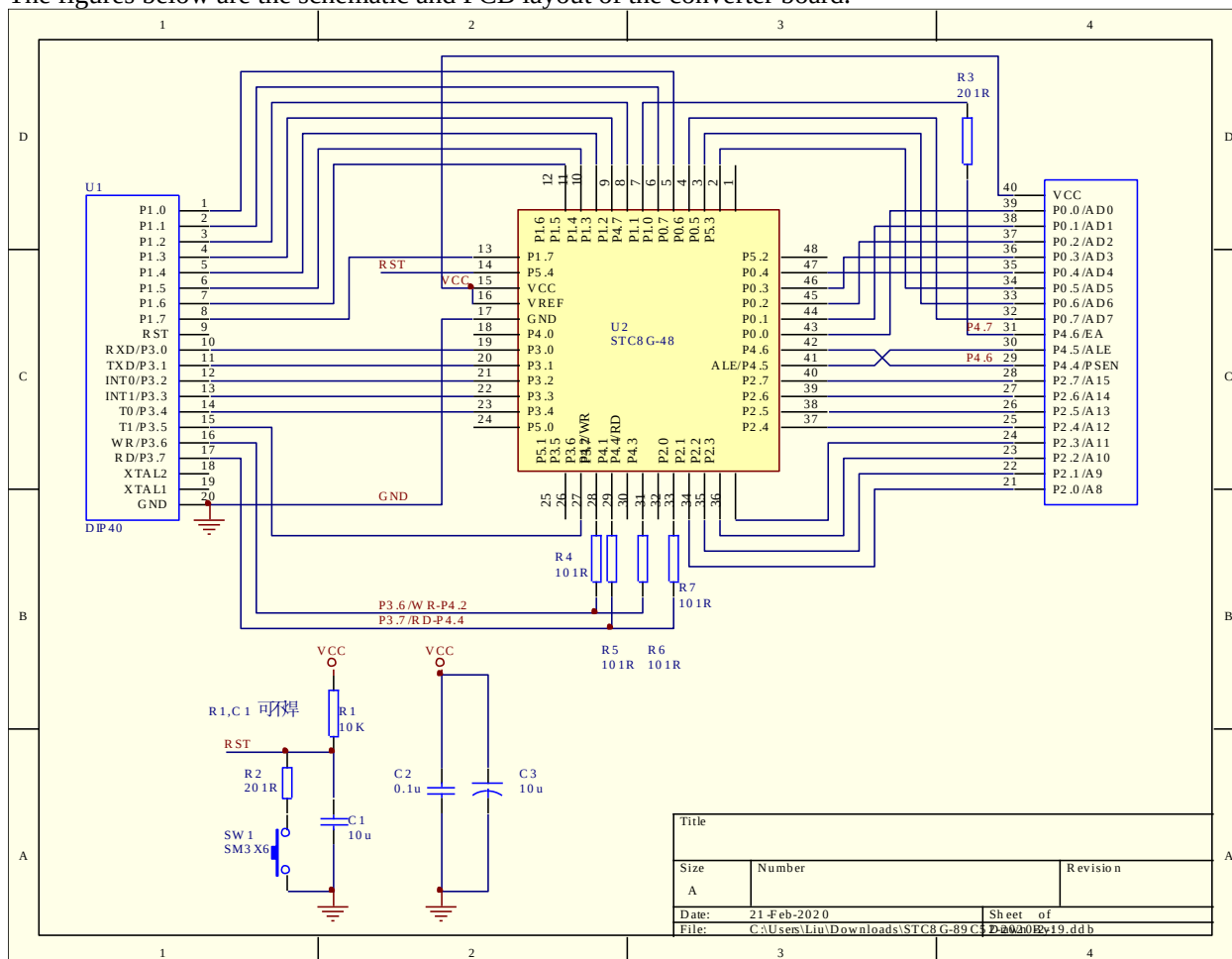
After the download is completed, the running water light program will be executed on all I/O ports. At this time, you can connect LEDs to the I/O ports or use an oscilloscope to see the waveform.

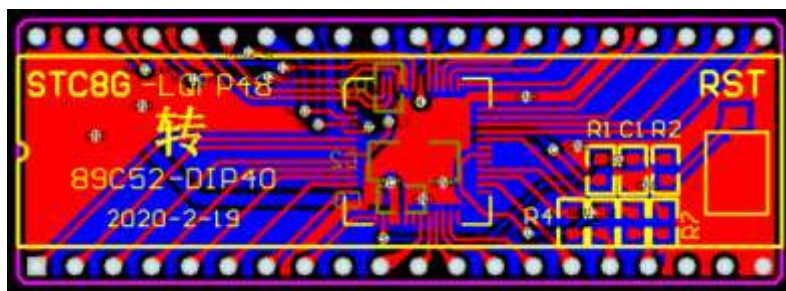
Appendix C How to Make the Traditional 8051

MCU EVB Emulatable

The traditional 8051 microcontroller EVB does not have simulation function. To enable the traditional 8051 microcontroller EVB to be simulated, a conversion board is needed. The physical picture of the conversion board is shown below. The converted pin arrangement is basically the same as that of the traditional 8051. Therefore, the simulation function of the standard 8051 learning board can be realized.

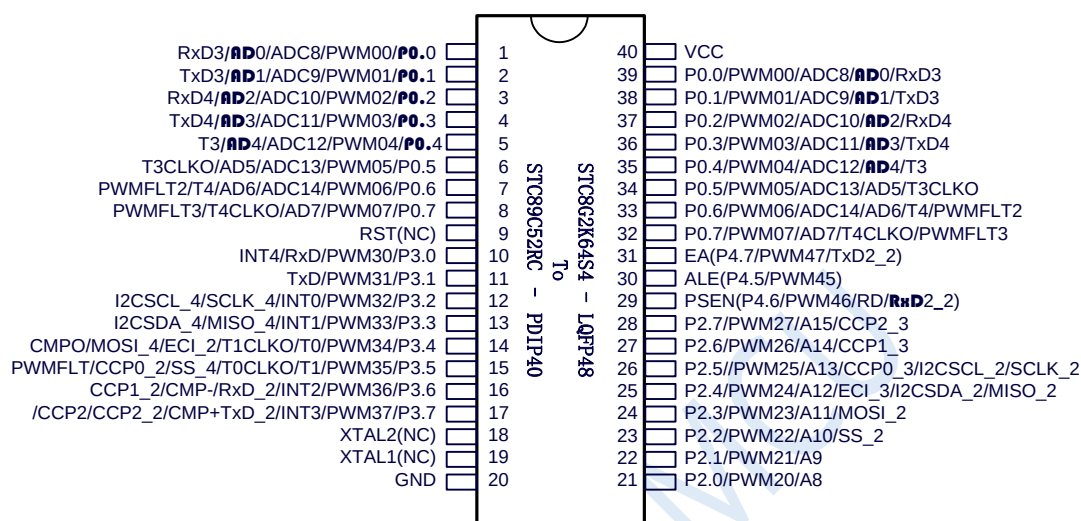
The figures below are the schematic and PCB layout of the converter board.





This conversion board can be used for STC8H series LQFP48 to STC89C52RC / STC89C58RD + series simulation.

The following figure is a functional diagram of the conversion board.



Note:

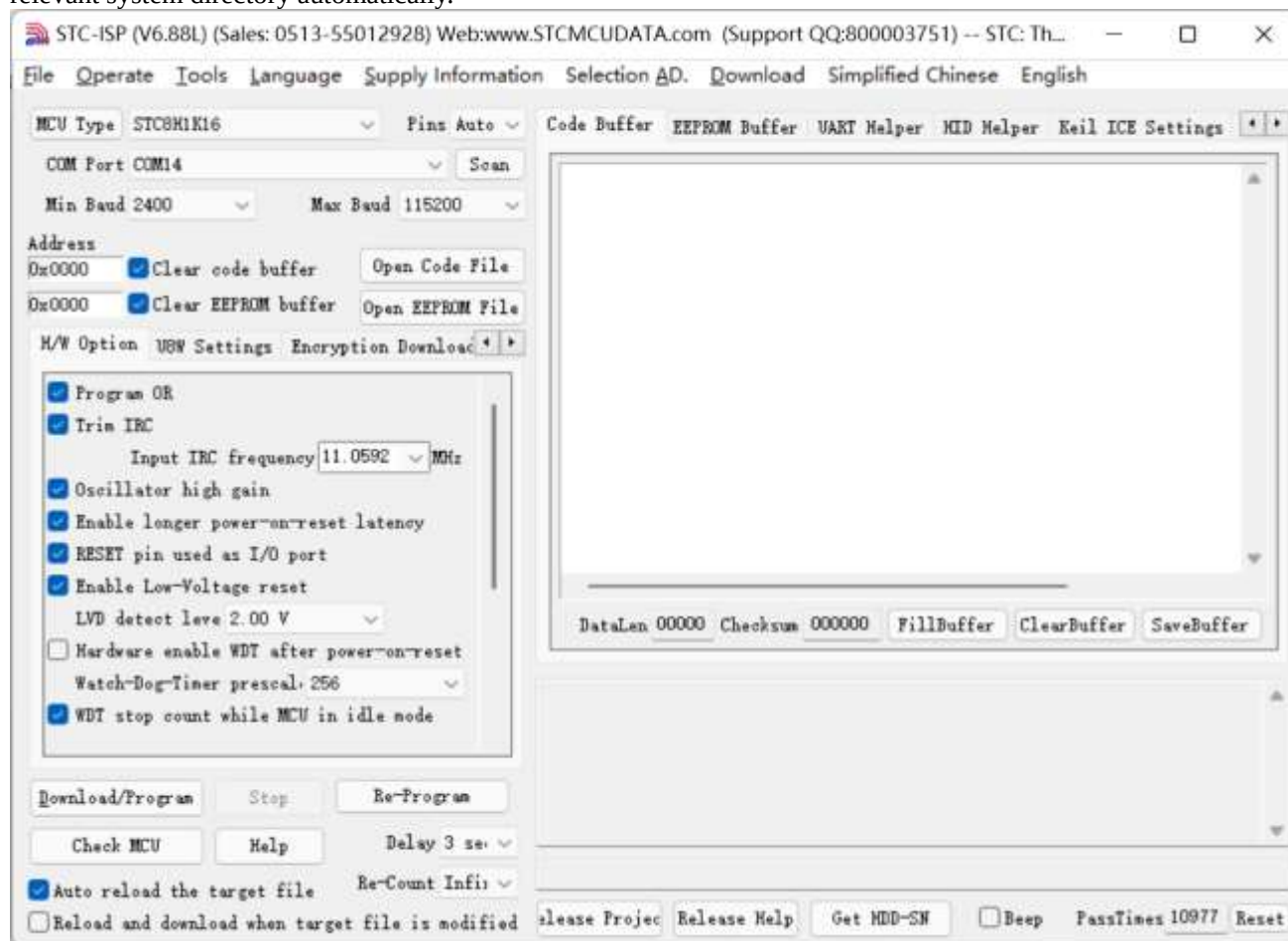
- ✓ Due to the built-in high-precision R/C clock, no external crystal is needed, XTAL1 and XTAL2 can be empty.
- ✓ WR and RD are P4.2/ WR and P4.4/ RD respectively, not traditional WR/P3.6 and RD/P3.7. **(In the conversion board, P4.2 and P3.6 are connected together, and P4.4 and P3.7 are connected together. When this conversion board is used to access the external bus, P3.6 and P3.7 should be set to high-impedance input mode, so that P4.2 and P4.4 can normally output the bus read and write signals. If the external bus is not needed to be accessed, P4.2 and P4.4 should be set to high-impedance input mode, and P3.6 and P3.7 are ordinary I/O.)**
- ✓ The STC8H series MCUs are low-level reset, it is not compatible with the high-level reset of the traditional 8051, so the RST pin is left floating, and replaced by the reset button and reset circuit on the conversion board.

Appendix D STC-USB Driver Installation

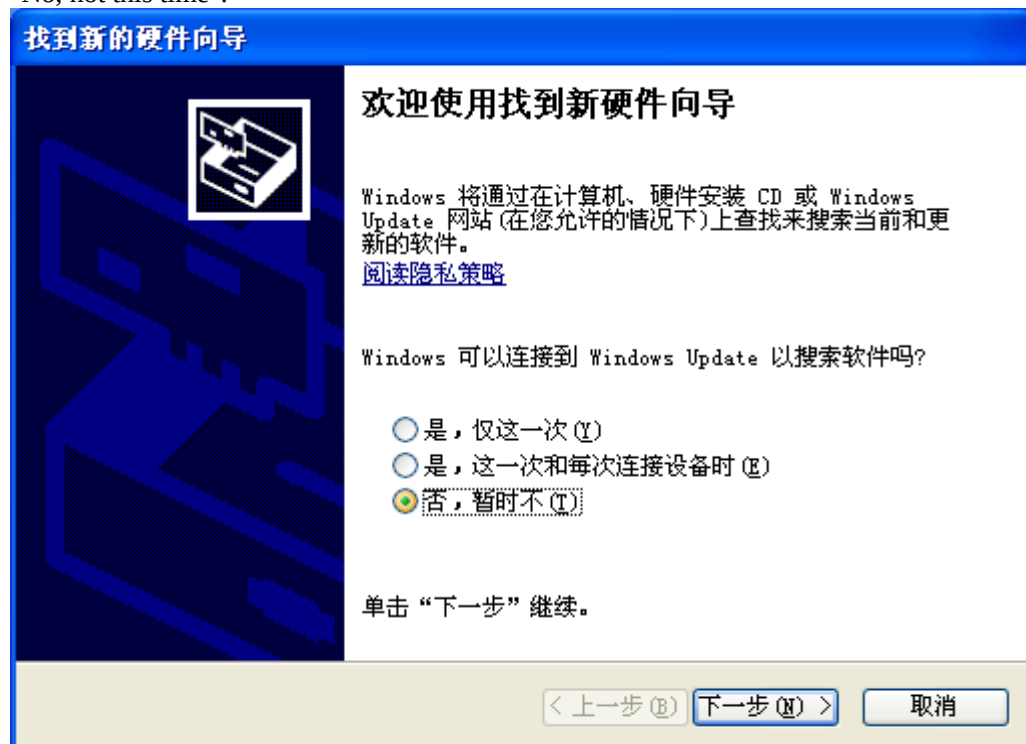
Instructions

Installation Instructions in Windows XP

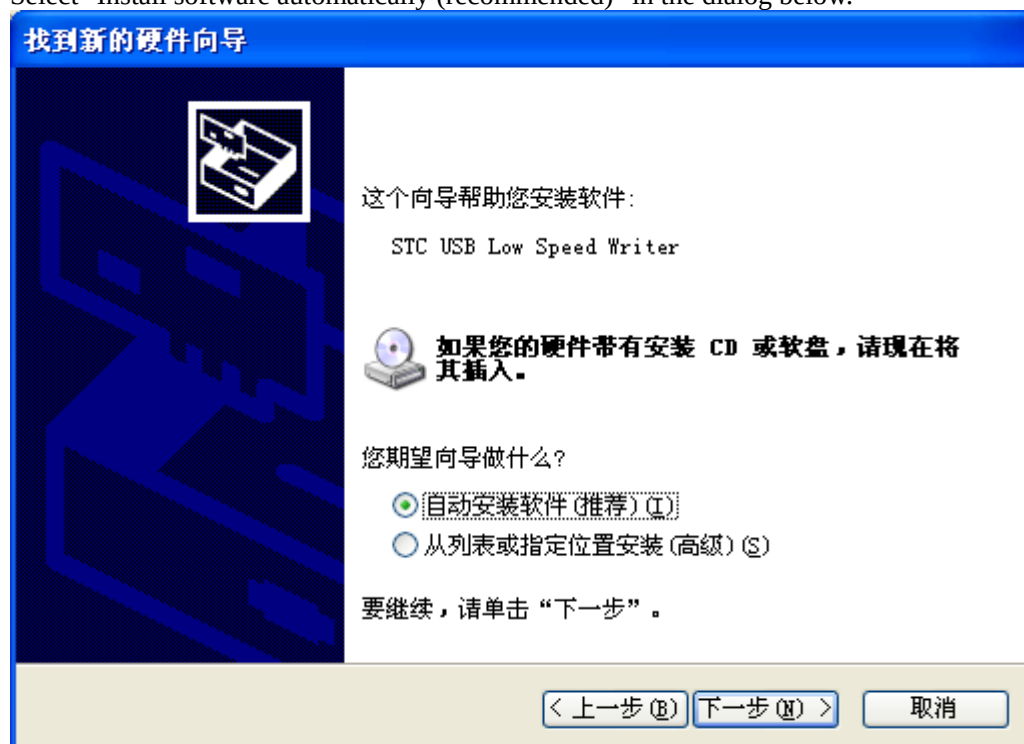
Open the STC-ISP download software of V6.79 (or later). The download software will copy the driver files to the relevant system directory automatically.



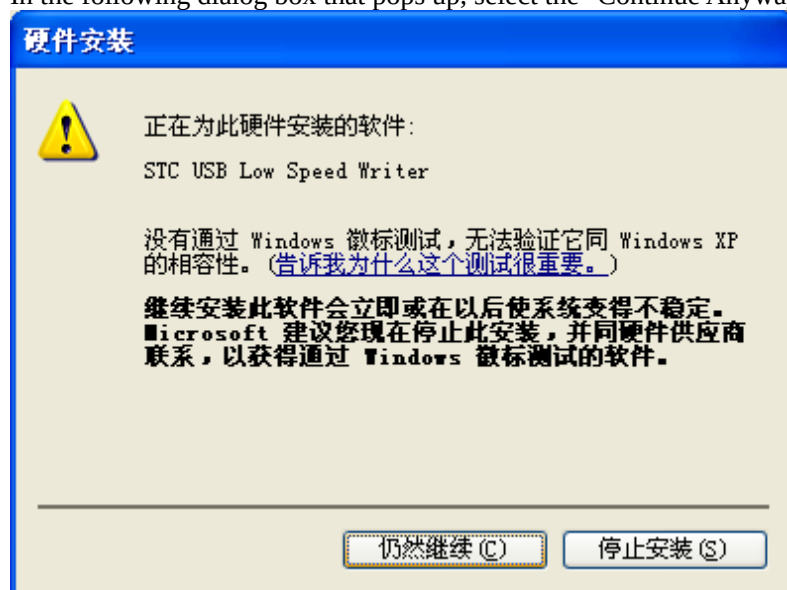
Plug in the USB device, the system will pop up the following dialog box automatically after finding the device, select "No, not this time".



Select "Install software automatically (recommended)" in the dialog below.



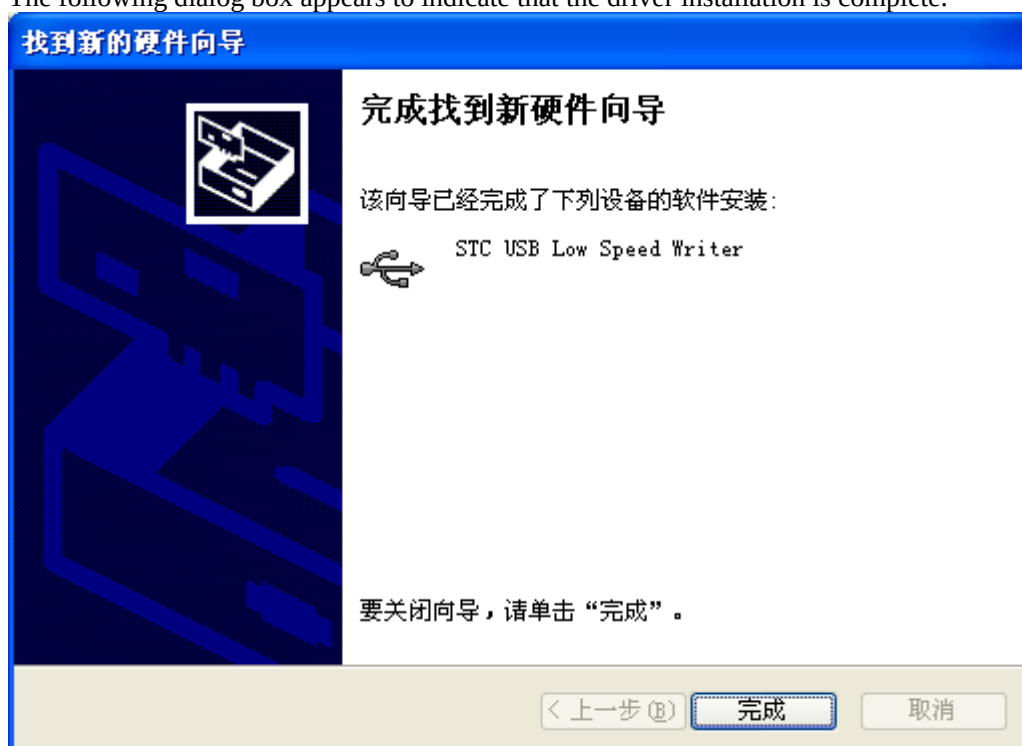
In the following dialog box that pops up, select the "Continue Anyway" button.



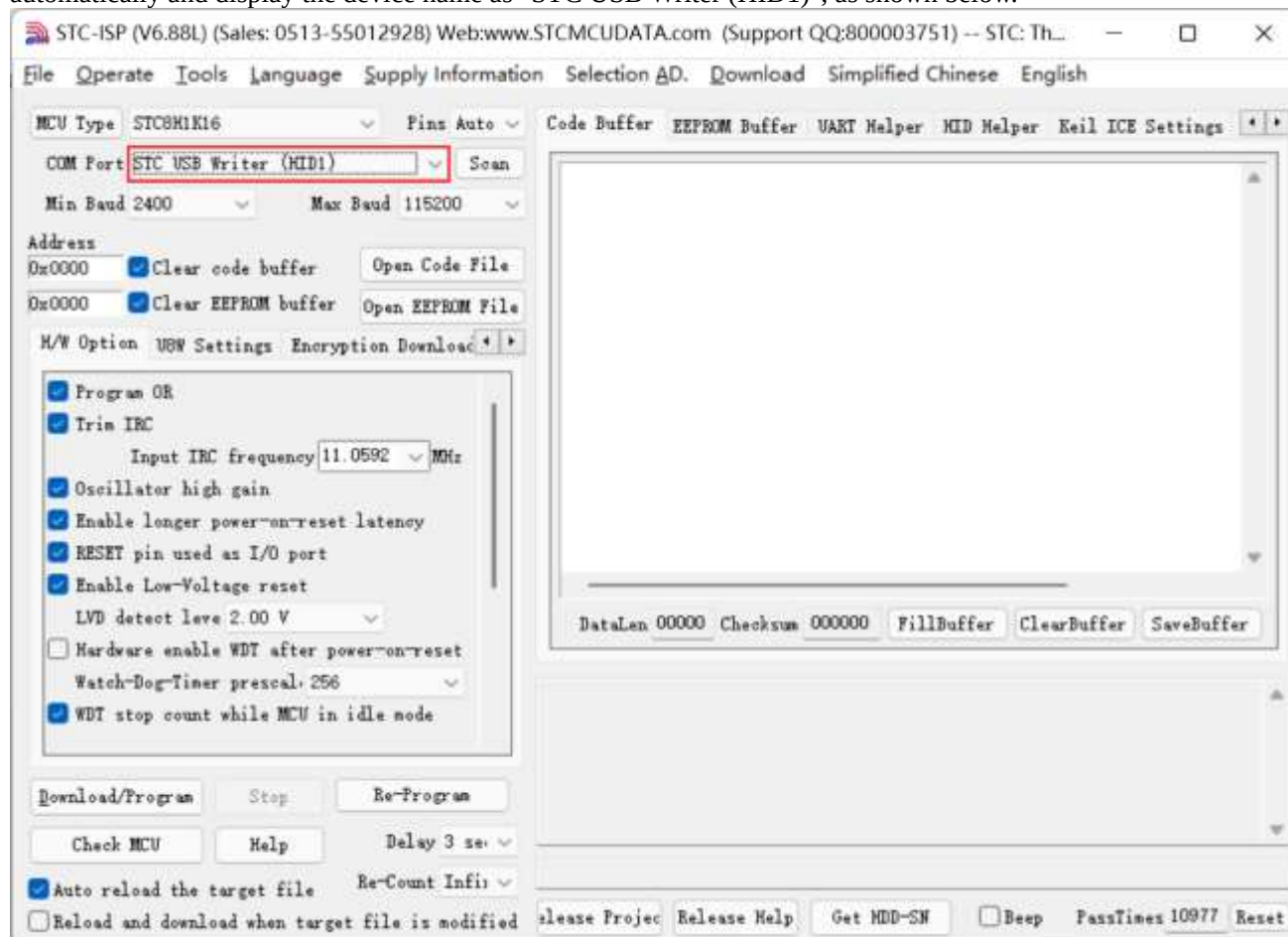
The system will automatically install the driver when connected, as shown below



The following dialog box appears to indicate that the driver installation is complete.

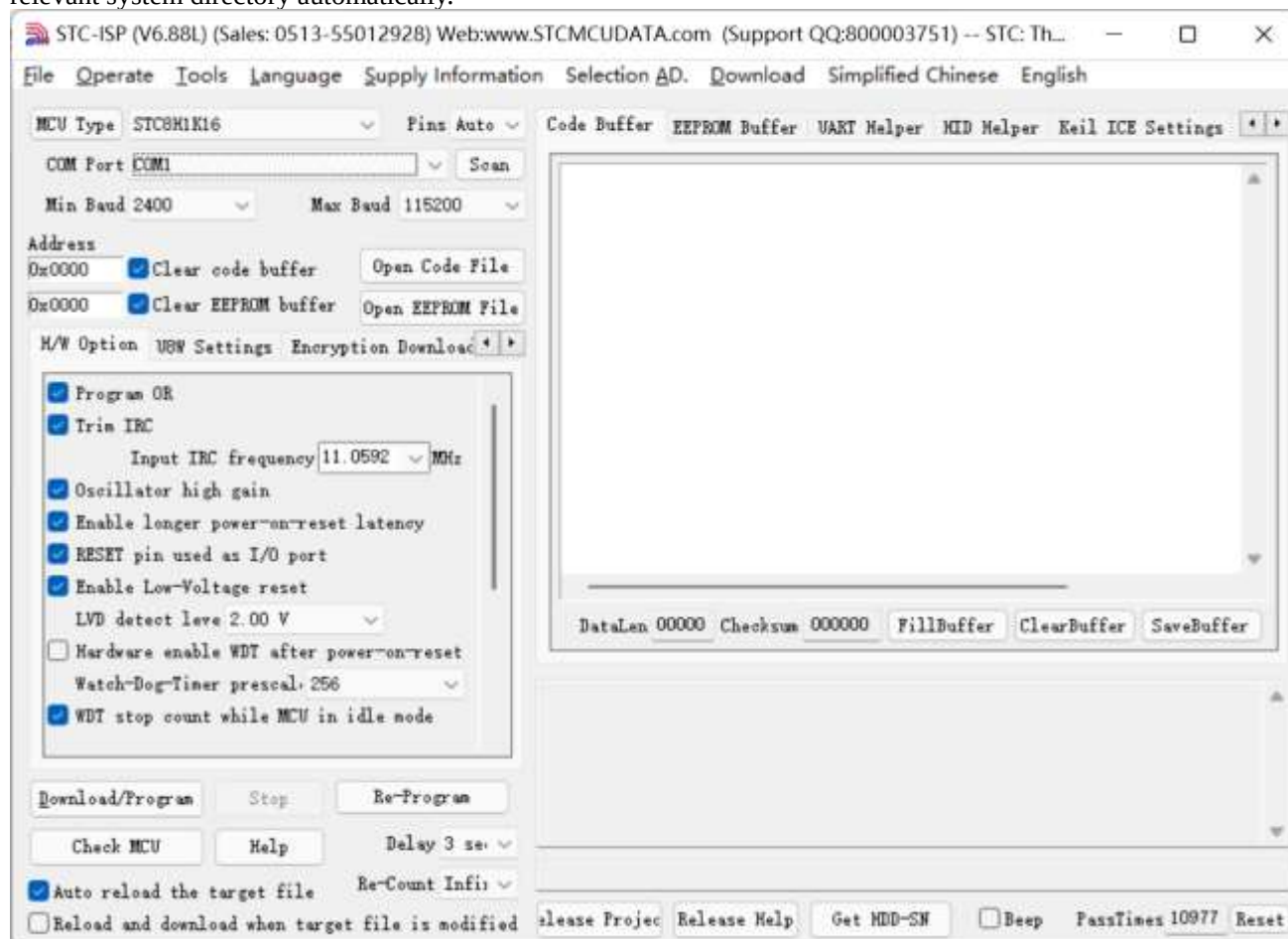


Now, the serial number list in the previously opened STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (HID1)", as shown below.



Installation Instructions in Windows 7 (32-bit)

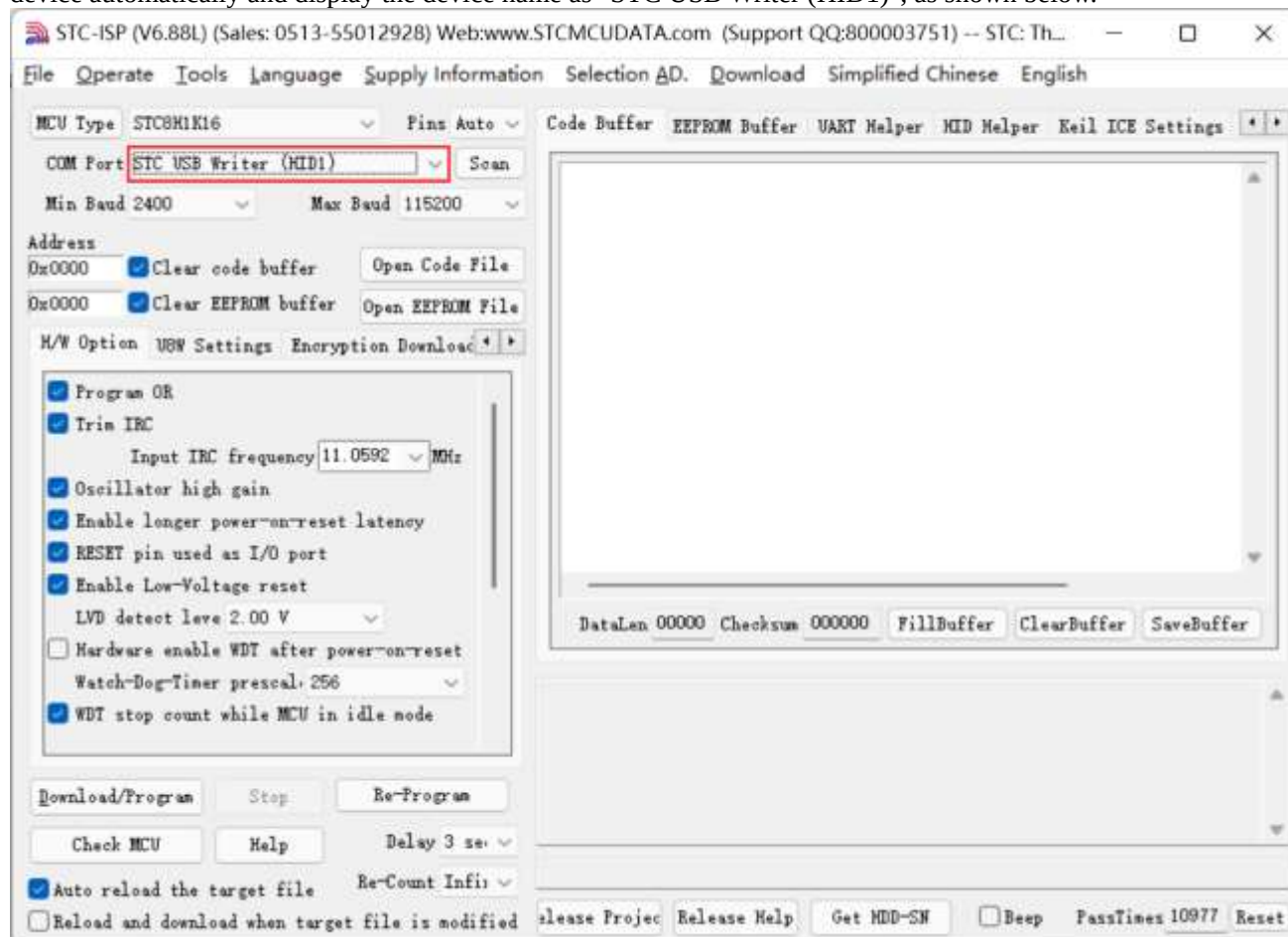
Open the STC-ISP download software of V6.79 (or later). The download software will copy the driver files to the relevant system directory automatically.



Plug in the USB device, and the system will install the driver automatically when it finds the device. After the installation is complete, the following prompt box will appear.



Now, the serial port number list in the previously opened STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (HID1)", as shown below.

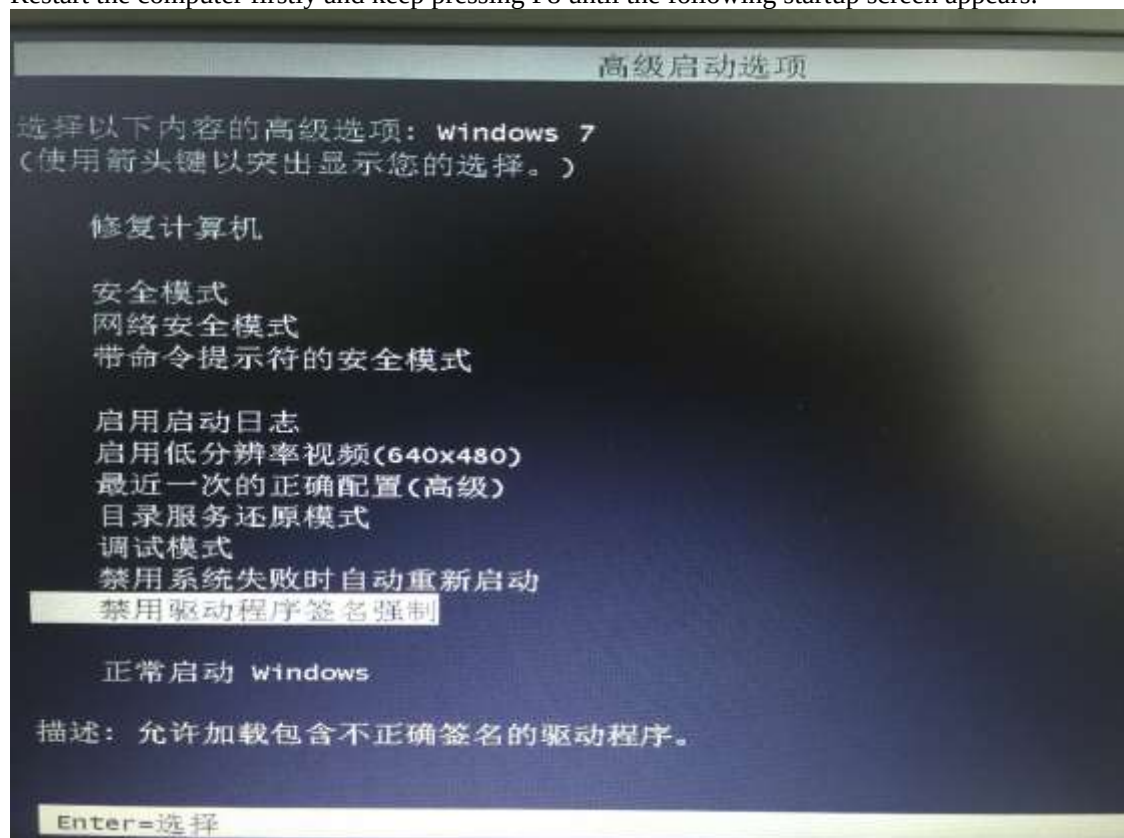


Note: If the system does not install the driver automatically in Windows 7, please refer to the installation method of Windows 8 (32-bit) for the driver installation method.

Installation Instructions in Windows 7 (64-bit)

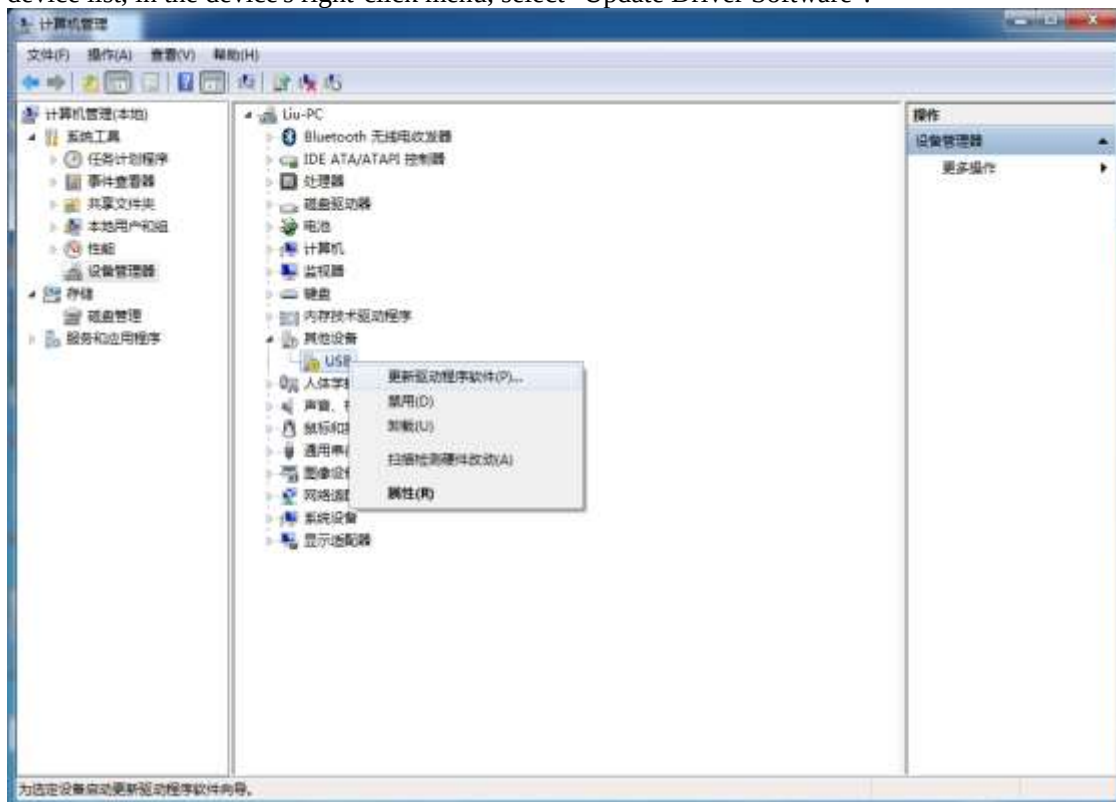
By default, the driver without digital signature cannot be successfully installed in Windows 7 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

Restart the computer firstly and keep pressing F8 until the following startup screen appears.



Select 'Disable Driver Signature Enforcement'. The digital signature verification function is temporarily turn off after startup.

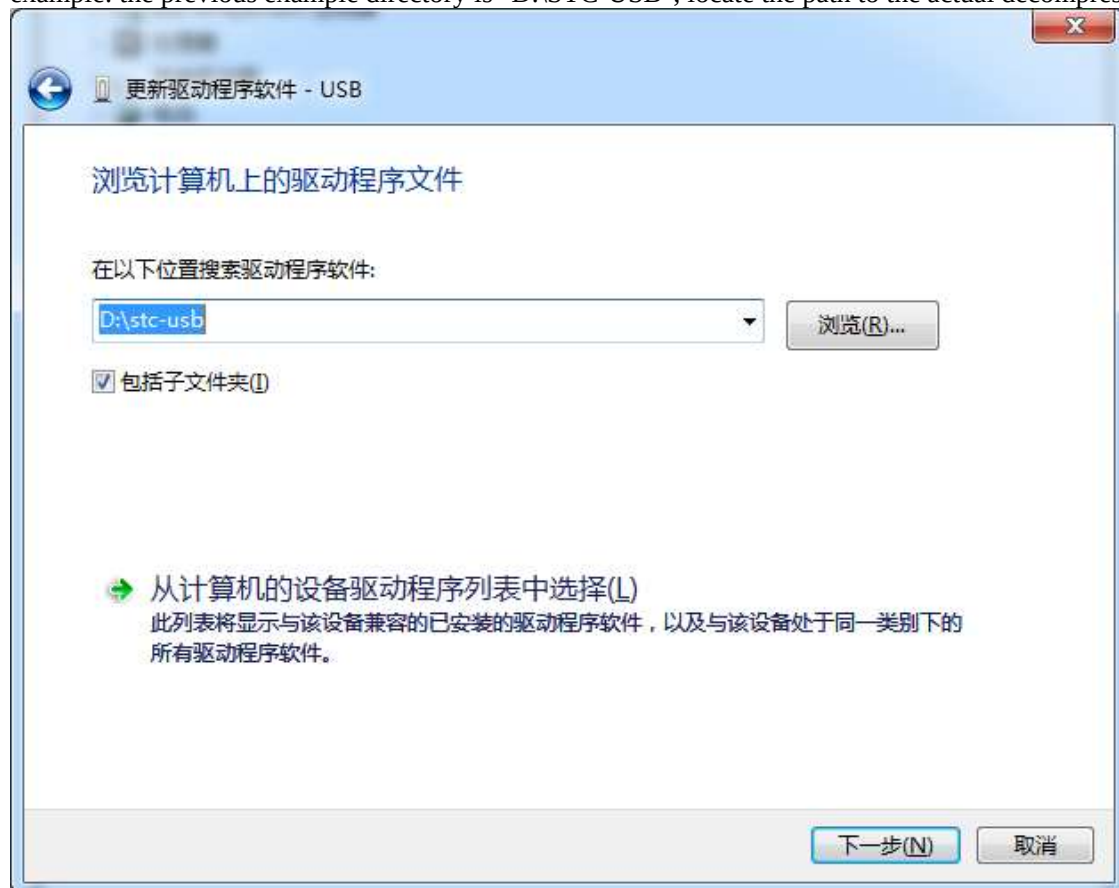
Plug in the USB device and open the Device Manager. Find the USB device with a yellow exclamation mark in the device list, in the device's right-click menu, select "Update Driver Software".



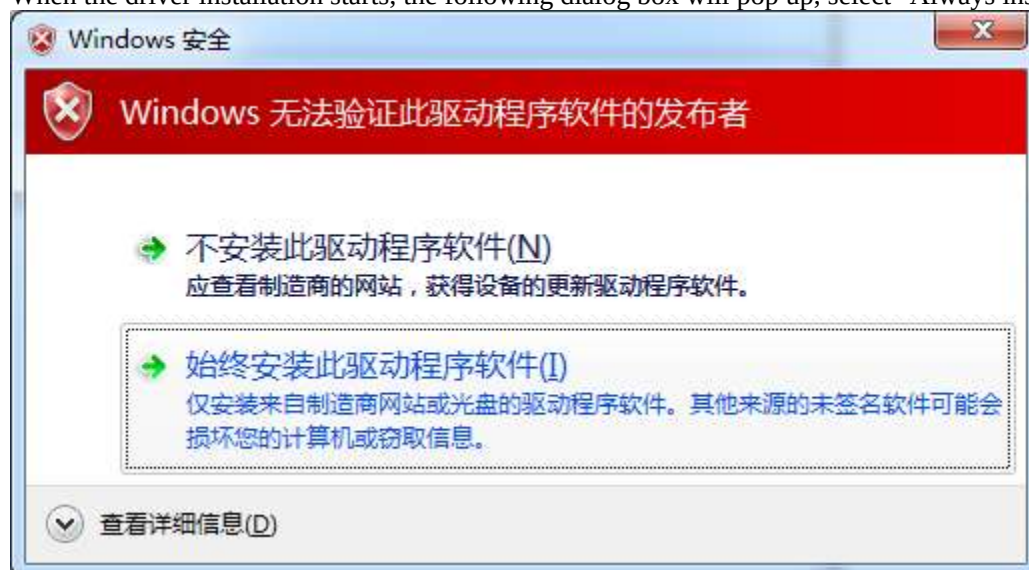
Select "Browse my computer for driver software" in the dialog below.



Click the "Browse" button in the dialog below to find the directory of the previous STC-USB driver stored (for example: the previous example directory is "D:\STC-USB", locate the path to the actual decompression directory).



When the driver installation starts, the following dialog box will pop up, select "Always install this driver software".



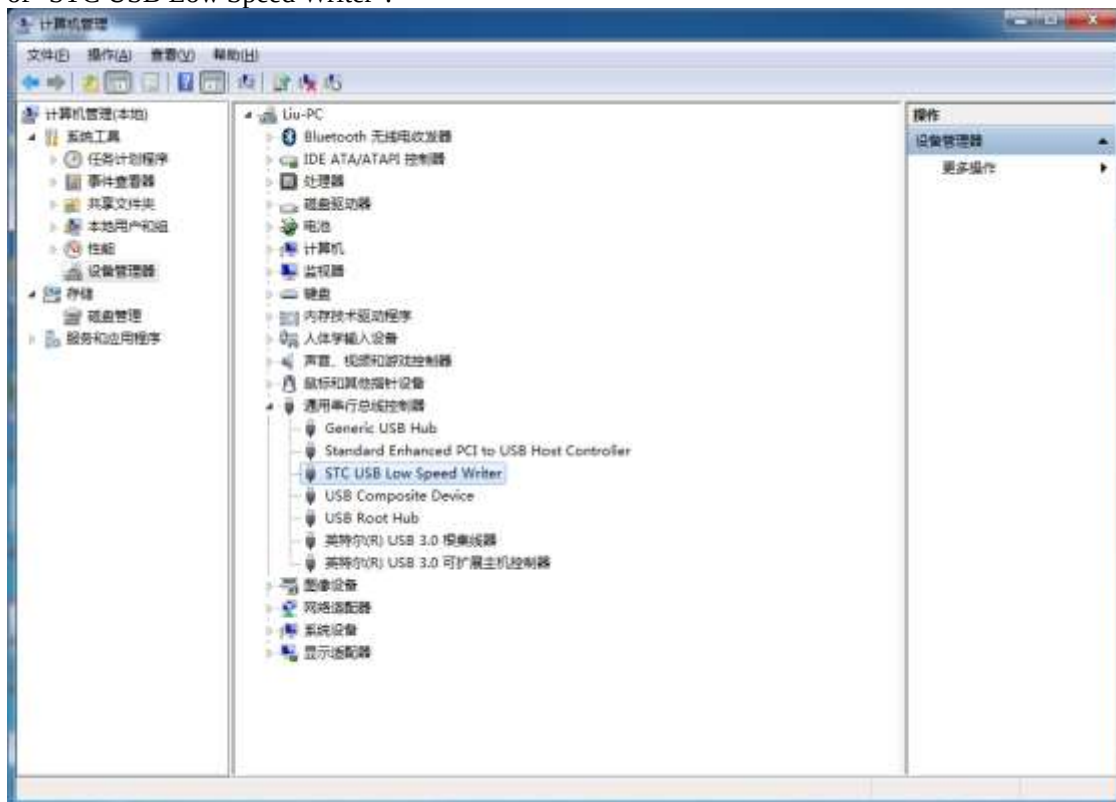
Next, the system will install the driver automatically, as shown below.



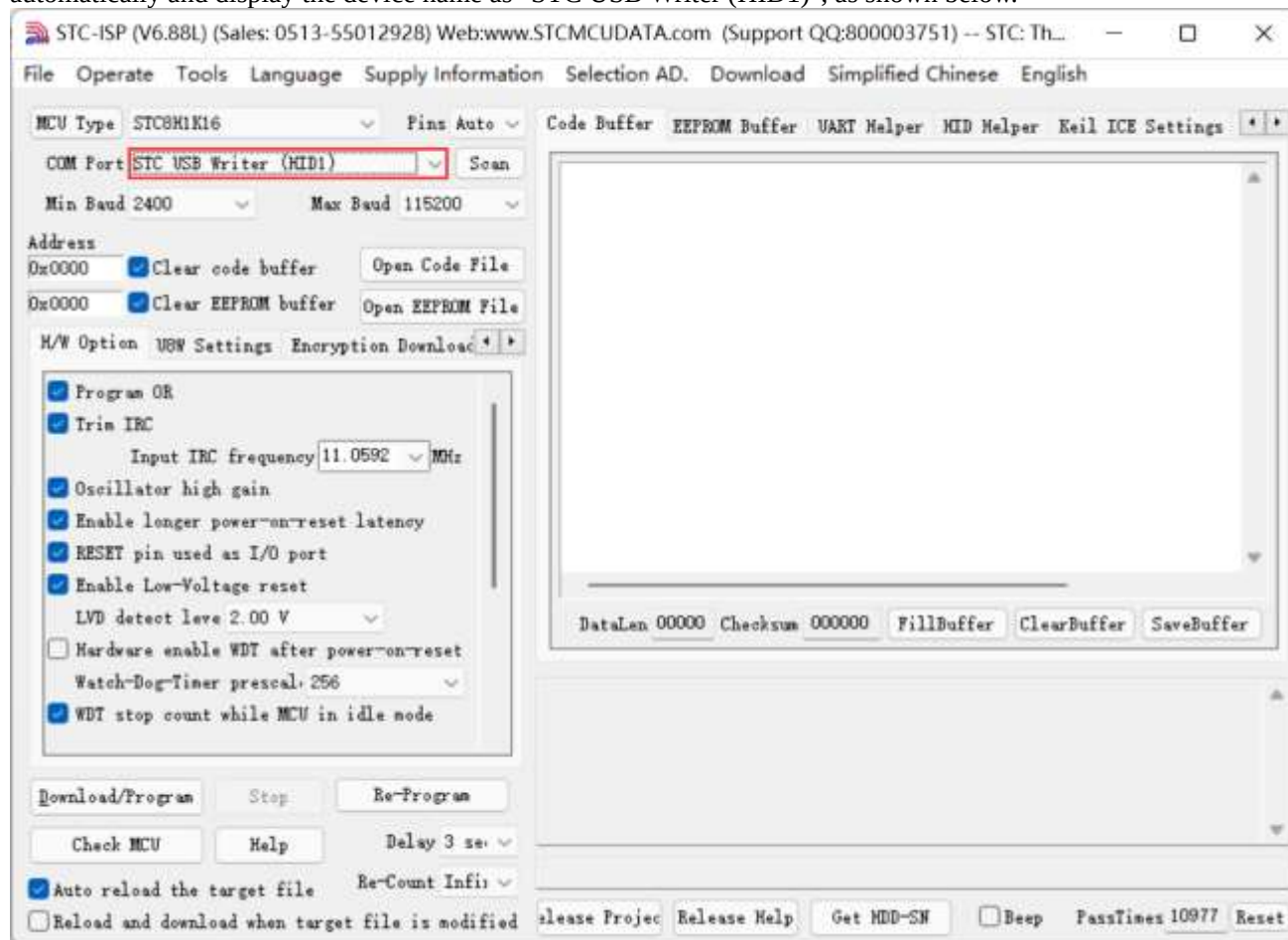
The following dialog box appears to indicate that the driver installation is complete.



Now in the device manager, the device with the yellow exclamation mark before will be displayed as the device name of "STC USB Low Speed Writer".

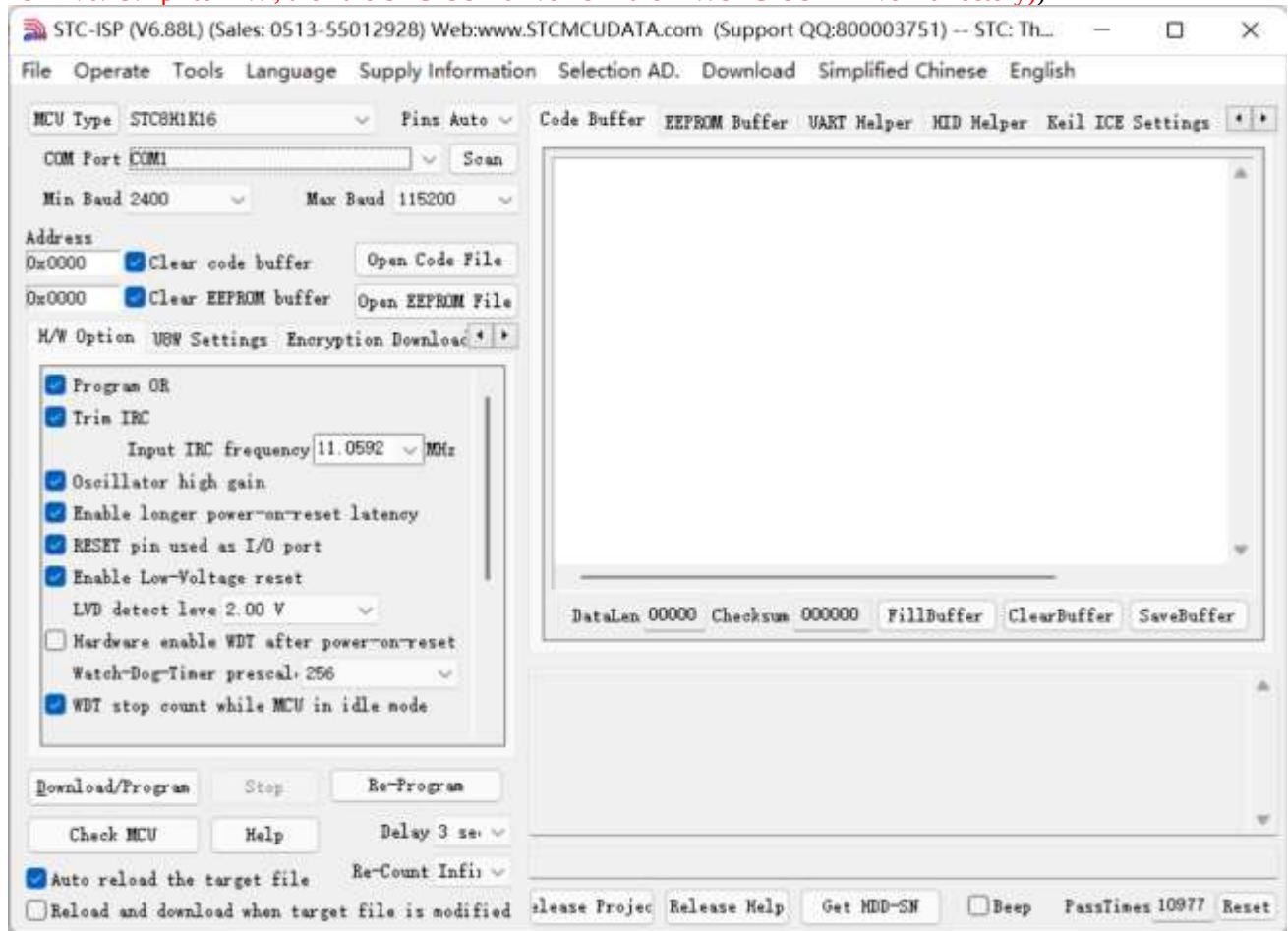


The serial number list in the previously downloaded STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (HID1)", as shown below.

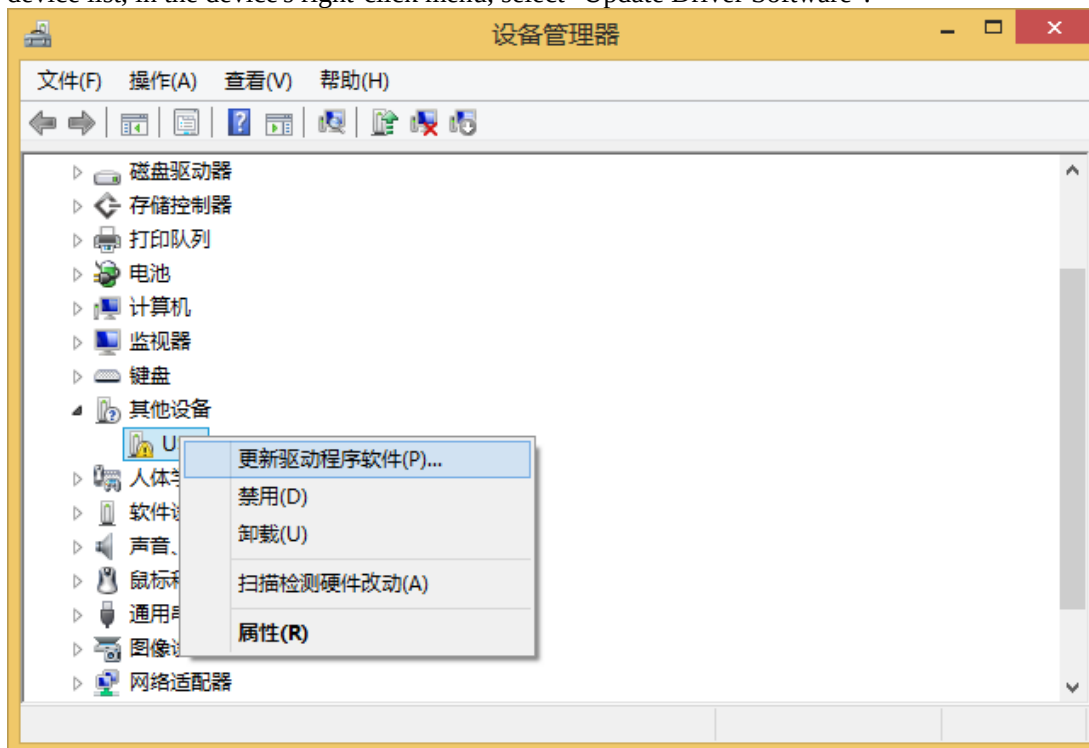


Installation Instructions in Windows 8 (32-bit)

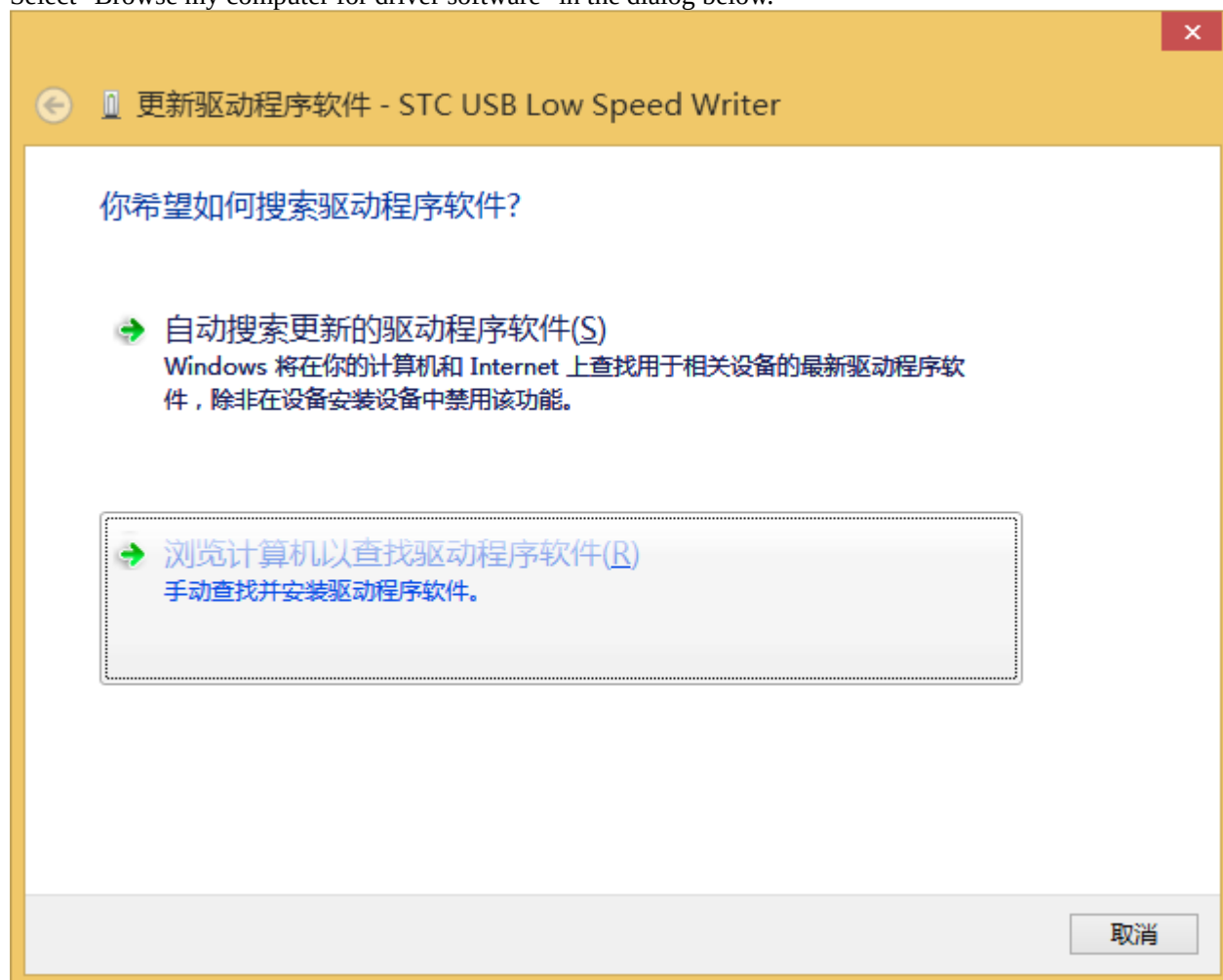
Open the STC-ISP download software of V6.79 (or newer version) (Due to permission reasons, downloading the software in Windows 8 will not copy the driver files to the relevant system directory. It requires manual installation by the user. Firstly, download "stc-isp-15xx-v6.79.zip" (or newer version) from the STC official website, and decompress it to the local disk after downloading, then the STC-USB driver file will also be decompressed to the "STC-USB Driver" folder of the current folder. (For example, decompress the downloaded compressed file "stc-isp-15xx-v6.79.zip" to "F:", then the STC-USB driver is in the "F:\STC-USB Driver" directory))



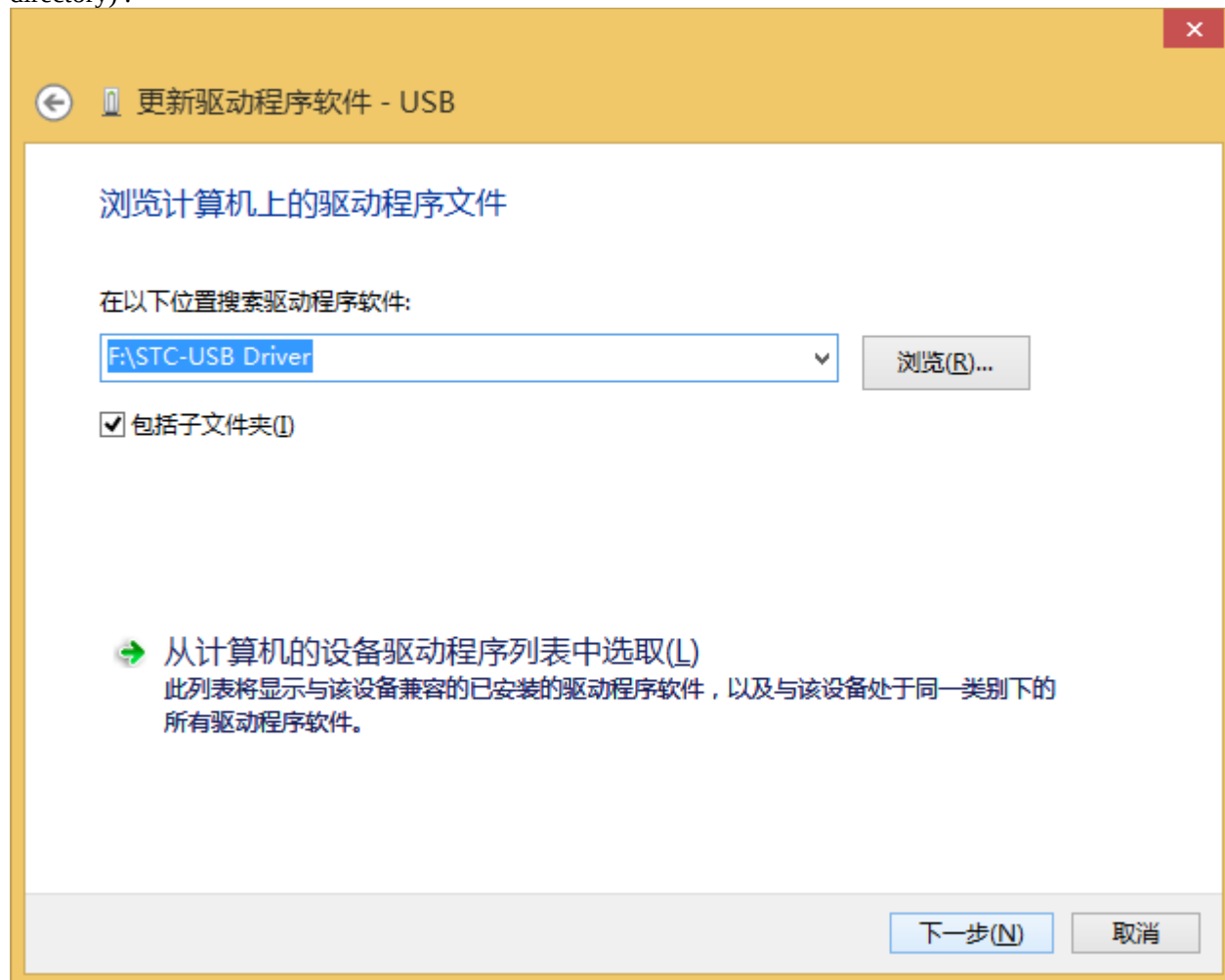
Plug in the USB device and open the Device Manager. Find the USB device with a yellow exclamation mark in the device list, in the device's right-click menu, select "Update Driver Software".



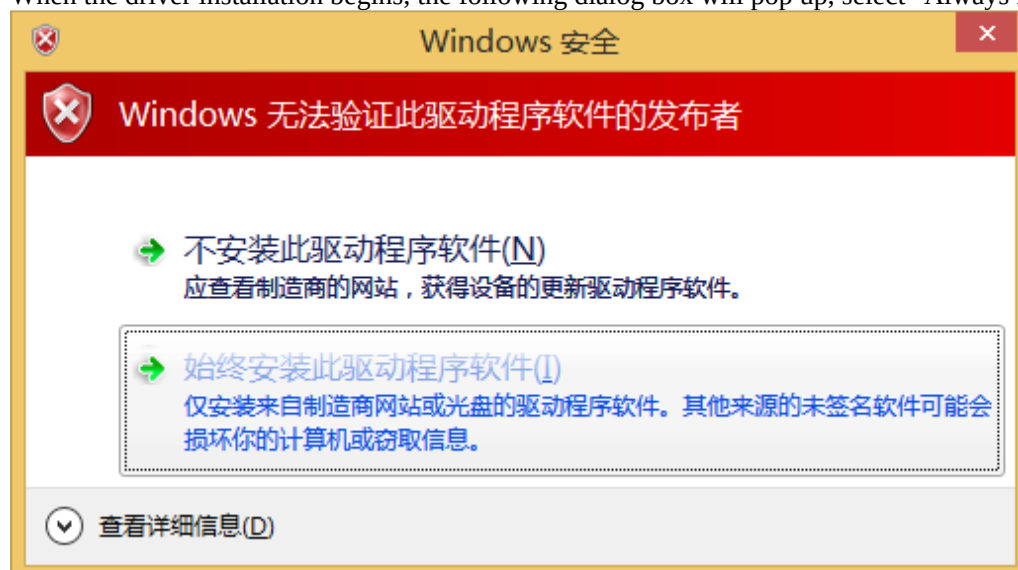
Select "Browse my computer for driver software" in the dialog below.



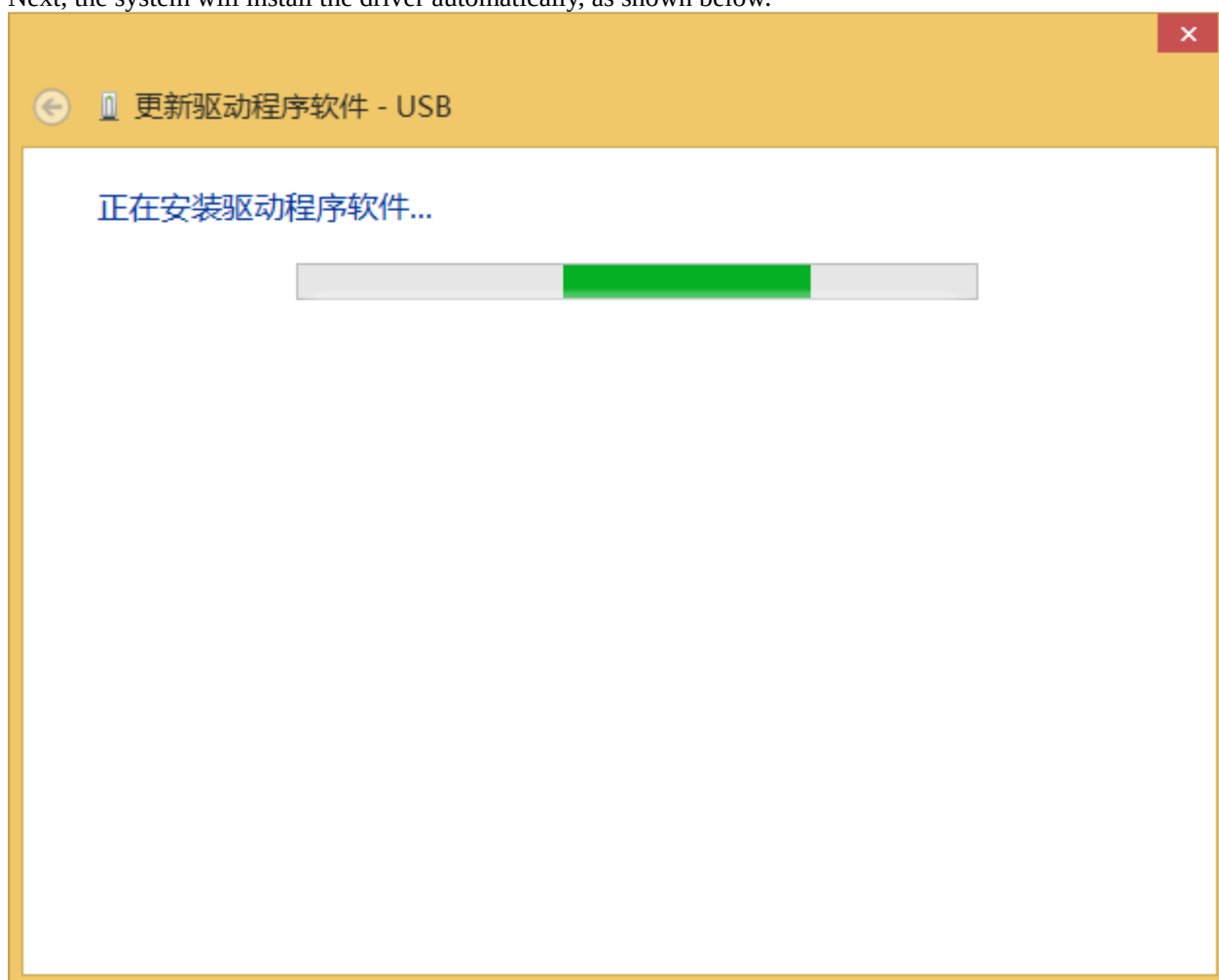
Click the "Browse" button in the dialog below to find the directory where the STC-USB driver was stored (for example: the previous example directory is "F: \ STC-USB Driver", locate the path to the actual decompression directory).



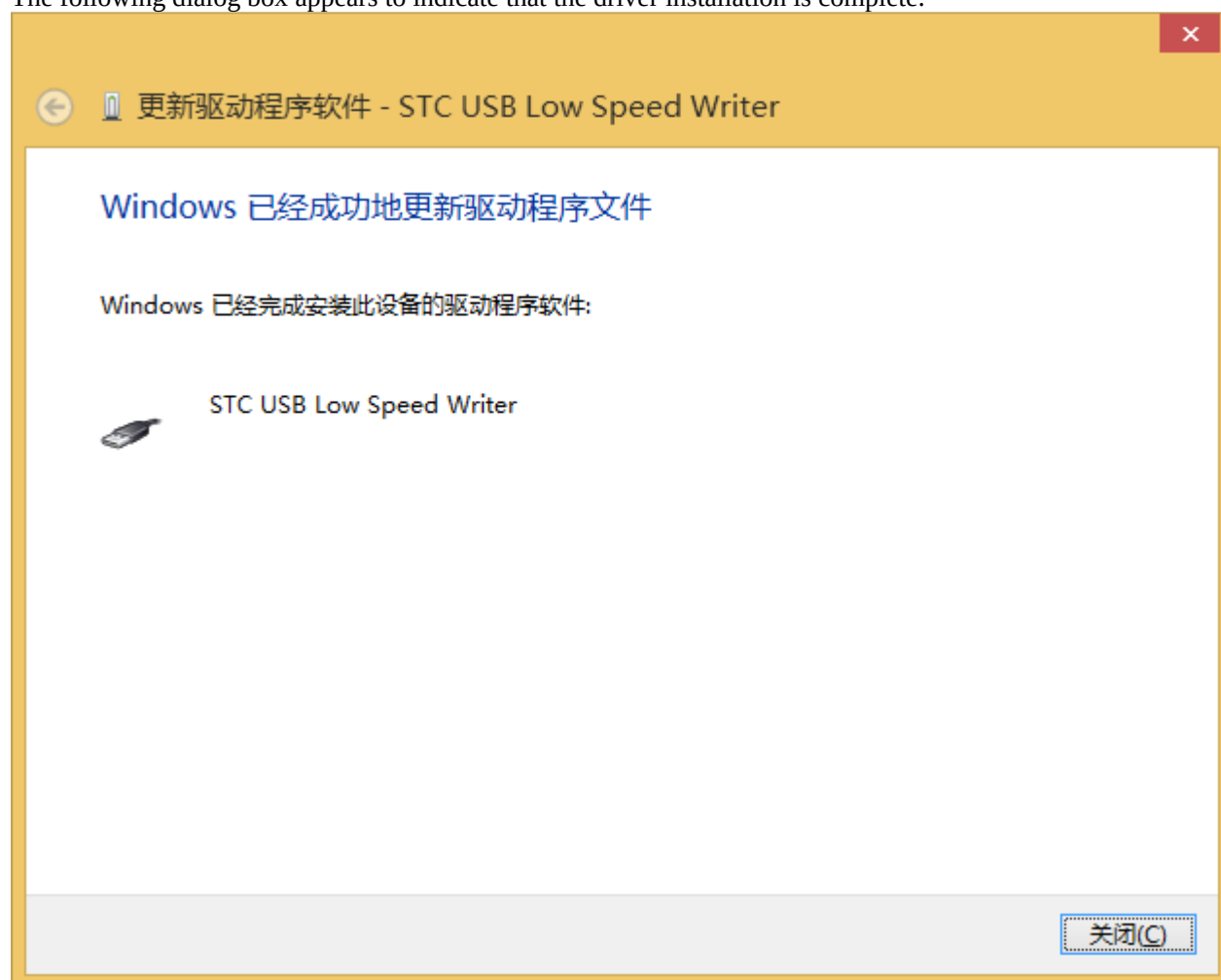
When the driver installation begins, the following dialog box will pop up, select "Always install this driver software".



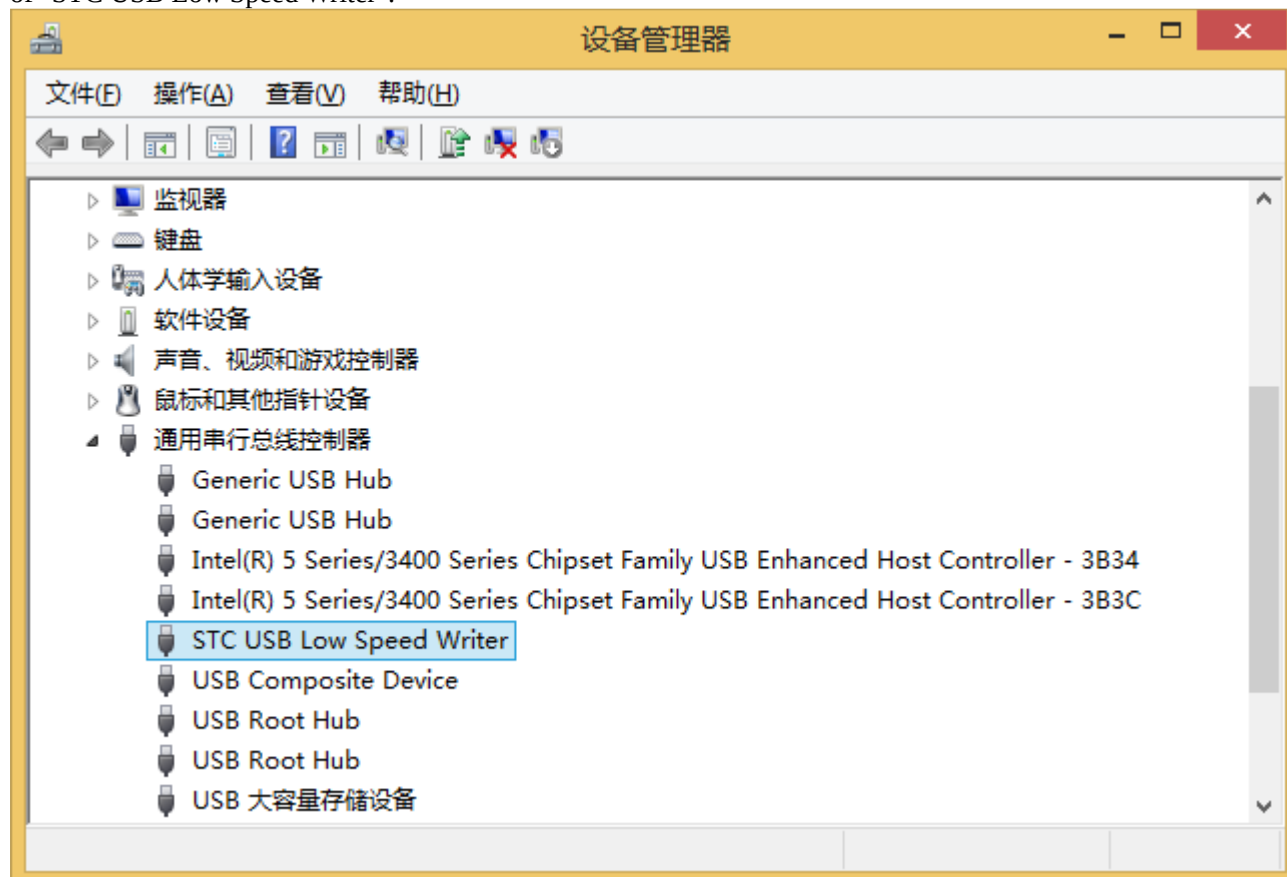
Next, the system will install the driver automatically, as shown below.



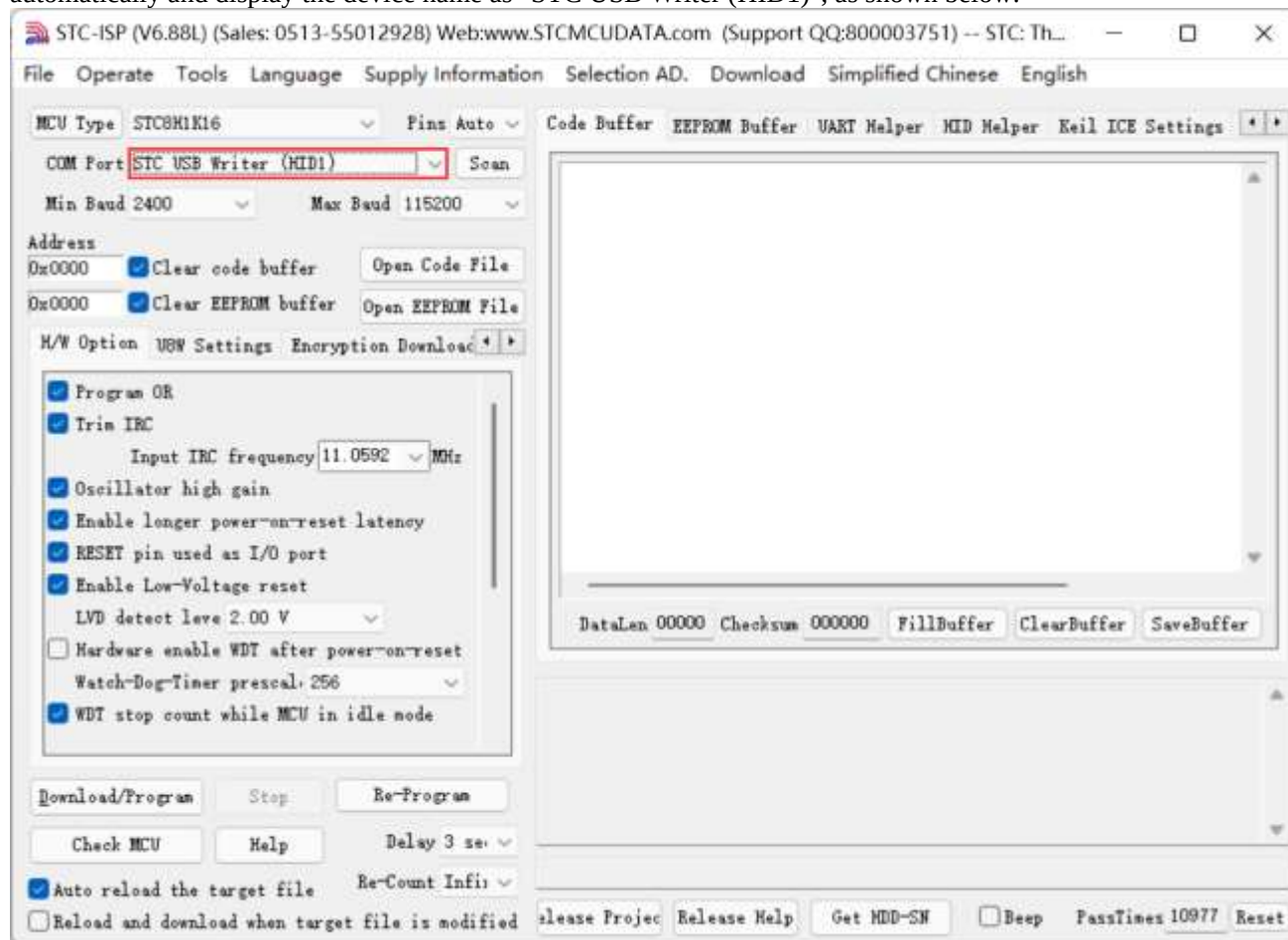
The following dialog box appears to indicate that the driver installation is complete.



Now in the device manager, the device with the yellow exclamation mark before will be displayed as the device name of "STC USB Low Speed Writer".



The serial number list in the previously downloaded STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (HID1)", as shown below:



Installation Instructions in Windows 8 (64-bit)

By default, the driver without digital signature cannot be successfully installed in Windows 8 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

Firstly, move the mouse to the lower right corner of the screen and select the "Settings" button.



Then select the "Change PC settings" item in the settings window.



In the computer settings, select the "Start Now" button under the "Advanced Startup" item in the "General" property page.



In the window below, select the "Troubleshooting" item.



Then select "Advanced Options" in "Troubleshooting".



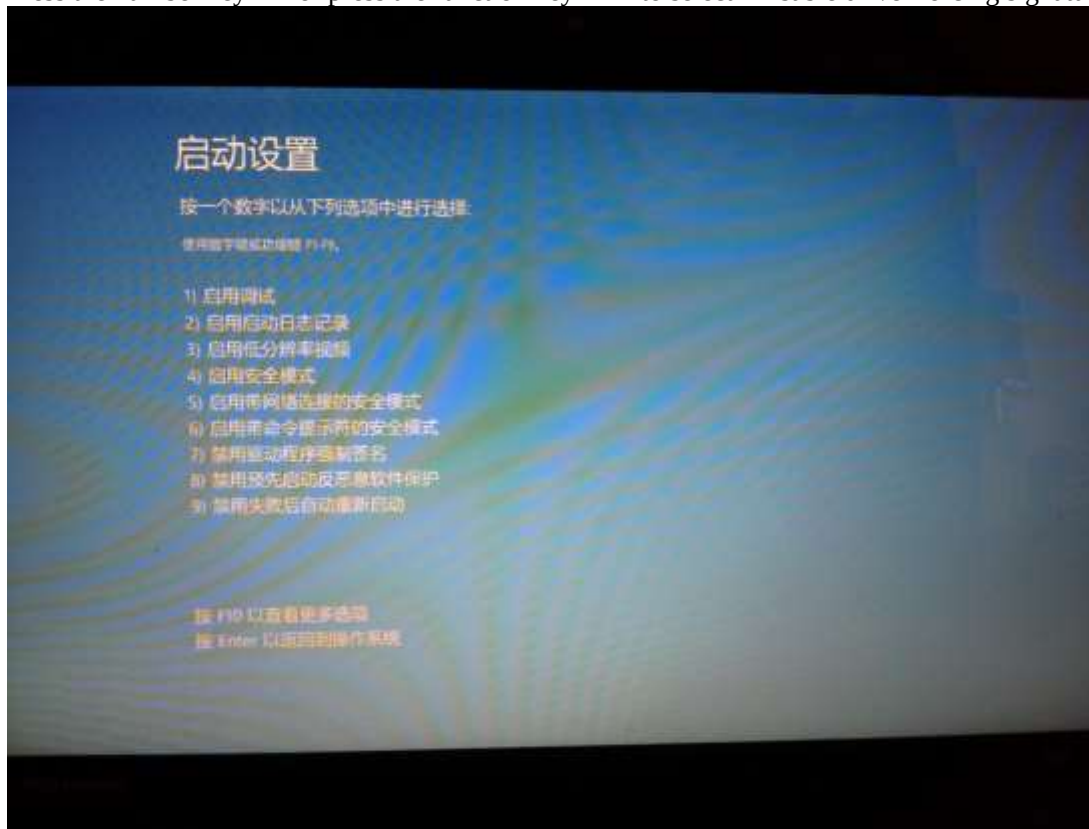
In the "Advanced Options" window below, select "Startup Settings".



In the "Startup Settings" window below, click the "Restart" button to restart the computer.



After the computer restarts, it will enter the "Startup Settings" window automatically as shown in the figure below. Press the number key "7" or press the function key "F7" to select "Disable driver forcing signature" to start.



After booting to Windows 8, follow the Windows 8 (32-bit) installation method to complete the driver installation.

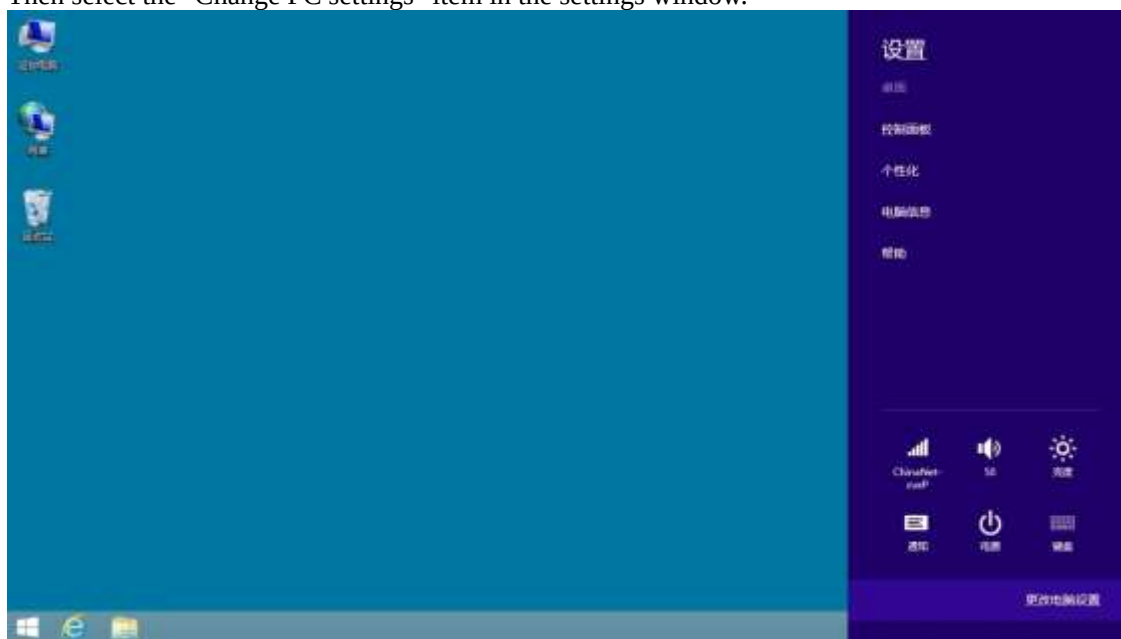
Installation Instructions in Windows 8.1 (64-bit)

Windows 8.1 has different method for entering the advanced boot menu with respect to Windows 8, which is specifically explained [here](#).

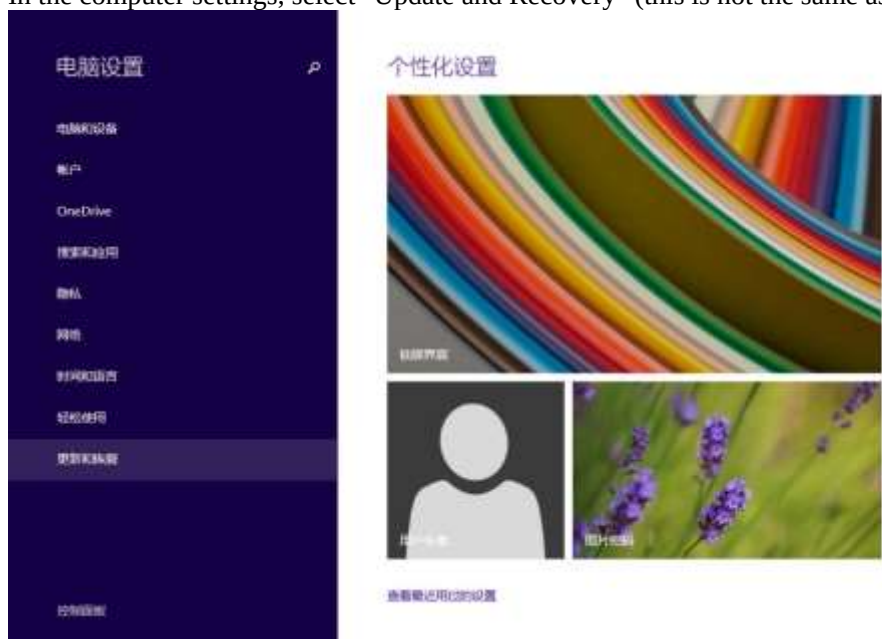
Firstly, move the mouse to the lower right corner of the screen and select the "Settings" button.



Then select the "Change PC settings" item in the settings window.



In the computer settings, select "Update and Recovery" (this is not the same as Windows 8, which is "General").



Select the "Restore" property page in the update and recovery page, and click the "Start Now" button under the "Advanced Startup" item.



The following steps are the same as those of Window 8.
In the window below, select the "Troubleshooting" item.



Then select "Advanced Options" in "Troubleshooting".



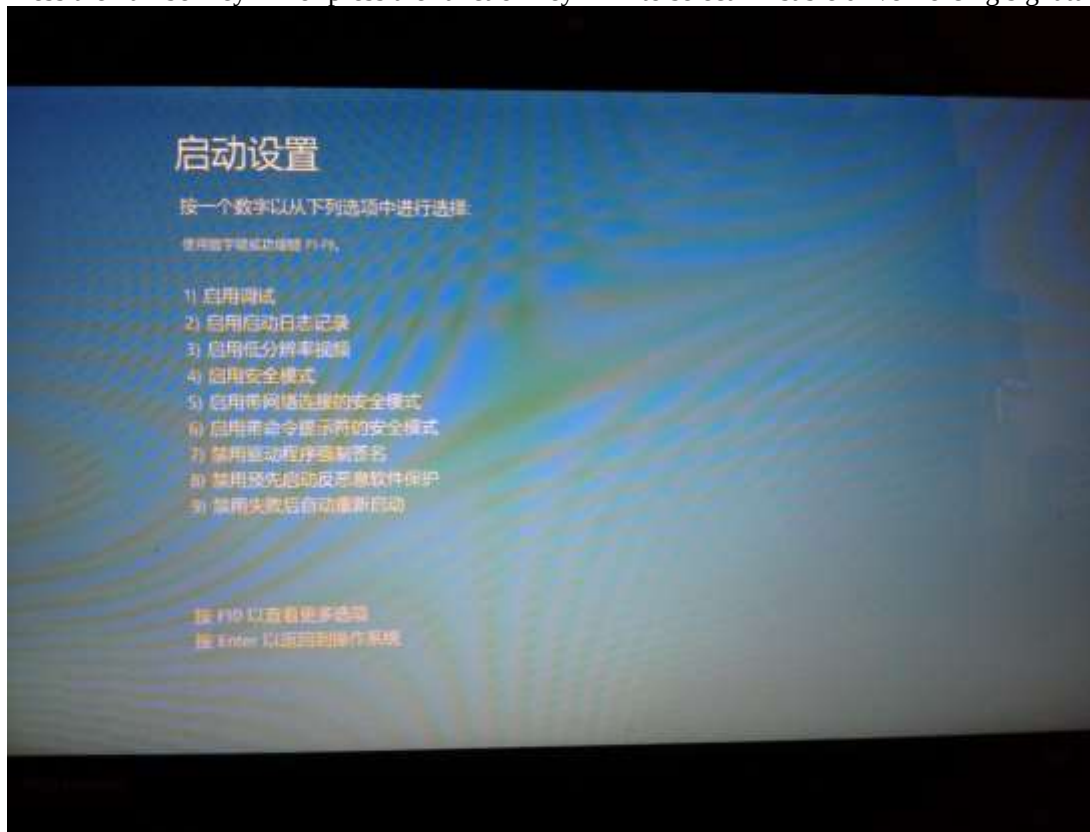
In the "Advanced Options" window below, select "Startup Settings".



In the "Startup Settings" window below, click the "Restart" button to restart the computer.



After the computer restarts, it will enter the "Startup Settings" window automatically as shown in the figure below. Press the number key "7" or press the function key "F7" to select "Disable driver forcing signature" to start.



After booting to Windows 8, follow the Windows 8 (32-bit) installation method to complete the driver installation.

Installation Instructions in Windows 10 (64-bit)

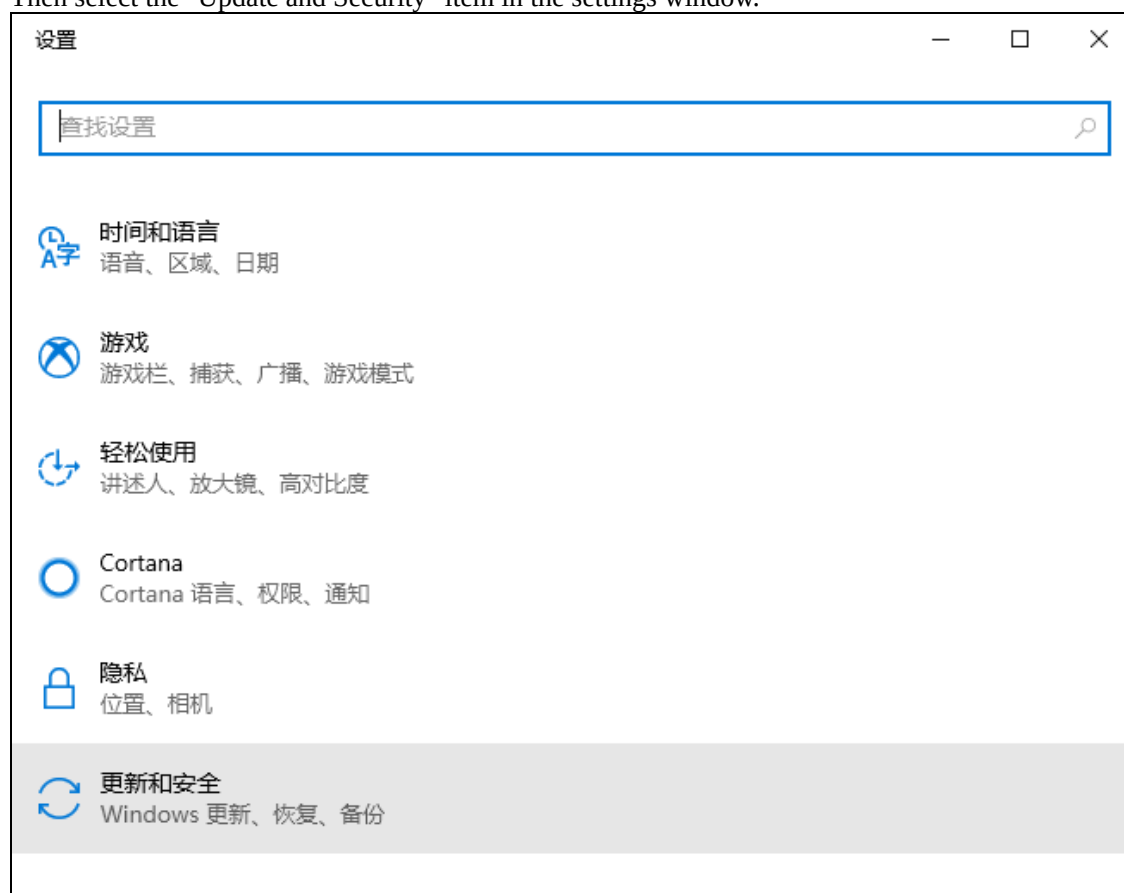
By default, the driver without digital signature cannot be successfully installed in Windows 10 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

Before installing the driver, you need to extract the "STC-USB Driver" folder to the hard disk from the STC-ISP download software package downloaded from the STC official website. Prepare the chip with USB download function, but don't connect the computer firstly.

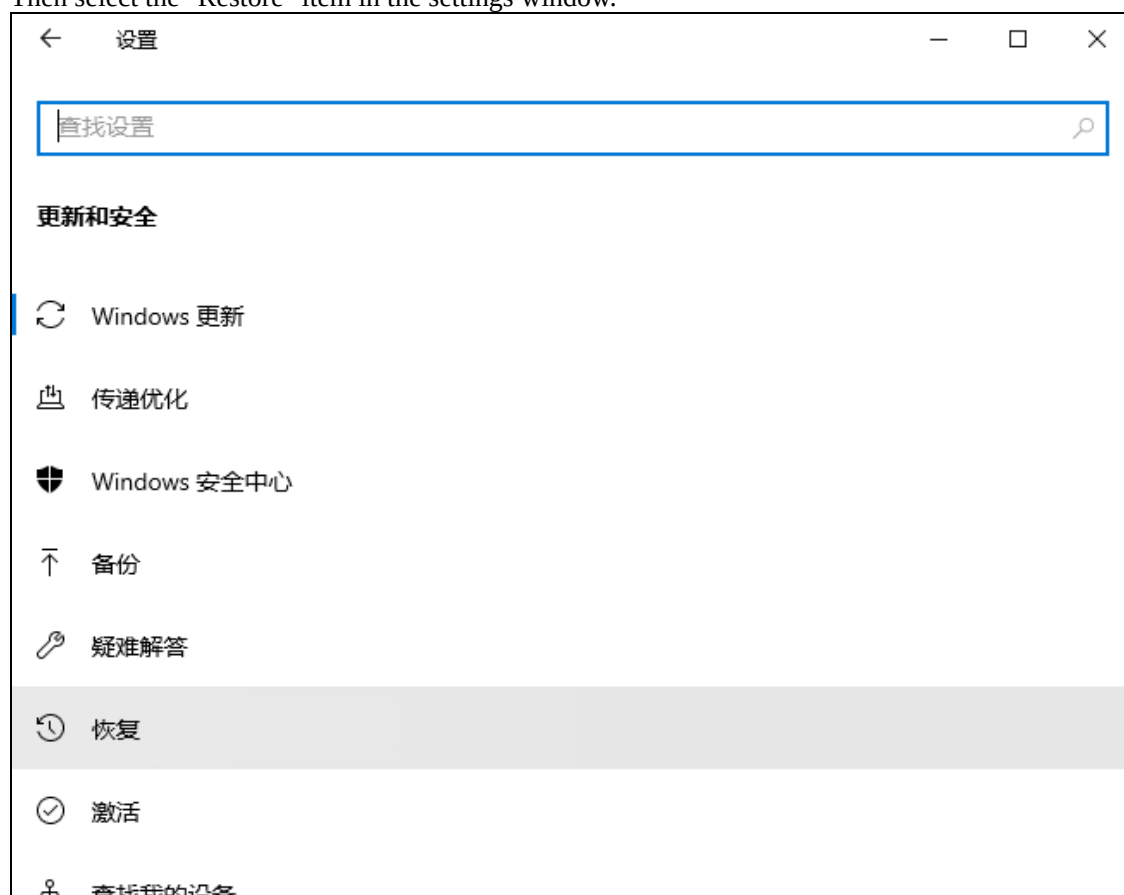
Right-click on the "Start" menu and select the "Settings" option.



Then select the "Update and Security" item in the settings window.



Then select the "Restore" item in the settings window.



In the recovery window, click the "Restart Now" button in the "Advanced Startup" item.



Before the computer restarts, the system will enter the following boot menu firstly and select the "Troubleshooting" item.



Select "Advanced Options" in the troubleshooting window.



Then select "View more recovery options".



Select the "Startup Settings" item.



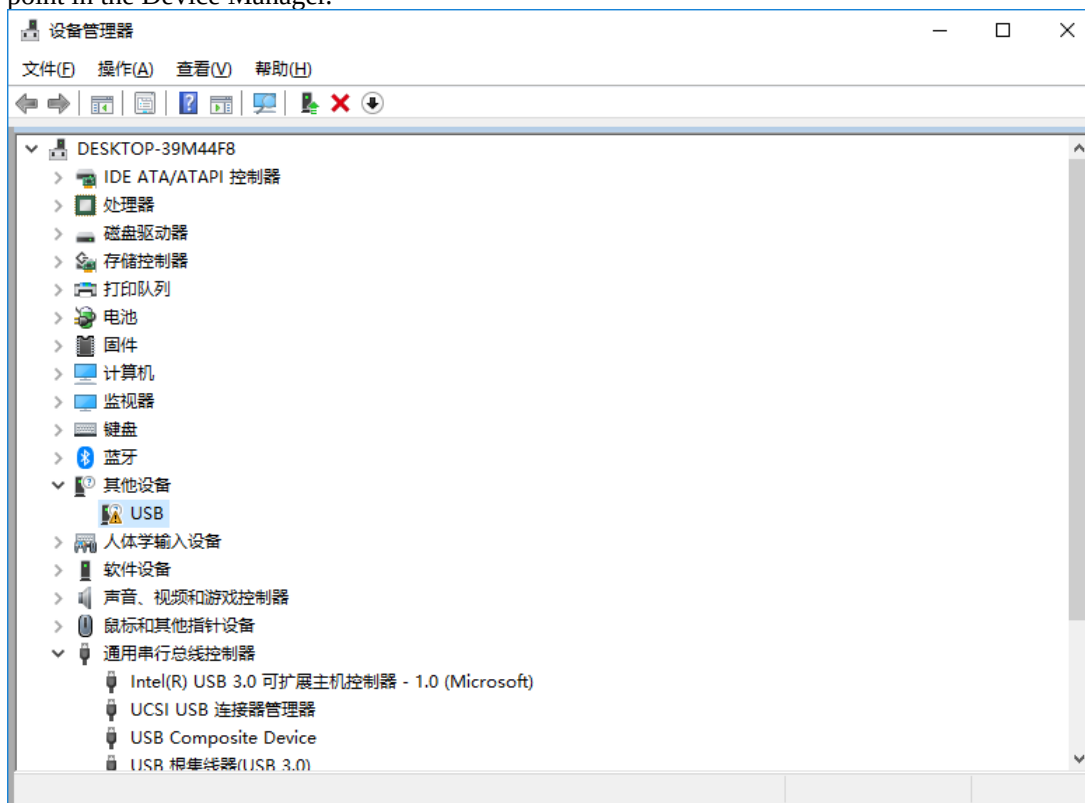
When the following screen appears, click the "Restart" button to restart the computer.



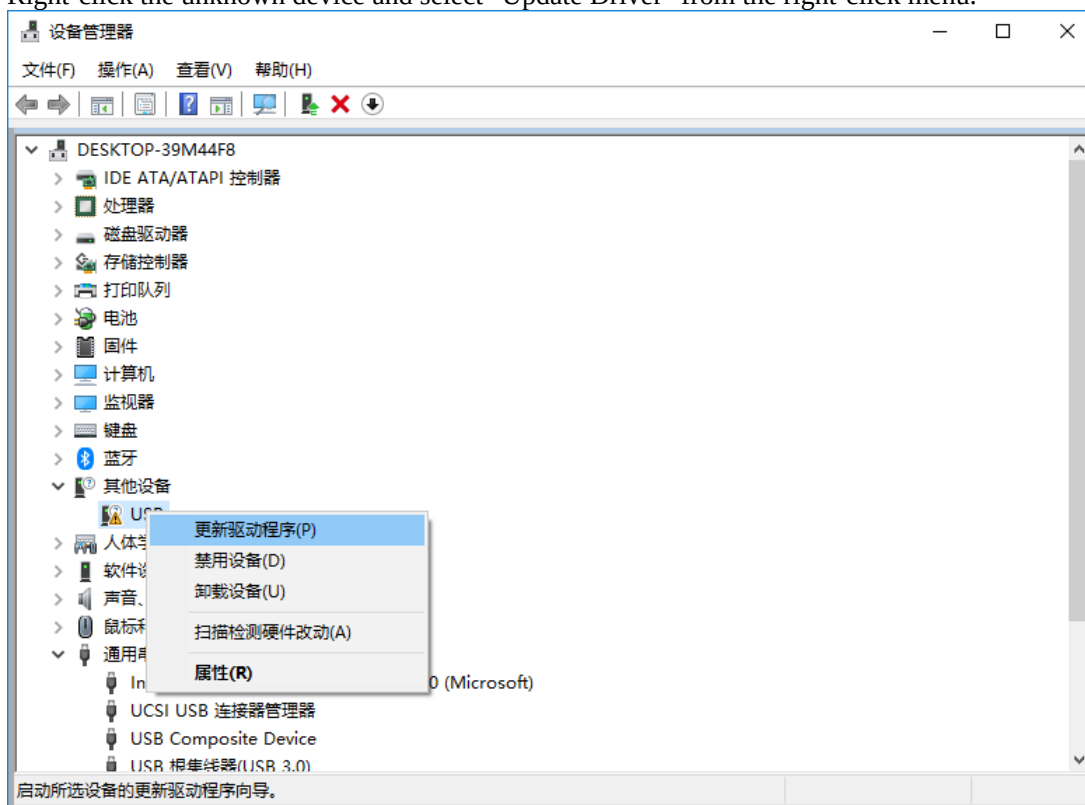
After the computer restarts, the "Startup Settings" window will pop up. Press the "F7" button to select the "Prohibit driver forcing signature" item.



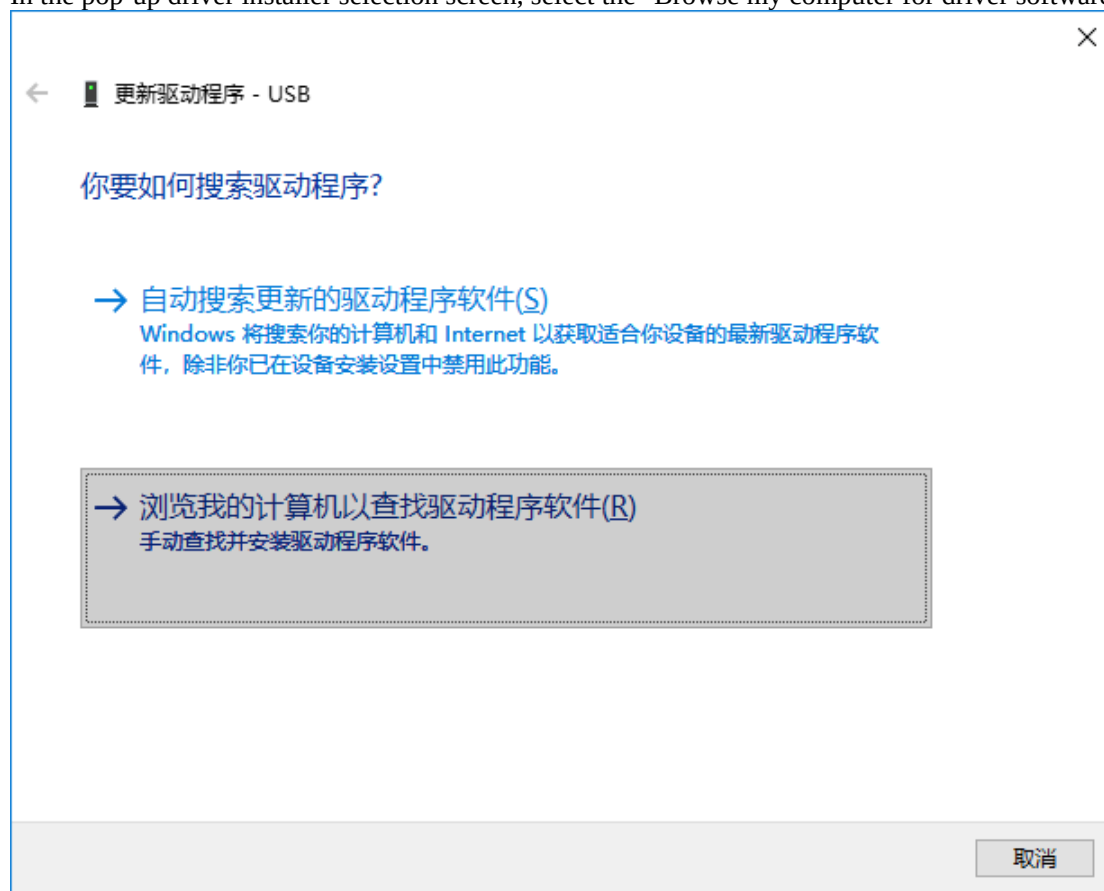
After the computer is started, connect the prepared chip to the computer with a USB cable and open the "Device Manager". Now, the driver has not yet been installed, so it will appear as an unknown device with an exclamation point in the Device Manager.



Right-click the unknown device and select "Update Driver" from the right-click menu.



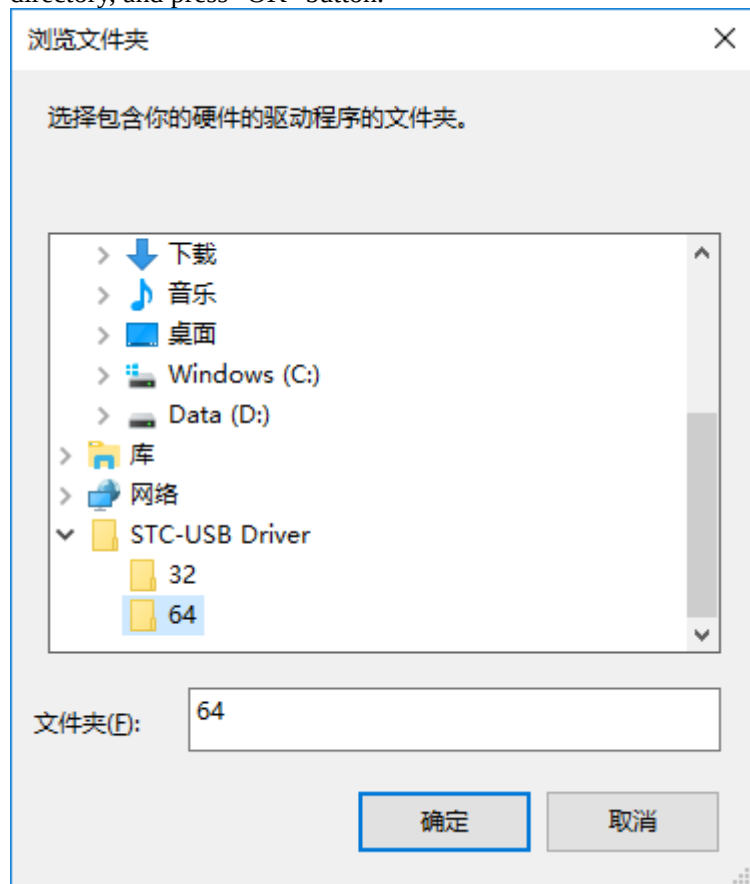
In the pop-up driver installer selection screen, select the "Browse my computer for driver software" item.



In the following window, click the "Browse" button.



Find the "STC-USB Driver" directory that was previously extracted to your hard disk, select the "64" directory in the directory, and press "OK" button.



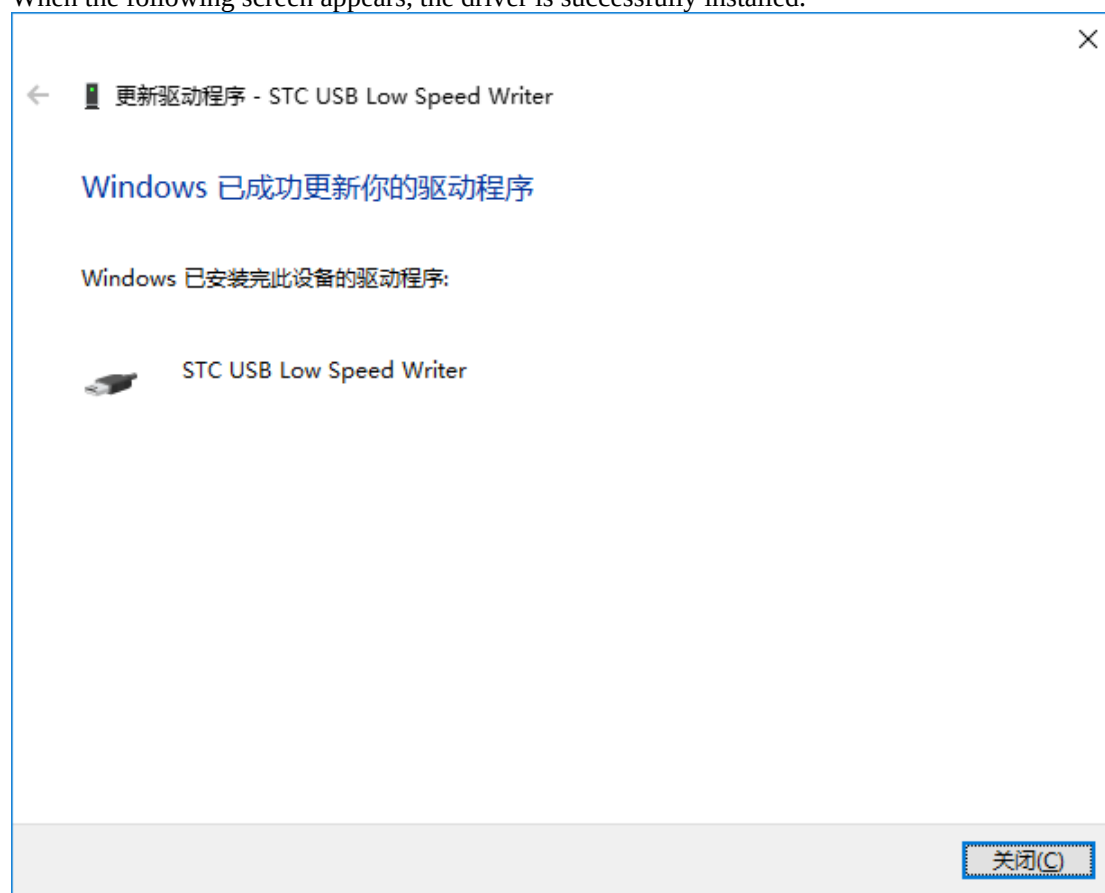
Click "Next" to start the driver installation.



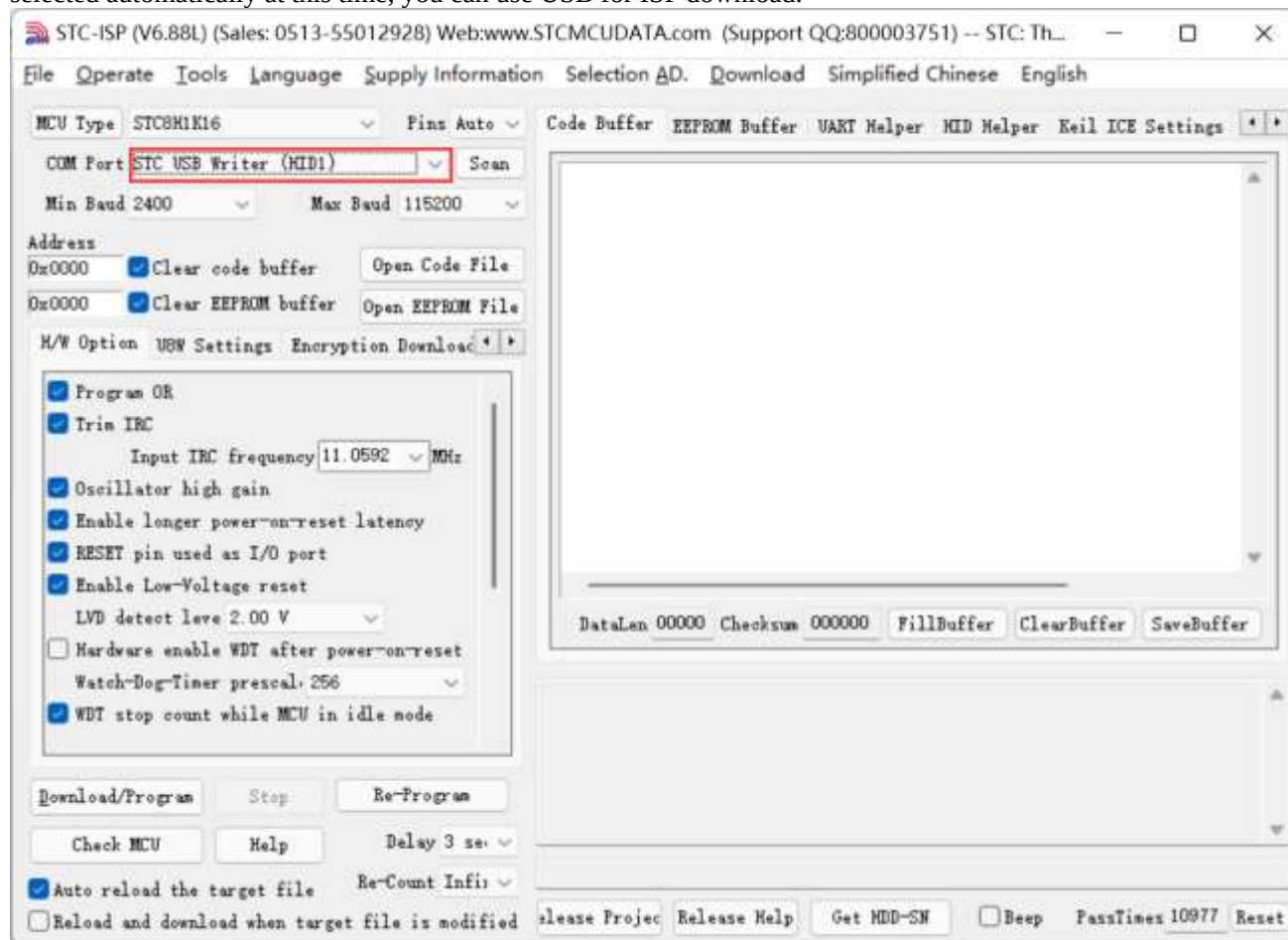
During the driver installation process, the following warning screen will pop up, select "Always install this driver software".



When the following screen appears, the driver is successfully installed.

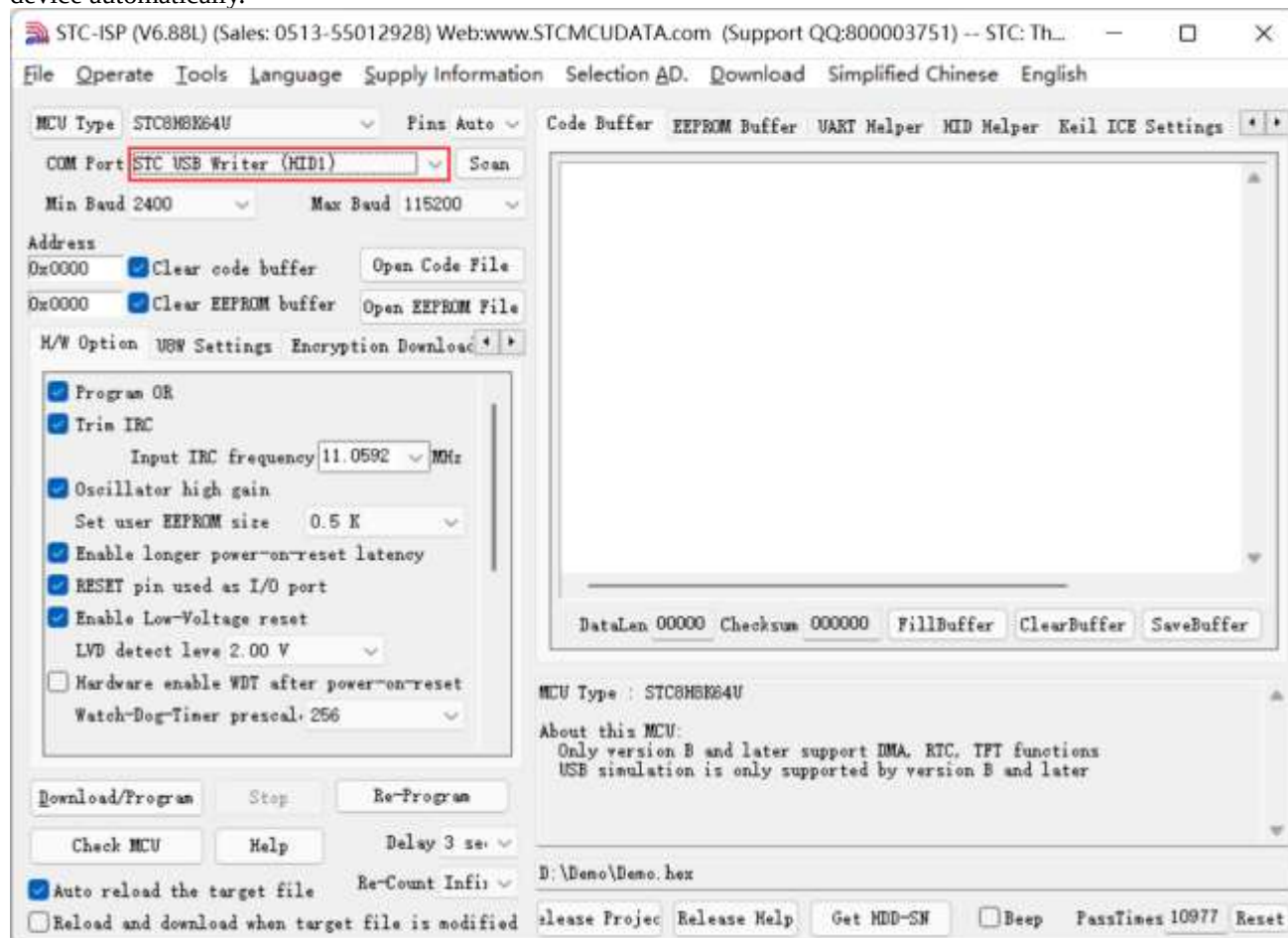


Back to the download software of STC-ISP, "STC USB Writer (HID1)" in the "Serial Port Number" drop-down list is selected automatically at this time, you can use USB for ISP download.

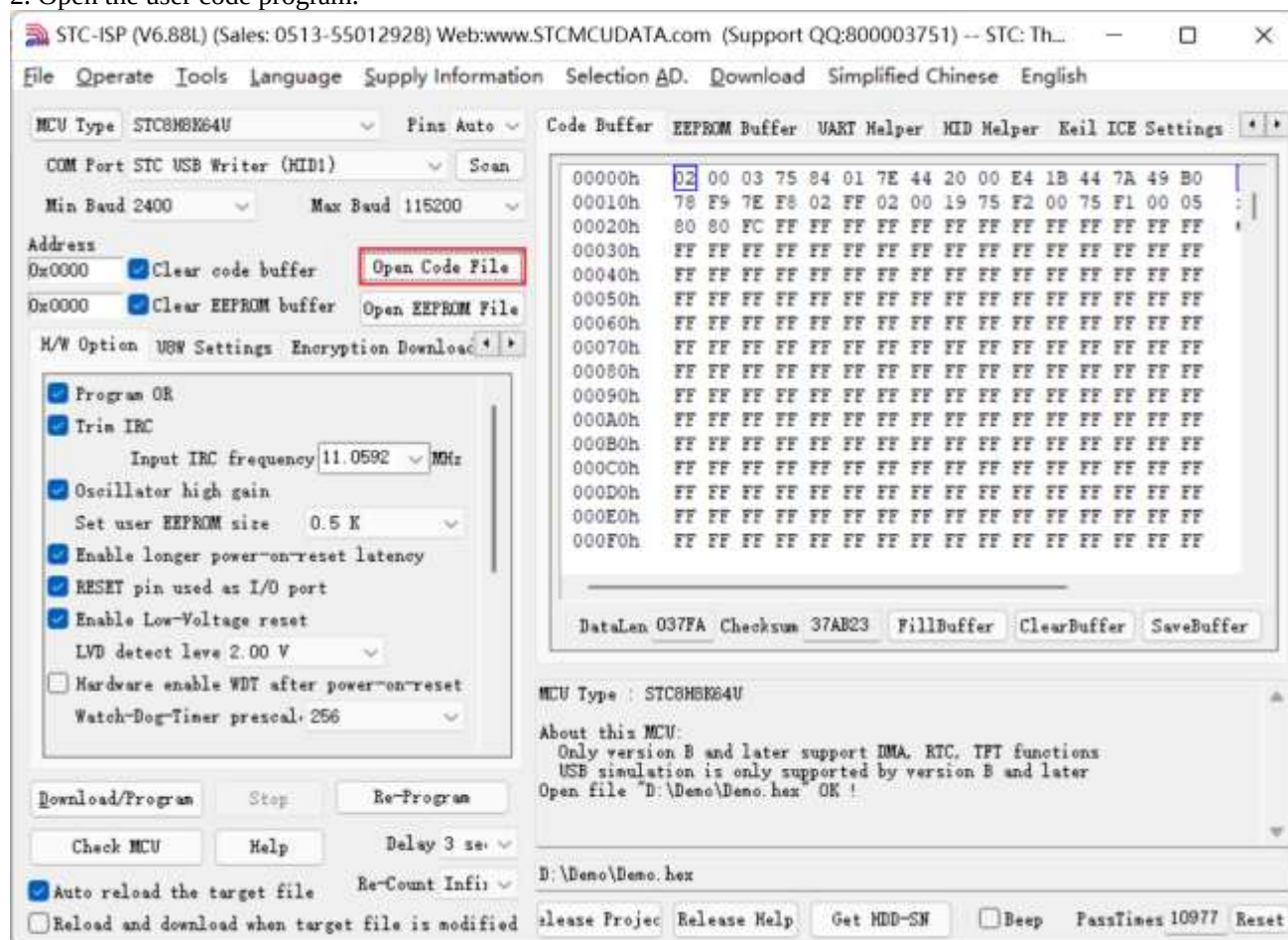


Appendix E Download Step Demo using USB

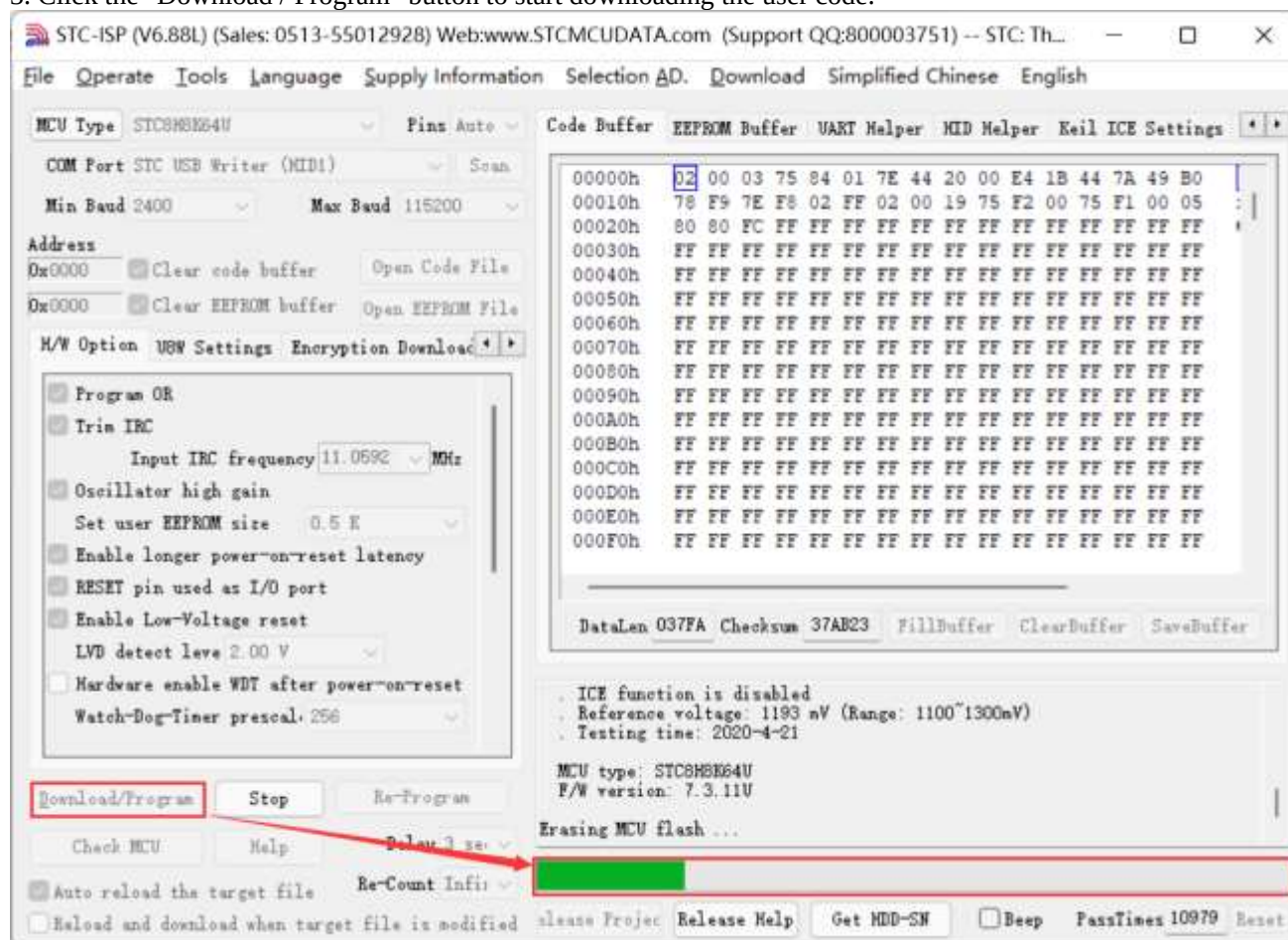
1. Refer to the application circuit diagram in P5.1.5 to connect the microcontroller firstly, and connect the P3.2 port of the target chip to GND, and then connect the system to the USB port on the PC side. Open the ISP to download the software, and the serial number of the downloaded software will search for the "STC USB Writer (HID1)" USB device automatically.



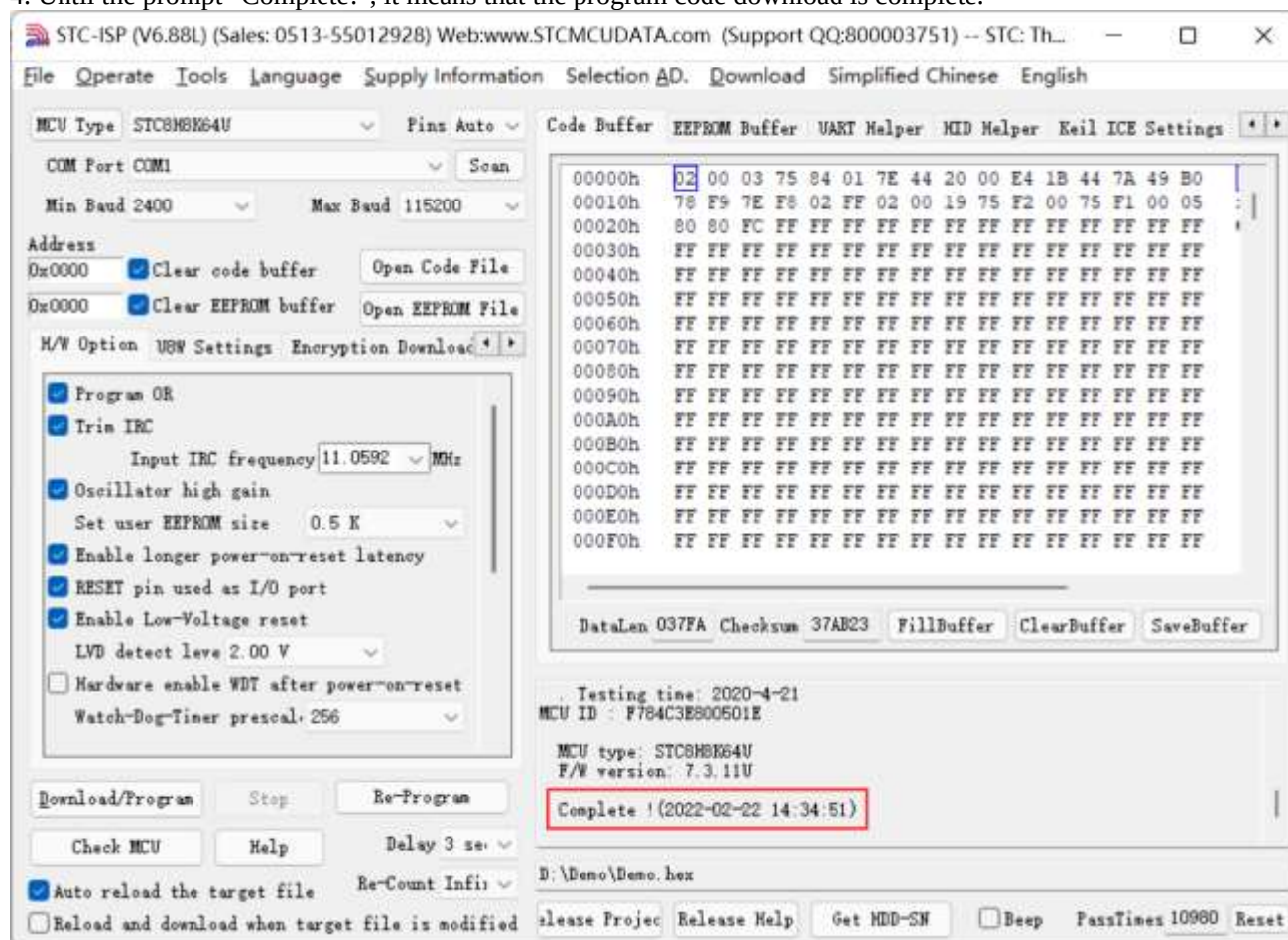
2. Open the user code program.



3. Click the "Download / Program" button to start downloading the user code.



4. Until the prompt "Complete!", it means that the program code download is complete.



Appendix F USB emulation step demonstration

All of the B version chips of the STC8H8K64U series and subsequent versions support the USB direct emulation function.

The specific operation steps are as follows:

1. Install the latest STC simulation driver in the Keil environment
Download the latest STC-ISP download software from the official website (www.STCMCUDATA.com), and follow the steps in Appendix A to install the STC emulation driver
2. Use serial ISP or USB ISP to make emulation chips

