```
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     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;                                    //P2.5~P2.0 Initialized to open-drain mode
        P2M1 = 0x3f;
        P2 = 0xff;
        CMPCR2 = 0x10;                                  //The comparator result has passed Output after a debounce clock
        CMPCR1 = 0x00;

        CMPCR1 &= ~0x08;                                //P3.7 for CMP+ The reference signal
        CMPCR1 &= ~0x04;                                //inside 1.19V source of the input pin is CMP- input pin
        CMPCR1 &= ~0x02;                                //Disable comparator output
        CMPCR1 |= 0x80;                                 //Enable comparator module

        while (1)
        {
                v = 0x00;                               //voltage <2.5V
                P2 = 0xfe;                              //P2.0output 0
                delay();
                if (! (CMPCR1 & 0x01)) goto ShowVol;
                v = 0x01;                               //voltage >2.5V
                P2 = 0xfd;                              //P2.1output 0
                delay();
                if (! (CMPCR1 & 0x01)) goto ShowVol;
                v = 0x03;                                   //voltage >3.0V
                P2 = 0xfb;                                  //P2.2output 0
                delay();
                if (! (CMPCR1 & 0x01)) goto ShowVol;
                v = 0x07;                                   //voltage >3.5V
                P2 = 0xf7;                                  //P2.3output 0
                delay();
                if (! (CMPCR1 & 0x01)) goto ShowVol;
                v = 0x0f;                                   //voltage >4.0V
                P2 = 0xef;                                  //P2.4output 0
                delay();
                if (! (CMPCR1 & 0x01)) goto ShowVol;
                v = 0x1f;                                   //voltage >4.5V
                P2 = 0xdf;                                  //P2.5output 0
                delay();
                if (! (CMPCR1 & 0x01)) goto ShowVol;
                v = 0x3f;                                   //voltage >5.0V
ShowVol:
P2 = 0xff;
P0 = ~v;
        }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

CMPCR1          DATA            0E6H
CMPCR2          DATA            0E7H

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

                ORG             0000H
                LJMP            MAIN

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

                MOV     P2M0,#00111111B         ;P2.5~P2.0      Initialized to open-drain mode
                MOV     P2M1,#00111111B
                MOV     P2,#0FFH
                MOV     CMPCR2,#10H             ;The comparator result has passed Output after a debounce clock
                MOV     CMPCR1,#00H
                ANL     CMPCR1,#NOT 08H         ;P3.7   for CMP+ The reference signal
                ANL     CMPCR1,#NOT 04H         ;inside 1.19V source of the input pin is CMP- input pin
                ANL     CMPCR1,#NOT 02H         ;Disable comparator
                ORL     CMPCR1,#80H             output ;Enable comparator
LOOP:                                           module ;voltage <2.5V
                MOV     R0,#00000000B
                MOV     P2,#11111110B           ;P2.0   output 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC. 0,SKIP
                MOV     R0,#00000001B           ;voltage >2.5V
                MOV     P2,#11111101B           ;P2.1output     0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC. 0,SKIP
                MOV     R0,#00000011B           ;voltage >3.0V
                MOV     P2,#11111011B             P2.2  output 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC. 0,SKIP
                MOV     R0,#00000111B           ;voltage >3.5V
                MOV     P2,#11110111B           ;P2.3output     0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC. 0,SKIP
                MOV     R0,#00001111B
                MOV     P2,#11101111B           ;voltage >4.0V
                CALL    DELAY                   ;P2.4output     0
                MOV     A,CMPCR1
                JNB     ACC. 0,SKIP
                MOV     R0,#00011111B
                MOV     P2,#11011111B           ;voltage >4.5V
                CALL    DELAY                   ;P2.5output     0
                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      Port display voltage
                JMP     LOOP

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

END

16 IAP/EEPROM/DATA-FLASH

STC12H The series of microcontrollers integrates a large-capacity EEPROM use ISP/IAP Technology can be internal DATA-FLASH when EEPROM, The number of erasures is More than ten thousand times. Can be divided into several sectors, each fan 512 byte The area contains EEPROM 10

attention : EEPROM The write operation can only transfer the bytes in 1 Write as, when you need to put the bytes Write as 0 , you must execute the sector Erase operation. EEPROM of reading/The write operation is performed in 1 The erase operation is based on the sector (512 bytes) 1 Carried out as a unit, and when the erase operation is performed, bytes, If there is data that needs to be retained in the target sector, You must read these data in advance to RAM Temporarily stored in, and then the saved data and the data that needs to be updated will be written back together after the erasure is comple

So in use EEPROM When, it is recommended that the data modified at the same time be placed in the same sector, not the data The same sector does not have to be full. The erasure operation of the data memory is performed by sector (per sector 512 Bytes)

EEPROM It can be used to save some parameter data that needs to be modified during the application process and is not lost after po correct EEPROM Perform byte reading/Byte programming/Sector erase operation. When the operating voltage is low, it is recommended not to EEPROM operation To avoid the loss of sending data.

16.1 EEPROM Operating time

1 4 Read bytes: a system clock (using MOVC Command reading is more convenient and fast)

Bytes: approximately programming 1 (The actual programming time is 6~7.5us , But you also need to add state transition time and various con Signal control and HOLD Time)

1 Erase sector (4 Bytes): Approximately ~ 6ms

EEPROM SETUP 512 The time required for operation is automatically controlled by the hardware, the user only needs to set it correctly IAP_TPS Register.

IAP_TPS =System operating frequency/1000000 (The decimal part is rounded for rounding)

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

Another example: The operating frequency of the system is 22.1184MHz, then IAP_TPS Set to 22

another example: The operating frequency of the system is 5.5296MHz, then IAP_TPS set to 6

16.2 EEPROM Related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP Data register	C2H									1111,1111
IAP_ADDRH	IAP High address register	C3H									0000,0000
IAP_ADDRL	IAP Low address register	C4H									0000,0000
IAP_CMD	IAP Command register	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 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								

In progress EEPROM During the read operation, the command is read out after EEPROM The data is stored IAP_DATA complete In the register.

EEPROM During the write operation, before executing the write command, the data to be written IAP_DATA must be stored in Register, and then Send a write command. EEPROM erase Command and IAP_DATA Register independent.

16.2.2 EEPROM Address register (IAP_ADDR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

EEPROM The destination address register for read, write, and erase operations. IAP_ADDRH Save the high byte of the address IAP_ADDRL , Save 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]: Send EEPROM Operation command

00: Empty operation

01: Read EEPROM command. Read the destination address 1 byte

10: write EEPROM command. Write the byte where 1 the destination address is located. Note: The write operation can only write the targ

Can't 0 0 Write as. Generally, when the target byte is not, it must be erased first. 1 FFH

11: Erase EEPROM 。 Erase the destination address 1 Page (sector 1 /512 Bytes). Note: The erase operation will erase 1 once

Sectors (512 Bytes), the contents of the entire sector all FFH become

16.2.4 EEPROM Trigger register (IAP_TRIG)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

To trigger the EEPROM register After reading, writing, and erasing the command register, address register, data register, and control register, yo after the setting is complete IAP_TRIG Write in turn 5AH, A5H (The order cannot be exchanged) Two trigger commands to trigger the correspondi Erase operation. After the operation is complete, address register IAP_ADDRH、 IAP_ADDRL and EEPROM Command register

EEPROM The content remains unchanged. If you want to operate on the data of the next address next, you need to manually update the a IAP_CMD And register IAP_ADDRL IAP_ADDRH The value of.

Note: every time EEPROM When operating, IAP_TRIG Write first 5AH, then write A5H , The corresponding command will take effect After writing the trigger command, you must be in the CPU IDLE Waiting state until the corresponding instruction. After the operation is completed CPU Will start from IDLE status CPU right position CPU

Return to normal state and continue execution

16.2.5 EEPROM Control register (IAP_CONTR)

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: Prohibited EEPROM Operation

1: Enable EEPROM operation

SWBS : The software resets to select the control bit, (it needs to be related to SWRST Used in conjunction)

0: After the software is reset, the program will be executed from the user code : After the software is reset, the program will be executed from the system monitoring code area. 1 ISP

SWRST: Software resets the control bit

0: No action

1: Generate software reset

CMD_FAIL : EEPROM The operation failed status bit needs to be cleared by the software

Correct operation 0 : EEPROM

1 : EEPROM Operation failed

16.2.6 EEPROM Waiting time control register (IAP_TPS)

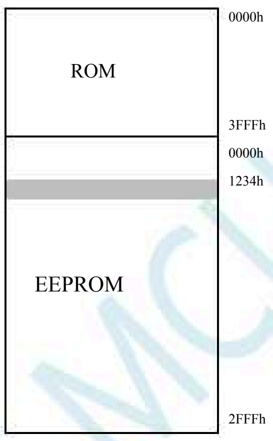
symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

It needs to be set according to the operating frequency , if the operating frequency is 12MHz, you need to IAP_TPS Set to 12 ; If the operating frequency is 24MHz, you need to IAP_TPS Set to 24 , frequency is other frequencies, and so on.

16.3 EEPROM Size and address

STC12H The series of microcontrollers are used to store user data inside. Operation method: read, yes EEPROM
Write and erase, where the erase operation is performed in sectors, each sector is 512 Bytes, that is, every time the erase command is executed, it w
Except for one sector, both reading data and writing data are operated in bytes, that is, only one byte can be read out or
written every time a read or write command is executed .

STC12H The internal read, write, and erase operations of EEPROM There are two ways to access : Way and MOVC way IAP The way is right
EEPROM the series of microcontrollers are performed, but IAP Can only be right MOVC Read operations are performed, but write and erase operations ca
Whether it is used IAP The way is still MOVC Way to access EEPROM , First of all, you need to set the correct destination address. IAP way
When the destination address to use EEPROM The actual physical addresses are the same. When they Start accessing, but to use 0000H MOVC
is read with a small offset from the EEPROM are all slave address data, the destination address must be EEPROM On the basis of the actual physical address, there is also a large pr
instruction. The following is STC12H1K16 Take this model as an example, and the destination address will be described in detail :



STC8H1K16

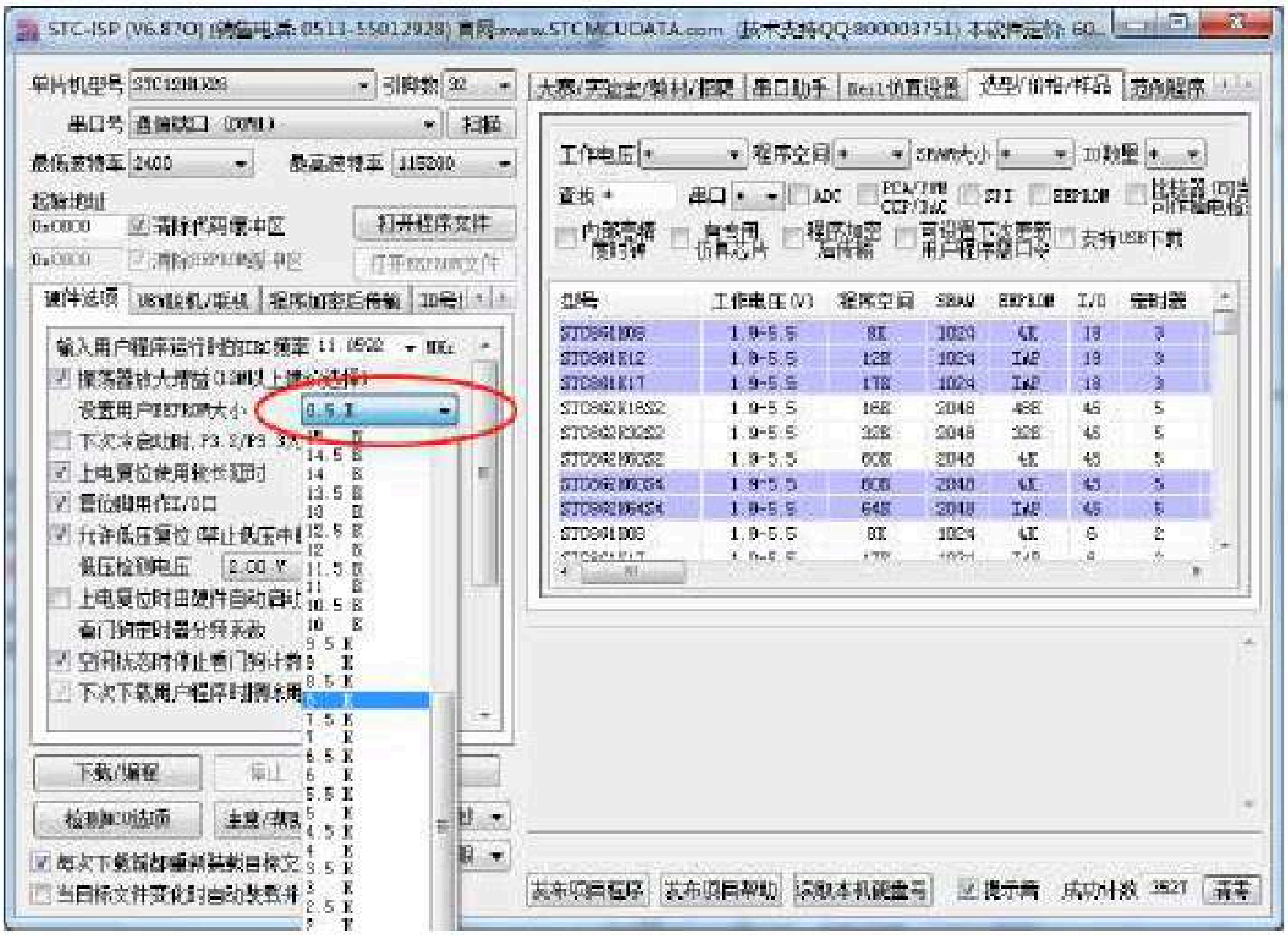
STC12H1K16 The program space is 16K Byte (0000h~3FFFh) , The space is 0000h~2FFFh). when EEPROM 12K (When the
Need to be right EEPROM the physical address When reading, writing, and erasing the unit, if IAP method is accessed, the purpose of the setting IAP
The target address is 1234H IAP_ADDRH that is you use the setting settings 12h , IAP_ADDRL 34H Then set the corresponding trigger command to 1234h
The unit has performed the correct operation. But if you use MOVC Way to read EEPROM Unit, you must be in the 1234h On the basis of 1234h
Plus the size of the space ROM 4000h, That is, it must be DPTR Set to 5234h, Before you can use MOVC The instruction is read.

Note: Due to the erasure method 512 The operation is performed in bytes, so the low bit of the destination address set when performing the era
It is meaningless. For example: when the erase command is executed, 1234H/1200H/1300H/13FFH , The final erasure action is
the byte is erased if the address is set to the same. 1200H~13FFH 512

Please refer to the table below EEPROM There will be differences in the size and access address, for each model EEPROM The detailed size and address of for the internals of different models

Model number	size	sector	IAP mode Read/write/erase start address	end address	MOVC reads the start address and the	end address
STC12H1K08	4K	8	0000h	0FFFh	2000h	6FFFh
STC12H1K16	12K	24	0000h	2FFFh	4000h	6FFFh
STC12H1K24	4K	8	0000h	0FFFh	6000h	6FFFh
STC12H1K28	user-defined[1]					
STC12H1K33	User-defined[1]					

[1] : This is a special model, this model EEPROM The size is available in ISP The user set it up by himself when downloading. As shown show :


STC-ISP (V6.87O) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60

单片机型号 STC12H1K28 引脚数 32

设置用户EEPROM大小 0.5 K

Users can use it according to their own FLASH No more than 100,000 square meters are planned in the space FLASH Size of EEPROM space space , needs throughout but need to pay attention : Always plan from the back to the front。

EEPROM

For example : STC12H1K28 The physical 28K , At this time, if the user wants to distinguish one of them 8K , then EEPROM

then EEPROM address of this model is the last 28K for FLASH , Of course, if the user uses IAP The way

To access, the destination address is still 0000h Start, to 8K' 1FFFh The physical address is End, when using MOVC 5000h ~ 6FFFh To read, you need to read from 5000h to start, to 6FFFh end.

16.4 Sample program

16.4.1 EEPROM Basic operation

C Language code

```
//The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
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 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;                  //close IAP function
        IAP_CMD = 0;                    //Clear command register
        IAP_TRIG = 0;                   //Clear trigger register
        IAP_ADDRH = 0x80;               //Set the address to non- IAP area
        IAP_ADDRL = 0;
}

char IapRead(int addr)
{
        char dat;

        IAP_CONTR = 0x80;               //Enable IAP
        IAP_TPS = 12;                   //Set waiting parameters 12MHz
        IAP_CMD = 1;                    //Set up IAP Read command
        IAP_ADDRL = addr;               //Set up IAP Low address
        IAP_ADDRH = addr >> 8;          //Set up 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();                      //close IAP function

        return dat;
}

void IapProgram(int addr, char dat)
{
        IAP_CONTR = 0x80;                       //Enable IAP
        IAP_TPS = 12;                           //Set waiting parameters 12MHz
        IAP_CMD = 2;                            //Set up IAP Write command
        IAP_ADDRL = addr;                       Low address//Set up IAP
        IAP_ADDRH = addr >> 8;                  High address//Set up IAP
        IAP_DATA = dat;                         data//write IAP
        IAP_TRIG = 0x5a;                        //Write trigger command (0x5a)
        IAP_TRIG = 0xa5;                        //Write trigger command (0xa5)
        _nop_();
        IapIdle();                              //close IAP function
}

void IapErase(int addr)
{
        IAP_CONTR = 0x80;                       //Enable IAP
        IAP_TPS = 12;                           //Set waiting parameters 12MHz
        IAP_CMD = 3;                            //Set up IAP Erase command
        IAP_ADDRL = addr;                       Low address//Set up IAP
        IAP_ADDRH = addr >> 8;                  High address//Set up IAP
        IAP_TRIG = 0x5a;                        //Write trigger command (0x5a)
        IAP_TRIG = 0xa5;                        //Write trigger command (0xa5)
        _nop_();                                //
        IapIdle();                              //close 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

```
;The test operating frequency 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

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

IAP_IDLE:
                MOV         IAP_CONTR,#0            ;close IAP function
                MOV         IAP_CMD,#0              ;Clear command register
                MOV         IAP_TRIG,#0             ;Clear trigger register
                MOV         IAP_ADDRH,#80H          ;Will addressSet to non-IAP area
                MOV         IAP_ADDRL,#0
                RET

IAP_READ:
                MOV         IAP_CONTR,#80H          ;Enable IAP
                MOV         IAP_TPS,#12             ;Set waiting parameters 12MHz
                MOV         IAP_CMD,#1              ;Set up IAP Read command
                MOV         IAP_ADDRL,DPL           ;Set up IAP Low address
                MOV         IAP_ADDRH,DPH           ;Set up 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                ;close IAP function
                RET

IAP_PROGRAM:
                MOV         IAP_CONTR,#80H          ;Enable IAP
                MOV         IAP_TPS,#12             ;Set waiting parameters 12MHz
                MOV         IAP_CMD,#2              ;Set up IAP Write command
                MOV         IAP_ADDRL,DPL           ;Set up IAP Low address
                MOV         IAP_ADDRH,DPH           ;Set up 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                ;close IAP function
                RET

IAP_ERASE:
                MOV         IAP_CONTR,#80H          ;Enable IAP
                MOV         IAP_TPS,#12              ;Set waiting parameters 12MHz
                MOV         IAP_CMD,#3               ;Set up IAP Erase command
                MOV         IAP_ADDRL,DPL            Low address;Set up IAP
                MOV         IAP_ADDRH,DPH            High address;Set up IAP
                MOV         IAP_TRIG,#5AH            ;Write trigger command (0x5a)
                MOV         IAP_TRIG,#0A5H           ;Write trigger command (0xa5)
                NOP
                LCALL       IAP_IDLE                 ;close 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 use MOVC read EEPROM

C Language code

```
// The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
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         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                         //STC12H1K16

void IapIdle()
{
        IAP_CONTR = 0;                                              //close IAP function
        IAP_CMD = 0;                                                //Clear command register
        IAP_TRIG = 0;                                               //Clear trigger register
        IAP_ADDRH = 0x80;                                           //Set the address to non- IAP area
        IAP_ADDRL = 0;
}

char IapRead(int addr)
{
        addr += IAP_OFFSET;                                         //use MOVC read EEPROM Need to add the corresponding offset
        return *(char code *)(addr);                                //use MOVC Read data
}

void IapProgram(int addr, char dat)
{
        IAP_CONTR = 0x80;                                           //Enable IAP
        IAP_TPS = 12;                                               //Set waiting parameters 12MHz
        IAP_CMD = 2;                                                //Set up IAP Write command
        IAP_ADDRL = addr;                                           //Set up IAP Low address
        IAP_ADDRH = addr >> 8;                                      //Set up 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();                                                  //close IAP function
}

void IapErase(int addr)
{
        IAP_CONTR = 0x80;                                           //Enable IAP
        IAP_TPS = 12;                                               //Set waiting parameters 12MHz
        IAP_CMD = 3;                                                //Set up IAP Erase command
        IAP_ADDRL = addr;                                           //Set up IAP Low address
        IAP_ADDRH = addr >> 8;                                      //Set up IAP High address
        IAP_TRIG = 0x5a;                                            //Write trigger command (0x5a)
        IAP_TRIG = 0xa5;                                            //Write trigger command (0xa5)
        _nop_();                                                    //
        IapIdle();                                                  //close 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

```
;The test operating frequency 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                   ;STC12H1K16

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

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

```
            MOV         IAP_ADDRL,#0
            RET

IAP_READ:
            MOV         A,#LOW IAP_OFFSET       ;use MOVC read EEPROM Need 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               ;use MOVC Read data
            RET

IAP_PROGRAM:
            MOV         IAP_CONTR,#80H          ;Enable IAP
            MOV         IAP_TPS,#12             ;Set waiting parameters 12MHz
            MOV         IAP_CMD,#2              ;Set up IAP Write command
            MOV         IAP_ADDRL,DPL           ;Set up IAP Low address
            MOV         IAP_ADDRH,DPH           ;Set up 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                ;close IAP function
            RET

IAP_ERASE:
            MOV         IAP_CONTR,#80H          ;Enable IAP
            MOV         IAP_TPS,#12             ;Set waiting parameters 12MHz
            MOV         IAP_CMD,#3              ;Set up IAP Erase command
            MOV         IAP_ADDRL,DPL           ;Set up IAP Low address
            MOV         IAP_ADDRH,DPH           ;Set up IAP High address
            MOV         IAP_TRIG,#5AH           ;Write trigger command (0x5a)
            MOV         IAP_TRIG,#0A5H          ;Write trigger command (0xa5)
            NOP
            LCALL       IAP_IDLE                ;close 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 Use serial port to send out EEPROM data

C Language code

```
//The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
#define     FOSC            11059200UL
#define BRT                 (65536 - FOSC / 115200 / 4)
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 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;                          //close IAP function
        IAP_CMD = 0;                            //Clear command register
        IAP_TRIG = 0;                           //Clear trigger register
        IAP_ADDRH = 0x80;                       //Set the address to non-IAP area
        IAP_ADDRL = 0;
}

char IapRead(int addr)
{
        char dat;

        IAP_CONTR = 0x80;                       //Enable IAP
        IAP_TPS = 12;                           //Set waiting parameters 12MHz
        IAP_CMD = 1;                            //Set up IAP Read command
        IAP_ADDRL = addr;                       //Set up IAP Low address
        IAP_ADDRH = addr >> 8;                  //Set up 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();                              //close IAP function

        return dat;
}

void IapProgram(int addr, char dat)
{
        IAP_CONTR = 0x80;                       //Enable IAP
        IAP_TPS = 12;                           //Set waiting parameters 12MHz
        IAP_CMD = 2;                            //Set up IAP Write command
        IAP_ADDRL = addr;                       //Set up IAP Low address
        IAP_ADDRH = addr >> 8;                  //Set up 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();                              //close IAP function
}

void IapErase(int addr)
{
        IAP_CONTR = 0x80;                       //Enable IAP
        IAP_TPS = 12;                           //Set waiting parameters 12MHz
        IAP_CMD = 3;                            //Set up IAP Erase command
        IAP_ADDRL = addr;                       //Set up IAP Low address
        IAP_ADDRH = addr >> 8;                  //Set up IAP High address
        IAP_TRIG = 0x5a;                        //Write trigger command (0x5a)
        IAP_TRIG = 0xa5;                        //Write trigger command (0xa5)
        _nop_();                                //
        IapIdle();                              //close 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

```
; The test operating frequency 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

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
```

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

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

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

IAP_READ:
            MOV     IAP_CONTR,#80H          ;Enable IAP
            MOV     IAP_TPS,#12             ;Set waiting parameters 12MHz
            MOV     IAP_CMD,#1              ;Set up IAP Read command
            MOV     IAP_ADDRL,DPL           ;Set up IAP Low address
            MOV     IAP_ADDRH,DPH           ;Set up 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                ;close IAP function
            RET

IAP_PROGRAM:
            MOV     IAP_CONTR,#80H          ;Enable IAP
            MOV     IAP_TPS,#12             ;Set waiting parameters 12MHz
            MOV     IAP_CMD,#2              ;Set up IAP Write command
            MOV     IAP_ADDRL,DPL           ;Set up IAP Low address
            MOV     IAP_ADDRH,DPH           ;Set up 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                ;close IAP function
            RET

IAP_ERASE:
            MOV     IAP_CONTR,#80H          ;Enable IAP
            MOV     IAP_TPS,#12             ;Set waiting parameters 12MHz
            MOV     IAP_CMD,#3              ;Set up IAP Erase command
            MOV     IAP_ADDRL,DPL           ;Set up IAP Low address
            MOV     IAP_ADDRH,DPH           ;Set up IAP High address
            MOV     IAP_TRIG,#5AH           ;Write trigger command (0x5a)
            MOV     IAP_TRIG,#0A5H          ;Write trigger command (0xa5)
            NOP
            LCALL   IAP_IDLE                ;close 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
```

17 ADC Analog-to-digital conversion, internal 1.19V Reference signal source

STC12H1K08 A series of microcontrollers are integrated internally 10 Bit high speed A/D converter. ADC The clock frequency is divided into the system fre The frequency is then divided again by the frequency division coefficient set by the user ADC The operating clock frequency range is SYSclk/2/1 to SYSclk/2/16.

STC12H Series of ADC Fastest speed : 10 bit ADC for 500K (Every second 50 Ten thousand times ADC Conversion)

ADC There are two data formats for the conversion result: left-aligned and right-aligned. It can be easily read and referenced by user p

attention : ADC The first 15 The channel can only be used to detect the internal reference signal source, and the 1.19V Due to the value of the reference sig Manufacturing errors and measurement errors cause the actual internal reference signal source to be compared 1.19V about 1% error. If the user needs to know Know the accurate internal reference signal source value of each chip, you can ADC The first 15 Channel measurement connect an external accurate reference signal source, and then use the standardSet.

17.1 ADC Related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	Control register ADC	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC Conversion result, high register	BDH									0000,0000
ADC_RESL	ADC , conversion result, low register	BEH									0000,0000
ADCCFG	ADC Configuration register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
ADCTIM	ADC Timing control register	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010

17.1.1 ADC Control register (ADC_CONTR) , PWM trigger ADC control

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

ADC_POWER : ADC Power control bit

0: Closed ADC power supply

1: Open ADC power supply.

It is recommended to enter the idle mode and power-down mode before ADC Power off to reduce power consumption

Pay special attention :

MCU 1, Give internal ADC After the module power is turned on, you need to wait for about ADC Stable power supply it and then let it go ADC work ;

2, Appropriately lengthen the sampling time of the external ADC The charging or discharging time of the internal sample-and-hold capac signal, that is, the internal potential is equal to the external potential.

ADC_START : ADC Convert the start control bit. Start after writing Conversion, after the conversion is complete, the hardware automatica

0: No effect. The conversion work has already started, write the It will not stop the conversion. A/D

Even if 1: Start ADC conversion ADC0, The hardware automatically clears this bit to zero after the conversion is complete.

ADC_FLAG : ADC Conversion end flag. when ADC After completing a conversion, the hardware will automatically transfer this location to propose Interrupt request. This flag must be cleared by software.

ADC_EPWMT : Enable PWM Real-time trigger ADC function. For details, please refer to Bit advanced PWM Timer chapter

ADC_CHS[3:0] : ADC Analog channel selection bit

(Note: Selected as ADC Input channel I/O Port, must be set PxM0/PxM1 The register will If the port mode is set to high Resistance input mode. In addition, if MCU Enter power-down mode/After the clock stops vibration mode, if still needs to be enabled ADC channel, you need to set PxIE (Turn off the digital input channel to prevent the external analog input signal from rising or falling and generate additional power c

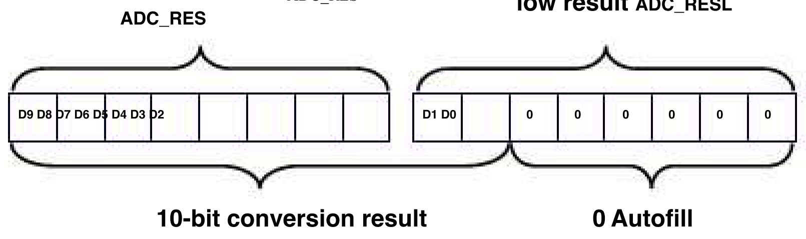
ADC_CHS[3:0]	channel ADC
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P1.2/ADC2
0011	P1.3/ADC3
0100	P1.4/ADC4
0101	P1.5/ADC5
0110	P1.6/ADC6
0111	P1.7/ADC7
1000	P2.0/ADC8
1001	P2.1/ADC9
1010	P2.4/ADC10
1011	P2.5/ADC11
1100	P2.6/ADC12
1101	P2.7/ADC13
1110	P3.7/ADC14
1111	Inside the test 1.19V

17.1.2 ADC Configuration register (ADCCFG)

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

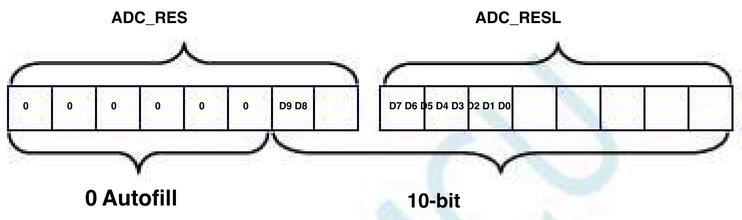
ADC RESFMT: Conversion result format control bit

0 : The conversion result is aligned to the left. ADC_RES Save the high position of the result 8 , ADC_RESL Save the low result 2 bit. The format is as follows :



RESFMT=0

1 :The conversion result is aligned to the right. ADC_RES Save the high result 2 bit , ADC_RESL Save the low result 8 bit. The format is as follows :



RESFMT=1

SPEED[3:0]: Set up ADC Operating clock frequency {F_{ADC} = SYSclk/2/(SPEED+1)}

SPEED[3:0]	give ADC The operating clock frequency ADC
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 Conversion result register (ADC_RES , ADC_RESL)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

Please refer to the A/D The bit conversion result will be automatically saved after the conversion is complete ADC_RES and ADC_RESL in. Save the data grid of the result current formula ADC_CFG 10 Settings in the register. RESFMT

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	Occupy ADC Number of working clocks
0	1(Default value)
1	2

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

CSHOLD[1:0]	Occupy ADC Number of working clocks
00	1
01	2(Default value)
10	3
11	4

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

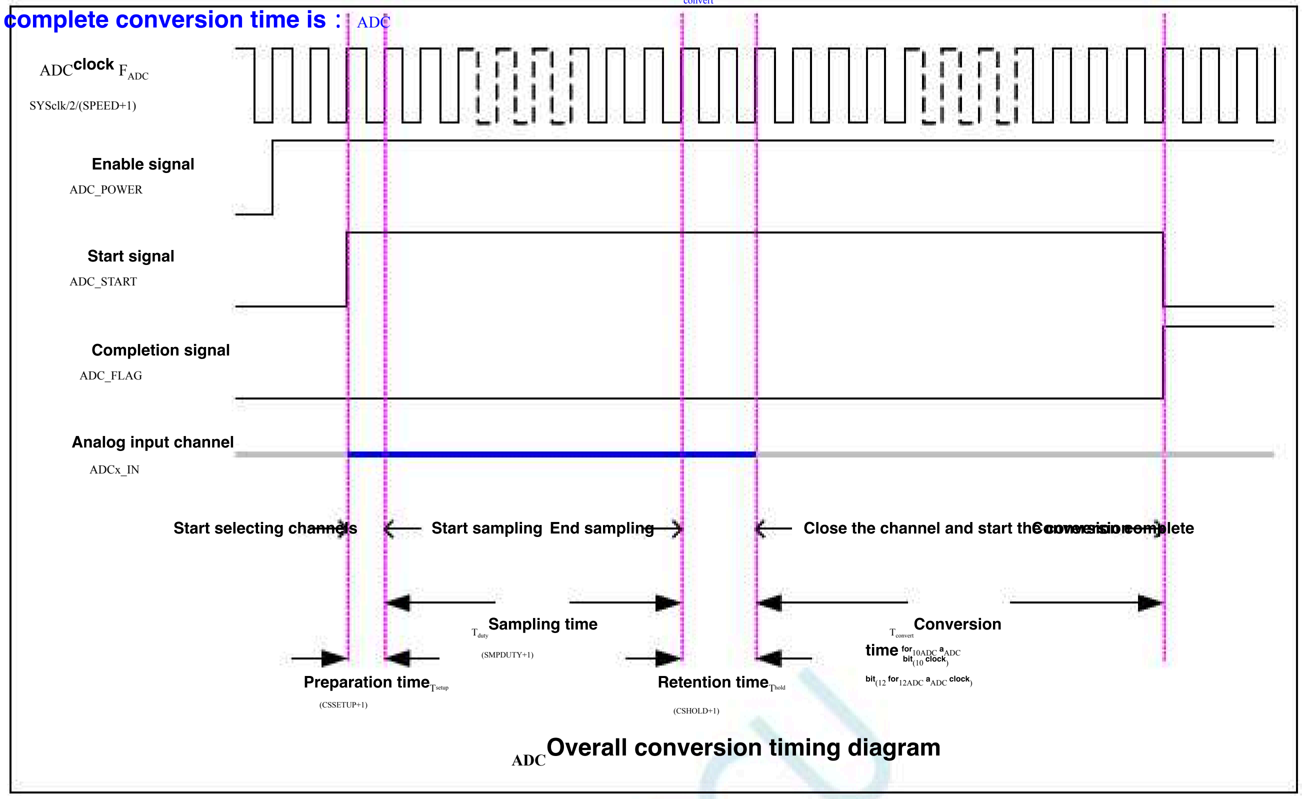
SMPDUTY[4:0]	Occupy ADC Number of working clocks
00000	1
00001	2
...	...
01010	11(Default value)
...	...
11110	31
11111	32

ADC $T_{convert}$ Digital-to-analog conversion time：

The conversion time is fixed as 10 Working clock ADC

bit 10 $T_{setup} + T_{duty} + T_{hold} + T_{convert}$, As shown in the figure below

ADC A complete conversion time is： ADC



ADC Overall conversion timing diagram

17.2 ADC Related calculation formula

Speed calculation formula 17.2.1

ADC ADC The conversion speed is determined by ADCCFG in the register SPEED and ADCTIM The registers are jointly controlled. The calculation formula for the As shown below :

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

attention :

bit 10 The speed cannot be higher than ADC 500KHz

SMPDUTY The value cannot be less than the 10, It is recommended to set to 15

CSSETUP default value that can be used for power-up 0

CHOLD You can use the default value for power-up (ADCTIM It is recommended to set to 3FH)

17.2.2 ADC Conversion result calculation formula

$$10\text{bit ADC Conversion result} = 1024 \times \frac{\text{ADC Input voltage of the converted channel } V_{in}}{\text{MCU Operating voltage } V_{cc}}$$

17.2.3 Push back ADC Input voltage calculation formula

$$\text{ADC Input voltage of the converted channel } V_{in} = \text{MCU Operating voltage } V_{cc} \times \frac{10\text{bit ADC Conversion result}}{1024}$$

17.2.4 Pushback working voltage calculation formula

$$\text{MCU Operating voltage } V_{cc} = 1024 \times \frac{\text{ADC Input voltage of the converted channel } V_{in}}{10\text{bit ADC Conversion result}}$$

17.3 10 bit ADC Static characteristics

symbol	description	Minimum value	Typical value	Maximum value	unit
RES	Resolution	-	10	-	Bits
E_T	overall error offset	-	1.3	3	LSB
E_O	error gain error differential	-	0.3	1	LSB
E_G	non-linearity error	-	0	1	LSB
E_D	integral non-linearity	-	0.7	1.5	LSB
E_I	error channel	-	1	2	LSB
R_{AIN}	equivalent resistance	-	∞	-	ohm
R_{ESD}	Antistatic resistor connected in series in front of the	-	700	-	ohm
C_{ADC}	sample-and-hold capacitor, internal sample-and-hold capacitor	-	16.5	-	pF

17.4 Sample program

17.4.1 ADC Basic operation (query method)

C Language code

```
//The test operating frequency 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 cha volatile xdata *)0xfea8)
sfr P1M1             =    0x91;
sfr P1M0             =    0x92;
sfr P0M1             =    0x93;
sfr P0M0             =    0x94;
sfr P2M1             =    0x95;
sfr P2M0             =    0x96;
sfr P3M1             =    0xb1;
sfr P3M0             =    0xb2;
sfr P4M1             =    0xb3;
sfr P4M0             =    0xb4;
sfr P5M1             =    0xc9;
sfr P5M0             =    0xca;

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;                                    //Set up P1.0 for ADC mouth
        P_SW2 |= 0x80;
        ADCTIM = 0x3f;                                  //Set up ADC Internal timing
        P_SW2 &= 0x7f;
        ADCCFG = 0x0f;                                  //Set up ADC The clock is the system clock/2/16
        ADC_CONTR = 0x80;                               //Enable ADC module

        while (1)
{
        ADC_CONTR |= 0x40;                          //Start AD convert
        _nop_();
        _nop_();
        while (! (ADC_CONTR & 0x20));               //query ADC Completion mark
        ADC_CONTR &= ~0x20;                         //Clearance completion mark
        P2 = ADC_RES;                               //read ADC result
    }
}
```

Assembly code

```
;The test operating frequency 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

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

                ORG         0000H
                LJMP        MAIN

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

                MOV         P1M0,#00H                   ;Set up P1.0 for ADC mouth
                MOV         P1M1,#01H
                MOV         P_SW2,#80H
                MOV         DPTR,#ADCTIM                ;Set up ADC Internal timing
                MOV         A,#3FH
                MOVX        @DPTR,A
```

```
            MOV     P_SW2,#00H
            MOV     ADCCFG,#0FH             ;Set up ADC The clock is the system clock/2/16
            MOV     ADC_CONTR,#80H          ;Enable ADC module
LOOP:
            ORL     ADC_CONTR,#40H          ;Start AD convert
            NOP
            NOP
            MOV     A,ADC_CONTR             ;query ADC Completion mark
            JNB     ACC. 5,$-2
            ANL     ADC_CONTR,#NOT 20H      ;Clearance completion mark
            MOV     P2,ADC_RES              ;read ADC result

            SJMP    LOOP

            END
```

17.4.2 ADC Basic operation (interrupt mode)

C Language code

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

void ADC_Isr() interrupt 5
{
        ADC_CONTR &= ~0x20;                     //Clear interrupt sign
        P2 = ADC_RES;                           //read ADC result
        ADC_CONTR |= 0x40;                      //continue AD convert
}

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;                              //Set up P1.0 for ADC mouth
        P_SW2 |= 0x80;
        ADCTIM = 0x3f;                            //Set up ADC Internal timing
        P_SW2 &= 0x7f;
        ADCCFG = 0x0f;                            //Set up The clock is the system clock/2/16

        ADC_CONTR = 0x80;                         //Enable ADC moduleADC
        EADC = 1;                                 //Enable ADC interrupt
        EA = 1;
        ADC_CONTR |= 0x40;                        //Start AD convert

        while (1);
}
```

Assembly code

```
; The test operating frequency 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

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

                ORG         0000H
                LJMP        MAIN
                ORG         002BH
                LJMP        ADCISR
                ORG             0100H
ADCISR:
                ANL             ADC_CONTR,#NOT 20H          ;Clearance completion mark
                MOV             P2,ADC_RES                  ;read ADC result
                ORL             ADC_CONTR,#40H              ;continue AD convert
                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 up P1.0 for ADC mouth
                MOV             P1M1,#01H
                MOV             P_SW2,#80H
                MOV             DPTR,#ADCTIM                ;Set up ADC Internal timing
                MOV             A,#3FH
                MOVX            @DPTR,A
                MOV             P_SW2,#00H
                MOV             ADCCFG,#0FH                 ;Set up ADC The clock is 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 convert

                SJMP            $

                END
```

17.4.3 format ADC Conversion result

C Language code

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

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x01;                            //Set up P1.0 for ADC mouth
        P_SW2 |= 0x80;
        ADCTIM = 0x3f;                          //Set up ADC Internal timing
        P_SW2 &= 0x7f;
        ADCCFG = 0x0f;                          //Set up The clock is the system clock/2/16

        ADC_CONTR = 0x80;                       //Enable ADC module
        ADC_CONTR |= 0x40;                      //Start AD convert
        _nop_();
        _nop_();
        while (! (ADC_CONTR & 0x20));           //query ADC Completion mark
        ADC_CONTR &= ~0x20;                     //Clearance completion mark

        ADCCFG = 0x00;                          //Set the result to the left to align
        ACC = ADC_RES;                          //A storage ADC of 10 The high position of the bit result 8
        B = ADC_RESL;                           //B[7:6]storage ADC of 10 The low position of the bit result 2

//      ADCCFG = 0x20;                          //Set the result to the right
//      ACC = ADC_RES;                          //A[1:0]storage ADC of 10 The high position of the bit result 2
//      B = ADC_RESL;                           //B storage ADC of 10 The low position of the bit result 8

        while (1);
}
```

Assembly code

```
;The test operating frequency 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

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

                ORG         0000H
                LJMP        MAIN

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

                MOV         P1M0,#00H                   ;Set up P1.0 for ADC mouth
                MOV         P1M1,#01H
                MOV         P_SW2,#80H
                MOV         DPTR,#ADCTIM                ;Set up ADC Internal timing
                MOV         A,#3FH
                MOVX        @DPTR,A
                MOV         P_SW2,#00H
                MOV         ADCCFG,#0FH                 ;Set up ADC The clock is the systemSystem clock/2/16
                MOV         ADC_CONTR,#80H              ;Enable ADC module

                ORL         ADC_CONTR,#40H              ;Start AD convert
                NOP
                NOP
                MOV         A,ADC_CONTR                 ;query ADC Completion mark
                JNB         ACC. 5,$-2
                ANL         ADC_CONTR,#NOT 20H          ;Clearance completion mark

                                                        ;Set the result to the left to align
                MOV         ADCCFG,#00H
                MOV         A,ADC_RES                   ;A storage ADC of 10 The high position of the bit result 8
                MOV         B,ADC_RESL                  ;B[7:6]storage ADC of 10 The low position of the bit result 2 ,B[5:0]for 0

;               MOV         ADCCFG,#20H                 ;Set the result to the right
;               MOV         A,ADC_RES                   ;A[3:0]storage ADC of 10 The high position of the bit result 2 ,A[7:2]for 0
;           MOV         B,ADC_RESL              ;B storage ADC of 10 8 The low position of the bit result
                SJMP        $
                END
```

17.4.4 use ADC The first 15 Channel measurement of external voltage or battery voltage

STC12H series ADC The first 15 The channel is used to measure the internal reference signal source. Since the internal reference signa And it will not change with the change of the operating voltage of the chip, so the 1.19V external Refer to the signal source, and then pass ADC of voltage or external battery voltage can be reversed by measuring the internal value.

Language code C

```
// The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
#define         FOSC                11059200UL
#define BRT                         (65536 - FOSC / 115200 / 4)
sfr AUXR                    =       0x8e;
sfr ADC_CONTR
sfr ADC_RES                 =       0xbc;
sfr ADC_RESL                =       0xbd;
sfr ADCCFG                  =       0xbe;
sfr P_SW2                   =       0xde;
#define ADCTIM
sfr P1M1                    =       0xba;
                                    (*(unsigned char volatile xdata *)0xfea8)
sfr P1M0
sfr P0M1                    =       0x91;
sfr P0M0                    =       0x92;
sfr P2M1                    =       0x93;
sfr P2M0                    =       0x94;
sfr P3M1                    =       0x95;
sfr P3M0                    =       0x96;
sfr P4M1                    =       0xb1;
sfr P4M0                    =       0xb2;
sfr P5M1                    =       0xb3;
sfr P5M0                    =       0xb4;
                            =       0xc9;
                            =       0xca;
int *BGV;                                               // The internal reference signal source value is stored in
                                                        //idata of EFH Address stores high bytes
                                                        //idata The address is F0H stored in low bytes
                                                        // The 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;                          //Set up ADC Internal timing
        P_SW2 &= 0x7f;

        ADCCFG = 0x2f;                          //Set up ADC The clock is the system clock/2/16
        ADC_CONTR = 0x8f;                       //Enable ADC Module and select the first 15 channel
}

        ADCRead()
int{int res;

        ADC_CONTR |= 0x40;                      //Start AD convert
        _nop_();
        _nop_();
        while (! (ADC_CONTR & 0x20));           //query ADC Completion mark
        ADC_CONTR &= ~0x20;                     //Clearance completion mark
        res = (ADC_RES << 8) | ADC_RESL;        //read ADC result

        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();                                  initialize //ADC
        UartInit();                                 //Serial port initialization

        ES = 1;
        EA = 1;
//      ADCRead();
//      ADCRead();                                  // The first two data are recommended to be discarded

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

        vcc = (int)(4096L * *BGV / res);            //(12 ADC Algorithm calculation) VREF The pin voltage is the battery voltage
//      vcc = (int)(1024L * *BGV / res);            //(10 ADC Attention to algorithm calculation) VREF The pin voltage is the battery voltage
                                                    , The unit of this voltage is millivolt(mV) //
        UartSend(vcc >> 8);                         //Output voltage value to serial port
        UartSend(vcc);

        while (1);
}
```

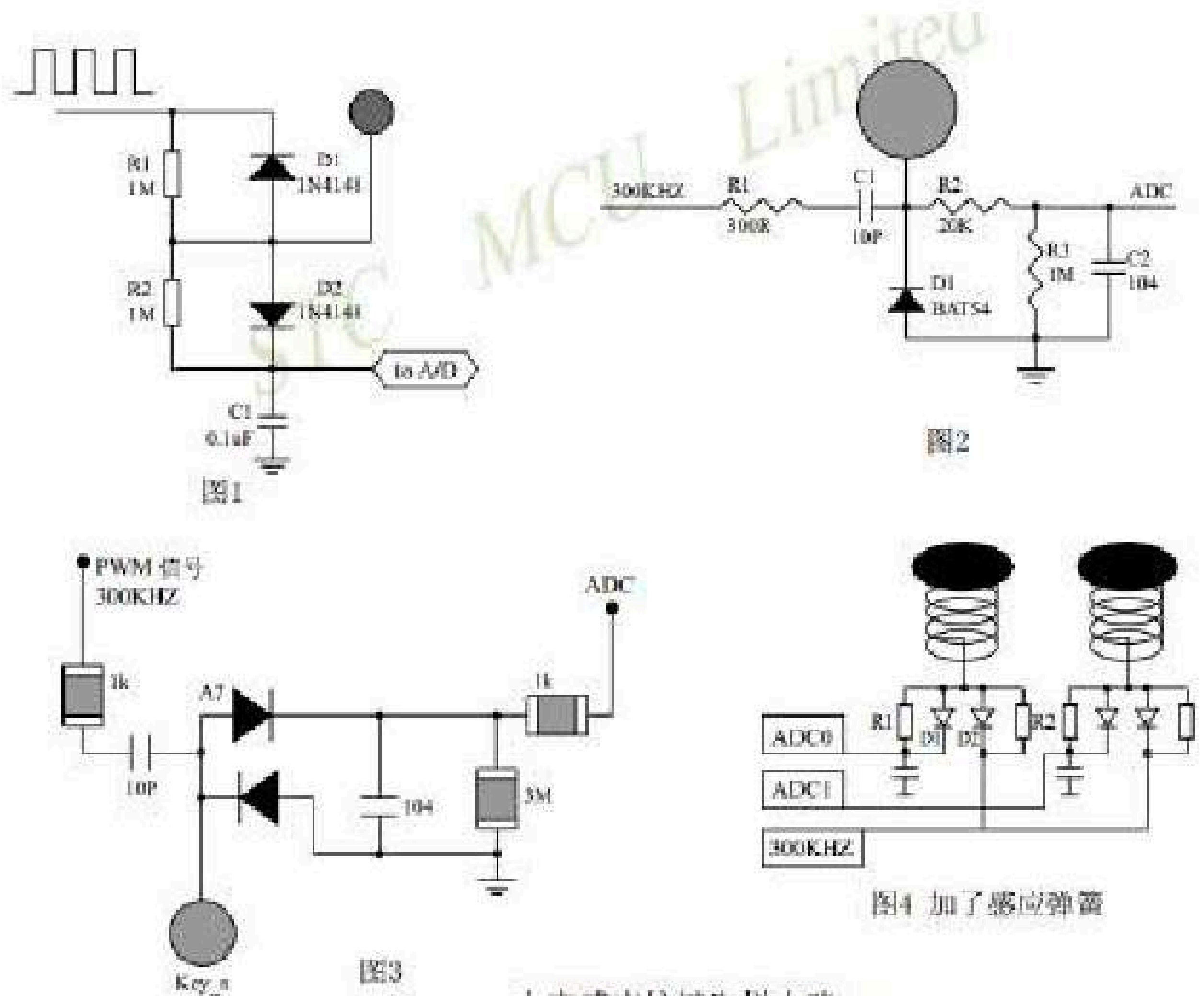
The above method is The channel pushes back the voltage ADC of the external battery. When the measurement of to use voltage and The measured value is proportional to 15, So you can also use the first channel pushes back the input voltage of the ADC The measured value of the voltage of the internal reference signal source BGV, internal reference signal source ADC external channel, assuming that the measured value is, currently obtained, and the input voltage of the external channel is $V_x = BGV / res_{bg} * res_x$ ADC res_x the input voltage of the external channel x ;

17.4.5 ADC Do capacitive touch buttons

Buttons are one of the most commonly used parts of circuits and an important input method for human-machine interfaces. We are most familiar with mechanical buttons, but mechanical buttons have easy to wear contacts. Instead of contact buttons (especially cheap buttons) . There are no mechanical contacts, long life and easy to use.

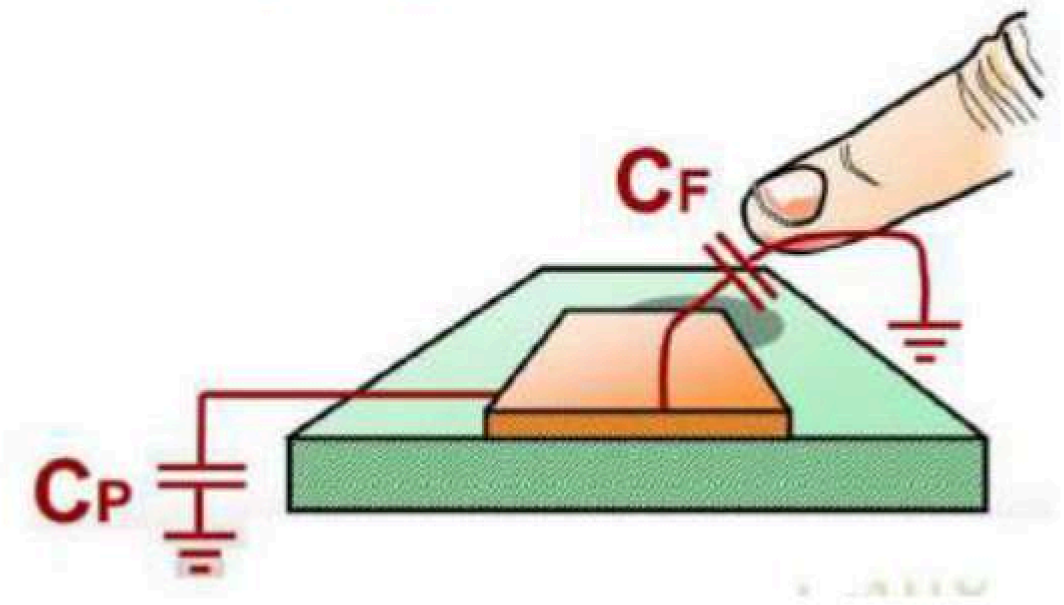
There are a variety of options for capacitive sensor buttons are a low-cost solution. Many years ago, special IC To achieve , buttons, with The strengthening of functions, as well as the practical experience of the technology of making capacitive sensor buttons is mature. The most typical and reliable of them is the use of ADC Do the plan.

This document details the use of STC with ADC The series MCU To do the plan, you can use any belt ADC Functional MCU Come on now. The previous figure below is the most used method. The principle is the same. This article uses the first 2 A picture

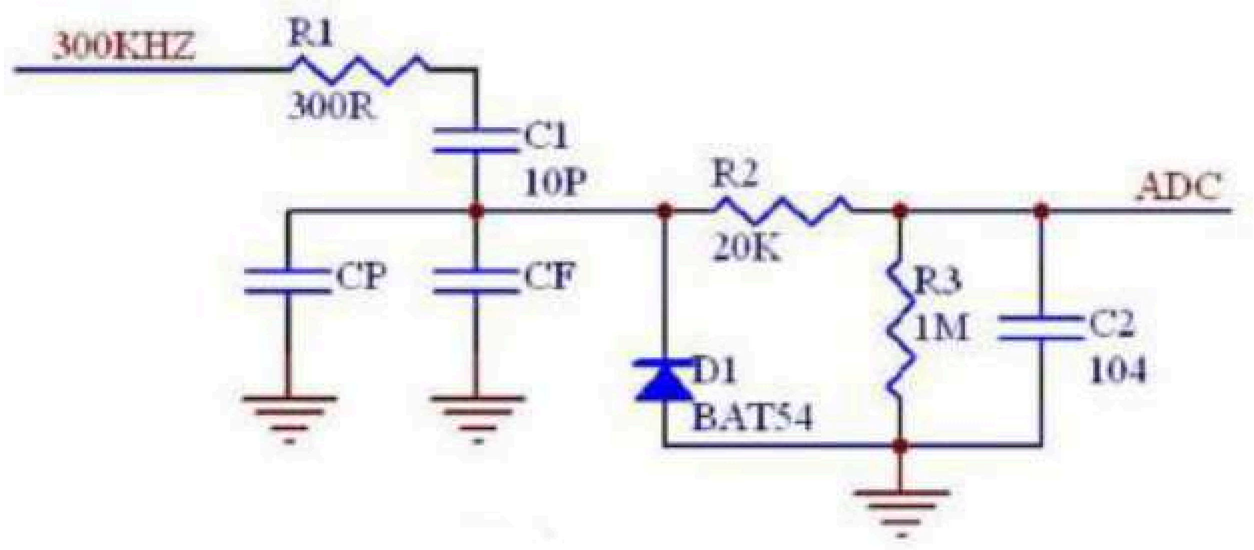


电容感应按键取样电路

In general, in practical applications, the induction spring shown is used to increase the area of the finger press. An equivalent piece of
It belongs to the board, and there is a capacitor CP to ground, and after the finger is pressed, a capacitor CF to ground is connected in parallel, as shown i



The following is a description of the circuit diagram. CP is the metal plate and the distributed capacitor, and CF is the finger capacitor. It is connected in parallel wit
voltage of the input 300KHZ square wave . After D1 is rectified, R2 and C2 are filtered and sent to the ADC. When the finger is pressed up, the voltage sent to the ADC is re
detect it.Press the button to act.



C Language code

// The test operating frequency is 24MHz

```
#include "reg51. h"
#include "intrins. h"
#define MAIN_Fosc        24000000UL                              // Define the master clock
#define Timer0_Reload (65536UL -(MAIN_Fosc / 600000))            //Timer 0 Reload value, corresponding 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;                                     //with series AD
sfr ADC_RES       =    0xBD;                                     //with series AD
sfr ADC_RESL      =    0xBE;                                     //with series AD
sfr AUXR          =    0x8E;
sfr AUXR2         =    0x8F;
#define CHANNEL        8                                         //ADC Number of channels
#define ADC_90T        (3<<5)                                    //ADC time 90T
#define ADC_180T       (2<<5)                                    //ADC time 180T
#define ADC_360T       (1<<5)                                    //ADC time 360T
#define ADC_540T       0                                         //ADC time 540T
#define ADC_FLAG       (1<<4)                                    //Software clearance 0
#define ADC_START      (1<<3)                                    //Automatic clearance
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 value ADC
u16 idata adc_prev[TOUCH_CHANNEL];                              //Previous value ADC
u16 idata TouchZero[TOUCH_CHANNEL];                             //0 Point value ADC
u8 idata TouchZeroCnt[TOUCH_CHANNEL];                           //0 Automatic tracking and counting of points
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;
        TR0 = 0;                                                //initialize Timer0 Output one 300KHZ clock
        AUXR |= 0x80;                                           //Timer0 set as 1T mode
        AUXR2 |= 0x01;                                          //Allow output clock
        TMOD = 0;                                               //Timer0 set as Timer, 16 bits Auto Reload.

        TH0 = (u8)(Timer0_Reload >> 8);
        TL0 = (u8)Timer0_Reload;
        TR0 = 1;
        ADC_init();                                             //ADC initialize
        delay_ms(50);                                           //50ms Delayed initialization point and previous
        for (i=0; i<TOUCH_CHANNEL; i++)                         // value and point automatic tracking count 0 0
        {
        adc_prev[i] = 1023;
        TouchZero[i] = 1023;
        TouchZeroCnt[i] = 0;
        }
        cnt_250ms = 0;
        while (1)
        {
                delay_ms(50);                                   //Every once in a while 50ms Process a button once
                ShowLED();
                if (++cnt_250ms >= 5)
                {
                        cnt_250ms = 0;
                        AutoZero();                                    //Every once in a while 250ms Automatic tracking of processing one-time points
                }
        }
}
void delay_ms(u8 ms)
{
        unsigned int i;

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

void ADC_init(void)
{
        P1M0 = 0x00;                                                   //8 road ADC
        P1M1 = 0xff;
        ADC_CONTR = 0x80;                                              //Allow 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) ;                                    ADC//wait End of conversion
        ADC_CONTR = 0x80;                                              //Clear flag
        return(((u16)ADC_RES << 2) | ((u16)ADC_RESL & 3));             //return ADC result

}

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

        for(i=0; i<TOUCH_CHANNEL; i++)                                 //Process a channel
        {
                j = adc[i];
                k = j - adc_prev[i];                                   //Subtract the
                F0 = 0;                                                previous reading //press
                if(k & 0x8000) F0 = 1, k = 0 - k;                      // Then find the difference between the two samples
                if(k >= 20)                                            //The change is relatively large
                {
                        TouchZeroCnt[i] = 0;                           // If the change is relatively large, then clear the counter 0
                        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 peristalsis, automatic point tracking 0
                {
                        if(++TouchZeroCnt[i] >= 20)          // Continuously detect small changes 20 times seconds
                        {
                                TouchZeroCnt[i] = 0;
                                TouchZero[i] = adc_prev[i];  // Slowly changing values as points 0
                        }
                adc_prev[i] = j;                             // Save the sampled value this time
        }
}
u8 check_adc(u8 index)                                       // 1 Number of calls // Get touch information function 50ms
                                                             // Press or release the judgment key, There is return control
{
        u16 delta;

        adc[index] = 1023 - Get_ADC10bitResult(index);       // value, Turn to press the key, ADC Value increase
        if(adc[index] < TouchZero[index]) return 0;          // 0 ADC // Get
        delta = adc[index] - TouchZero[index];               // ratio // The value of the point is still small, it is considered a key release
        if(delta >= 40) return 1;                            // Key press
        if(delta <= 20) return 0;                            // Key release
        return 2;                                            // Keep it in its original state
}

void ShowLED(void)
{
        u8 i;

        i = check_adc(0);
        if(i == 0) P_LED0 = 1;                               // Indicator light off
        if(i == 1) P_LED0 = 0;                               // The indicator light is on
        i = check_adc(1);
        if(i == 0) P_LED1 = 1;                               // Indicator light off
        if(i == 1) P_LED1 = 0;                               // The indicator light is on
        i = check_adc(2);
        if(i == 0) P_LED2 = 1;                               // Indicator light off
        if(i == 1) P_LED2 = 0;                               // The indicator light is on
        i = check_adc(3);
        if(i == 0) P_LED3 = 1;                               // Indicator light off
        if(i == 1) P_LED3 = 0;                               // The indicator light is on
        i = check_adc(4);
        if(i == 0) P_LED4 = 1;                               // Indicator light off
        if(i == 1) P_LED4 = 0;                               // The indicator light is on
        i = check_adc(5);
        if(i == 0) P_LED5 = 1;                               // Indicator light off
        if(i == 1) P_LED5 = 0;                               // The indicator light is on
        i = check_adc(6);
        if(i == 0) P_LED6 = 1;                               // Indicator light off
        if(i == 1) P_LED6 = 0;                               // The indicator light is on
        i = check_adc(7);
        if(i == 0) P_LED7 = 1;                               // Indicator light off
        if(i == 1) P_LED7 = 0;                               // The indicator light is on

}
```

Assembly code

```
; The test operating frequency is 24MHz

Fosc_KHZ        EQU        24000                            ; Define the master clock 24MHz
```

```
Reload          EQU     (65536 - Fosc_KHZ/600)      ,Corresponding to the reloaded value ;Timer 0 600KHz

ADC_CONTR       DATA    0xBC                        ;with AD 0 series
ADC_RES         DATA    0xBD                        ;with AD series
ADC_RESL        DATA    0xBE                        ;with AD series
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 Number of channels
ADC_90T         EQU     (3 SHL 5)                   ;ADC time 90T
ADC_180T        EQU     (2 SHL 5)                   ;ADC time 180T
ADC_360T        EQU     (1 SHL 5)                   ;ADC time 360T
ADC_540T        EQU     0                           ;ADC time 540T
ADC_FLAG        EQU     (1 SHL 4)                   ;Software clearance 0
ADC_START       EQU     (1 SHL 3)                   ;Automatic clearance

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                         ;30H~3FH, ADC Current Two bytes, one value
adc_prev        EQU     40H                         ;Previous, value ADC 40H~4FH, Two bytes, one value
TouchZero       EQU     50H                         ;0 50H~5FH, Two bytes, one value, point value ADC
TouchZeroCnt    EQU     60H                         ;0 60H~67H Automatic tracking and counting of points
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 Output one 300KHZ clock
                CLR         TR0
                ORL         AUXR,#080H                      ;Timer0 set as 1T mode
                ORL         AUXR2,#01H                      ;Allow output clock
                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 one 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 ;Initialization point 0 ADC 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 one touch key value
                DJNZ        cnt_250ms,L_MainLoop
                MOV         cnt_250ms,#5                        Automatic tracking of processing
                LCALL       F_AutoZero                      one-time points ;;250ms Automatic tracking of zero points
                SJMP        L_MainLoop

F_ADC_init:
                MOV         P1M0,#00H                       ;8 road ADC
                MOV         P1M1,#0FFH
                MOV         ADC_CONTR,#080H                 ;Allow 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            ADC;waitEnd of conversion
            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:                                            ;250ms    Call once
                                                       ; This is detected using the sum of the absolute values of the differences between
            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 ;press
            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,turn
                MOV     A,#LOW (TouchZeroCnt)       ; If the change is relatively large, then clear the counter 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 peristalsis, automatic point tracking
                                                    ;Continuously detect small changes
                                                    ; times/4 = 5 20 . 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),turn
                MOV     @R0,#0                      ;TouchZeroCnt[i]= 0;
                MOV     A,R5                        ; Slowly changing values as points0
                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
                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
```

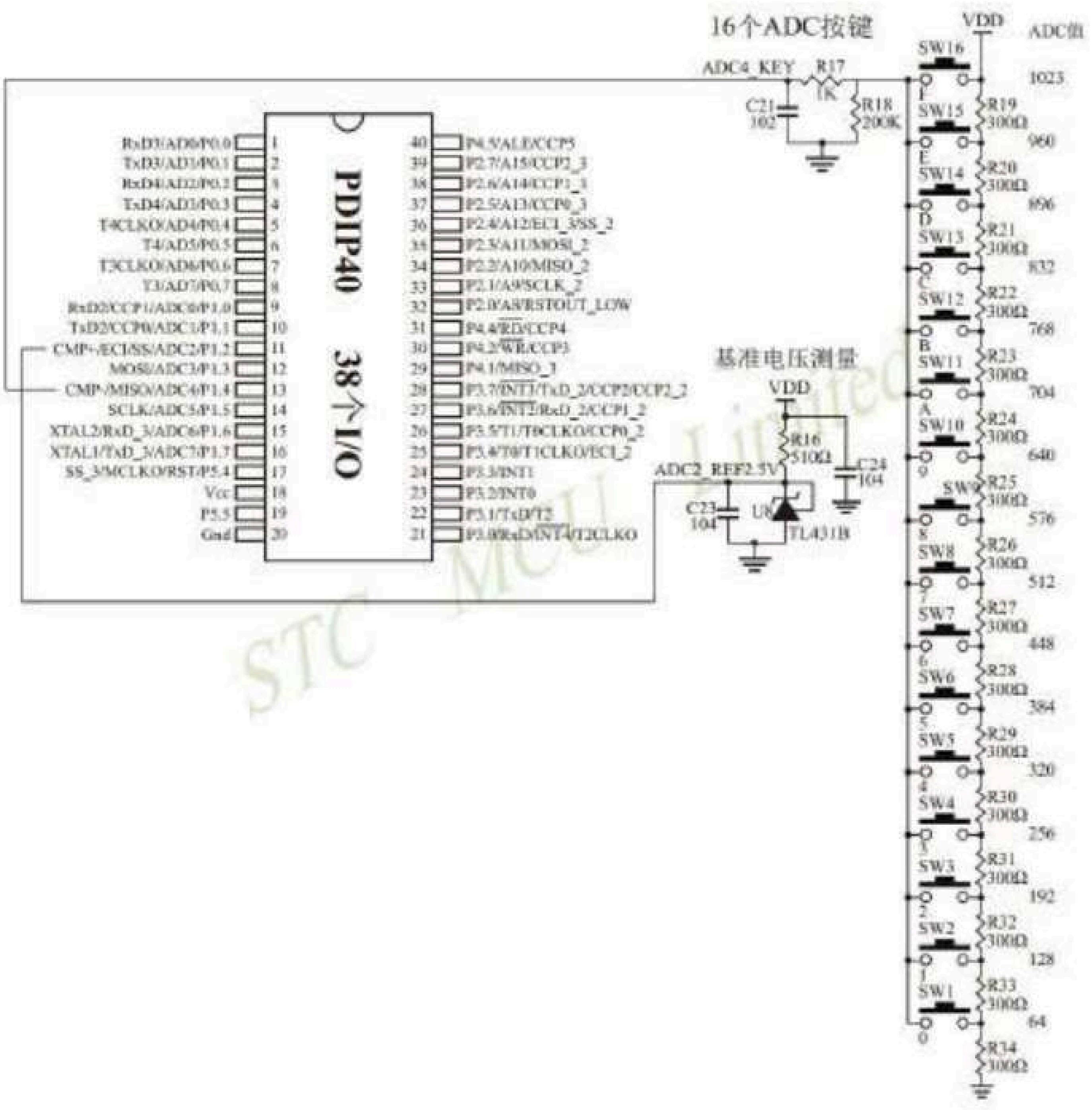
```
F_check_adc:                                          ; Press or release the judgment key There is return control
                MOV R4,7
                LCALL           F_Get_ADC10bitResult  ;returned ADC The value is [in R7]
                CLR             C
                MOV             A,#0FFH
                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
                MOV             A,R6                  ;calculate adc[index] - TouchZero[index]
                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),turn
                MOV         R7,#1                           ;if(delta >= 40) return 1; //Press the key to 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
```

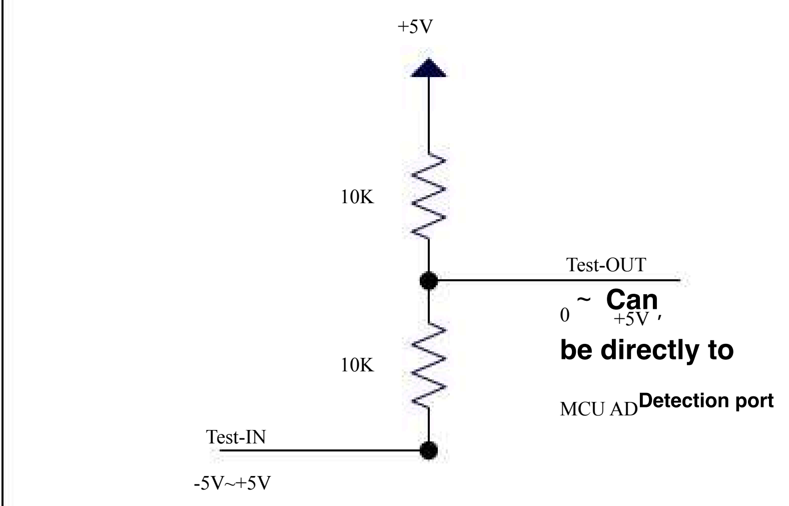
```
        POP                     3
        RET


        END
```

17.4.6 ADC Make button scanning application circuit diagram

How to read the ADC key: Read the ADC value every 10ms or so, and save the last 3 readings, compare the changes for a few hours, and then judge the key. When the judgment key is valid, a certain deviation is allowed, such as a deviation of ±16 words.

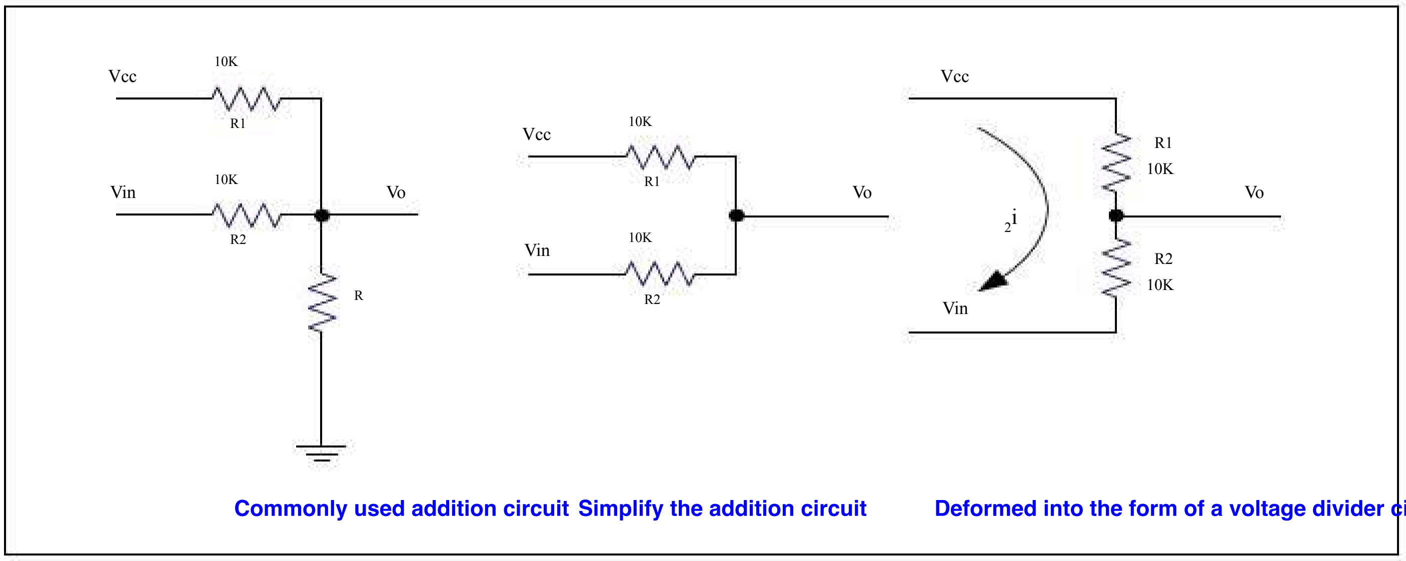


17.4.7 Reference circuit diagram for detecting negative voltage



Negative pressure conversion circuit

17.4.8 Commonly used addition circuits in the ADC application



Commonly used addition circuit Simplify the addition circuit Deformed into the form of a voltage divider circuit

Refer to the voltage divider circuit to get the formula 1

formula 1 : Vo = Vin + i_2 * R2

2 Formula: i_2 = (Vcc - Vin) / (R1 + R2) { Condition: Flow direction Vo 0 The current } }

Substitute into the formula to get the formula 3 R1=R2 2

3 formula : i_2 = (Vcc - Vin) / 2R2

Substitute the formula into the formula to get the formula 3 4

1 4 formula : Vo = (Vcc + Vin) / 2

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

In the analog-to-digital conversion measurement of the single-chip microcomputer, if the measured voltage is less than 0V You can use plus is required to be greater than and less than Vcc At this time, the addition circuit can increase the measurement voltage range of change of the measured voltage :

Substituting the above conditions into the formula 4 can get the following

(Vcc + Vin) / 2 > 0 Vin > -Vcc That

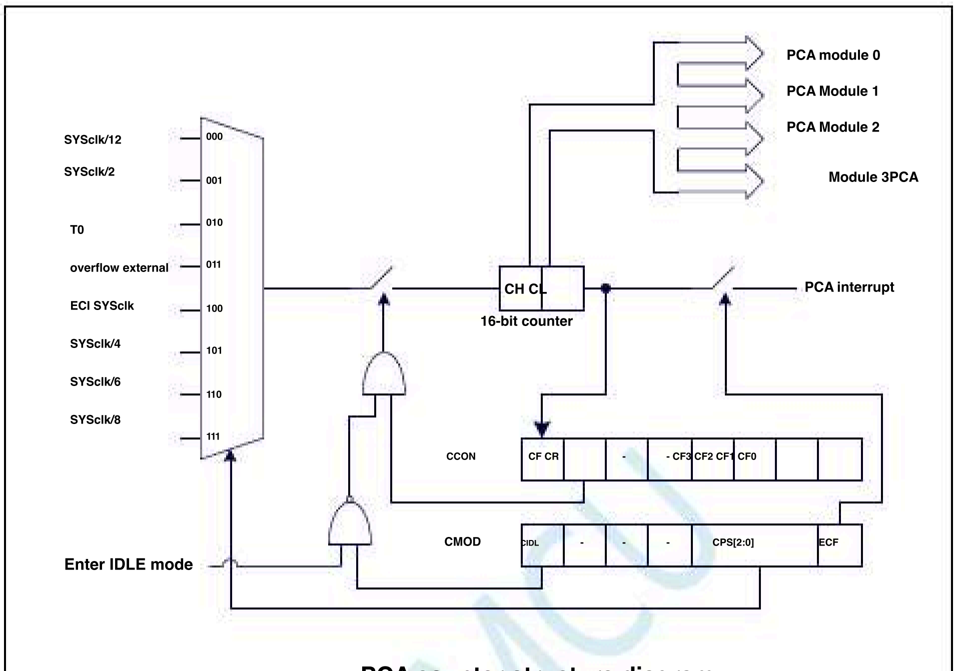
(Vcc + Vin) / 2 < Vcc is, that is Vin < Vcc

The above formula can be combined : -Vcc < Vin < Vcc

18 PCA/CCP/PWM application

STC12H The series of microcontrollers are integrated internally 3 Group programmable counter array (PCA/CCP/PWM) Module, can be used for software timing Device, external pulse capture, high-speed pulse output and PWM Pulse width modulated output.

PCA The interior contains a special 16 Bit counter, group 4 PCA The modules are all connected to it. The structure diagram of the counter is as fo PCA



PCA counter structure diagram

18.1 PCA Related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA Control register	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA Mode register module	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA Mode control register 0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0 x000,0000	
CCAPM1	PCA Module mode control register 1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1 x000,0000	
CCAPM2	PCA Module mode control register 2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAPM3	PCA Module mode control register 3	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CL	PCA Counter low byte	E9H									0000,0000
CCAP0L	PCA 0Module low byte	EAH									0000,0000
CCAP1L	PCA Module low byte 1	EBH									0000,0000
CCAP2L	PCA Module low byte 2	ECH									0000,0000
CCAP3L	PCA Module low byte 3	FD55H									0000,0000
PCA_PWM0	PCA0 of PWM Mode register	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 of PWM Mode register	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 of PWM Mode register	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000
PCA_PWM3	PCA3 of PWM Mode register	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L	0000,0000
CH	PCA Counter high byte	F9H									0000,0000
CCAP0H	PCA module 0 High byte	FAH									0000,0000
CCAP1H	PCA module 1 High byte	FBH									0000,0000
CCAP2H	PCA Module 2 high byte	FCH									0000,0000
CCAP3H	PCA 3Module high byte	FD56H									0000,0000

18.1.1 PCA Control register (CCON)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
CCON	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0

CF：PCA The counter overflows the interrupt flag. when PCA When the bit counter counts that an overflow occurs, the hardware automatically p

CPU Make an interrupt request. This flag needs to be cleared by the software.

CR：The counter allows control bits PCA

0: Stop count PCA

1: start count PCA

CCFn（n=0,1,2,3）: Module interrupt flag. when PCA When the module is matched or captured, the hardware automatically sends this lo

CPU Make an interrupt request. This flag needs to be cleared by the software.

18.1.2 PCA Mode register (CMOD)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF

CIDL: Whether to stop in idle mode PCA count.

0: In idle mode PCA Keep counting PCA

1: In idle mode PCA Stop counting

CPS[2:0]: PCA Counting pulse source selection bit

CPS[2:0]	PCA Input clock source
000	System clock/12
001	System clock/2
010	Timer 0 Overflow pulse
011	ECI External input clock of the pin
100	System clock
101	system clock/4
110	System clock/6
111	System clock/8

ECF：PCA The counter overflows the interrupt permission bit.

0: Prohibited PCA Counter overflow interrupt

1: Enable PCA Counter overflow interrupt

18.1.3 PCA Counter register (CL，CH)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
CL	E9H								
CH	F9H								

by CL and CH The two bytes are combined into a 16 Bit counter, CL Is low 8 Bit counter，CH For high 8 Bit counter. each PCA clock 16 and the counter is automatically added 1.

18.1.4 PCA Module mode control register (CCAPMn)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2
CCAPM3	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

ECOMn: Allow PCA n Module comparison function

CCAPPn: Allow module PCA Perform rising edge capture

CCAPNn: Allow n Module for falling edge capture PCA n

MATn: Allow PCA n Matching function of the module , high-speed pulse output

TOGn: Allow PCA n function of the module

PWMn: Allow PCA n Pulse width modulation output function of the module

ECCFn: Allow PCA Module matching/Capture interrupt

18.1.5 PCA Module mode capture value/Comparison value register (CCAPnL , CCAPnH)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
CCAP0L	EAH								
CCAP1L	EBH								
CCAP2L	ECH								
CCAP3L	FD55H								
CCAP0H	FAH								
CCAP1H	FBH								
CCAP2H	FCH								
CCAP3H	FD56H								

when PCA When the module capture function is enabled , Used to save the sum when the capture occurs The count value (CL and CH) ; CCAPnL When the module comparison function is enabled , The controller will send the current The count value in and stored in CCAPnL And the controller PCA when PCA The values in are compared, and the comparison results are given; when the matching function is enabled , will send the current CCAPnH values in the current are compared with the values stored in and CCAPnH , and gives CL Compare the values in to see if they match (equal) Matching result. CH CCAPnL and

18.1.6 PCA module PWM Mode control register (PCA_PWMn)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PCA_PWM0	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L
PCA_PWM1	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L
PCA_PWM2	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L
PCA_PWM3	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L

EBSn[1:0]: PCA n Module of PWM Digit control

EBSn[1:0]	Number of digits PWM	Overload	Compare
00	8 Position PWM	value {EPCnH,	values {EPCnL,
01	7 position PWM	CCAPnH[7:0]} {EPCnH, CCAPnH[6:0]}	CCAPnL[7:0]} {EPCnL, CCAPnL[6:0]}
10	6 position PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10 PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]: 10 9 The first and second PWM 10 The overloaded value of the

XCCAPnL[1:0]: The first and second 9 bit, the comparison value of the bit

EPCnH: PWM 10 10 PWM 9 The first place, the first place The first place, the first place The first 7 bit, 10 bit PWM In mode, the highest bit of the overload value (8 bit The first Bit)

EPCnL: PWM In mode, the highest bit of the comparison value (8 bit The first place, the first place The first place, the first place The first 7 bit The first PWM Bit) bit, 10 bit PWM

Note: In the update 10 When overloading the value, you must first write the upper two digits XCCAPnH[1:0] Then write the low bit CCAPnH[7:0]。

Working mode 18.2 PCA

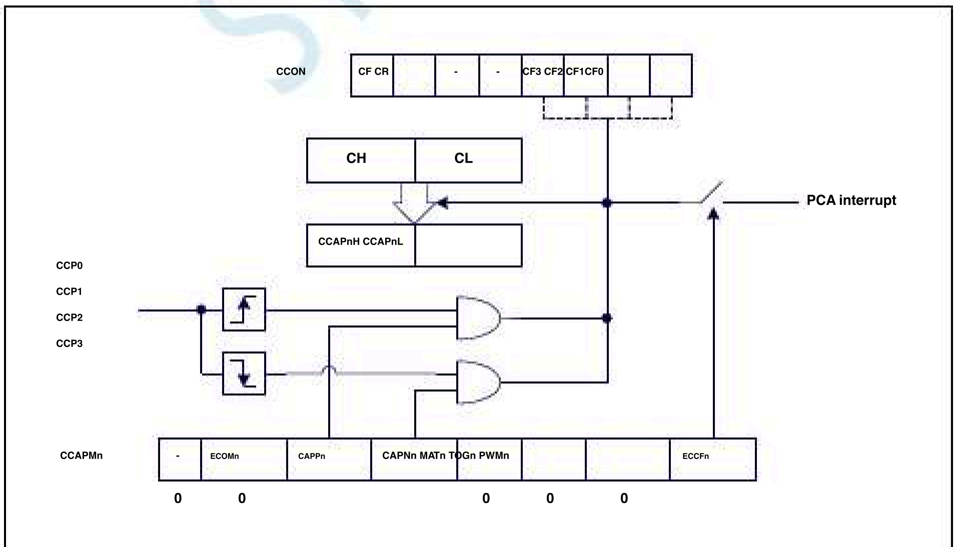
STC12H A total of series of microcontrollers group Modules, each group of modules can independently set the working mode. The mode settings are a

PCA								Module function
-	ECOMn	CAPPn	4 CCAPMn CAPNn	MATn	TOGn	PWMn	ECCFn	
-	0	0	0	0	0	0	0	No operation
-	1	0	0	0	0	1	0	bit Mode, no interruption PWM
-	1	1	0	0	0	1	1	6/7/8/10 PWM position Mode, generating a rising edge interrupt
-	1	0	1	0	0	1	1	6/7/8/10 6/7/8/10 PWM mode, generating a falling edge interrupt
-	1	1	1	0	0	1	1	6/7/8/10 bit PWM Mode, generating edge interrupt
-	0	1	0	0	0	0	x	16 bit, rising edge capture
-	0	0	1	0	0	0	x	16 bit, falling edge capture bit
-	0	1	1	0	0	0	x	16 Bit edge capture
-	1	0	0	1	0	0	x	16 bit software timer bit high-speed
-	1	0	0	1	1	0	x	16 pulse output

18.2.1 Capture mode

To make a PCA The module works in capture mode, register CCAPMn in At least one must be set (It can also be set in both positions). PCA When the module is operating in capture mode, the external of the module and CAPNn CAPPn Line sampling. When a valid transition is sampled, The controller will immediately counter PCA CCP0/CCP1/CCP2 and CL The input of the pin transitions into The count value in is loaded into the capture of the m PCA CCAPnH, At the same time will CCON The corresponding in the register CCFn set, if CCAPMn in ECCFn bit Register and CCAPnL PCA The interrupt entry address of the module is shared, so it needs to be judged in the interrup Is set to, an interrupt will be generated. Due to all The interrupt is which module generated the interrupt, and note that the interrupt flag needs to be cleared by the software.

PCA The structure diagram of the module working in capture mode is shown in the following figure :

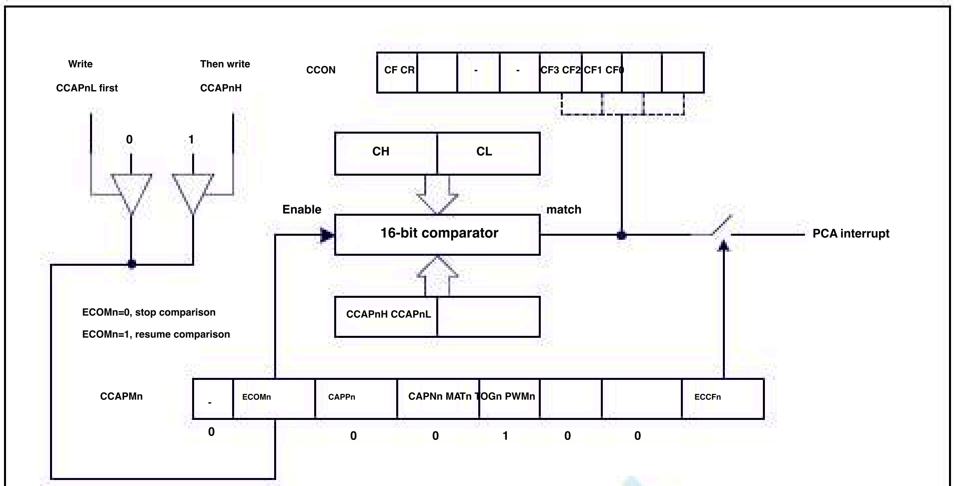

CCON
CF CR - - CF3 CF2 CF1 CF0
CH
CL
CCAPnH CCAPnL
PCA interrupt
CCP0
CCP1
CCP2
CCP3
CCAPMn
- ECOMn CAPPn CAPNn MATn TOGn PWMn ECCFn
0 0 0 0 0

PCA module capture mode

18.2.2 Software timer mode

By setting Register of CCAPMn ECOM and MAT Bit, can make The module is used as a software timer. Counter value PCA and CHCL Capture the value of the register with the module and CCAPnH CCAPnL PCA Compare, when the two are equal , CCON in CCFn Will be 1 Set, if CCAPMn in ECCFn An interrupt will be generated when it is set to 1. The flag bit CCFn needs to be cleared by the software.

PCA The structure diagram of the module operating in the software timer mode is shown in the following figure :

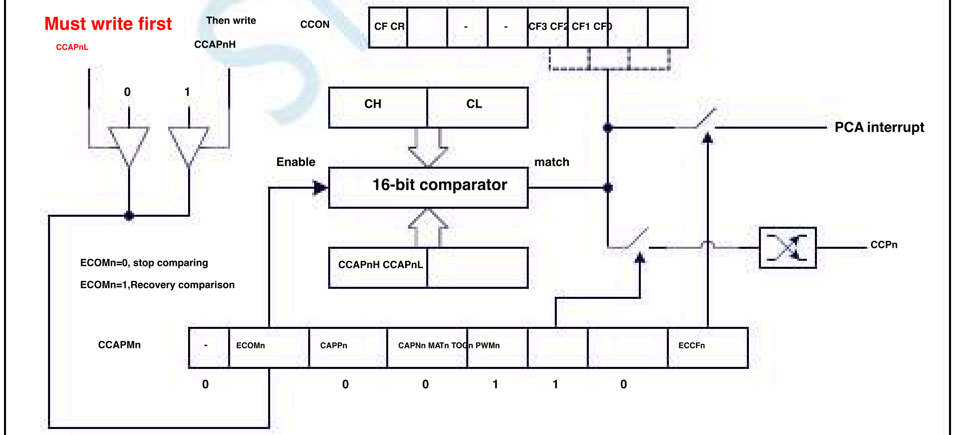


Module software timer mode PCA

18.2.3 High-speed pulse output mode

When the count value of the counter matches the value of the module capture register, PCA The output will be flipped. To be modular CCPn Activate the high-speed pulse output mode , CCAPMn Sum of registers TOGn MATn ECOMn 1 All positions must be set.

PCA The structure diagram of the module operating in the high-speed pulse output mode is shown in the following figure :



PCA Module high-speed pulse output mode

18.2.4 PWM Pulse width modulation mode and frequency calculation formula

18.2.4.1 8 bit PWM pattern

Pulse width modulation is a technique that uses a program to control the duty cycle, period, and phase waveforms of a waveform, drive

D/A

It is widely used in conversion and other occasions. For 8 bit PWM8 6 bit PWM Or bit STC8 The modules of the Series of microcontrollers can be made to work by setting their re mode. To enable 8bit PWM Module of PWM Functions, modules, etc memory CCAPMn of PWMn and ECOMn or PWM

1 The position must be set. 10

PCA_PWMn PCA The module works in place 8 PWM Mode, at this time will {0,CL[7:0]}

Use one in the {EPCnL,CCAPnL[7:0]} EBSn[1:0] Set to 00 when Make a comparison. when PCA The module works in 8 bit PWM In mode, since all modules have a total of

register and the capture register PCA Counters, all of them have the same output frequency. The output duty cycle of each module uses registers {EPCnL,CCAPnL[7:0]}

Set it up. when {0,CL[7:0]} The value is {EPCnL,CCAPnL[7:0]} When, the output is low; when {0,CL[7:0]} The value is equal to or

less than greater than {EPCnL,CCAPnL[7:0]} When the output is high. when CL[7:0] The value is determined by FF become 00 by When it overflows, {EPCnH,CCAPnH[7:0]}

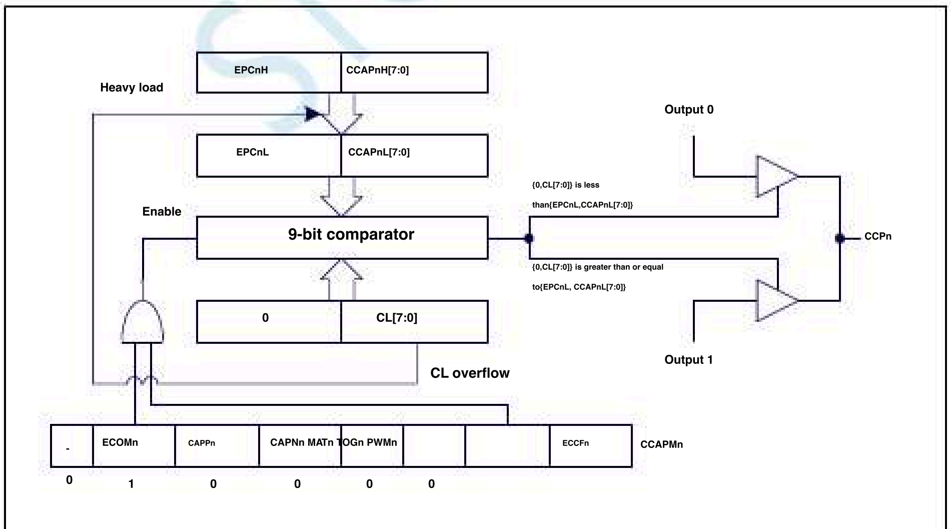
Reload the content to {EPCnL,CCAPnL[7:0]} in. This allows for interference-free updates PWM

8 Bit mode PWM Frequency= $\dfrac{\text{PCA clock input source frequency}}{256}$

When EPCnH=0 and CCAPnH=00H, the PWM fixed output is high

. When EPCnH=1 and CCAPnH=FFH, the PWM fixed output is low.

PCA The module works in 8 bit PWM The structure diagram of the pattern is shown in the figure below :



8 PWM mode PCA module bit

18.2.4.2 7 bit PWM pattern

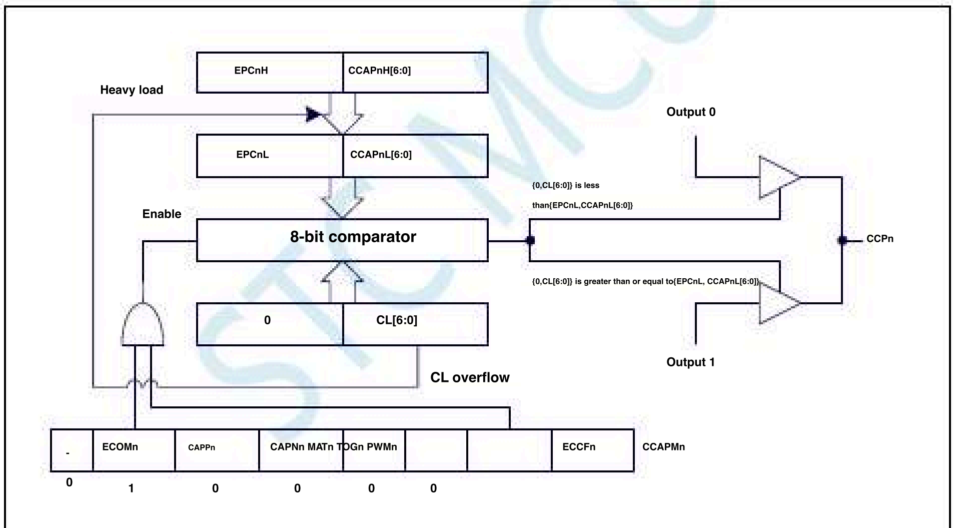
PCA_PWMn EBSn[1:0] Set to PCA The module works in place $_{7}$ bit PWM when Mode, at this time will {0,CL[6:0]}
Use one in the {EPCnL,CCAPnL[6:0]} Make a comparison. when PCA The module works in 7 bit PWM In mode, since all modules have a total of
register and the capture register PCA Counters, all of them have the same output frequency. The output duty cycle of each module uses registers {EPCnL,CCAPnL[6:0]}
Set it up. when {0,CL[6:0]} The value is {EPCnL,CCAPnL[6:0]} When, the output is low; when {0,CL[6:0]} The value is equal to or less than greater than {EPCnL,CCAPnL[6:0]} When the output is high. when CL[6:0] The value is determined by 7F becomes 00 When it overflows, {EPCnH,CCAPnH[6:0]}
Reload the content to {EPCnL,CCAPnL[6:0]} in. This allows for interference-free updates PWM.

7-bit mode PWM Frequency= $\dfrac{\text{PCA clock input source frequency}}{128}$

When EPCnH=0 and CCAPnH=00H, the PWM fixed output is high

. When EPCnH=1 and CCAPnH=FFH, the PWM fixed output is low.

PCA The module works in 7 bit PWM The structure diagram of the pattern is shown in the figure below :



7 PWM mode PCA module bit

18.2.4.3 6 bit PWM pattern

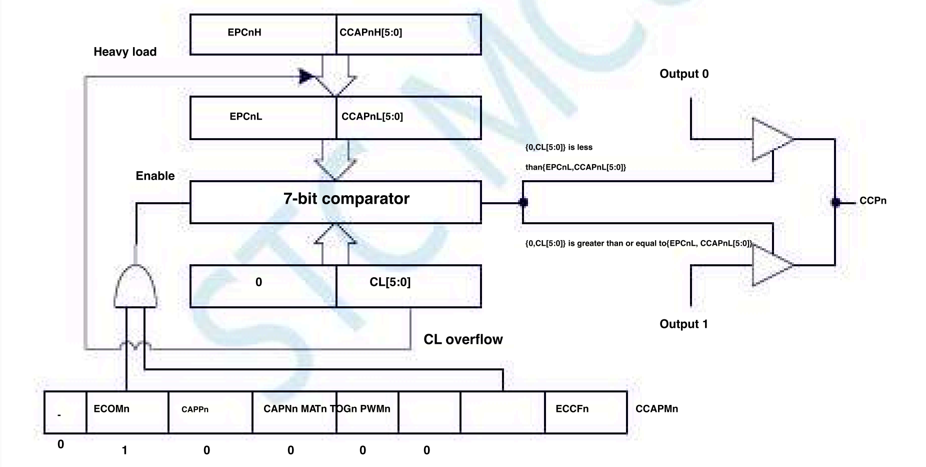
PCA_PWMn EBSn[1:0] Set to PCA The module works in place PWM when 10 Mode, at this time will 6 {0,CL[5:0]} Use one in the {EPCnL,CCAPnL[5:0]} Make a comparison. when PCA The module works in 6 bit PWM In mode, since all modules have a total of register and the capture register PCA Counters, all of them have the same output frequency. The output duty cycle of each module uses registers {EPCnL,CCAPnL[5:0]} Set it up. when {0,CL[5:0]} The value is {EPCnL,CCAPnL[5:0]} When, the output is low; when {0,CL[5:0]} The value is equal to or less than greater than {EPCnL,CCAPnL[5:0]} When the output is high. when CL[5:0] The value is determined by 3F become 00 When it overflows, {EPCnH,CCAPnH[5:0]} Reload the content to {EPCnL,CCAPnL[5:0]} in. This allows for interference-free updates PWM.

8 Bit mode PWM Frequency= $\dfrac{\text{PCA clock input source frequency}}{64}$

When EPCnH=0 and CCAPnH=00H, the PWM fixed output is high

. When EPCnH=1 and CCAPnH=FFH, the PWM fixed output is low.

PCA The module works in 6 bit PWM The structure diagram of the pattern is shown in the figure below :



6 PWM mode PCA module bit

18.2.4.4 10 bit PWM pattern

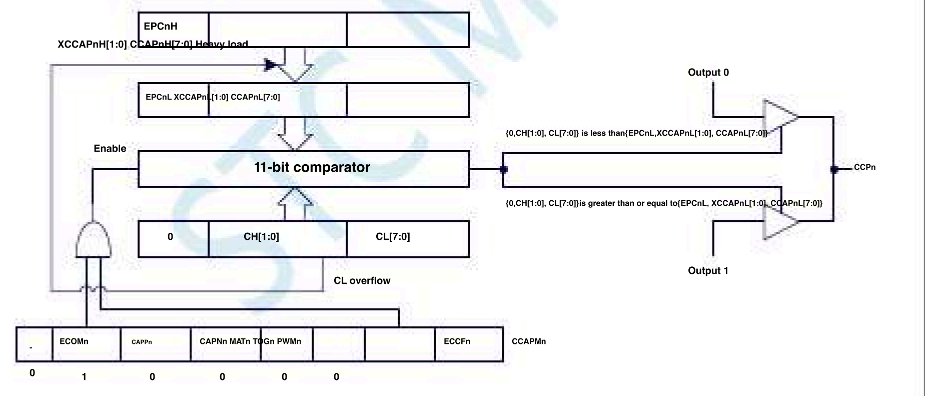
PCA_PWMn In the register EBSn[1:0] Set to 11 when , PCA module n Work in 10 bit Mode, at this time will PWM {CH[1:0],CL[7:0]} With capture {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]} Make a comparison. when The module works in 10PCA register bits PWM mode, since all modules share one PCA Counters, all of them have the same output frequency. The output duty cycle of each r Better than using a register {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]} Set it up. when {CH[1:0],CL[7:0]} The value of is less than the va {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]} When, the output is low; {CH[1:0],CL[7:0]} of is equal to or greater than {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]} when, the output is high. when {CH[1:0],CL[7:0]} The value is determined by 3FF become 00 overflow When, so that you can achieve {EPCnH,XCCAPnH[1:0],CCAPnH[7:0]} Reload the content to {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]} in. interference-free updates PWM。

$$10\text{-bit mode PWM Frequency} = \frac{\text{PCA clock input source frequency}}{1024}$$

When EPCnH=0, XCCAP0H=0 and CCAPnH=00H, the PWM fixed output is high

when EPCnH=1, XCCAP0H=3 and CCAPnH=FFH, the PWM fixed output is low

PCA The module works in 10 bit PWM The structure diagram of the pattern is shown in the figure below :


EPCnH
XCCAPnH[1:0] CCAPnH[7:0] Heavy load
EPCnL XCCAPnL[1:0] CCAPnL[7:0]
Enable
11-bit comparator
0
CH[1:0]
CL[7:0]
CL overflow
Output 0
{0,CH[1:0], CL[7:0]} is less than{EPCnL,XCCAPnL[1:0], CCAPnL[7:0]}
CCPn
{0,CH[1:0], CL[7:0]}is greater than or equal to{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}
Output 1
-
ECOMn
CAPPn
CAPNn MATn TOGn PWMn
ECCFn
CCAPMn
0
1
0
0
0
0

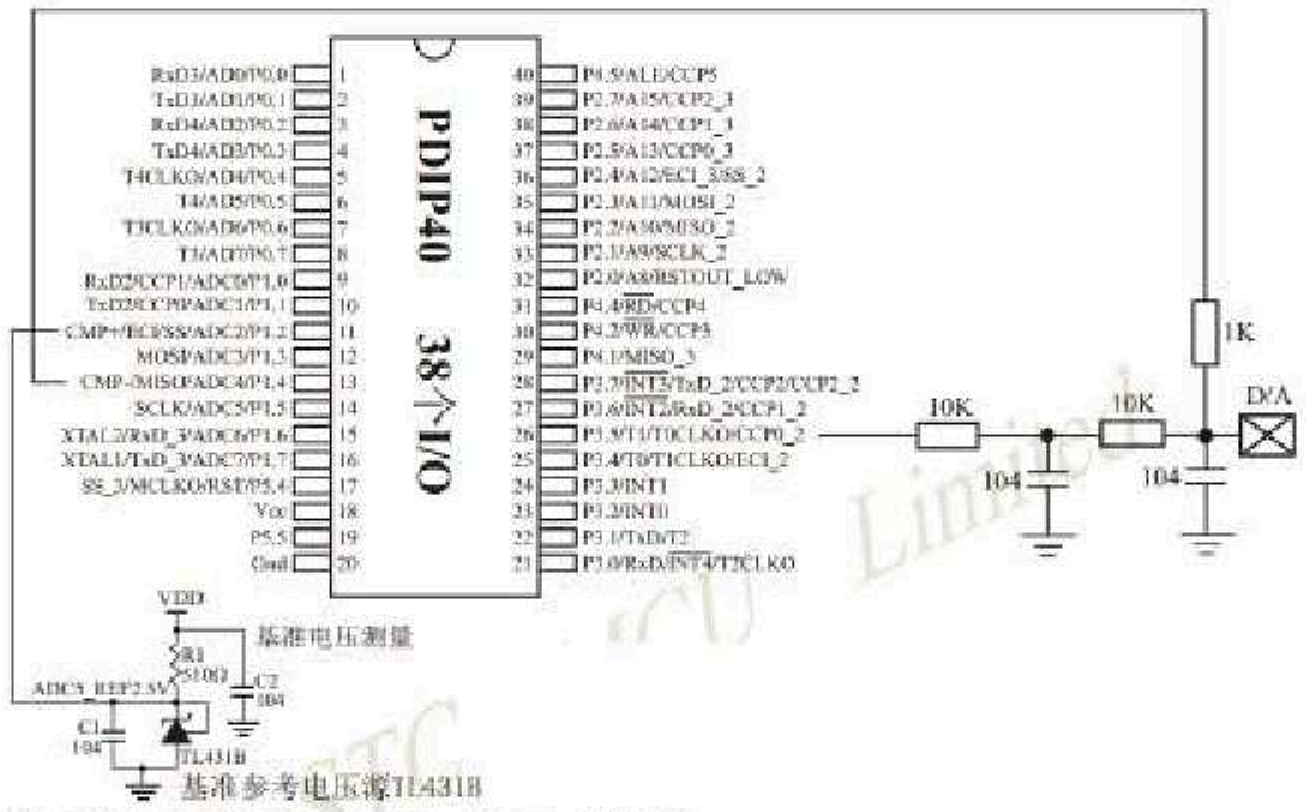
PWM mode PCA module 10 bits

18.2.4.5 How to control PWM Fixed output, high level/low level

Dangdang PCA_PWMn &= 0xC0 , CCAPnH = 0x00 when , PWM Fixed output high level

PCA_PWMn |= 0x3F , CCAPnH = 0xFF when , PWM fixed output low level

18.3 use CCP/PCA/PWM Module implementation 8-16 bit DAC Reference circuit diagram of



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

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

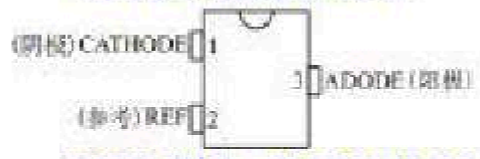
提示：

（1）PWM频率越高，输出波形越平滑。

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

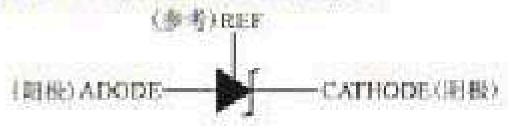
（3）如果要输出高精准电压，建议用A/D检测输出的电压值，然后根据A/D检测的电压值逐步调整到所需要的电压。

基准参考电压源TL431B



SOT23-3封装，RMB￥0.15~0.3

基准参考电压源TL431B的符号



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

18.4 Sample program

18.4.1 PCA output PWM (6/7/8/10 Bit)

C Language code

```
// The test operating frequency is 11.0592MHz

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

sfr     CCON        =   0xd8;
sbit CF             =   CCON^7;
sbit CR             =   CCON^6;
sbit CCF2           =   CCON^2;
sbit CCF1           =   CCON^1;
sbit CCF0           =   CCON^0;
sfr CMOD            =   0xd9;
sfr CL              =   0xe9;
sfr CH              =   0xf9;
sfr CCAPM0          =   0xda;
sfr CCAP0L          =   0xea;
sfr CCAP0H          =   0xfa;
sfr PCA_PWM0        =   0xf2;
sfr CCAPM1          =   0xdb;
sfr CCAP1L          =   0xeb;
sfr CCAP1H          =   0xfb;
sfr PCA_PWM1        =   0xf3;
sfr CCAPM2          =   0xdc;
sfr CCAP2L          =   0xec;
sfr CCAP2H          =   0xfc;
sfr PCA_PWM2        =   0xf4;
sfr P0M1            =   0x93;
sfr P0M0            =   0x94;
sfr P1M1            =   0x91;
sfr P1M0            =   0x92;
sfr P2M1            =   0x95;
sfr P2M0            =   0x96;
sfr P3M1            =   0xb1;
sfr P3M0            =   0xb2;
sfr P4M1            =   0xb3;
sfr P4M0            =   0xb4;
sfr P5M1            =   0xc9;
sfr P5M0            =   0xca;
void main()
{

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

```
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;
        CCON = 0x00;
        CMOD = 0x08;                            //PCA The clock is the system clock
        CL = 0x00;
        CH = 0x00;
--6 bit//    PWM--
        CCAPM0 = 0x42;                                          PWM Working mode
                                                0The module output is 6 bit //PCA PWM
        PCA_PWM0 = 0x80;
        CCAP0L = 0x20;                          //PCA           50%[(40H-20H)/40H]
        CCAP0H = 0x20;                          //PWM The duty cycle is 6
--7 bit//    PWM--
        CCAPM1 = 0x42;                                          PWM Working mode
        PCA_PWM1 = 0x40;                        1The module output is 7 bit //PCA PWM
        CCAP1L = 0x20;                          //PCA           75%[(80H-20H)/80H]
        CCAP1H = 0x20;                          //PWM The duty cycle is 7
--8 bit//    PWM--
//      CCAPM2 = 0x42;                                          PWM Working mode
//      PCA_PWM2 = 0x00;                        2The module output is 8 bit //PCA PWM
//      CCAP2L = 0x20;                          //PCA           87.5%[(100H-20H)/100H]
//      CCAP2H = 0x20;                          //PWM The duty cycle is 8
//-- 10      bit  PWM--
        CCAPM2 = 0x42;                          2The module is //PCA Working mode PWM
        PCA_PWM2 = 0xc0;                        //PCA //PWM     bit PWM10
        CCAP2L = 0x20;                          2The output duty cycle of the module is
        CCAP2H = 0x20;
        CR = 1;                                 //Start  PCA  Timer

        while (1);
}
```

Assembly code

```
; The test operating frequency is 11.0592MHz

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

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

                ORG             0000H
                LJMP            MAIN

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

                MOV             CCON,#00H
                MOV             CMOD,#08H                       ;PCA The clock is the system clock
                MOV             CL,#00H
                MOV             CH,#0H
--6 bit;        PWM--
                MOV             CCAPM0,#42H                     ;PCA Module 0 The working
                MOV             PCA_PWM0,#80H                   ;PCA module is 6 output PWM bit PWM
                MOV             CCAP0L,#20H                     ;PWM The duty cycle is 50%[(40H-20H)/40H]
                MOV             CCAP0H,#20H
--7 bit;        PWM--
                MOV             CCAPM1,#42H                     ;PCA Module 1 The working
                MOV             PCA_PWM1,#40H                   ;PCA module is 7 output PWM bit PWM
                MOV             CCAP1L,#20H                     ;PWM The duty cycle is 75%[(80H-20H)/80H]
                MOV             CCAP1H,#20H
--8 bit;        PWM--
;               MOV             CCAPM2,#42H                     ;PCA Module 2 The working
;               MOV             PCA_PWM2,#00H                   ;PCA module is 8 output PWM bit PWM
;               MOV             CCAP2L,#20H                     ;PWM The duty cycle is 87.5%[(100H-20H)/100H]
;               MOV             CCAP2H,#20H
;-- 10           bit PWM--
                MOV             CCAPM2,#42H                     ;PCA For the Module 2 Working mode PWM
                MOV             PCA_PWM2,#0C0H                  ;PCA output duty 10 bit PWM
                MOV             CCAP2L,#20H                     ;PWM cycle is 96.875%[(400H-20H)/400H]
                MOV             CCAP2H,#20H
                SETB            CR                              ;Start PCA Timer

                JMP             $
```

```
END
```

18.4.2 PCA Capture and measure pulse width

C Language code

```
//The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
sfr      CCON        =   0xd8;
sbit CF              =   CCON^7;
sbit CR              =   CCON^6;
sbit CCF2            =   CCON^2;
sbit CCF1            =   CCON^1;
sbit CCF0            =   CCON^0;
sfr CMOD             =   0xd9;
sfr CL               =   0xe9;
sfr CH               =   0xf9;
sfr CCAPM0           =   0xda;
sfr CCAP0L           =   0xea;
sfr CCAP0H           =   0xfa;
sfr PCA_PWM0         =   0xf2;
sfr CCAPM1           =   0xdb;
sfr CCAP1L           =   0xeb;
sfr CCAP1H           =   0xfb;
sfr PCA_PWM1         =   0xf3;
sfr CCAPM2           =   0xdc;
sfr CCAP2L           =   0xec;
sfr CCAP2H           =   0xfc;
sfr PCA_PWM2         =   0xf4;
sfr P0M1             =   0x93;
sfr P0M0             =   0x94;
sfr P1M1             =   0x91;
sfr P1M0             =   0x92;
sfr P2M1             =   0x95;
sfr P2M0             =   0x96;
sfr P3M1             =   0xb1;
sfr P3M0             =   0xb2;
sfr P4M1             =   0xb3;
sfr P4M0             =   0xb4;
sfr P5M1             =   0xc9;
sfr P5M0             =   0xca;
unsigned char        cnt;                //Store PCA Timing overflow times
unsigned long        count0;             //Record the last captured value of the record
unsigned long        count1;             //Record the capture value this time
unsigned long        length;             //The length of time to store the signal

void PCA_Isr() interrupt 7
{
        if (CF)
{
        CF = 0;
        cnt++;                                  //PCA Timing overflow times +1
        }
        if (CCF0)
        {
                CCF0 = 0;
                count0 = count1;                // Back up the last captured value
((unsigned char *)&count1)[3] = CCAP0L;
((unsigned char *)&count1)[2] = CCAP0H;
((unsigned char *)&count1)[1] = cnt;
((unsigned char *)&count1)[0] = 0;
length = count1 - count0;                       //length What is saved is the captured pulse width
        }
}
void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;
        cnt = 0;                                //User variable initialization
        count0 = 0;
        count1 = 0;
        length = 0;
        CCON = 0x00;
        CMOD = 0x09;                            //PCA The clock is the system clock, Enable Timing interruption
        CL = 0x00;
        CH = 0x00;
        CCAPM0 = 0x11;                          //PCA The module 0 is 16 Bit capture mode (falling edge capture)
//      CCAPM0 = 0x21;                          //PCA The module 0 is 16 Bit capture mode (rising edge capture)
//      CCAPM0 = 0x31;                          //PCA The module 0 is 16 Bit capture mode (edge capture)
        CCAP0L = 0x00;
        CCAP0H = 0x00;
        CR = 1;                                 //Start PCA Timer
        EA = 1;
        while (1);
}
```

Assembly code

```
; The test operating frequency is 11.0592MHz

CCON            DATA            0D8H
CF              BIT             CCON. 7
CR              BIT             CCON. 6
CCF2            BIT             CCON. 2
CCF1            BIT             CCON. 1
CCF0            BIT             CCON. 0
```

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


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


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


            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR


            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            JNB     CF,CHECKCCF0
            CLR     CF                          ;Clear interrupt sign
            INC     CNT                         ; PCA Timing overflow times+1
CHECKCCF0:
            JNB     CCF0,ISREXIT
            CLR     CCF0
            MOV     COUNT0,COUNT1               ; Back up the last captured value
            MOV     COUNT0+1,COUNT1+1
            MOV     COUNT0+2,COUNT1+2
            MOV     COUNT1,CNT                  ;Save the captured value this time
            MOV     COUNT1+1,CCAP0H
            MOV     COUNT1+2,CCAP0L
            CLR     C                           ; Calculate the capture difference between the two times
            MOV     A,COUNT1+2
            SUBB    A,COUNT0+2
            MOV     LENGTH+2,A
```

```
                MOV     A,COUNT1+1
                SUBB    A,COUNT0+1
                MOV     LENGTH+1,A
                MOV     A,COUNT1
                SUBB    A,COUNT0
                MOV     LENGTH,A                    ;LENGTH What is saved is the captured pulse width
ISREXIT:
                POP     PSW
                POP     ACC
                RETI

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

                CLR     A
                MOV     CNT,A                       ;User variable initialization
                MOV     COUNT0,A
                MOV     COUNT0+1,A
                MOV     COUNT0+2,A
                MOV     COUNT1,A
                MOV     COUNT1+1,A
                MOV     COUNT1+2,A
                MOV     LENGTH,A
                MOV     LENGTH+1,A
                MOV     LENGTH+2,A

                MOV     CCON,#00H
                MOV     CMOD,#09H                   ;PCA The clock is the system clock, Enable Timing interruption
                MOV     CL,#00H
                MOV     CH,#0H
                MOV     CCAPM0,#11H                 ;PCA Module 0 For 16 Bit capture mode (falling edge capture)
;               MOV     CCAPM0,#21H                 ;PCA module 0 for 16 Bit capture mode (rising edge capture)
;               MOV     CCAPM0,#31H                 ;PCA module 0 for 16 Bit capture mode (edge capture)
                MOV     CCAP0L,#00H
                MOV     CCAP0H,#00H
                SETB    CR                          ;Start PCA Timer
                SETB    EA

                JMP     $

                END
```

18.4.3 PCA realize 16 Bit software timing

C Language code

```
// The test operating frequency is 11.0592MHz

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

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

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

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

sbit P10        =   P1^0;

unigned int     value;

void PCA_Isr() interrupt 7
{
        CCF0 = 0;
        CCAP0L = value;
        CCAP0H = value >> 8;
        value += T50HZ;
        P10 = ! P10;                            // 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;
        CCON = 0x00;
        CMOD = 0x00;                                    //PCA The clock is the system clock/12
        CL = 0x00;
        CH = 0x00;
        CCAPM0 = 0x49;                                  //PCA The module is 0 Bit timer mode

        value = T50HZ;
        CCAP0L = value;
        CCAP0H = value >> 8;
        value += T50HZ;
        CR = 1;                                         //Start PCA Timer
        EA = 1;
        while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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


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


P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
```

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

                ORG             0000H
                LJMP            MAIN
                ORG             003BH
                LJMP            PCAISR

                ORG             0100H
PCAISR:
                PUSH            ACC
                PUSH            PSW
                CLR             CCF0
                MOV             A,CCAP0L
                ADD             A,#LOW T50HZ
                MOV             CCAP0L,A
                MOV             A,CCAP0H
                ADDC            A,#HIGH T50HZ
                MOV             CCAP0H,A
                CPL             P1.0                    ; The flashing frequency of the test port is , 50Hz
                POP             PSW
                POP             ACC
                RETI

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

                MOV             CCON,#00H
                MOV             CMOD,#00H               ;PCA The clock is the system clock/12
                MOV             CL,#00H
                MOV             CH,#0H
                MOV             CCAPM0,#49H             ;PCA module 0 for 16 Bit timer mode
                MOV             CCAP0L,#LOW T50HZ
                MOV             CCAP0H,#HIGH T50HZ
                SETB            CR                      ;Start PCA Timer
                SETB            EA

                JMP             $

                END
```

18.4.4 PCA Output high-speed pulse

C Language code

```
// The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins.h"
#define T38K4HZ (11059200L / 2 / 38400)
sfr CCON = 0xd8;
sbit CF = CCON^7;
sbit CR = CCON^6;
sbit CCF2 = CCON^2;
sbit CCF1 = CCON^1;
sbit CCF0 = CCON^0;
sfr CMOD = 0xd9;
sfr CL = 0xe9;
sfr CH = 0xf9;
sfr CCAPM0 = 0xda;
sfr CCAP0L = 0xea;
sfr CCAP0H = 0xfa;
sfr PCA_PWM0 = 0xf2;
sfr CCAPM1 = 0xdb;
sfr CCAP1L = 0xeb;
sfr CCAP1H = 0xfb;
sfr PCA_PWM1 = 0xf3;
sfr CCAPM2 = 0xdc;
sfr CCAP2L = 0xec;
sfr CCAP2H = 0xfc;
sfr PCA_PWM2 = 0xf4;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;
unsigned int value;

void PCA_Isr() interrupt 7
{
        CCF0 = 0;
        CCAP0L = value;
        CCAP0H = value >> 8;
        value += T38K4HZ;
}
```

```
void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;
        CCON = 0x00;
        CMOD = 0x08;                                    //PCA The clock is the system clock
        CL = 0x00;
        CH = 0x00;
        CCAPM0 = 0x4d;                                  //PCA The module 0 is Bit timer mode and enable pulse output

        value = T38K4HZ;
        CCAP0L = value;
        CCAP0H = value >> 8;
        value += T38K4HZ;
        CR = 1;                                         //Start PCA Timer
        EA = 1;
        while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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


T38K4HZ         EQU             90H                             ;11059200/2/38400
```

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

                ORG             0000H
                LJMP            MAIN
                ORG             003BH
                LJMP            PCAISR

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

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

                MOV             CCON,#00H
                MOV             CMOD,#08H                       ;PCA The clock is the system clock
                MOV             CL,#00H
                MOV             CH,#0H
                MOV             CCAPM0,#4DH                     ;PCA module 0 for 16 Bit timer mode and enable pulse output
                MOV             CCAP0L,#LOW T38K4HZ
                MOV             CCAP0H,#HIGH T38K4HZ
                SETB            CR                              ;Start PCA Timer
                SETB            EA

                JMP             $
```

```
END
```

18.4.5 PCA Extended external interrupt

C Language code

```
// The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
sfr     CCON        =   0xd8;
sbit CF             =   CCON^7;
sbit CR             =   CCON^6;
sbit CCF2           =   CCON^2;
sbit CCF1           =   CCON^1;
sbit CCF0           =   CCON^0;
sfr CMOD            =   0xd9;
sfr CL              =   0xe9;
sfr CH              =   0xf9;
sfr CCAPM0          =   0xda;
sfr CCAP0L          =   0xea;
sfr CCAP0H          =   0xfa;
sfr PCA_PWM0        =   0xf2;
sfr CCAPM1          =   0xdb;
sfr CCAP1L          =   0xeb;
sfr CCAP1H          =   0xfb;
sfr PCA_PWM1        =   0xf3;
sfr CCAPM2          =   0xdc;
sfr CCAP2L          =   0xec;
sfr CCAP2H          =   0xfc;
sfr PCA_PWM2        =   0xf4;
sfr P0M1            =   0x93;
sfr P0M0            =   0x94;
sfr P1M1            =   0x91;
sfr P1M0            =   0x92;
sfr P2M1            =   0x95;
sfr P2M0            =   0x96;
sfr P3M1            =   0xb1;
sfr P3M0            =   0xb2;
sfr P4M1            =   0xb3;
sfr P4M0            =   0xb4;
sfr P5M1            =   0xc9;
sfr P5M0            =   0xca;
sbit P10            =   P1^0;

void PCA_Isr() interrupt 7
{
        CCF0 = 0;
        P10 = ! P10;
}

void main()
```

```
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;
        CCON = 0x00;
        CMOD = 0x08;                          //PCA The clock is the system clock
        CL = 0x00;
        CH = 0x00;
        CCAPM0 = 0x11;                        //Extended external port Interrupt port for falling edge CCP0
//      CCAPM0 = 0x21;                        //Extended external port Interrupt port for rising edge
//      CCAPM0 = 0x31;                        //Extended external port Is the edge interrupt port
        CCAP0L = 0;
        CCAP0H = 0;
        CR = 1;                               //Start PCA Timer
        EA = 1;
        while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
```

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

                ORG             0000H
                LJMP            MAIN
                ORG             003BH
                LJMP            PCAISR

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

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

                MOV             CCON,#00H
                MOV             CMOD,#08H                       ;PCA The clock is the system clock
                MOV             CL,#00H
                MOV             CH,#0H
                MOV             CCAPM0,#11H                     ;Extended external port Interrupt port for falling edge CCP0
;               MOV             CCAPM0,#21H                     ;Extended external port Interrupt port for rising edge
;               MOV             CCAPM0,#31H                     ;Extended external port Is the edge interrupt port
                MOV             CCAP0L,#0
                MOV             CCAP0H,#0
                SETB            CR                              ;Start PCA Timer
                SETB            EA

                JMP             $

                END
```

Synchronous serial peripheral interface SPI 19

STC12H A high-speed serial communication interface is integrated inside the series of microcontrollers interface. SPI is a kind of full-duplex high-speed synchronization Communication bus. SPI STC12H Series integrated SPI The interface provides two operating modes: master mode and slave mode.

19.1 SPI Related registers

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

19.1.1 SPI Status register (SPSTAT)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF : SPI Interrupt flag.

When sending and receiving is complete / 1 After the bytes of data, the hardware automatically puts this location 1 and send the CPU Make an interrupt request. when SSIG Bit is set SS The change in the pin level makes the main of the device / When the slave mode changes, this flag will also be automatically set to the time by the hardware, due to 1, To mark a change in the device mode.

Note: This flag must be written to this bit by the user through software Clear to zero.

SPI WCOL : SPI Write the conflict flag.

When writing In the process of data transmission SPDAT When registering, the hardware puts this location. 1

Note: This flag must be written to this bit by the user through software to clear it.

19.1.2 SPI Control register (SPCTL) , SPI Speed control

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

SS SSIG : The pin Pin function control bit determines whether the device is a SS master or a slave 0 :

1 : Ignore Pin function, use SS MSTR Determine whether the device is a master or a slave

SPEN : SPI Enable control bit

0 : Closed functionSPI

1 : Enable function SPI

DORD : SPI Data bit transmission/Order of reception

: Send first/The high position of the received data (0 MSB)

: Send first/The low bit of the received data (LSB)

MSTR : device master/ Set the host mode from the mode selection bit :

If you SSIG=, then SS The pin must be high and set MSTR for 1

set the slave 1SSIG, You only need to set MSTR 1For (ignore SS The level of the pin)

mode :

Ruo SSIG=, then SS The pin must be low (with MSTR Position independent)

Ruo = 1SSIG, You only need MSTR 0For (ignore SS pinLevel)

CPOL : to set the clock polarity control SCLK

0 : Low when idle , SCLK The front clock edge is the rising edge, and the rear clock edge is the falling edge

1 : SCLK High when idle , SCLK . The front clock edge is the falling edge, and the rear clock edge is the rising edge.

CPHA : SPI Clock phase control

The pin is low and drives the first bit of data and is SCLK The rear clock edge changes the data, and the front clock edge samples : data SS

Must SSIG =) 0

1 : The data is in SCLK The front clock edge is driven, and the rear clock edge is sampled

SPR[1:0] : SPI Clock frequency selection

SPR[1:0] SCLK	frequency
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/32

19.1.3 SPI Data register (SPDAT)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI send/Receive data buffer.

Communication method 19.2

SPI **SPI** The communication methods usually have 3 Type: single master and single slave (one host device is connected to one slave device), Mutual master and slave (two devices connec

Connection, the device and each other are the host and slave), single master and multiple slaves (one host device is connected to multiple sl

19.2.1 Single master single slave

Two devices are connected, one of which is fixed as the host

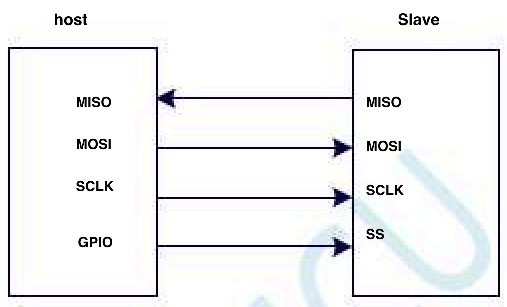
and the other is fixed as the slave. Host settings : SSIG 1 Set to , MSTR Set to, fixed as the host mode. The host can use any port to connect to the slave's

Pin, lower The foot can enable the slave SS

the slave setting of the slave : The pin is used as the chip selection signal of the slave.

SSSSIG 0 Set to , SS

The configuration diagram of the single master and single slave connection is shown below :



Single master single slave configuration

19.2.2 Mutual master and slave

The two devices are connected, and the master and slave are not fixed.

Setting method 1: When both devices are initialized, they are set to SSIG 0 MSTR Set to 1 Set to, and will SS The foot is set to a two-way port

The mode output is high. At this time, when both SS The host mode. When one of the devices needs to start the transm

devices do not ignore the input, they set to the output mode and the output is low, pulling the opponent's SS Foot, so the other one

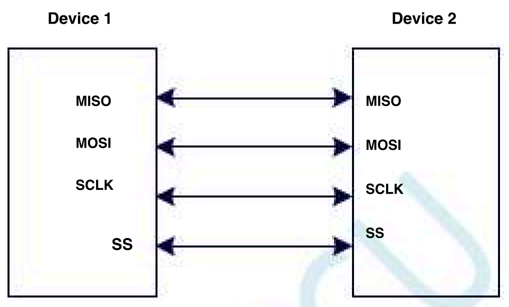
The device is forcibly set to slave mode.

Setting method 2: Both devices will set themselves to ignore when they are initialized SS Slave mode, Coming soon 1 Set to , MSTR SSIG

Set to. When one of the devices needs to start the transmission, first detect the level SS of the pin, if it is high at the time ,

Just set yourself to ignore the main mode, and you can transfer data. SS

The mutual master-slave connection configuration diagram is shown below :

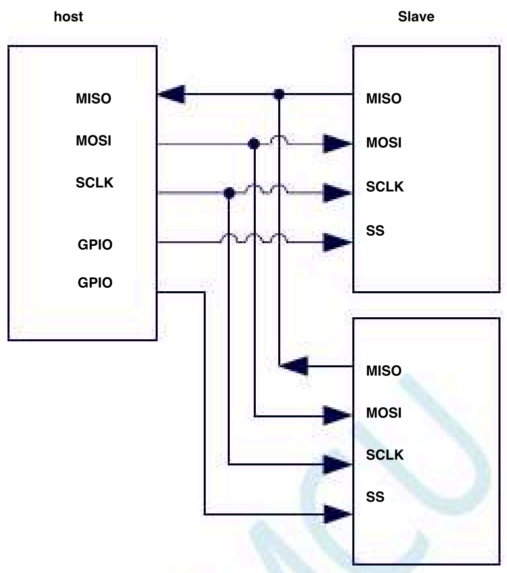


Mutual master and slave configuration

19.2.3 Single master and multiple slaves

Multiple devices are connected, one device is fixed as the host, and the other device is fixed as the slave. Host settings : SSIG 1 Set to , MSTR Set to, fixed as the host mode. The host can use any port to connect each pin separately, A slave SS and pull down the pin of one of the slaves to enable the corresponding slave equipment. SS Slave settings : SSIG 0 Set to , SS The pin is used as the chip selection signal of the slave.

The configuration diagram of the single master and multiple slave connection is shown below :



Single master and multiple slaves configuration

19.3 configuration SPI

Control			Communication				description
SPEN	bit SSIG	MSTR	SS	port MISO	MOSI	SCLK	
0	x	x	x	Input input		Input	Function, turn off SS/MOSI/MISO/SCLK SPI All are ordinary IO
1	0	0	0	output input high		input	Slave mode, and the slave mode
1	0	0	1	impedance input		input	is selected, but the slave mode
1	0	1→0	0	output	input	input	is not selected, it is not ignored and SS MSTR 1 For the host mode, it will be automatic When the pin is pulled down , MSTR when SS cleared by the hardware , The working mode will be passively set to slave mode
1	0	1	1	input	High impedance High impedance		Host mode, idle
					output	output	state host mode,
1	1	0	x	Output	input	input	active state slave
1	1	1	x	input	output	output	mode host mode

Precations for slave mode :

when CPHA = time 0, SSIG Must be (that is, cannot be ignored SS 0 At the beginning of each serial byte, the front pin must be pulled SS Low, and must be reset to high after the serial byte is sent. SS also sent) When the pin is low, it cannot Register performs write operation SPDAT Otherwise, it will result in a write conflict error. CPHA=0 and SSIG=1 The operation at the time is not defined.

when CPHA = time 1, SSIG Can be set (that is, feet can be ignored). if SSIG = 1 , 0 SS The foot can be protected between continuous tra Valid (that is, it has been fixed to a low level). This method is suitable for fixed single-master and single-slave systems.

Precautions for host mode :

In SPI In, the transmission is always started by the host. If SPI Enabled (SPEN=1) And when selected as the host, the host pair data SPI the register SPDAT Clock generator and data transmission. The write operation on the data write will start SPDAT The next half to one SPI After the bit time, the data will fall. Data written to the host register from SPDAT MOSI MOSI The foot is moved out and sent to the slave MOSI appear in the foot. Simultaneously, the data of the slave register is moved from the foot to the foot of the host. SPDAT MISO MISO

After transferring a byte , SPI The clock generator stops, and the transmission completion flag (If the interrupt is enabled, it will proc Give birth to an interruption. Master and slave The two shift registers can be seen as one CPU SPI 16 Bit cyclic shift register. When the data is from While the master shifts and transmits to the slave, the data also moves in in the opposite direction. This means that in a shift cycle, the data of the master and slave are exchanged with each other.

pass SS Change mode

if SPEN=1 , SSIG=0 and MSTR=1 , SPI Enable it as the host mode, and will SS The feet can be configured as input mode or quasi-d To the mouth mode. In this case, another host can drive the pin low, thereby selecting the device to Slave and send it to it SPI send data. To avoid competing for the bus , SPI The system clears the slave to zero , MOSI and MSTR SCLK Force it to input mode, and MISO Then it becomes the output mode, and at the same time SPSTAT of SPIF Flag position.

The user software must always be right The bit is detected, if the bit is passively cleared by a slave selecting an action, and the user wants to will SPI As the host, you must reset MSTR Bit, otherwise it will always be in slave mode.

Write conflict

SPI It is single buffered when sending and double buffered when receiving. In this way, new data cannot be written until the previous tr

Enter the shift register. When the data register is sent during the transmission process SPDAT When performing a write operation, The position will be set 1 To indicate the occurrence of data WCOL Write conflict error. In this case, the currently sent data continues to be sent, and the newly written data will be lost.

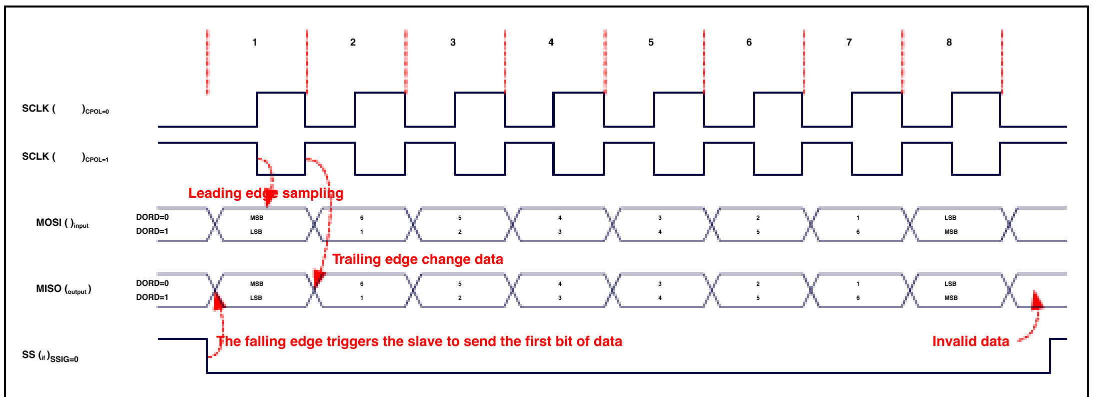
When a write conflict is detected on a host or slave, it is rare for a host to have a write conflict because the host has full control over da transmission. However, a write conflict may occur from the slave, because when the host starts the transmission, the slave cannot control it.

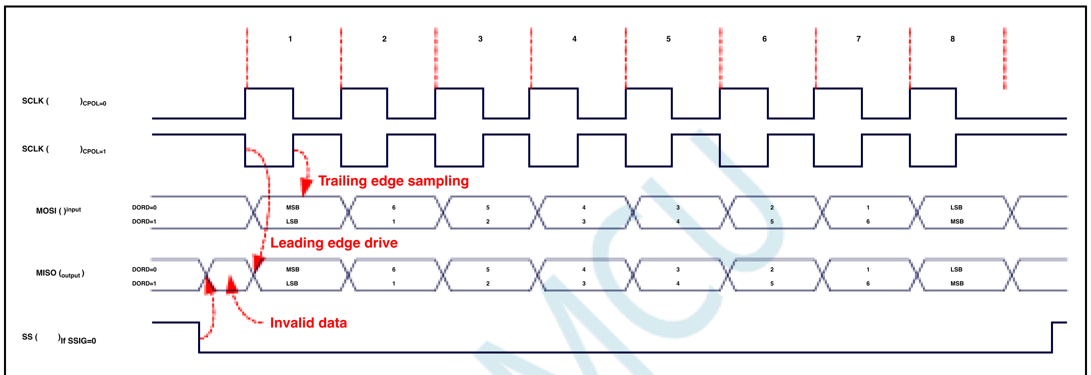
When receiving data, the received data is transferred to a parallel reading data buffer, which will release the shift register for the next data reception. But it must be removed from the data register before the next character is completely moved inRead out the received data, otherwise, the previous received data will be lost.

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

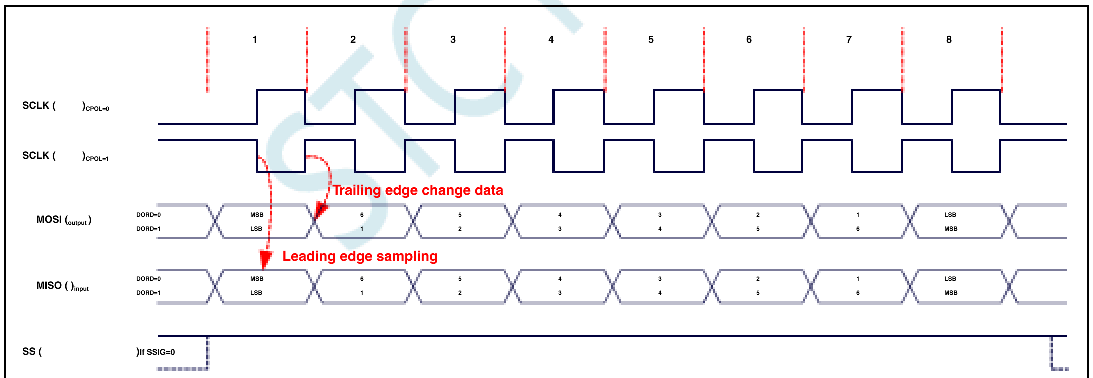
19.4 Data mode

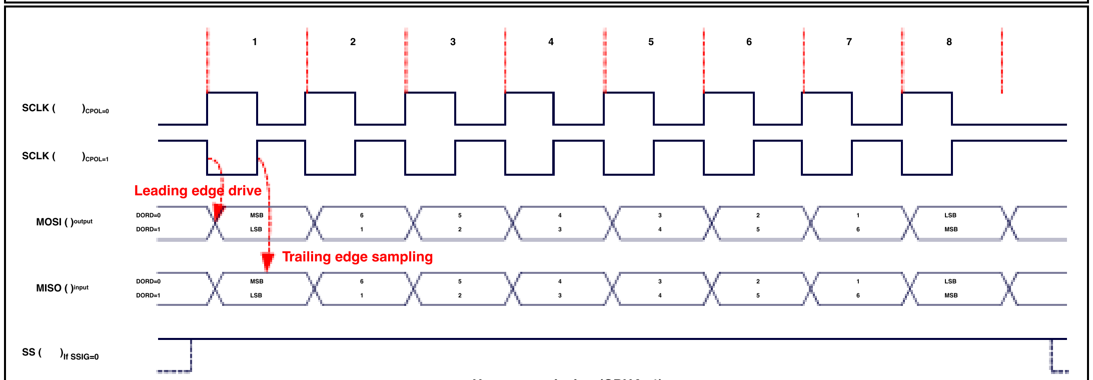
SPI Clock phase control bit CPHA Allows the user to set the clock edge when the data is sampled and changed. Clock polarity bit CPOL Let the user set the clock polarity. The legend below shows the different clock phases and polarity settings SPI Communication timing.


1 2 3 4 5 6 7 8
SCLK ( )CPOL=0
SCLK ( )CPOL=1
Leading edge sampling
MOSI ( )input
DORD=0 MSB 6 5 4 3 2 1 LSB
DORD=1 LSB 1 2 3 4 5 6 MSB
Trailing edge change data
MISO (output)
DORD=0 MSB 6 5 4 3 2 1 LSB
DORD=1 LSB 1 2 3 4 5 6 MSB
SS (if)SSIG=0
The falling edge triggers the slave to send the first bit of data
Invalid data
Slave transmission (CPHA=0)



Slave transmission () CPHA=1


1 2 3 4 5 6 7 8
SCLK ( )CPOL=0
SCLK ( )CPOL=1
Trailing edge change data
MOSI (output)
DORD=0 MSB 6 5 4 3 2 1 LSB
DORD=1 LSB 1 2 3 4 5 6 MSB
Leading edge sampling
MISO ( )input
DORD=0 MSB 6 5 4 3 2 1 LSB
DORD=1 LSB 1 2 3 4 5 6 MSB
SS ( )If SSIG=0
Host transmission (CPHA=0)



Host transmission (CPHA=1)

19.5 Sample program

19.5.1 SPI Single master single slave system host program (interrupt mode)

C Language code

```
//The test operating frequency 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 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 SS             =     P1^0;
sbit LED            =     P1^1;

bit busy;

void SPI_Isr() interrupt 9
{
        SPSTAT = 0xc0;                                  //Clear interrupt sign
        SS = 1;                                         //Pull up the slave SS pin
        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 Host mode
        SPSTAT = 0xc0;                                  //Clear interrupt sign
        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

```
;The test operating frequency is    11.0592MHz

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

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

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             004BH
                LJMP            SPIISR

                ORG             0100H
SPIISR:
                MOV             SPSTAT,#0C0H            ;Clear interrupt sign
                SETB            SS                      ;Pull up the slave SS pin
                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 Host mode
                MOV         SPSTAT,#0C0H                       ;Clear interrupt sign
                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
```

19.5.2 SPI Single master single slave system slave program (interrupt mode)

C Language code

```
//The test operating frequency 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 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        LED         =       P1^1;

void SPI_Isr() interrupt 9
{
        SPSTAT = 0xc0;                                          //Clear interrupt sign
        SPDAT = SPDAT;                                          // Pass the received data back to the host
        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 sign
        IE2 = ESPI;                                             //Enable SPI interrupt
        EA = 1;
        while (1);

}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

LED             BIT             P1.1

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             004BH
                LJMP            SPIISR

                ORG             0100H
SPIISR:
                MOV             SPSTAT,#0C0H                ;Clear interrupt sign
                MOV             SPDAT,SPDAT                 ; Pass the received data back to the host
                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 sign
                MOV             IE2,#ESPI                   ;Enable SPI interrupt
                SETB            EA

                JMP             $

                END
```

19.5.3 SPI Single master single slave system host program (query method)

C Language code

```
//The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
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         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 Host mode
        SPSTAT = 0xc0;                                  //Clear interrupt sign
        while (1)
        {
            SS = 0;                                     //Pull down the slave SS pin
            SPDAT = 0x5a;                               //Send test data
            while (! (SPSTAT & 0x80));                  //Query completion mark
            SPSTAT = 0xc0;                              //Clear interrupt sign
            SS = 1;                                     //Pull up the slave SS pin
            LED = ! LED;                                //Test port
        }
}
```

Assembly code

```
;The test operating frequency 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
```

```
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

                SETB            LED
                SETB            SS

                MOV             SPCTL,#50H                    Host mode ;Enable SPI
                MOV             SPSTAT,#0C0H                  ;Clear interrupt sign

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

                END
```

19.5.4 SPI Single master single slave system slave program (query method)

C Language code

```
//The test operating frequency 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 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 LED        =   P1^1;

void SPI_Isr() interrupt 9
{
        SPSTAT = 0xc0;                          //Clear interrupt sign
}

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

Assembly code

```
;The test operating frequency is 11.0592MHz
```

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

LED             BIT             P1.1

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

                ORG             0000H
                LJMP            MAIN

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

                MOV             SPCTL,#40H                      Slave mode ;Enable SPI
                MOV             SPSTAT,#0C0H                    ;Clear interrupt sign

LOOP:
                MOV             A,SPSTAT                        ;Query completion mark
                JNB             ACC. 7,$-2                      ;Clear interrupt sign
                MOV             SPSTAT,#0C0H
                MOV             SPDAT,SPDAT                     ; Pass the received data back to the host
                CPL             LED
                JMP             LOOP

                END
```

19.5.5 SPI Mutual master and slave system program (interrupt mode)

C Language code

```
// The test operating frequency 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 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 SS         =   P1^0;
sbit LED        =   P1^1;
sbit KEY        =   P0^0;

void SPI_Isr() interrupt 9
{
        SPSTAT = 0xc0;                          //Clear interrupt sign
        if (SPCTL & 0x10)                       //Host mode
        {
                SS = 1;                         //Pull up the slave pin SS
                SPCTL = 0x40;                   // Reset to slave standby
        }
        else                                    //Slave mode
        {
                SPDAT = SPDAT;                  // Pass the received data back to the host
        }
        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 Standby in slave mode
        SPSTAT = 0xc0;                              //Clear interrupt sign
        IE2 = ESPI;                                 //Enable SPI interrupt
        EA = 1;

        while (1)
        {
                if (!
                KEY) {                              //Wait for the button to trigger
                        SPCTL = 0x50;               //Enable SPI Host mode
                        SS = 0;                     //Pull down the slave SS
                        SPDAT = 0x5a;               //Send test data
                        while (! KEY);              //Wait for the button to release
                }
        }
}
```

Assembly code

```
;The test operating frequency 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

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             004BH
                LJMP            SPIISR

                ORG             0100H
SPIISR:
                PUSH            ACC
                MOV             SPSTAT,#0C0H                ;Clear interrupt sign
                MOV             A,SPCTL
                JB              ACC. 4,MASTER
```

```
SLAVE:
                MOV     SPDAT,SPDAT             ; Pass the received data back to the host
                JMP     ISREXIT
MASTER:
                SETB    SS                      ;Pull up the slave SS pin
                MOV     SPCTL,#40H              ; Reset to slave 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 Standby in slave mode
                MOV     SPSTAT,#0C0H            ;Clear interrupt sign
                MOV     IE2,#ESPI               ;Enable SPI interrupt
                SETB    EA

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

                END
```

19.5.6 SPI Mutual master and slave system program (query method)

C Language code

```
//The test operating frequency 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         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        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 Standby in slave mode
        SPSTAT = 0xc0;                                  //Clear interrupt sign
        while (1)
        {
            if (!
            KEY) {                                      //Wait for the button to trigger
                SPCTL = 0x50;                           //Enable SPI Host mode
                SS = 0;                                 //Pull down the slave SS
                SPDAT = 0x5a;                           //Send test data
                while (! KEY);                          //Wait for the button to release
            }
            if (SPSTAT & 0x80)
            {
                SPSTAT = 0xc0;                          //Clear interrupt
                if (SPCTL & 0x10)                       sign //Host mode
                {
                    SS = 1;                             //Pull up the slave pin SS
                    SPCTL = 0x40;                       // Reset to slave standby
                }
                else
                {                                           //Slave mode
                        SPDAT = SPDAT;                      // Pass the received data back to the host
                }
                LED = ! LED;                                //Test port
        }
    }
}
```

Assembly code

```
;The test operating frequency 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

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

            SETB        SS
            SETB        LED
            SETB        KEY
            MOV     SPCTL,#40H              ;Enable SPI Standby in slave mode
            MOV     SPSTAT,#0C0H            ;Clear interrupt sign

LOOP:
            JB      KEY,SKIP                ;Wait for the button to trigger
            MOV     SPCTL,#50H              ;Enable SPI Host mode
            CLR     SS                      ;Pull down the slave SS pin
            MOV     SPDAT,#5AH              ;Send test data
            JNB     KEY,$                   ;Wait for the button to release
SKIP:
            MOV     A,SPSTAT
            JNB     ACC.7,LOOP
            MOV     SPSTAT,#0C0H            ;Clear interrupt sign
            MOV     A,SPCTL
            JB      ACC.4,MASTER
SLAVE:
            MOV     SPDAT,SPDAT             ; Pass the received data back to the host
            CPL     LED
            JMP     LOOP
MASTER:
            SETB    SS                      ;Pull up the slave SS pin
            MOV     SPCTL,#40H              ; Reset to slave standby
            CPL     LED
            JMP     LOOP

            END
```

bus 20 I² C

SCL (Clock line) and STC12H Serial bus controller. The series of microcontrollers have an internal integrated one I² C use SDA C (Data cable) The two lines communicate synchronously and the port allocation of the communication envoy is ,

The single-chip microcomputer provides a SCL and SDA Switch to a different I/O STC8 Series of SDA switching mode, which can be time-sharing multiplexed. On the port, in order to facilitate users to treat a set of buses as mult

With standard I² C Compared with the agreement, the following two mechanisms are ignored :

Send a start signal (START) No arbitration after that

) No timeout detection when staying at low power

Clock signal (SCL For the output port, send a synchronous clock signal) and

STC12H The bus provides two operating modes: host mode (SCL I² C Series of

Slave mode (SCL is the input port, receiving a synchronous clock signal)

STC innovation When the serial bus controller is operating in slave mode The falling edge signal of the pin can wake up and enter

Electric mode STC I² SDA I² C The transmission speed is relatively fast , the first packet of data after the wake-up is generally incorrect)

C MCU° (Note: due to MCU

20.1 I² C Related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I² C Configuration register	FE80H	ENI2C	MSSL	MSSPEED[5:0]						0000,0000
I2CMSCR	I² C Host control register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I² C Host status register	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I² C Slave control register	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I² C Slave status register	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I² C Slave address register	FE85H	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I² C Data transmission register	FE86H									0000,0000
I2CRXD	I² C Data receiving register	FE87H									0000,0000
I2CMSAUX	I² C Host auxiliary control register	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

20.2 I² C Host mode

20.2.1 I2C Configuration register (I2CCFG), bus speed control

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

ENI2C : I² C Function enable control bit

0: Prohibited I² C function

1: Allow I² C function

MSSL : I² C Operating mode selection bit

0:Slave mode

1: Host mode

MSSPEED[5:0] : I² C Bus speed (number of waiting clocks) control, Bus speed= F_{OSC} / 2 / (MSSPEED * 2 + 4)

MSSPEED[5:0]	I2C Corresponding number of clocks
0	4
1	6
2	8
…	…
x	2x+4
…	…
62	128
63	130

Only when I² C When the module is operating in host mode, The waiting parameter for the parameter setting is only valid. This waiting par

The following signals of the host mode : MSSPEED

T_{SSTA} : The settling time of the starting signal (Setup Time of START)

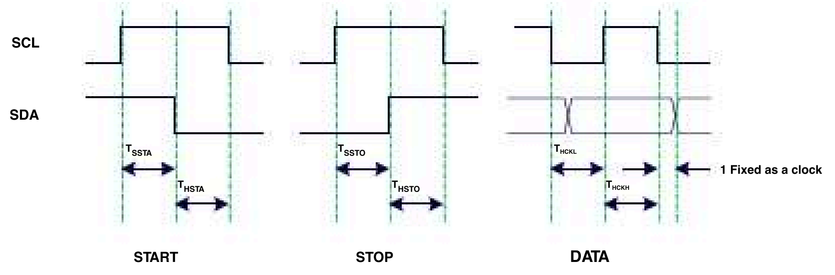
T_{HSTA} : The holding time of the starting signal (Hold Time of START)

T_{SSTO} : The settling time of the stop signal (Setup Time of STOP)

T_{HSTO} : The holding time of the stop signal (: Hold Time of STOP)

T_{HCKL} The low-level holding time of the clock signal (: Hold Time of SCL Low)

T_{HCKH} The high-level holding time of the clock signal (: Hold Time of SCL High)



1 Example: when T when , T_{SSTA} = MSSPEED = 10 T_{HSTO} = 24/FOSC

2 Example: when 24MHz The operating frequency is required T_{SSTO} =

MSSPEED = (24M / 400K / 2 - 4) / 2 = I2C 13

$_{HCKL}$ = T_{HSTA} = At bus speed ,

20.2.2 I2C Host control register (I2CMSCR)

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

EMSI : Host mode interrupt enable

0 control bit : Turn off host mode interrupt

1 : Allow host mode interrupt

MSCMD[3:0]: Host command

0000: Standby, no action.

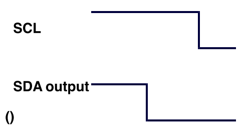
0001: Start command.

send START signal. If the current I² C The controller is in an idle state, that is, I2CMSST.7) for MSBUSY (0 when ,

Writing this command will cause the controller to enter a MSBUSY busy state, Status position, and start sending START1 Signal

and the hardware will automatically change the number; if the current The controller is busy START signal。 send START

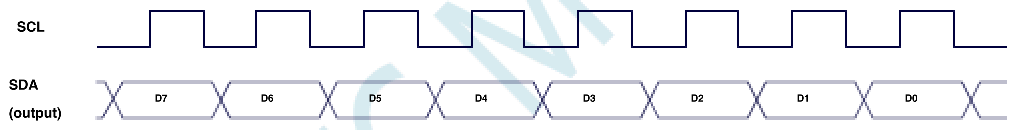
C The waveform is shown in the figure below Write this command to trigger sending



0010: Send data command.

After writing this command, The bus controller will be Generated on the SCL pin A clock, and will I2CTXD Data in the register

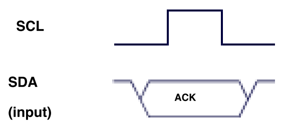
C Delivered bitwise On the SDA pin (send high-bit data first). The waveform of the transmitted data is shown in the figure below :



command.

ACK 0011: After receiving this command , I² C The bus controller SCL A clock is generated on the pin and will be SDA Data read on the port

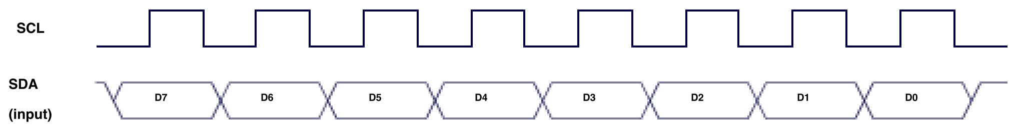
Save to will be MSACKI (I2CMSST. 1). receive ACK The waveform is shown in the figure below :



0100: Receive data commands.

After writing this I² C command, A clock is generated on the pin, and the slave bus controller will be SDA Data read on the port SCL

move left to I2CRXD Register (receive high-bit data first)。The waveform of the received data is shown in the figure below :

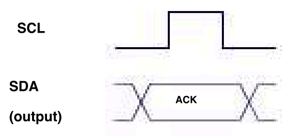


command.

ACK 0101: After sending this command , I² C The bus controller will be SCL A clock, and will be generated on the pin MSACKO (I2CMSST. 0) 1

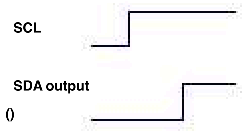
SDA port. send ACK The waveform is shown in the figure below :

C The data in is sent to



0110: Stop command.

Send STOP The bus controller starts sending signal. After the signal is sent, the hardware STOP automatically MSBUSY The status bit is cleared. STOP The waveform of the signal is shown in the figure below :



0111:Reserved.

1000:Reserved.

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

This command is a command 0001, command 0010, command 0011 A combination of three commands, after this command is issued, the contr

of these three commands.

1010 : Send data command + receive ACK command.

This command is a command 0010, command 0011 A combination of two commands, after issuing this command, the controller will execute th

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

This command is a command 0100, command 0101 A combination of two commands, after issuing this command, the controller will execute th

Note: The response signal returned by this ACK (0), not affected by MSACKO The impact of the bit。

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

This command is a command 0100, command 0101 A combination of two commands, after issuing this command, the controller will execute th

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

20.2.3 I2C Host auxiliary control register (I2CMSAUX)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA: In host mode I² C Automatic data transmission, allow bits

0: Automatic transmission is prohibited

1 : Enable automatic transmission

If the automatic sending function is enabled, when Execution is complete MCU After the write operation I² C The controller will automatically tou

Send" 1010 "Command, that is, automatically send data and receive ACK of the data register, the signal.

20.2.4 I2C Host status register (I2CMSST)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY: In host mode I² C Controller status bit (read-only bit)

0 :The controller is idle

1 : The controller is busy

when When the controller is in host mode, in the idle state, the transmission is completed START After the signal, the controller enters a busy state ,

C The busy state will be maintained until the successful delivery is completed STOP Signal, after which the state will return to the idle state again.

MSIF : The interrupt request bit (interrupt flag bit) of the host mode. When in host mode The controller completes the execution register and send I2CMSCR

An interrupt signal is generated after the command, and the hardware automatically sends this bit to the CPU request for an interrupt, After responding to the interrupt, Position required MSIF M

Must be cleared by software.

: In host mode, send "MSACKI 0011 "The order is here I2CMSCR The signal. data.

MSACKO : In host mode, after preparing the bit to be sent out, When sending ACK Received after the bit MSCMD

the controller will automatically read the data of this bit as ACK send to SDA。 ACK 0101 "The order is here I2CMSCR of MSCMD

20.3 I² C Slave mode

20.3.1 I2C Slave control register (I2CSLCR)

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

ESTAI : Received in slave mode START Signal interrupt permission bit

0 : Received when slave mode is disabled: interrupt occurs when the signal is interrupted.

1 Received when slave mode is enabled: START interrupt occurs when the signal is interrupted . After the

ERXI Received when slave mode is enabled byte of data, the interrupt permission bit is allowed.

0 : Interrupt occurs after receiving data when slave mode is disabled

: Interrupt occurs after receiving bytes of data when slave mode is enabled 1 1

: In slave mode, interrupt the 1 allowable bit after sending the completed byte of data in slave mode ETXI

0 : When the slave mode is disabled, an interrupt occurs after sending the completed data : When the slave mode is enabled, an interrupt occurs after sending the completed byte of data. 1 1

: Received in slave mode STOP ESTOI Signal interrupt permission bit

0 : Received when slave mode is disabled STOP An interrupt occurs when the signal is interrupted,

1 : received when slave mode is enabled STOP an interrupt occurs when the signal is interrupted

SLRST : Reset slave mode

20.3.2 I2C Slave status register (I2CSLST)

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

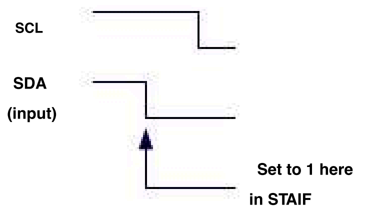
SLBUSY : In slave mode I² C Controller status bit (read-only bit)

0 :The controller is idle

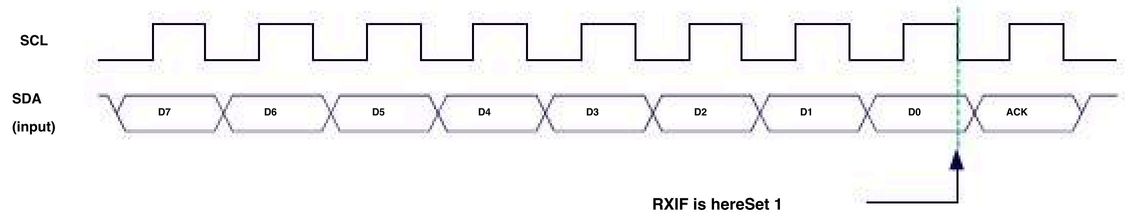
1 : The controller is busy

when When the controller is in slave mode, in the idle state, it receives the transmission from the host I²C After the signal, the controller will continue to det Subsequent device address data, if the device address is the same as the current one I2CSLADR When the slave address set in the register is the same, the control The device then enters a busy state, and the busy state will be maintained until the host successfully receives the STOP Signal, after which the status will be again transmission and returns to the idle state.

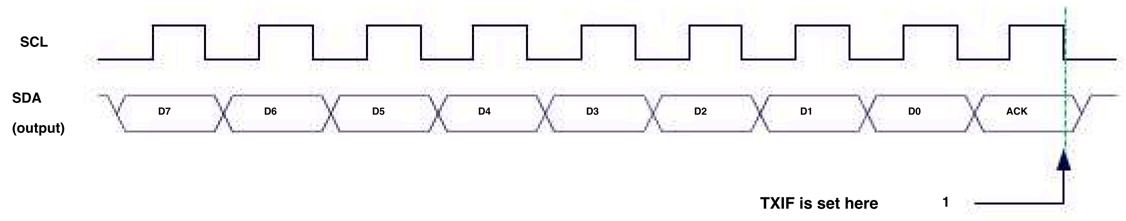
STAIF : The interrupt request bit after the signal is received in slave mode. Slave mode I²C The controller receives START After the signal , The hardware will automatically place this location and send a request to interrupt, after responding to the interrupt C STAIF The bit must be cleared by software. Set 1 STAIF The point in time is shown in the figure below :



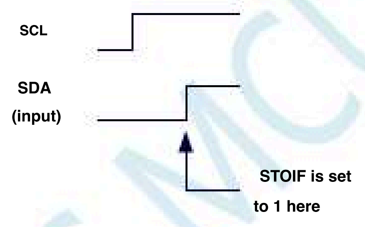
RXIF : When in slave mode, it I²C The interrupt request bit after the byte of data. Slave mode The controller receives 1 After bytes of data , is received in the first 8 When the falling edge of a clock, the hardware will automatically put this position and send it to the CPU Send a request interrupt, after responding to the interrupt RXIF Bit must Use the software to clear it to zero. RXIF The set time point is shown in the figure below :



TXIF : The interrupt request bit after sending the completed byte of data in slave mode. Slave mode The number of completed bytes sent by the contro
According to and after successfully receiving the bit signal, in the first When the falling edge of a clock, the hardware will automatically put this position ar
Request an interrupt, and the bit must be cleared by software after responding to the interrupt The setting point is shown in the figure below :

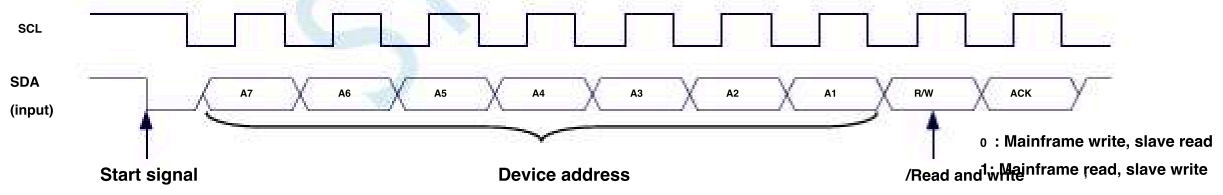


: Received in slave mode STOP STOIF The interrupt request bit after the signal. Slave mode The controller receives STOP After the signal, hard
The piece will automatically move this Send an request interrupt, after responding to the interrupt The bit must be cleared by software. Set STOIF 1
and to the point in time as shown in the figure below :



: Received in slave mode SLACKI ACK data.

SLACKO : In slave mode, prepare what will be sent out ACK signal.



20.3.3 I2C Slave address register (I2CSLADR)

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

I2CSLADR[7:1]:Slave device address

when I2C When the controller is in slave mode, the controller START receive After the signal, it will continue to detect the settings sent by the host

Prepare address data and read/Write a signal. When the device address sent by the host The slave equipment set in the ground I2CSLADR

the address, the controller will issue an interrupt request and the request will be processed. I2C Event, otherwise, if the device address is different ,

The controller continues to monitor, waits for the next start signal, and continues

MA to compare the next device address. :Slave device address comparison control

I2CSLADR[7:1] The same settings accept all device addresses 0: The device address must be the same as 1: Ignore I2CSLADR[7:1]

description : I2C Bus protocol regulations Up to can be mounted on the bus I2C Equipment (theoretical value), different Use different equipment I2C

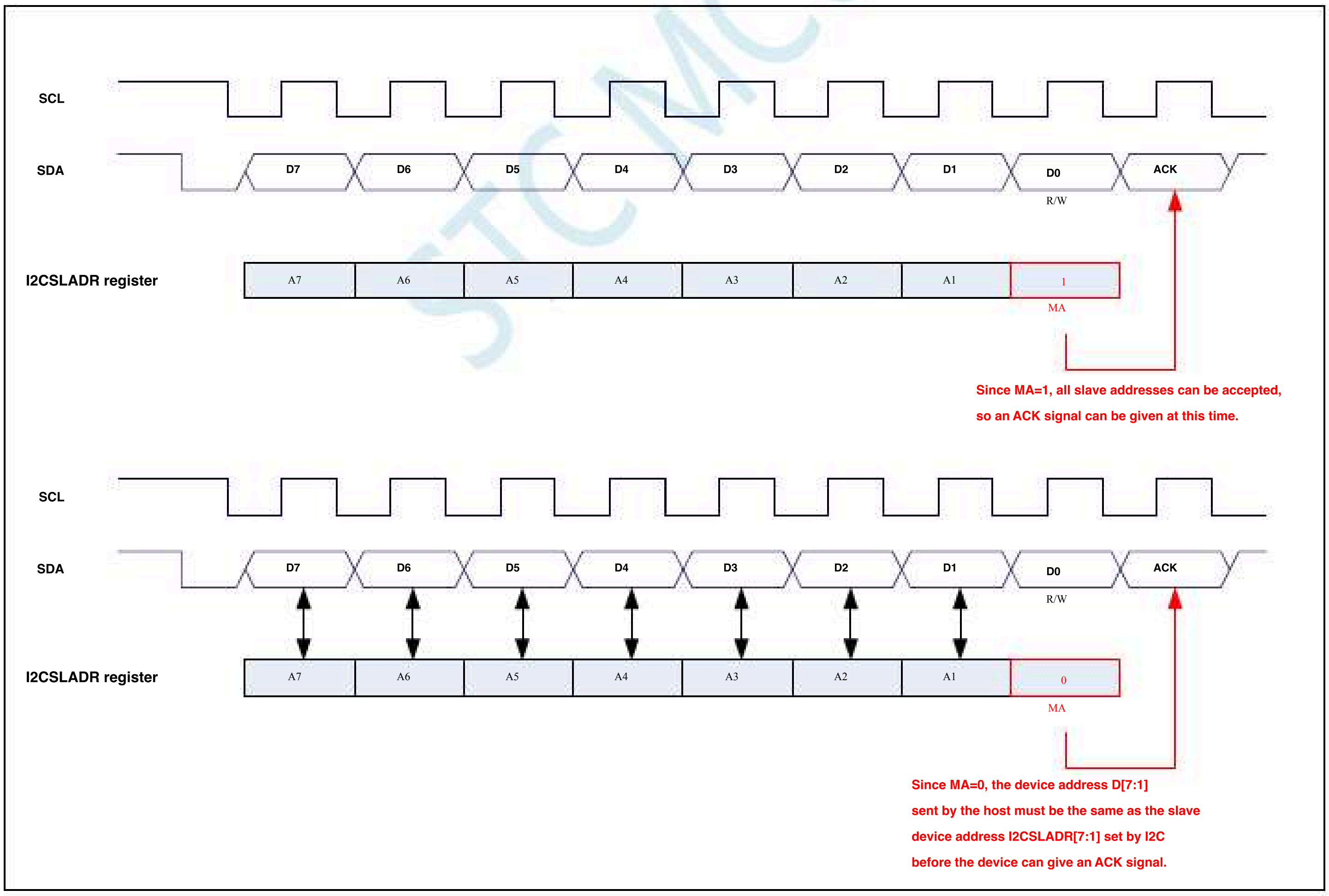
The address of the slave device is identified. After the host completes the start signal, the first data sent (DATA0) The height of 7

The bit is the address of the slave device (DATA0[7:1] When Device address), the lowest bit is the read and write signal. Device slave address to send it is a memory, it means MA (I2CSLADR. 0) for 1 Slave can accept All device addresses, any sent by the host at this time I2C

Device address, that is DATA0[7:1] For any value, the slave can respond. When the device slave address register I2C MA (I2CSLADR. 0)

For the time 0, The device address sent by the host DATA0[7:1] Must be the same as the device address of the slave I2CSLADR[7:1] You can only access this at the same time

Slave equipment


SCL
SDA
D7
D6
D5
D4
D3
D2
D1
D0
ACK
R/W
I2CSLADR register
A7
A6
A5
A4
A3
A2
A1
1
MA
Since MA=1, all slave addresses can be accepted,
so an ACK signal can be given at this time.
SCL
SDA
D7
D6
D5
D4
D3
D2
D1
D0
ACK
R/W
I2CSLADR register
A7
A6
A5
A4
A3
A2
A1
0
MA
Since MA=0, the device address D[7:1]
sent by the host must be the same as the slave
device address I2CSLADR[7:1] set by I2C
before the device can give an ACK signal.

20.3.4 I2C Data register (I2CTXD ， I2CRXD)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD Yes I² C Send data register, store the received data I² C Data

I2CRXD yes I² C register to be sent , store the received data I² C register data

20.4 Sample program

20.4.1 I2 C Host mode access AT24C256(Interrupt method)

C Language code

```
//The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
sfr                  P_SW2               =        0xba;
#define I2CCFG
#define I2CMSCR                                   (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSST                                   (*(unsigned char volatile xdata *)0xfe81)
#define I2CSLCR                                   (*(unsigned char volatile xdata *)0xfe82)
                                                  (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST                                   (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR                                  (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD                                    (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD                                    (*(unsigned char volatile xdata *)0xfe87)
sfr P1M1
sfr P1M0                                 =        0x91;
                                         =        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;
                                         =        0xca;
sfr P5M0
sbit SDA                                 =        P1^4;
                                         =        P1^5;
sbit SCL
bit busy;


void I2C_Isr() interrupt 24
{
        _push_(P_SW2);
        P_SW2 |= 0x80;
        if (I2CMSST & 0x40)
        {
        I2CMSST &= ~0x40;                                                   //Clear interrupt sign
        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 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;                     //Set up ACK signal
        busy = 1;
        I2CMSCR = 0x85;                     //send ACK command
        while (busy);
}

void SendNAK()
{
        I2CMSST = 0x01;                     //Set up 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 Host mode
        I2CMSST = 0x00;
        EA = 1;
        Start();
        SendData(0xa0);   //Send start command
        RecvACK();                              // Send device address+Write command
        SendData(0x00);
        RecvACK();                              // Send storage address high byte
        SendData(0x00);                         // Send storage address low byte
        RecvACK();
        SendData(0x12);                                    1//Write test data
        RecvACK();
        SendData(0x78);                         //Write test data
        RecvACK();
        Stop();                                 //Send stop command
        Delay();                                //Wait for the device to write data
        Start();
        SendData(0xa0);                         //Send start command 2
        RecvACK();                              // Send device address+Write command
        SendData(0x00);
        RecvACK();                              // Send storage address high byte
        SendData(0x00);
        RecvACK();                              // Send storage address low byte
        Start();
        SendData(0xa1);                         //Send start command
        RecvACK();                              // Send device address+Read command
        P0 = RecvData();
        SendACK();                                   1//Read data
        P2 = RecvData();                        //Read data
        SendNAK();                              //Send stop command 2
        Stop();
        P_SW2 = 0x00;
        while (1);

}
```

Assembly code

```
;The test operating frequency 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

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

                ORG             0000H
                LJMP            MAIN
                ORG             00C3H
                LJMP            I2CISR

                ORG             0100H
I2CISR:
                PUSH            ACC
                PUSH            DPL
                PUSH            DPH

                MOV             DPTR,#I2CMSST                   ;Clear interrupt sign
                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 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
RECVDATA:
                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                ;Set up 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                ;Set up 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 to complete
                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 up I2C The module is the host mode
                    MOV             DPTR,#I2CCFG
                    MOVX            @DPTR,A
                    MOV             A,#00000000B
                    MOV             DPTR,#I2CMSST
                    MOVX            @DPTR,A
                    SETB            EA

                    CALL            START                   ;Send start command send device
                    MOV             A,#0A0H                 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
                    MOV             A,#12H                  ;Write test data1
                    CALL            SENDDATA
                    CALL            RECVACK
                    MOV             A,#78H                  ;Write test data2
                    CALL            SENDDATA
                    CALL            RECVACK
                    CALL            STOP                    ;Send stop command

                    CALL            DELAY                   ;Wait for the device to write
                                                            data ;Send start command send
                    CALL            START                   device address+Write command
                    MOV             A,#0A0H                 ;
                    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 send device address +Read command
        MOV         A,#0A1H
        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
```

20.4.2 I²C Host mode access AT24C256(Inquiry method)

C Language code

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

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

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 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;                                          //Set up signal ACK
        I2CMSCR = 0x05;                                          //send ACK command
        Wait();
}

void SendNAK()
{
        I2CMSST = 0x01;                                          //Set up signal NAK
        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 Host 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 is lowbyte
        RecvACK();
        SendData(0x12);                                 1//Write test data
        RecvACK();
        SendData(0x78);                                 //Write test data
        RecvACK();
        Stop();                                         //Send stop command
        Delay();                                        //Wait for the device to write data
        Start();                                        //Send start command 2
        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();                                1//Read data
        SendACK();
        P2 = RecvData();                                //Read data
        SendNAK();
        Stop();                                         //Send stop command 2
        P_SW2 = 0x00;
        while (1);
}
```

Assembly code

```
;The test operating frequency 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

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
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 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                    ;Set up 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                    ;Set up 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 sign
                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 up I2C The module is the host mode
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START                   ;Send start command send device
        MOV     A,#0A0H                 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
        MOV     A,#12H                  ;Write test data1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H                  ;Write test data2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP
                                        ;Send stop command
                                        ;Wait for the device to write
        CALL    DELAY                   data ;Send start command send

        CALL    START                   device address+Write command
        MOV     A,#0A0H                 ;
        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 send device
        MOV     A,#0A1H                 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
```

20.4.3 I^2C Host mode access PCF8563

C Language code

```
//The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins.h"
sfr         P_SW2             =     0xba;
#define I2CCFG
#define I2CMSCR                     (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSST                     (*(unsigned char volatile xdata *)0xfe81)
#define I2CSLCR                     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLST                     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLADR                    (*(unsigned char volatile xdata *)0xfe84)
#define I2CTXD                      (*(unsigned char volatile xdata *)0xfe85)
#define I2CRXD                      (*(unsigned char volatile xdata *)0xfe86)
                                    (*(unsigned char volatile xdata *)0xfe87)
sfr P1M1
sfr P1M0                      =     0x91;
sfr P0M1                      =     0x92;
sfr P0M0                      =     0x93;
sfr P2M1                      =     0x94;
sfr P2M0                      =     0x95;
sfr P3M1                      =     0x96;
sfr P3M0                      =     0xb1;
sfr P4M1                      =     0xb2;
sfr P4M0                      =     0xb3;
sfr P5M1                      =     0xb4;
sfr P5M0                      =     0xc9;
                              =     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 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;                                  //Set up signal ACK
        I2CMSCR = 0x05;                                  //send ACK command
        Wait();
}

void SendNAK()
{
        I2CMSST = 0x01;                                  //Set up signal NAK
        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 Host mode
        I2CMSST = 0x00;
        Start();                                        //Send start command
        SendData(0xa2);                                 // Send device address+Write command
        RecvACK();
        SendData(0x02);                                 //Send storage address
        RecvACK();
        SendData(0x00);                                 //Set the second value
        RecvACK();
        SendData(0x00);                                 //Set the minute value
        RecvACK();
        SendData(0x12);                                 //Set the 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 the second value
                SendACK();
                P2 = RecvData();                        //Read minute value
                SendACK();
                P3 = RecvData();                        //Read hourly value
                SendNAK();
                Stop();                                 //Send stop command

                Delay();
        }
}
```

Assembly code

```
;The test operating frequency 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

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
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 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        ;Set up 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        ;Set up 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 sign
```

```
                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 up I2C The module is the host mode
                MOV             DPTR,#I2CCFG
                MOVX            @DPTR,A
                MOV             A,#00000000B
                MOV             DPTR,#I2CMSST
                MOVX            @DPTR,A

                CALL            START                           ;Send start command send device
                MOV             A,#0A2H                         address+Write command
                CALL            SENDDATA                        ;
                CALL            RECVACK
                MOV             A,#002H                         ;Send storage address
                CALL            SENDDATA
                CALL            RECVACK
                MOV             A,#00H                          ;Set the second value
                CALL            SENDDATA
                CALL            RECVACK
                MOV             A,#00H                          ;Set the minute value
                CALL            SENDDATA
                CALL            RECVACK
                MOV             A,#12H                          ;Set the hour value
                CALL            SENDDATA
                CALL            RECVACK
                CALL            STOP                            ;Send stop command
```

```
LOOP:
            CALL        START               ;Send start command send device
            MOV         A,#0A2H             address +Write command
            CALL        SENDDATA
            CALL        RECVACK
            MOV         A,#002H             ;Send storage address
            CALL        SENDDATA
            CALL        RECVACK
            CALL        START               ;Send start command send device
            MOV         A,#0A3H             address +Read command
            CALL        SENDDATA
            CALL        RECVACK
            CALL        RECVDATA            ;Read the second value
            MOV         P0,A
            CALL        SENDACK
            CALL        RECVDATA            ;Read minute value
            MOV         P2,A
            CALL        SENDACK
            CALL        RECVDATA            ;Read hourly value
            MOV         P3,A
            CALL        SENDNAK
            CALL        STOP                ;Send stop command

            CALL        DELAY

            JMP         LOOP

            END
```

20.4.4 I²C Slave mode (interrupt mode)

C Language code

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

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

bit isda;                                                       //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;                                       START // If the processing is a repeated
        isda = 1;                                               // start signal, this setting must be made
        }
        else if (I2CSLST & 0x20)
        {
                I2CSLST &= ~0x20;                               //Deal with RECV event
                if (isda)
                {
                        isda = 0;                               //Deal with RECV Event ( RECV DEVICE ADDR )
                }
                else if (isma)
                {
                        isma = 0;                               //Deal with RECV Event ( RECV MEMORY ADDR )
                        addr = I2CRXD;
                        I2CTXD = buffer[addr];
                }
                else
                {
                        buffer[addr++] = I2CRXD;                //Deal with RECV Event ( RECV DATA )
                }
        }
        else if (I2CSLST & 0x10)
        {
                I2CSLST &= ~0x10;                               //Deal with SEND event
                if (I2CSLST & 0x02)
                {
                        I2CTXD = 0xff;                          //received NAK Then stop reading data
                }
                else
                {
                        I2CTXD = buffer[++addr];                //received ACK Then continue to read the data
                }
        }
        else if (I2CSLST & 0x08)
        {
                I2CSLST &= ~0x08;                               //Deal with 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;                                // setting the slave device address register I2CSLADR=0101_1010B
                                                        // I2CSLADR[7:1]=010_1101B,MA=0B° //
                                                        // Since the device address sent to the host must be the same as MA 0,
                                                        // I2CSLADR[7:1] Same to access this I2C Slave equipment.
                                                        // If the host needs to write data, it must be sent 5AH(0101_1010B)
                                                        // . If the host needs to read data, it must be sent 5BH(0101_1011B)
        I2CSLST = 0x00;
        I2CSLCR = 0x78;                                 // Enable slave mode interrupt
        EA = 1;

        isda = 1;                                       // User variable initialization
        isma = 1;
        addr = 0;
        I2CTXD = buffer[addr];

        while (1);
}
```

Assembly code

```
;The test operating frequency 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

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

                ORG             0000H
                LJMP            MAIN
                ORG             00C3H
                LJMP            I2CISR

                ORG             0100H
I2CISR:
                PUSH            ACC
                PUSH            PSW
                PUSH            DPL
                PUSH            DPH
                MOV             DPTR,#I2CSLST           ;Detect slave status
                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              ;Deal withSTART event
                MOVX            @DPTR,A
                SETB            ISDA
                JMP             ISREXIT

RXIF:
                ANL             A,#NOT 20H              ;Deal withRECV event
                MOVX            @DPTR,A
                MOV             DPTR,#I2CRXD
                MOVX            A,@DPTR
                JBC             ISDA,RXDA
                JBC             ISMA,RXMA
                MOV             R0,ADDR                 ;Deal withRECV Event ( RECV DATA )
                MOVX            @R0,A
                INC             ADDR
                JMP             ISREXIT

RXDA:
                JMP             ISREXIT                 ;Deal withRECV Event ( RECV DEVICE ADDR )
RXMA:
```

```
                MOV         ADDR,A              ;Deal with RECV Event ( RECV MEMORY ADDR )
                MOV         R0,A
                MOVX        A,@R0
                MOV         DPTR,#I2CTXD
                MOVX        @DPTR,A
                JMP         ISREXIT
TXIF:
                ANL         A,#NOT 10H          ;Deal with 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          ;Deal with 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 register I2CSLADR=0101_1010B
                                                ;namely I2CSLADR[7:1]=010_1101B,MA=0B°
                                                ;Since the device address sent to the host must be the same as MA 0,
                                                ;I2CSLADR[7:1] The same can access this host. Slave equipment.
                                                ;you need to write data, you have to send it 5AH(0101_1010B)
                                                ; If the host needs to read data, it must send it 5BH(0101_1011B)
                MOV         DPTR,#I2CSLADR
                MOVX        @DPTR,A
                MOV         A,#00000000B
                MOV         DPTR,#I2CSLST
                MOVX        @DPTR,A
```

```
        MOV     A,#01111000B            ;Enable slave mode interrupt
        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
```

20.4.5 I² C Slave mode (query method)

C Language code

```
//The test operating frequency is 11.0592MHz

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

sfr     P_SW2           =       0xba;
#define I2CCFG
#define I2CMSCR                 (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSST                 (*(unsigned char volatile xdata *)0xfe81)
#define I2CSLCR                 (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLST                 (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLADR                (*(unsigned char volatile xdata *)0xfe84)
#define I2CTXD                  (*(unsigned char volatile xdata *)0xfe85)
#define I2CRXD                  (*(unsigned char volatile xdata *)0xfe86)
                                (*(unsigned char volatile xdata *)0xfe87)
sfr P1M1
sfr P1M0                =       0x91;
sfr P0M1                =       0x92;
sfr P0M0                =       0x93;
sfr P2M1                =       0x94;
sfr P2M0                =       0x95;
sfr P3M1                =       0x96;
sfr P3M0                =       0xb1;
sfr P4M1                =       0xb2;
sfr P4M0                =       0xb3;
sfr P5M1                =       0xb4;
sfr P5M0                =       0xc9;
                        =       0xca;

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

bit isda;                                       //Device address flag
```

```
                isma; bit                                              //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;                                                 //        I2C //Enable Slave mode
        I2CSLADR = 0x5a;                                               // I2CSLADR=0101_1010B
                                                                       // setting the slave device address register
                                                                       //    I2CSLADR[7:1]=010_1101B,MA=0B° //
                                                                       //Since the device address sent to the host must be the same as MA 0,
                                                                       //I2CSLADR[7:1] Same to access this I2C Slave equipment.
                                                                       // If the host needs to write data, it must be sent 5AH(0101_1010B)
                                                                       // . If the host needs to read data, it must be sent. 5BH(0101_1011B)
        I2CSLST = 0x00;
        I2CSLCR = 0x00;                                                //Disable slave mode interruption

        isda = 1;                                                      //User variable initialization
        isma = 1;
        addr = 0;
        I2CTXD = buffer[addr];
        while (1)
        {
                if (I2CSLST & 0x40)
                {
                I2CSLST &= ~0x40;                                      START // If the Event processing is a repeated
                isda = 1;                                              // start signal, this setting must be made
                }
                else if (I2CSLST & 0x20)
                {
                        I2CSLST &= ~0x20;                              //Deal with RECV event
                        if (isda)
                        {
                                isda = 0;                              //Deal with RECV Event ( RECV DEVICE ADDR )
                        }
                        else if (isma)
                        {
                                isma = 0;                              //Deal with RECV Event ( RECV MEMORY ADDR )
                                addr = I2CRXD;
                                I2CTXD = buffer[addr];
                        }
                        else
                        {
                                buffer[addr++] = I2CRXD;               //Deal with RECV Event ( RECV DATA )
```

```
                    }
            }
            else if (I2CSLST & 0x10)
            {
                    I2CSLST &= ~0x10;                              //Deal with SEND event
                    if (I2CSLST & 0x02)
                    {
                            I2CTXD = 0xff;                         //received NAK Then stop reading data
                    }
                    else
                    {
                            I2CTXD = buffer[++addr];               //received ACK Then continue to read the data
                    }
            }
            else if (I2CSLST & 0x08)
            {
                    I2CSLST &= ~0x08;                              //Deal with STOP event
                    isda = 1;
                    isma = 1;
            }
    }
}
```

Assembly code

```
;The test operating frequency 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

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

                ORG             0000H
                LJMP            MAIN

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

                MOV             P_SW2,#80H

                MOV             A,#10000001B                ;Enable I2C Slave mode
                MOV             DPTR,#I2CCFG
                MOVX            @DPTR,A
                MOV             A,#01011010B                ; Set the slave device address register I2CSLADR=0101_1010B
                                                            ;namely I2CSLADR[7:1]=010_1101B,MA=0B°
                                                            ;Since the device address sent to the host must be the same as MA 0,
                                                            ;I2CSLADR[7:1] The same can access this host. If Slave equipment.
                                                            ;you need to write data, you have to send it 5AH(0101_1010B)
                                                            ; If the host needs to read data, it must send it 5BH(0101_1011B)
                MOV             DPTR,#I2CSLADR
                MOVX            @DPTR,A
                MOV             A,#00000000B
                MOV             DPTR,#I2CSLST
                MOVX            @DPTR,A
                MOV             A,#00000000B                ;Disable slave mode interruption
                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                  ;Deal with START event
                MOVX            @DPTR,A
                SETB            ISDA
```

```
            JMP         LOOP
RXIF:
            ANL         A,#NOT 20H          ;Deal with RECV event
            MOVX        @DPTR,A
            MOV         DPTR,#I2CRXD
            MOVX        A,@DPTR
            JBC         ISDA,RXDA
            JBC         ISMA,RXMA
            MOV         R0,ADDR             ;Deal with RECV Event ( RECV DATA )
            MOVX        @R0,A
            INC         ADDR
            JMP         LOOP
RXDA:
            JMP         LOOP                ;Deal with RECV Event ( RECV DEVICE ADDR )
RXMA:
            MOV         ADDR,A              ;Deal with RECV Event ( RECV MEMORY ADDR )
            MOV         R0,A
            MOVX        A,@R0
            MOV         DPTR,#I2CTXD
            MOVX        @DPTR,A
            JMP         LOOP
TXIF:
            ANL         A,#NOT 10H          ;Deal with 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          ;Deal with STOP event
            MOVX        @DPTR,A
            SETB        ISDA
            SETB        ISMA
            JMP         LOOP

            END
```

20.4.6 test I^{2}C The host code of the slave mode code

C Language code

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

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

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 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;                                         //Set up signal ACK
        I2CMSCR = 0x05;                                         //send ACK command
        Wait();
}

void SendNAK()
{
        I2CMSST = 0x01;                                         //Set up signal NAK
        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 Host 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
                                                     //Send start command
        Start();                                     //Send device address (010_1101B)+Write command (0B)
        SendData(0x5a);
        RecvACK();
        SendData(0x00);                              // Send storage address high byte
        RecvACK();
        Start();                                     //Send start command
        SendData(0x5b);                              //Send device (010_1101B)+Read command (1B) address
        RecvACK();
        P0 = RecvData();                             //Read data 1
        SendACK();
        P2 = RecvData();                             //Read data 2
        SendNAK();
        Stop();                                      //Send stop command

        P_SW2 = 0x00;

        while (1);
}
```

Assembly code

```
;The test operating frequency 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

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
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 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                    ;Set up 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                    ;Set up 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 sign
                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 up I2C The module is the host mode
                MOV             DPTR,#I2CCFG
                MOVX            @DPTR,A
                MOV             A,#00000000B
                MOV             DPTR,#I2CMSST
                MOVX            @DPTR,A

                CALL            START                   ;Send start command
                MOV             A,#5AH                  ;
                CALL            SENDDATA                (010_1101B)+Write command ;Send
                CALL            RECVACK                 device address ;Send storage address (0B)
                MOV             A,#000H
                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
                                                        ;Wait for the device to
                CALL            DELAY
                                                        write data ;Send start command
                CALL            START                   ;Send device address
                MOV             A,#5AH                  (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
```

21 16 Bit advanced PWM Timer, support quadrature encoder

STC12H The series of microcontrollers are integrated internally Bit advanced PWM Timer, divided into two sets of cycles can be different PWM Named separately PWMA Harmony channel 8 16 PWM1 and PWM2 , But it is easy to be confused with the name of the chi Therefore, it was changed to PWMA PWMB (The previous data sheet was named PWM/PWMA and can be configured as a group. /Symmetrical/ Dead zone con of PWM Or capture external signals, the second group) can be set separately. First group 4 Can be configured as a group PWM Output or capture external signals. PWM/PWMB

First group PWM/PWMA The clock frequency can be the system clock through the register PWMA_PSCRH enter and PWMA_PSCRL The clock after the line is divided by the division, the division value 65535 Any value between. PWM/PWMB The clock frequency can be system time can be the clock passing through the register and PWMB_PSCRH PWMB_PSCRL Second group For the clock after dividing by frequency, the division value can be 65535 Any value between value. Two groups PWM The clock frequency can be set independently.

First group PWM A channel (PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N) There is a timer /PWMA Output (complementary symmetrical channels Output) Capture and Comparison function, the second group 4 PWMB Each channel can be implemented with dead zones PWM can be set (PWM5、PWM6、PWM7、PWM8), each channel is also Can be implemented independently PWM Output, capture, and compare functions. Two groups , while The only difference between the timer is that the first set can output compl Symmetrical PWM the second group can only output single-ended PWM, Other functions are exactly the same. The following introduction to the advanced timer Take the first group as an example to illustrate.

When using the first group Timer PWM output PWM When the waveform is set, it can be enabled separately PWM1P/PWM2P/PWM3P/PWM4P Output, It can also be enabled separately and PWM1N/PWM2N/PWM3N/PWM4N output. For example: if the output is enabled separately, then PWM1N PWM1P can no longer be output and PWM1P PWM1N Form a set of complementary symmetrical outputs. The output of the channel can be indepen PWMA of 4 independently, unless it is set, for example: the PWM1P and PWM2N Output, can also be enabled separately PWM2N and PWM3N output. If needed first group can be enabled separately PWM Timer for capture function or measurement When measuring the pulse width, the input signal can only be input from t PWM1P/PWM2P/PWM3P/PWM4P Only the capture function and the pulse width measurement function are available.

Two groups of advanced PWM When the timer captures the external signal, it can choose to capture the rising edge or the falling edge. If necessary Capture the rising and falling edges, then the input signal can be connected PWM Enables channels at the same time the rising edge and the other to ca Just get the falling edge. Even more powerful is to connect the external input signal to two channels at the same time A When, the period value and occupation of the signal can b Empty ratio。

Three kinds of hardware PWM Compare:

Compatible with tradition 8051 PCA/CCP/PWM : Can output PWM Waveform, capture external input signals, and output high-speed pulses. Can be e bit/7 Out 6 bit/8 bit/10 Bit of PWM 6 Waveform, bit PWM The frequency of the waveform is PCA Module clock source frequency PWM /64 bit The frequency of the waveform is Module clock source frequency PCA /128; 8 bit PWM The frequency of the waveform is Module clock source frequency PCA bit /256 ; PWM The frequency of the waveform is the frequency of the module clock source 10 /1024 8 Capture external input signals, you can capture rising edges, falling edges, o When capturing the rising and falling edges.

STC8G Series of 15 Bit enhanced : PWM Waveform, no input capture function. External output The frequency of PWM And the duty cycle can be set arbitrarily. Through software intervention, multiple complementarities can be achieved Can only be exported externally PWM waveform. There is an external excepti PWM /Symmetrical/ The detection function with dead zone and the real-time trigger conversion function. ADC

Series of STC8H/STC12H Bit advanced PWM Timer : Is currently STC The most powerful PWM , Can output any frequency externally to 16 And any duty cycle PWM waveform. Output without software intervention. /Symmetrical/ With dead zone PWM waveform. Can capture external output The input signal can capture the rising and falling edges, or capture the rising and falling edges at the same time. When measuring the and duty cycle of the waveform can be measured at the same time. There is an orthogonal coding function, an external Conversion function, anomaly detectio

In the description below , PWMA Represents the first group PWM Timer , PWMB Represents the second group PWM Timer

The first Timer/PWMA Internal signal description

TI1 PWM Pin signal or PWM1P/PWM2P/PWM3P Different or later signals)

: External clock input signal (PWM1P1 Group advanced

IC1F TI1F : Digitally filtered After Signal

passing by TI1FP : After CC2P TI1

passing by TI1F_ED : TI1F The edge signal Signal after the edge detector TI1F

TI1FP1: After passing by CC1P Capture TI1F Signal

TI1FP2: After passing by CC2P input signal of the selected channel after TI1F signal

IC1: Pass CC1S the edge detector after the edge detector 1

OC1REF : The reference waveform output by the output channel (intermediate waveform)

OC1 : the main output signal of the channel (after polarity processing OC1REF CC1P Signal)

OC1N : The complementary output signal of the channel (after CC1NP After polarity treatment OC1REF Signal)

TI2 : External clock input signal (PWM2P2 Pin signal)

IC2F TI2F : Digitally filtered After TI2 signal

passing by TI2F_ED : TI2F edge signal

TI2FP: After passing by After the edge detector CC1P/CC2P signal

TI2FP1: After passing by CC1P After the edge detector TI2F Signal

TI2FP2: After passing by CC2P After the edge detector TI2F signal

IC2: Pass CC2S Captured input signal of the selected channel : the reference

OC2REF waveform output by the output channel (intermediate waveform) :

OC2 the main output signal of the channel (after polarity processing OC2REF CC2P2 Signal)

OC2N : The complementary output signal of the channel (after CC2NP After polarity treatment OC2REF Signal)

TI3 : External clock input signal (PWM3P3 Pin signal)

IC3F TI3F : Digitally filtered After TI3 signal

passing by TI3F_ED : TI3F edge signal

TI3FP: After passing by After the edge detector CC3P/CC4P signal

TI3FP3: After passing by CC3P After the edge detector TI3F Signal

TI3FP4: After passing by CC4P After the edge detector TI3F signal

IC3: Pass CC3S Captured input signal of the selected channel : the reference

OC3REF waveform output by the output channel (intermediate waveform) :

OC3 the main output signal of the channel (after polarity processing OC3REF CC3P Signal)

OC3N: channel 3 Complementary output signal (after CC3NP After polarity treatment OC3REF Signal)

TI4 : External clock input signal (PWM4P4 Pin signal)

IC4F TI4F : Digitally filtered After TI4 signal

passing by TI4F_ED : TI4F edge signal

TI4FP: After passing by After the edge detector CC3P/CC4P signal

TI4FP3: After passing by CC3P After the edge detector TI4F Signal

TI4FP4: After passing by After the edge detector TI4F signal

IC4: Pass CC4S CC4P Captured input signal of the selected channel : the

OC4REF reference waveform output by the output channel (intermediate waveform)

The main output signal (after CC4P OC4 : Channel After polarity treatment OC4REF Signal)

OC4N : Channel 4 Complementary output signal (after CC4NP After polarity treatment OC4REF Signal)

ITR1 : Internal trigger input signal

: Internal trigger input signal 2 ITR2

TRC: Fixed as TI1_ED

TRGI: After passing by Trigger input signal

TRGO: After passing by Trigger output signal after multiplexer MMS

ETR : External trigger input signal (PWMETI1 Pin signal)

ETRP: After passing by ETP Edge detector and After the divider ETR ETPS signal

ETRF: After passing by ETF digitally filtered ETRP signal

BRK: Brake input signal (PWMFLT)

CK_PSC: Prescaler clock, Input clock of prescaler PWMA_PSCR

CK_CNT : PWMA_PSCR Output clock of prescaler , PWM Timer clock

Timer/PWMB Internal signal description

TI5 PWM 2 Advanced group Pin signal or PWM5/PWM6/PWM7 Different or later signals)

: External clock input signal (

IC5F TI5F : After Digitally filtered signal PWM5 5

passing by TI5FP : After CC5P CC6P TI5

passing by TI5F_ED : TI5F The edge signal Signal after the edge detector TI5F

TI5FP5: After passing by CC5P Capture TI5F Signal

TI5FP6: After passing by CC6P Input signal of the selected channel after TI5F signal

IC5: Pass CC5S the edge detector after the edge detector 5

OC5REF : The reference 5 waveform output by the output channel (intermediate waveform)

OC5 : the main output signal of the channel (after polarity processing OC5REF 5 CC5P Signal)

TI6 : External clock input signal (PWM6 6 Pin signal)

IC6F TI6F : After Digitally filtered signal TI6

passing by TI6F_ED : TI6F edge signal

TI6FP: After passing by After the edge detector CC5P/CC6P signal

TI6FP5: After passing by CC5P After the edge detector TI6F Signal

TI6FP6: After passing by CC6P After the edge detector TI6F signal

IC6: Pass CC6S The channel selected to capture the input signal 6

OC6REF : The reference 6 waveform output by the output channel (intermediate waveform)

OC6 : The main 6 output signal of the channel (after CC6P After polarity treatment OC6REF Signal)

TI7 : External clock input signal (PWM7 7 Pin signal)

IC7F TI7F :Digitally filtered signal TI7

Jingover TI7F_ED : TI7F edge signal

TI7FP: After passing by After the edge detector CC7P/CC8P signal

TI7FP7: After passing by CC7P After the edge detector TI7F Signal

TI7FP8: After passing by CC8P After the edge detector TI7F signal

IC7: Pass CC7S The channel selected to capture the input signal 7

OC7REF : The reference 7 waveform output by the output channel (intermediate waveform)

OC7 : The main 7 output signal of the channel (after CC7P After polarity treatment OC7REF Signal)

TI8 : External clock input signal (PWM88 Pin signal)

IC8F TI8F :After :Digitally filtered signal TI8

passing by TI8F_ED : TI8F edge signal

TI8FP: After passing by After the edge detector CC7P/CC8P signal

TI8FP7: After passing by CC7P After the edge detector TI8F Signal

TI8FP8: After passing by CC8P After the edge detector TI8F signal

IC8: Pass CC8S Capture input signal of the selected channel

OC8REF: Output channel 8 Output reference waveform (intermediate waveform)

OC8 : The main output signal of the channel 8 (after CC8P After polarity treatment OC8REF Signal)

21.1 introduction

PWMA It consists of an automatic loading counter of bits, which is driven by a programmable prescaler. By a 16

PWMA Suitable for many different purposes :

Basic timing

Measure the pulse width of the input signal (input capture) to generate

an output waveform (output comparison) ,

PWM And single pulse mode) Interrupts corresponding to different events (capture,

comparison, Overflow, brake, trigger) and external signals (external clock, reset signal, trigger and enable signal) synchronization

Widely used in a variety of control applications, including those that require PWM intermediate alignment mode The application of this module supports complen

Output and dead time control. PWMA The clock source can be an internal clock or an external signal, which can be obtained by configuring th

Make a choice.

21.2 Main features

PWMA The characteristics include :

16 Position up, down, up/Automatically load the counter under

Allows the repetition counter of the timer register to be

updated after a specified number of counter cycles

16 Bit programmable (can be modified in real time) prescaler, the frequency division coefficient of the counter clock frequency is Any between

Numerical synchronization circuit, used to control the

timer using an external signal and the timer interconnection

Up to Independent channels can be configured as :

Input capture

-

Output

comparison- - PWM Output (edge or middle alignment mode)

PWM Six-step

output- single pulse

mode output -

Complementary output support on a channel with programmable dead time - 4

Brake input signal (PWMFLT) YesTo put the timer output signal in a

External trigger input pins (reset state or a determined state PWMETI)

events that generate interrupts include :

- Update: The counter overflows upward/Overflow downwards, the counter is initialized (through software or internal/External

- trigger) Trigger event (counter start, stop, initialize, or by internal/External trigger count)

- Input capture,

- external interrupt for measuring pulse width

- Output comparison
- brake signal input

21.3 Time base unit

PWMA The time base unit includes:

16 bit upward/The downward counter

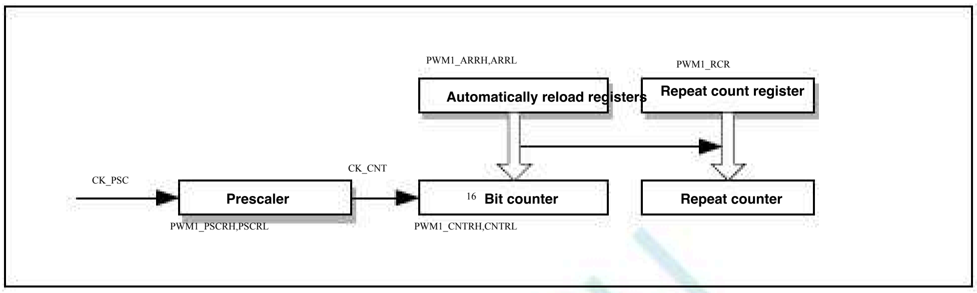
bit automatically reloads the register

16

Repeat counter

Prescaler

PWMA Time base unit



16 The bit counter, prescaler, automatic reload register, and repeat counter register can all be read and written by software. automatic The overload register consists of a preload register and a shadow register.

Can be written in two modes to automatically reload the register :

Automatic preload is enabled (PWMA_CR1 Register of ARPE Position is)1。 In this mode, write to the auto-reload register The data will be saved in the preload register and in the next update event (UEV) When transferred to the shadow register.

Automatic pre-loading has been disabled (PWMA_CR1 Register of ARPE Position is)0。 In this mode, write to the auto-reload register The data will be written to the shadow register immediately.

Update the conditions under which the event is generated :

The counter overflows up or down.

The software is set PWMA_EGR Register UG bit.

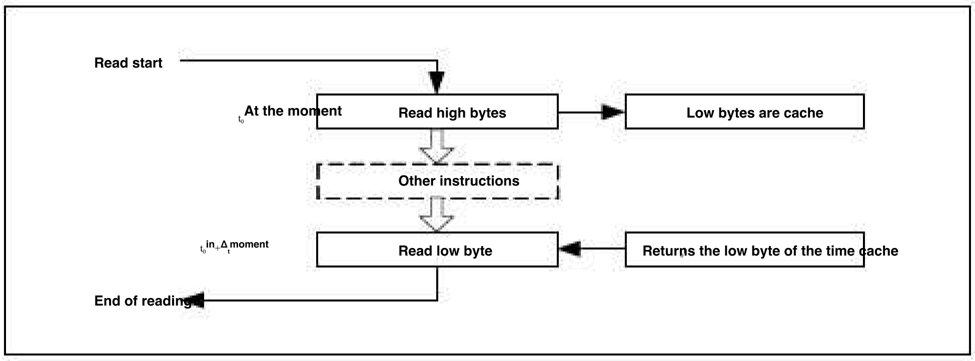
clock/The trigger controller generates a trigger event.

When the preload is enabled (ARPE=1) , If an update event occurs, preload the value in the register (PWMA_ARR) Will be written in the shadow register, and the bit PWMA_PSCR The value in the register will be written to the prescaler. Set Register of PWMA_CR1 UDIS will prohibit the update event (UEV). Output of the counter prescaler CK_CNT Drive the counter, and CK_CNT Only in PWMA_CR1 storage enable bit of the device (CEN) It is only valid when it is set. Note: The actual counter is in CEN The count does not start until one clock cycle after the bit is enabled.

Bit counter read and write 21.3.1 16

There is no cache for the operation of writing the counter, and it can be written at any time PWMA_CNTRH and PWMA_CNTRL Register, so In order to avoid writing the wrong value, it is generally recommended not to write a new value when the counter is running.

The operation of reading the counter has 8 Bit cache. The user must first read the high byte of the timer. After the user reads the high byte, the low byte Is automatically cache, and the cache data will be maintained until 16 The read operation of the bit data is completed.



21.3.2 16 bit PWMA_ARR Register write operation

The value in the preload register will be written to 16 Bit of PWMA_ARR In the register, this operation is completed by two instructions, each of which

1 Bytes. You must write the high byte first, and then the low byte.

The shadow register is locked when the high byte is written and remains until the low byte is finished.

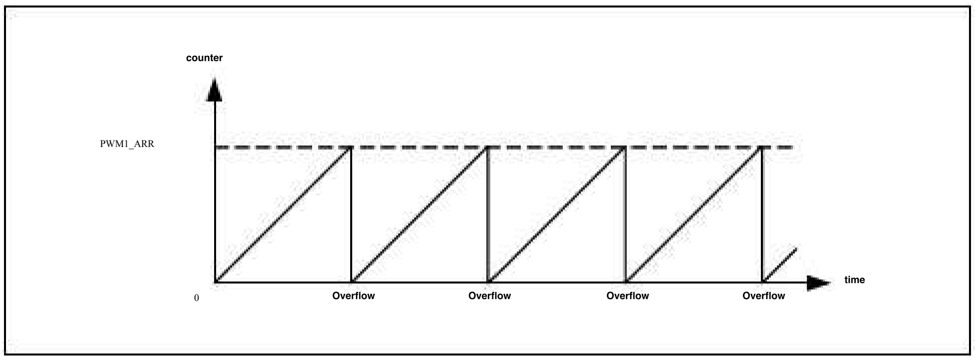
21.3.3 Prescaler

PWMA The prescaler is based on a 16 Bit register (PWMA_PSCR) Controlled 16 Bit counter. Because of this control. The memory has a buffer, so it can be changed at runtime. The prescaler can press the clock frequency of the counter between 1 to 65536 Divide by any value. The value of the prescaler is written from the preload register, and the shadow register that holds the currently used value is loaded when the low byte is written. Since two separate write operations are required to write 16 Bit register, so it is necessary to ensure that the high byte is written first. The value of the new p It will be adopted when the next update event arrives. The read operation of the register is completed by preloading the register. PWMA_PSCR

Counter frequency calculation formula : $f_{CK\_CNT} = f_{CK\_PSC} / (PSCR[15:0] + 1)$

21.3.4 Counting up mode

In count-up mode, the counter starts from 0 Count to a user-defined comparison value (PWMA_ARR The value of the register), and then re-from 0 Start counting and generate a counter overflow event, at this time if PWMA_CR1 Register of UDIS Bit Yes 0, it will produce a more New event (UEV).



Set the event by software or by using the trigger controller PWMA_EGR Register of UG Bits can also generate an update . Use software to set the register PWMA_CR1 UDIS Bit, you can disable the update event, so that you can avoid that the update ever When the register is updated, the shadow register is not updated. The UDIS update preload bit is cleared. But when an update event should occur , The counter will still be cleared, and the count of the prescaler will also be cleared 0 (But the value of the prescaler remains the same). In addition, if it is set PWMA_CR1 In the register URS Bit (select update request) set UG The bit will generate an update event UEV But the hardware is not set set UIF Flag (that is, no interrupt request is generated) 0. This is to avoid both update and capture when the counter is cleared in capture mode. break.

When an update event occurs, all registers are updated, and the hardware is based on URS Set the update flag at the same time (PWMA_SR

Register of UIF Bit) :

The auto-loaded shadow register is re-placed in the value of the pre-loaded register (PWMA_ARR).

The buffer of the prescaler is placed in the value of the pre-loaded register (PWMA_PSC).

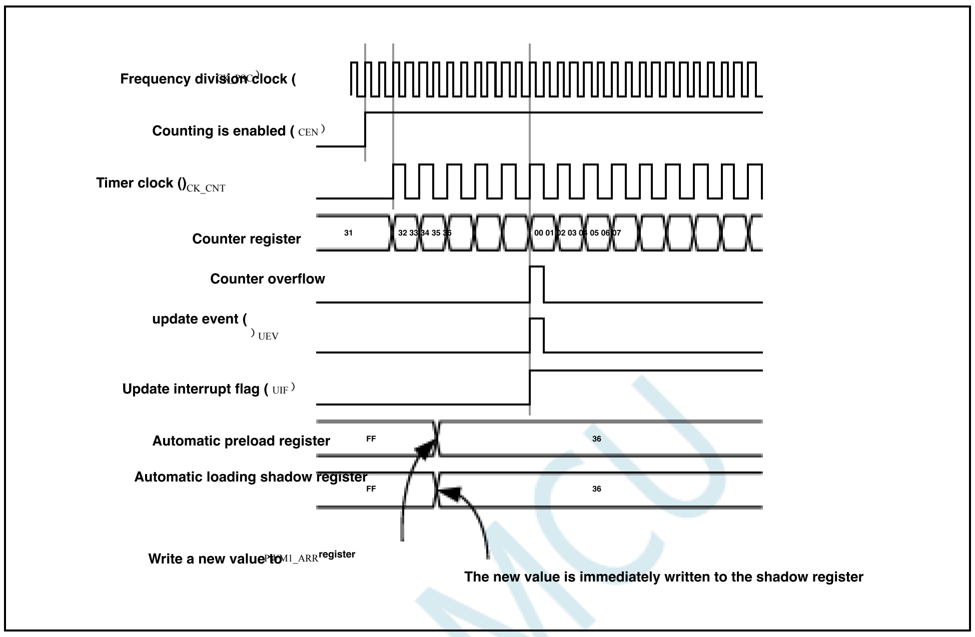
The figure below gives some examples to When, the action of the counter at different clock frequencies. The prescaler in t

2 illustrate when , therefore, the clock of the counter (CK_CNT) The frequency is the prescaler clock (CK_PSC) Half of the frequency. The automatic loading function is

Can (ARPE=0), so the counter reaches 0x36 When the counter overflows, the shadow register is updated immediately, and an update is g

event.

when ARPE=0 (ARR , Counter update when the prescaler is: not pre-loaded) 2

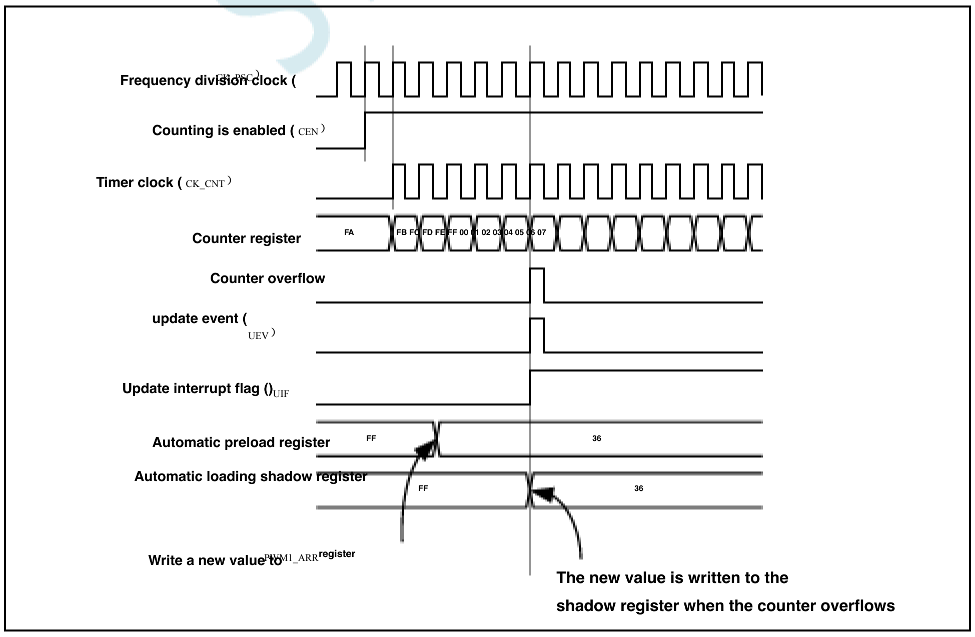

Frequency division clock (CK_PSC)
Counting is enabled (CEN)
Timer clock () CK_CNT
Counter register
31
32 33 34 35 36
00 01 02 03 04 05 06 07
Counter overflow
update event (UEV)
Update interrupt flag (UIF)
Automatic preload register
FF
36
Automatic loading shadow register
FF
36
Write a new value to PWM1_ARR register
The new value is immediately written to the shadow register

The prescaler of the figure below is 1, so and CK_CNT The frequency CK_PSC and consistent. Automatic overloading is enabled in the figure (ARPE=1) so

is generated when the counter reaches. 0x36 0xFF Will be written when it overflows, and an update event will be generated

at the same time.

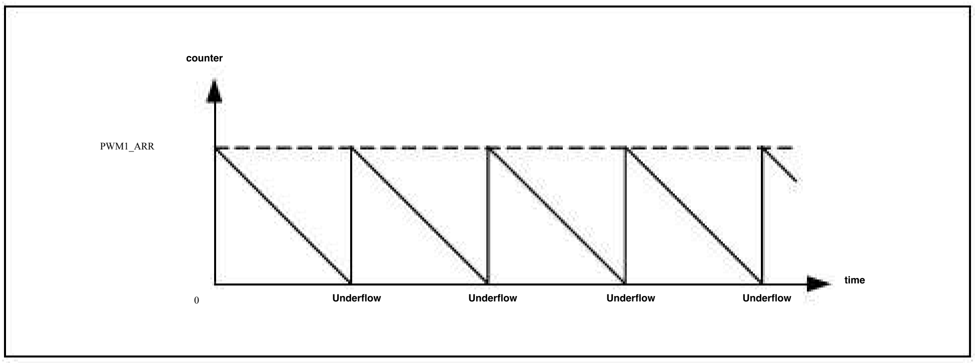
when ARPE=1(PWMA_ARR Preload), The counter update when the prescaler is : 1


Frequency division clock (CK_PSC)
Counting is enabled (CEN)
Timer clock (CK_CNT)
Counter register
FA
FB FC FD FE FF 00 01 02 03 04 05 06 07
Counter overflow
update event (UEV)
Update interrupt flag () UIF
Automatic preload register
FF
36
Automatic loading shadow register
FF
36
Write a new value to PWM1_ARR register
The new value is written to the shadow register when the counter overflows

21.3.5 Count down mode

In down mode, the counter is automatically loaded from the value (The value of the PWMA_ARR register) starts to count down to 0, and then from Automatically loaded values restart counting, and a counter overflow event is generated. if PWMA_CR1 Register of UDIS The position is cleared

In addition, an update event will be generated (UEV).



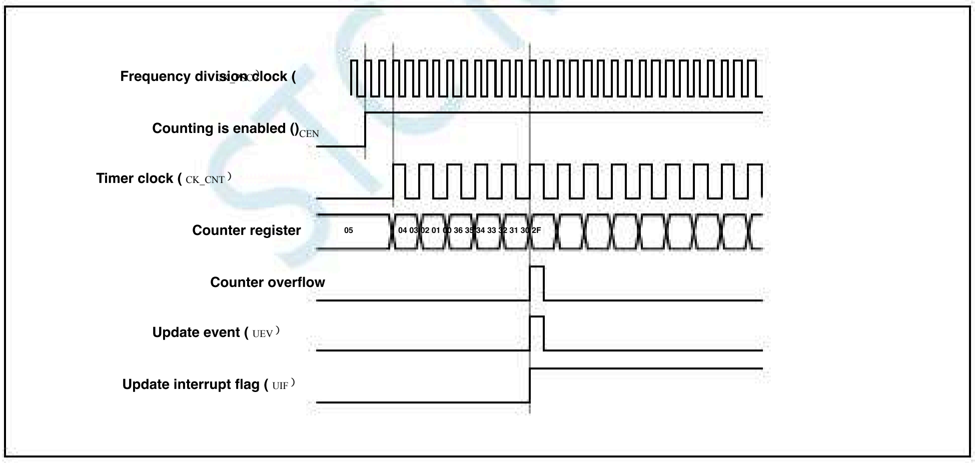
Set the event by software or by using the trigger controller Bits can also generate an update register PWMA_EGR . Setting the bits of the register can disable PWMA_CR1 UDIS UEV event. like thisCan avoid more changes when updating the pre-loaded re New shadow register. therefore UDIS No update event will be generated until the bit is cleared. However, the counter will still reopen from the curr Start counting, and the counter of the prescaler starts again from 0 Start (but the prescaler cannot be modified). In addition, if it is set PWMA_CR1 Bits in the register (select update request), set URS UG The bit will generate an update event UEV But not set UIF Logo (therefore No interruption) This is to avoid simultaneous update and capture interrupts when a capture event occurs and the counter is cleared. When a update event occurs, all registers are updated, and the hardware sets the update flag bit at the same time according to the bit (PWMA_SR URS send UIF Memory of Bit) :

The auto-loaded shadow register is re-placed in the value of the pre-loaded register (PWMA_ARR).

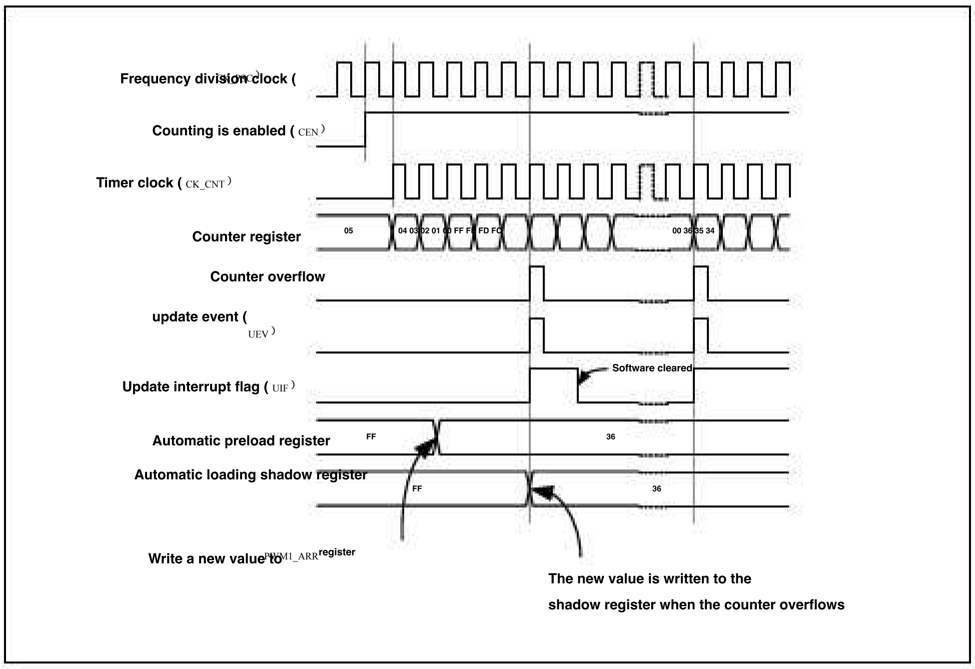
The buffer of the prescaler is placed in the value of the pre-loaded register (PWMA_PSC).

Here are some when PWMA_ARR=0x36 When, the chart of the counter at different clock frequencies. The figure below describes the cou Next, when the preload is not enabled, the new value will be written in the next cycle.

when ARPE=0 (ARR Not pre-loaded), The prescaler is 2 When the counter is updated :

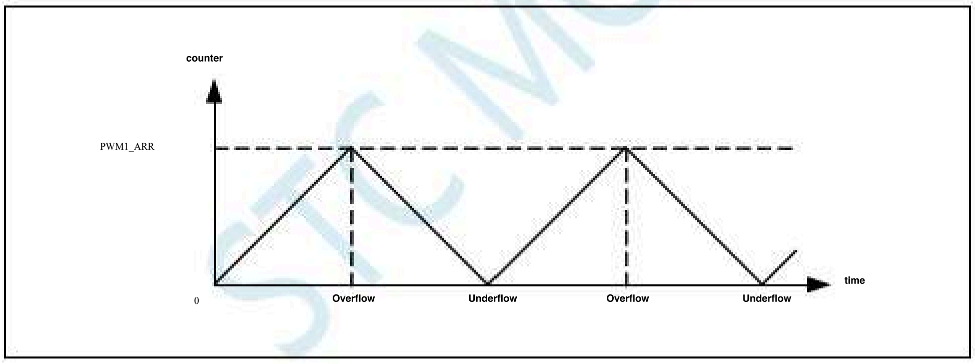

Frequency division clock (CK_PSC)
Counting is enabled (CEN)
Timer clock (CK_CNT)
Counter register
05
04 03 02 01 00 36 35 34 33 32 31 30 2F
Counter overflow
Update event (UEV)
Update interrupt flag (UIF)

when ARPE=1 (ARR Pre-loaded), the counter is updated when the prescaler is 1


Frequency division clock (CK_PSC)
Counting is enabled (CEN)
Timer clock (CK_CNT)
Counter register
05
04 03 02 01 00 FF FE FD FC
00 36 35 34
Counter overflow
update event (UEV)
Update interrupt flag (UIF)
Software cleared
Automatic preload register
FF
36
Automatic loading shadow register
FF
36
Write a new value to PWM1_ARR register
The new value is written to the shadow register when the counter overflows

21.3.6 Middle alignment mode (up/Count down)

In the central alignment mode, the counter starts from 0 The value of the register generates a counter overflow event, and then
Then count down from the value of the register to PWMA_ARR And a counter underflow event is generated; then recount from the beginning.
In this mode, cannot write PWMA_CR1 in DIR Direction bit. It is updated by the hardware and indicates the current counting directio



If the timer has a repeat counter, after the specified number of times is repeated (PWMA_RCR The value of) will be produced after the upward and downw
Birth update event (UEV). Otherwise, every time Overflow up and down will generate update events. Triggered by software or by using
Controller setting PWMA_EGR UG Bits can also generate an update event. At this time, the counter starts again from 0 Start counting
The number of registers, the prescaler is also counted again from the beginning. In the setting register UDIS 0 PWMA_CR1 Bits can be prohibited UEV event. This can
To avoid updating the shadow register when updating the pre-loaded register. Therefore, the position is cleared as UDIS 0 No update event will be generated before. however ,
The counter will still continue to count up or down based on the current automatically reloaded value. If the timer has a repetition counter, sin
the repetition register does not have a double buffer, the new repetition value will take effect immediately, so you need to be careful when mod
In the register, the setting bit will generate an update event but the flag is not set (therefore URS UG UEV UIF Bit (select update request)
No interruption) This is to avoid simultaneous update and capture interrupts when a capture event occurs and the counter is cleared.

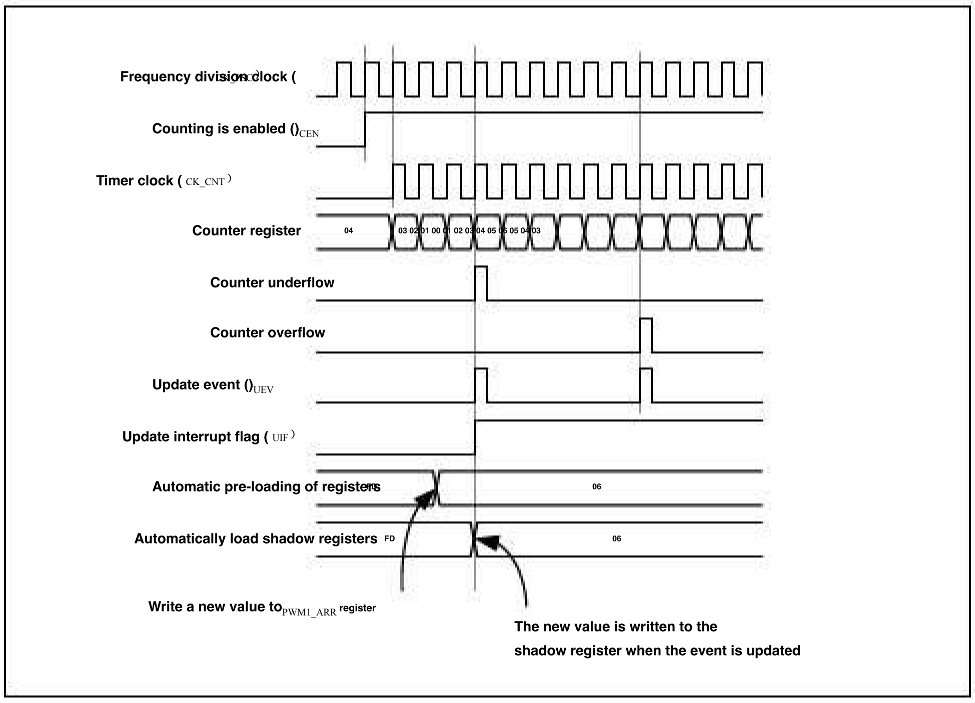
When an update event occurs, all registers URS Bit update flag bit (PWMA_SR In the register UIF are updated, the hardware is based on bits) :

The buffer of the prescaler is loaded as the preloaded value (PWMA_PSCR)

the current autoloading register is updated to the preloaded value (PWMA_ARR)

It should be noted that if an update occurs due to a counter overflow, the automatic reload register will be updated before the counter is reloaded, so the next cycle is the expected value (the counter is loaded as a new value)

The following are some examples of the operation of counters at different clock frequencies: the internal clock division factor is 1, PWMA_ARR=0x6, ARPE =


Frequency division clock (CK_PSC)
Counting is enabled (CEN)
Timer clock (CK_CNT)
Counter register
04
03 02 01 00 01 02 03 04 05 06 05 04 03
Counter underflow
Counter overflow
Update event (UEV)
Update interrupt flag (UIF)
Automatic pre-loading of registers FD
06
Automatically load shadow registers FD
06
Write a new value to PWM1_ARR register
The new value is written to the shadow register when the event is updated

Tips for using the central alignment mode :

When the central alignment mode is activated, the counter will follow the original upward/The downward configuration counts. In PWMA_CR1 , the bits in the memory will determine whether the counter counts up or down. In addition, the software cannot be modified at the DIR Bit sum CMS The value of the bit.

It is not recommended to write the value of the counter while the counter is counting in the central alignment mode, which will lead to unforeseen consequences. Specifically :

- When a value larger than the auto-loaded value is written to the counter (PWMA_CNT>PWMA_ARR), but the counter

The counting direction does not change. For example, the counter has overflowed upwards, but the counter still counts upwards. Wrote to the counter 0 PWMA_ARR - The value, but the update event does not occur.

The safe way to use the counter in the central alignment mode is to use the software (set) before starting the counter. PWMA_EGR storage

The bit of the counter) generates an update event, and does not modify the value of the counter when the counter is counted。 UG

21.3.7 Repeat counter

The time base unit explains the counter upward/Update the event when it is generated, but in fact it How is it generated downwards (UEV

It is generated when the value of the counter is reached. This feature produces can only be very useful in repeating signals. PWM

This means that at each When the count overflows or underflows, the data is transferred from the preload register to the shadow register (PWMA_ARR automatic reload, enter the register , PWMA_CCRx), Pre-loaded registers, as well as capture in comparison mode/Compare register PWMA_PSCR N

Yes PWMA_RCR Repeat the value in the count register.

The repeat counter decrements when any of the following conditions are true :

Count up mode Every time the counter overflows up , count

down mode every time the counter overflows down

Each overflow and each underflow in central alignment mode. PWM The maximum cycle period is 128, but it

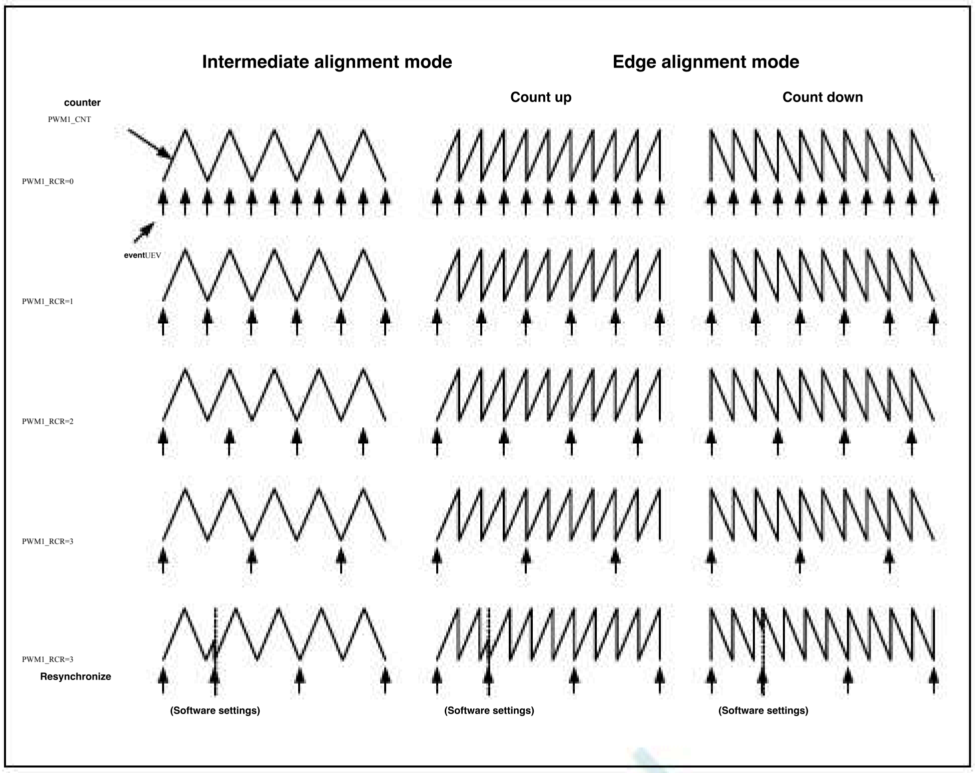
Although this limits the ability to The duty cycle is in many cycles updated every time. In the central alignment mode, because the waveforms are

PWM If the comparison register is refreshed only once in the cycle, the maximum resolution is 2*t CK_PSC。

The repetition counter is loaded automatically, and the repetition rate is determined by PWMA_RCR The value defined in the register. When the update event is generated

Clock through hardware/When the trigger controller is generated, no matter what the value of the repeat PWMA_RCR counter is, an update event occurs immediately, and the contents of the register are overloaded into the repeat counter.

Examples of update rates in different modes, and PWMA_RCR Register settings of



21.4 Clock/trigger controller

The clock/trigger controller allows the user to select the clock source of the counter, input the trigger signal and output the signal ,

21.4.1 Prescaler clock (CK_PSC)

The prescaler clock of the time base unit (CK_PSC) Can be provided by the following sources :

Internal clock (f MASTER)

External clock mode1: external clock input (TIx)

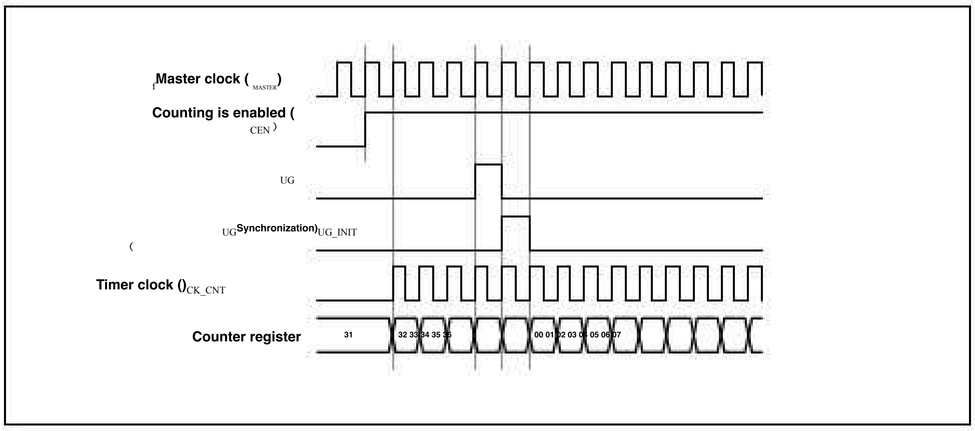
External clock mode2: external trigger input ETR

Internal trigger input (ITRx): Use one PWM of TRGO As another PWM The prescaler clock.

21.4.2 Internal clock source (f MASTER)

If both the clock trigger mode controller and the external trigger input are disabled (PWMA_SMCR Register of SMS=000 , PWMA_ETR Register of ECE=), then CEN、DIR and UG The bit is the actual control bit and can only be modified by the software (UG The bit is still automatically cleared). The bits are CEN written as 1 the clock of the prescalerIt is provided by the internal clock.

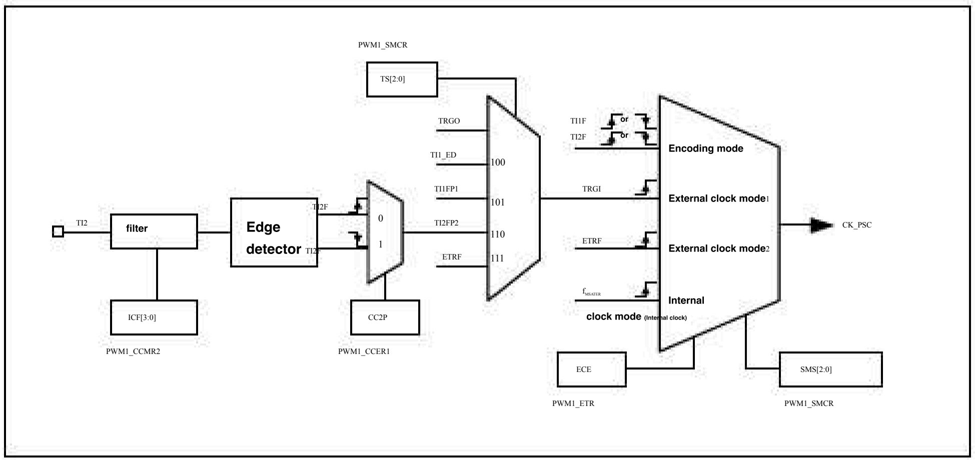
The figure below describes the operation of the control circuit and the up counter in normal mode without a prescaler. Control circuit and The divider factor is normal mode , f MASTER 1


Master clock (f MASTER)
Counting is enabled (CEN)
UG
(UG Synchronization) UG_INIT
Timer clock ()CK_CNT
Counter register
31
32 33 34 35 36
00 01 02 03 04 05 06 07

21.4.3 External clock source mode 1

When PWMA_SMCR Register of SMS=111 When this mode is selected. Then pass PWMA_SMCR Register of TS choosing TRGI The signal source. The counter can count on each rising or falling edge of the selected input.

The following example takes TI2 As an external clock



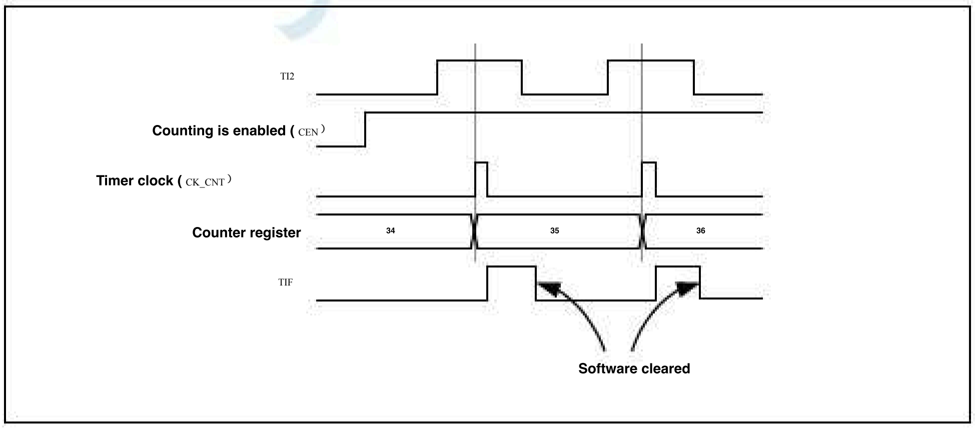
For example, to configure the up counter in TI2 To count the rising edge of the input, use the following steps :

1. Configuration PWMA_CCMR2 The rising edge of the input of the register CC2S=01, Use channel detection TI2
2. configuration PWMA_CCMR2 The bits of the register, select the input filter bandwidth IC2F[3:0]
3. configuration PWMA_CCER1 Register of CC2P=0, Select the polarity of the rising edge
4. configuration PWMA_SMCR Register of the SMS=111 , Configure the counter to use an external clock mode 1
5. configuration settings PWMA_SMCR register of the TS=110, selected TI2 As input source
6. PWMA_CR1 register of the CEN=1, Start the counter

when the rising edge appears in TI2 , The counter counts once, and the identification bit is triggered (PWMA_SR1 Register of TIF Position) is set, such as If the interrupt is enabled (in PWMA_IER Configured in the register), an interrupt request will be generated.

The delay between the rising edge of the counter and the actual clock of the counter depends on the TI2 Resynchronization circuit at the input.

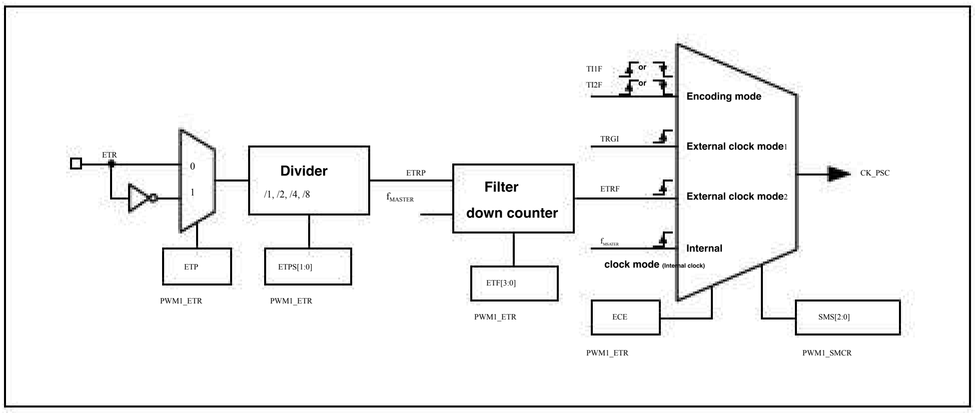
Control circuit in external clock mode 1



21.4.4 External clock source mode 2

The counter can trigger the input externally ETR Each rising or falling edge of the signal is counted. will PWMA_ETR Register of ECE Bit write 1, you can select this mode. (PWMA_SMCR Register of SMS=111 and PWMA_SMCR the register of TS=111 when , You can also choose this mode)

Overall block diagram of external trigger input :

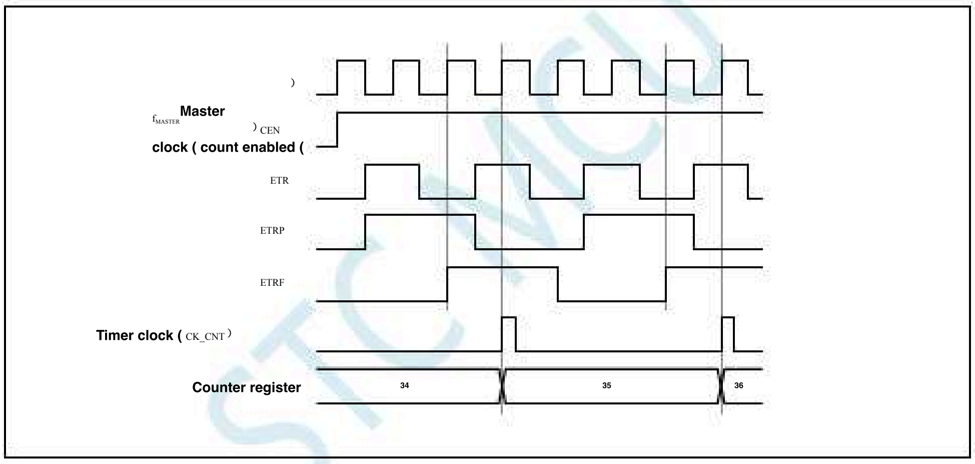


For example, to configure the counter in ETR Count up once on each rising edge, you need to use the following steps: each of the sign

1. In this example, no filter is required, the configuration PWMA_ETR Register of ETF[3:0]=0000
2. is set to the prescaler, and the configuration is set to the prescaler. PWMA_ETR Register of ETPS
3. Selected rising edge detection, configuration PWMA_ETR Register of ETP=0
4. Turn on the external clock mode, configure 2 In the register ECE=1
5. the start counter, and write PWMA_CR1 The rising edge of PWMA_ETR CEN=1

2 The counter is in each the register is counted once. ETR

External clock mode 2 Under the control circuit



21.4.5 Trigger synchronization

PWMA The counter uses three modes to synchronize with the external trigger signal: standard trigger mode , reset trigger mode , gated trigger mode

Standard trigger mode

) Depends on the event on the selected input. Enable the counter (CEN

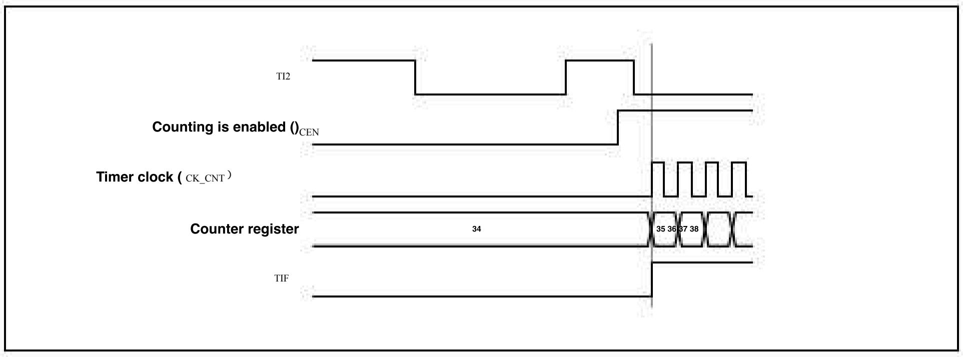
In the following example, the counter starts counting up on the rising edge of the input : TI2

1. Configuration PWMA_CCER1 The register of CC2P=, choose The rising edge of the trigger condition. TI2
2. configuration PWMA_SMCR the register is used as SMS=110 , Select the counter as the trigger mode. configuration PWMA_SMCR Register of TS=110, Choose TI2 the input source.

when When a rising edge appears, the counter starts to count under the drive of the internal clock and is set at the same time TIF logo. TI2 Rising edges and enter TI2

The delay between the counter starting and counting depends on the resynchronization circuit at the input. TI2

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 PWMA_CR1

Register of ，UEV Then all the pre-loaded registers (PWMA_ARR， The bit is low, an update event is also generated

PWMA_CCRx) will be updated.

In the following example， The rising edge of the input causes the upward counter to be cleared to zero：

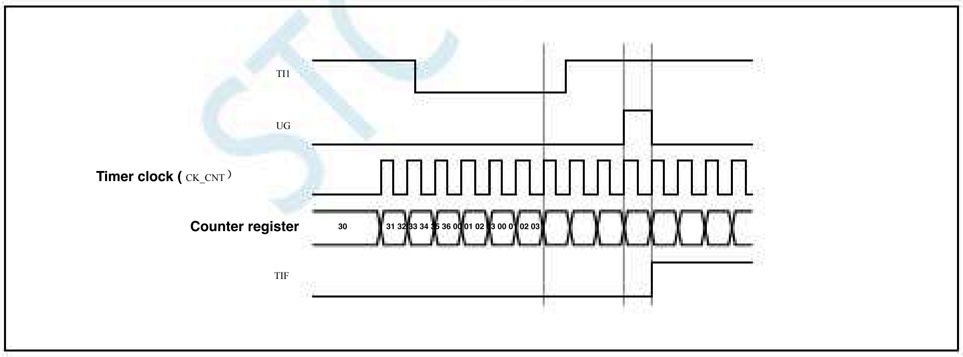
1. Configure the configuration PWMA_CCER1 Register of To choose CC1P=0 Polarity (only detected TI1 The rising edge). TI1
2. PWMA_SMCR the register of SMS=100 , Select the timer as the reset trigger mode. configuration PWMA_SMCR storage of the configurator TS=101, choose TI1 As an input source.
3. PWMA_CR1 Register of CEN=1, Start the counter.

The counter starts to count against the internal clock, and then counts TI1 A rising edge appears. At this time, the counter is cleared and then normally until the count restarts. At the same time, the trigger flag (PWMA_SR1 Register of TIF Bit) is set, if interrupt is enabled (PWMA_IER The bit of the register), then an interrupt request is generated.

TIE The action of the time TI1 Between the rising edge and the actual reset of the counter

The figure below shows when the register is automatically reloaded PWMA_ARR=0x36

The delay depends on the resynchronization circuit at the input. TI1 Reset the control circuit in trigger mode



Gated trigger mode

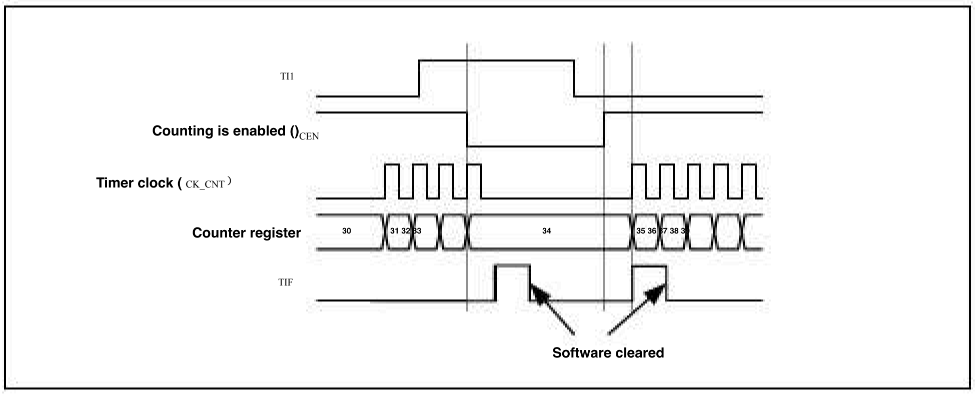
The counter is enabled by the level of the selected input signal.

In the following example, the counter is only in TI1 Count up when it is low：

1. Configure in the PWMA_CCER1 Register of To determine CC1P=1 Polarity (only detected TI1 On the low level)
2. PWMA_SMCR the register of SMS=101 , Select the timer as the gated trigger mode, configure PWMA_SMCR storage configuration TS=101, choose TI1 As an input source. configurator
3. PWMA_CR1 Register of CEN=1 , Start the counter (in gated mode, if CEN=0, The counter cannot be started Move, regardless of the trigger 。 input level)

As long as TI1 is low, the counter starts to count based on the internal clock, once TI1 It gets higher, the count stops. When the counter starts or stops TIF The flag bits will be set. TI1 The delay between the rising edge and the actual stop of the counter depends on TI1 Resynchronization circuit at the input terminal.

Control circuit in gated trigger mode



External clockMode joint trigger mode

External clock mode Can be used with the trigger mode of another input signal. For example, The signal is used as the output of an external c Input, another input signal ETR can be used as a trigger input (supporting standard trigger mode, reset 。 Be careful not to pass trigger mode and gated trigger mode) , the bits of the register are configured as PWMA_SMCR TS ETR TRGI。

A rising edge appears on the counter, that is, Each falling edge of the counter counted up once :

pass PWMA_ETR ETR The register is configured with an external trigger input circuit configuration Disable prescaler, configure ETPS=00 monitoring ETR The rising edge of the ECE=1 Enable external clock mode.2

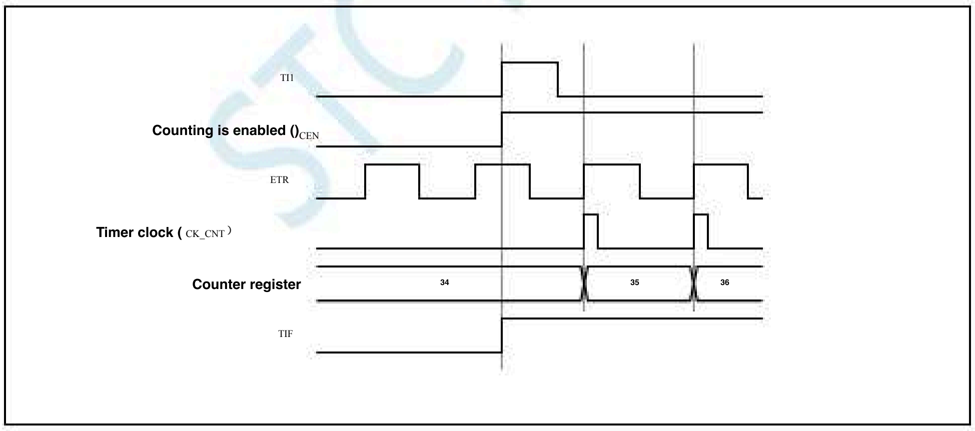
2. Configuration signal, configure PWMA_CCER1 Register of CC1P=0 The rising edge of the trigger. To choose TI1

3. configuration PWMA_SMCR Register of SMS=110 To select the timer as the trigger mode. configuration PWMA_SMCR Register of TS=101 As an input source. To choose TI1

when 1. TI1 When an rising edge appears on The flag is set and the counter starts at The rising edge count of the risin

The delay between the actual clock of the counter and the counter depends on the input terminal. The rising edge of the signal and the actual clock of the ETR ETR The delay between them depends on the resynchronization circuit at the input. ETRP

External clock mode Control circuit in trigger mode



21.4.6 with PWMB sync

In the chip, the timer is internally connected to each other for synchronization or linking of the timer. When a timer is configured as the main mode, a trigger signal can be output (TRGO) To those timer configured as slave mode to complete the reset operation, start operation, stop operation, or as the drive clock of those timer.

Use of PWMB as TRGO PWMA The

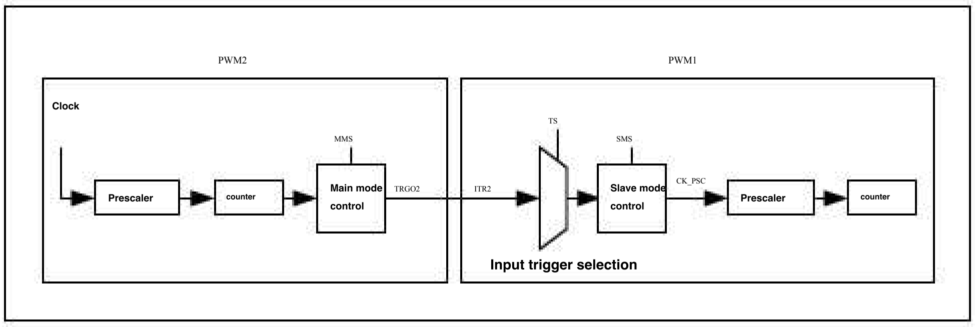
For example, the user can configure PWMB as prescaler The prescaler clock needs to be configured as follows :

1. Configuration PWMB As the main mode, so that in each update event UEV) Outputs a periodic trigger signal. PWMB_CR2 register , so that when each update event MMS=010 The configuration can output a rising edge.

2. PWMB Output of TRGO Signal link to PWMA。 PWMA It needs to be configured to trigger the slave ITR2 As input Trigger signal. The above operations can be configured by PWMA_SMCR mode, using the register implementation. TS=010

3. configuration Register of PWMA_SMCR SMS=111 Turn the clock The trigger controller is set to an external clock mode.
PWMB Output periodic trigger signal TRGO This operation will make the rising edge drive the clock. PWMA
Finally, set PWMB 4. of CEN Bit (PWMB_CR1 In register) , Enable two PWM。

Example of master trigger slave mode



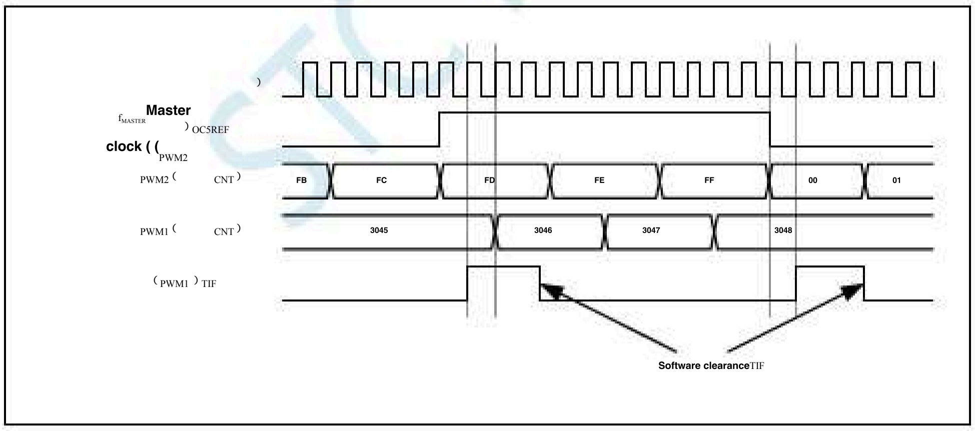
Enable use PWMB

In this example, we use PWMB The comparison output is enabled Only in PWMA。 PWMA Pressed when the signal is high P
Count according to your own driving clock. Two PWM All use the main mode of frequency division, The clock ( ) is fMASTER CK_CNT OC1REF = f /4). MASTER

1. Configured PWMB and the output signal will be compared (OC5REF output as a trigger signal. (Configuration register PWMB_CR2 MMS=100).
2. Configuration PWMB The handle OC5REF The waveform of the signal (PWMB_CCMR1 Register) 。
3. Configuration PWMA PWMB The output is used as its own trigger input signal (configuration PWMA_SMCR Register of TS=010).
4. Configuration PWMA For the gated trigger mode (configuration Register SMS=101).
5. setting CEN bit (PWMA_CR1 Register), enable of PWMA.
6. position CEN bit (PWMB_CR1 Register), enable PWMB。 PWMA_SMCR

Note: Two PWM The clock is not synchronized, but only affects The enable signal.

PWMB Output gating trigger PWMA

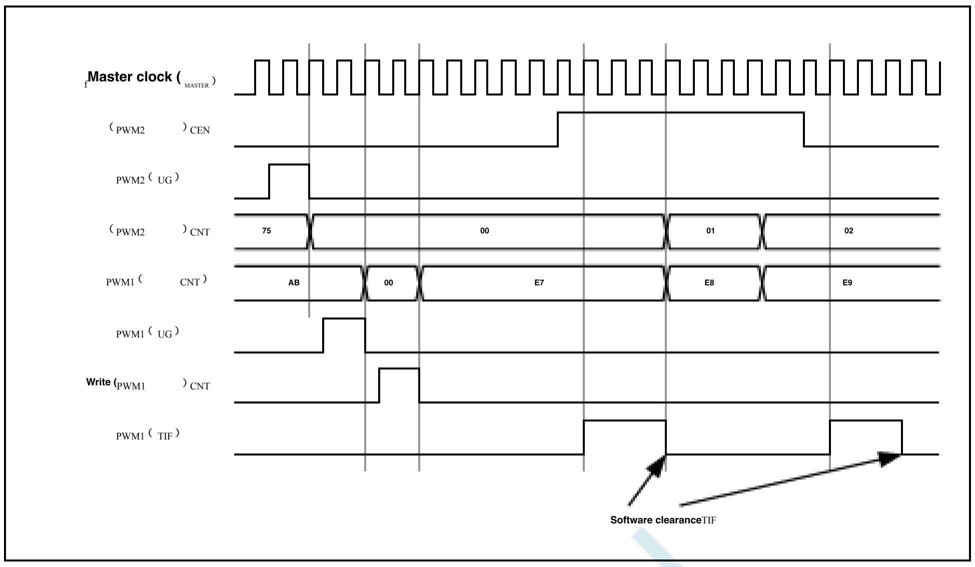


In the picture above, Neither the counter nor the prescaler are initialized before startup, so they are all counted from the existing value. If PWMA the two timer are reset before, the user can write the desired value to the counter to start from the specified value. PWMA
At startup PWMB The reset operation can be written by software PWMA_EGR Register of UG Bit implementation.
Start counting. correct PWMA

In the following example, we make PWMB and PWMA synchronization. PWMB Master mode and 0 slave Count for startup. PW
Trigger slave mode and count from startup. Two PWM 0xE7 The same frequency division coefficient is used. When cleared PWMB_CR1 Register of CEN
Bit time , PWMB Banned, at the same time PWMA Stop counting.

1. Configured In the main mode, the output signal (OC5REF) Output as a trigger signal. (configuration PWMB_CR2 register PWMB MMS=100).
2. Configuration PWMB The handle OC5REF The waveform of the signal (PWMB_CCMR1 Register) 。
3. configuration PWMA PWMB The output is used as its own trigger input signal (configuration PWMA_SMCR Register of TS=010).
4. configuration PWMA For the gated trigger mode (configuration PWMA_SMCR Register of SMS=101).

5. Pass UG Bit (PWMB_EGR Register) Write 1, reset PWMB·
6. UG Bit (PWMA_EGR register) Write, 1 reset PWMA。
7. , pass, pass, pass 0xE7 write In the counter (PWMA_CNTRL), initialize the PWMA。
8. Pass , pass, pass, pass, CEN Bit (PWMA_CR1 register) Write, 1 enable PWMA·
9. CEN Bit (PWMB_CR1 Register) Write 1, start PWMB·
10. pass, pass, pass CEN Bit (PWMB_CR1 Register) Write 0, stop PWMB。



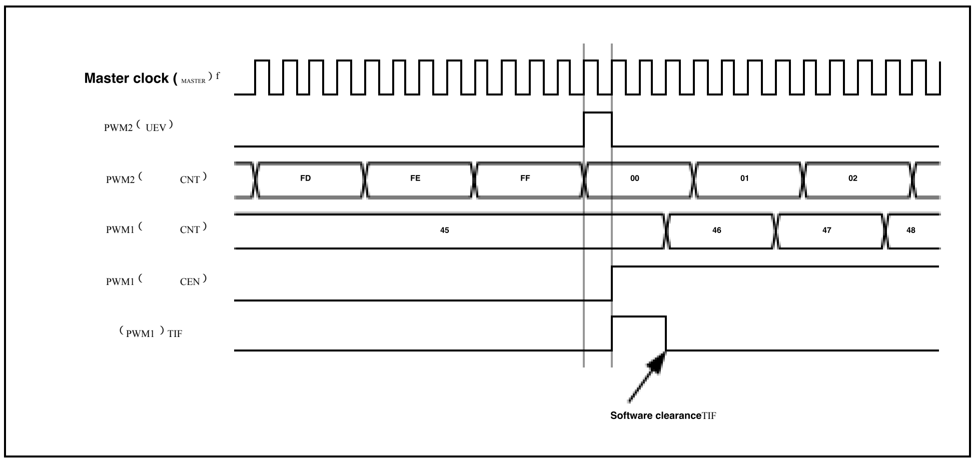
Start to use PWMB PWMA

In this example, we use PWMB To start the update event , follow the PWMA。

PWMA in update event when the update event occurs. Its own drive clock starts counting from its existing value (which can be Value)·PWMA Automatically enabled after receiving the trigger signal CEN and starts counting until the user sends it to the register).0 PWMA_CR1 of 0 Bit write. Two PWM CEN 4 All use divider fMASTER fCK_CNT As the drive clock (= fMASTER /4)·

1. configuration PWMB In the main mode, the output UEV)(Configuration PWMB_CR2 Register of MMS=010)·
2. configuration PWMB update signal (the period (PWMB_ARR Register).
3. configuration PWMA use PWMB The output of the trigger signal as the input (configuration PWMA_SMCR Register of TS=010)·
4. Configuration PWMA For the trigger mode (configuration PWMA_SMCR Register SMS=110)·
5. settings CEN bit (PWMB_CR1 Register) Start of PWMB·

PWMB The update event (PWMB-UEV) Trigger PWMA



As in the previous example, the user can also initialize the counters before starting them.

Trigger two synchronously with an external signal PWM

In this example, the use TI1 The rising edge is enabled PWMB And at the same time enable PWMA 。 In order to keep the timer aligned , PWMB needs to be configured as the main/Slave mode (for TI1 the signal to be in slave mode, for PWMA Main mode) 。

configuration PWMB 1. Main mode ,Take the output enable signal as PWMA Trigger (configuration PWMB_CR2 Register of MMS=001)·

2. Configuration PWMA The signal is used as the input trigger signal (configured as slave mode, the TI1 PWMB_SMCR Register of TS=100).
3. configuration PWMB Trigger mode (configuration
4. configuration PWMB -based/Slave mode (configured
5. configuration PWMA to PWMB Register of TS=010).
6. configuration PWMA The trigger mode (configuration register of PWMB_SMCR SMS=110). Register of PWMB_SMCR MSM=1). The output is the input trigger sig

when TI1 When the rising edge appears, the two timer start counting TIF All positions are set up.

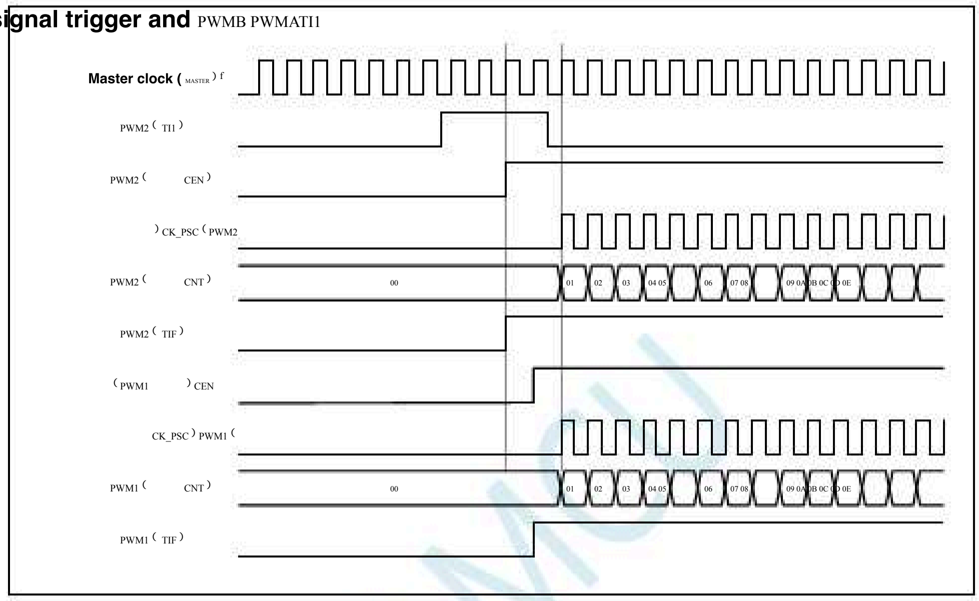
synchronously, and note: in this example, both timer are initialized before , So they all count from the beginning, bits) UG

starting (set but the user can also modify the counter register (PWMA_CNT) To insert an offset, in this case, in the PWMB

CK_PSC A delay will be inserted between the signals.

PWMB CNT_EN Signal

and signal trigger and PWMB PWMATI1

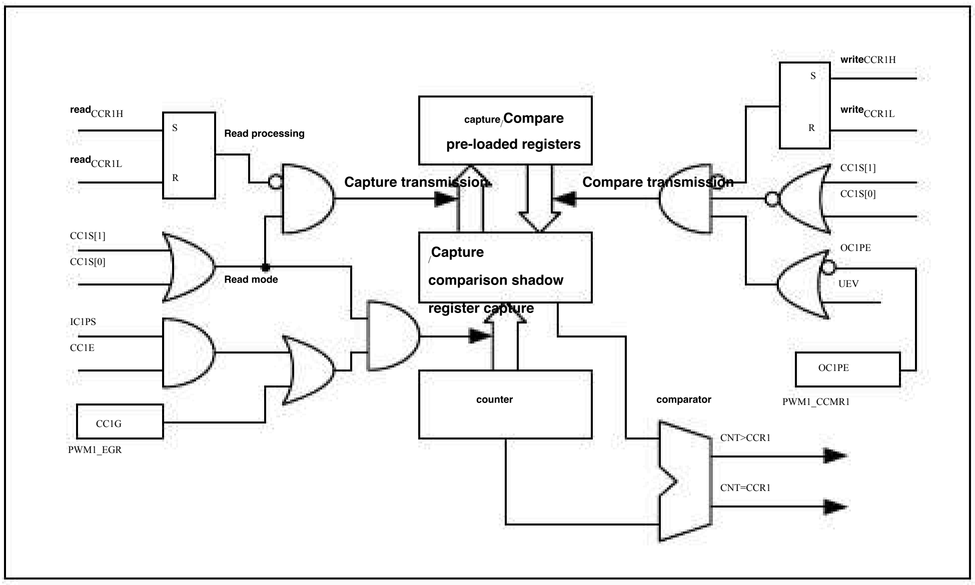


21.5 /Capture comparison channel

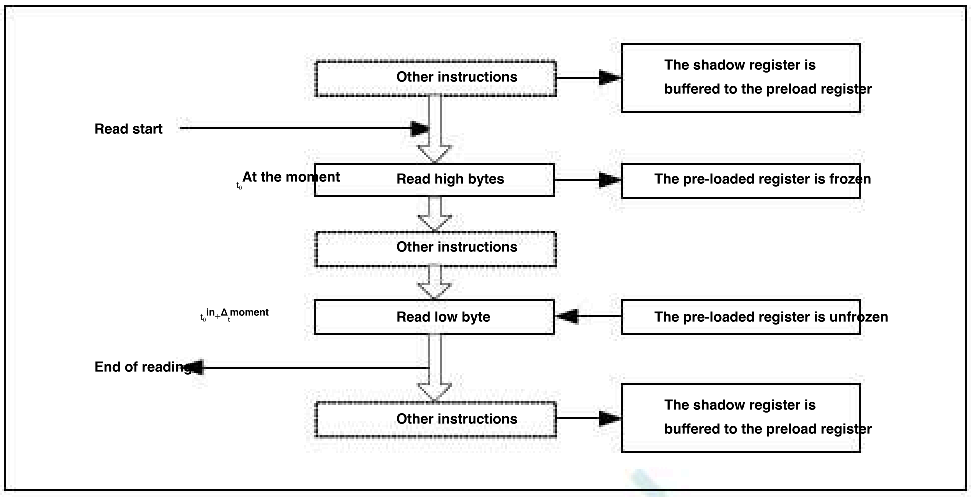
PWM1P PWM3P/PWM3N Can be used as input capture PWM1P/PWM1N, PWM2P/PWM2N, Can output comparison, this function can be configured to capture and compare the channel mode register PWMA_CCMRi) of PWM3P PWM4P PWM2P PWM4P/PWM4N / CCiS The channel selection bit is implemented, the i The representative channels.

Every capture/Comparison channels are all around a capture/Comparison register (including shadow register) is constructed, including the captured input part (digital filtering, multiplexing, and prescaler) and the output part (comparator and output control)

capture/Compare the 1 main circuit of the channel (other channels are similar to this)



capture/The comparison module consists of a preload register and a shadow register. The read-write process only operates the pre-loaded registers. In capture mode, the capture takes place on the shadow register and then copied to the preload register. In comparison mode, the contents of the preloaded register are copied to the shadow register, and then the contents of the shadow register are compared with the cou
When the channel is configured as an output mode, it can be accessed PWMA_CCRi register.
at any time. When the channel is configured as an input mode, The read operation of a register is PWMA_CCRi similar to the read operation of a counter. Whe
When the contents of the counter are captured to in the PWMA_CCRi shadow register, it is then copied to the preload register. The read operation is in prog
The pre-loaded register is frozen.


Other instructions
The shadow register is buffered to the preload register
Read start
At the moment
Read high bytes
The pre-loaded register is frozen
Other instructions
in Δ moment
Read low byte
The pre-loaded register is unfrozen
End of reading
Other instructions
The shadow register is buffered to the preload register

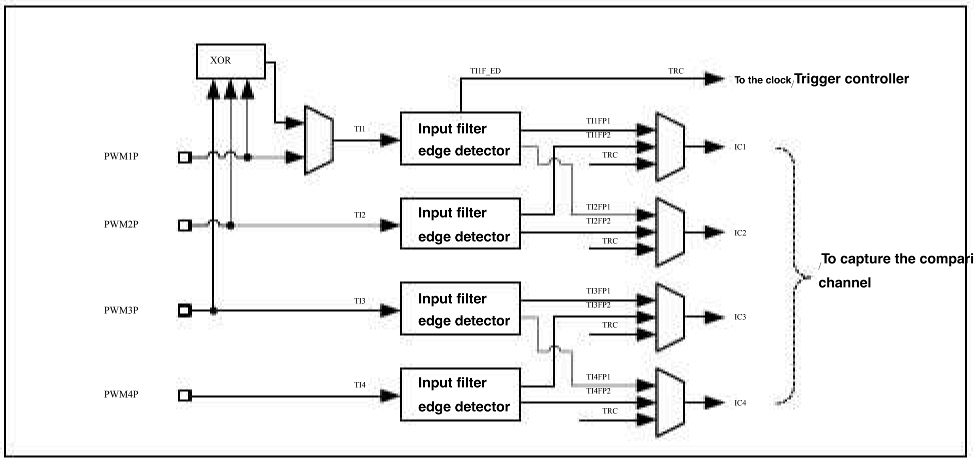
The picture above describes Bit of 16 The read operation process of the register, the data being cached will remain unchanged until the end of t
After the reading process is over, if you only read PWMA_CCRi Register, returns the low bit of the counter value. If you read the low-bit data to
After reading the high-bit data, the same low-bit data will no longer be returned.

21.5.1 bit 16 PWMA_CCRi Register writing process

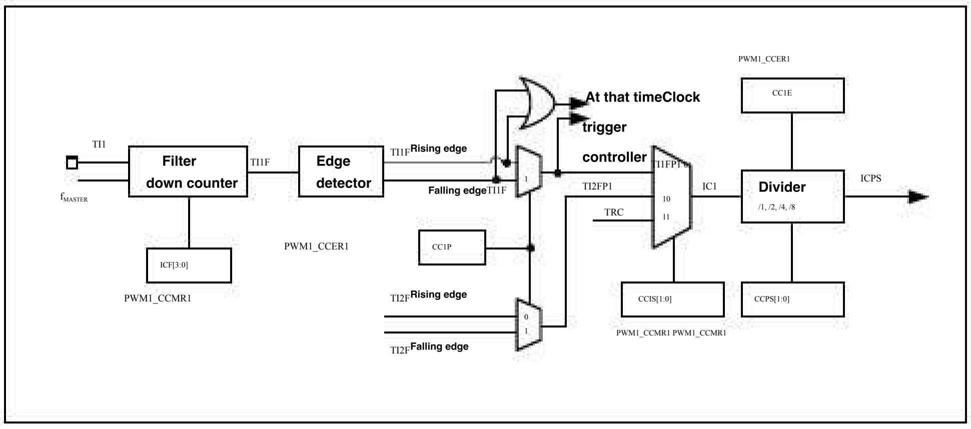
16 bit PWMA_CCRi The write operation of the register is completed by preloading the register. Two instructions must be used to com
Each instruction corresponds to one byte. It is necessary to write the high-bit byte first. When writing the high-bit byte, the update of the shadow register is prohibited until the low-bit byte is written.

21.5.2 Input module

Block diagram of the input module


XOR
TI1F_ED
TRC
To the clock/Trigger controller
PWM1P
TI1
Input filter edge detector
TI1FP1
TI1FP2
TRC
IC1
PWM2P
TI2
Input filter edge detector
TI2FP1
TI2FP2
TRC
IC2
To capture the comparison channel
PWM3P
TI3
Input filter edge detector
TI3FP1
TI3FP2
TRC
IC3
PWM4P
TI4
Input filter edge detector
TI4FP1
TI4FP2
TRC
IC4

As shown in the figure, the input part pairs the corresponding input signal is sampled TIx and a filtered signal is generated TIxF. Then, a band with polarity is selected
The selected edge monitor generates a signal (TIxFPx), it can be triggered as an input to the
The signal enters the capture register after passing the prescaler (ICxPS). trigger mode controller or as a capture control.



21.5.3 Input capture mode

After the corresponding edge on the signal, the current value of the counter is latched to the capture

(When a capture event occurs, the flag corresponding (PWMA_SR Register) is set. if PWMA_IER CCiIF 1 PWMA_CCRx) in.

Register of CCiIE If the bit is set, that is, an interrupt is enabled, an interrupt request will be generated. If the flag is already when the captur

Once it is high, the captured data in the repeated CCiOF (PWMA_SR2 Register) is set. 1 write CCiIF=0 Or read stored in PWMA_CCRiL

capture flag register can be cleared CCiIF。 write CCiOF=0 Can be cleared CCiOF。

PWM Capture when the input signal rises on the edge

The following example shows how to capture the value of the counter on the rising edge of the input to the register, the steps are as follows: , At

1. Select a valid input terminal and set PWMA_CCMR1 In the register CC1S=01 this time the channel is configured as input, and PWMA_CCR1 The register becomes read-only.

2. According to the input signal The characteristics configured by PWMA_CCMR1 In the register IC1F Bits to set the corresponding input filter The filtering time of the device. Assuming that the input signal dithers within the time of the most 5 clock cycles, we must configure the 5 be longer than the clock cycle; therefore, we can continuously sample times to confirm the real edge transformation in the last time, t Write in the register PWMA_CCMR1, At this time, only continuous 8 samples the same to IC1F=0011 TI1 Signal, signal only Is valid (sampling frequency is f_{MASTER}) f。

3. choose TI1 The effective conversion edge of the channel, in PWMA_CCER1 Write in the register CC1P=0 (Rising edge).

4. Configure the input prescaler. In this example, we want the capture to occur at every valid level conversion moment, so the prescaler is disabled (write register PWMA_CCMR1 IC1PS=00). Setting the

5. register allows the value of the counter to be captured in the capture register. PWMA_CCER1 CC1E=1

6. If necessary, allow related interrupt requests by setting the bits in the register. PWMA_IER CC1IE

When an input capture occurs :

When a valid level conversion is generated, the value of the counter is transmitted to PWMA_CCR1 register

CC1IF The flag is set. When at least one 2 consecutive capture occurs, CC1IF and When it has not been cleared, Also set 1. If set CC1OF

up CC1IE Bit, an interrupt will be generated.

In order to handle the capture overflow event (CC1OF Bit), it is recommended to read the data before reading out the duplicate capture flag, this is to

Repeated capture information that may occur after the capture overflow flag is read out and before the data is read.

Note: Settings PWMA_EGR Register phase corresponding CCiG Bits, input capture interrupts can be generated by the software.

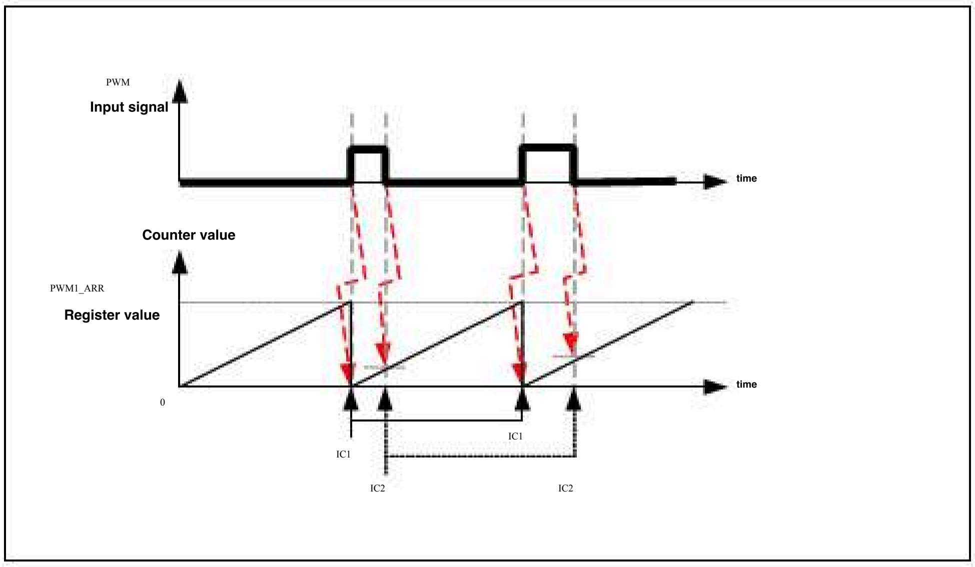
PWM Input signal measurement

This mode is a special case of the input capture mode, except for the following differences, the operation is the same as the input captur

Two of these ICi The signal is mapped to the same TIi input

two ICi The polarity of the effective edge of the signal is opposite.

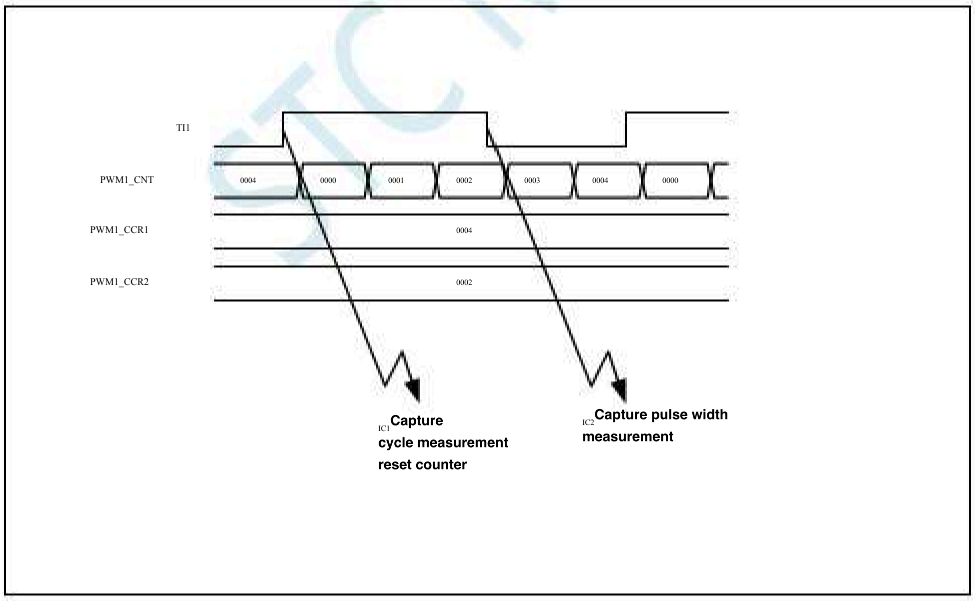
One of them TIiFP The signal is used as the trigger input signal, and the trigger mode controller is configured to reset the trigger



For example, you can measure in the following ways TI1 Enter on PWM The period of the signal (PWMA_CCR1 Register) and duty cycle (Register). PWMA_CCR2

1. Choose PWMA_CCR1 Effective input: set PWMA_CCMR1 Register of CC1S=01 (Selected TI1FP1).
2. choose TI1FP1 choose effective polarity: set CC1P=0 (The rising edge is valid).
3. choose PWMA_CCR2 Valid input: set the valid Register of PWMA_CCMR2 CC2S=10 (Selected TI1FP2).
4. choose TI1FP2 polarity (capture data to PWMA_CCR2): Set CC2P=1 (The falling edge is valid).
5. Select a valid trigger input signal: set PWMA_SMCR In the register TS=101 (Choose TI1FP1).
6. Configure the trigger mode controller to reset the trigger PWMA_SMCR in SMS=100。
7. mode: Set to enable capture: Set to register PWMA_CCER1 CC1E=1， CC2E=1。

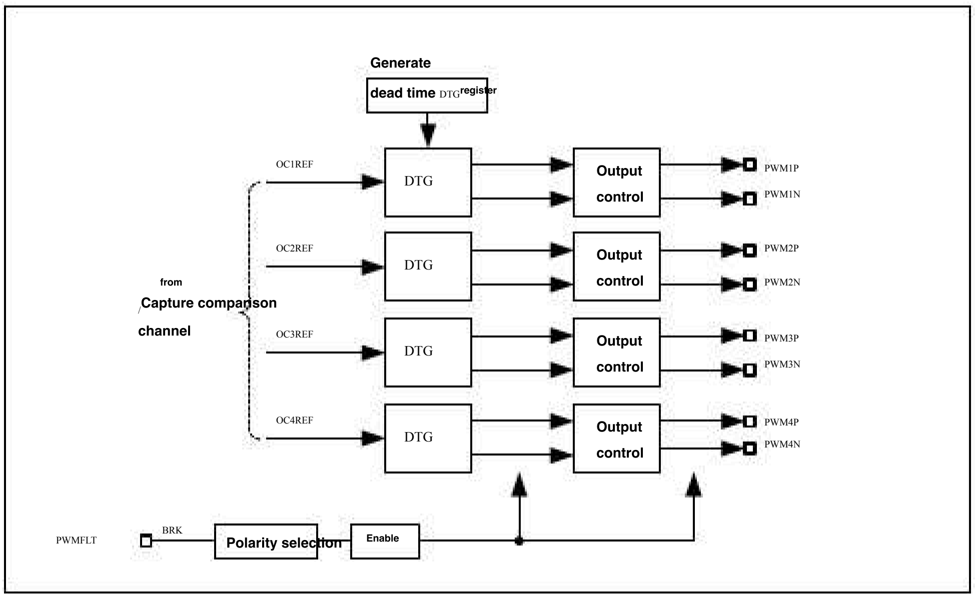
PWM Input signal measurement example


TI1
PWM1_CNT
0004
0000
0001
0002
0003
0004
0000
PWM1_CCR1
0004
PWM1_CCR2
0002
IC1 Capture cycle measurement reset counter
IC2 Capture pulse width measurement

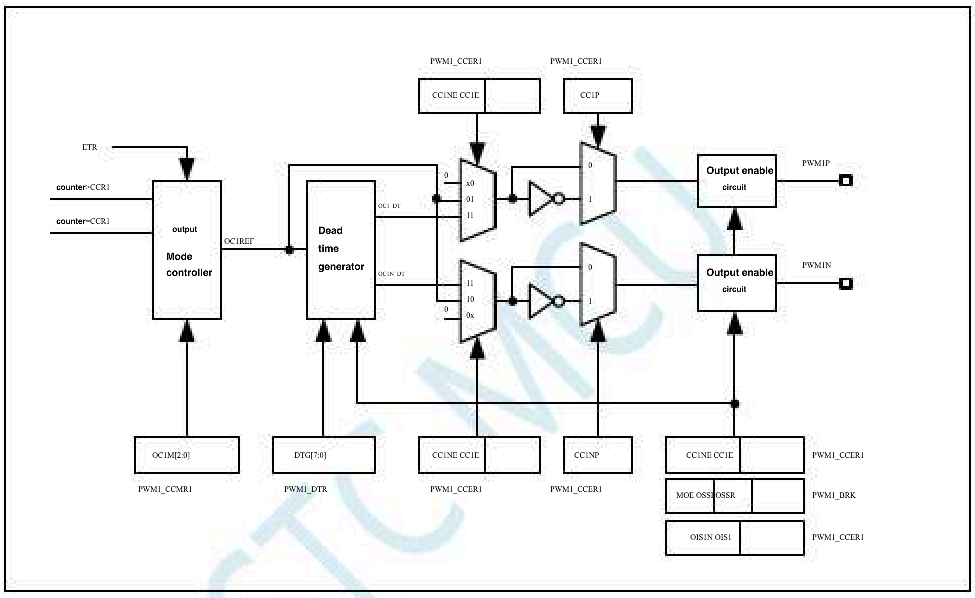
21.5.4 Output module

The output module will generate an intermediate waveform for 。 The braking function and polarity are processed in reference, which is called the final processing of the module.

Output module block diagram



Detailed block diagram of the output module with complementary outputs for the channel (similar to other channels)



21.5.5 Forced output mode

In the output mode, the output comparison signal can be directly forced to a high or low state by the software, without relying on the comparison result between the output comparison register and the counter.

set PWMA_CCMRi Whether the output of OCiM=101, Can be forced OCiREF The signal is

PWMA_CCMRi the register of the register OCiM=100, Can be forced OCiREF low. The signal is

OCi/OCiN is high or low depends on CCiP/CCiNP low. Polarity flag.

In this mode, the PWMA_CCRi The comparison between the shadow register and the counter is still in progress, and the corresponding interrupt will still be generated.

21.5.6 Output comparison 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 contents of the capture comparison register, there are the following operations :

According to different output comparison modes, the corresponding OCi Output signal

- Remains OCiM=000)
- unchanged (set to valid level (OCiM=001)
- set to invalid level (OCiM=010)

- Flip (OCiM=011)

Set the flag bit in the interrupt status register (PWMA_SR1 In the register CCiIF Bit).

If the corresponding interrupt enable bit is set (PWMA_IER In the register CCiIE , An interrupt is generated. Bit)

PWMA_CCMRi Register of OCiM Bits are used to select the output comparison mode, and PWMA_CCERi Bit register CCiP

To select valid and invalid level polarity. PWMA_CCMRi Register of OCiPE Bits are used to select PWMA_CCRi has no effect on whether the register

You need to use the pre-loaded register. In output comparison mode, update events UEV correct and OCi is output or not. Time essence

Degree is a counting cycle of the counter. The output comparison mode can also be used to output a single pulse.

Configuration steps for output comparison mode:

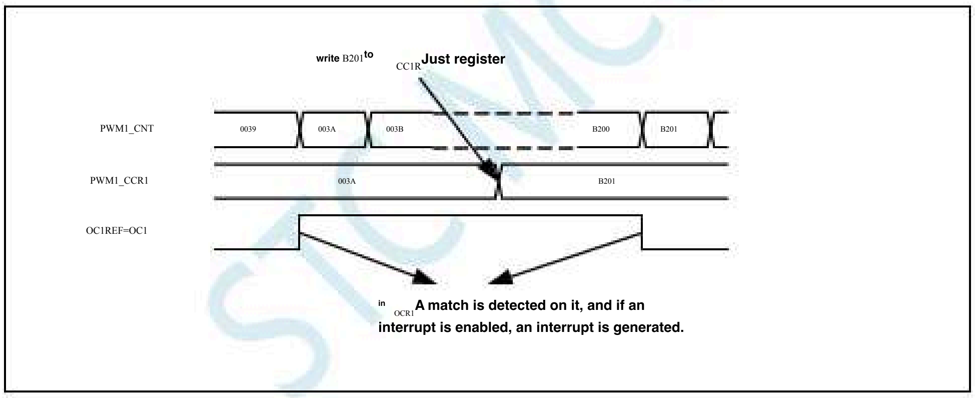
1. selectCounter clock (internal, external, or prescaler).
2. Write the corresponding data to and PWMA_ARR PWMA_CCRi In the register.
3. If you want to generate an interrupt request, set the bit. CCiIE
4. To select the output mode :
   1. Set CCRi OCiM=011 At the Flip when matching OCiM Pin output
   2. up OCiPE = 0 counter and , disable the preload register
   3. Set up CCiP , Select the high level as the effective level
   = 0 Set up
   4. CCiE = 1'

Enable output settings PWMA_CR1 5. Register of CEN Bit to start the counter

PWMA_CCRi The register can be updated at any time through software to control the output waveform, provided that the pre-loaded

Device (PWMA_OCiPE=0), otherwise The shadow register can only be updated when the next update event occurs.

Output comparison mode, flip OC1


write B201 to CC1R Just register
PWM1_CNT
0039
003A
003B
B200
B201
PWM1_CCR1
003A
B201
OC1REF=OC1
in OCR1 A match is detected on it, and if an interrupt is enabled, an interrupt is generated.

pattern 21.5.7 PWM

Pulse width modulation (PWM) The pattern can produce a PWMA_ARR The register determines the frequency by PWMA_CCRi storage

The signal of the device to determine the duty cycle.

Set each In the register PWMA_CCMRi Bit write OCiM 110 (PWM 2 Mode), able to be independent

on the ground OCi The output channel generates one way Must be set PWMA_CCMRi 111 (PWM 1 Mode) or

Load register, you can also set the type PWMA_CR1 Register of ARPE Bits enable automatic reloading register of OCiPE Bit enables the corresponding pre-installed

or in the central symmetrical mode)

Since the preloaded register can only be transferred to the shadow register when an update event occurs

, all registers must be initialized by setting bits before the counter starts counting. Register of PWMA_EGR UG

The polarity can be determined by the software in the register PWMA_CCERi CCiP Bit setting, which can be set to active high

The level is valid. OCi The output is enabled by PWMA_CCERi PWMA_BKR or low in the register CCiE、 MOE、 OISi、 OSSR

and OSSI A combination of bits to control.

Mode (mode or mode) under , PWMA_CNT 2 In harmony PWM Always comparing, (According to the counter

The counting direction) to determine whether it meets PWMA_CCRi≤ PWMA_CNT or PWMA_CNT≤ PWMA_CCRi。 PWMA_CCRi

According to PWMA_CR1 The state of the bit field, the timer can generate an edge-aligned or centrally aligned CMS Signal register PWM

PWM signal.

Edge alignment mode PWM

Count up configuration

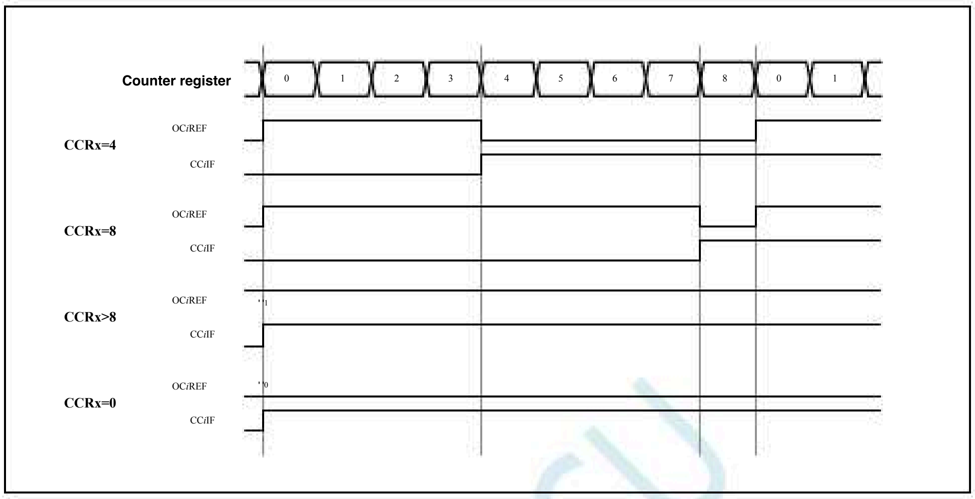
When the following Position at the time 0, Perform an upward count.

is the one In the register PWMA_CR1 when , PWM Reference signal For high,

otherwise it is low. if DIR PWM 1 Examples of patterns, when The comparison value in is greater than the automatic reload value (PWMA_ARR), then OCiREF keep it high.

If the comparison value is 0, then OCiREF Keep it low.

Edge alignment , PWM 1 The waveform of the mode (ARR=8)



Configuration that counts down

When PWMA_CR1 Register of DIR The bit is 1 When, perform a downward count.

in PWM When, when mode 1 Time reference signal PWMA_CNT>PWMA_CCRi Is low, otherwise it is high. if OCiREF

PWMA_CCRi The comparison value in is greater than PWMA_ARR The automatic reload value in, then OCiREF Keep it high. Cannot be produced in this mode

0 The raw duty cycle is % PWM waveform.

PWM Central alignment mode

when PWMA_CR1 In the register CMS The bit is not '00' When it is in the central alignment mode (all other OCiREF/OC letter configurations are right

All numbers have the same effect).

Depending CMS Bit setting, the comparison flag can be set when the counter counts up, down, or up and down.

1◦ on the PWMA_CR1 The count direction bit in the register (DIR) Update by hardware, do not modify it

Some centrally aligned PWM with software. Examples of waveforms :

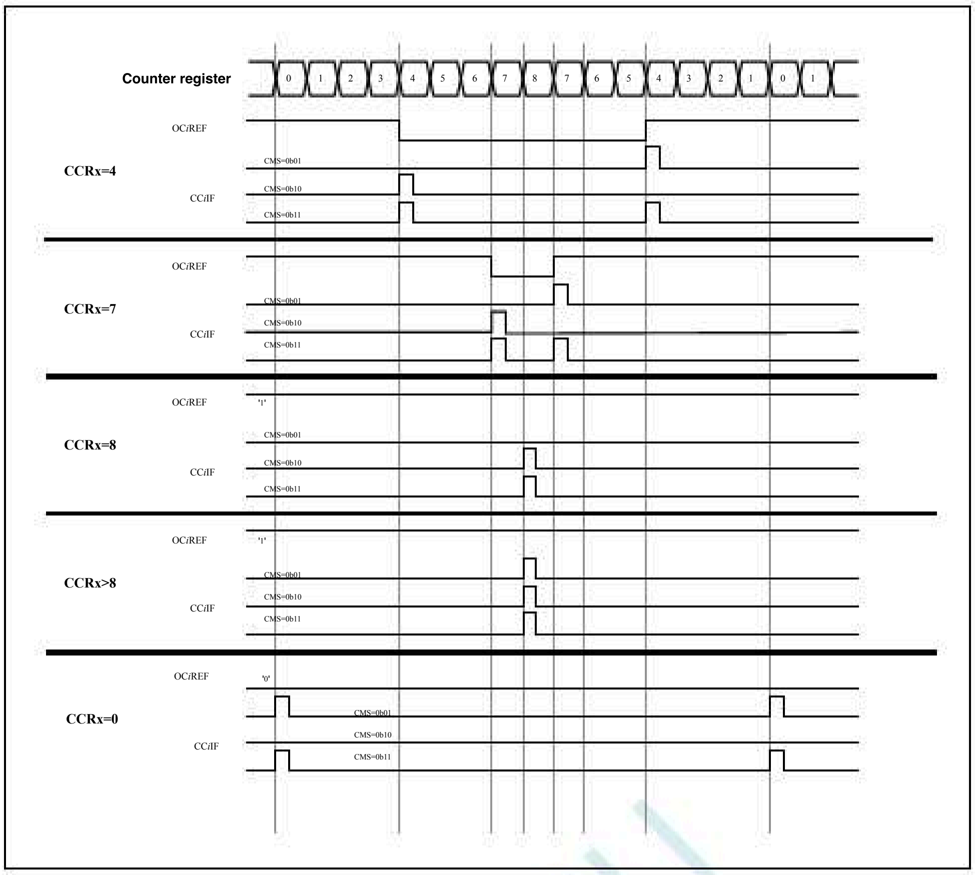
ones are given below PWMA_ARR=8

PWM pattern 1

The flag is set in the following three cases :

- Only when the counter counts down (CMS=01)
- Only when the counter counts up (CMS=10)
- When the counter counts up and down (CMS=11)

the center-aligned waveform (PWM ARR=8)



Single pulse mode

Single pulse mode (OPM) Is a special case of the many aforementioned patterns. This mode allows the counter to respond to an excitation After a sequence-controlled delay, a pulse with a controllable pulse width is generated.

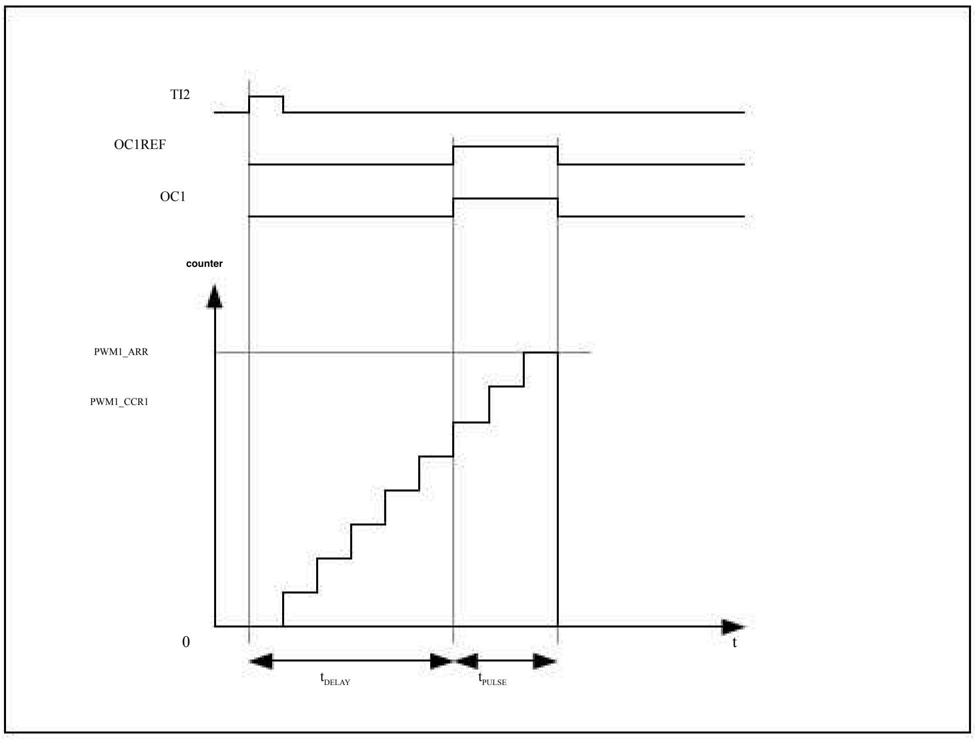
You can turn on the clock to trigger the controller to start the counter, in the output compare mode or PWM A waveform is generated in the mode. Set up PWMA_CR1 Register of OPM The bit will select the monopulse mode, at which time the counter will automatically stop in the next event UEV When updating is

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), it must be configured as follows：

Counting up mode: Counter ≤ CNTARR，

counting down mode: Counter < CCRi CNT > CCRi。

single pulse mode legend



For example, from Delay after a rising edge is detected on the input pin t_{DELAY} OC1 Generate one on t_{PULSE} Positive pulse width :
(Assumed as a trigger IC2 1 The trigger source of the channel)

Set PWMA_CCMR2 Register of CC2S=01, put Map to IC2 TI2. Ability to
set PWMA_CCER1 the register of CC2P=0, make IC2 detect rising edges.
PWMA_SMCR Register of TS=110, make IC2 As a clock/The trigger source of the trigger controller (TRGI).
PWMA_SMCR the register of SMS=110 (Trigger mode) , IC2 Is used to start the counter. OPM The waveform is written by
The value of the input comparison register is determined (the clock frequency and counter prescaler must be considered)
t_{DELAY} The value in the register is defined.
t_{PULSE} PWMA_CCR1 Defined by PWMA_ARR - PWMA_CCR1). The waveform, when the
the difference between the autoloaded value and the counter reaches the preloaded value, a slave is generated. to 1 0
Assuming that when a comparison match occurs, the waveform of the slave is to be generated, and the waveform of the slave must first be set Register of OCiM=111, enter PWMA_CCMR0 to 11 Of the setting register PWMA_CCMR1 Mode, there are options according to needs
Load the register, and then in PWMA_CCR1 OC1PE=1, Set PWMA_CR1 In the register ARPE, Enable automatic installation
Load value, set UG Fill in the comparison value in the register, in PWMA_CCR1 by filling in the pre-installed register
Bit to generate an update event, and then wait in An external trigger event on. TI2
In this example , PWMA_CR1 In the register DIR and CMS The position should be set low.
Because only one pulse is required, set PWMA_CR1 In the register OPM=1 , In the next update event (when the counter is from
Stop counting when the auto-loaded value is flipped to 0).

Fast enable (special case) OCx

In single pulse mode, yes TIi The edge detection of the input pin will be set CEN Bit to start the counter, and then the difference between the counter a
Comparison operation produces The output of a single pulse. However, these operations require a
certain clock cycle, so it limits the minimum delay available. t_{DELAY}

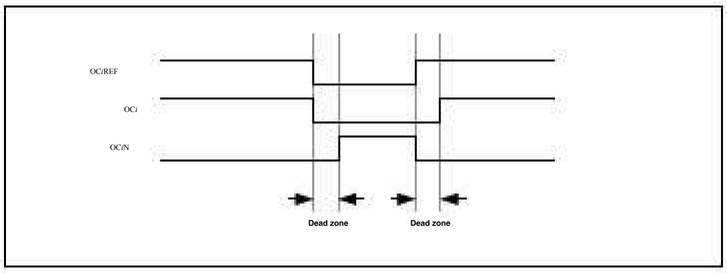
If you want to output the waveform PWMA_CCMRi In the register OCiFE Bit, at this time forced OCiREF (And only in the channel
with minimum delay, you can set it Directly respond to the excitation without relying on the result of the comparison, and the output waveform is the same as the wavefor configuration
It works when set to mode. PWMB and PWMA

Complementary output and dead zone insertion

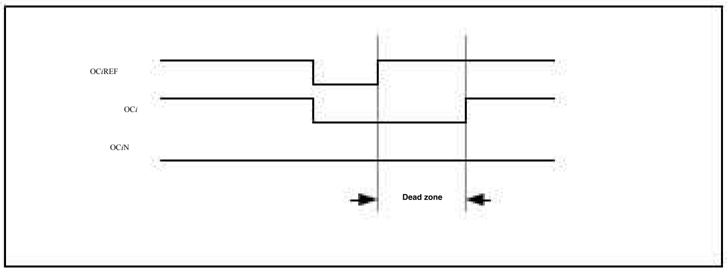
PWMA Can output two complementary signals, and can manage the instantaneous shutdown and on of the output, this period of time i
The user should adjust the dead time according to the connected output devices and their characteristics (delay of level conversion, delay of

Configuration for complementary PWMA_CCERi CCiP And in the Bit, you can independently select the polarity for each output (main output CCiN
output and OCi OCiN register OCiN OCi It is controlled by a combination of the following control bits: register's PWMA_CCERi
CCiE and CCiNE Complementary signal bits , In the register MOE、 OISi、 OISiN、 OSSI and OSSR bit. In particular, in the transfer
Change to IDLE Down to) dead zone control is activated. When in state (MOE PWMA_BKR 0
Set at the same time CCiNE and CCiE The bit will be inserted into the dead zone, if there is a brake circuit, then MOE Bit must also be set
There is one 8 Bit of dead zone generator.

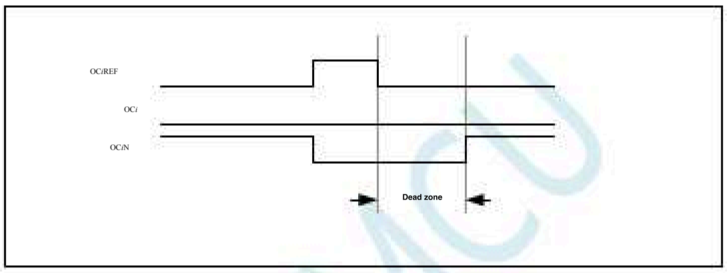
if OCi and OCiN For high effectiveness :
OCi The output signal is the same as OCiREF The same, except that its rising edge is relative to OCiREF There is a delay on the rising edge.
OCiN The output signal is the same as OCiREF On the contrary, it's just that its rising edge is relative to There is a delay on the falling edge. OCiREF
If the delay is greater than the currently valid output width (OCi or OCiN), no corresponding pulse will be generated.
The following pictures show the output signal of the dead zone generator and the current reference signal OCiREF The relationship between. (Assuming CCiP=0
MOE=1、 CCiE=1 CCiNP=0, and CCiNE=1)
Complementary output with dead zone insertion



The dead-zone waveform delay is greater than the negative pulse



The dead-zone waveform delay is greater than the positive pulse



The dead zone delay of each channel is the same, which is determined by the register PWMA_DTR DTG Bit programming configuration.

To redirect OCi or OCiN OCiREF

In output mode (Forced output, output comparison or PWM Output) Through configuration PWMA_CCERi Register of CCiE and CCiNE bit , OCiREF Can be redirected to OCi or OCiN The output.

This function can be used when the complementary output is at an invalid level, send a specific waveform (e.g. PWM Or quiet level) on a certain output. Another effect is to keep the two outputs at an invalid level at the same time, or at an active level at the same time (at this time it is still a dead zone). Note: When only OCiN enabled), it will not reverse phase, but will be effective immediately when it becomes high. For example , OCiN (CCiE=0, CCiNE=1 OCiREF

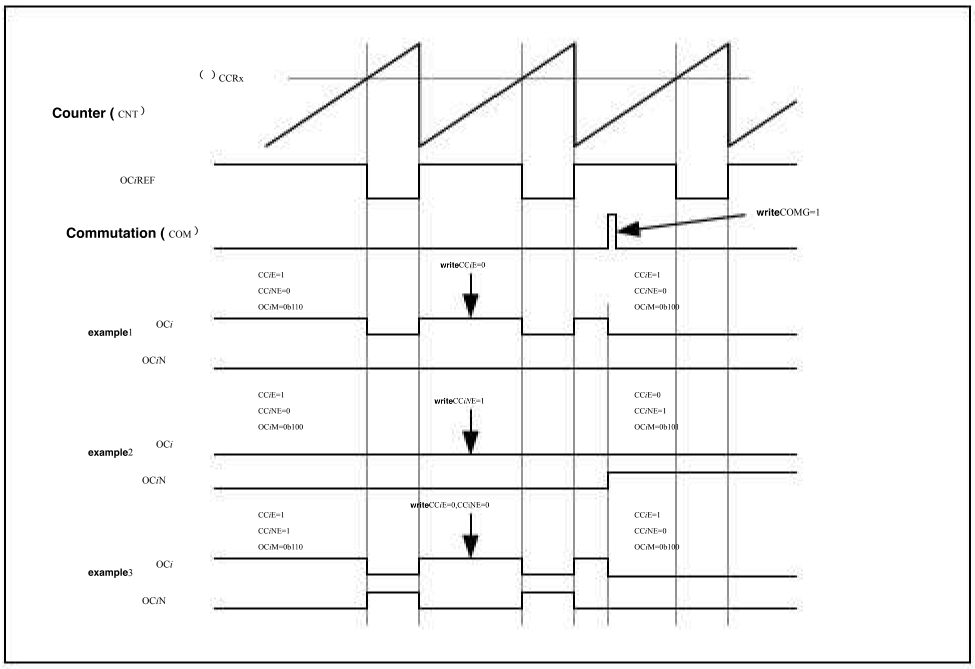
If it is OCiN=OCiREF。 On the other hand, when CCiNP=0 and OCi It is When all are enabled (CCiE=CCiNE=1), when OCiREF high OCi effective; on the contrary, when it is low OCiN OCiN OCiREF Become effective.

Six steps for motor control PWM output

When complementary outputs are required on one channel, the preload bits have OCiM CCiE and CCiNE Happening COM In the event of a commutation event, These preload bits are transferred to the shadow register bits. This way you can pre-set the configuration for the next step and modify and change the configuration of all channels at the same time at the same time. The bits of the register are generated by the software, or on COM PWMA_EGR COMG TRGI The rising edge is generated by the hardware.

The figure below shows when the time of the event, under three different configurations OCx and OCxN output

Produce six steps PWM, use COM An example of (OSSR=1)



21.5.8 Use the brake function (PWMFLT)

The brake function is commonly used in motor control. When using the brake function (PWMA_BKR According to the corresponding control bit In the register MOE OSSI and OSSR Bit), the output enable signal and the invalid level will be modified.

After the system is reset, the brake circuit is disabled. The position is low. Set up In the register PWMA_BKR Position can enable the brake BKE function. The polarity of the brake input signal can be configured by configuring the polarity MOE Bit selection BKP in the same register BKE Can be modified at the same time.

MOE The falling edge can be asynchronous with respect to the clock module, so in the actual signal (acting on the output terminal) and

PWMA_BKR A resynchronization circuit is set up between the registers). This resynchronization circuit will produce between the asynchr

Life is delayed. In particular, if you write when it is low MOE=1 , then before reading it out You must insert a delay (empty instruction) before you can r

Correct value. This is because the asynchronous signal is written and the synchronous signal is read.

When braking occurs (the selected , There are the following actions :

level appears at the brake input) The bits are cleared MOE asynchronously, and the output is placed in an invalid state, an idle OSSI Bit selection) this

state, or a reset state (the oscillator with a characteristic is still valid when it is turned off. MCU

once , The output of each output channel is composed PWMA_OISR Register of OISi The level set by the bit. if OSSI=0 ,

The timer no longer controls the output enable signal, otherwise the

output enable signal is always high. When using complementary outputs :

The output is first placed in a reset state, that is, an invalid state (depending on the polarity). This is an asynchronous operation, even if t

have a clock- time, this function is valid. If the timer's clock still exists,

the dead zone generator will take effect again. After the dead zone, according to the sum bit index, the dead zone generator will take effe

The level shown drives the output port. Even in this case , OCi And cannot be driven to an effective level at the same time. OCiN

Note: Because of resynchronization MOE, the dead time is longer than usual (approximately 2 clock cycles).

If the bit of the register is set, when the brake status flag (PWMA_SR1 Bits in the register) PWMA_IER BIE BIF

For the time 1, An interrupt is generated.

If a bit in the register is set, the bit is automatically set in the next update event. PWMA_BKR AOE UEV MOE

For example, this can be used for waveform control, otherwise , 1

MOE Always keep it low until it is set again. This feature can be used in

safety. You can connect the brake input to a power-driven alarm output, a thermal sensor, or other safety devices. Note:

The brake input is valid at the level. Therefore, when the brake input is valid, it cannot be set at the same time (automatically or through s

At the same time, the status flag BIF Cannot be cleared.

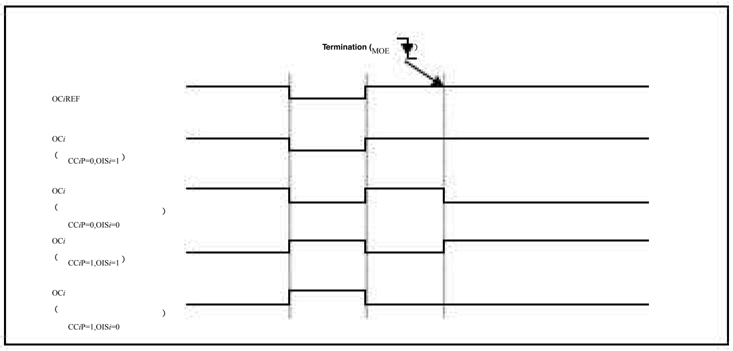
The brakes are made of BKR Input generation, its effective polarity is programmable, and it is generated by PWMA_BKR Register of BKE Bits are turned on or off.

In addition to brake input and output management, write protection is also implemented in the brake circuit to ensure the safety of the applica

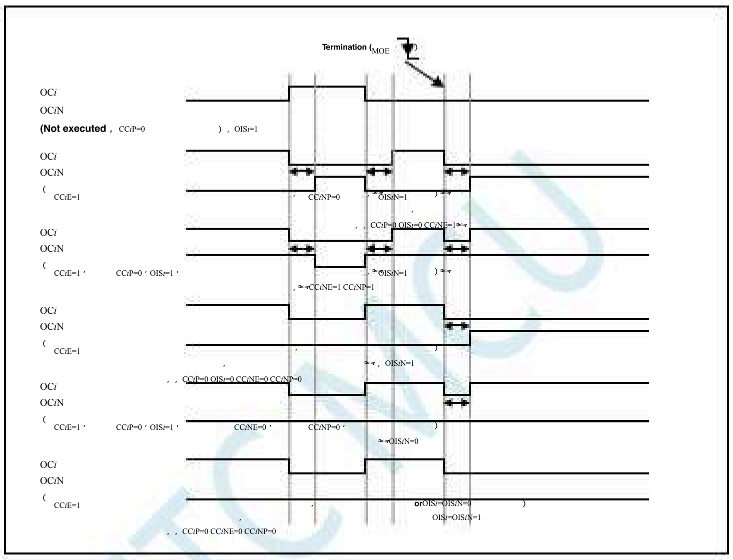
allows users to freeze several configurations

Set parameters (Polarity and state when prohibited , OCi OCiM Configuration, brake enable and polarity) Users can pass PWMA_BKR storage Device of LOCK Bit, choose one of the three levels of protection. MOE After reset LOCK The bit field can only be modified once.

Output of brake response (channel without complementary output)



With complementary transmission The output of the brake response (Complementary output) PWMA


Termination (MOE
OCi
OCiN
(Not executed , CCiP=0) , OISi=1
OCi
OCiN
(CCiE=1 , CCiNP=0 , OISiN=1) Delay
, , CCiP=0 OISi=0 CCiNE=1 Delay
OCi
OCiN
(CCiE=1 , CCiP=0 , OISi=1 , OISiN=1) Delay
, Delay CCiNE=1 CCiNP=1
OCi
OCiN
(CCiE=1
Delay , OISiN=1
, , CCiP=0 OISi=0 CCiNE=0 CCiNP=0
OCi
OCiN
(CCiE=1 , CCiP=0 , OISi=1 , CCiNE=0 , CCiNP=0 , Delay OISiN=0)
OCi
OCiN
(CCiE=1
or OISi=OISiN=0)
OISi=OISiN=1
, , CCiP=0 CCiNE=0 CCiNP=0

21.5.9 Clear when an external event occurs OCiREF signal

For a given channel, at ETRF Input terminal (set PWMA_CCMRi The corresponding signal in the register Bit '1') of OCiCE a high level, it is possible to put OCiREF Signal down , OCiREF will remain low until the next update event occurs UEV° This function only Can be used to output comparison PWM Mode, and cannot be used for mandatory mode.

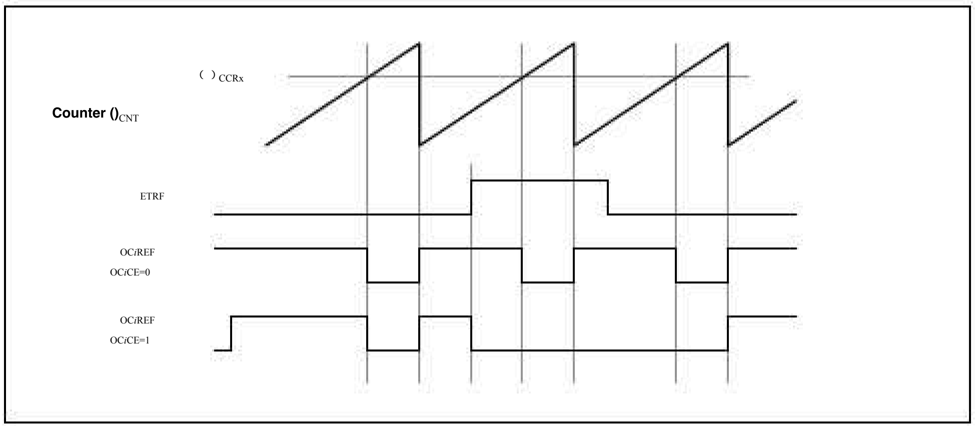
The signal can be connected to the output of a comparator for controlling the current. At this time, Must be configured as follows :

1. ETR For example , OCiREF
2. The externally triggered prescaler must be turned off : PWMA_ETR In the register ETPS[1:0]=00°
3. External clock mode must be disabled : PWMA_ETR2 In the register ECE=0°

External trigger polarity (ETP) and external trigger filter (ETF) Can be configured as needed. When the input becomes high, the corresponding difference is shown in the signal below. ETRF

In this example, the timer is placed in mode. PWMA PWM

ETR clear PWMA of OCiREF



21.5.10 Encoder interface mode

Encoder interface mode is generally used for motor control.

The method of selecting the encoder interface mode is :

If the counter is only there TI2 The edge count, then set PWMA_SMCR In the register SMS=001 ;

, if only there TI1 Edge count, then set SMS=010 ;

If the counter is passing TI1 Edge count, then set SMS=011。

the setting at the same time PWMA_CCER1 and TI2 Bit, you can choose TI1 and TI2 Polarity; if necessary, also

The input filter can be programmed. In the registerof and CC1P CC2P

Two inputs Is used as an interface for incremental encoders. Assume that the counter has been started (In the register PWMA_CR1

CEN=1 and TI1 TI1 and TI2

The signal after passing through the input filter and polarity control. If there is no filtering and polarity conversion, then TI2), the counter counts each time a valid transition occurs on or TI1 And yes TI1FP1 TI2FP2 TI2FP2 TI2FP2=TI2 According to TI1FP1=TI1, The tra

generates a counting pulse and a direction signal. According to the transition sequence of the two input signals, the counter counts up or

down, and the hardware sets the bits accordingly. Register of Regardless of whether the counter depends on counting or relying on PWMA_CR1 DIR TI1

Rely on counting or rely on and counting at the same time, at either input terminal (TI1 Or) The transition will recalculate the bits. TI2 TI1 TI2 TI2 DIR

The encoder interface mode is basically equivalent to using an external clock with direction selection. This means that the counter is onl

PWMA_ARR Continuous counting between the auto-loaded values of the register (depending on the direction, or to 0 Count ARR ARR 0 To count).

So you must configure it before you start counting 。 In this mode, the trap, comparator, prescaler, and repeat counter、

The trigger output characteristics, etc. are still working as usual. The encoder mode and the external clock mode are not compatible, so they PWMA_ARR

In the encoder interface mode, the counter is automatically modified in accordance with the speed and direction of the incremental enco

of the counter always indicates the position of the encoder, and the counting direction corresponds to the direction of rotation of the connect

The following table lists all possible combinations (assuming that and TI1 TI2 Do not change at the same time).

between the counting direction and the encoder signal

Effective edge	Corresponding to the level of the relative signal (TI1FP1 Correspond to TI2, TI2FP2	TI1FP1 signal		TI2FP2 signal	
		rise	Down , down, count,	rise	Drop does not
Only in TI1 count	High	up, count,	not count , up,	count,	count does not count
	low	down, count,	not count		
Only in TI2 count	high	Do not	Do not count,	count up,	count down
	low	count do not	count count,	count down,	count up
in TI1 and TI2 Count up	high	Count down,	count up ,	count down,	count down, count
	low	up, count down,	count down,	count down,	count up

An external incremental encoder can be directly connected to MCU The connection does not require external interface logic. However, generally use a comp

The differential output of the device is converted into a digital signal, which greatly increases the anti-noise interference ability. The third

signal output by the encoder represents the mechanical zero point, which can be connected to an external interrupt input and trigger a counte

The following is an example of a counter operation, showing the generation and direction control of the counting signal. It also shows how input jitter is suppressed when two edges are selected; jitter may occur when the position of the sensor is close to a conversion point. In this example, we assume that the configuration is as follows :

PWMA_CCMR1CC1S=01 (Map to register , IC1FP1 TI1)

CC2S=01 (register , IC2FP2 TI2)

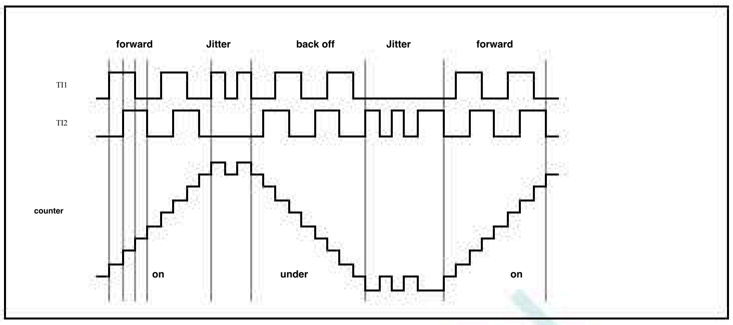
CC1P=0 (register , IC1 Not inverted,

CC2P=0 (register , IC2 Not inverted, mapped to IC1=TI1) IC2=TI2)

SMS=011 (Register, all inputs are valid on the rising and falling edges).

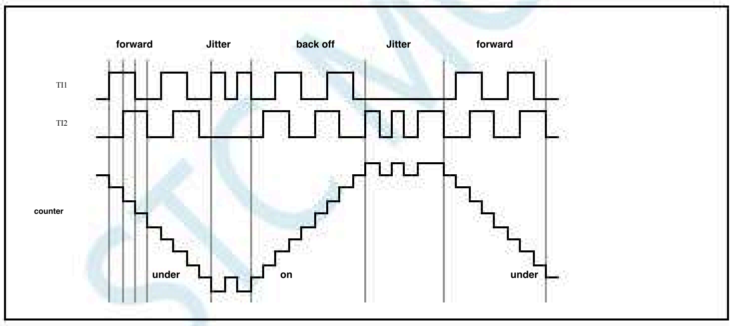
CEN=1 (PWMA_CCMR2 PWMA_CCER1 PWMA_CCER1 PWMA_SMCR PWMA_CR1 Register, counter enabled)

Example of counter operation in encoder mode



The picture below shows the operation of the counter when the polarity is reversed (CC1P=1). A example of the operation of the counter when IC1 is inverted. Other configurations are the same as in the example above)

IC1 Examples of inverted encoder interface modes



When the timer is configured in encoder interface mode, it provides information about the current position of the sensor. Using another t configured in capture mode to measure the interval between the two encoder events, dynamic information (speed, acceleration, deceleration). The encoder output indicating the mechanical zero point can be used for this purpose. According to the interval between the two events, the at a certain time interval. If possible, you can latch the value of the counter to the third input capture register (the capture signal must be peri generated by another timer).

21.6 interrupt

PWMA/PWMB Each has 8 Source of interrupt request：

Brake interrupt

trigger interrupt

COM Event interruption

Input capture output comparison interrupt / 4

Input capture/Output comparison interrupt 3

input capture/Output comparison interrupt 2

input capture/Output comparison interrupt 1

Update event interruption (such as counter overflow, underflow and initialization)

In order to use the interrupt feature, for each interrupt channel being PWMA_IER/PWMB_IER The corresponding interrupt in the register e used, set the energy TIE bit: that is, the bit. CCiIE By setting BIE，COMIE，UIE PWMA_EGR/PWMB_EGR The corresponding bit in the register， You can also use software to generate each of the above interrupt sources.

21.7 PWMA/PWMB Register description

21.7.1 Output enable register (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: Prohibited output PWM8

1: Enable output PWM8

ENO7P : PWM7 Output control bit

0: Prohibited output PWM7

1: Enable output PWM7

ENO6P : PWM6 Output control bit

0: Prohibited output PWM6

1: Enable output PWM6

ENO5P : PWM5 Output control bit

0: Prohibited output PWM5

1: Enable output PWM5

ENO4N : PWM4N Output control bit

0: Prohibited Output PWM4N

1: Enable output PWM4N

ENO4P : PWM4P Output control bit

0: Prohibited output PWM4P

1: Enable output PWM4P

ENO3N : PWM3N Output control bit

0: Prohibited Output PWM3N

1: Enable output PWM3N

ENO3P : PWM3P Output control bit

0: Prohibited Output PWM3P

1: Enable output PWM3P

ENO2N : PWM2N Output control bit

0: Prohibited Output PWM2N

1: Enable output PWM2N

ENO2P : PWM2P Output control bit

0: Prohibited Output PWM2P

1: Enable output PWM2P

ENO1N : PWM1N Output control bit

0: Prohibited Output PWM1N

1: Enable output PWM1N

ENO1P : PWM1P Output control bit

0: Prohibited PWM1P Output

1: Enable PWM1P output

21.7.2 Output additional enable register (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 bits

0 : PWM8 The output is directly determined by control ENO8P

1 : PWM8 ENO8P PWMB_BKR The output of the sub is Joint control

AUX7P : PWM7 determined by the output additional control bits

0 : PWM7 The output is directly determined by control ENO7P

1 : PWM7 ENO7P PWMB_BKR The output of the sub is Joint control

AUX6P : PWM6 determined by the output additional control bits

0 : PWM6 The output is directly determined by control ENO6P

1 : PWM6 ENO6P PWMB_BKR The output of the sub is Joint control

AUX5P : PWM5 determined by the output additional control bits

0 : PWM5 The output is directly determined by control ENO5P

1 : PWM5 ENO5P PWMB_BKR The output of the sub is Joint control

AUX4N : PWM4N determined by the output additional control bits

0 : PWM4N The output is directly determined by control ENO4N

1 : PWM4N The output is composed of ENO4N and PWMA_BKR Joint control

AUX4P : PWM4P Output additional control bits

0 : PWM4P The output is directly determined by control ENO4P

1 : PWM4P PWMA_BKR The output of the sub is determined by Joint control ENO4P

AUX3N : PWM3N Output additional control bits

0 : PWM3N The output is directly determined by control ENO3N

1 : PWM3N The output is composed of ENO3N and PWMA_BKR Joint control

AUX3P : PWM3P Output additional control bits

0 : PWM3P The output is directly determined by control ENO3P

1 : PWM3P PWMA_BKR The output of the sub is determined by Joint control ENO3P

AUX2N : PWM2N Output additional control bits

0 : PWM2N The output is directly determined by control ENO2N

1 : PWM2N The output is composed of ENO2N and PWMA_BKR Joint control

AUX2P : PWM2P Output additional control bits

0 : PWM2P The output is directly determined by control ENO2P

1 : PWM2P PWMA_BKR The output of the sub is determined by Joint control ENO2P

AUX1N : PWM1N Output additional control bits

0 : PWM1N The output is directly determined by control ENO1N

1 : PWM1N The output is composed of ENO1N and PWMA_BKR Joint control

PWM1P Output additional control bits

The output is directly determined by AUX1P : 0 : PWM1P control ENO1P

1 : PWM1P The output is composed of ENO1P and PWMA_BKR Joint control

21.7.3 1 Control register (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

: Automatic pre-loading allowed bits (ARPEn n=A,B)

PWMn_ARR 0 : The register is not buffered, it can be written directly

1 : PWMn_ARR to the register buffered by the pre-loaded buffer

CMSn[1:0] : Select the alignment mode (n= A,B)

CMSn[1:0]	Alignment mode	description
00	Edge alignment mode	The counter counts up or down based on the direction bit (DIR) , and the counter counts up and down alternately.
01	Central alignment mode 1	The output comparison interrupt flag of the channel configured as the output is only set to 1 when the counter counts down.
10	Central alignment mode 2	The counter counts up and down alternately. The output comparison interrupt flag of the channel configured as the output is only set to 1 when the counter counts up.
11	Central alignment mode 3	The counter counts up and down alternately. The output comparison interrupt flag of the channel configured as the output is set to 1 when the counter counts up and down.

Note: When the counter is turned on (CEN=1), It is not allowed to switch from edge alignment mode to center alignment mode.

Note: In the central alignment mode, the encoder mode (SMS=001 , 010 , 011) Must be banned.

: The counting direction of the counter (DIRn n= A,B)

0 : The counter counts up;
1 : The counter counts down.

Note: When the counter is configured for central alignment 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, clear CEN Bit, the counter stops.

URSn : Update the source of the request (n= A,B)

0 : If UDIS If an update event is allowed, any of the following events will generate an update interrupt :

- The register is updated (counter overflow/Underflow)
- Software settings UG bit
- clock/Trigger the update generated by the controller

1 : If UDIS If an update event is allowed, the update interrupt will only be generated when the following events occur, and UIF set :

- The register is updated (counter overflow/Underflow)

UDISn : Updates are prohibited (n= A,B)

0 : Once the following events occur, an update (UEV) Event : will be generated

- Counter overflow/underflow generate
- software update events
- clock/The hardware reset generated by the trigger mode controller is buffered and the registers are loaded with their pre-loaded va

1 : No update event is generated, the shadow register (ARR、PSC、CCRx) Keep their values. If set UG Bit or clock/The trigger controller issues a hardware reset, and the counter and prescaler are reinitialized.

CENn: Allow counter (n= A,B)

0: Disable counter ;

1 : Enable the counter.

Note: It is set up in the software CEN After the bit, the external clock, gating mode, and encoder mode can only work. However, the trigger mode

Set by hardware CEN bit.

21.7.4 2Control register (PWMx_CR2), and real-time trigger ADC

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: First group PWM/PWMA TI1 choose The input pin

0 : PWM1P is connected to TI1 (Input of digital filter) ;

1 : PWM1P、PWM2P and PWM3P The pin is connected to the PWM of TI1。

TI5S: Second group PWM/PWMB of TI5 first set of selections (input of

0 : PWM5 The input pin is TI5 the digital filter) after XOR ;

1 : connected to PWM5、PWM6 and PWM7 The pins are connected to the second group after XOR PWM of TI5

MMSA[2:0]: Main mode selection

MMSA[2:0]	Main mode	description
000	reset	The UG bit of the PWMA_EGR register is used as the trigger output (TRGO). If the trigger input (clock/trigger controller is configured as reset mode) generates a reset, the signal on the TRGO will have a delay counter
001	Enable	enable signal relative to the actual reset and will be used as the trigger output (TRGO). I used to start the ADC so that the control enables the ADC within a period of time. The co enable signal is generated by the logic of the CEN control bit and the trigger input signa mode . Unless the master/slave mode is selected , there will be a delay on TRGO when the counter enable signal is controlled by the trigge Note: When you need to use PWM to trigger the ADC conversion, you need to first set the ADC_POWER, ADC_CHS, and ADC_EPWMT in the ADC_CONTR register. When the PWM generates the TRGO internal signal, the system will automatically set ADC_START to start the AD conversion. Please refer to the sample program for detailed use"Use PWM's CEN to start the PWMA timer and trigger the ADC in real time"
010	update	The update event is selected as the trigger output (TRGO).
011	Compare pulses	Once a capture occurs 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 the trigger output (TRGO)
101	compare	OC2REF signal is used as the trigger output (TRGO)
110	compare	OC3REF signalls used as a trigger output (TRGO)
111	compare	OC4REF signal is used as a trigger output (TRGO)

MMSB[2:0]: Main mode selection

MMSB[2:0]	Main mode	description
000	reset	The UG bit of the PWMB_EGR register is used as the trigger output (TRGO). If the trigger input (clock/trigger controller is configured as reset mode) generates a reset, the signal on the TRGO will have a delay counter
001	Enable	enable signal relative to the actual reset and will be used as the trigger output (TRGO). It is used to start multiple PWM so that the control enables the slave PWM over a period of time. The counter enable signal is generated by the logic of the CEN control bit and the trigger input signal in the gated 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
010	update	input. The update event is selected as the trigger output (TRGO).
011	Compare pulses	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	Compare	OC5REF signal is used as the trigger output (TRGO)
101	compare	OC6REF signal is used as the trigger output (TRGO)
110	compare	OC7REF signalls used as a trigger output (TRGO)
111	compare	OC8REF signal is used as a trigger output (TRGO)

Note: Only the first group PWM of TRGO Can be used to trigger ADC the start , can

Note: Only the second PWM group TRGO be used for the first group of PWM ITR2

COMSn :Capture/Compare the update control selection of the control bit (n=A,B)

0: When When, only CCPCn=1 COMG These control bits are only updated when

1: When when, only when : CCPCn=1 COMG the position is located at These control bits are only updated when the rising edge occurs

CCPCn capture/Compare the pre-loaded control bits (n=A,B)

CCINE , 0 : CCIE , and OCIM Bits are not preloaded

1 : CCIE , CCiP , CCINE , CCiP , CCiNP OCIM and Bits are preloaded; after this bit is set, they are only set COMG

It is updated after the bit.

Note: This bit only works on channels with complementary outputs.

21.7.5 Slave mode control register (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: Trigger input (TRGI) The event on is delayed to allow PWMn With its slave PWM Perfect synchronization between (through TRGO)

TSA[2:0]: Trigger source selection

TSA[2:0]	Trigger source
000	-
001	-
010	Internal trigger ITR2
011	-
100	TI1's 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's edge detector (TI5F_ED)
101	Filtered timer input 1 (TI5FP5)
110	Filtered timer input 2 (TI6FP6)
111	External trigger input (ETRF)

Note: These bits can only be used in SMS=000 Time is changed to avoid incorrect edge detection when changing.

: clock/trigger/Select from mode SMSA[2:0]

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 at the edge
010	Encoder mode 2	of TI2FP2 According to the level of TI2FP2, the counter counts up/down at the
011	encoder mode 3	edge of TI1FP1 According to the level of another input, the counter counts up/down at the edge of TI1FP1
100	Reset mode	and TI2FP2 on the rising edge of the selected trigger input (TRGI)When the counter is reinitialized, and a signal to update the register is generated
101	Gated mode	. When the trigger input (TRGI) is high, the clock of the counter is turned on. Once the trigger input becomes low, the counter stops (but does not reset). The start and stop of the counter are controlled. The
110	Trigger mode	counter starts (but does not reset) on the rising edge of the trigger input , Only the meter TRGI . 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 gated mode. This is because TI1F_ED only outputs a pulse every time TI1F changes , but the gating mode checks the level of the trigger input.

: clock/trigger/Select from mode SMSB[2:0]

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
010	Encoder mode 2	of TI6FP6 According to the level of TI6FP6, the counter counts up/down on the
011	encoder mode 3	edge of TI5FP5 According to the level of another input, the counter counts up/down on the edge of TI5FP5
100	Reset mode	and TI6FP6 on the rise of the selected trigger input (TRGI)Reinitialize the counter at the edge, and generate a signal to update the register
101	Gated mode	. When the trigger input (TRGI) is high, the clock of the counter is turned on. Once the trigger input becomes low, the counter stops (but does not reset). The start and stop of the counter are controlled. The
110	Trigger mode	counter starts (but does 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 gated mode. This is because TI5F_ED only outputs a pulse every time TI5F changes , but the gating mode checks the trigger input.level

21.7.6 External trigger register (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: External trigger ETR The polarity (n= A,B)

0 : Valid for high level or rising edge

1 : valid for low level or falling edge

ECEn: External clock is enabled (n= A,B)

0 : Disable external clock mode

1 : Enable the external clock mode, the clock of the counter is The effective edge drive.

ECE The effect and choice of TRGI Connect to effective edge. The external clock mode is the same (PWMn_SMCR register set 1) note :

SMS=111 Can be used at the same time as the following modes: trigger standard mode; trigger reset mode;

Note: External clock mode 2). In Yes, at this time trigger gating mode. But it must not be connected to (in the register ,

Note: The external clock mode 1 Can't be enabled at the same time as the external clock mode 2, and the external clock input is 1 2

ETPSn : External trigger prescaler, external trigger signal The maximum frequency cannot exceed (MASTER/4. EPRP Prescaler can be used to reduce The frequency when EPRP ETRP It is very useful when the frequency is very high: (n= A,B)

00: The prescaler is turned off

01 : EPRP The frequency of /2

02 : EPRP The frequency of /4

03 : EPRP The frequency of /8

ETFn[3:0] :External trigger filter selection, this bit field defines (n= A,B) The sampling frequency and the length of the digital filter.

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

21.7.7 Interrupt enable register (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 to enable (n= A,B)

0: Prohibit triggering interrupts;

1: Enable to trigger an interrupt.

COMIE: Allow Interrupt (n= A,B) COM

0: Prohibited interrupt ; COM

1: Allow COM interrupt.

CCnIE : Allow capture/Compare interrupts (n=1,2,3,4,5,6,7,8)

0 :No capture/Compare n interrupts; : Allow

1 capture/Compare n interruptions.

UIEn: Allow update interruption (n= A,B)

0: Prohibit update interruption;

1: Allow update interruption.

21.7.8 Status register 1(PWMx_SR1)

symbol	address	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 interrupt mark. Once the brake input is valid, the position is determined by the hardware. If the brake input is invalid, this bit can

0。 (n= A,B)

0: No brake event is generated

1 : a valid level is detected on the brake input

TIFn : Trigger interrupt mark. When a trigger event occurs, the hardware pairs the location. Cleared by the software. (1 0 n= A,B)

0 : No trigger event generation : trigger

1 interrupt waiting for response

COMIFn : COM Interrupt mark. COM Event This bit is set by 1 the hardware. Cleared by the software. 0 (n= A,B)

0: None COM Once an event occurs

COM Interrupt waiting for response 1 :

CC8IF :Capture/compare$_8$ Interrupt mark, refer to: CC1IF description

CC7IF :Capture/compare$_7$ Interrupt mark, refer to: CC1IF description

CC6IF :Capture/compare$_6$ Interrupt mark, refer to: CC1IF description

CC5IF :Capture/compare$_5$ Interrupt mark, refer to: CC1IF description

CC4IF Capture/compare$_4$ Interrupt mark, refer to: CC1IF description

CC3IF Capture/compare$_3$ Interrupt mark, refer to: CC1IF description

CC2IF to: Capture/compare$_2$ Interrupt mark, refer to: CC1IF description

CC1IF to: Capture/compare$_1$ Interrupt mark.

If the channel CC1 Configured as output mode :

This bit is set by the hardware when the counter value matches the comparison value 1 , Except in centrosymmetric mode. It is cleared b

. 0: No match occurred ;

1 : PWMA_CNT The value of and PWMA_CCR1 The value matches.

Note: In the central symmetry mode, when the counter value is When, count up, when the counter value is When counting down (itfrom ARR

Count up to ARR ARR-1, and then by Count down to) 1。 Therefore, for all SMS Bit value, neither of these two values is marked

the mark. But if CCR1>ARR, then when CNT Reach ARR On time , CC1IF 1 set.

if the channel CC1 Configured as input mode :

This bit is set by the hardware when the capture event occurs 1 , It is cleared by the software 0 PWMA_CCR1 Clear 0

Or by reading 0: No input capture generation

1 : The counter value has been captured to PWMA_CCR1

UIFn : Update interrupt flag When an update event is generated, this bit is set by the hardware. (PWMn_SR) It is cleared by the software. 0

1 0 : No update event occurred

1 : The update event is waiting for a response. This bit is set by the hardware when the register is updated

- Ruo PWMn_CR When the , When the counter overflows or underflows UDIS=0
- Ruo PWMn_CR1 register of the register UDIS=0、 URS=0, when set PWMn_EGR Register of UG Bit software pair counting device CNT is reinitialized
- if PWMn_CR1 Register of UDIS=0、 URS=0, when the counter CNT When the triggered event is reinitialized

21.7.9 Status register 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 the capture mark. See also: CC1OF description.

CC7OF Capture/compare 7 Repeat the capture mark. See also: CC1OF description.

CC6OF Capture/compare 6 Repeat the capture mark. See also: CC1OF description.

CC5OF Capture/compare 5 Repeat the capture mark. See also: CC1OF description.

CC4OF Capture/compare 4 Repeat the capture mark. See also: CC1OF description.

CC3OF Capture/compare 3 Repeat the capture mark. See also: CC1OF description.

Capture/compare 2 Repeat the capture mark. See also CC2OF description.

CC1OF :Capture/compare 1 Repeat the capture mark. Only when the corresponding channel is configured as input capture, the mark can be set by

Clear this bit.

0 : No repeated capture is generated; : The

1 value of the counter is captured to PWMA_CCR1 Register time , CC1IF 1 The status is already.

21.7.10 Event generation register (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 the software to generate a Brake event, automatically cleared by the hardware (0 n= A,B)

0 : No action

1 : A braking event is generated. At this time BIF=1, If the corresponding interrupt is turned on (BIE=1), then a corresponding interrupt is generated

TGn : Generate a trigger event. This bit is set by the software to generate a trigger event, which is automatically cleared by the hardware (0 n= A

0: No action

1 : TIF=1 , If the corresponding interrupt is turned on (TIE=1), then a corresponding interrupt is generated

COMGn :Capture/Compare events to generate control updates. This bit is set by the software and automatically cleared by the hardware (0 n= A,B)

0: No action

1 : CCPC=1, Allow updates CCIE、 CCINE、 CCiP、 CCiNP、 OCIM bit.

Note: This bit is only valid for channels with complementary outputs

CC8G : Generate capture/Compare events. Reference CC1G Description Description

CC7G : Generate capture/Compare events. reference CC1G Description

CC6G : Generate capture comparison/ event. reference CC1G Description 6

CC5G : Generate capture/Compare events. Reference CC1G Description

CC4G : Generate capture/Compare events. Reference CC1G Description

CC3G : Generate capture/Compare events. Reference CC1G Description

CC2G : Generate capture/Compare events. reference CC1G Description

CC1G: Generate capture comparison event. Generate capture/compare 1 event. This bit is set by the software to generate a capture/Compare events , Automatically cleared by the hardware. 0

0: No action;

1: In the channel CC1 Generate a capture on/Compare events.

If the channel CC1 Configured as output: set CC1IF=1 , If the corresponding interrupt is turned on, a corresponding interrupt will be generated

If the channel CC1 Configured as input: the current counter value is captured to PWMx_CCR1 Register, set CC1IF=1, If turned on

The corresponding interrupt generates a corresponding interrupt. if CC1IF Already, then set CC1OF=1。

UGn : An update event is generated. This bit is set by the software and automatically cleared by the hardware. 0 (1 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 (but the prescaler coefficient remains unchanged) 。 If in centrosymmetric mode or DIR=0 (Count up)

Then the counter is cleared; if DIR=1 (Count down) then the counter takes PWMx_ARR The value of.

capture/Comparison mode register 1 (PWMx_CCMR1) 21.7.11

The channel can be used to capture the input mode or compare the output mode, and CCnS Bit definition. The role of other bits of this register the direction of the channel is different from the corresponding input and output modes. Describes the function of the channel in the output mode, Describes the channel in the input mode Work under OCxx can. Therefore, it must be noted that the functions of the same bit in the output mode and the input mode are different.

The channel is configured to compare the 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 comparison Clear to enable. This bit is used to enable the use PWMETI External events on the pin to clear the channel n The output signal (OCnREF) (n=1,5)

0: OCnREF Not affected Impact of input; ETRF

1: Once detected ETRF Input high level, OCnREF=0.

OCnM[2:0]: Output comparison OCnREF Bits define the output reference signal The action, and OCnREF Decided OCn The value of. OCnREF mode. the n CCnP the position. (n=1,5)

OCnM[2:0]	3 Is the high level valid, and the effective level depends on Oc	description
000	Set channel n when the	The comparison between PWMn_CCR1 and PWMn_CNT has no effect on OCnREF
001	pattern freezes and matches Set the channel n when	When PWMn_CCR1=PWMn_CNT, OCnREF output is high
010	the output is a valid level match The output is an invalid level	When PWMn_CCR1=PWMn_CNT, OCnREF output is low
011	Flip forced	When PWMn_CCR1=PWMn_CNT, flip OCnREF
100	to invalid level forced	to force OCnREF to be low
101	to valid level	and force OCnREF to be high
110	PWM mode 1	when counting up, when PWMn_CNT<PWMn_CCR1 OCnREF output is high, otherwise OCnREF output is low when counting down, when PWMn_CNT>PWMn_CCR1 OCnREF output is low, otherwise OCnREF output is high when
111	PWM mode 2	counting up, when PWMn_CNT<When PWMn_CCR1, the OCnREF output is low, otherwise the OCnREF output is high . When counting down, when PWMn_CNT>PWMn_CCR1, the OCnREF output is high, otherwise the OCnREF output is low.

3 The level is set to (PWMn_BKR 1 Note: once In the register LOCK LOCK bit) and CCnS=00 (This channel is configured as Output) Then this bit cannot be modified.

PWM 2 Note: Before pattern In, only if the comparison result changes or from frozen mode in the output comparison mode switching to PWM Or mode 1 The level just changed.

Note: On channels with complementary outputs, these bits are preloaded. If PWMn_CR2 Register of CCPC=1, OCM bit Mode time PWMOCnREF Only in COM When an event occurs, a new value is taken from the preload bit.

OCnPE : Output comparison Pre-loaded to enable (n=1,5) The preload function of

0: Prohibited PWMn_CCR1 the register can be written at any time PWMn_CCR1 Register, and the newly written value

It works immediately.

1: Open PWMn_CCR1 The pre-loading function of the register, the read and write operation only operates on the pre-loaded register, PWMn_CCR1 The pre

The loaded value is loaded into the current register when the update event arrives.

Note: once LOCK In the register LOCK bit) and CCnS=00 (This channel is configured as The level is set to (3

Output) Then this bit cannot be modified.

Note2: In order to operate correctly, in The preload function must be enabled in mode. But in single pulse mode (PWMn_CR1 storage

Device of OPM=1 PWM), it is not necessary.

OCnFE : The output is relatively fast to enable. This bit is used to speed up Outputs a response to the triggered input event.

0: According to the counter and The value of CCn Normal operation, even if the trigger is turned on. When the input of the trigger has a valid

Along the time, activate The minimum delay of the output is a clock cycle.

1 : The effective edge entered into the trigger acts as if a comparison match has occurred. Therefore, set the level with the comparison

OC The result is irrelevant. The effective edge sum of the sampling trigger The delay between outputs is shortened to clock cycle. OCFE Only in the channel

It works when it is configured as or mode. PWMA PWMB

CC1S[1:0] :Capture/Compare 1 options. These two digits 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 and IC1 is mapped on TRC.
11	input	This mode only works when the internal flip-flop input is selected (selected by the TS bit of the PWMA_SMCR register)

CC5S[1:0] :Capture/Compare options. These two digits define the direction of the channel (input/Output), and the selection of input pins 5

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 flip-flop input is selected (selected by the TS bit of the PWM5_SMCR register)

note : CC1S Only when the channel is closed (PWMA_CCER1 Register of CC1E=0) Is writable.

note : CC5S only when the channel is closed (PWMB_CCER1 the register of CC5E=0) Is writable.

The channel is configured to capture the input mode

symbol	address	B7	B6	B5	B4		B2		B0
PWMA_CCMR1	FEC8H	IC1F[3:0]				B3 IC1PSC[1:0]		B1 CC1S[1:0]	
PWMB_CCMR1	FEE8H	IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]	

ICnF[3:0] : Input capture filter selection, this bit field defines TIn The sampling frequency and the length of the digital filter. （ n=1,5）

ICnF$_{[3:0]}$	clocknumber	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 non-preloaded and will not be considered CCR2 (PWMn_CR2 storage

The value of

the device) ICnPSC[1:0] Input/Capture the prescaler. These two define CCn Input (IC1) The prescaler coefficient. （ n=1,5）

00 : No prescaler, every edge detected on the capture input port triggers a capture

2 Each event triggers a capture 01:

each: Each event

4 triggers a capture once 10

8 : Each event triggers a capture once

CC1S[1:0] 11 :Capture/Compare 1 options. These two digits 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 and IC1 is mapped on TRC.
11	input	This mode only works when the internal flip-flop input is selected (selected by the TS bit of the PWMA_SMCR register)

CC5S[1:0] :Capture/Compare options. These two digits define the direction of the channel (input/Output), and the selection of input pins 5

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 flip-flop input is selected (selected by the TS bit of the PWM5_SMCR register)

note : CC1S Only when the channel is closed (PWMA_CCER1 Register of CC1E=0) Is writable.

note : CC5S only when the channel is closed (PWMB_CCER1 the register of CC5E=0) Is writable.

capture/Comparison mode register 2 (PWMx_CCMR2)

21.7.12 The channel is configured to compare the 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 comparison Clear to enable. This bit is used to enable the use PWMnETR External events on the pin to clear the channel The output signal (OCnREF) (n=2,6)

0 : OCnREF Not affected by input ; ETRF

1: Once detected ETRF Input high level , OCnREF=0。

OCnM[2:0]: Output comparison Mode, reference OC1M。 (n=2,6)

OCnPE: Output comparison 2 is pre-loaded and enabled, refer to OC1PE。 (n=2,6)

CC2S[1:0]:Capture comparison 2 choose. These two digits 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	and IC2 is mapped on TRC.

CC6S[1:0] :Capture/Compare options. These two digits define the direction of the channel (input/Output), and the selection of input pins 6

CC6S[1:0]	Direction	Input pin
00	output	
01	input	IC6 is mapped on TI6FP6,
10	input	IC6 is mapped on TI5FP6,
11	input	and IC6 is mapped on TRC.

The channel is configured to capture the 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 filter selection, reference IC1F。 (n=2,6)

ICnPSC[1:0]: Input capture / n Prescaler, refer to IC1PSC。 (n=2,6)

CC2S[1:0] 2 :Capture/Compare options. These two digits 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	and IC2 is mapped on TRC.

CC6S[1:0]:Capture comparison 6 choose. These two digits 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	and IC6 is mapped on TRC.

capture/Comparison mode register 3 (PWMx_CCMR3)

21.7.13 The channel is configured to compare the 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 comparison Clear to enable. This bit is used to enable the use of PWMETI External events on the pin to clear the channel The output signal (OCnREF) (n=3,7)

0: OCnREF Not affected by input ETRF;

1: Once detected ETRF Input high level, OCnREF=0.

OCnM[2:0]: Output comparison Mode, reference OC1M. (n=3,7)

OCnPE: Output comparison is pre-loaded and enabled, refer to OC1PE. (n=3,7)

OCnFE: Output comparison 3 fast enable, refer to OC1FE.

CC3S[1:0]: Capture/Compare 3 options. These two digits 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 mapping on TI3FP3
10	input	IC3 mapping on TI4FP3
11	input	IC3 mapping on TRC.

CC7S[1:0]:Capture comparison 7 choose. These two digits 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	and IC7 is mapped on TRC.

The channel is configured to capture the input mode

symbol	address	B7	B6	B5	B4		B2		B0
PWMA_CCMR3	FECAH	IC3F[3:0]				B3 IC3PSC[1:0]		B1 CC3S[1:0]	
PWMB_CCMR3	FEEAH	IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]	

ICnF[3:0] : Input capture filter selection, reference IC1F。（ n=3,7）

: Input/Capture the prescaler, refer to ICnPSC[1:0] IC1PSC。（ n=3,7）

CC3S[1:0] :Capture/Compare options. These two digits define the direction of the channel (input/Output), and the selection of input pins 3

CC3S[1:0]	Direction	Input pin
00	output	
01	input	IC3 is mapped on TI3FP3,
10	input	IC3 is mapped on TI4FP3,
11	input	and IC3 is mapped on TRC.

CC7S[1:0] :Capture comparison 7 choose. These two digits 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	and IC7 is mapped on TRC.

21.7.14 capture/Comparison mode register 4 (PWMx_CCMR4)

The channel is configured to compare the output mode

symbol	address	B7	B6	B5	B4	B3	B2		B0
PWMA_CCMR4	FECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	B1 CC4S[1:0]	
PWMB_CCMR4	FEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]	

OCnCE: Output comparison n Clear to enable. This bit is used to enable the use PWMETI External events on the pin to clear the channel. The output signal （ OCnREF）（ n=4,8）

0：OCnREF Not affected by input ; ETRF

1: Once detected ETRF Input high level , OCnREF=0。

OCnM[2:0]: Output comparison n Mode, reference OC1M。（ n=4,8）

OCnPE: Output comparison n Pre-loaded to enable, refer to OC1PE（ n=4,8）

CC4S[1:0] :Capture/Compare 4 options. These two digits 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	and IC4 is mapped on TRC.

CC8S[1:0] :Capture comparison 8 choose. These two digits 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	and IC8 is mapped on TRC.

The channel is configured to capture the 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 filter selection, reference IC1F。 (n=4,8)

ICnPSC[1:0]: Input capture / n Prescaler, refer to IC1PSC。 (n=4,8)

CC4S[1:0] 4 :Capture/Compare options. These two digits 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	and IC4 is mapped on TRC.

CC8S[1:0] :Capture/Compare options. These two digits define the direction of the channel (input/Output), and the selection of input pins 8

CC8S[1:0]	Direction	Input pin
00	output	
01	input	IC8 is mapped on TI8FP8,
10	input	IC8 is mapped on TI7FP8,
11	input	and IC8 is mapped on TRC.

21.7.15 capture/Compare the enable register (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 the output polarity. Reference CC1P

CC6E : OC6 input capture/The comparison output is enabled. Reference CC1E

CC5P : OC5 input capture/Compare the output polarity. Reference CC1P

CC5E : OC5 input capture/The comparison output is enabled. reference CC1E

CC2NP : OC2N Compare the output polarity. The reference CC1NP

CC2NE : OC2N comparison output is enabled. reference CC1NE

CC2P : OC2 Input capture/Compare the output polarity. reference CC1P

CC2E : OC2 Input capture/The comparison output is enabled. reference CC1E

CC1NP : OC1N Compare output polarity

0: The high level is valid;

1: the low level is valid.

Note: once Level (PWMA_BKR 1 LOCK In the register LOCK Bit) Set to or and 3 CC1S=00(The channel configuration is Output), then this bit cannot be modified.

Note: For channels with complementary outputs, this bit is preloaded. if CCPC=1 (PWMA_CR2 Register) , only in COM The bit takes the new value from the pre-loaded bit. When the incident occurred ,

CC1NE : OC1N Compare output enable

0: Turn off the comparison output

1 : Turn on the comparison output, the output level depends on OSSR、MOE、OIS1、OIS1N The value of the bit. and CC1E

Note: For channels with complementary outputs, this bit is preloaded. if CCPC=1 (PWMA_CR2 Register), only in COM When the incident occurred , The bit takes the new value from the pre-loaded bit.

CC1P : OC1 Input capture/Compare output polarity CC1NE

CC1 The channel is configured as an output :

0: Valid at high level

1: Valid at low level

CC1 The channel is configured as input or capture :

0:The capture occurred in TI1F or TI2F The rising edge of;

1:The capture occurred in TI1F or TI2F The falling edge of

CC1E : OC1 Input capture/Compare output enable

0 : Turn off input capture/Compare output;

1 : Turn on input capture/Compare the output.

Note: once Level (PWMA_BKR LOCK In the register LOCK Bit) set to or 3 , then this bit cannot be modified.

Note: For channels with complementary outputs, this bit is preloaded. if CCPC=1 (PWMA_CR2 Register) , only in COM When the incident occurred , The bit takes the new value from the pre-loaded bit.

Complementary output channels OCi and OCiN with brake function The control bit

Control bit					Output status	
MOE	OSSI	OSSRCCiE	CCiNE		OCi Output status	Output status OCiN
1	X	0	0	0	Output forbidden	Output is prohibited
		0	0	1	output forbidden	Polar OCiREF
		0	1	0	Polar with OCiREF	Output is prohibited
		0	1	1	polarity and dead zone OCiREF	Reverse with polarity and dead zone OCiREF
		1	0	0	Output disable	Output is prohibited
		1	0	1	off state (The output is enabled and at an invalid level) OCi=CCiP	Polar OCiREF
		1	1	0	Polar OCiREF	Off state (output is enabled and at an invalid level) OCiN=CCiNP
		1	1	1	With polarity and dead zone OCiREF	Reverse with polarity and dead zone OCiREF
0	0	X	X	X	Output is prohibited	
	1				Off state (the output is enabled and at an invalid level) asynchronously: then, if the clock exists: after a dead time the output is enabled and at an invalid level) asynchronously: then, if the clock exists: after a dead time OCi=CCiP, OCiN=CCiNP; with OISi and OCi=OISi, OCiN=OISiN, assuming The effective level of. OCiN	

Note: The pins are connected OCi and OCiN to complementary registers. Outside of the channel I/O Not all correspond OISiN OCi And channel status and OCiN GPIO OCi The status of the pins depends on

21.7.16 capture/Compare the enable register (2 PWMx_CCER2)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER2	FECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E
PWMB_CCER2	FEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E

CC8P : OC8 Input capture/Compare the output polarity. Reference CC1P

CC8E : OC8 input capture/The comparison output is enabled. Reference CC1E

CC7P : OC7 input capture/Compare the output polarity. Reference CC1P

CC7E : OC7 input capture/The comparison output is enabled. reference CC1E

CC4NP : OC4N Compare the output polarity. The reference CC1NP

CC4NE : OC4N comparison output is enabled. reference CC1NE

CC4P : OC4 Input capture/Compare the output polarity. reference CC1P

CC4E : OC4 Input capture/The comparison output is enabled. reference CC1E

CC3NP : OC3N Compare the output polarity. The reference CC1NP

CC3NE : OC3N comparison output is enabled. reference CC1NE

CC3P : OC3 Input capture/Compare the output polarity. reference CC1P

CC3E : OC3 Input capture/The comparison output is enabled. reference CC1E

21.7.17 8 Counter high bit (PWMx_CNTRH)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRH	FECEH	CNT1[15:8]							
PWMB_CNTRH	FEEEH	CNT2[15:8]							

CNTn[15:8] : The high value of the counter (8 n= A,B)

21.7.18 8 Counter low bit (PWMx_CNTRL)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRL	FECFH	CNT1[7:0]							
PWMB_CNTRL	FEEFH	CNT2[7:0]							

CNTn[7:0] :The counter is low 8 Bit value (n= A,B)

21.7.19 8 Prescaler high bit (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]							

: The high value of the prescaler. (PSCn[15:8] 8 n= A,B)

The prescaler is used to pair CK_PSC Divide by frequency. The clock frequency of the counter (fCK_CNT) Equal to (fCK_PSC/ PSCR[15:0]+1).

PSCR Contains the value written to the current prescaler register when the update event is generated (the update event includes the c TIM_EGR

Or cleared by the slave controller operating in reset mode). This means that in order for the new value to work, it must be gene 0

Bit clear UG

An update event 0。

PWM Output frequency calculation formula

PWMA and Two groups PWM PWMB The output frequency calculation formula is the same, and each group can be set to a different frequency.

Alignment mode	PWM Output frequency calculation formula
Edge alignment	PWM Output frequency = System operating frequency SYSclk / ((PWMx_PSCR + 1) × (PWMx_ARR + 1))
Middle alignment	PWM Output frequency = System operating frequency SYSclk / ((PWMx_PSCR + 1) × PWMx_ARR × 2)

21.7.20 8 Prescaler low bit (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 low value of the prescaler. (8 n= A,B)

21.7.21 Automatic reloading register high 8 Bit (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]: High automatic reloading 8 Bit value (n= A,B)

ARR Contains the value to be loaded into the actual automatic reload register. When the value of automatic reloading is 0, The counter i

21.7.22 Automatic reloading register low 8 Bit (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]: Low automatic reloading 8 Bit value (n= A,B)

21.7.23 Repeat counter register (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 the preload function is turned on, these bits allow the user to set the ratio The update rate of the higher register (that is, it is transm

Input to the current register); if an update interrupt is allowed, it will also affect the rate at which an update interrupt is

generated. Start Counting, each time you count down the counter value by REP_CNT 0 REP_CNT REP Reached, an update event will be generated and the counter Re-from

Because the value is only overloaded when a periodic update event occurs, write to the register REP_CNT U_RC REP PWMn_RCR

The new value entered will only take effect when the next periodic update event occurs. This means that in the pattern (PWM REP+1)

Corresponds to: -In edge alignment mode , PWM The number of cycles;

-in the central symmetry mode , PWM Number of half-cycles ;

/Capture comparison register 1/5 high 8 Bit (PWMx_CCR1H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	FED5H	CCR1[15:8]							
PWMB_CCR5H	FEF5H	CCR5[15:8]							

:Capture/Relatively high 8 Bit value (n=1,5) CCRn[15:8]

if CCn The channel is configured as Output: CCRn Contains the current comparison value of the load (preload value). If in PWMn_CCMRn storage

Device (OCnPE If the preload function is not selected in the bit), the written value will be immediately transferred to the current regis

, this preload value is transmitted to the current capture only when an update event occurs / In the comparison register. The current c

The value is compared, and an output signal is generated on the port. OCn

if CCn The channel is configured as Input: CCRn Contains the counter value when the last input capture event occurred (at this time, this register

Read) °

21.7.24 /Capture comparison register 1/5 low 8 Bit (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 comparison n 8 The low value (n=1,5)

21.7.25 /Capture comparison register 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 comparison n The height of 8 Bit value (n=2,6)

21.7.26 /Capture comparison register 2/6 low 8 Bit (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 comparison n 8 The low value (n=2,6)

21.7.27 /Capture comparison register 3/7 high 8 Bit (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 /Relatively high n 8 Bit value (n=3,7)

21.7.28 /Capture comparison register 3/7 low 8 Bit (PWMx_CCR3L)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3L	FEDAH	CCR3[7:0]							
PWMB_CCR7L	FEFAH	CCR7[7:0]							

CCRn[7:0] :Capture comparison n 8 The low value (n=3,7)

21.7.29 /Capture comparison register 4/8 high 8 Bit (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 /Relatively high n 8 Bit value (n=4,8)

21.7.30 /Capture comparison register 4/8 low 8 Bit (PWMx_CCR4L)

symbol	address	B7	B6	B5		B3	B2	B1	B0
PWMA_CCR4L	FEDCH	B4							
PWMB_CCR8L	FEFCH	CCR4[7:0] CCR8[7:0]							

CCRn[7:0]:Capture comparison n 8 The low value (n=4,8)

21.7.31 Brake register (PWMx_BKR)

symbol	address	B7	B6	B5	B4	B3	B2		B0
PWMA_BKR	FEDDH	MOEA	AOEA	BKPA	BKEA	OSSRA	OSSIA	B1 LOCKA[1:0]	
PWMB_BKR	FEFDH	MOEB	AOEB	BKPB	BKEB	OSSRB	OSSIB	LOCKB[1:0]	

MOEn : The main output is enabled. Once the brake input is valid, this bit is asynchronously cleared by the hardware. According to AOE The setting value of the bit, which can be 0 determin

Pieces are set or automatically set. It is only valid for channels configured as outputs. (1 1 n= A,B)

0: Prohibited Output or force to idle state and OC OCN

1 : If the corresponding enable bit is set (PWMn_CCERX Register of CCIE Bit) , then enable OC and OCN output.

AOEn: Automatic output is enabled (n= A,B)

1 Can only be set by software ; 0 : MOE

1 : MOE Or be automatically set in the next update event (if the brake input is invalid).

Note: Once it can be set by the software 1 Level In the register (PWMn_BKR LOCK LOCK Bit) is set to 1, then the bit cannot be modified

BKPn: Brake input polarity (n= A,B)

0 : The brake input is valid at low level

1 : the brake input is valid at high level

Note: once LOCK Level (PWMn_BKR In the register LOCK Bit) is set to 1, then the bit cannot be modified

BKEn: The brake function is enabled (n= A,B)

0: Disable brake input (BRK)

1: Turn on the brake input (BRK)

Note: once Level (PWMn_BKR LOCK In the register LOCK Bit) is set to 1, then the bit cannot be modified.

OSSRn : Select "off state" in operation mode. The position is MOE=1 And it is valid when the channel is set to output (n= A,B)

0: When When not working, it is forbidden PWM Output (OC/OCN OC/OCN Enable output signal =0) ;

1: When PWM When not working, once CCiE=1 Or CCiNE=1 , first open OC/OCN And outputs an invalid level, and then sets OC/OCN enable the output signal =1。

Note: once Level (PWMn_BKR LOCK In the register LOCK Bit) is set to 2, then the bit cannot be modified.

OSSIn : Select "off state" in idle mode. The position is MOE=0 And it is valid when the channel is set to output. (n= A,B)

0: When PWM When not working, when not OC/OCN Output (OC/OCN Enable output signal =0) ;

1: When PWM working is prohibited, once the CCiE=1 or CCiNE=1 , OC/OCN First output its idle level, and then OC/OCN output signal is enabled =1。

Note: once LOCK Level (PWMn_BKR In the register LOCK Bit) is set to 2, then the bit cannot be modified.

: Lock settings. The write protection measures provided by this bit to prevent software errors (LOCKn[1:0] n= A,B)

LOCKn[1:0]	Protection level	Protect content
00	Unprotected	Register without write protection
01	lock level 1	The BKE, BKP, and AOE bits of the PWMn_BKR register cannot be written, and the OISI bits of the
10	Lock level 2	PWMn_OISR register cannot be written to everyone in lock level 1, nor can they be written to the CC polarity bits and the OSSR/OSSI
11	Lock level 3	bits cannot be written to everyone in lock level 2, nor can they be written to the CC control bits.

note : Due to BKE、BKP、AOE、OSSR、OSSI Bits can be locked (depends on LOCK , So write the bit for the first time) PWMn_BKR They must be set when registering.

21.7.32 Dead zone register (PWMx_DTR)

symbol	address	B7	B6	B5		B3	B2	B1	B0
PWMA_DTR	FEDEH	B4							
PWMB_DTR	FEFEH	DTGA[7:0] DTGB[7:0]							

: Dead zone generator setting. (n= A,B) DTGn[7:0]

These bits define the duration of the dead zone between insertion and output. (t_{CK_PSC} PWMn Clock pulse)

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}

21.7.33 Output idle status register (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: When idle Output level OC8

OIS7: When idle OC7 Output level

OIS6: When idle OC6 Output level

OIS5: When idle OC5 Output level

OIS4N: When idle Output level OC4N

OIS4: When idle OC4 Output level

OIS3N: When idle Output level OC3N

OIS3: When idle OC3 Output level

OIS2N: When idle Output level OC2N

OIS2: When idle OC2 Output level

OIS1N: When idle Output level OC1N

0: When MOE=0 When, it is after a dead time, OC1N=0 ;

1: When MOE=0 when, it is after a dead time , OC1N=1。

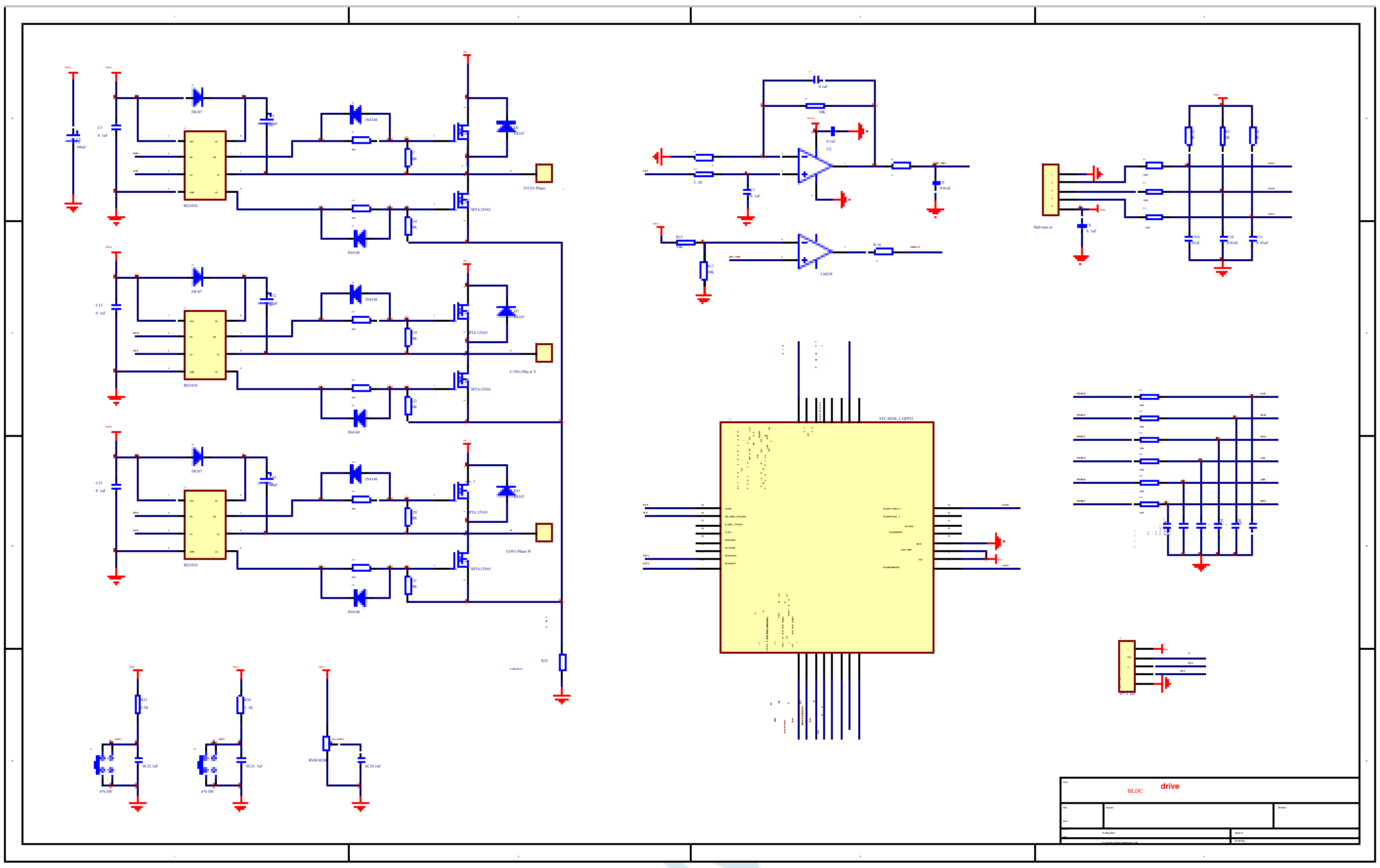
OIS1: When idle Output level OC1

0: When MOE=0 When, if OC1N If enabled, after a dead zone , OC1=0;

1: When MOE=0 When, if OC1N If enabled, after a dead zone , OC1=1。

21.8 Sample program

21.8.1 Six steps PWM Drive brushless DC motor (with HALL)



Language code C

```
// The test operating frequency 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;                              /*! < 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             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)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                     //Serial data input
#define         RCLK                    P24                     // Clock pulse signal-valid on the rising edge
#define         SCLK                    P25                     // Incoming signal----The rising edge is valid


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 Choose
        PWMB_PS = 0xd5;                     //PWMB pin choose

/************************************************************
Output comparison mode PWMx_duty = [CCRx/(ARR + 1)]*100
************************************************************/
/************PWMB         Pick up    sensor      *****************/
///////////Time base unit///////////
        TIM2-> PSCRL = 15;
        TIM2-> ARRH = 0xff;                 // Automatic reloading of registers, counters point
        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 &  ///////////
```

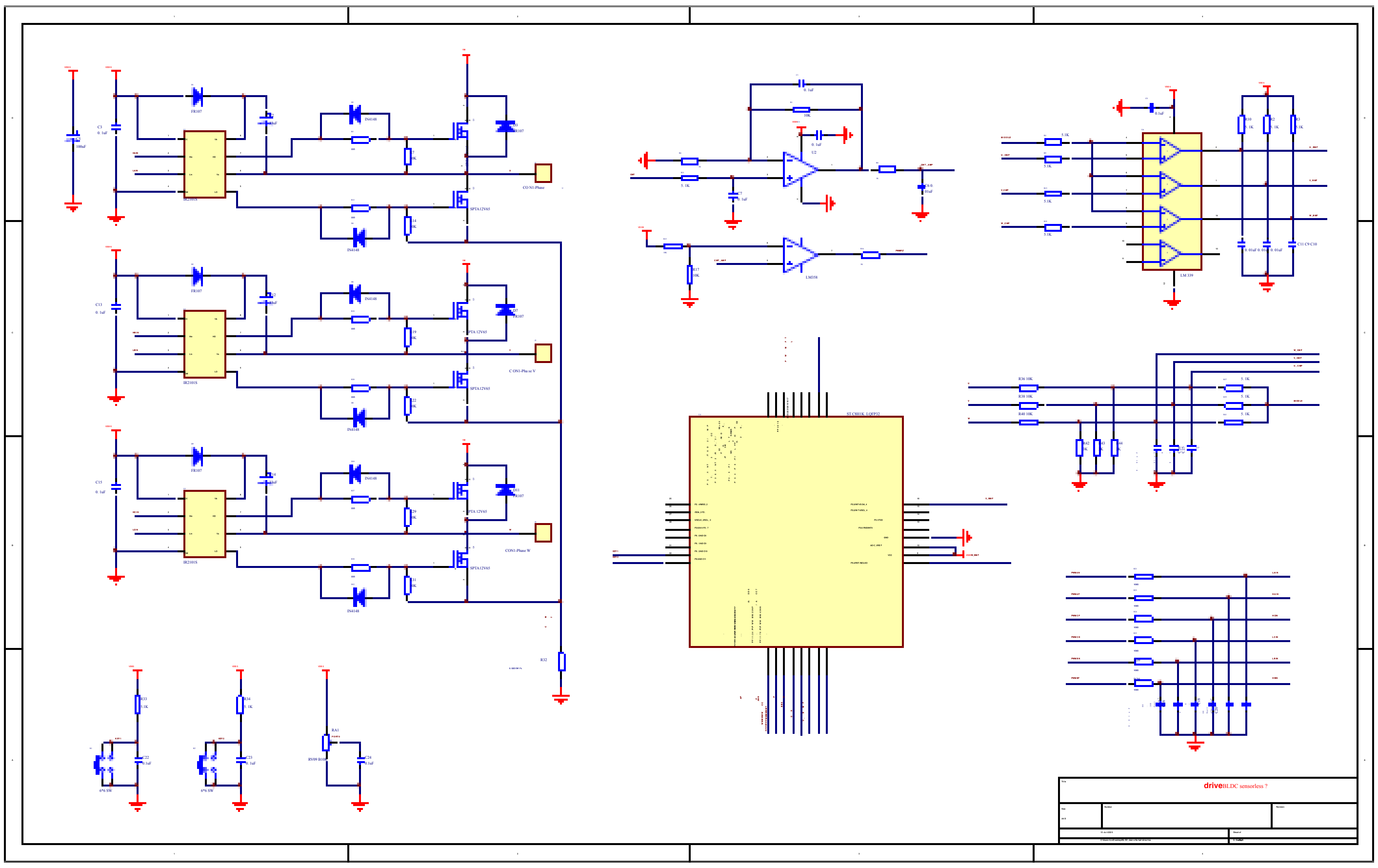
```
        TIM2-> BKR = 0x80;                                  //Main output enabled
        TIM2-> IER = 0x02;                                  //Enable interrupt

/************PWMA Control motor commutation
//////////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;                                // Configure channel output enable and polarity
        TIM1-> CCER2 = 0x01;                                // Configure channel output enable and polarity
        TIM1-> OISR = 0xAA;                                 //configuration MOE=0 Output level of each channel at the time

//////////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_RESL>>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;                             // The main output is enabled, which is equivalent to the main 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;
}
```

21.8.2 BLDC Brushless DC motor drive(none HALL)



C Language code

```
// The test operating frequency is 11.0592MHz
// This routine implements the following functions: Through the group control without hall Demonstration of channel
// This routine is only applicable STC8G The motor is 24V motor operation under no load conditions

#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 Choose
        PWMB_PS = 0xD5; //PWMB pin choose

        /************************************************************
        Output comparison mode PWMx_duty = [CCRx/(ARR + 1)]*100
        *************************************************************/
                          BMF /*******input **PWMB*****************/
        ///////////      Time base unit///////
        TIM2-> PSCRL = 15;
        TIM2-> ARRH = 0xff;                         // Automatic reloading of registers, counters 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;
                    Interrupt configuration enable ///////////
TIM2-> BKR =& 0x80;                                     //Main output enabled
TIM2-> IER = 0x02;                                      //Enable interrupt
/************PWMA       Control motor commutation
///////////     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;                                    // Configure channel output enable and polarity
TIM1-> CCER2 = 0x01;                                    // Configure channel output enable and polarity
TIM1-> OISR = 0xAA;                                     //configuration MOE=0   Output level of each channel at the time
///////////     Mode configuration
TIM1-> CR1 = 0xA0;
TIM1-> CR2 = 0x24;
TIM1-> SMCR = 0x20;
TIM1-> BKR = 0x0c;
                    Interrupt configuration enable ///////////
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 regulation
                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 the 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) //Commutation              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;                                                 1      pattern //The timer is 1T
        TMOD = 0x20;                                           //Timer as mode 1   0(16  )Automatic bit reloading


        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_RESL>>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);                    ADC// Speed control knob 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;                                       // The main output is enabled, which is equivalent to the main 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;
}
```

Quadrature encoder mode 21.8.3

C Language code

```
//The test operating frequency is11.0592MHz

#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;

unsigned char cnt_H, cnt_L;

void main(void)
{
        P_SW2 = 0x80;

        P1M1 = 0x0f;
        P1M0 = 0x00;

        PWMA_ENO = 0x00;                                 TRGI // Configured pin Need to be turned off ENO Correspond to bit And equipped with input
        PWMA_PS = 0x00;                                  to //00:PWM at P1

        TIM1-> PSCRH = 0x00;                             // Prescaler register
        TIM1-> PSCRL = 0x00;

        TIM1-> CCMR1 = 0x21;                             // The channel mode is configured as input and connected to the encoder, filter clock
        TIM1-> CCMR2 = 0x21;                             // The channel mode is configured as input and connected to the encoder, filter clock

        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 reading 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;
        }
}
```

Single pulse mode (trigger control pulse output) 21.8.4

C Language code

```
//The test operating frequency 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;                                        output PWM //IO
        PWMA_PS = 0x00;                                         //00:PWM at P1

        /*************************************************************
        PWMx_duty = [CCRx/(ARR + 1)]*100
        *************************************************************/
                    Need to be turned off Correspond to and equipped with input
                    of // Configured to TRGI
        pin TIM1-> PSCRH = 0x00;                                // Prescaler register
        TIM1-> PSCRL = 0x00;                                    // Dead time configuration
        TIM1-> DTR = 0x00;                                      // Channel mode configuration
        TIM1-> CCMR1 = 0x68;                                    // Configured as an input channel
        TIM1-> CCMR2 = 0x01;
        TIM1-> CCMR3 = 0x68;
        TIM1-> CCMR4 = 0x68;

        TIM1-> SMCR = 0x66;

        TIM1-> ARRH = 0x08;                                     // Automatic reloading of registers, counters 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;                                      // The main output is enabled, which is equivalent to the 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;
        }
}
```

Gated mode (input level enables counter) 21.8.5

C Language code

```
//The test operating frequency 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;                                    output PWM //IO
        PWMA_PS = 0x00;                                     //00:PWM at P1

/************************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
************************************************************/
                  Need to be turned off ENO Corresponding Configured to // and equipped with input pin TRGI

        pin TIM1-> PSCRH = 0x00;                            //Prescaler register
        TIM1-> PSCRL = 0x00;                                //Dead time configuration
        TIM1-> DTR = 0x00;                                  //Channel mode configuration
        TIM1-> CCMR1 = 0x68;                                //Configured as an input channel
        TIM1-> CCMR2 = 0x68;
        TIM1-> CCMR3 = 0x68;
        TIM1-> CCMR4 = 0x68;
        TIM1-> SMCR = 0x75;                                 //Gated trigger mode ETRF input
        TIM1-> ARRH = 0x08;                                 // Automatic reloading of registers, counters 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;                                  // The main output is enabled, which is equivalent to the 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;
        }
}
```

External clock mode 21.8.6

C Language code

```
//The test operating frequency 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;                                              PWM //IO output
        PWMA_PS = 0x00;                                               //00:PWM at P1

/************************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
************************************************************/

                        Need to be turned of Correspond and equipped with input
                of // Configured to TRGI
        pin TIM1-> PSCRH = 0x00;                                      //Prescaler register
        TIM1-> PSCRL = 0x00;                                          //Dead time configuration
        TIM1-> DTR = 0x00;                                            //Channel mode configuration
        TIM1-> CCMR1 = 0x68;                                          //Configured as an input channel
        TIM1-> CCMR2 = 0x68;
        TIM1-> CCMR3 = 0x68;
        TIM1-> CCMR4 = 0x68;

        TIM1-> SMCR = 0x77;                                           //ETRF input

        TIM1-> ARRH = 0x08;                                           // Automatic reloading of registers, counters 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;                      // The main output is enabled, which is equivalent to the 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;
        }
}
```

21.8.7 Input capture mode to measure the pulse period (capture rising edge to rising edge or falling edge Along to the falling edge)

C Language code

```
// The test operating frequency 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;                                  output PWM //IO
        PWMA_PS = 0x00;                                  //00:PWM at P1

                    Need to be turned off Correspond to And equipped with input */
                of /* Configured to TRGI
        pin TIM1-> PSCRH = 0x00;                         // Prescaler register
        TIM1-> PSCRL = 0x00;                             // Dead time configuration
        TIM1-> DTR = 0x00;                               // Channel mode configuration

        TIM1-> CCMR1 = 0x68;                             // Configured as an input channel
        TIM1-> CCMR2 = 0x01;
        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
                                                         // Enable counter
        TIM1-> CR1 |= 0x01;
        EA = 1;
        while (1);
}

/* Channel input, capture data through TIM1-> CCR2H / TIM1-> CCR2L read */
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;
        }
}
```

Input capture mode to measure pulse high-level width (capture rising edge to falli

C Language code

```
//The test operating frequency 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 *)0xfecc)
#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 capture TI1 Rising edge ,CC2 capture TI1 )Falling edge
        PWMA_CCER1 = 0x00;
        PWMA_CCMR1 = 0x01;                          //CC1 Is the input mode, And mapped to TI1FP1 Up
        PWMA_CCMR2 = 0x02;                          //CC2 to input mode, And mapped to TI1FP2 up
        PWMA_CCER1 = 0x11;                          //Enable CC1/CC2 Capture function on
        PWMA_CCER1 |= 0x00;                         //Set the capture polarity to The rising edge of CC1
        PWMA_CCER1 |= 0x20;                         //Set the capture polarity to The falling edge of
        PWMA_CR1 = 0x01;

        PWMA_IER = 0x04;                            //Enable CC2 Capture interrupt
        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
        }
}
```

Input capture mode to measure the low-level width of the pulse (capture the fallin

C Language code

```
// The test operating frequency 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 *)0xfecc)
#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 capture TI1 Rising edge,CC2 capture TI1 )Falling edge
        PWMA_CCER1 = 0x00;
        PWMA_CCMR1 = 0x01;                                //CC1 Is the input mode,And mapped TI1FP1 Up
        PWMA_CCMR2 = 0x02;                                //CC2 to input mode,And mapped to TI1FP2 up
        PWMA_CCER1 = 0x11;                                //Enable CC1/CC2 Capture function on
        PWMA_CCER1 |= 0x00;                               //Set the capture polarity to The rising edge of CC1
        PWMA_CCER1 |= 0x20;                               //Set the capture polarity to The falling edge of
        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
        }
}
```

21.8.10 Input capture mode simultaneously measures pulse period and duty cycle

Note: Only PWM1P、PWM2P、PWM5、PWM6 Only on these ports can the cycle and duty cycle be measured at the same time

C Language code

```
// The test operating frequency 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 *)0xfecc)
#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 Rising edge ,CC2 capture TI1 ) Falling edge
                                                //CC1 Capture cycle width ,CC2 Capture high-level width
        PWMA_CCER1 = 0x00;
        PWMA_CCMR1 = 0x01;                      //CC1 Is the input mode ,And mapped TI1FP1 Up
        PWMA_CCMR2 = 0x02;                      //CC2 to input mode ,And mapped to TI1FP2 up
        PWMA_CCER1 = 0x11;                      //Enable CC1/CC2 Capture function on
        PWMA_CCER1 |= 0x00;                     //Set the capture polarity to The rising edge of CC1
        PWMA_CCER1 |= 0x20;                     //Set the capture polarity to The falling edge of
        PWMA_SMCR = 0x54;                       //TS=TI1FP1,SMS=TI1 Rising edge reset mode
        PWMA_CR1 = 0x01;

        PWMA_IER = 0x06;                        //Enable CC1/CC2 Capture interrupt
        EA = 1;

        while (1);
}

void PWMA_ISR() interrupt 26
{
        unsigned int cnt;

        if (PWMA_SR1 & 0x02)
        {
                PWMA_SR1 &= ~0x02;

                cnt = PWMA_CCR1;                //CC1 Capture cycle width
        }
        if (PWMA_SR1 & 0x04)
        {
                PWMA_SR1 &= ~0x04;

                cnt = PWMA_CCR2;                //CC2 Capture duty cycle (high level width)
        }
}
```

21.8.11 With dead zone control PWM Complementary output

Language code C

```
// The test operating frequency 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;

        PWMA_ENO = 0xFF;                                  output PWM //IO
        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

                                                //Channel mode configuration
        TIM1-> CCMR1 = 0x68;
        TIM1-> CCMR2 = 0x68;
        TIM1-> CCMR3 = 0x68;
        TIM1-> CCMR4 = 0x68;

        TIM1-> ARRH = 0x08;                     // Automatic reloading of registers, counters 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;                      // The main output is enabled, which is equivalent to the 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;
        }
}
```

21.8.12PWM The port does external interrupts (falling edge interrupts or rising edge interrupts)

Language code C

```
//The test operating frequency 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 *)0xfecc)

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;

                                                        //Rising edge falling edge ) //(capture PWM1P
        PWMA_CCER1 = 0x00;
        PWMA_CCMR1 = 0x01;                              //CC1 Is the input mode, And mapped to on TI1FP1
        PWMA_CCER1 = 0x01;                              //Enable CC1 Capture function on
        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
        PWMA_CR1 = 0x01;
        PWMA_IER = 0x02;
        EA = 1;

        while (1);
}

void PWMA_ISR() interrupt 26
{
        if(PWMA_SR1 & 0X02)
        {
                P37 = ~P37;
                PWMA_SR1 &=~0X02;
        }
}
```

21.8.13 Output waveforms of any period and any duty cycle

C Language code

```
//The test operating frequency 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;                              //Must be cleared before write CCMRx CCERx Close the channel
        PWMA_CCMR1 = 0x60;                              //Set up CC1 to output mode PWMA
        PWMA_CCER1 = 0x01;                              //Enable CC1 channel
        PWMA_CCR1 = 100;                                //Set the duty cycle time
        PWMA_ARR = 500;                                 //Set cycle time
        PWMA_ENO = 0x01;                                //Enable PWM1P Port output
        PWMA_BKR = 0x80;                                //Enable main output
        PWMA_CR1 = 0x01;                                //Start timing

        while (1);
}
```

21.8.14 use PWM of CEN Start PWMA Timer, triggered in real time ADC

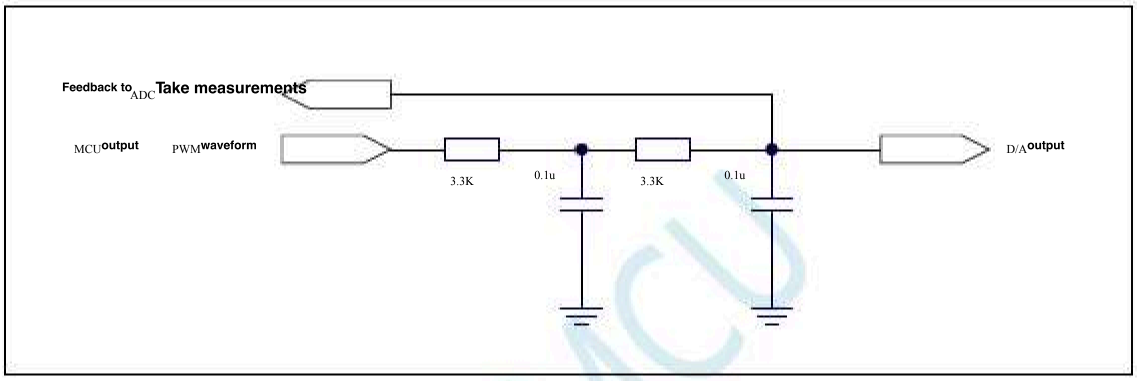
Language code C

```
//The test operating frequency 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 *)0xfecc)
#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;        //choose P1.0 for ADC Input channel
        delay();                                      //wait ADC Stable power supply
        EADC = 1;

        PWMA_CR2 = 0x10;                              //CEN The signal is TRGO, Can be used to trigger ADC
        PWMA_ARR = 5000;
        PWMA_IER = 0x01;
        PWMA_CR1 = 0x01;                              //Set up CEN Start PWMA Timer, triggered in real time ADC
        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;
        }
}
```

21.8.15 use PWM Bit implementation 16 DAC Reference circuit diagram of

STC12H Advanced series of microcontrollers PWM Timer can output bits of 16 PWM The waveform can be generated after two stages of low-pass filte 16 Bit of DAC Signal, by adjusting PWM The high-level duty cycle of the waveform can be achieved DAC Signal change. The application circuit diagram is shown in Show that the output of DAC The signal can be input to MCU of ADC Perform feedback measurements.



21.8.16 use PWM Achieve complementarity SPWM

Timer advanced PWM PWM1P/PWM1N , PWM2P/PWM2N , PWM4P/PWM4NPWM3P/PWM3N Each channel Can independently realize Output, or pairwise complementary symmetrical output. Demo use Produce complementary SPWM, PWM1P , PWM1N the master clock to select PWM Clock selection PWM cycle 2400, Dead zone 12 A clock (0.5us), For sine wave meters 200 point , the output sine wave frequency =24000000/2400/200=50 HZ, PWM=24MHZ

This program is just a SPWM The demonstration program, the user can modify it through the above calculation method PWM Some of the number of points of the period ar amplitude. The output frequency of this program is fixed. If the frequency conversion is required, the user is requested to design the frequen

Language code C

```
//The test operating frequency is 24MHz

#include "reg51. h"
#include "intrins. h"
#define         MAIN_Fosc               24000000L               //Define the master 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                                         N:P1.3//P:P1.2/P5.4
#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, 899, 865, 830,
                                  969,     934,
         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 Look-up table index

/********************          Main function*************************/
void main(void)
{
        P0M1 = 0;    P0M0 = 0;    //Set to prevail two-way port
        P1M1 = 0;    P1M0 = 0;    //Set to prevail two-way port
        P2M1 = 0;    P2M0 = 0;    //Set to prevail two-way port
        P3M1 = 0;    P3M0 = 0;    //Set to prevail two-way port
        P4M1 = 0;    P4M0 = 0;    //Set to prevail two-way port
        P5M1 = 0;    P5M0 = 0;    //Set to prevail two-way port
        P6M1 = 0;    P6M0 = 0;    //Set to prevail two-way port
        P7M1 = 0;    P7M0 = 0;    //Set to prevail two-way port

        PWMA_Duty = 1220;

        P_SW2 |= 0x80;

        PWMA_CCER1 = 0x00;                                 //write CCMRx Must be cleared before 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 cycle time
        PWMA_ARRL = 0x60;

        PWMA_CCR1H = (u8)(PWMA_Duty >> 8);                 //Set the 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 bit
        PWMA_PS |= PWMA_3;                                 //choose PWMA_3 Channel
//      PWMA_PS |= PWMB_3;                                 //choose PWMB_3 Channel Channel
//      PWMA_PS |= PWM3_3;                                 //choose PWM3_3 Channel
//      PWMA_PS |= PWM4_3;                                 //choose PWM4_3 Channel

        PWMA_BKR = 0x80;                                   //Enable main output
        PWMA_IER = 0x01;                                   //Enable interrupt
        PWMA_CR1 |= 0x01;                                  //Start timing
        P_SW2 &= 0x7f;

        EA = 1;                                                              //Open total interrupt

        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);                                   //When setting the duty cyclebetween
        PWMA_CCR1L = (u8)(PWMA_Duty);
        }
        PWMA_SR1 = 0;
        P_SW2 &= 0x7f;
}
```

22 Enhanced dual data pointer

Two sets of 16-bit data pointers are integrated into the STC12H series of microcontrollers. Through program control, the automatic increment or decrement function of the data pointer and the automatic switching function of the two sets of data pointers can be realized.

22.1 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)	82H									0000,0000
DPH	Data pointer (high byte)	83H									0000,0000
DPL1	The second set of data pointers (low bytes)	E4H									0000,0000
DPH1	The second set of data pointers (high bytes)	E5H									0000,0000
DPS	Pointer selector DPTR	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR Timing control register	AEH									0000,0000

22.1.1 The first group 16 Bit data pointer register (DPTR0)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL is low 8 Bit data (low byte)

DPH is high 8 Bit data (high bytes)

DPL and DPH The combination is the first group 16 Bit data pointer register DPTR0

22.1.2 2 Group 1 16 Bit data pointer register (DPTR1)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1 is low 8 Bit data (low byte)

DPH1 is high 8 Bit data (high bytes)

DPL1 and DPH1 Combined into a second group 16 Bit data pointer register DPTR1

22.1.3 Data pointer control register (DPS)

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

ID1: DPTR1 Automatic increment method

Control automatic increment DPTR1

0 :

1 : Automatic decrement

ID0 DPTR1

: control DPTR0 Automatic increment method

0 : DPTR0 Auto increment

1 : DPTR0 auto decrement

TSL: Automatic switching control (automatic Reverse) pair DPTR0/DPTR1 SEL

0 : Turn off the automatic switching function

1 : Enable the automatic switching function

when TSL After the position is set 1, Whenever the relevant instructions are SEL The bit is reversed.

with TSL executed, the system will automatically include the relevant

instructions as follows: MOV DPTR,#data16

INC DPTR

MOVC A,@A+DPTR

MOVX A,@DPTR

MOVX @DPTR,A

AU1/AU0: Enable / offClose ID1/ID0 use DPTR1/DPTR0 The control bit is automatically incremented/Decrement control

0 automatic increment/Decrement function : Enable

1 automatic increment/Decrement function

Note: In write protected mode , AU0 Bits cannot be enabled directly separately, if enabled separately and AU1 Bit will also

Is automatically enabled AU0, triggered Bit, then AU1

by a separate enable register DPS The protection mechanism (reference, no effect. If you need to enable it separately you must use TA Description of the re ° In addition, only after executing the following 3 instructions

DPTR0/DPTR1 will be automatically

incremented/decremented. The 3 relevant instructions are as follows: MOVC

A,@A+DPTR MOVX A,@DPTR

MOVX @DPTR,A

SEL: Choose DPTR0/DPTR1 As the current goal DPTR

0: Choose DPTR0 As a DPTR

1: Choose DPTR1 goal as a DPTR

SEL Choose a goal DPTR Valid for the

target MOV following instructions: DPTR,#data16

INC	DPTR
MOVC	A,@A+DPTR
MOVX	A,@DPTR
MOVX	@DPTR,A
JMP	@A+DPTR

22.1.4 Data pointer control register (TA)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

TA The register is correct DPS In the register AU1 and AU0 Write protection. Because the program cannot be correct DPS AU0 Make a separate AU1 And Write, so when it needs to be enabled separately AU1 or AU0 When, you must use TA The register is triggered. TA registers are write-only registers.

, when it needs to be AU1 or AU0 When enabling separately, you must follow the steps below:

```
CLR     EA          ;Shutdown interrupt (required)
MOV     TA,#0AAH    ;Write trigger command sequence 1
                    ;There can be no other instructions here
MOV     TA,#55H     ;Write trigger command sequence 2
                    ;There can be no other instructions here, write protection is temporarily turned off, you can send it to DPS Write any value in
MOV     DPS,#xxH    ;DSP Write protection status again
SETB    EA          ; Turn on the interrupt (if necessary)
```

22.2 Sample program

22.2.1 Sample code 1

Copy the 4 bytes of data from the program space 1000Hto 1003H in reverse to the 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

```
;The test operating frequency is 11.0592MHz

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

                ORG             0000H
                LJMP            MAIN

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

                MOV             DPS,#00100000B          TSL,And choose;Enable DPTR0
                MOV             DPTR,#1000H             ;will 1000H DPTR0 write After selection DPTR1 for DPTR
                MOV             DPTR,#0103H             0103H;will DPTR1 Writing is in
                MOV             DPS,#10111000B          ;Set up DPTR1 decreasing mode,DPTR0 , Enable the current for the
                                                        and AU1,And choose DPTR0 incremental mode DPTR
                MOV             R7,#4                   ;;AU0 Set the number of data copies
COPY_NEXT:
                CLR             A                       ;
                MOVC            A,@A+DPTR               DPTR0;from The program space referred to reads data,
                                                        ;After completion 1 Automatically DPTR1 Set to DPTR
                MOVX            @DPTR,A                 ;will ACC add and write the DPTR1 data to Refers to XDATA ,
                                                        ;After completion DPTR0 1 Automatically reduce and merge DPTR0 Set to DPTR
                DJNZ            R7,COPY_NEXT                        ;

                SJMP            $

                END
```

22.2.2 Sample code 2

Send the data in 0100H ~ 0103H of the extended

RAM to the P0 port assembly code in turn

```
;The test operating frequency is 11.0592MHz

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

                CLR             EA                                  ;Turn off interrupt
                MOV             TA,#0AAH                            ;write DPS Write protection trigger command
                MOV             TA,#55H                             ;write DPS Write protection trigger command
                MOV             DPS,#00001000B                      ;DPTR0 Incrementally enabled separately, choose DPTR0
                SETB            EA                                  ;Turn on interrupt
                MOV             DPTR,#0100H                         ;will 0100H write DPTR0 In
                MOVX            A,@DPTR                             ;from DPTR0 Refers to XRAM After reading the data DPTR0 Automatic addition
                MOV             P0,A                                ;Data output to P0 mouth
                MOVX            A,@DPTR                             ;from DPTR0 Refers to XRAM After reading the data DPTR0 Automatic addition
                MOV             P0,A                                ;Data output to P0 mouth
                MOVX            A,@DPTR                             ;from DPTR0 Refers to XRAM After reading the data DPTR0 Automatic addition
                MOV             P0,A                                ;Data output to P0 mouth
                MOVX            A,@DPTR                             ;from DPTR0 Refers to XRAM After reading the data DPTR0 Automatic addition
                MOV             P0,A                                ;Data output to P0 mouth
```

```
        SJMP        $

        END
```

23 MDU16 hardware 16 Bit multiplication and division method

Some models of STC12H series microcontrollers have integrated MDU16/16-bit hardware multiplication and division device.

Support the following data operations：

Data normalization (It takes 3 to 20 clocks to calculate the time)

Logical left shift (It takes 3 to 18 clocks to calculate the time)

Logical right shift (It takes 3 to 18 clocks to calculate the time)

16 bits multiplied by 16 bits (It takes 10 clocks to calculate the time)

16 bits divided by 16 bits (It takes 9 clocks to calculate the time)

32 bits divided by 16 bits (It takes 17 clocks to calculate the time)

All operations are based on unsigned shaping data types.

23.1 Related special function registers

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 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 Data	FCF4H	MD5[7:0]								0000,0000
MD4	MDU Register Mode Control	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU Register Operation	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU Control Register	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000

23.1.1 Operand 1 data register (MD0 ~ MD3)

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]							

23.1.2 Operand 2 data register (MD4 ~ MD5)

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

32 Divide by bits 16 Bit division :

Divisible number : {MD3,MD2,MD1,MD0}

Divisor : {MD5,MD4}

Quotient : {MD3,MD2,MD1,MD0}

remainder : {MD5,MD4}

16 Divide by bits 16 Bit division :

Divisible number : {MD1,MD0}

Divisor : {MD5,MD4}

Quotient : {MD1,MD0}

remainder : {MD5,MD4}

16 Multiply by 16 Bit multiplication :

Multiplier : {MD1,MD0}

multiplier : {MD5,MD4}

product : {MD3,MD2,MD1,MD0}

32 Bit logic shift to the left/Logical right shift

32 Operand : {MD3,MD2,MD1,MD0}

Bit data normalization : Operand : {MD3,MD2,MD1,MD0}

23.1.3 MDU Mode control register (ARCON), the number of clocks required for the operation

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]	pattern	Number of clocks	Operation instructions
1	Logical right shift	~ 18	will {MD3,MD2,MD1,MD0} Shift the data to the SC[4:0] bit , right MD3 The high replenishment 0
2	Logical left shift	3 ~ 18	will {MD3,MD2,MD1,MD0} The data in the shift SC[4:0] bit , to the left MD0 The low complement 0
3	Data normalization	3 ~ 20	correct {MD3,MD2,MD1,MD0} The data in it is logically shifted to the left, and the data High-level 0 Remove all of them so that the highest position is 1 , The number of digits of the log Is recorded in SC[4:0] In
4	16 Bit×bit 16	10	{MD1,MD0}× {MD5,MD4} = {MD3,MD2,MD1,MD0}
5	16 Bit 位 bit	9	{MD1,MD0}÷ {MD5,MD4} = {MD1,MD0}··· {MD5,MD4}
6	32 Bit 位 bit	17	{MD3,MD2,MD1,MD0}÷ {MD5,MD4} = {MD3,MD2,MD1,MD0}··· {MD5,MD4}
other	invalid		

SC[4:0] : Number of digits of data movement

when MDU Used to set the left shift/When the number of digits shifted to the right is in movement mode ,

when MDU SC Is the actual number of digits moved by the data after the data is normalized

When it is a data normalization mode , SC

23.1.4 MDU Operation control register (OPCON)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
OPCON	FCF7H	-	MDOV	-	-	-	-	RST	ENOP

MDU MDOV : Overflow flag (read-only flag)

In the following cases , 0

MDOV Will be automatically set by the hardware The product of : 1

multiplication is greater than When the divisor is 0FFFFH2) Or write the software writes ARCON When the hardware will automatically clear OPCON MDOV

1 (

MDU RST : Software reset Multiply and divide unit. Write trigger software reset 1 The hardware is automatically cleared to zero after

Note: Software reset MDU When multiplying and dividing units , the reset is complete. The value of the register will be cleared.

ENOP : MDU ARCON 1 The module starts to calculate After the calculation is completed, the hardware will automatic ENOP Clear to zero

The software can be right MDU ENOP Set back 1, Circular query ENOP, when ENOP change from to to indicate that the calculation is complete. 1 0

The module is enabled. Write trigger MDU

23.2 Sample program

C Language code

```
// The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
#define MD3U32      (*(unsigned long volatile xdata *)0xfcf0)
#define MD3U16      (*(unsigned int volatile xdata *)0xfcf0)
#define MD1U16      (*(unsigned int volatile xdata *)0xfcf2)
#define MD5U16      (*(unsigned int volatile xdata *)0xfcf4)
#define MD3         (*(unsigned char volatile xdata *)0xfcf0)
#define MD2         (*(unsigned char volatile xdata *)0xfcf1)
#define MD1         (*(unsigned char volatile xdata *)0xfcf2)
#define MD0         (*(unsigned char volatile xdata *)0xfcf3)
#define MD5         (*(unsigned char volatile xdata *)0xfcf4)
#define MD4         (*(unsigned char volatile xdata *)0xfcf5)
#define ARCON       (*(unsigned char volatile xdata *)0xfcf6)
#define OPCON       (*(unsigned char volatile xdata *)0xfcf7)
sfr P_SW2 = 0xBA;

//////////////////////////////////////////////////////////////////////
//16 Bit multiplication 16 bit
//////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned int dat1, dat2;
P_SW2 |= 0x80;                    // Access to extended xsfr registers
MD1U16 = dat1;                    // dat1 User given
MD5U16 = dat2;                    // dat2 user given
ARCON = 4 << 5;                   // 16 position *16 , Bit multiplication mode
OPCON = 1;                        // Start the calculation
while((OPCON & 1) ! = 0);         // Wait for the calculation to complete
res = MD3U32;                     //32 Bit result
//////////////////////////////////////////////////////////////////////

//32 Divide by bits 16 bit
//////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned int dat2;
P_SW2 |= 0x80;                    // Access to extended xsfr registers
MD3U32 = dat1;                    // dat1 User given
MD5U16 = data2;                   // dat2 user given
ARCON = 6 << 5;                   // 32 position /16 , Bit division mode
OPCON = 1;                        // Start the calculation
while((OPCON & 1) ! = 0);         // Wait for the calculation to complete
res = MD3U32;                     //32 Position quotient, The remainder MD5U16 of the digits is 16
```

```
//////////////////////////////////////////////////////////////////////
// Move left or right
```

```
//////////////////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned char num;                              // Number of shifted digits, Given by the user
MD3U32 = dat1;                                  //dat1 User-given
ARCON = (2 << 5) + num;                         //32 bit shift mode
//ARCON = (1 << 5) + num;                       //32, bit shift mode, bit shift mode
OPCON = 1;                                      // Start the calculation
while((OPCON & 1) ! = 0);                       // Wait for the calculation to complete
res = MD3U32;                                   //32    Bit result
```

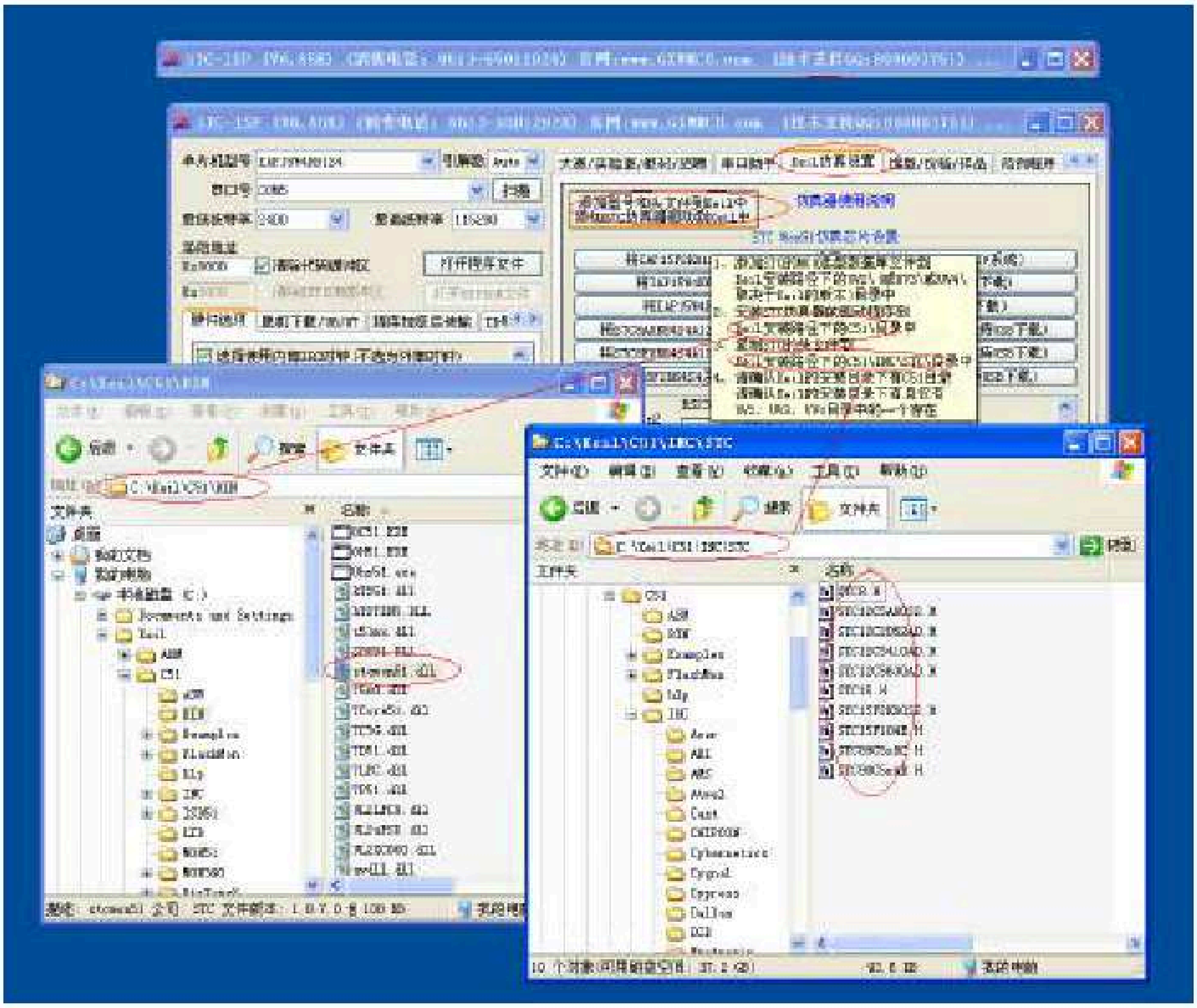
A Appendix Compiler (assembler)/Emulator usage guide

Q: Any old-fashioned 8051 What kind of compiler should be used for the MCU?/Assembler? A: STC compiler/Assemblers can be supported, and they are now popular to use Keil C51

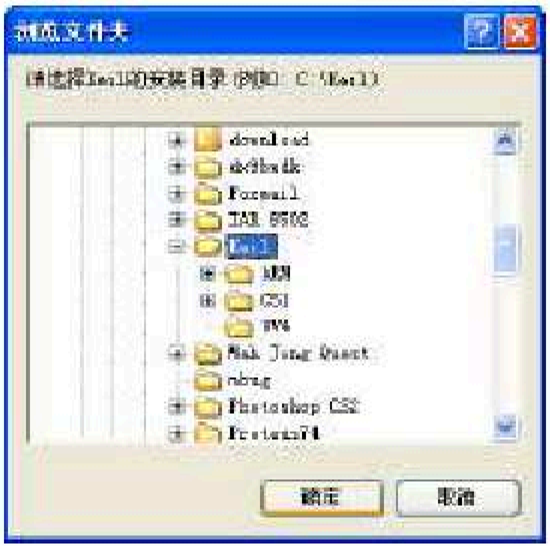
Q Keil How should header files be included in the environment
: After installing the driver and header files according to the steps shown below, select when creating a new project, STC The corresponding MCU model is in the source file

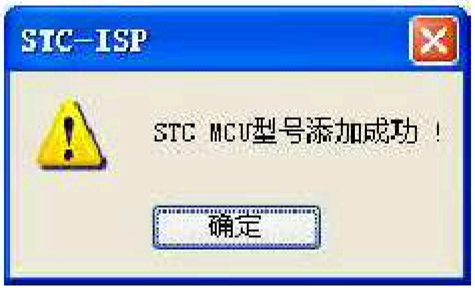
Direct use" #include <STC12H. h> "That is, the inclusion of the header file can be completed. If selected when building a new project
8052/87C52/87C54/87C58 or Philips Compilation, the header file contains P87C52/P87C54/P87C58 <reg51. h> That's it, but
STC The new special function register needs to be declared by the user.

1, installation Keil Version of the simulation driver



As shown in the figure above, first select the "Keil Simulation Settings" page, click "Add MCU model to Keil", and in the following directory se
window that appears, navigate to the installation directory of Keil (generally C:\Keil\"), click "OK", after the prompt letter shown on the right in the figure l
information, indicating that the installation was successful. The emulation driver that will be installed at the same time as adding the header f
The installation directory of the driver and header files is shown in the figure above.



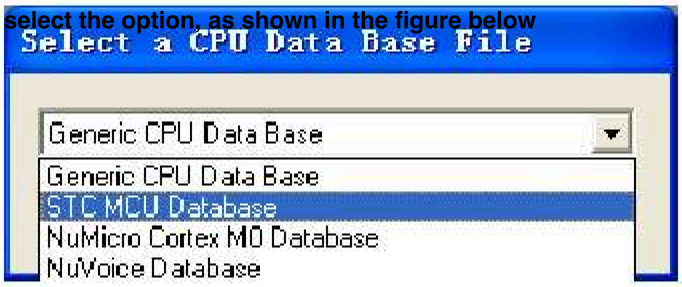


Create a project in

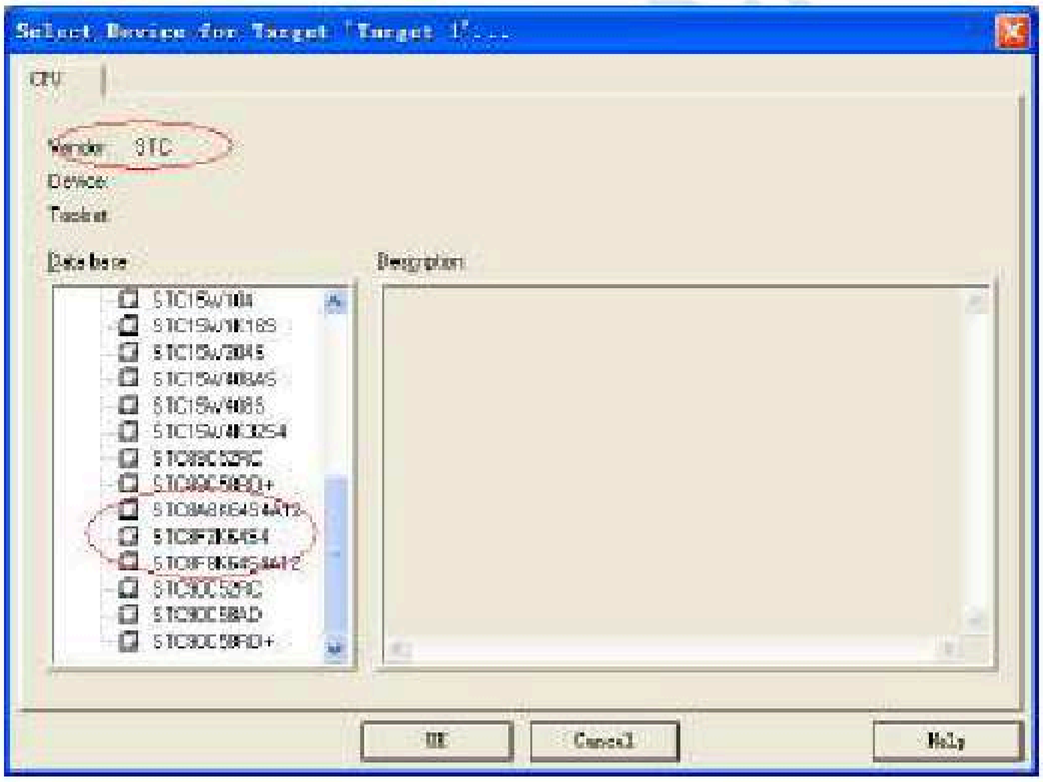
Keil 2, If the Keil When you select the chip model when creating a new project, "The choice will be " STC MCU Database

driver installation in the first step is successful, then

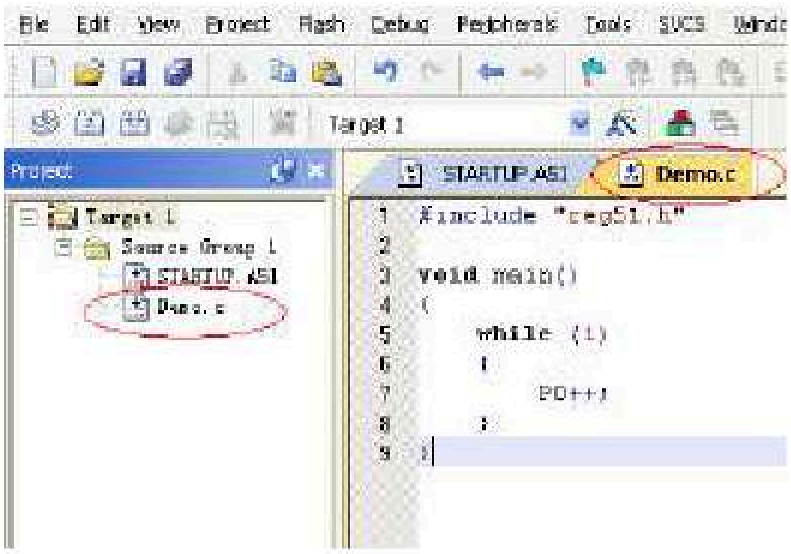
select the option, as shown in the figure below



Then select the response from the STC list Model, we choose here" "Of the model, click "OK" to complete the selection STC



Add the source code file to the project, as shown in the figure below :

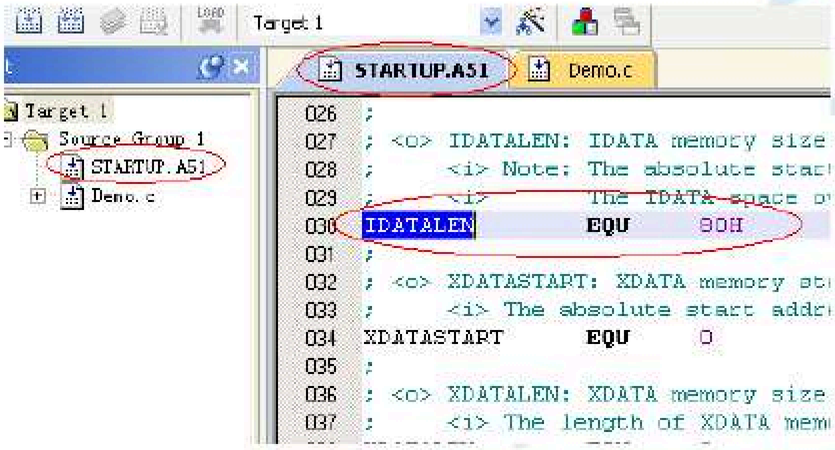


```
#include "reg51.h"

void main()
{
    while (1)
    {
        P0++;
    }
}
```

Save the project, if the compilation is correct, you can

set up the following project An additional note:

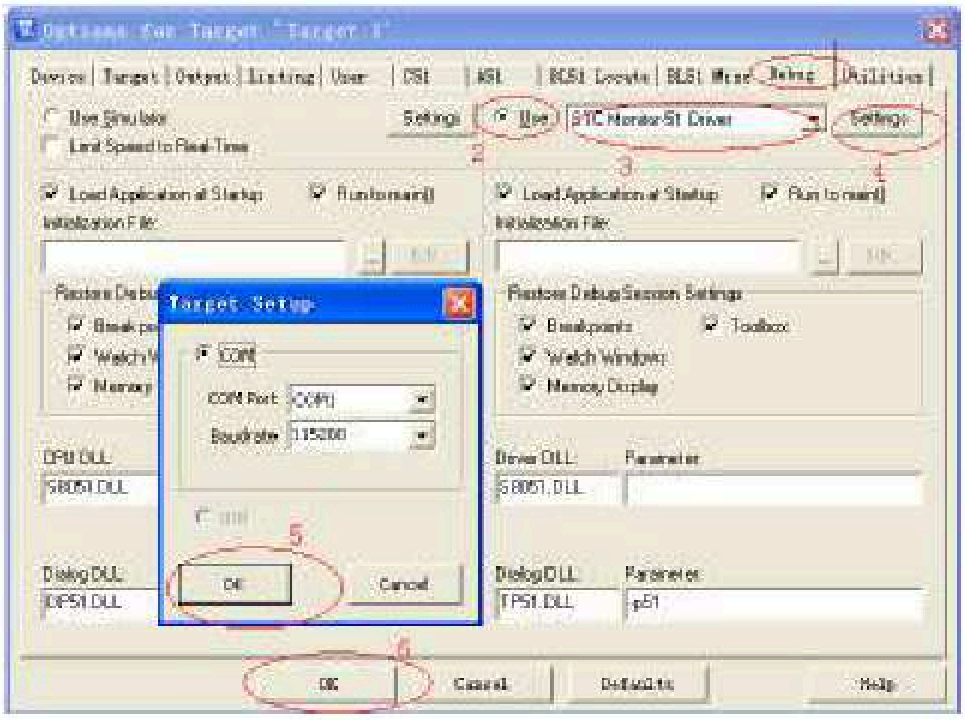
when it is created is C Language project, and there will be a startup file""When added to the project, there is a macro named "IDATALEN "The macro definition, when it is used to IDATA size in it. The default value is 128, That is, hexadecimal 80H, IDATA The same size. So when defined as IDATA 80H, then STARTUP. A51 define it, it also needs to be initialized to in the startup file. 0 The code inside will be IDATA 00-7F RAM 0 Initialized to; similarly, if it is defined as IDATA 0FFH, It will IDATA of 00-FF RAM 0 Initialized to.



```
;
; <o> IDATALEN: IDATA memory size
;     <i> Note: The absolute star
;     <i>       The IDATA space o
IDATALEN        EQU     80H
;
; <o> XDATASTART: XDATA memory st
;     <i> The absolute start addr
XDATASTART      EQU     0
;
; <o> XDATALEN: XDATA memory size
;     <i> The length of XDATA mem
```

Although the series of microcontrollers IDATA: The size is 256 Byte (00-7F of DATA and 80H-FFH of IDATA), but because of the RASTC8M Be sure not to write the last byte 17 Number and related test parameters, if the user needs to use this part of the data in the program, th Defined as IDATALEN 256。

3, Project settings, select STC Simulation driver

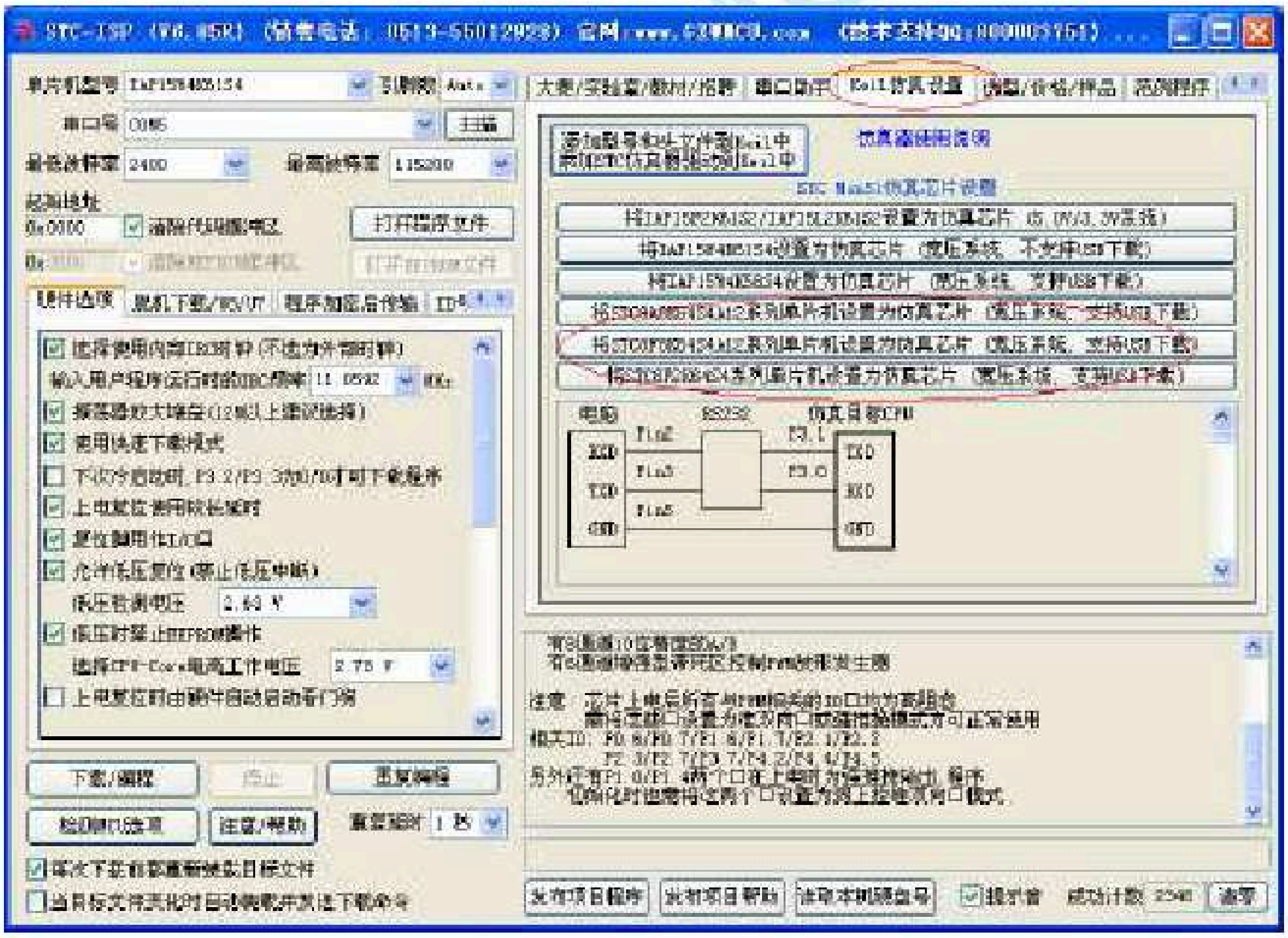


As shown in the figure above, first go to the project's settings page "Settings page," Debug step 1 select the hardware simulation on the right" Use ..."

Step 1, select "In the simulation driver drop-down list" STC Monitor-51 Driver "Item, then click " Settings "Button, go to the next

On the setting screen on the surface, set the port number and baud rate of the serial port, and the baud rate is generally selected. At this point, the setup is complete. 115200

4, Create an simulation chip



Prepare one Series or STC8F STC8A Series of chips, and connect to the computer's serial port through the download board, and then as

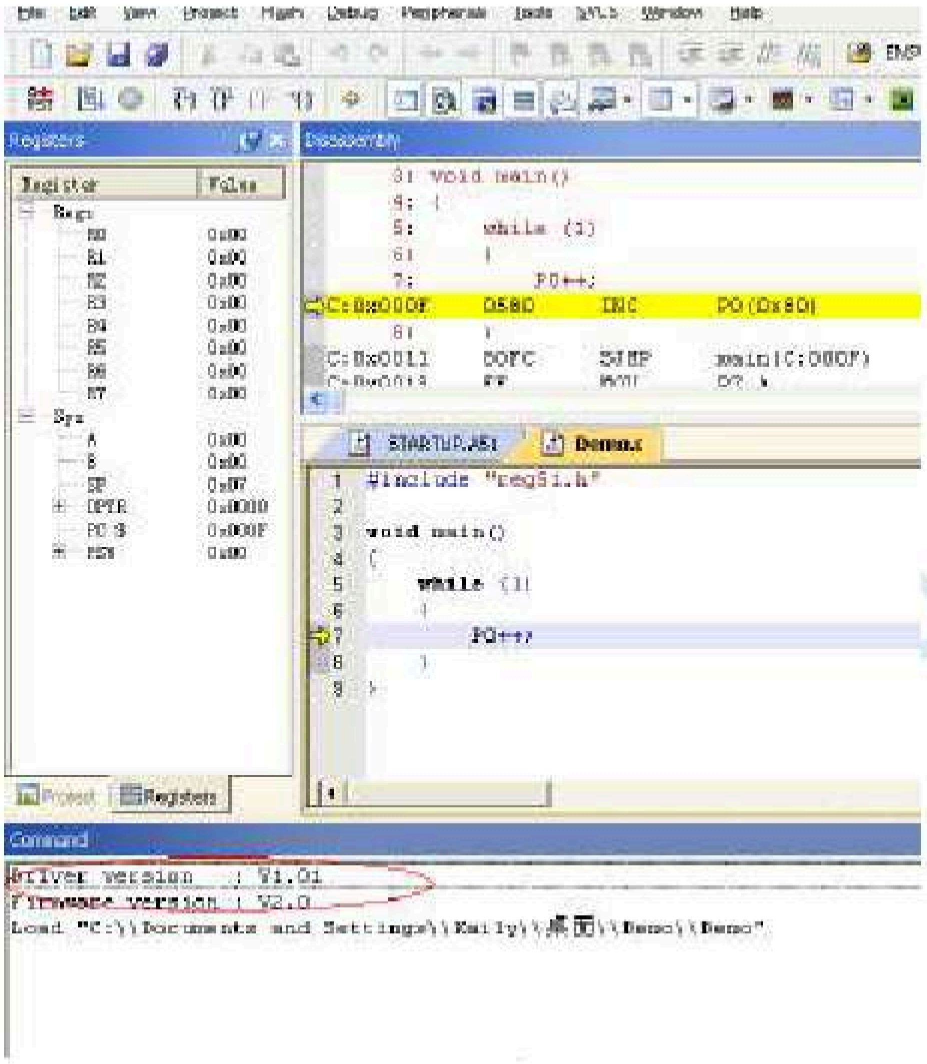
The correct chip model, and then go to " Keil the "Simulation settings" page, click the button of the corresponding model, when the program dow

The production is complete.

5 , Start simulation

Connect the completed simulation chip to the computer through the serial port. After compiling the project we created earlier to no errors, press Ctrl+F5 "Start debugging.

If the hardware connection is correct, it will enter a debugging interface similar to the following, and the current simulation driver version number and the current simulation monitoring code firmware version number will be displayed in the command output window . The maximum allowable number before the number of breakpoints is set (in theory, any one can be set, but setting too many breakpoints will affect the speed of debugging).



```
Regs
  R0    0x00
  R1    0x00
  R2    0x00
  R3    0x00
  R4    0x00
  R5    0x00
  R6    0x00
  R7    0x00
Sys
  A     0x00
  B     0x00
  SP    0x07
  DPTR  0x0000
  PC $  0x000F
  PSW   0x00
```

```
     3: void main()
     4: {
     5:     while (1)
     6:     {
     7:         P0++;
C:0x000F    0580     INC     P0(0x80)
     8:     }
C:0x0011    80FC     SJMP    main(C:000F)
```

```
#include "reg51.h"

void main()
{
    while (1)
    {
        P0++;
    }
}
```

```
Driver version    : V1.01
Firmware version  : V2.0
Load "C:\\Documents and Settings\\Emily\\桌面\\Demo\\Demo"
```

Simulation precautions :

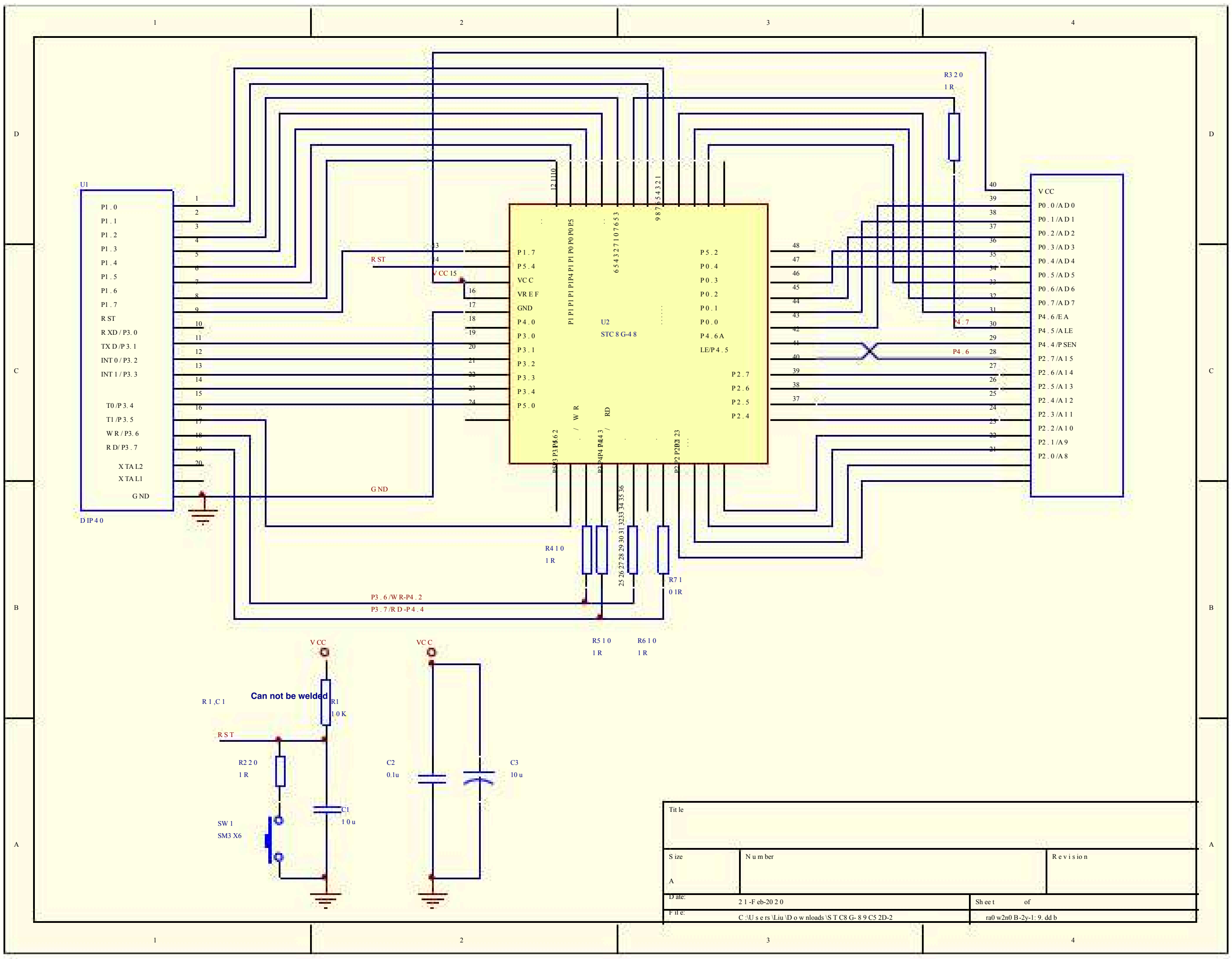
1 、Simulation monitoring program occupies P3.0/P3.1 Two ports, but does not occupy the serial port, the user can connect the serial port Switch to P3.6/P3.7 or P1.6/P1.7 Re-use

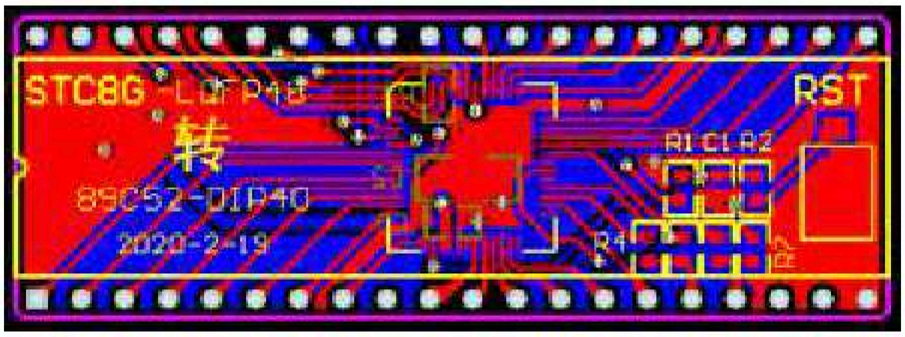
2 、Simulation monitoring program occupies internal expansion RAM(XDATA) The last 768 Bytes, the user cannot access this area XDATA enter Line write operation

B Appendix How to make the traditional 8051 Single-chip microcomputer learning board

The traditional 8051 single-chip microcomputer learning board does not have a simulation function. For the traditional 8051 single-chip microcompu
required . The physical picture of the conversion board is shown in the figure below. The pin arrangement after conversion is basically the same as that of
can realize the simulation function of the standard 8051 learning board.

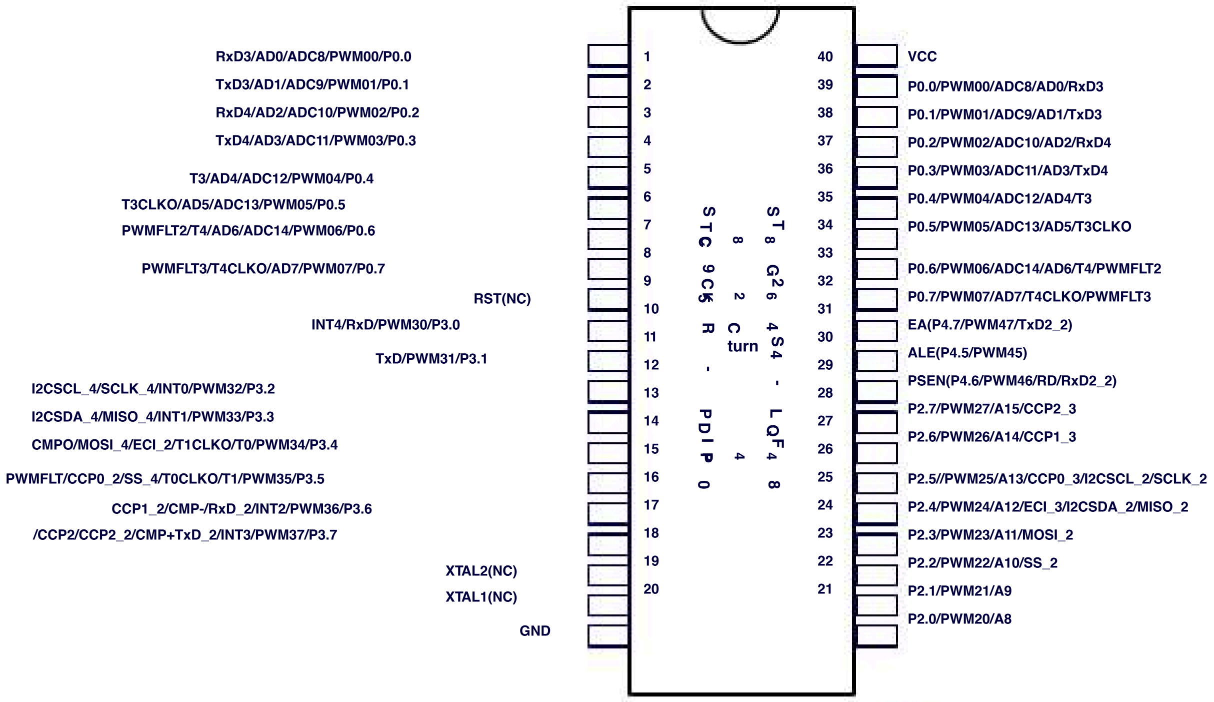
The following figure is the schematic diagram of the conversion board and PCB Board drawing





The conversion board can be used for STC8G series LQFP48 to STC89C52RC/STC89C58RD+ series simulation.

The picture below is a schematic diagram of the function of the conversion board


RxD3/AD0/ADC8/PWM00/P0.0
TxD3/AD1/ADC9/PWM01/P0.1
RxD4/AD2/ADC10/PWM02/P0.2
TxD4/AD3/ADC11/PWM03/P0.3
T3/AD4/ADC12/PWM04/P0.4
T3CLKO/AD5/ADC13/PWM05/P0.5
PWMFLT2/T4/AD6/ADC14/PWM06/P0.6
PWMFLT3/T4CLKO/AD7/PWM07/P0.7
RST(NC)
INT4/RxD/PWM30/P3.0
TxD/PWM31/P3.1
I2CSCL_4/SCLK_4/INT0/PWM32/P3.2
I2CSDA_4/MISO_4/INT1/PWM33/P3.3
CMPO/MOSI_4/ECI_2/T1CLKO/T0/PWM34/P3.4
PWMFLT/CCP0_2/SS_4/T0CLKO/T1/PWM35/P3.5
CCP1_2/CMP-/RxD_2/INT2/PWM36/P3.6
/CCP2/CCP2_2/CMP+TxD_2/INT3/PWM37/P3.7
XTAL2(NC)
XTAL1(NC)
GND
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
STC8G2K64S4-LQFP48
turn
STC89C5 R-PDIP40
VCC
P0.0/PWM00/ADC8/AD0/RxD3
P0.1/PWM01/ADC9/AD1/TxD3
P0.2/PWM02/ADC10/AD2/RxD4
P0.3/PWM03/ADC11/AD3/TxD4
P0.4/PWM04/ADC12/AD4/T3
P0.5/PWM05/ADC13/AD5/T3CLKO
P0.6/PWM06/ADC14/AD6/T4/PWMFLT2
P0.7/PWM07/AD7/T4CLKO/PWMFLT3
EA(P4.7/PWM47/TxD2_2)
ALE(P4.5/PWM45)
PSEN(P4.6/PWM46/RD/RxD2_2)
P2.7/PWM27/A15/CCP2_3
P2.6/PWM26/A14/CCP1_3
P2.5//PWM25/A13/CCP0_3/I2CSCL_2/SCLK_2
P2.4/PWM24/A12/ECI_3/I2CSDA_2/MISO_2
P2.3/PWM23/A11/MOSI_2
P2.2/PWM22/A10/SS_2
P2.1/PWM21/A9
P2.0/PWM20/A8

attention :

Due to the built-in high-precision R/C clock, no external crystal oscillator is required. XTAL1 and XTAL2 are empty . WR and RD are (WR/P4.2 and RD/P4.4) instead of the 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 the user needs to use this conversion board to access the external bus, P3.6 and P3.7 need to be set to the high impedance input mode, so that P4.2 and P4.4 normally output bus read and write signals; if you do not need to access the external bus, you need to P4.2 and P4.4 Set the high impedance input mode, 3.6 and P3.7 is ordinary I/O.)

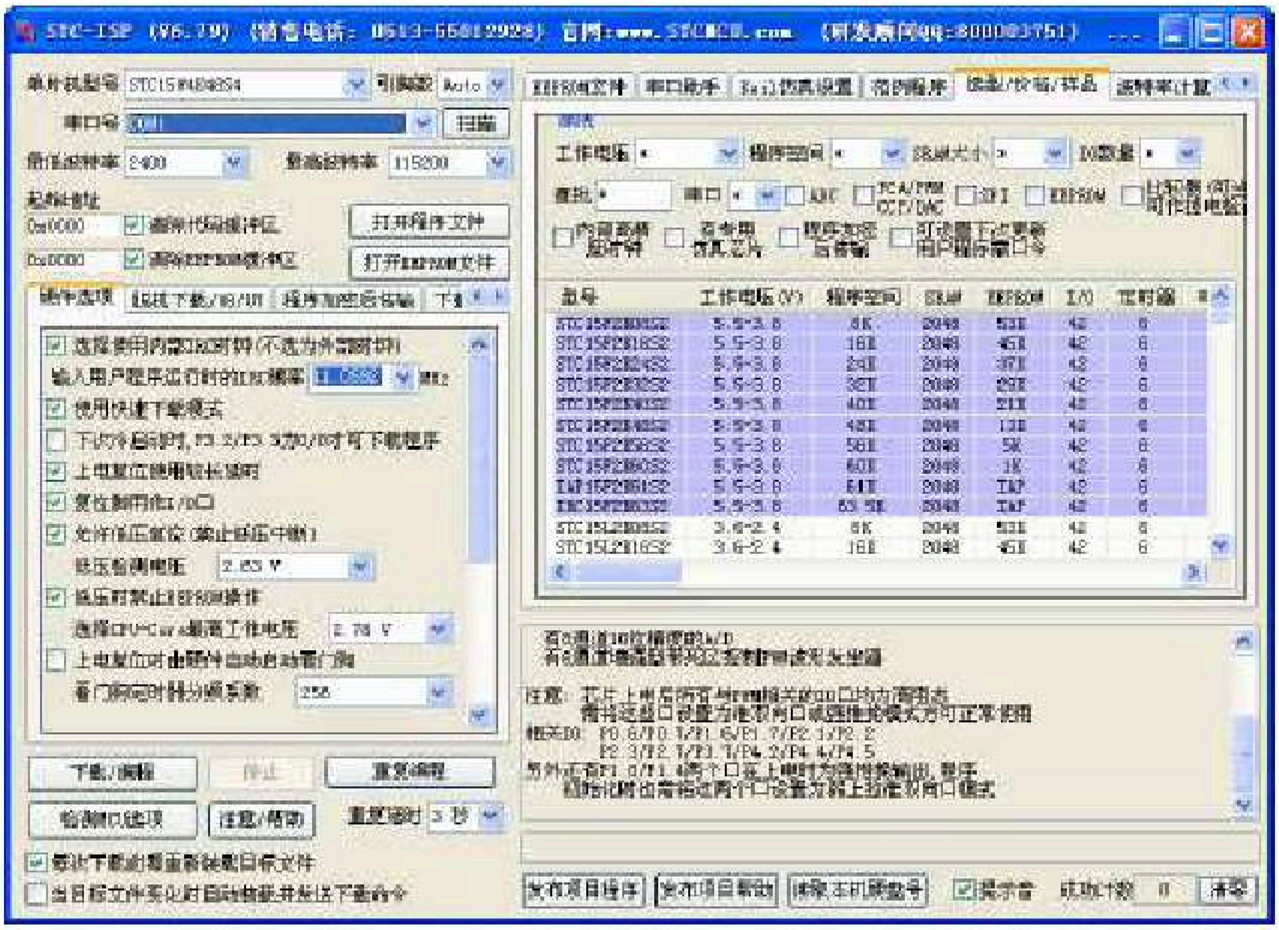
Since the STC8G series MCU is a low-level reset, it is not compatible with the high-level reset of the traditional 8051, so the RST pin is floating, and the reset button on the conversion board is replaced by a reset circuit.

C Appendix STC-USB Driver installation instructions

Windows XP installation method

Open V6.79 Version (or updated version) of STC-ISP Download the software, the downloaded software will automatically copy the driver fil

directory



STC-ISP (V6.79) (销售电话: 0513-55012928) 官网: www.STCMCU.com (研发顾问QQ:800003751)

单片机型号 STC15W4K48S4 引脚数 Auto

串口号 COM1 扫描

最低波特率 2400 最高波特率 115200

起始地址

0x0000 ☑ 清除代码缓冲区 打开程序文件

0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | 下载

☑ 选择使用内部IRC时钟(不选为外部时钟)

输入用户程序运行时的IRC频率 11.0592 MHz

☑ 使用快速下载模式

☐ 下次冷启动时, P3.2/P3.3为0/0时才可下载程序

☑ 上电复位使用较长延时

☑ 复位脚用作I/O口

☑ 允许低压复位(禁止低压中断)

低压检测电压 2.83 V

☑ 低压时禁止EEPROM操作

选择CPU-Core最高工作电压 2.78 V

☐ 上电复位时由硬件自动启动看门狗

看门狗定时器分频系数 256

下载/编程 | 停止 | 重复编程

检测MCU选项 | 注意/帮助 | 重复延时 3 秒

☑ 每次下载前都重新装载目标文件

☐ 当目标文件变化时自动装载并发送下载命令

EEPROM文件 | 串口助手 | Keil仿真设置 | 范例程序 | 选型/价格/样品 | 波特率计算器

筛选

工作电压 * 程序空间 * SRAM大小 * IO数量 *

管脚 * 串口 * ☐ ADC ☐ PCA/PWM/CCP/DAC ☐ SPI ☐ EEPROM ☐ 比较器(可当ADC使用,可作掉电检测)

☐ 内部高精准时钟 ☐ 有专用复位芯片 ☐ 程序加密后传输 ☐ 可设置下次更新用户程序需口令

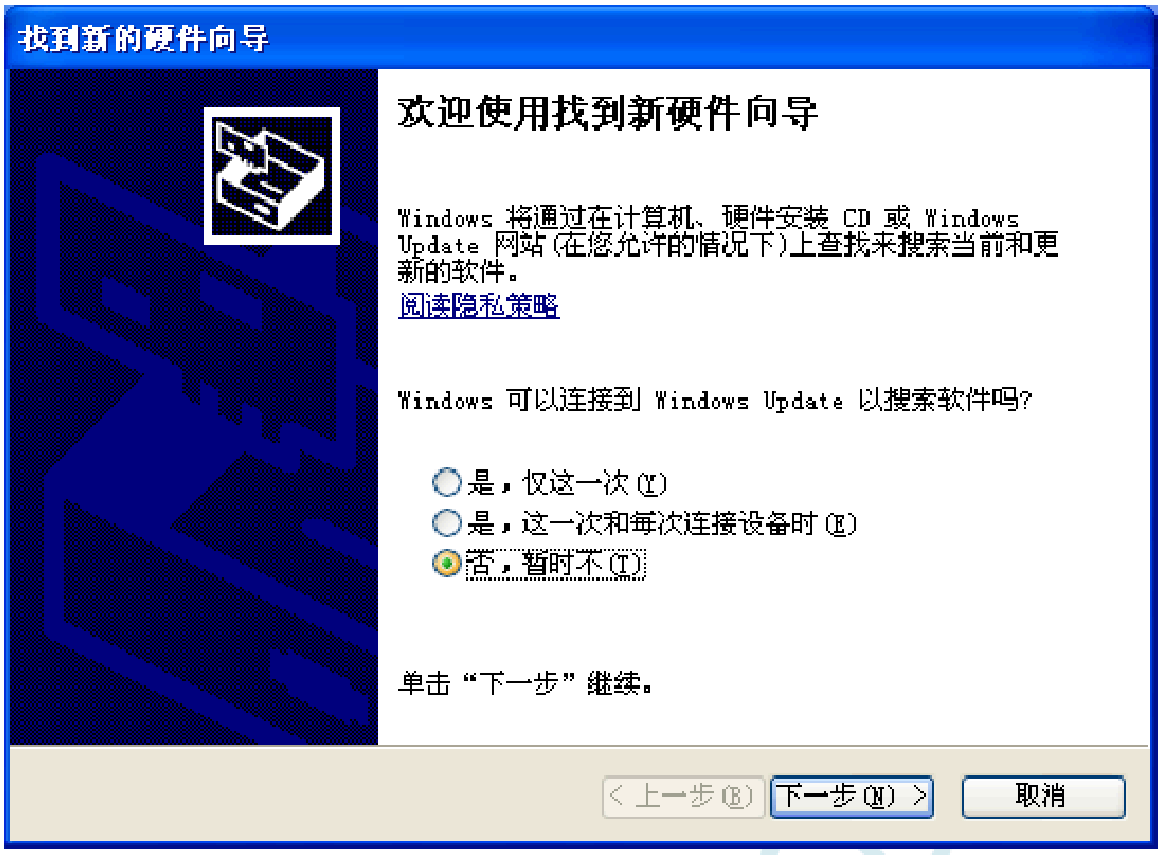
型号	工作电压(V)	程序空间	SRAM	EEPROM	I/O	定时器
STC15F2K08S2	5.5-3.8	8K	2048	53K	42	6
STC15F2K16S2	5.5-3.8	16K	2048	45K	42	6
STC15F2K24S2	5.5-3.8	24K	2048	37K	42	6
STC15F2K32S2	5.5-3.8	32K	2048	29K	42	6
STC15F2K40S2	5.5-3.8	40K	2048	21K	42	6
STC15F2K48S2	5.5-3.8	48K	2048	13K	42	6
STC15F2K56S2	5.5-3.8	56K	2048	5K	42	6
STC15F2K60S2	5.5-3.8	60K	2048	1K	42	6
IAP15F2K61S2	5.5-3.8	61K	2048	IAP	42	6
IRC15F2K63S2	5.5-3.8	63.5K	2048	IAP	42	6
STC15L2K08S2	3.6-2.4	8K	2048	53K	42	6
STC15L2K16S2	3.6-2.4	16K	2048	45K	42	6

有8通道10位精度的A/D
有6通道捕获/比较单元/PWM/DAC

注意: 芯片上电后所有与PWM相关的IO口均为高阻态
需将这些口设置为准双向口或强推挽模式方可正常使用
相关IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
另外还有P1.0/P1.4两个口在上电时为强推挽输出, 程序
初始化时也需将这两个口设置为弱上拉准双向口模式

发布项目程序 | 发布项目帮助 | 读取本机硬盘号 | ☑ 提示音 | 成功计数 0 | 清零

insert USB Device, after the system finds the device, the following dialog box will automatically pop up, select the "No, not for the time being


找到新的硬件向导

欢迎使用找到新硬件向导

Windows 将通过在计算机、硬件安装 CD 或 Windows Update 网站(在您允许的情况下)上查找来搜索当前和更新的软件。

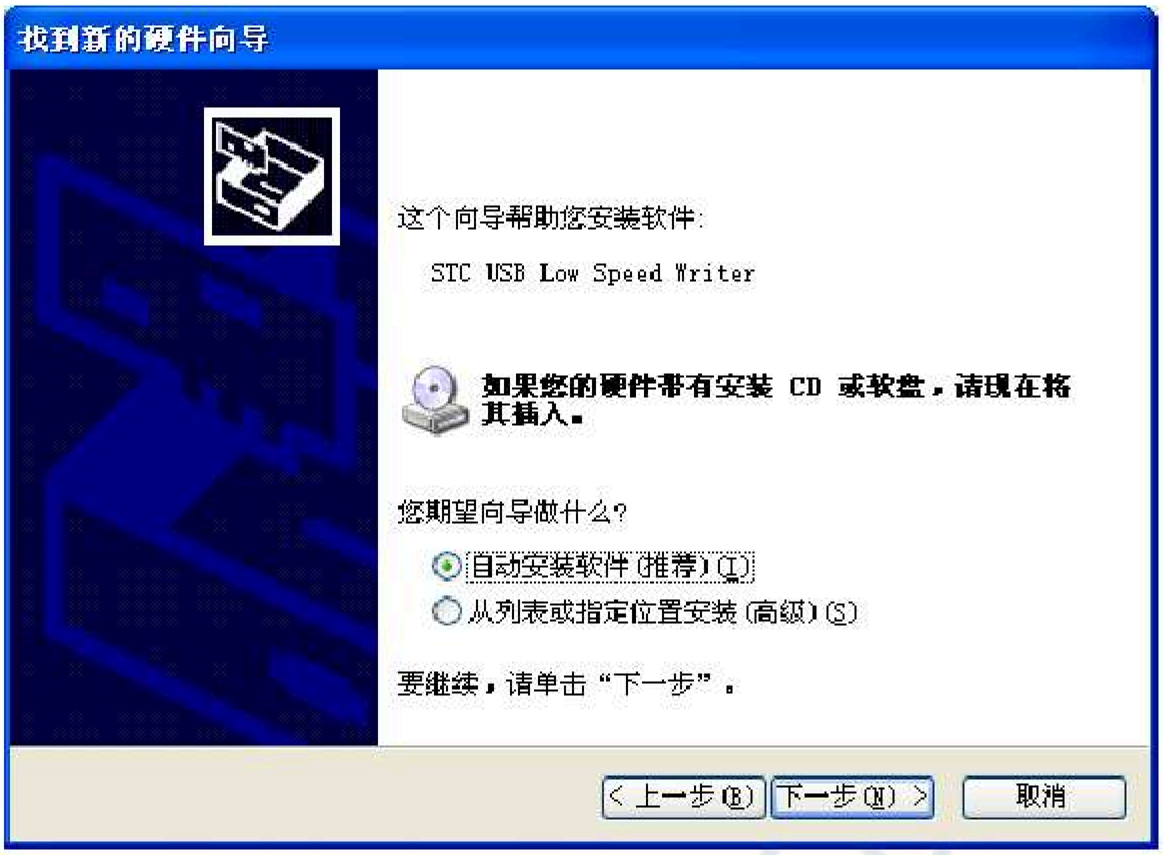
阅读隐私策略

Windows 可以连接到 Windows Update 以搜索软件吗?

- 是，仅这一次(Y)
- 是，这一次和每次连接设备时(E)
- [✓ 否，暂时不(T)]

单击“下一步”继续。

Select "Automatically Install Software" in the dialog box below(recommend)"item


找到新的硬件向导

这个向导帮助您安装软件:

STC USB Low Speed Writer

如果您的硬件带有安装 CD 或软盘，请现在将其插入。

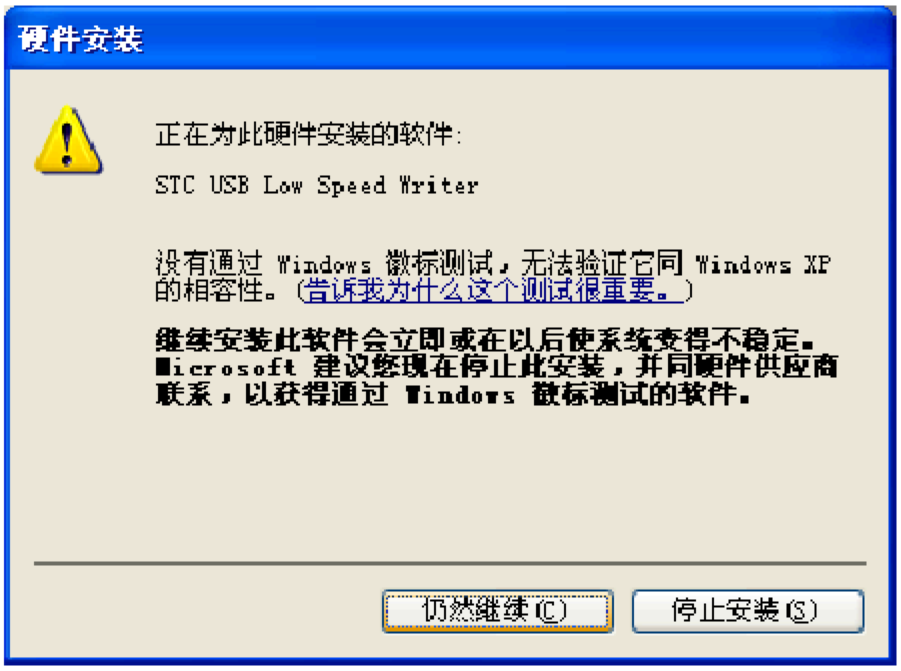
您期望向导做什么?

[✓ 自动安装软件(推荐)(I)]

从列表或指定位置安装(高级)(S)

要继续，请单击"下一步"。

In the following dialog box that pops up, select the "Still Continue" button


硬件安装

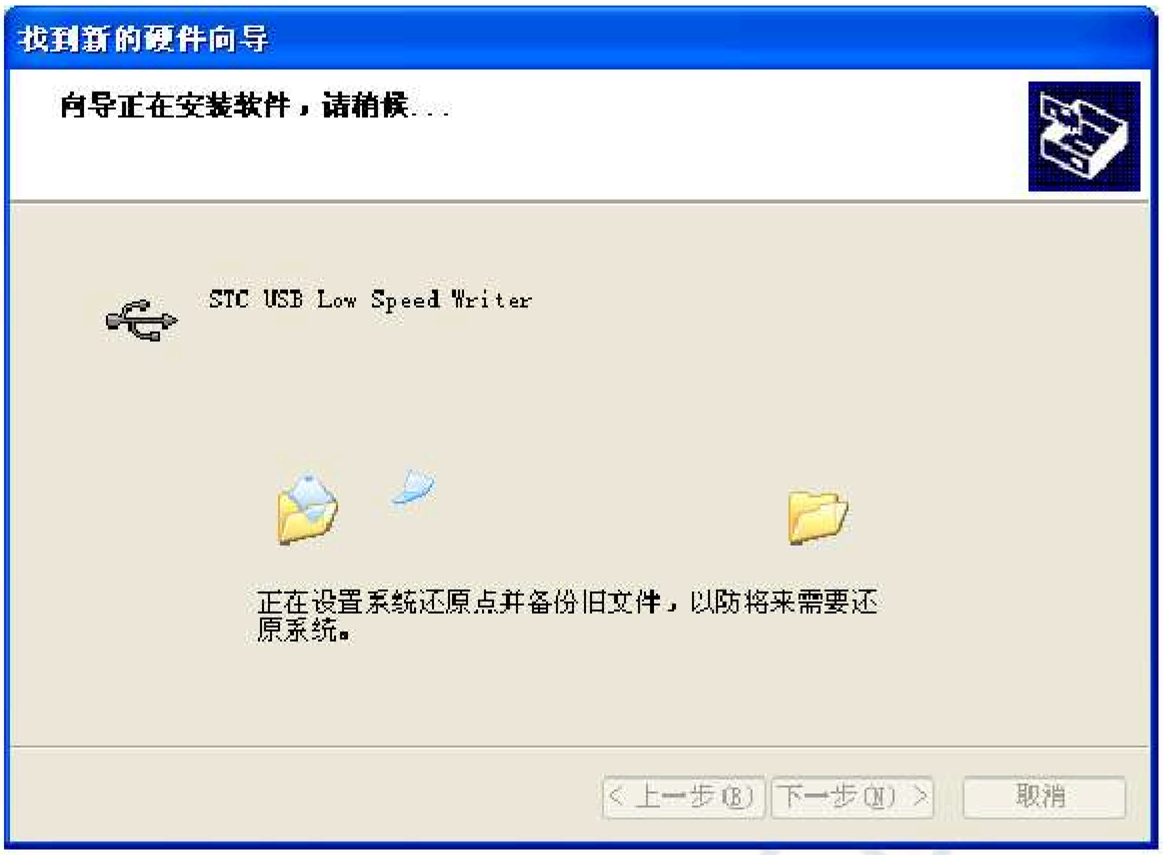
正在为此硬件安装的软件:

STC USB Low Speed Writer

没有通过 Windows 徽标测试，无法验证它同 Windows XP 的相容性。(告诉我为什么这个测试很重要。)

继续安装此软件会立即或在以后使系统变得不稳定。Microsoft 建议您现在停止此安装，并同硬件供应商联系，以获得通过 Windows 徽标测试的软件。

Next, the system will automatically install the driver, as shown in the figure below

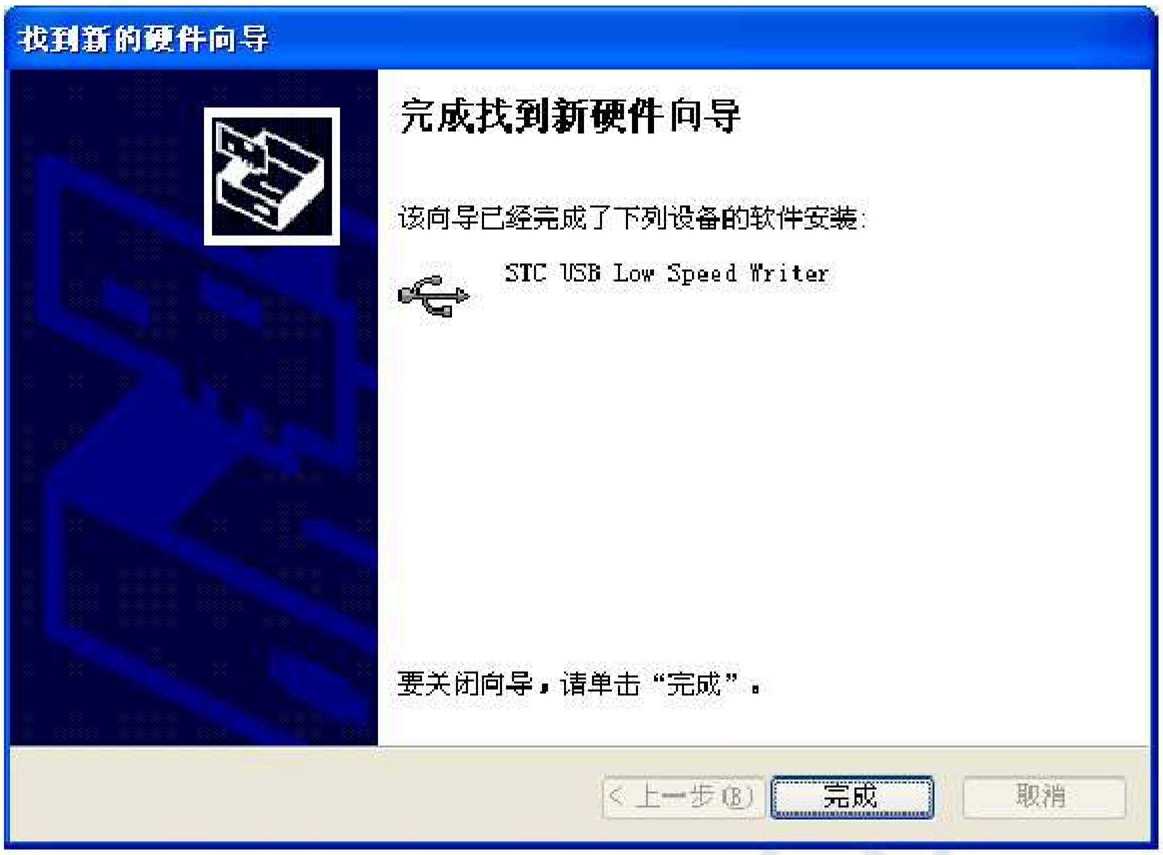

找到新的硬件向导

向导正在安装软件，请稍候...

STC USB Low Speed Writer

正在设置系统还原点并备份旧文件，以防将来需要还原系统。

The following dialog box appears to indicate that the driver installation is complete


找到新的硬件向导

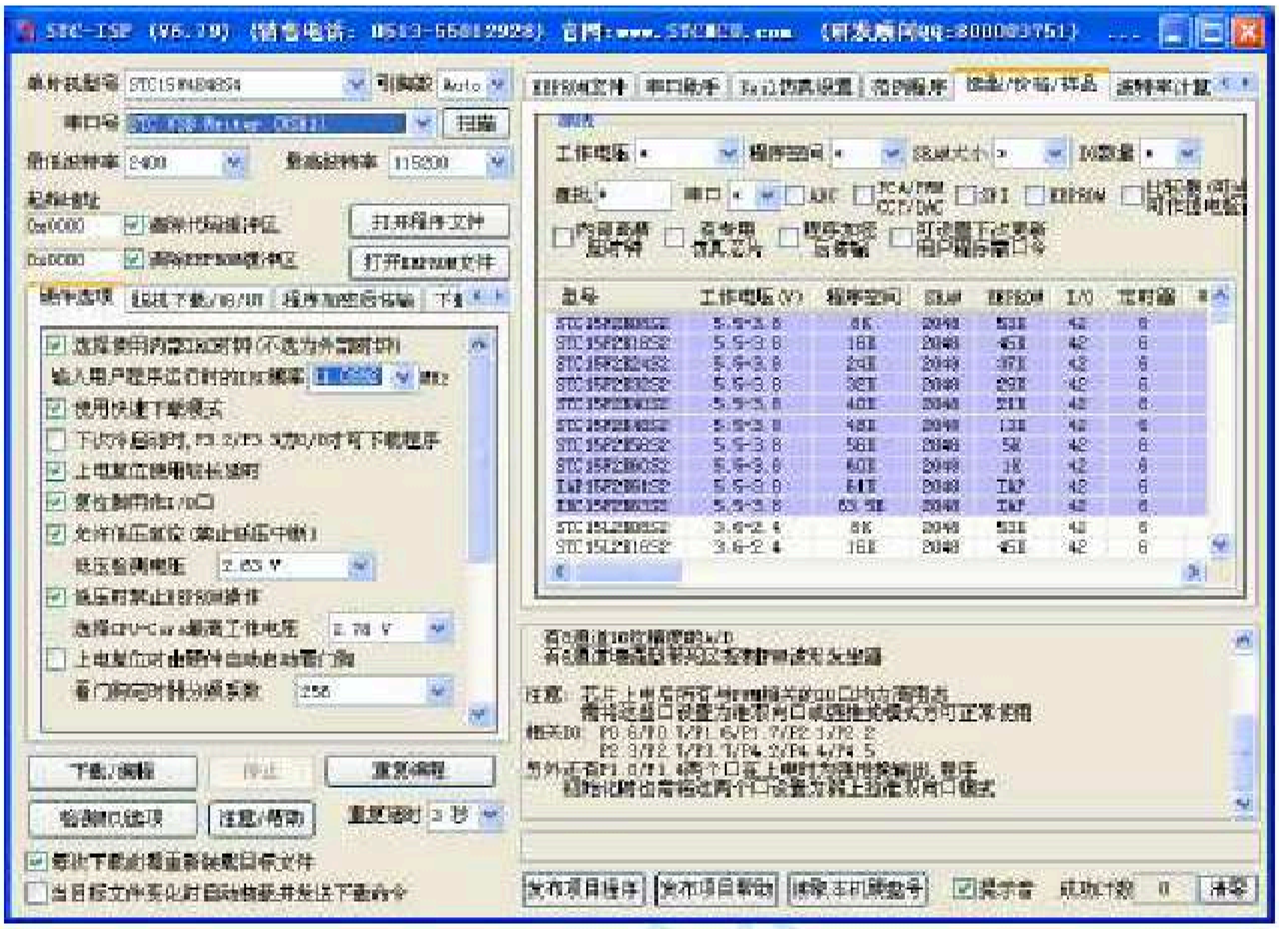
完成找到新硬件向导

该向导已经完成了下列设备的软件安装:

STC USB Low Speed Writer

要关闭向导，请单击“完成”。

At this time, the STC-ISP **The list of serial port numbers in the downloaded software** USB **will automatically select the device and display the inserted one previously opened "** STC USB Writer (USB1) **", as shown below :**



STC-ISP (V6.79) (销售电话: 0513-55012928) 官网:www.STCMCU.com (研发顾问QQ:800003751) ...

单片机型号 STC15W4K48S4 引脚数 Auto
串口号 STC USB Writer (USB1) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区 打开程序文件
0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件

☑ 选择使用内部IRC时钟(不选为外部时钟)
输入用户程序运行时的IRC频率 11.0592 MHz
☑ 使用快速下载模式
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.63 V
☑ 低压时禁止EEPROM操作
选择CPU-Core最高工作电压 2.78 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256

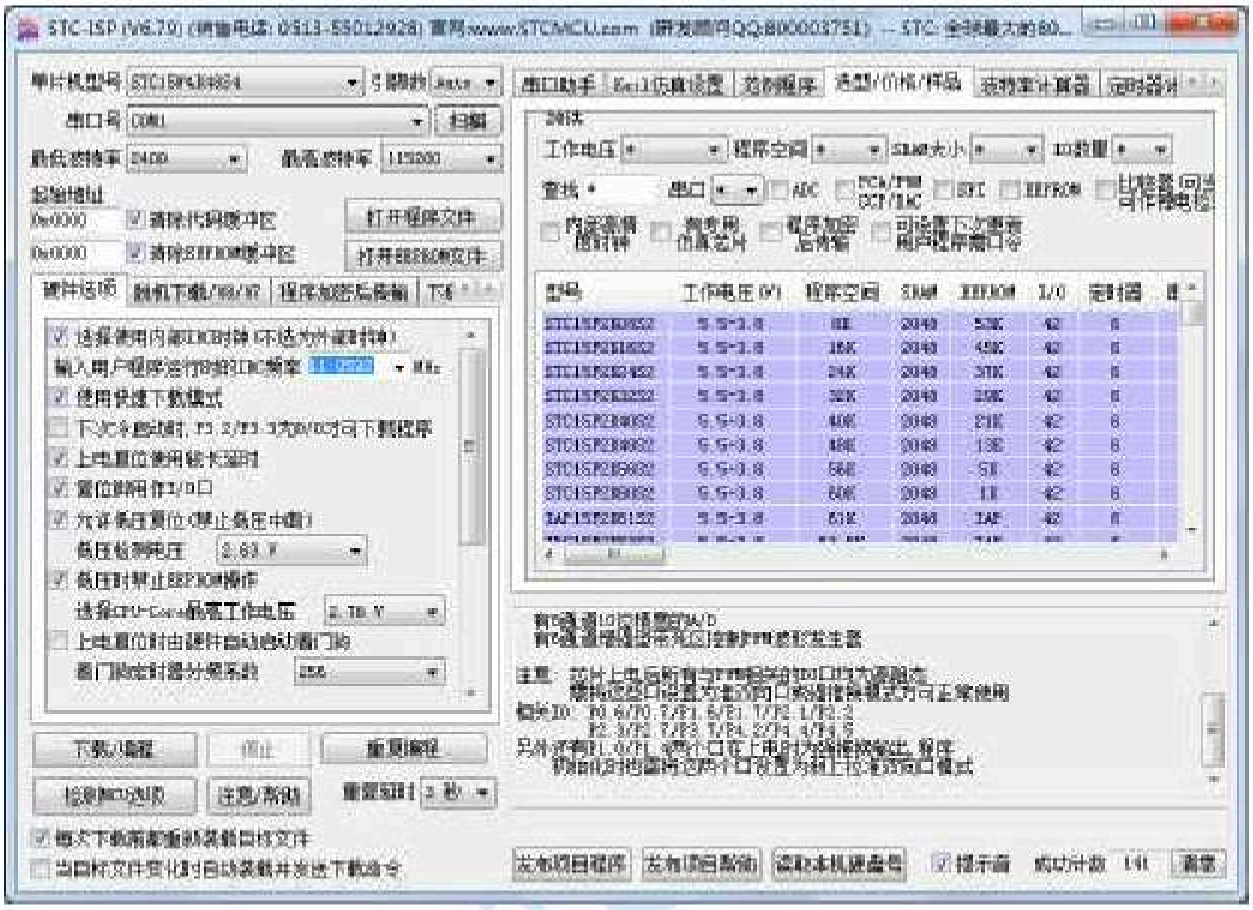
工作电压 程序空间 SRAM大小 IO数量
串口 ☐ ADC ☐ PCA/PWM CCP/DAC ☐ SPI ☐ EEPROM ☐ 比较器
☐ 内部高精度时钟 ☐ 可专用仿真芯片 ☐ 程序加密后传输 ☐ 可设置下次更新用户程序口令

型号	工作电压(V)	程序空间	SRAM	EEPROM	I/O	定时器
STC15F2K08S2	5.5~3.8	8K	2048	53K	42	6
STC15F2K16S2	5.5~3.8	16K	2048	45K	42	6
STC15F2K24S2	5.5~3.8	24K	2048	37K	42	6
STC15F2K32S2	5.5~3.8	32K	2048	29K	42	6
STC15F2K40S2	5.5~3.8	40K	2048	21K	42	6
STC15F2K48S2	5.5~3.8	48K	2048	13K	42	6
STC15F2K56S2	5.5~3.8	56K	2048	5K	42	6
STC15F2K60S2	5.5~3.8	60K	2048	1K	42	6
IAP15F2K61S2	5.5~3.8	61K	2048	IAP	42	6
IRC15F2K63S2	5.5~3.8	63.5K	2048	IAP	42	6
STC15L2K08S2	3.6~2.4	8K	2048	53K	42	6
STC15L2K16S2	3.6~2.4	16K	2048	45K	42	6

注意: 芯片上电后所有与PWM相关的IO口均为高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
相关IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
P2.3/P2.7/P3.7/P4.2/P4.4/P4.5

Windows 7 (32-bit) installation method

Open V6.79 Version (or updated version) of STC-ISP Download the software, the downloaded software will automatically copy the driver fil directory



STC-ISP (V6.79) (销售电话: 0513-55012928) 官网:www.STCMCU.com (研发顾问QQ:800003751) -- STC: 全球最大的8051...

单片机型号 STC15F2K60S2　引脚数 Auto
串口号 COM1　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | 下载

☑ 选择使用内部IRC时钟(不选为外部时钟)
输入用户程序运行时的IRC频率 [illegible] MHz
☑ 使用快速下载模式
☐ 下次冷启动时, P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.63 V
☑ 低压时禁止EEPROM操作
选择CPU-Core最高工作电压 2.78 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 3 秒
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

串口助手 | Keil仿真设置 | 范例程序 | 选型/价格/样品 | 波特率计算器 | 定时器计

筛选
工作电压 *　程序空间 *　SRAM大小 *　IO数量 *
查找 *　串口 *　☐ ADC　☐ PCA/PWM DCP/DAC　☐ RTC　☐ EEPROM　☐ 比较器 可当可作掉电检
☐ 内部高精度时钟　☐ 可专用仿真芯片　☐ 程序加密后传输　☐ 可设置下次更新用户程序命令

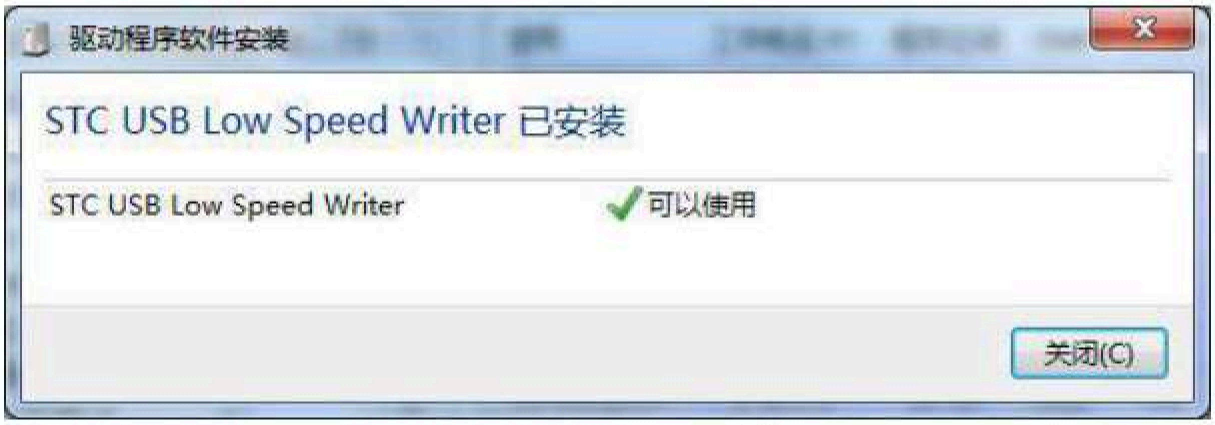
型号	工作电压(V)	程序空间	SRAM	EEPROM	I/O	定时器
STC15F2K08S2	5.5~3.8	8K	2048	53K	42	6
STC15F2K16S2	5.5~3.8	16K	2048	45K	42	6
STC15F2K24S2	5.5~3.8	24K	2048	37K	42	6
STC15F2K32S2	5.5~3.8	32K	2048	29K	42	6
STC15F2K40S2	5.5~3.8	40K	2048	21K	42	6
STC15F2K48S2	5.5~3.8	48K	2048	13K	42	6
STC15F2K56S2	5.5~3.8	56K	2048	5K	42	6
STC15F2K60S2	5.5~3.8	60K	2048	1K	42	6
IAP15F2K61S2	5.5~3.8	61K	2048	IAP	42	6

有8通道10位精度的A/D
有3通道捕获/比较单元(可实现PWM或DAC)发生器

注意: 芯片上电后所有与PWM相关的IO口均为高阻输入
需将这些口设置为准双向口或强推挽模式方可正常使用
相关IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
另外P1.0/P1.4两个口在上电时为强推挽输出, 程序
初始化时需要将这两个口设置为弱上拉准双向口模式

发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 141　清零

insert USB Device, the system will automatically install the driver after finding the device. After the installation is complete, there will be the f

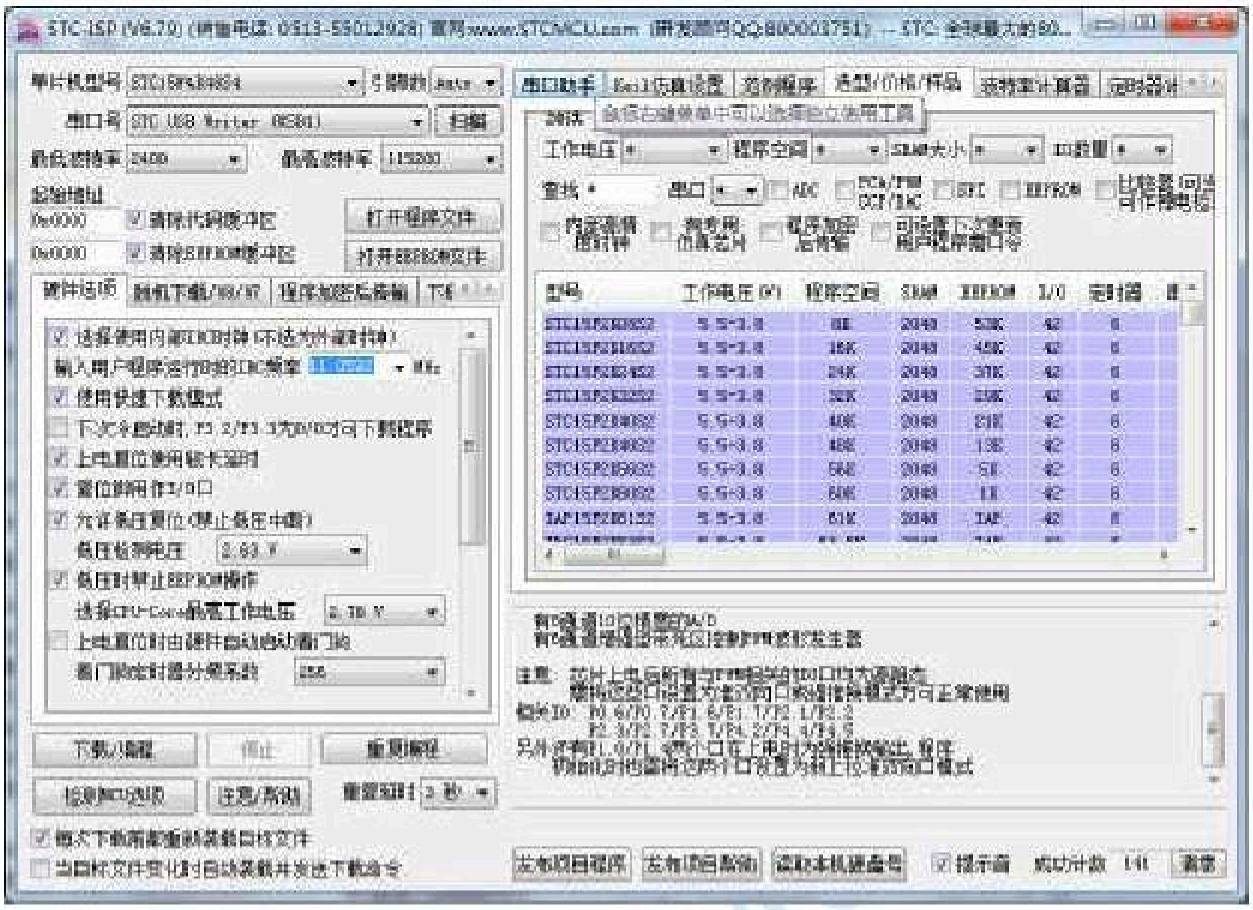

驱动程序软件安装

STC USB Low Speed Writer 已安装

STC USB Low Speed Writer ✓可以使用

关闭(C)

At this time, the STC-ISP The list of serial port numbers in the downloaded software will automatically display the inserted USB device, and by select the same one previously opened "STC USB Writer (USB1)" as shown below:



STC-ISP (V6.79) (销售电话: 0513-55012928) 官网:www.STCMCU.com (研发顾问QQ:800003751) -- STC: 全球最大的8...

单片机型号 STC15F2K60S2　引脚数 Auto
串口号 STC USB Writer (USB1)　扫描
最低波特率 2400　最高波特率 115200
起始地址 0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | ...

☑ 选择使用内部IRC时钟(不选为外部时钟)
输入用户程序运行时的IRC频率 11.0592 MHz
☑ 使用快速下载模式
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.83 V
☑ 低压时禁止EEPROM操作
选择CPU-Core最高工作电压 2.78 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 3 秒
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

串口助手 | Keil仿真设置 | 范例程序 | 选型/价格/样品 | 波特率计算器 | 定时器计算

工作电压 *　程序空间 *　SRAM大小 *　IO数量 *
查找 *　串口 *　☐ ADC　☐ PCA/PWM DCP/DAC　☐ RTC　☐ EEPROM　☐ 比较器可作掉电检测
☐ 内部高精度时钟　☐ 可仿真芯片　☐ 程序加密后传输　☐ 可设置下次更新用户程序口令

型号	工作电压(V)	程序空间	SRAM	EEPROM	I/O	定时器
STC15F2K08S2	5.5~3.8	8K	2048	53K	42	6
STC15F2K16S2	5.5~3.8	16K	2048	45K	42	6
STC15F2K24S2	5.5~3.8	24K	2048	37K	42	6
STC15F2K32S2	5.5~3.8	32K	2048	29K	42	6
STC15F2K40S2	5.5~3.8	40K	2048	21K	42	6
STC15F2K48S2	5.5~3.8	48K	2048	13K	42	6
STC15F2K56S2	5.5~3.8	56K	2048	5K	42	6
STC15F2K60S2	5.5~3.8	60K	2048	1K	42	6
IAP15F2K61S2	5.5~3.8	61K	2048	IAP	42	6

有8通道10位精度的A/D
有3通道捕获/比较单元(PCA)控制的PWM/DAC发生器
注意: 芯片上电后所有与PWM相关的IO口均为高阻态
需将这些口设置为准双向口或强推挽模式方可正常使用
相关IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
另外P1.0/P1.4/P5.4作外口在上电时为强推挽输出, 程序
初始化时也需将这些口设置为弱上拉准双向口模式

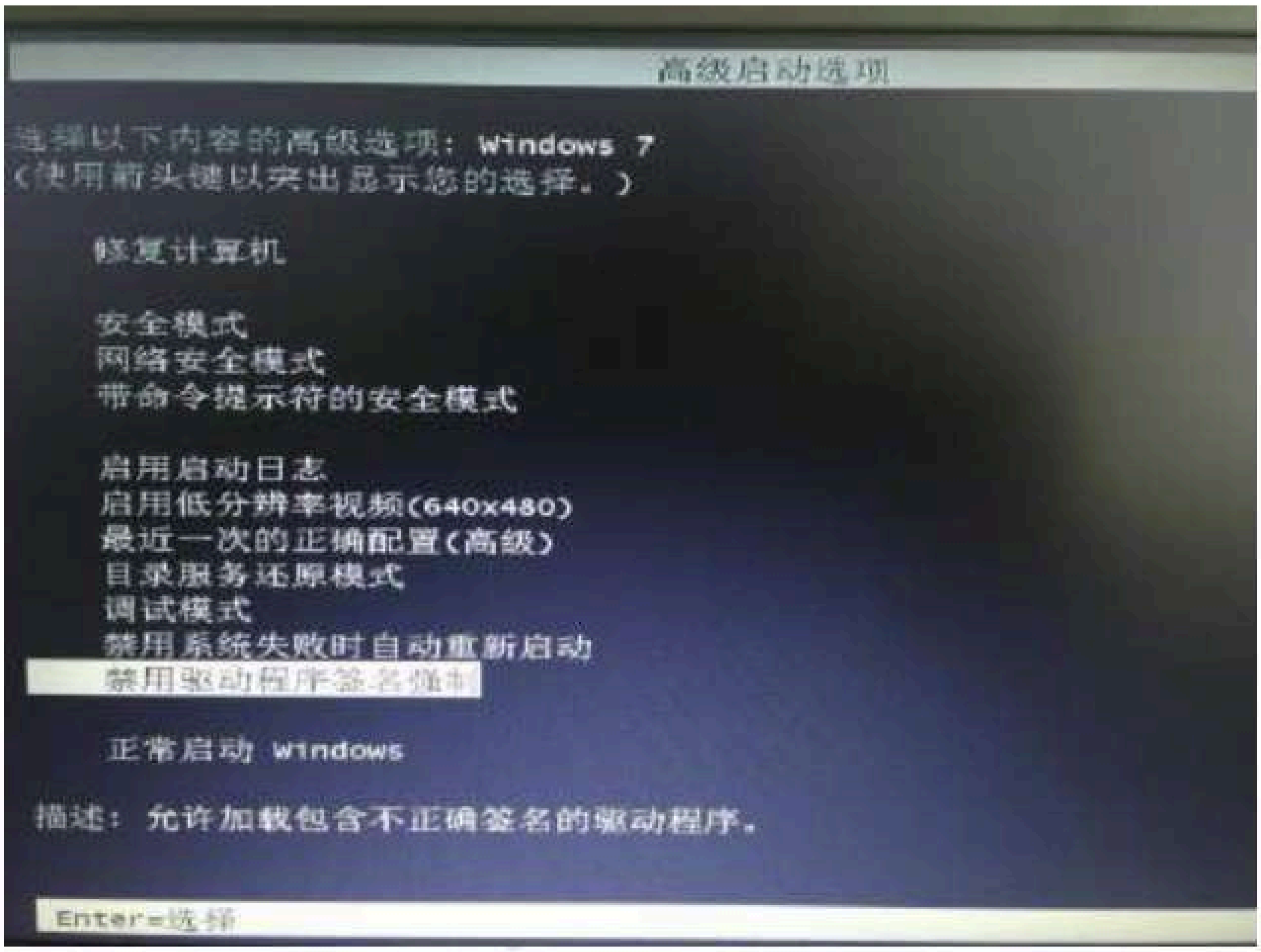
发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 141　清零

Note: if Windows 7 Next, the system does not automatically install the driver, please refer to the installation method of the driver. Windows 8 (32 Installation of this driver.

Windows 7 (64-bit) installation method

Due to Windows7 64 Under the default state of the bit operating system, drivers that are not digitally signed cannot be installed successfully
install STC-USB , before driving, you need to follow the steps below to temporarily skip the digital signature, and the installation will be suc

Restart the computer first and keep pressing F8 until the following startup screen appears



高级启动选项

选择以下内容的高级选项: Windows 7
(使用箭头键以突出显示您的选择。)

修复计算机

安全模式
网络安全模式
带命令提示符的安全模式

启用启动日志
启用低分辨率视频(640x480)
最近一次的正确配置(高级)
目录服务还原模式
调试模式
禁用系统失败时自动重新启动
禁用驱动程序签名强制

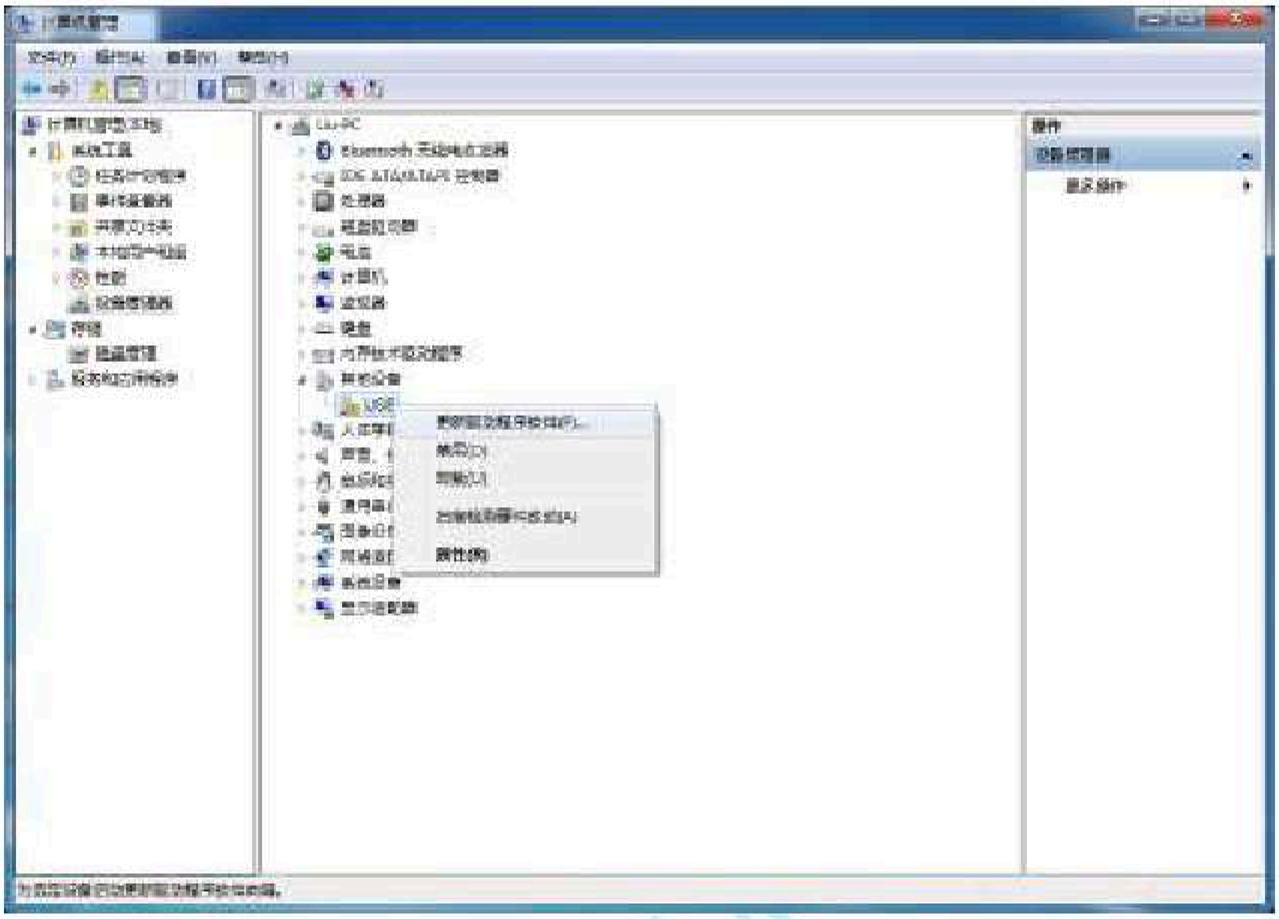
正常启动 Windows

描述: 允许加载包含不正确签名的驱动程序。

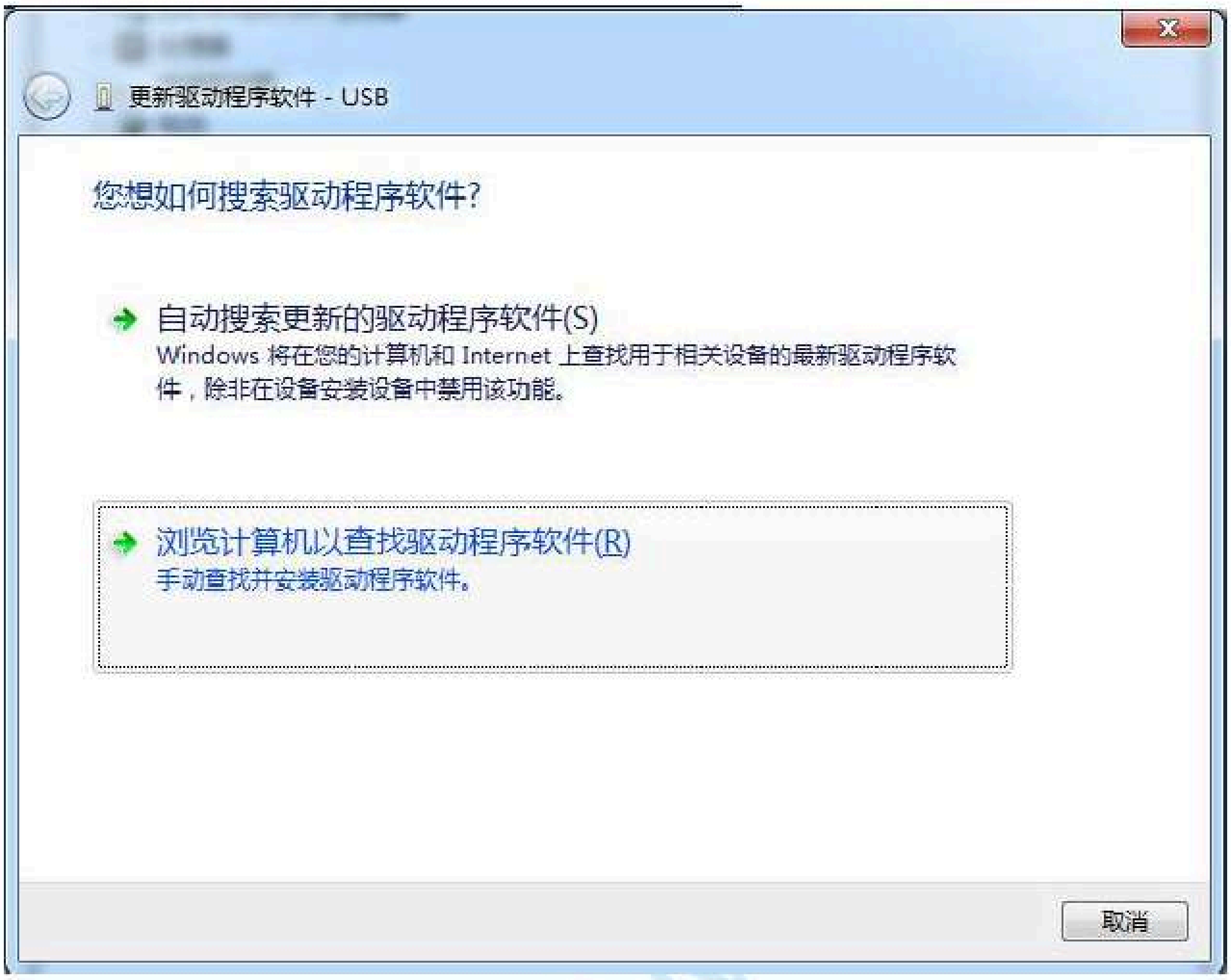
Enter=选择

Select "Disable driver signature enforcement", The digital signature verification function can be temporarily turned off after startup

Device, and open "Device Manager" Find the insert with a yellow exclamation mark in the device list USB Device, in the right-click menu of the device In, select "Update driver software"



In the dialog box below, select "Browse computer to find driver software"



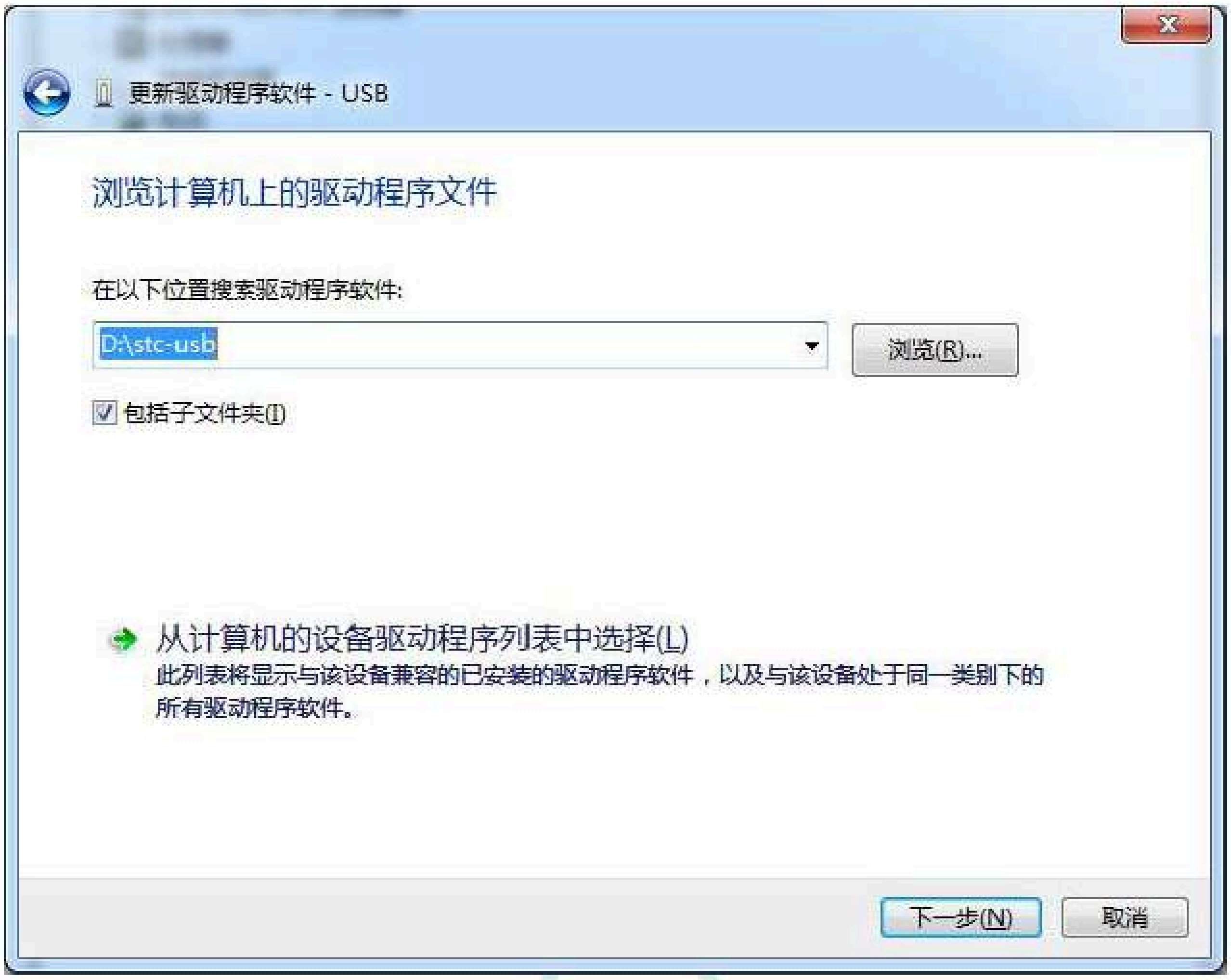
您想如何搜索驱动程序软件?

自动搜索更新的驱动程序软件(S)
Windows 将在您的计算机和 Internet 上查找用于相关设备的最新驱动程序软件，除非在设备安装设备中禁用该功能。

浏览计算机以查找驱动程序软件(R)
手动查找并安装驱动程序软件。

Click the "Browse" button in the dialog box below to find the previous storage directory of the driver (for example: the previous sample directory

For "", the user sets the path D:\STC-USB Bit to actually The decompression Contents)


更新驱动程序软件 - USB

浏览计算机上的驱动程序文件

在以下位置搜索驱动程序软件:

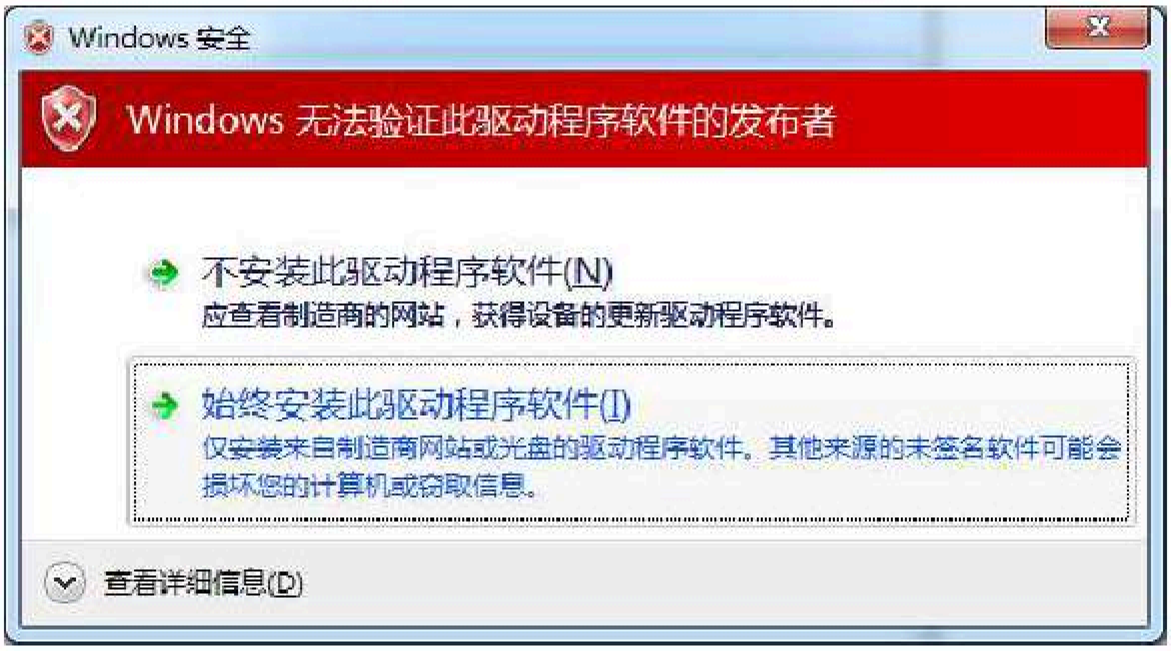
D:\stc-usb

☑ 包括子文件夹(I)

从计算机的设备驱动程序列表中选择(L)

此列表将显示与该设备兼容的已安装的驱动程序软件，以及与该设备处于同一类别下的所有驱动程序软件。

When the driver starts to be installed, the following dialog box will pop up, select "Always install this driver software"



Windows 安全

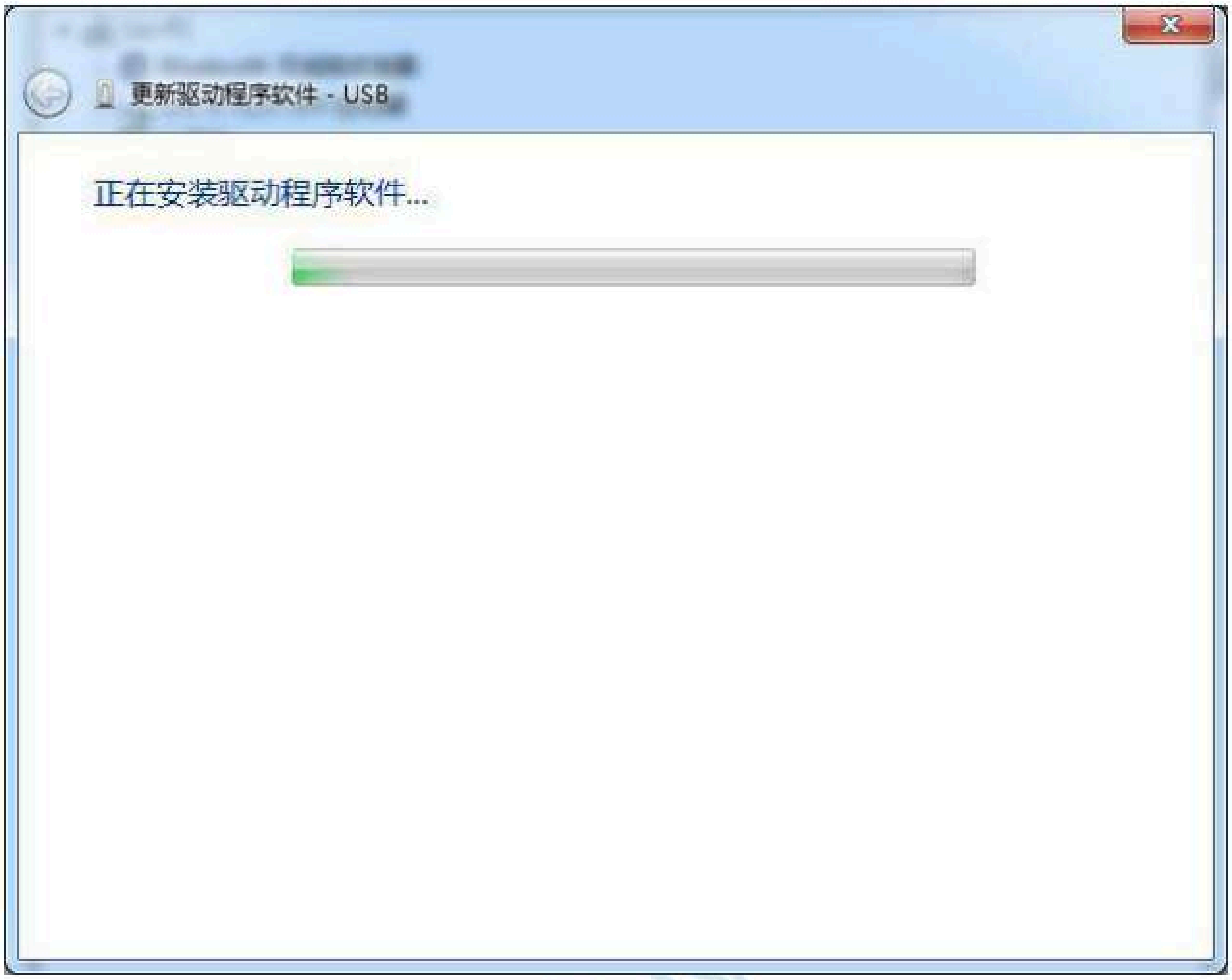
Windows 无法验证此驱动程序软件的发布者

不安装此驱动程序软件(N)
应查看制造商的网站，获得设备的更新驱动程序软件。

始终安装此驱动程序软件(I)
仅安装来自制造商网站或光盘的驱动程序软件。其他来源的未签名软件可能会损坏您的计算机或窃取信息。

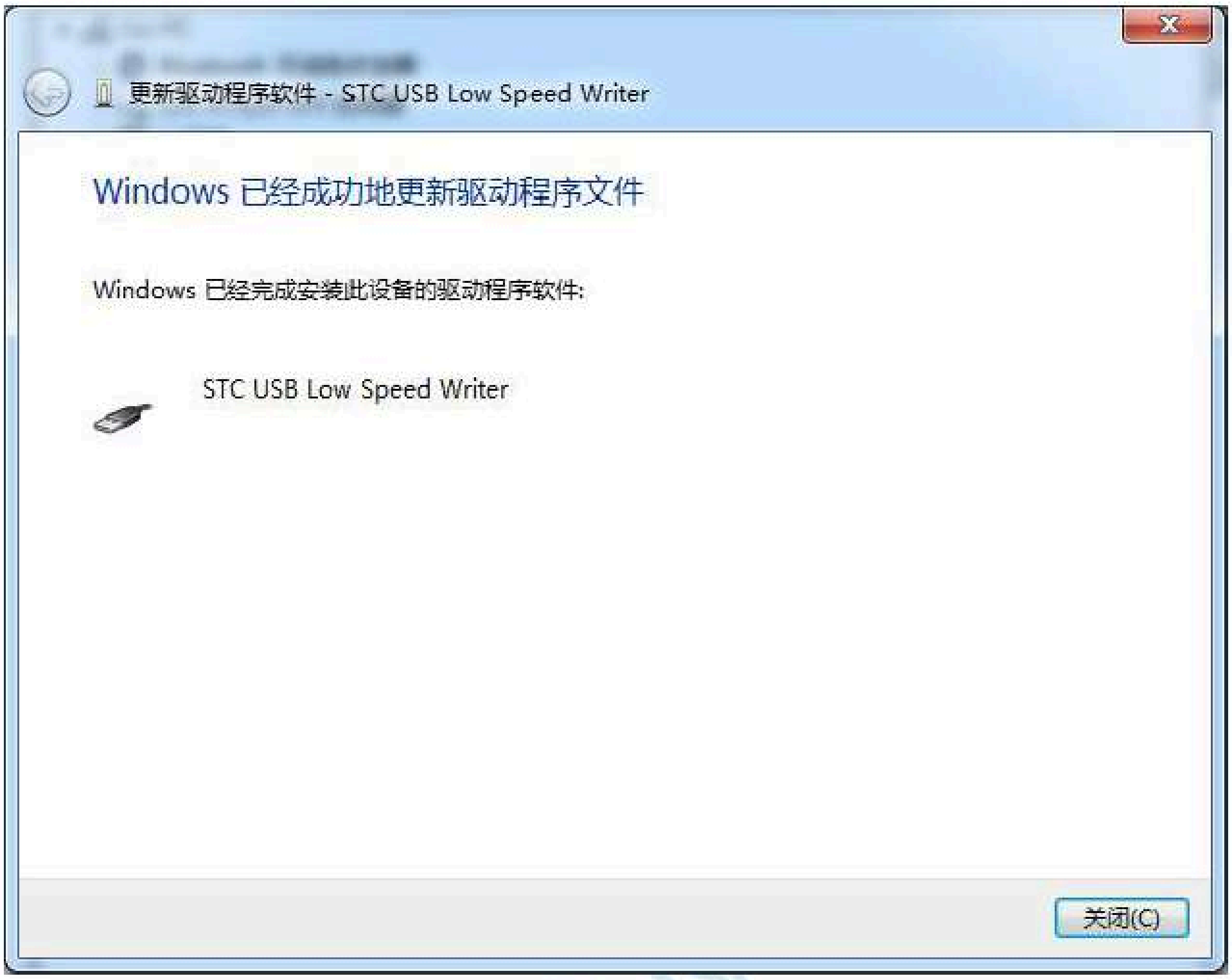
查看详细信息(D)

Next, the system will automatically install the driver, as shown in the figure below


更新驱动程序软件 - USB

正在安装驱动程序软件...

The following dialog box appears to indicate that the driver installation is complete

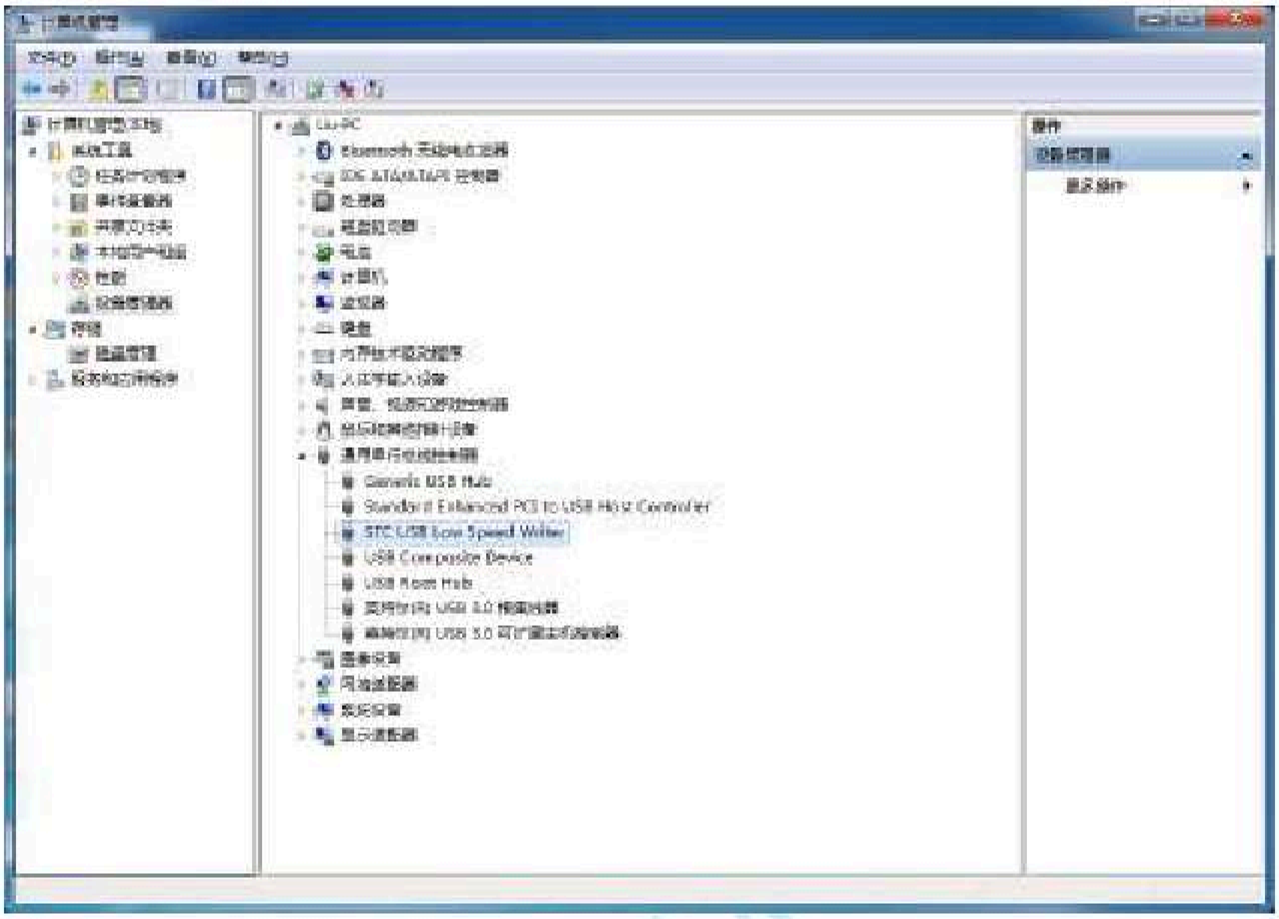

更新驱动程序软件 - STC USB Low Speed Writer

Windows 已经成功地更新驱动程序文件

Windows 已经完成安装此设备的驱动程序软件:

STC USB Low Speed Writer

At this time, in the device manager, the device with the yellow exclamation mark before will be displayed as " Backup name" at this time. STC USB Low Speed Writer **"The setting**


计算机管理

文件(F) 操作(A) 查看(V) 帮助(H)

- 计算机管理(本地)
  - 系统工具
    - 任务计划程序
    - 事件查看器
    - 共享文件夹
    - 本地用户和组
    - 性能
    - 设备管理器
  - 存储
    - 磁盘管理
  - 服务和应用程序

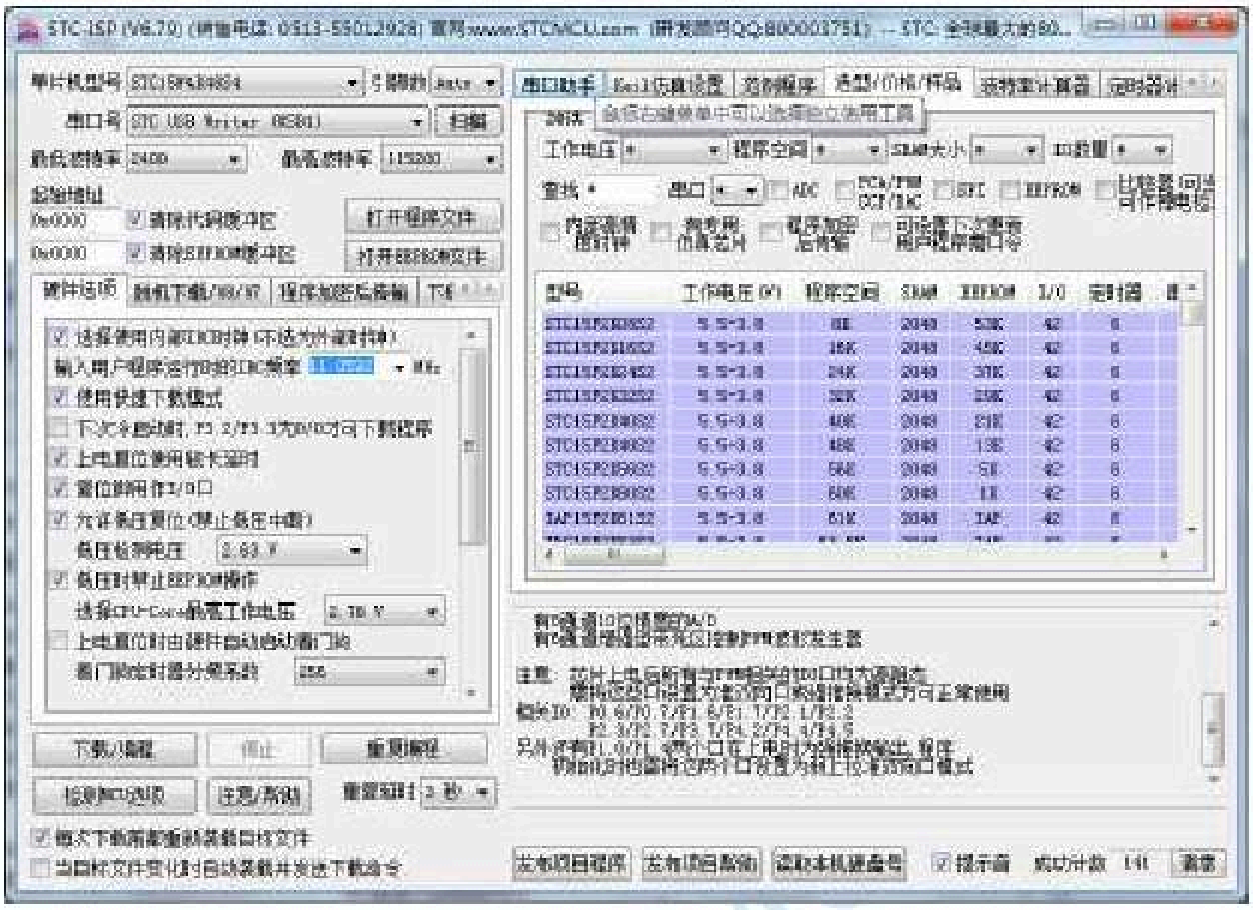
- Liu-PC
  - Bluetooth 无线电收发器
  - IDE ATA/ATAPI 控制器
  - 处理器
  - 磁盘驱动器
  - 电池
  - 计算机
  - 监视器
  - 键盘
  - 内存技术驱动程序
  - 人体学输入设备
  - 声音、视频和游戏控制器
  - 鼠标和其他指针设备
  - 通用串行总线控制器
    - Generic USB Hub
    - Standard Enhanced PCI to USB Host Controller
    - STC USB Low Speed Writer
    - USB Composite Device
    - USB Root Hub
    - 英特尔(R) USB 3.0 根集线器
    - 英特尔(R) USB 3.0 可扩展主机控制器
  - 图像设备
  - 网络适配器
  - 系统设备
  - 显示适配器

操作

设备管理器

更多操作

Opened before STC-ISP download The list of serial port numbers in the software will automatically select the inserted one USB Device, and display the device name as " STC USB Writer (USB1) ", as shown below：


STC-ISP (V6.79) (销售电话: 0513-55012928) 官网:www.STCMCU.com (研发顾问QQ:800003751) -- STC: 全球最大的8051...

单片机型号 STC15F2K60S2　引脚数 Auto
串口号 STC USB Writer (USB1)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | TC

☑ 选择使用内部IRC时钟 (不选为外部时钟)
输入用户程序运行时的IRC频率 11.0592 MHz
☑ 使用快速下载模式
☐ 下次冷启动时, P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位 (禁止低压中断)
低压检测电压 2.83 V
☑ 低压时禁止EEPROM操作
选择CPU-Core最高工作电压 2.70 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 3 秒
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

串口助手 | Keil仿真设置 | 范例程序 | 选型/价格/样品 | 波特率计算器 | 定时器计算

鼠标右键菜单中可以选择独立使用工具
筛选
工作电压 *　程序空间 *　SRAM大小 *　IO数量 *
查找 *　串口 *　☐ ADC　☐ PCA/PWM CCP/DAC　☐ SPI　☐ EEPROM　☐ 比较器 可当作掉电检测
☐ 内部高精度时钟　☐ 可专用仿真芯片　☐ 程序加密后传输　☐ 可设置下次更新用户程序口令

型号	工作电压 (V)	程序空间	SRAM	EEPROM	I/O	定时器
STC15F2K08S2	5.5~3.8	8K	2048	53K	42	6
STC15F2K16S2	5.5~3.8	16K	2048	45K	42	6
STC15F2K24S2	5.5~3.8	24K	2048	37K	42	6
STC15F2K32S2	5.5~3.8	32K	2048	29K	42	6
STC15F2K40S2	5.5~3.8	40K	2048	21K	42	6
STC15F2K48S2	5.5~3.8	48K	2048	13K	42	6
STC15F2K56S2	5.5~3.8	56K	2048	5K	42	6
STC15F2K60S2	5.5~3.8	60K	2048	1K	42	6
IAP15F2K61S2	5.5~3.8	61K	2048	IAP	42	6

有8通道10位精度的A/D
有6通道增强型可实现PWM的硬件PWM发生器

注意: 芯片上电后所有与PWM相关的IO口均为高阻态
需将这些口设置为准双向口或强推挽模式方可正常使用
相关IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
另外P1.0/P1.4两个口在上电时为强推挽输出, 程序
初始化时也需将这两个口设置为弱上拉准双向口模式

发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 141　清零

Windows 8 (32-bit) installation method

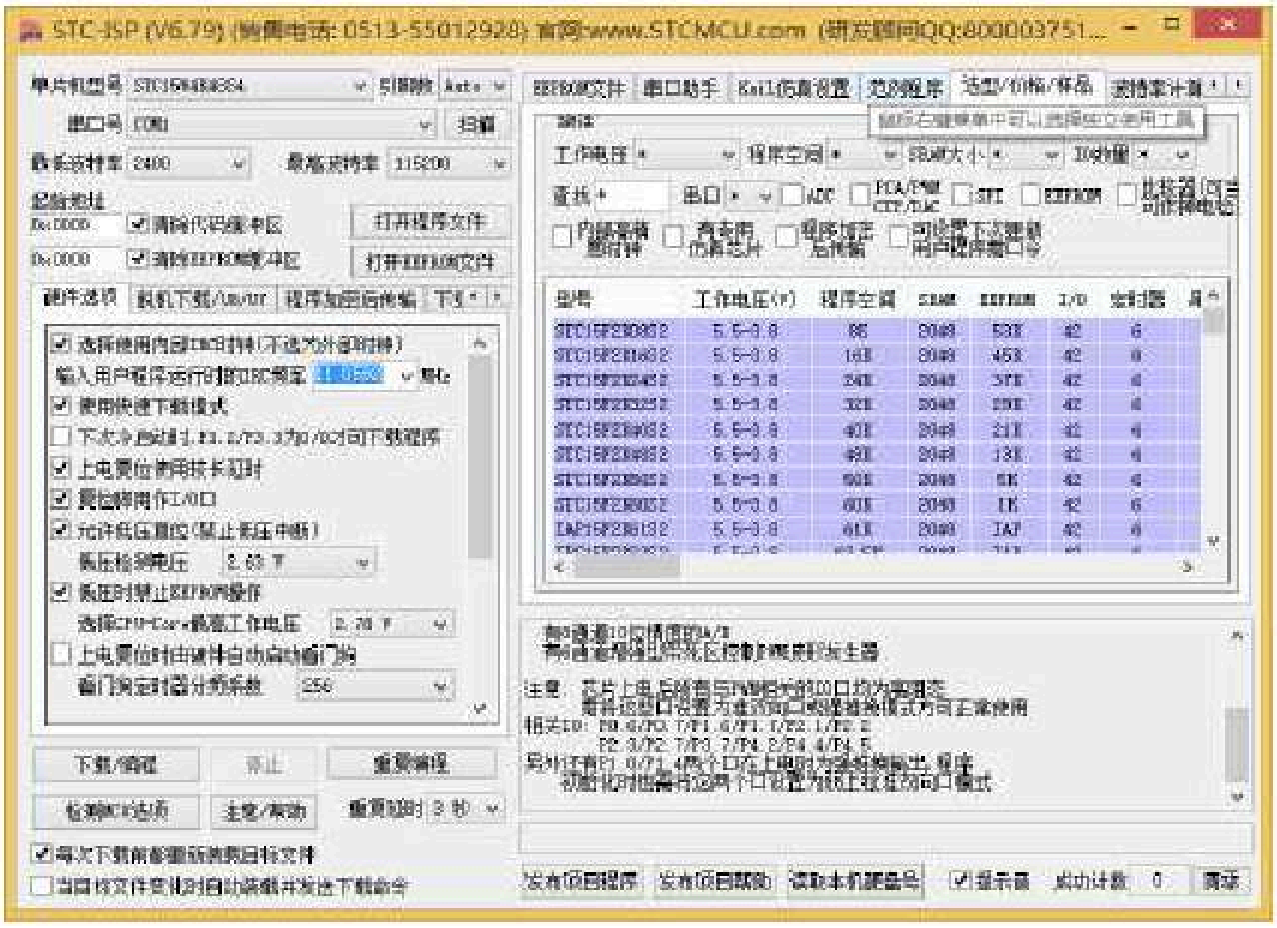
open V6.79 Version (or updated version) of STC-ISP Download software (Due to permissions, in Windows system Do not download the software

The driver files will be copied to the relevant system directory and need to be installed manually by the Official website download STC

user. First from "(or later version), download and unzip to the local disk, then stc-isp-15xx-v6.79. zip STC-USB " The driver file will also be extracted

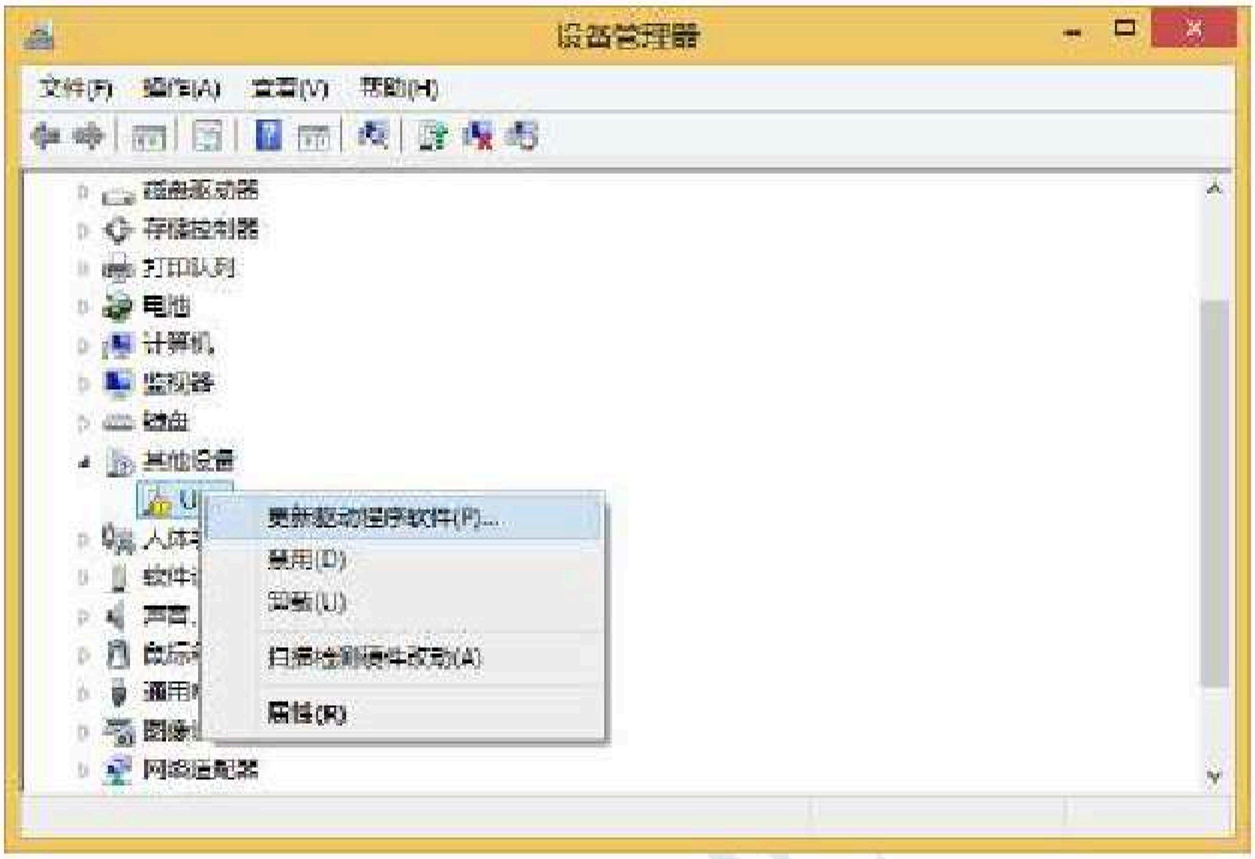
Go to "" in the current decompression directory (for example, the downloaded compressed file "Unzip to" stc-isp-15xx-v6.79. zip F:\ , STC-USB

Driver The driver is in the "" directory)) STC-USB F:\STC-USB Driver

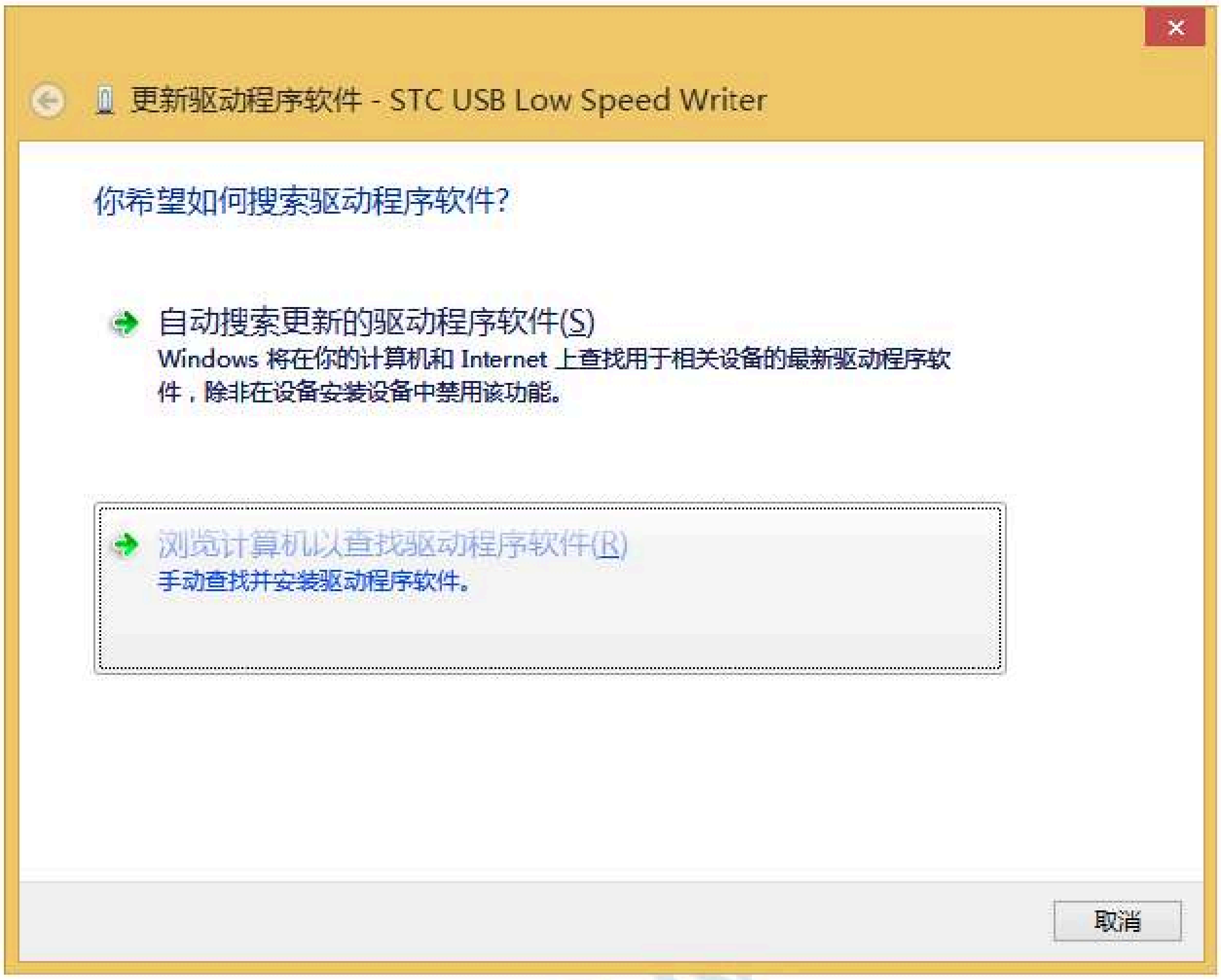

STC-ISP (V6.79) (销售电话: 0513-55012928) 官网:www.STCMCU.com (研发顾问QQ:800003751...

型号	工作电压(V)	程序空间	SRAM	EEPROM	I/O	定时器
STC15F2K08S2	5.5-3.8	8K	2048	53K	42	6
STC15F2K16S2	5.5-3.8	16K	2048	45K	42	6
STC15F2K24S2	5.5-3.8	24K	2048	37K	42	6
STC15F2K32S2	5.5-3.8	32K	2048	29K	42	6
STC15F2K40S2	5.5-3.8	40K	2048	21K	42	6
STC15F2K48S2	5.5-3.8	48K	2048	13K	42	6
STC15F2K56S2	5.5-3.8	56K	2048	5K	42	6
STC15F2K60S2	5.5-3.8	60K	2048	1K	42	6
IAP15F2K61S2	5.5-3.8	61K	2048	IAP	42	6

First insert USB Device, and open "Device Manager" to find the device with a yellow exclamation mark in the device list, in the right-click menu of the device In, select "Update driver software"



In the dialog box below, select "Browse computer to find driver software"



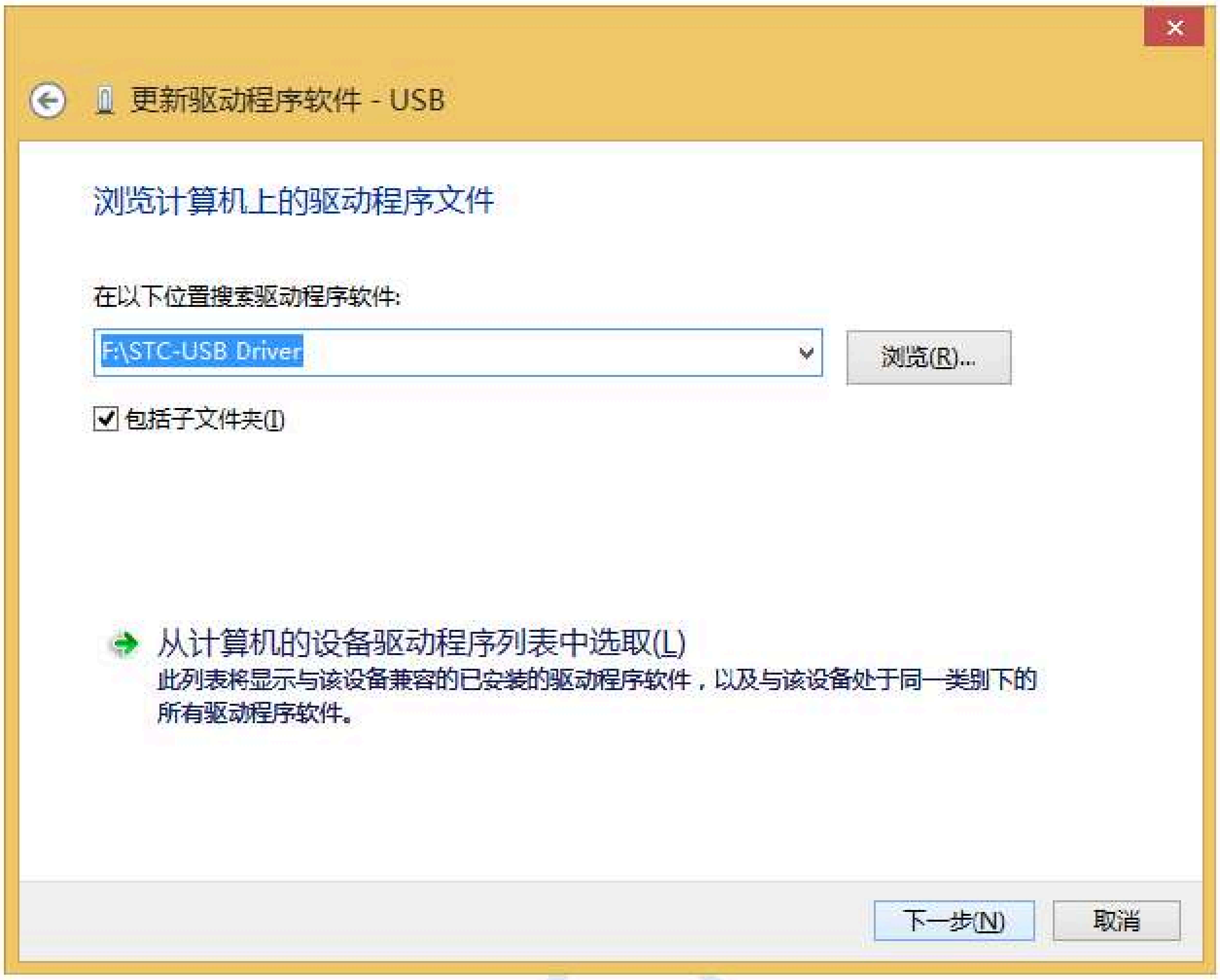
更新驱动程序软件 - STC USB Low Speed Writer

你希望如何搜索驱动程序软件?

自动搜索更新的驱动程序软件(S)
Windows 将在你的计算机和 Internet 上查找用于相关设备的最新驱动程序软件，除非在设备安装设备中禁用该功能。

浏览计算机以查找驱动程序软件(R)
手动查找并安装驱动程序软件。

Click the "Browse" button in the dialog STC-USB **The storage directory of the driver (for example: the previous sample dir**
box below to find the previous "user-located" The user locates the path to the actual decompression directory)


更新驱动程序软件 - USB

浏览计算机上的驱动程序文件

在以下位置搜索驱动程序软件:

F:\STC-USB Driver

浏览(R)...

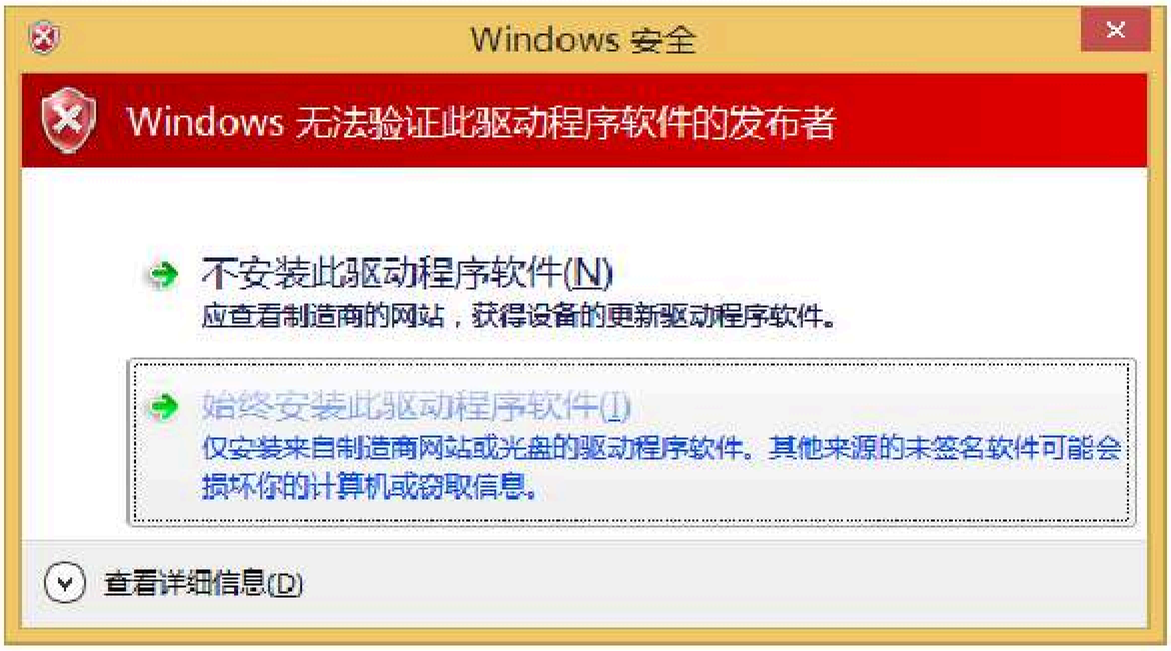
☑ 包括子文件夹(I)

从计算机的设备驱动程序列表中选取(L)

此列表将显示与该设备兼容的已安装的驱动程序软件，以及与该设备处于同一类别下的所有驱动程序软件。

下一步(N) 取消

When the driver starts to be installed, the following dialog box will pop up, select "Always install this driver software"



Windows 安全

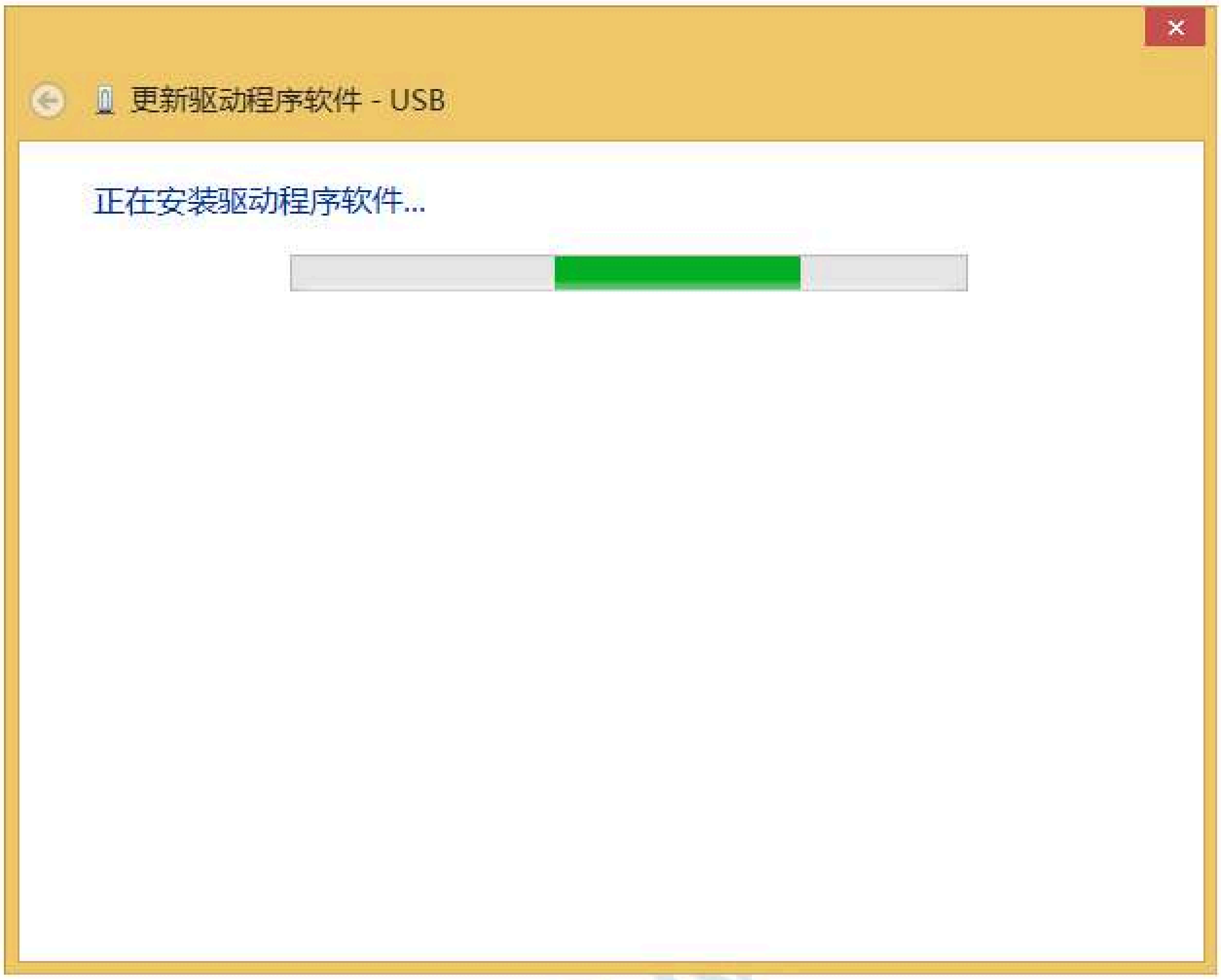
Windows 无法验证此驱动程序软件的发布者

不安装此驱动程序软件(N)
应查看制造商的网站，获得设备的更新驱动程序软件。

始终安装此驱动程序软件(I)
仅安装来自制造商网站或光盘的驱动程序软件。其他来源的未签名软件可能会损坏你的计算机或窃取信息。

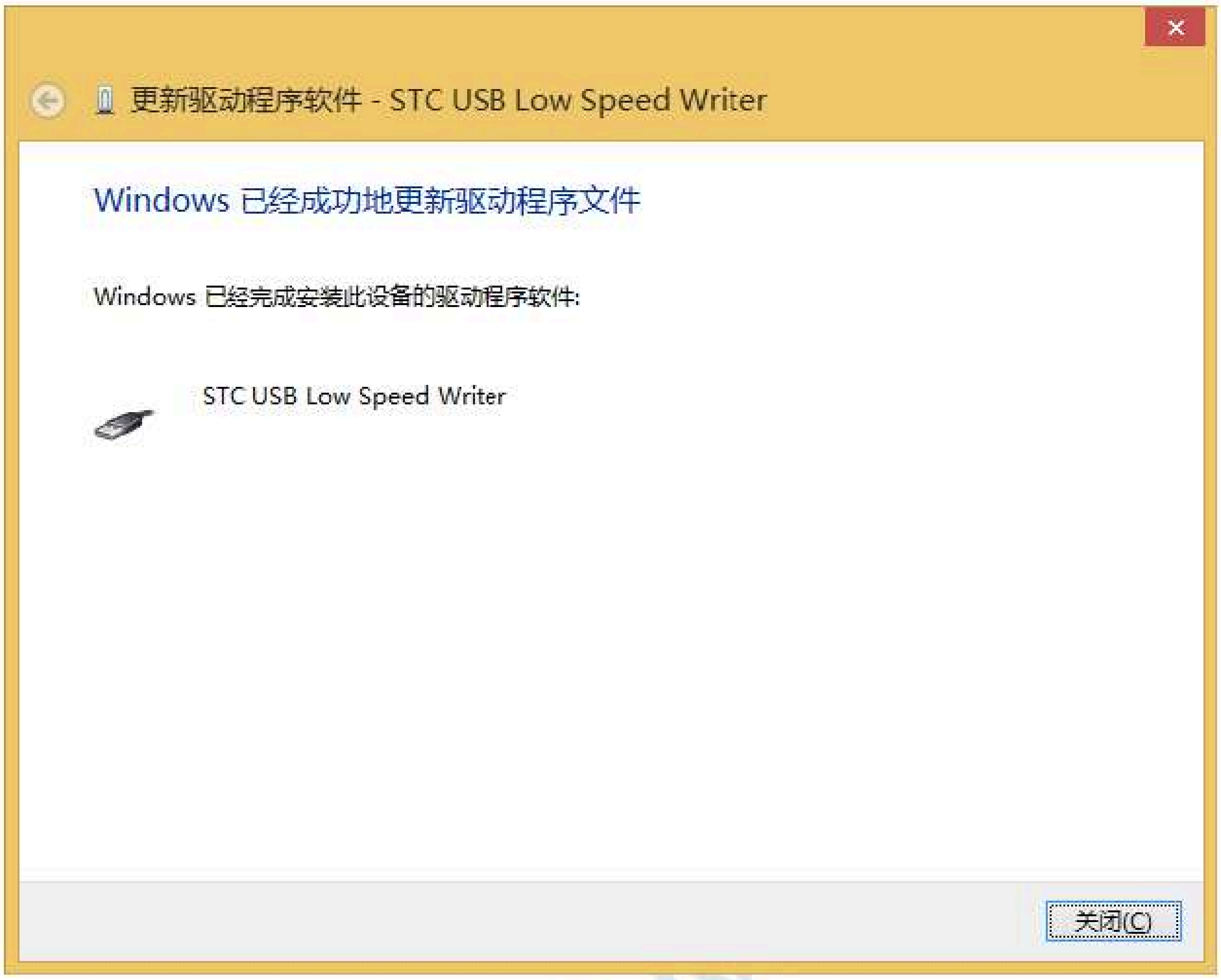
查看详细信息(D)

Next, the system will automatically install the driver, as shown in the figure below



正在安装驱动程序软件...

The following dialog box appears to indicate that the driver installation is complete

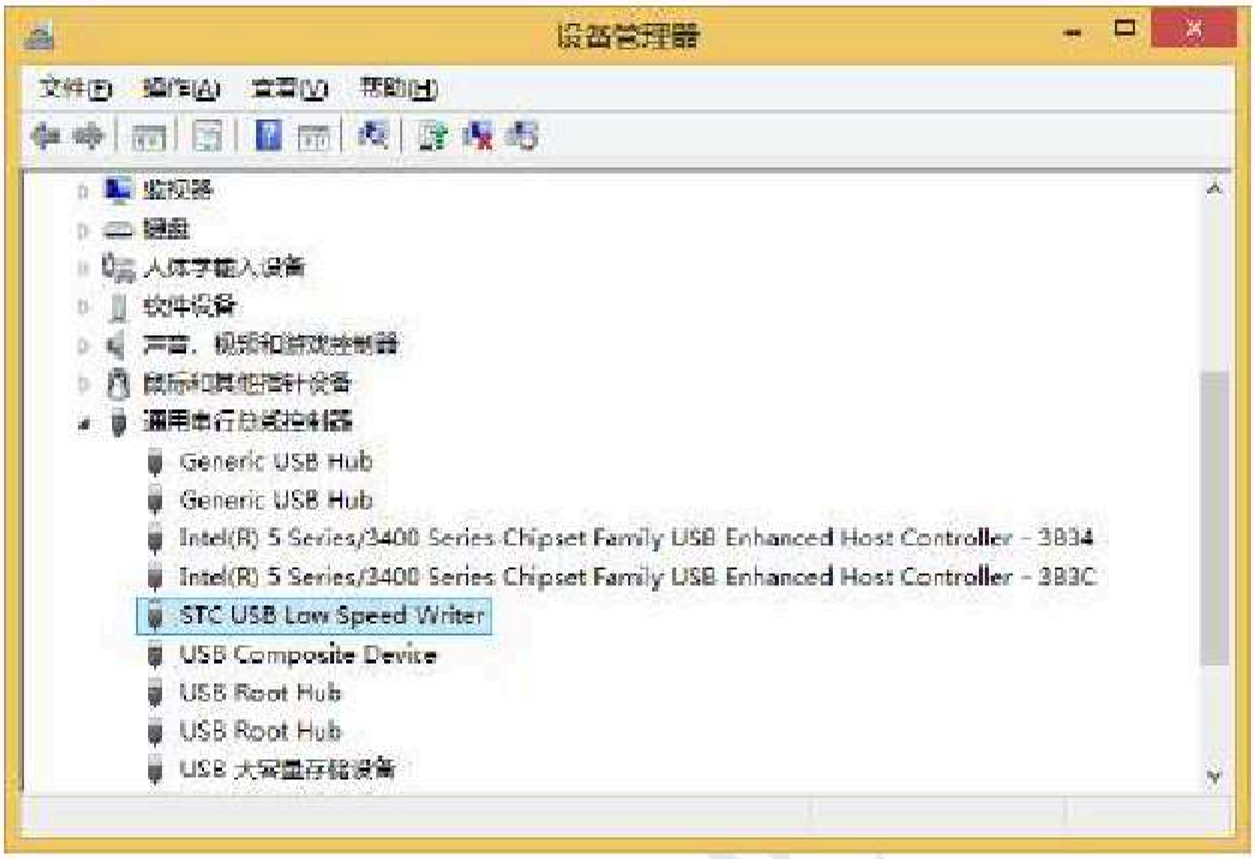

更新驱动程序软件 - STC USB Low Speed Writer

Windows 已经成功地更新驱动程序文件

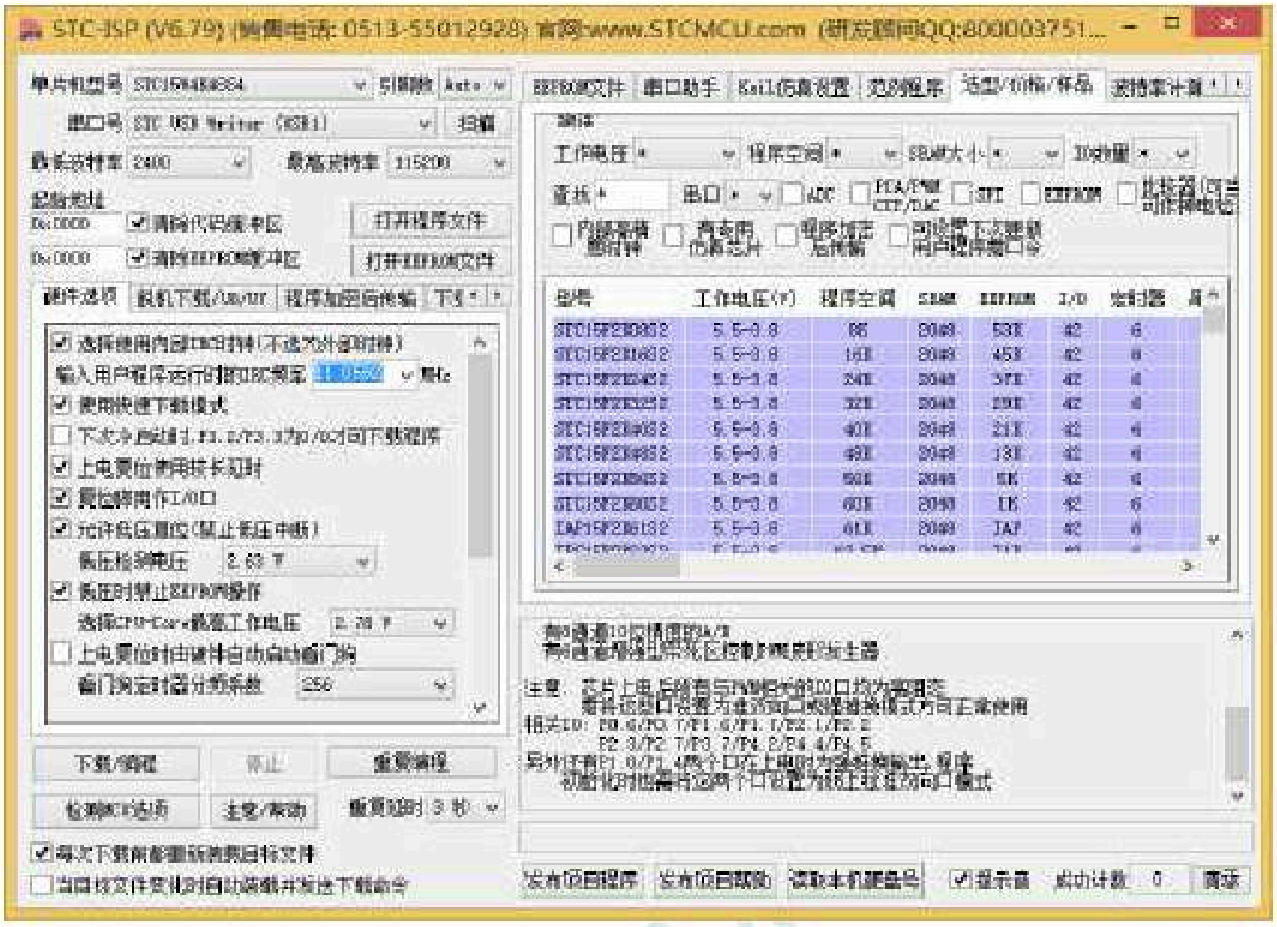
Windows 已经完成安装此设备的驱动程序软件:

STC USB Low Speed Writer

At this time, in the device manager, the device with the yellow exclamation mark before will be displayed as " Backup name" at this time. STC USB Low Speed Writer "The setting



Opened before STC-ISP The list of serial port numbers in the downloaded software USB will automatically select the inserted one Device, and display the device name as " STC USB Writer (USB1) ", as shown below :


STC-ISP (V6.79) (销售电话: 0513-55012928) 官网www.STCMCU.com (研发顾问QQ:800003751...

单片机型号 STC15W4K48S4 引脚数 Auto
串口号 STC USB Writer (USB1) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区 打开程序文件
0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | 下载 ...

☑ 选择使用内部IRC时钟(不选为外部时钟)
输入用户程序运行时的IRC频率 11.0592 MHz
☑ 使用快速下载模式
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.63 V
☑ 低压时禁止EEPROM操作
选择CPU-Core最高工作电压 2.28 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256

下载/编程 | 停止 | 重复编程
检测MCU选项 | 注意/帮助 | 重复延时 3 秒
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

EEPROM文件 | 串口助手 | Keil仿真设置 | 范例程序 | 选型/价格/样品 | 波特率计算器

筛选
工作电压 * 程序空间 * SRAM大小 * I/O数量 *
查找 * 串口 * ☐ ADC ☐ PCA/PWM CCP/DAC ☐ SPI ☐ EEPROM ☐ 比较器(可当ADC使用)
☐ 内部高精度时钟 ☐ 内部复位 ☐ 程序加密后传输 ☐ 可设置下次更新用户程序口令

型号	工作电压(V)	程序空间	SRAM	EEPROM	I/O	定时器
STC15F2K08S2	5.5-3.8	8K	2048	53K	42	6
STC15F2K16S2	5.5-3.8	16K	2048	45K	42	6
STC15F2K24S2	5.5-3.8	24K	2048	37K	42	6
STC15F2K32S2	5.5-3.8	32K	2048	29K	42	6
STC15F2K40S2	5.5-3.8	40K	2048	21K	42	6
STC15F2K48S2	5.5-3.8	48K	2048	13K	42	6
STC15F2K56S2	5.5-3.8	56K	2048	5K	42	6
STC15F2K60S2	5.5-3.8	60K	2048	1K	42	6
IAP15F2K61S2	5.5-3.8	61K	2048	IAP	42	6

[illegible]
注意: 芯片上电后所有与PWM相关的IO口均为高阻态,
需将这些口设置为准双向口或强推挽模式方可正常使用
相关IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
另外还有P1.0/P1.4两个口在上电时为强推挽输出,程序
初始化时需将这两个口设置为准双向口或高阻口模式

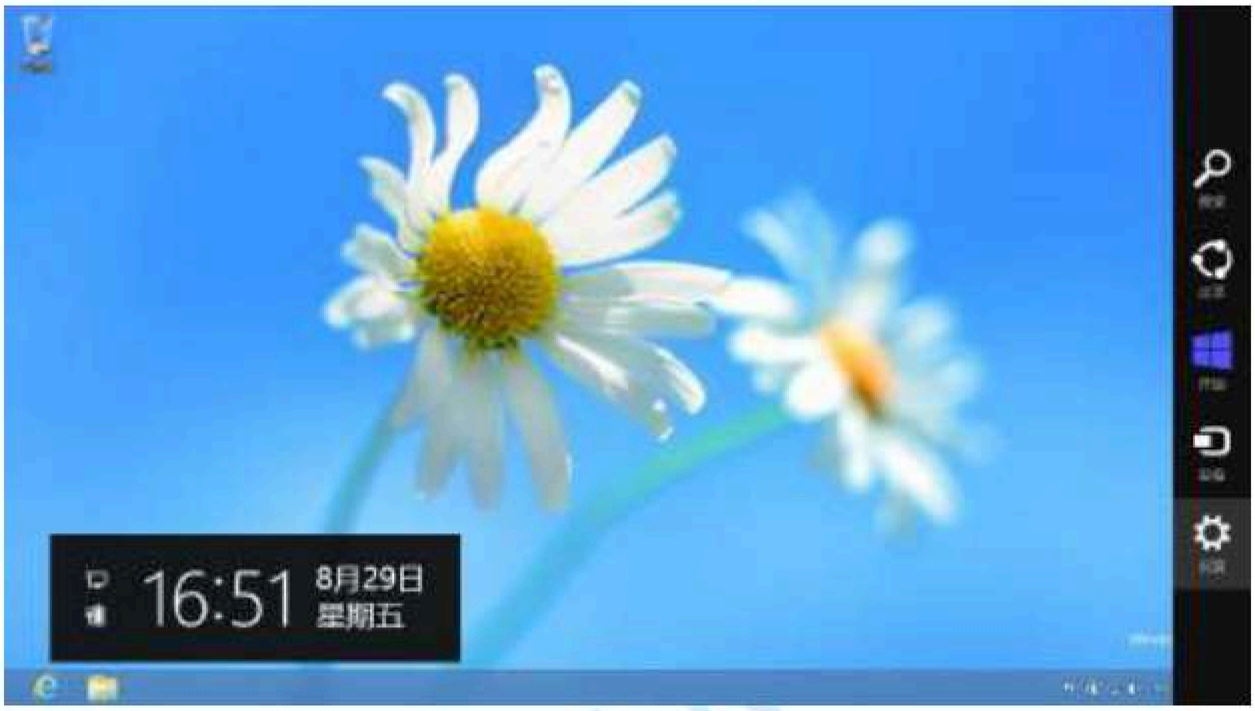
发布项目程序 | 发布项目帮助 | 读取本机硬盘号 | ☑ 提示音 | 成功计数 0 | 清零

Windows 8 (64-bit) installation method

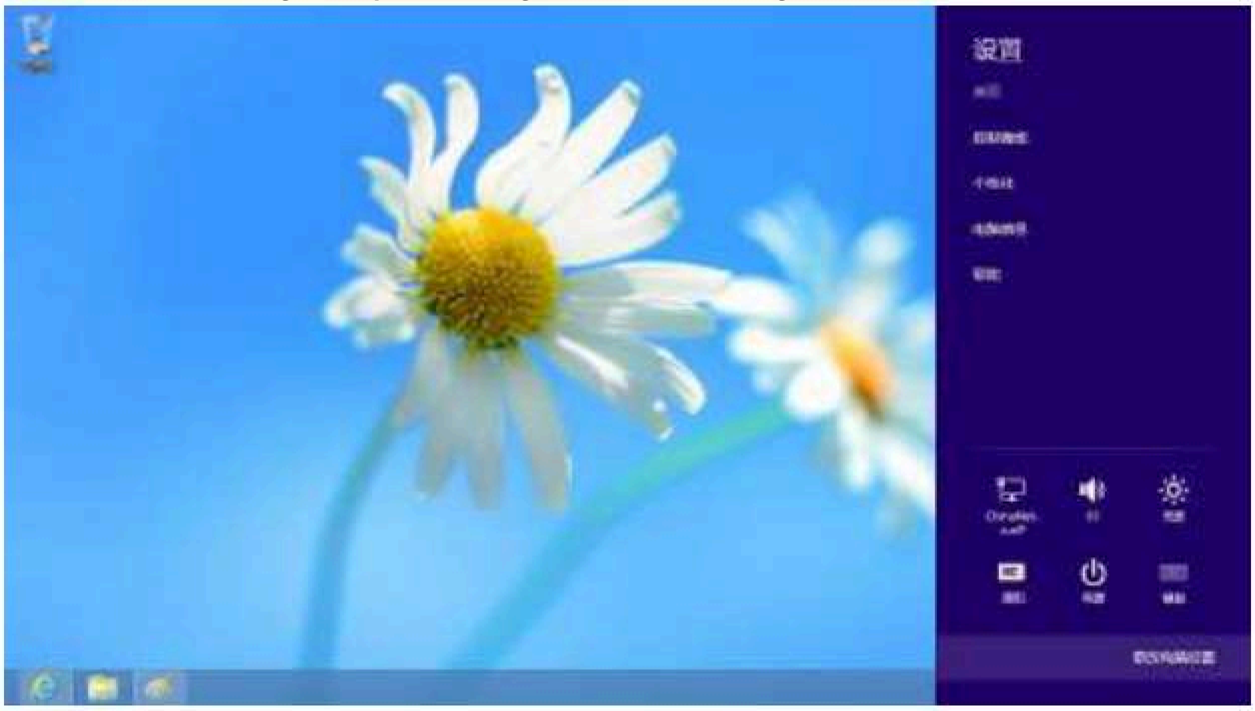
Due to Windows8 64 Under the default state of the bit operating system, drivers that are not digitally signed cannot be installed successfully

install STC-USB , before driving, you need to follow the steps below to temporarily skip the digital signature, and the installation will be suc

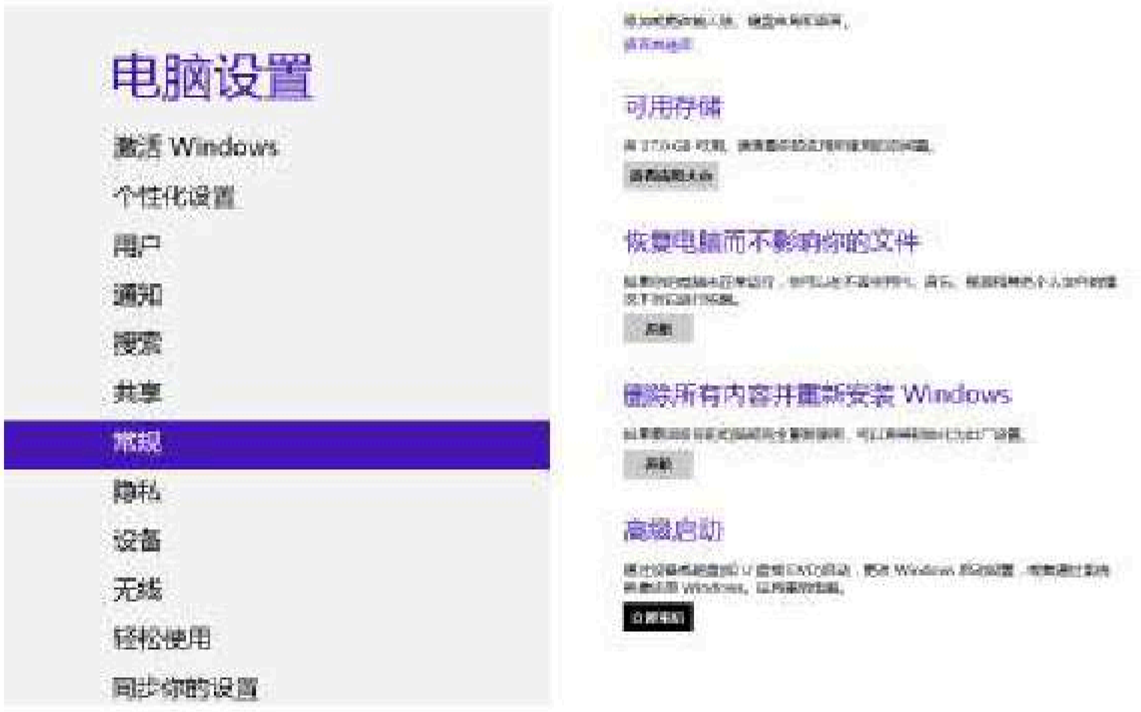
First move the mouse to the lower right corner of the screen and select the "Settings" button



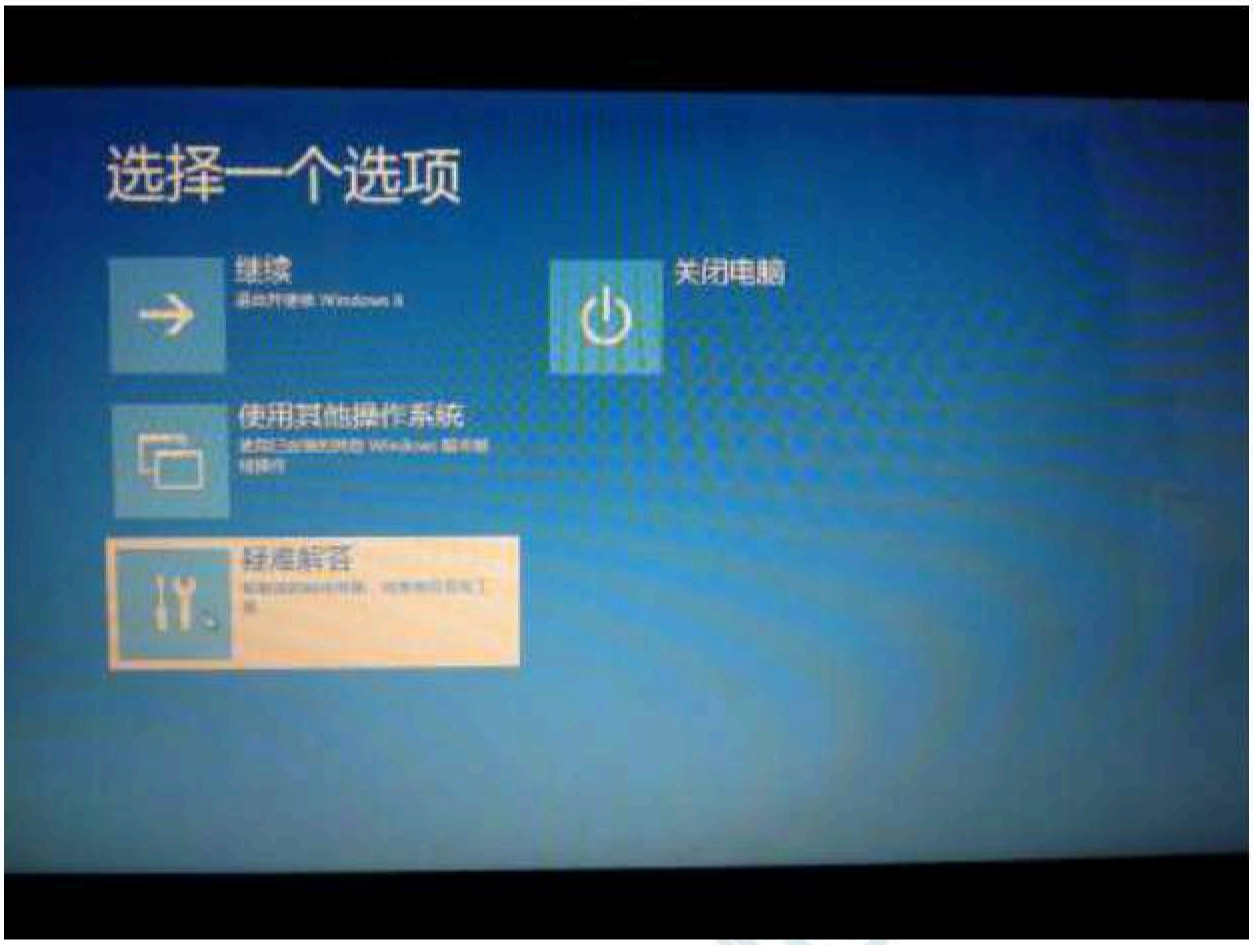
Then select the "Change Computer Settings" item in the settings interface



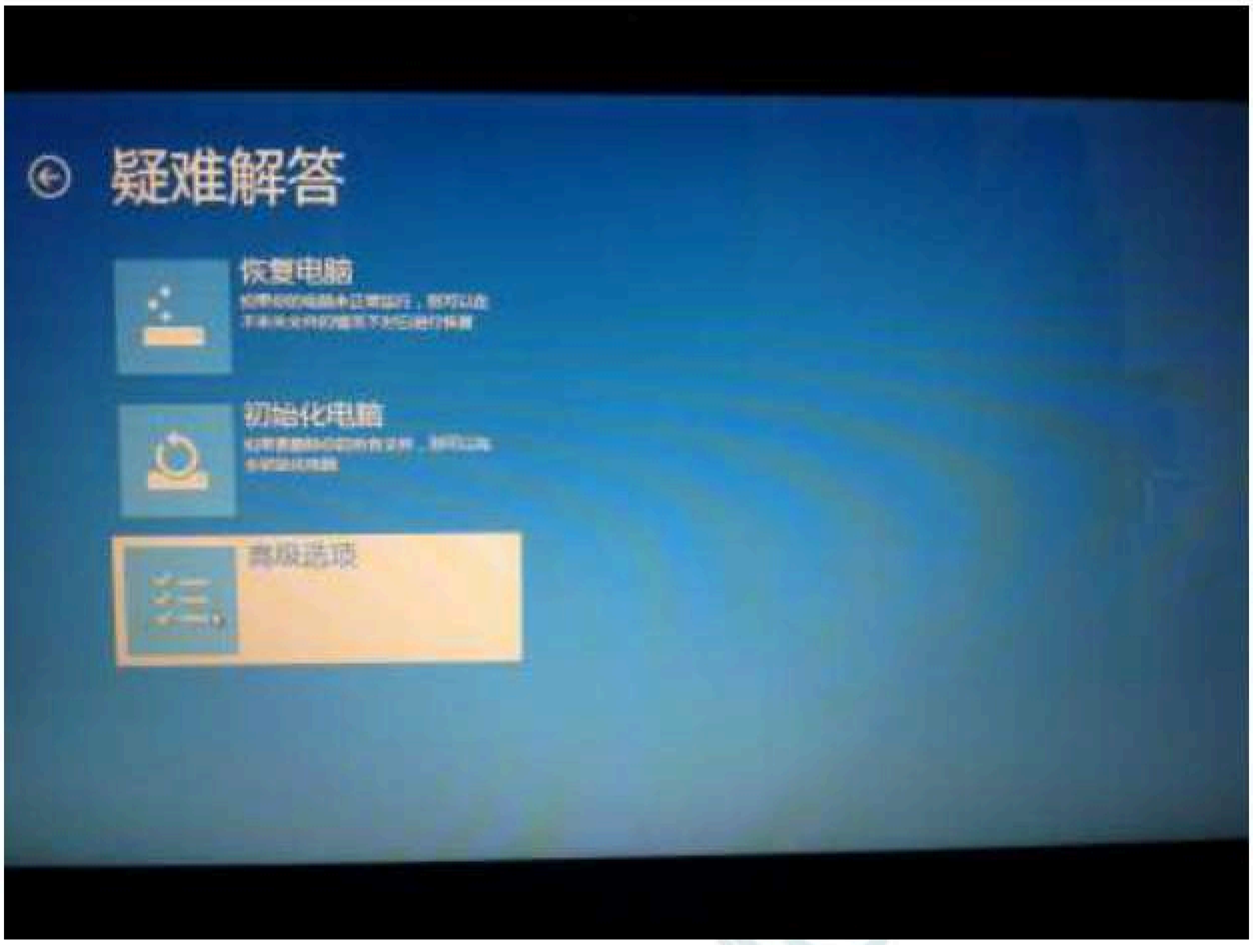
In the computer settings, select the "Start Now" button under the "Advanced Startup" item in the "General" property page.



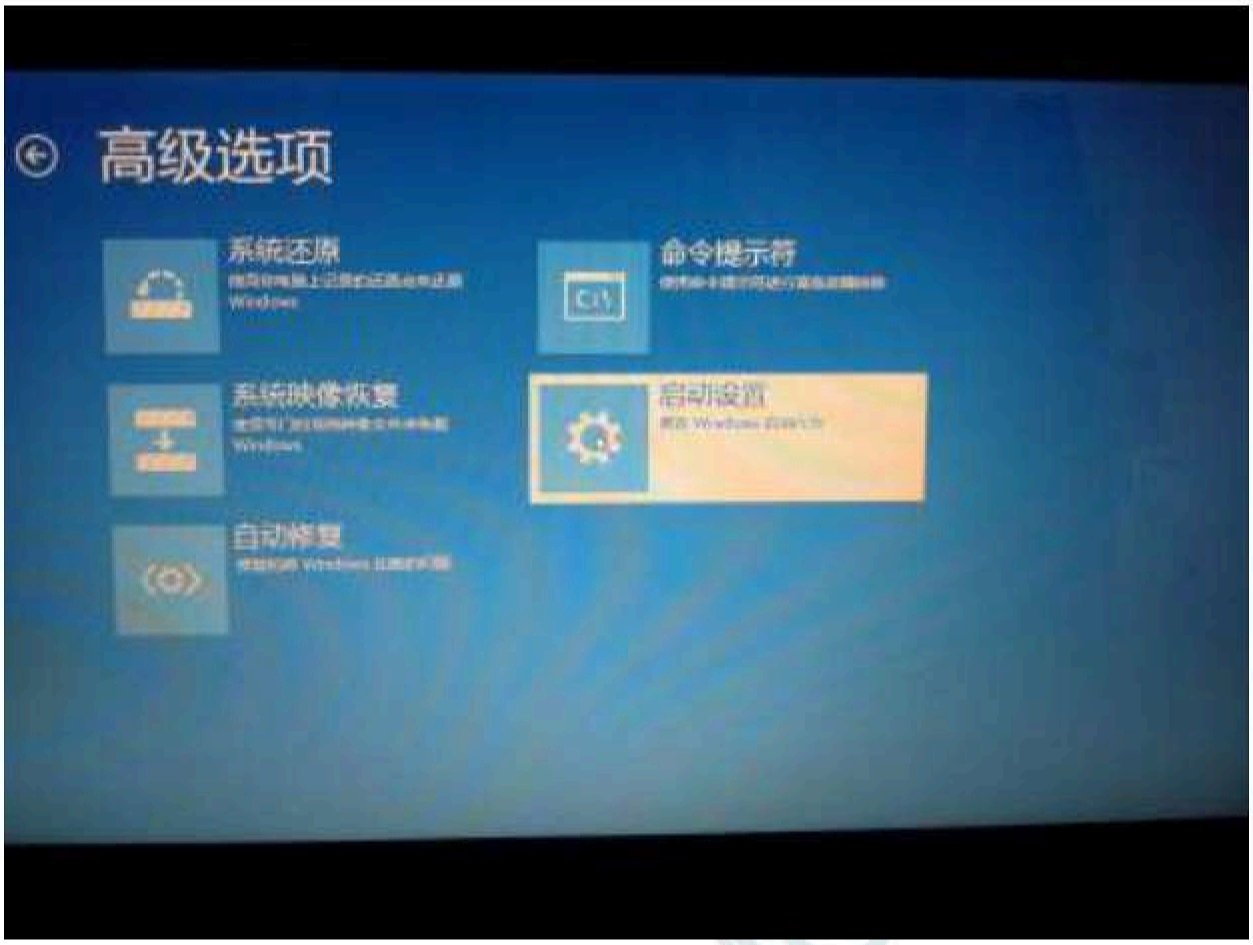
In the interface below, select the "Troubleshooting" item



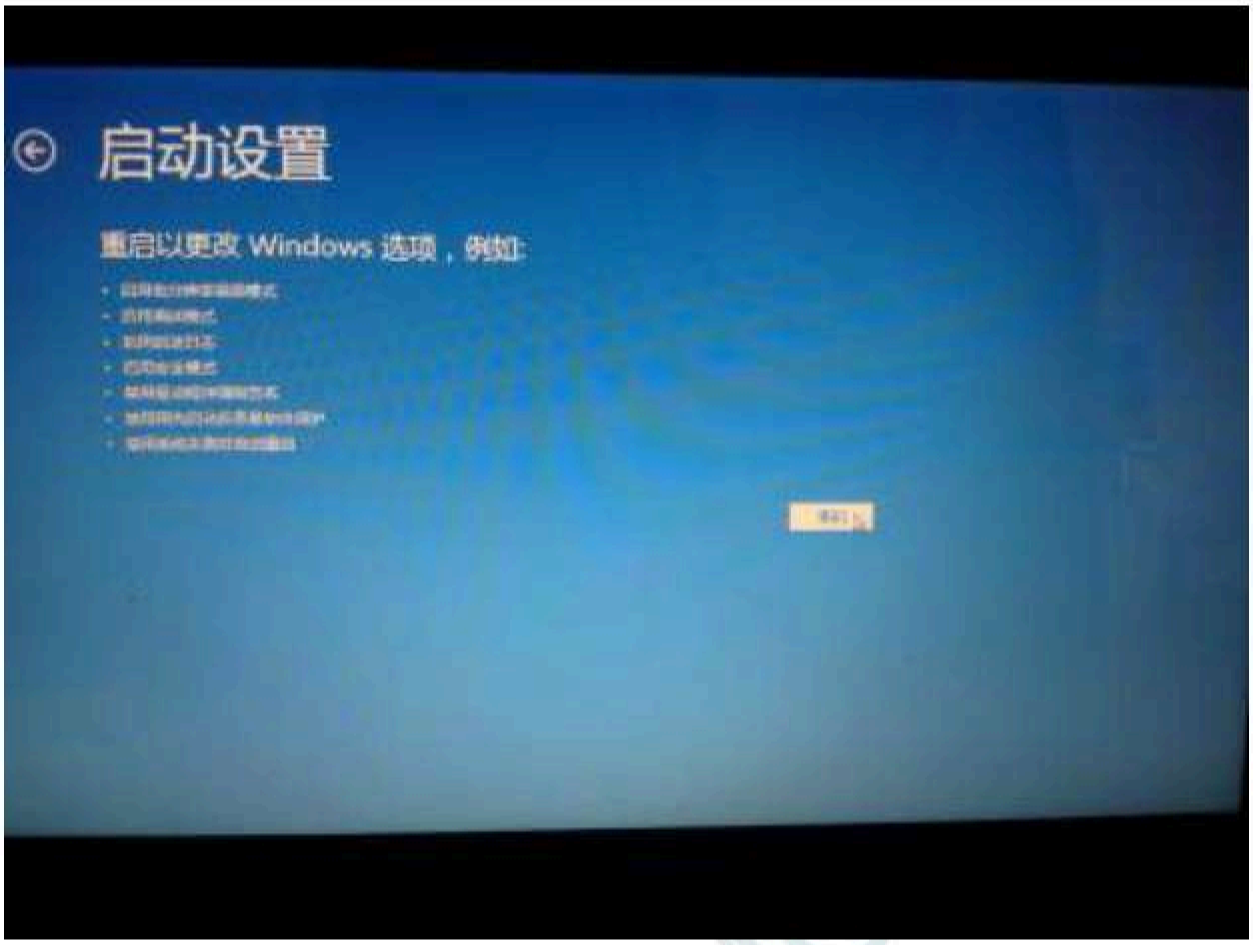
Then select "Advanced Options" in "Troubleshooting"



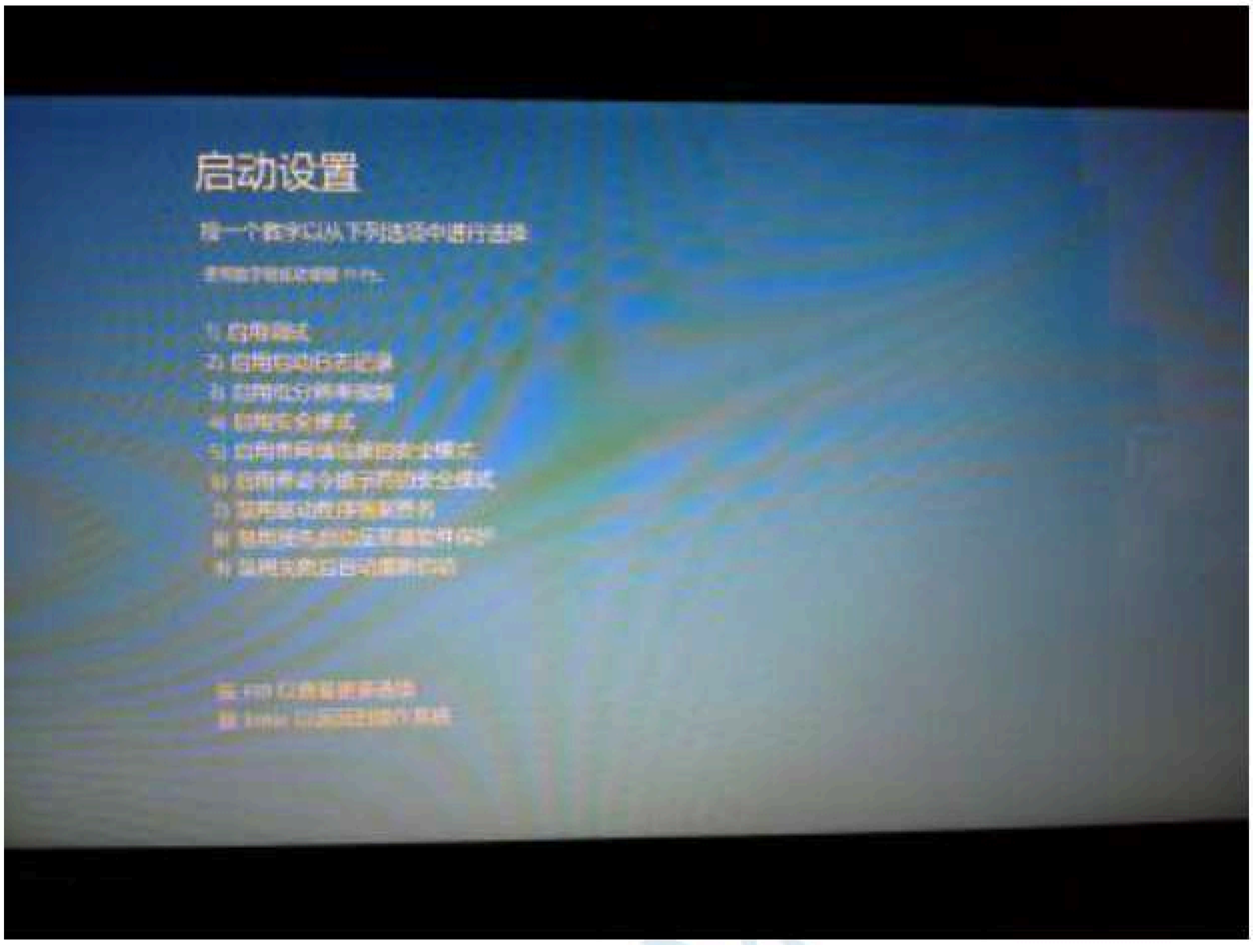
In the "Advanced Options" interface below, select "Startup Settings"



In the "Startup Settings" interface below, click the "Restart" button to restart the computer



After the computer restarts, it will automatically enter the "Startup settings" interface shown in the figure below, choose 7 press the number key "" or press the function key" to select "Disable driver forced signature" to start



启动设置

按一个数字以从下列选项中进行选择

使用数字键或功能键 F1-F9。

1) 启用调试
2) 启用启动日志记录
3) 启用低分辨率视频
4) 启用安全模式
5) 启用带网络连接的安全模式
6) 启用带命令提示符的安全模式
7) 禁用驱动程序强制签名
8) 禁用预先启动反恶意软件保护
9) 禁用失败后自动重新启动

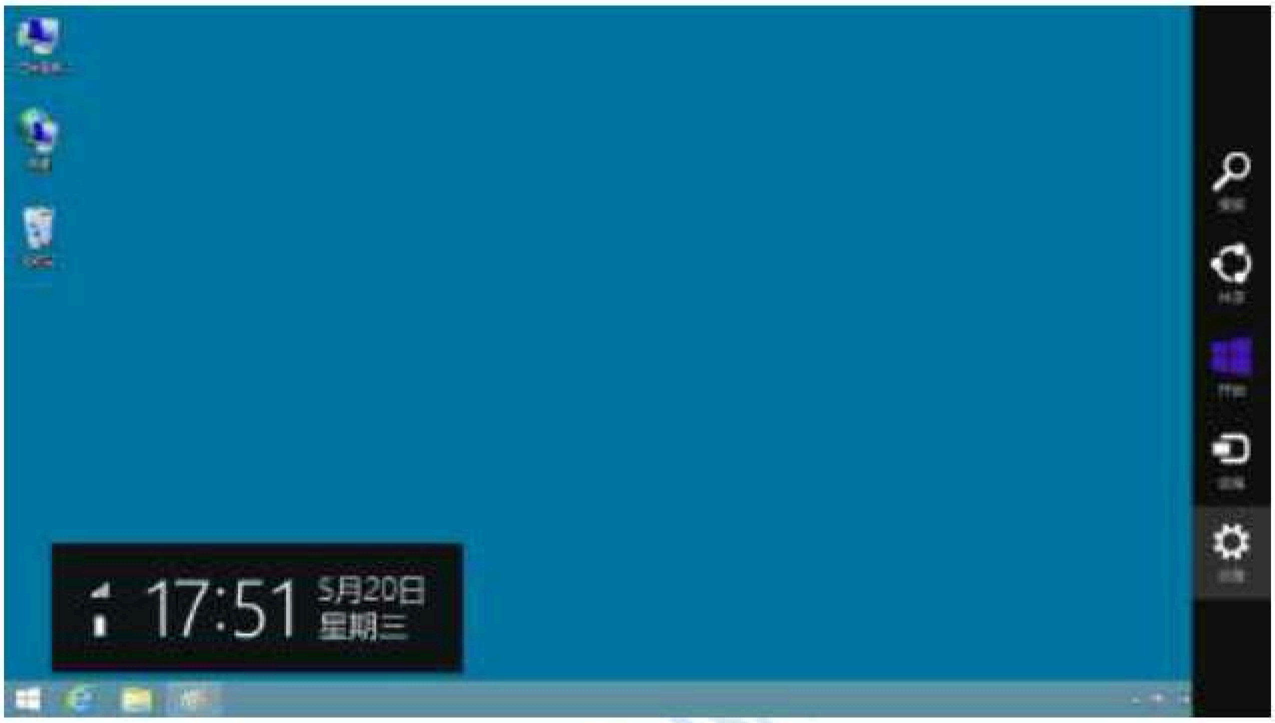
按 F10 以查看更多选项
按 Enter 以返回到操作系统

Boot to Windows 8 After that, follow Windows 8 (32 Bit) installation method You can complete the installation of the driver

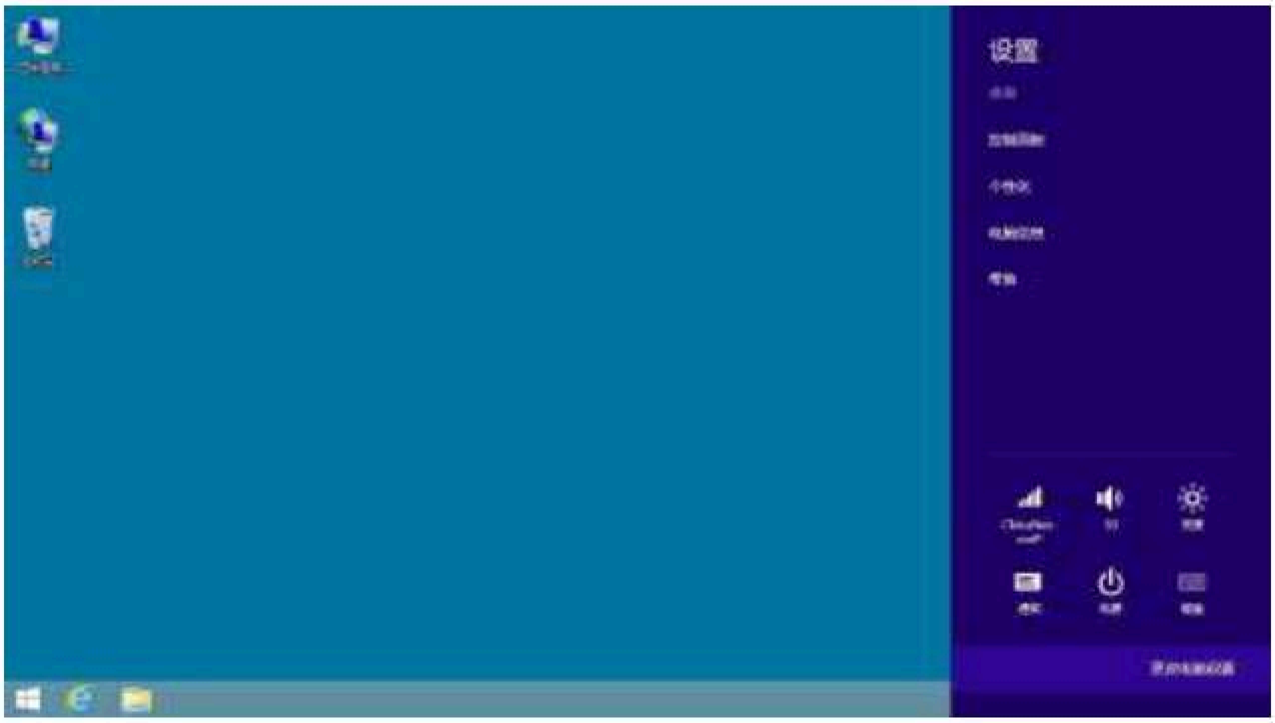
Windows 8.1 (64-bit) installation method

Windows 8.1 with Windows 8 The method of entering the advanced startup menu is different and will be explained here specifically.

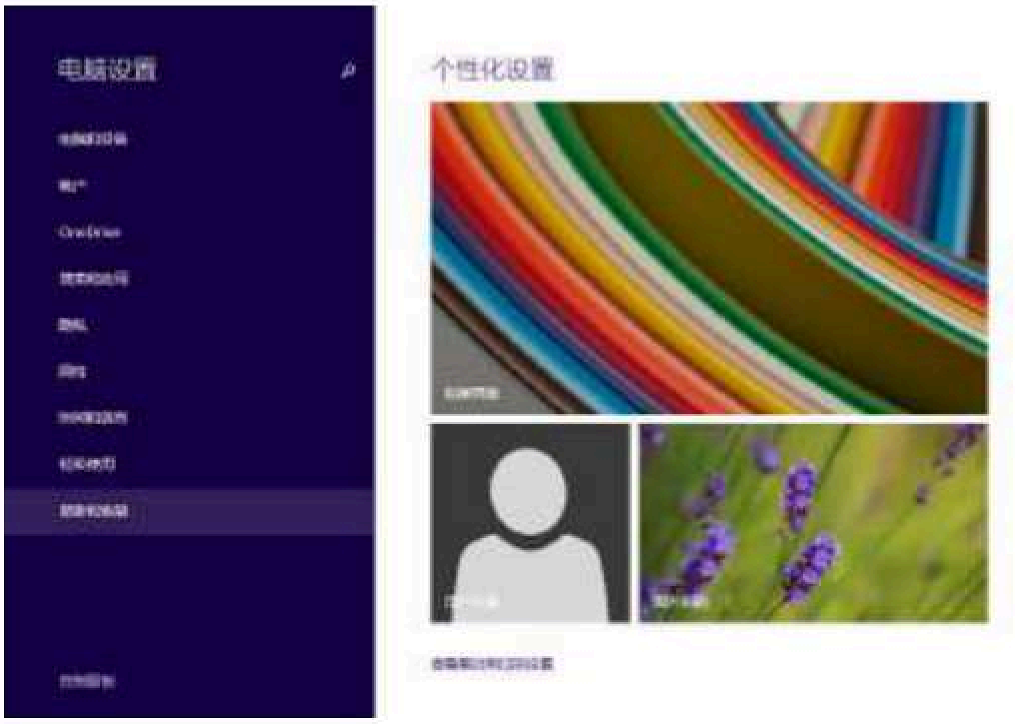
First move the mouse to the lower right corner of the screen and select the "Settings" button



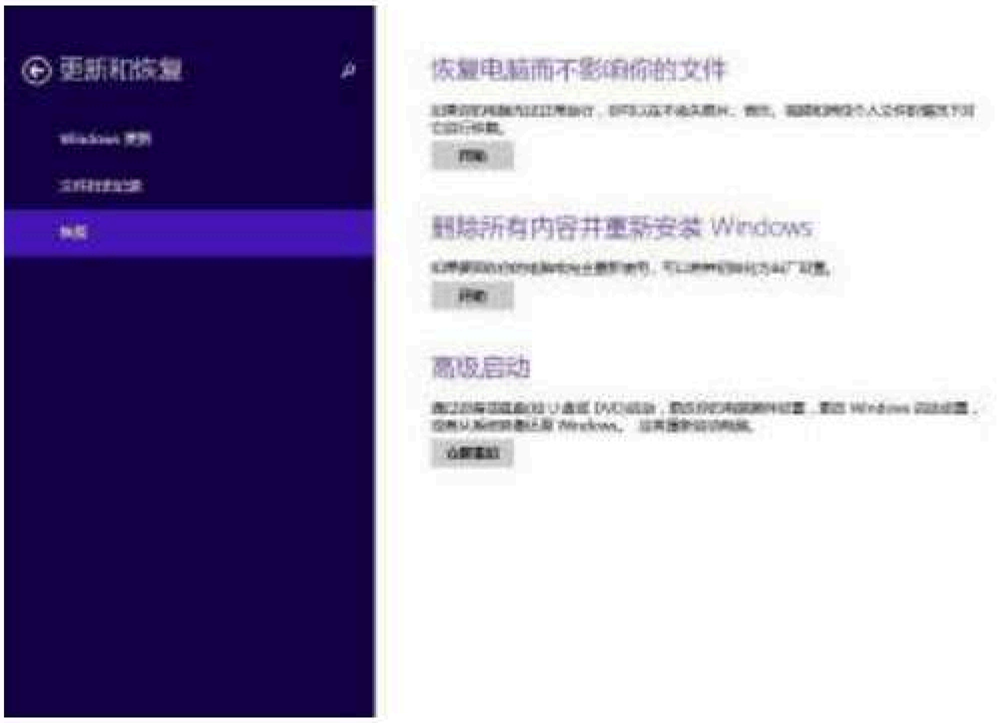
Then select the "Change Computer Settings" item in the settings interface



In the computer settings, select "Update and Restore" (Hard Windows 8 different , Windows 8 The choice is "regular")

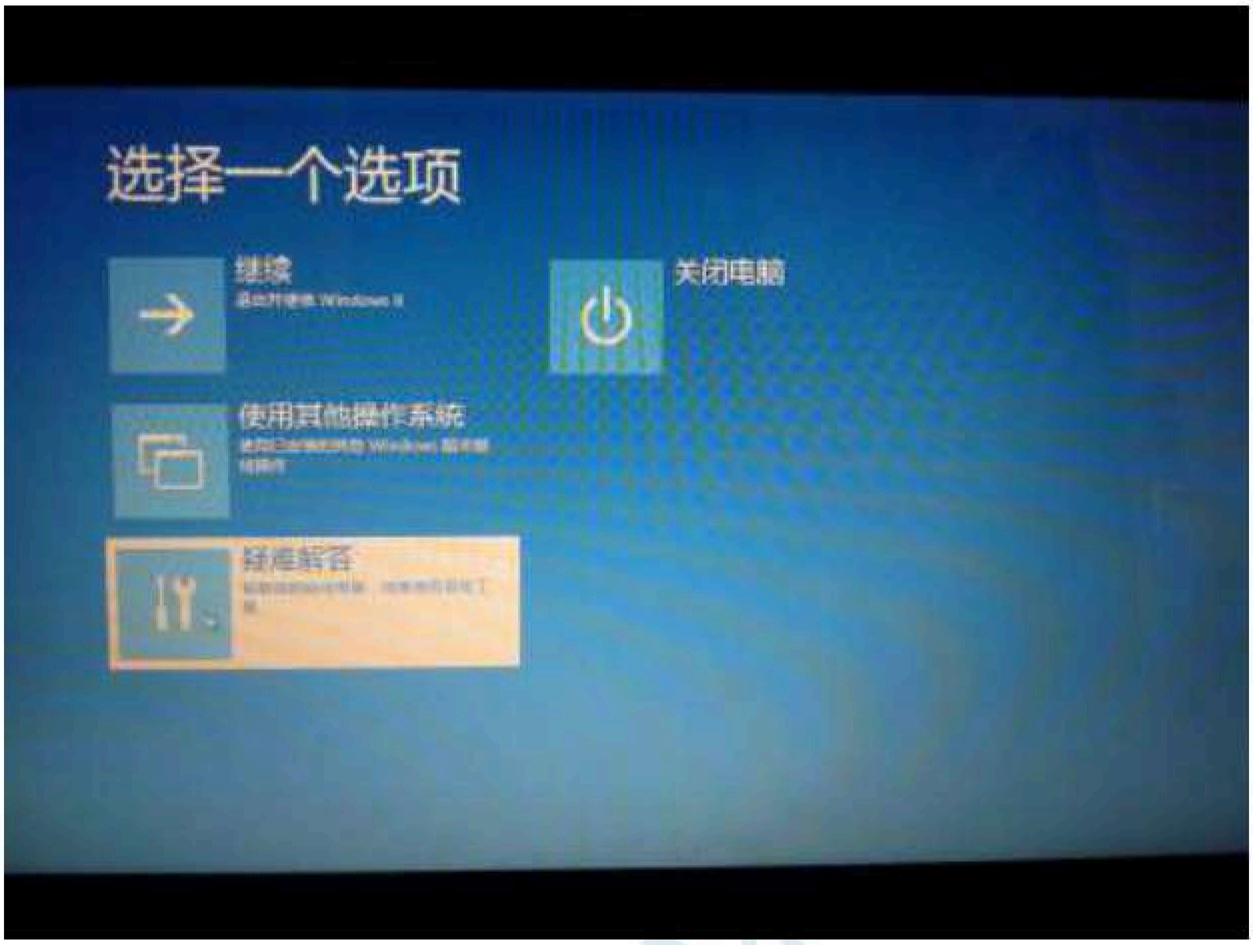


Select the "Recovery" property page in the Update and Recovery page, and click the "Start Now" button under the "Advanced Star

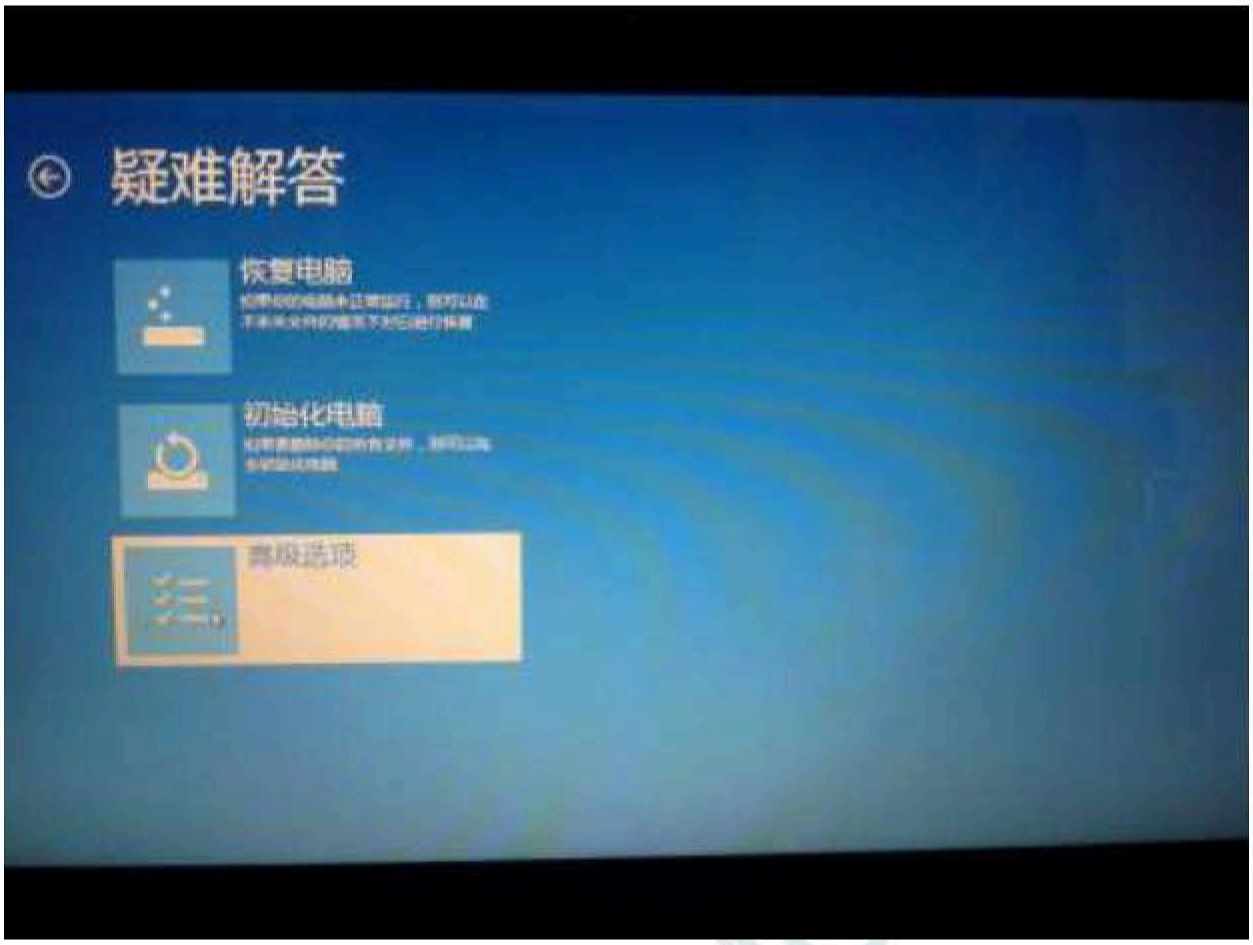


The next operation is related to Windows 8 The steps are the same

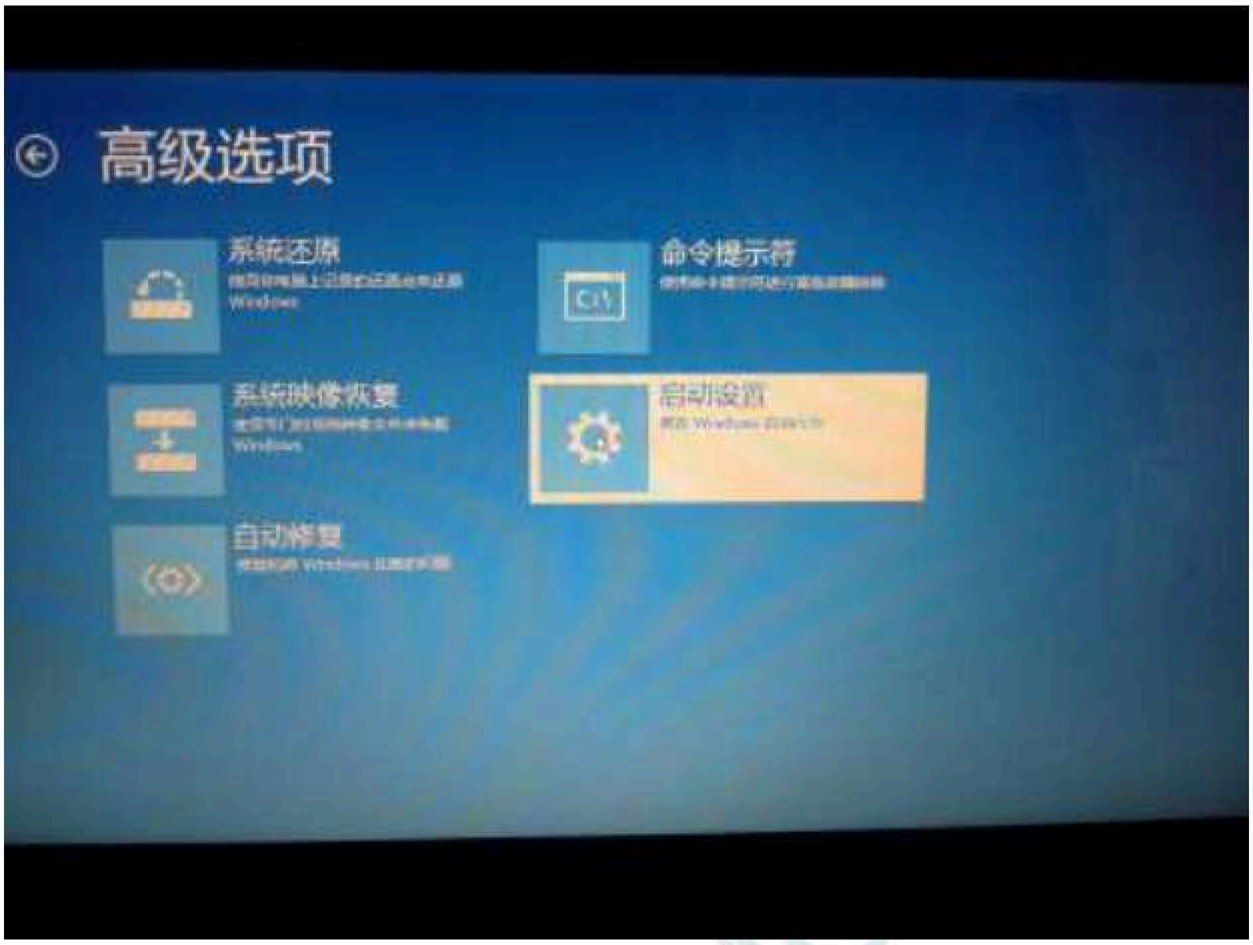
In the interface below, select the "Troubleshooting" item



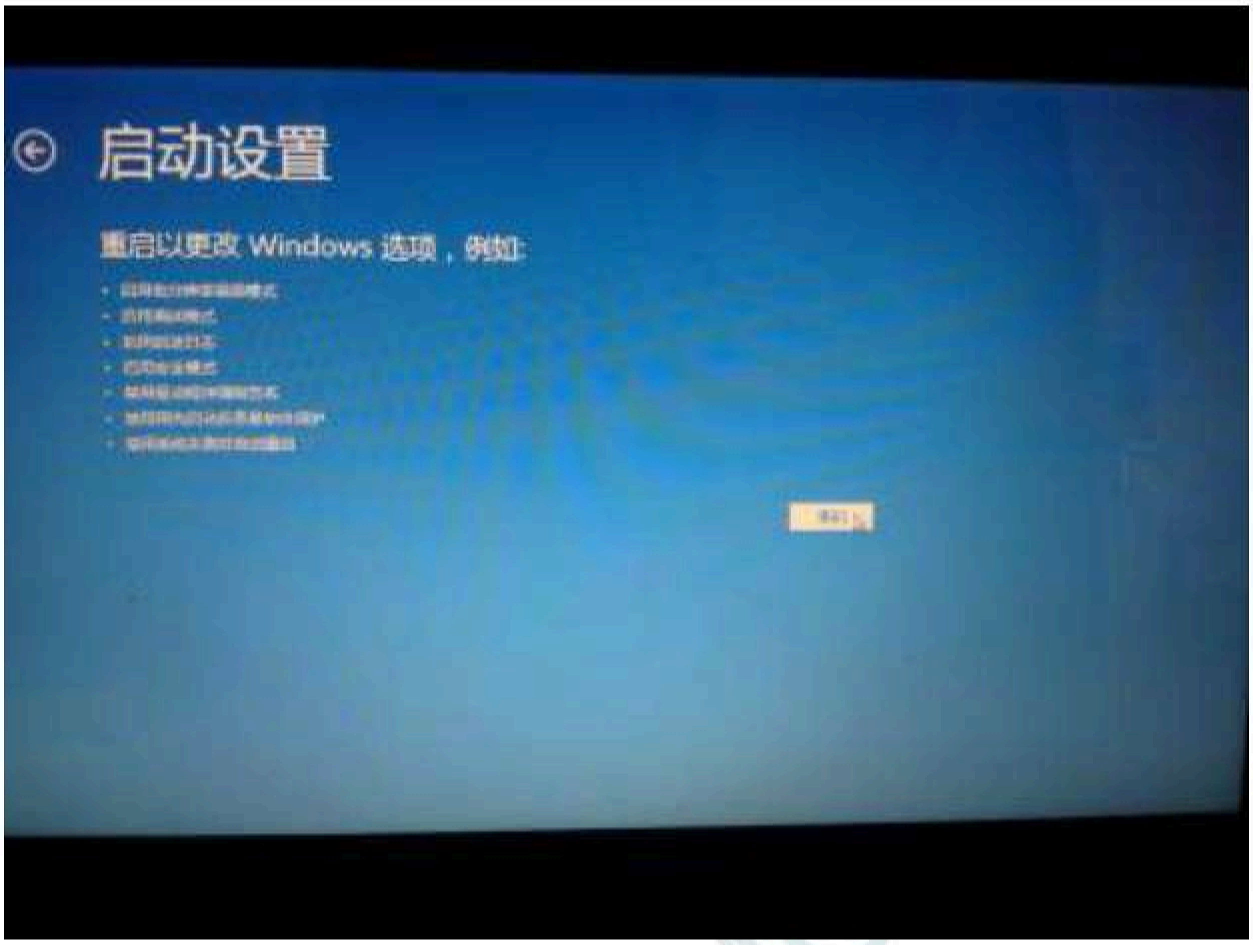
Then select "Advanced Options" in "Troubleshooting"



In the "Advanced Options" interface below, select "Startup Settings"



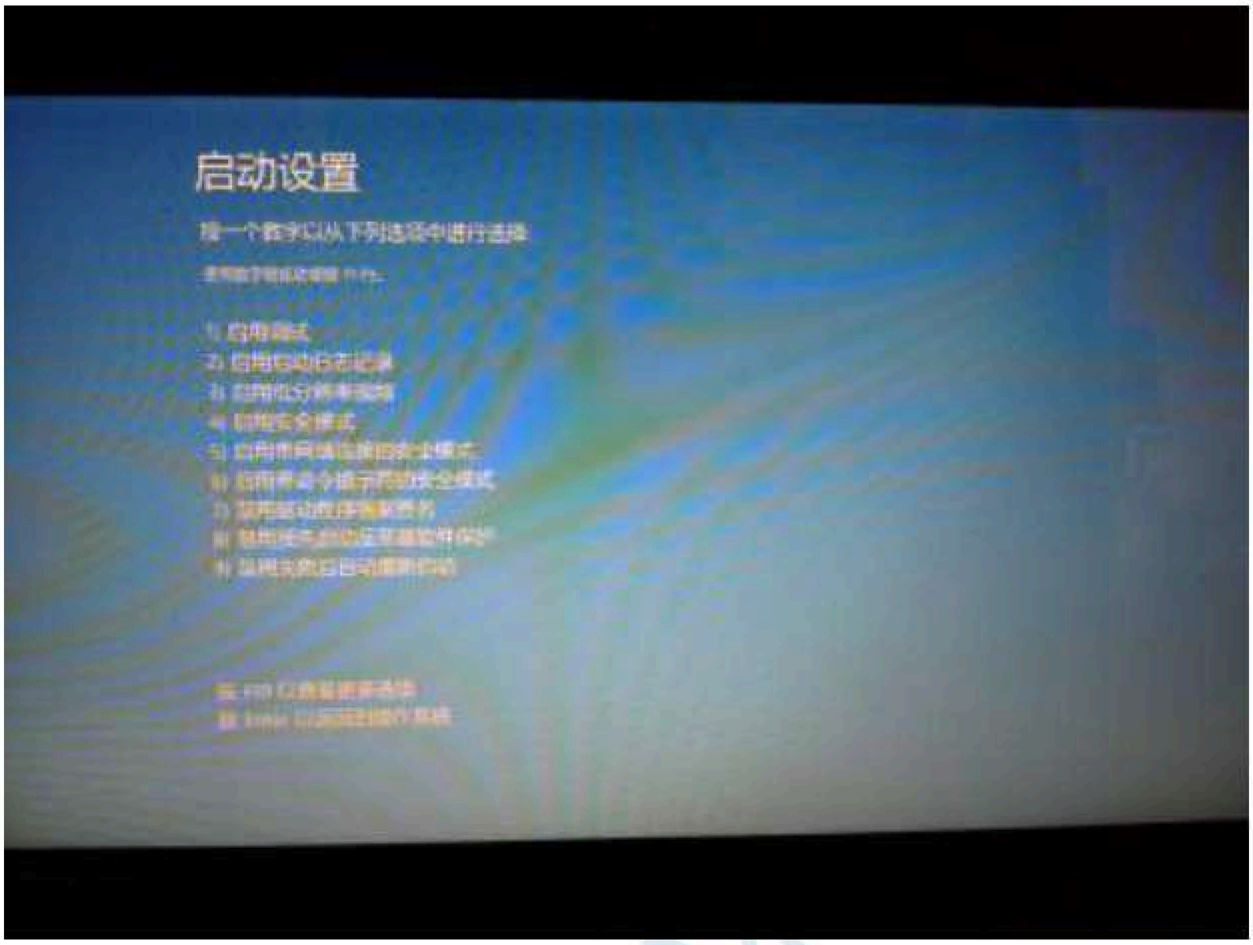
In the "Startup Settings" interface below, click the "Restart" button to restart the computer


启动设置

重启以更改 Windows 选项，例如:

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

After the computer restarts, it will automatically enter the "Startup settings" interface shown in the figure below, 7 "choose press the number key "" or press the function key" to select "Disable driver forced signature" to start


启动设置

按一个数字以从下列选项中进行选择:

使用数字键或功能键 F1-F9。

1) 启用调试
2) 启用启动日志记录
3) 启用低分辨率视频
4) 启用安全模式
5) 启用带网络连接的安全模式
6) 启用带命令提示符的安全模式
7) 禁用驱动程序强制签名
8) 禁用预先启动反恶意软件保护
9) 禁用失败后自动重新启动

按 F10 以查看更多选项
按 Enter 以返回到操作系统

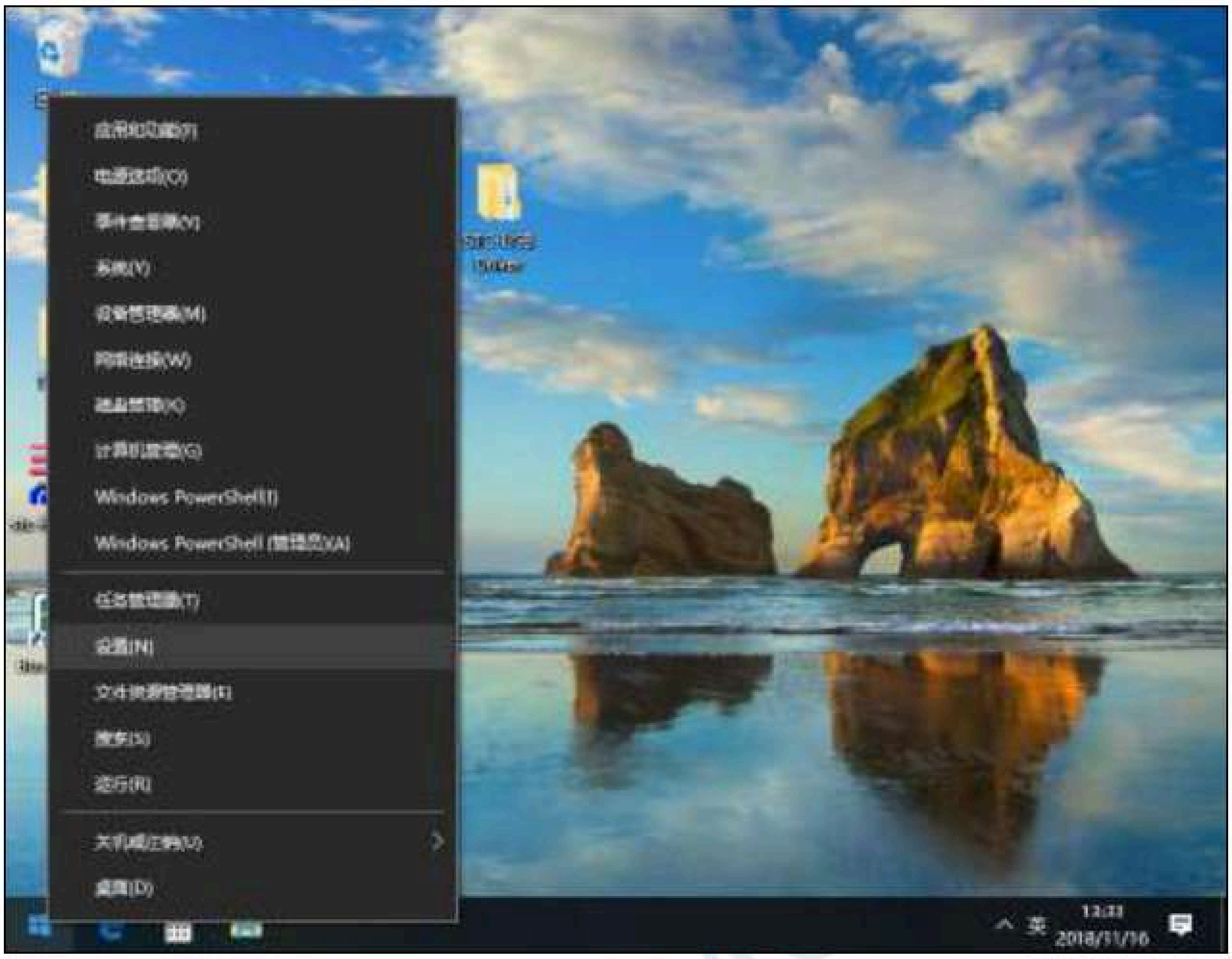
Boot to Windows 8 After that, follow Windows 8 (32 Bit) installation method You can complete the installation of the driver

Windows 10 (64-bit) installation method

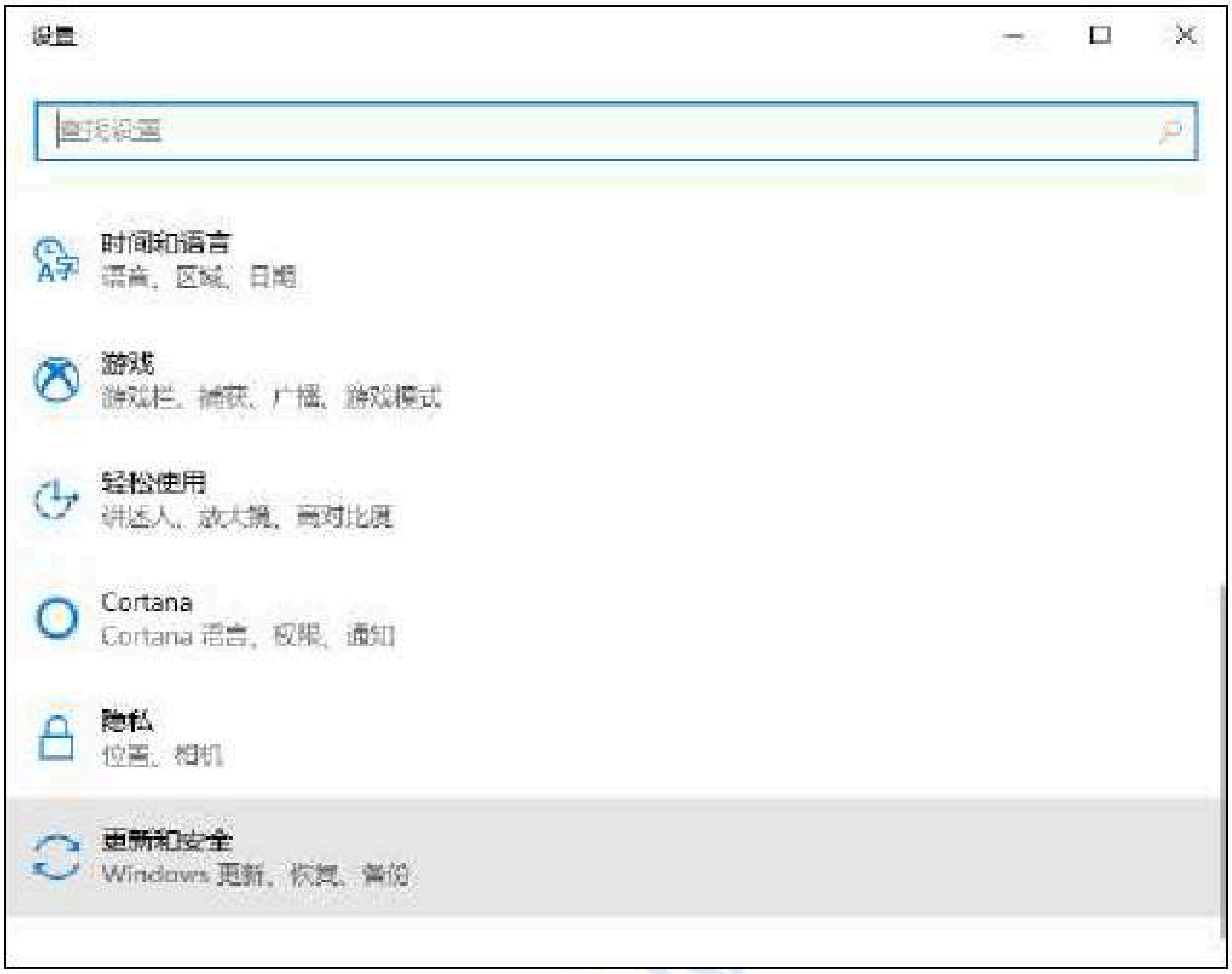
Due to Windows10 64 Under the default state of the bit operating system, drivers that are not digitally signed cannot be installed successfu
Installing STC-USB , before driving, you need to follow the steps below to temporarily skip the digital signature, and the installation will be s

Before installing the driver STC Need to Download from the official website STC-ISP the software archive." "Unzip the folder to hard STC-USB Driver be from the disk. Will have USB The chip for the download function is ready, but do not connect to the computer first

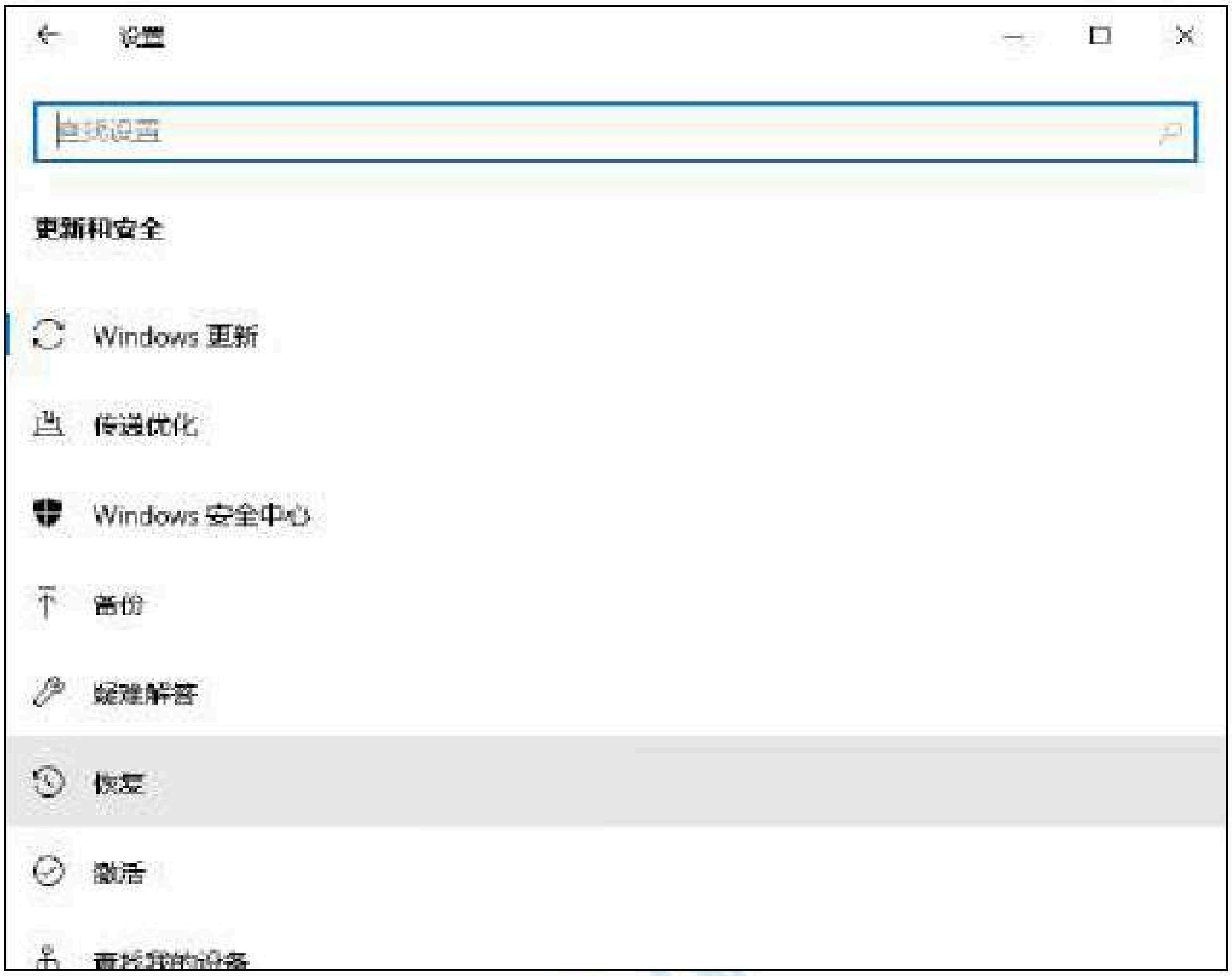
Right-click on the "Start" menu and select the "Settings" option


应用和功能(F)
电源选项(O)
事件查看器(V)
系统(Y)
设备管理器(M)
网络连接(W)
磁盘管理(K)
计算机管理(G)
Windows PowerShell(I)
Windows PowerShell (管理员)(A)
任务管理器(T)
设置(N)
文件资源管理器(E)
搜索(S)
运行(R)
关机或注销(U)
桌面(D)
英
13:33
2018/11/16

Then select the "Update and Security" item in the settings interface



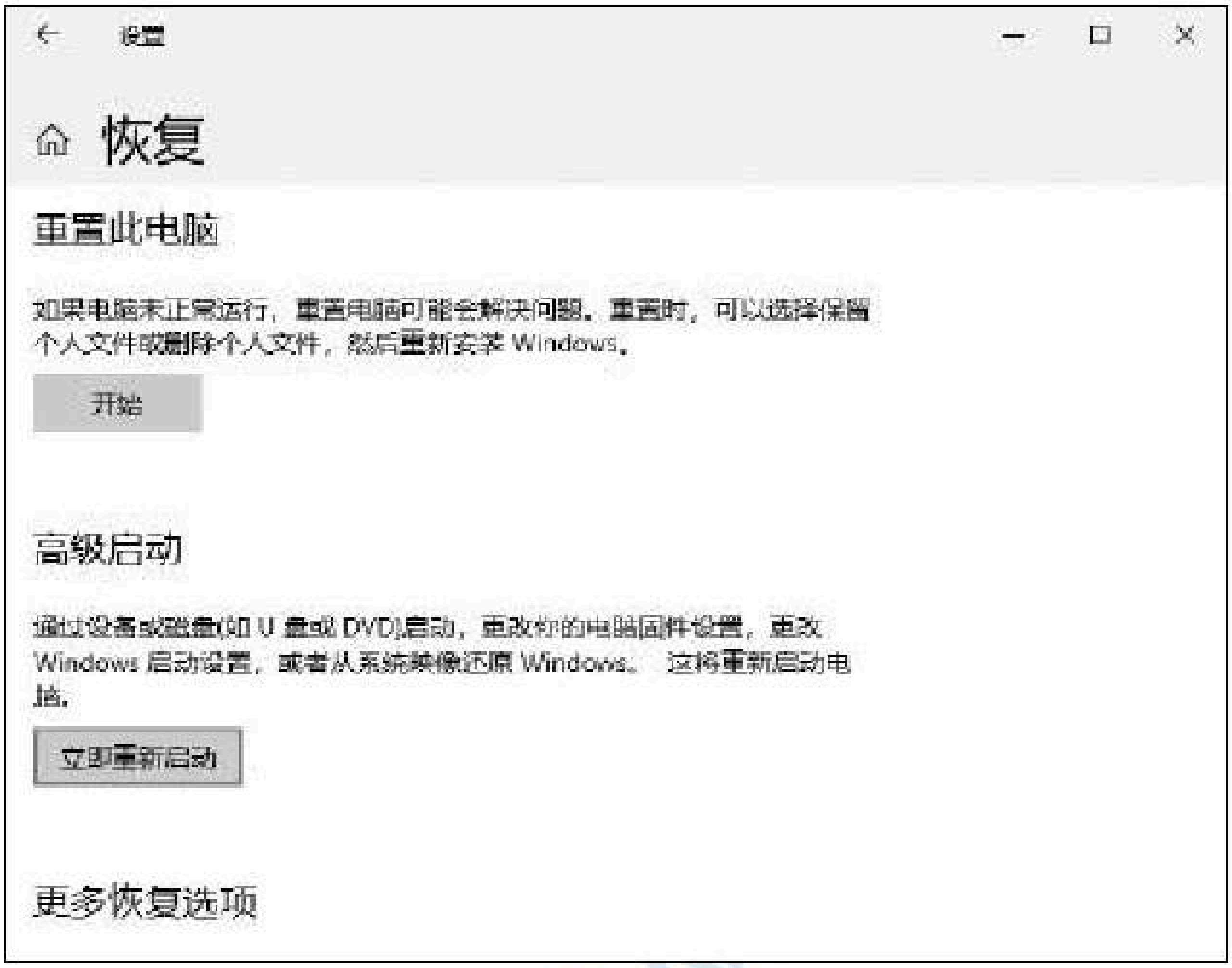
Then select the "Restore" item in the settings interface


设置

查找设置

更新和安全

In the recovery interface, click the "Restart Now" button in the "Advanced Startup" item


设置

恢复

重置此电脑

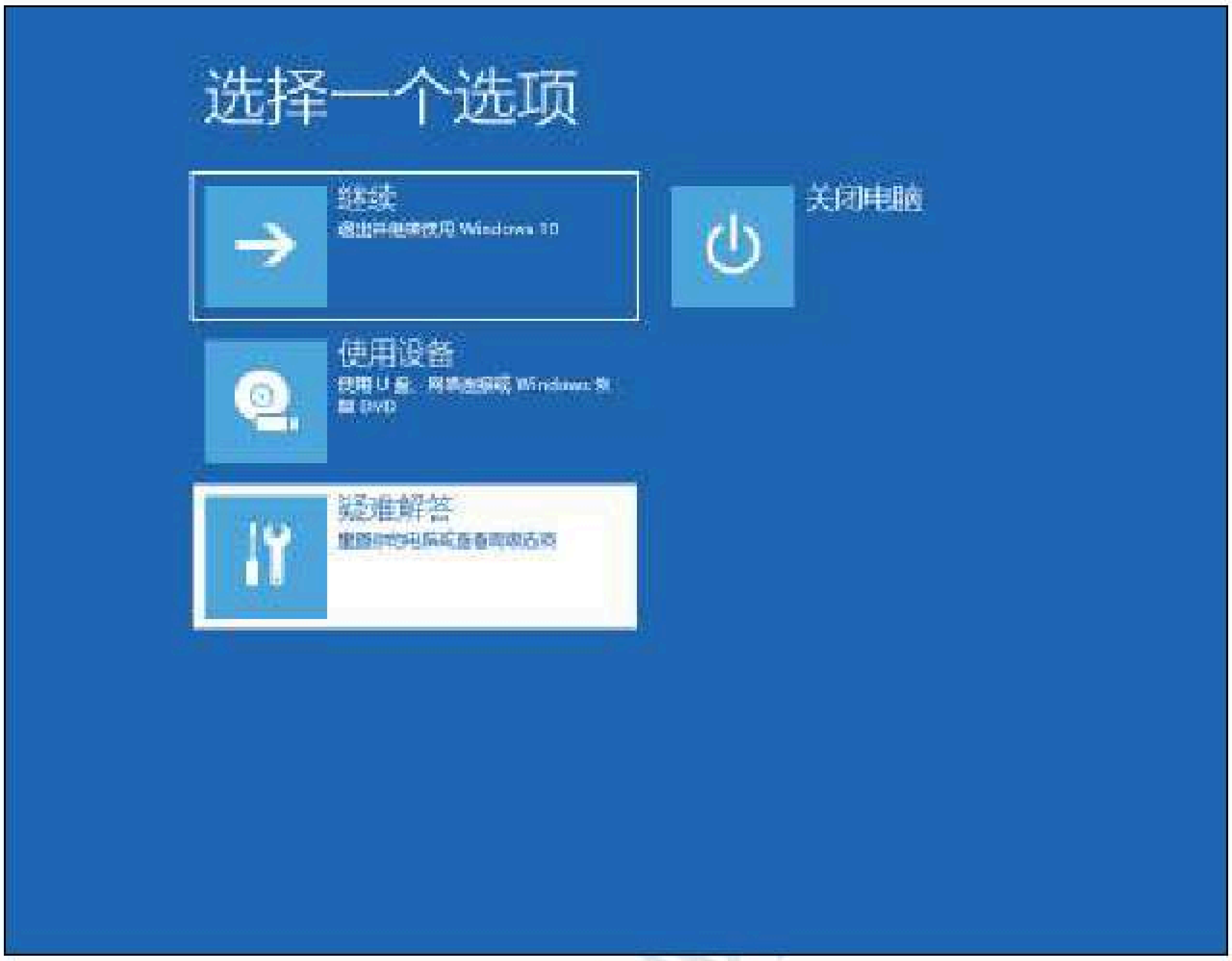
如果电脑未正常运行，重置电脑可能会解决问题。重置时，可以选择保留个人文件或删除个人文件，然后重新安装 Windows。

高级启动

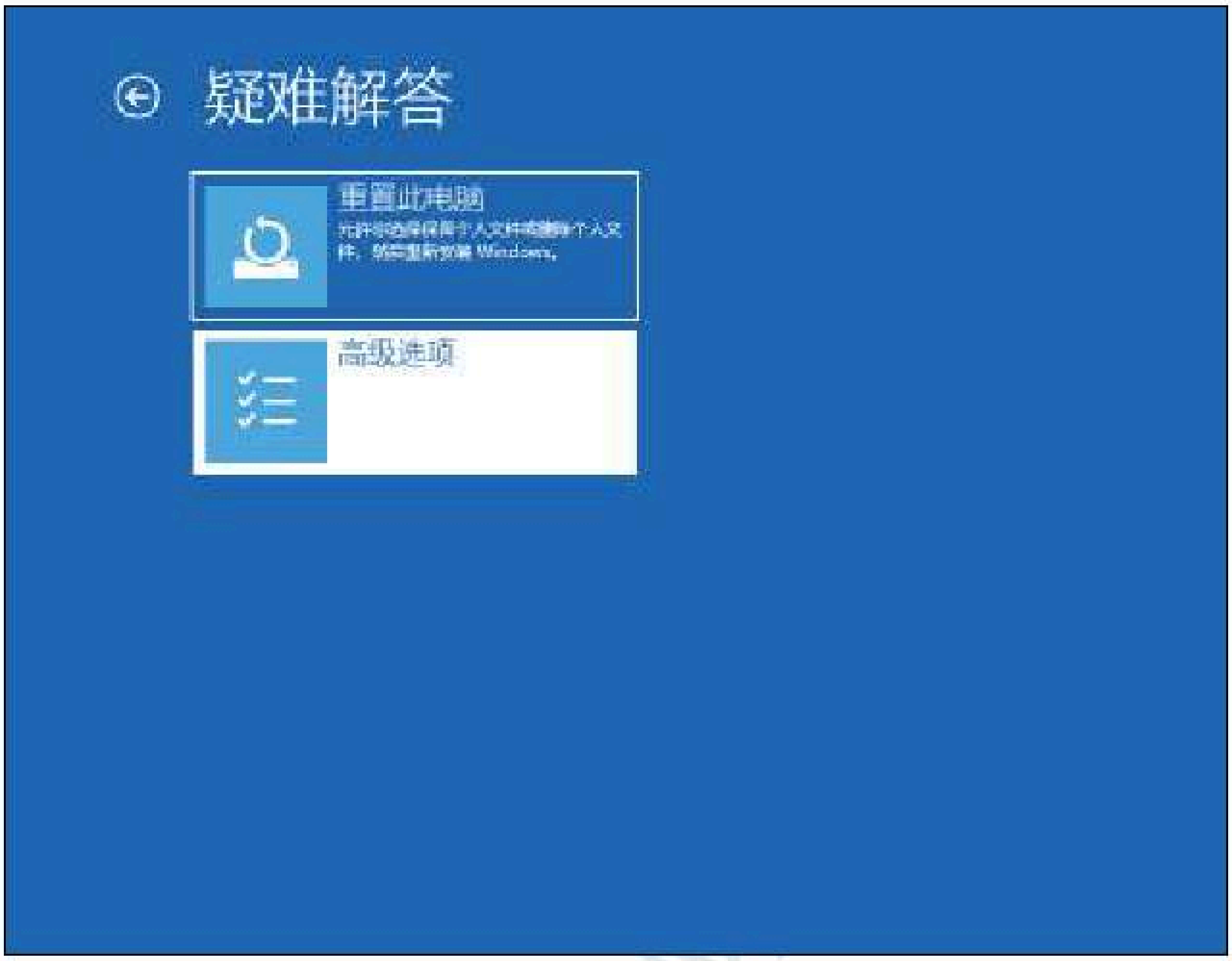
通过设备或磁盘(如 U 盘或 DVD)启动，更改你的电脑固件设置，更改 Windows 启动设置，或者从系统映像还原 Windows。 这将重新启动电脑。

更多恢复选项

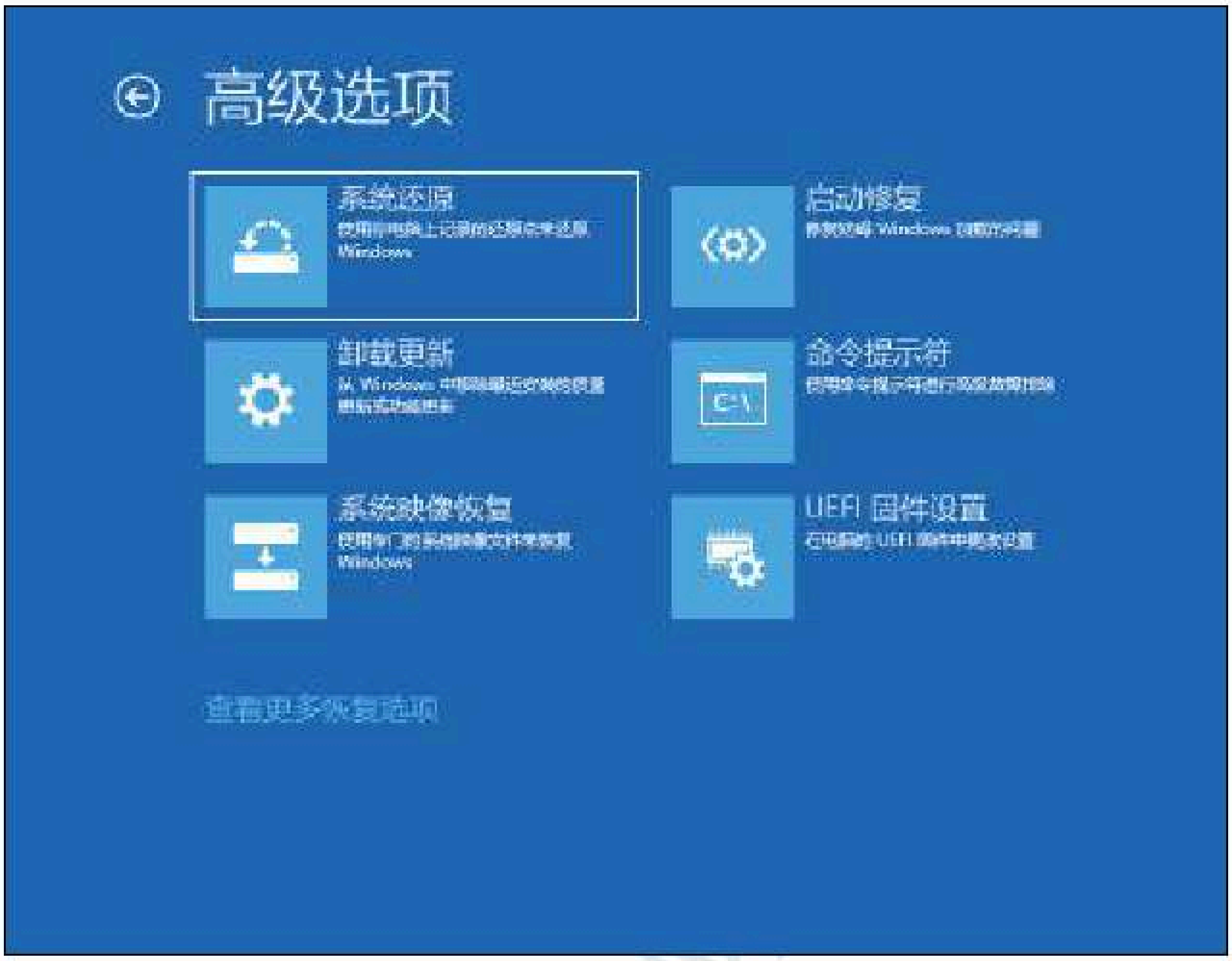
Before the computer restarts, the system will first enter the following startup menu and select the "Troubleshooting" item


选择一个选项
继续
退出并继续使用 Windows 10
关闭电脑
使用设备
使用 U 盘、网络连接或 Windows 恢复 DVD
疑难解答
重置你的电脑或查看高级选项

Select "Advanced Options" in the troubleshooting interface



Then select "View more recovery options"



高级选项

系统还原
使用你电脑上记录的还原点来还原 Windows

启动修复
修复妨碍 Windows 加载的问题

卸载更新
从 Windows 中删除最近安装的质量更新或功能更新

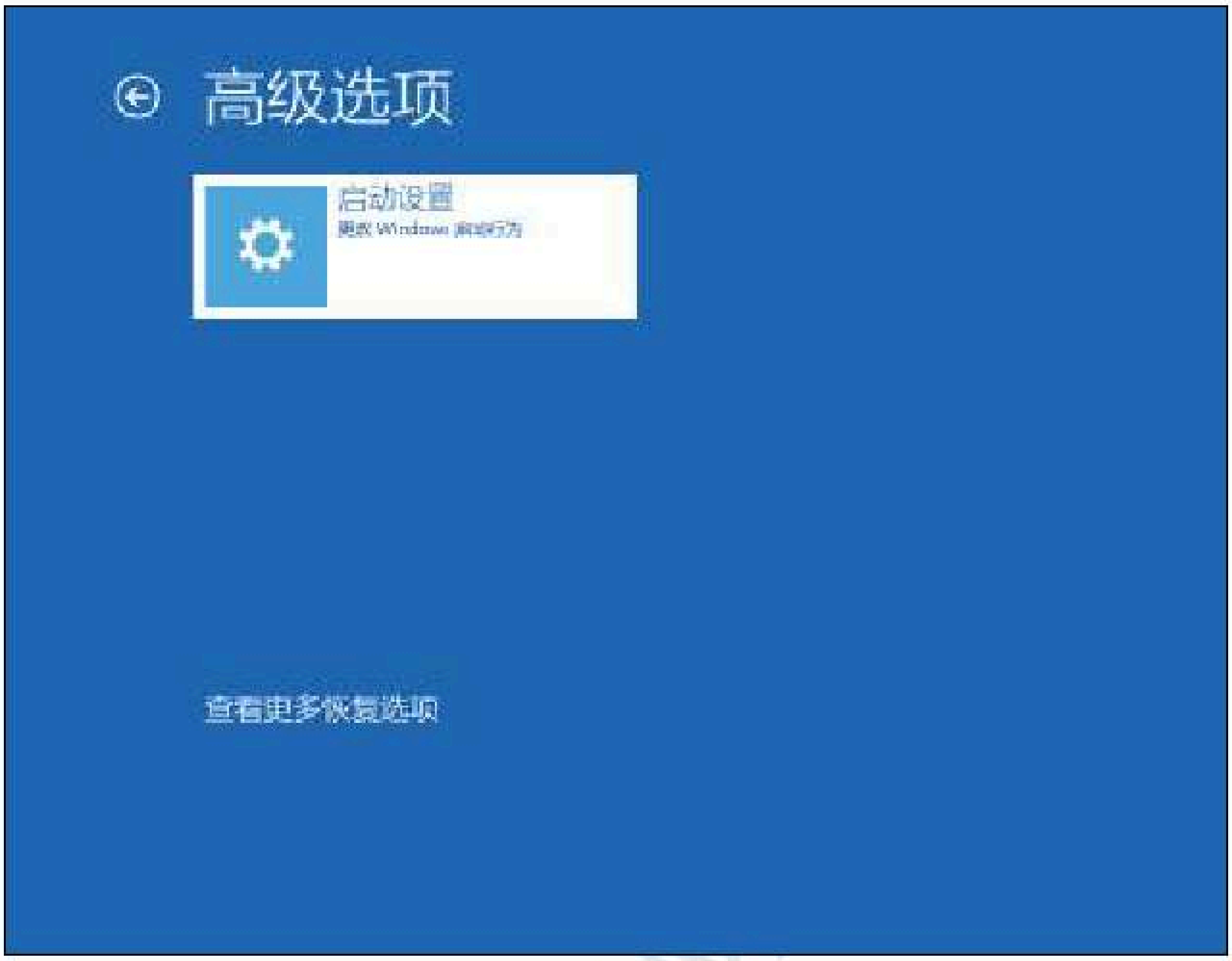
命令提示符
使用命令提示符进行高级故障排除

系统映像恢复
使用特定的系统映像文件来恢复 Windows

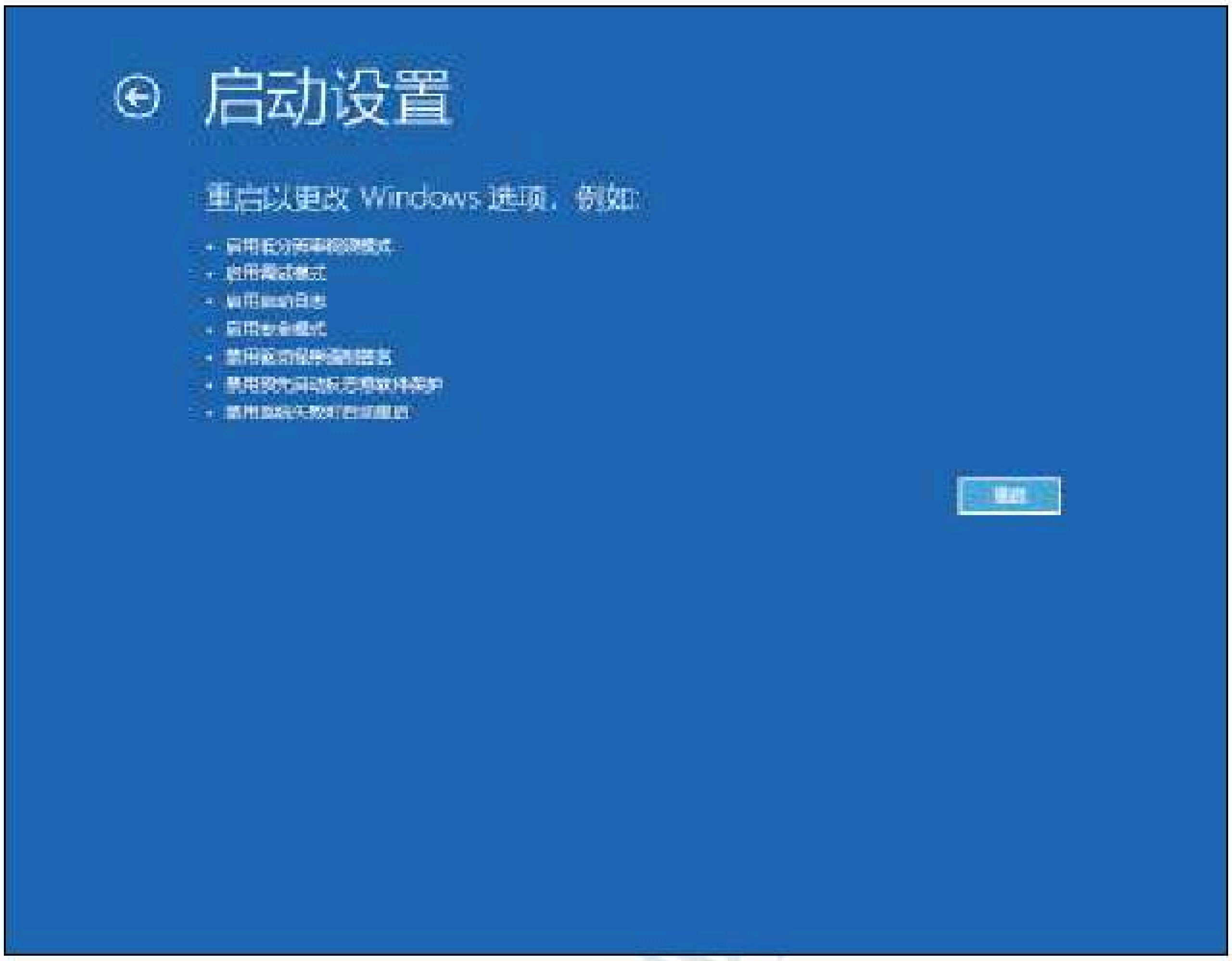
UEFI 固件设置
在电脑的 UEFI 固件中更改设置

查看更多恢复选项

Select the "Startup settings" item



After the following screen appears, click the "Restart" button to restart the computer

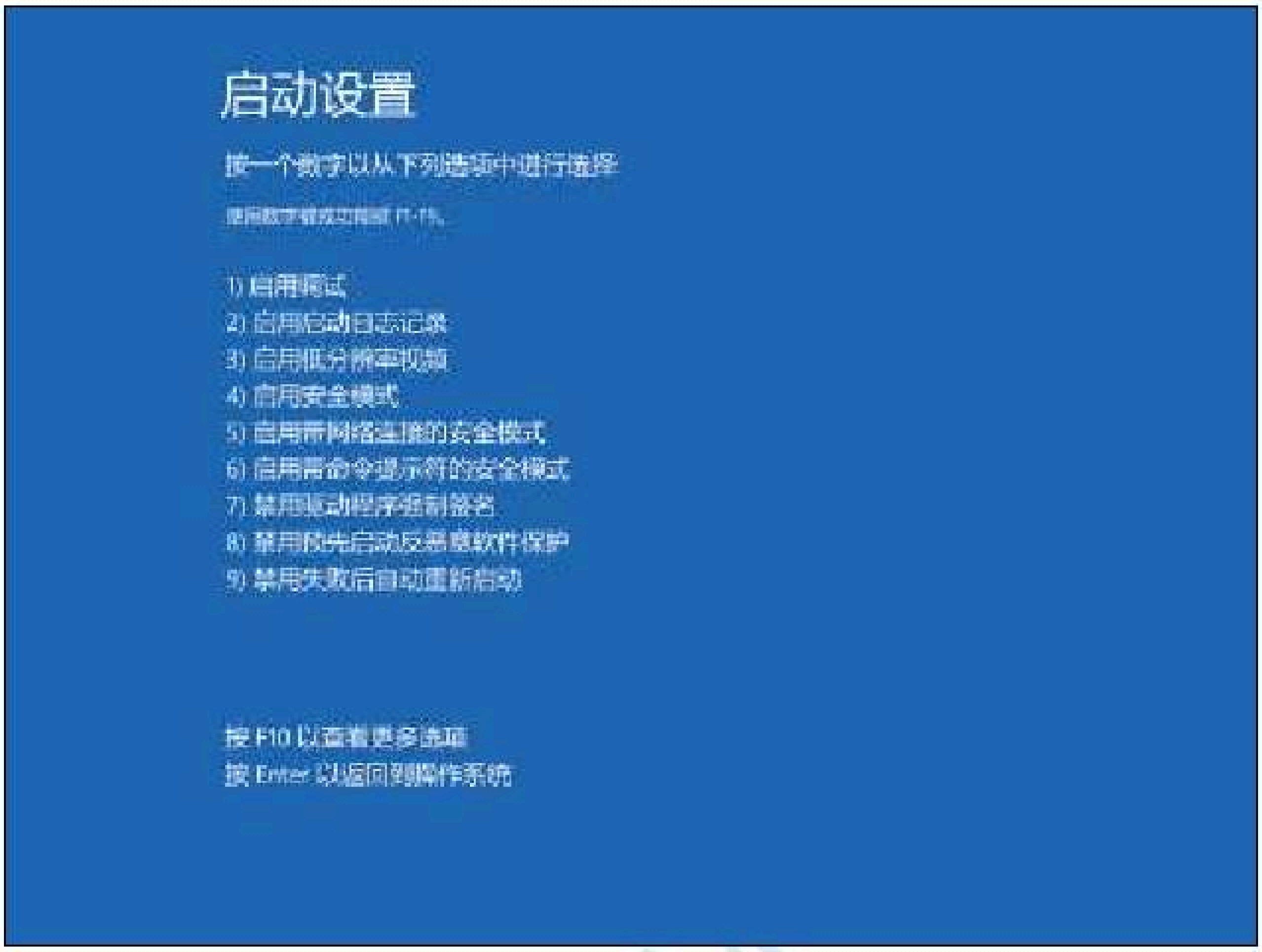


启动设置

重启以更改 Windows 选项，例如:

- 启用低分辨率视频模式
- 启用调试模式
- 启用启动日志
- 启用安全模式
- 禁用驱动程序强制签名
- 禁用预先启动反恶意软件保护
- 禁用失败后自动重新启动

After the computer restarts, the "Startup Settings" interface will pop up, press "F7" to select the "Prohibit driver from forcibly signing" item



启动设置

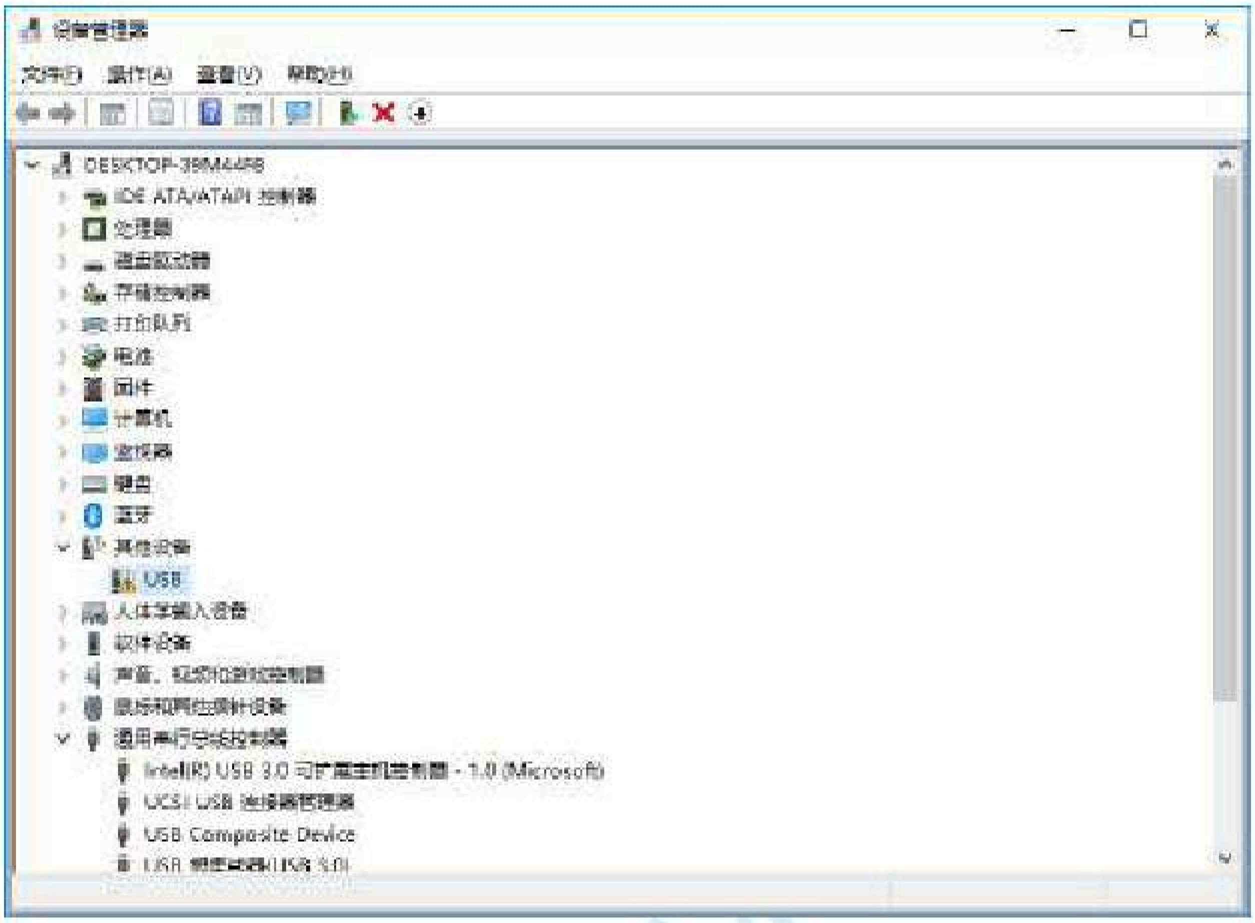
按一个数字以从下列选项中进行选择

使用数字键或功能键 F1-F9。

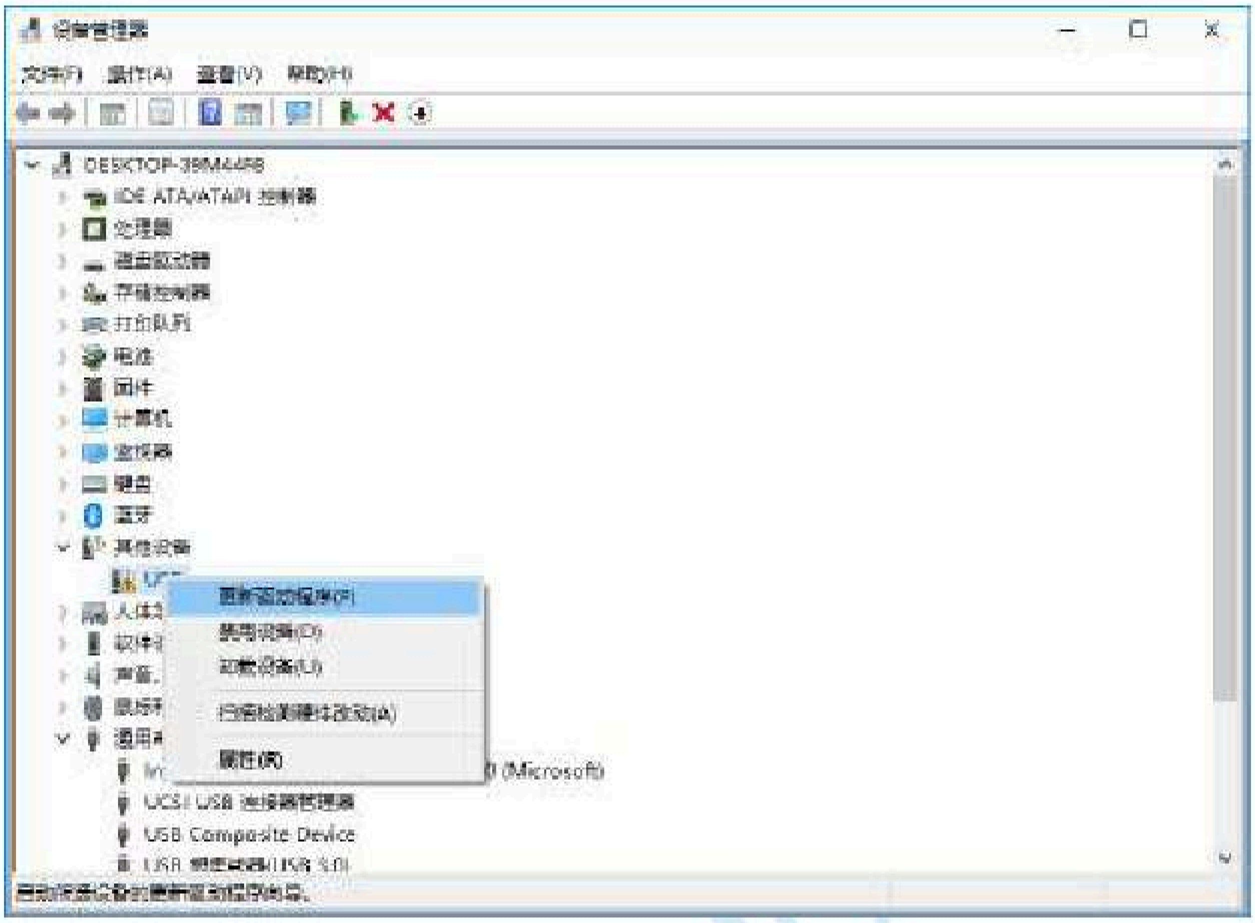
1) 启用调试
2) 启用启动日志记录
3) 启用低分辨率视频
4) 启用安全模式
5) 启用带网络连接的安全模式
6) 启用带命令提示符的安全模式
7) 禁用驱动程序强制签名
8) 禁用预先启动反恶意软件保护
9) 禁用失败后自动重新启动

按 F10 以查看更多选项
按 Enter 以返回到操作系统

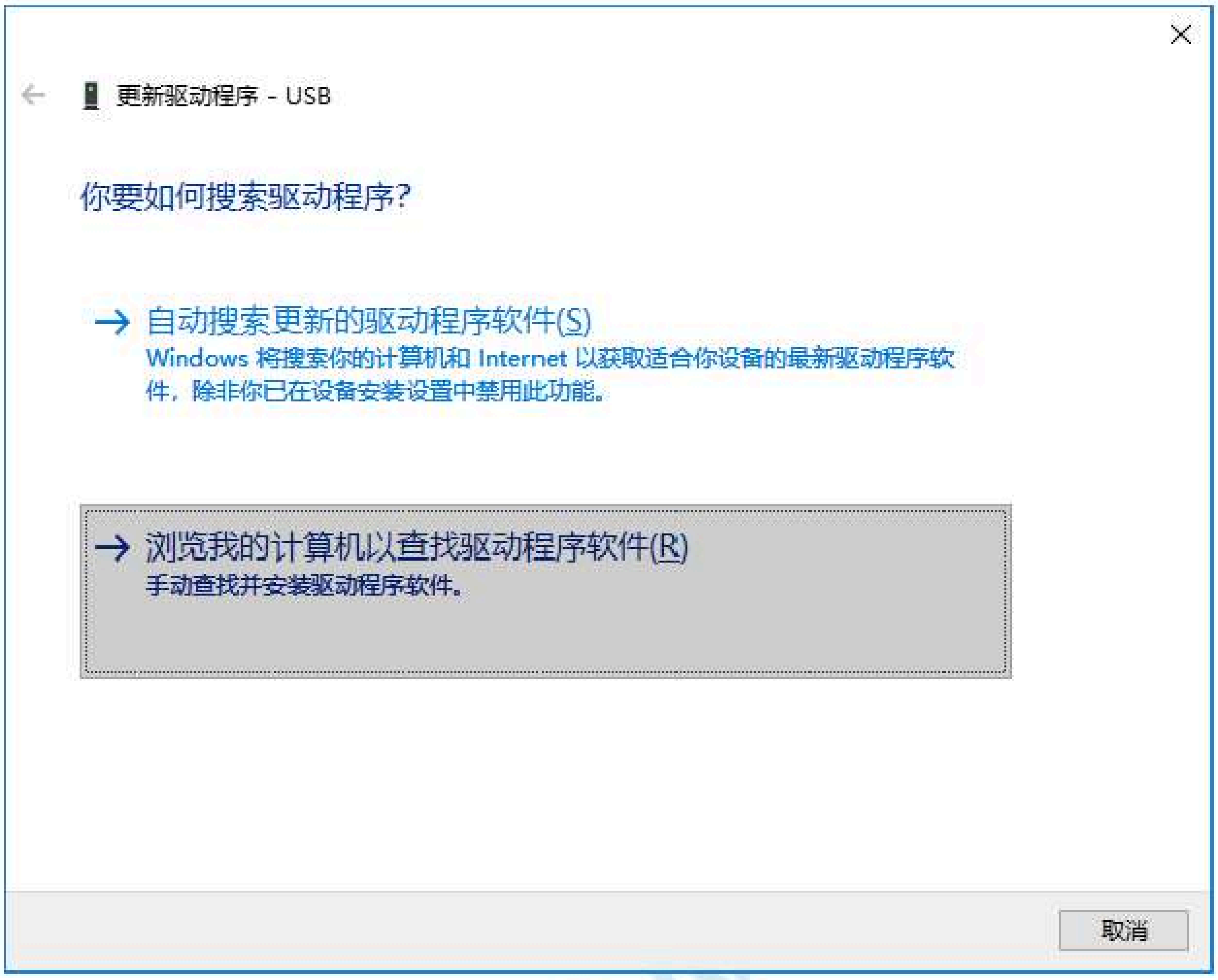
After the computer starts, use the prepared chip USB Connect the cable to the computer and open the "Device Manager". At this time, because the driver has not yet Start the installation, so it will be displayed as an unknown device with an exclamation mark in the device manager



Right-click the unknown device and select "Update Driver" in the context menu



In the pop-up driver installer selection screen, select the "Browse my Computer to find driver software" item


更新驱动程序 - USB

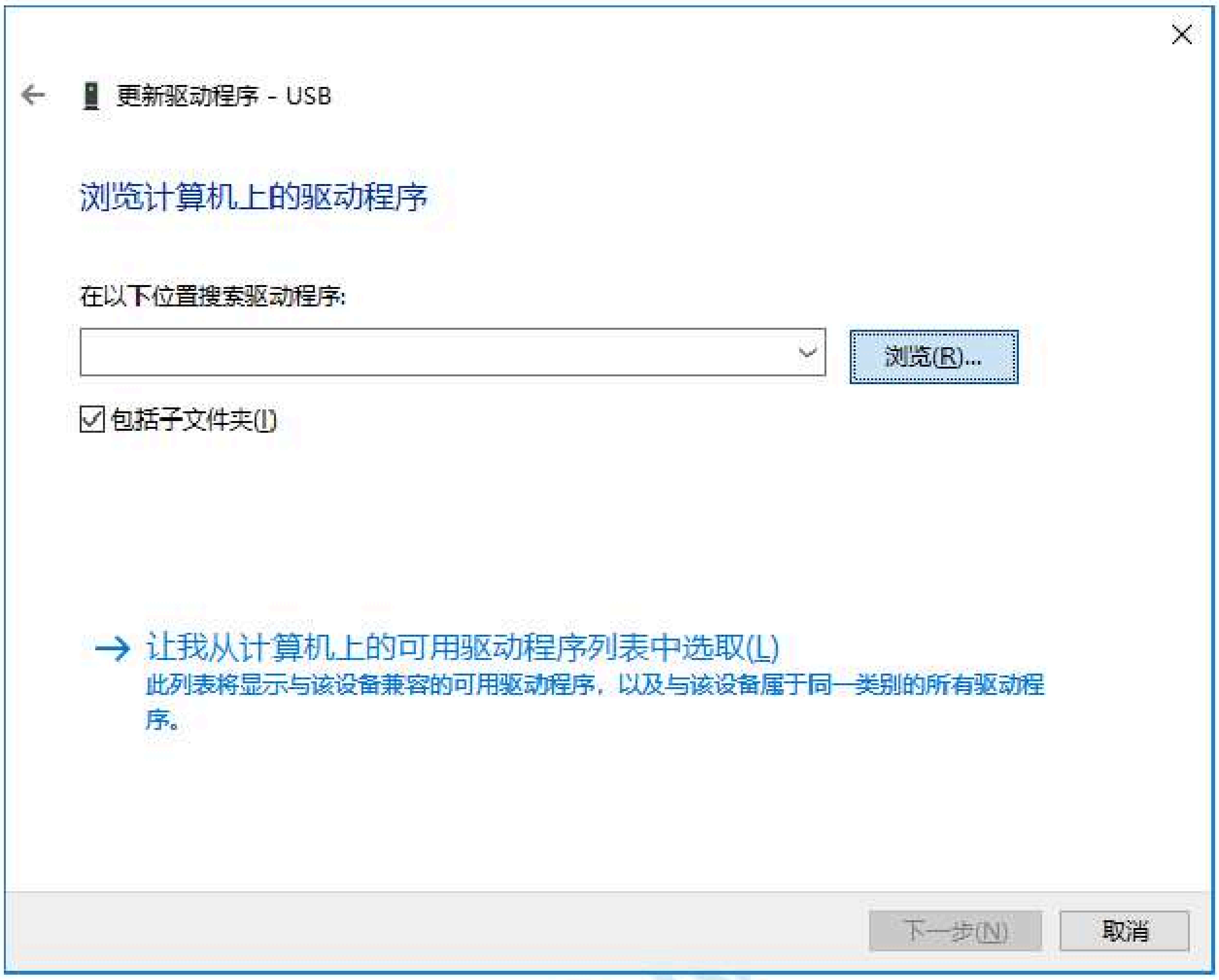
你要如何搜索驱动程序?

→ 自动搜索更新的驱动程序软件(S)
Windows 将搜索你的计算机和 Internet 以获取适合你设备的最新驱动程序软件，除非你已在设备安装设置中禁用此功能。

→ 浏览我的计算机以查找驱动程序软件(R)
手动查找并安装驱动程序软件。

取消

In the following interface, click the "Browse" button



更新驱动程序 - USB

浏览计算机上的驱动程序

在以下位置搜索驱动程序:

浏览(R)...

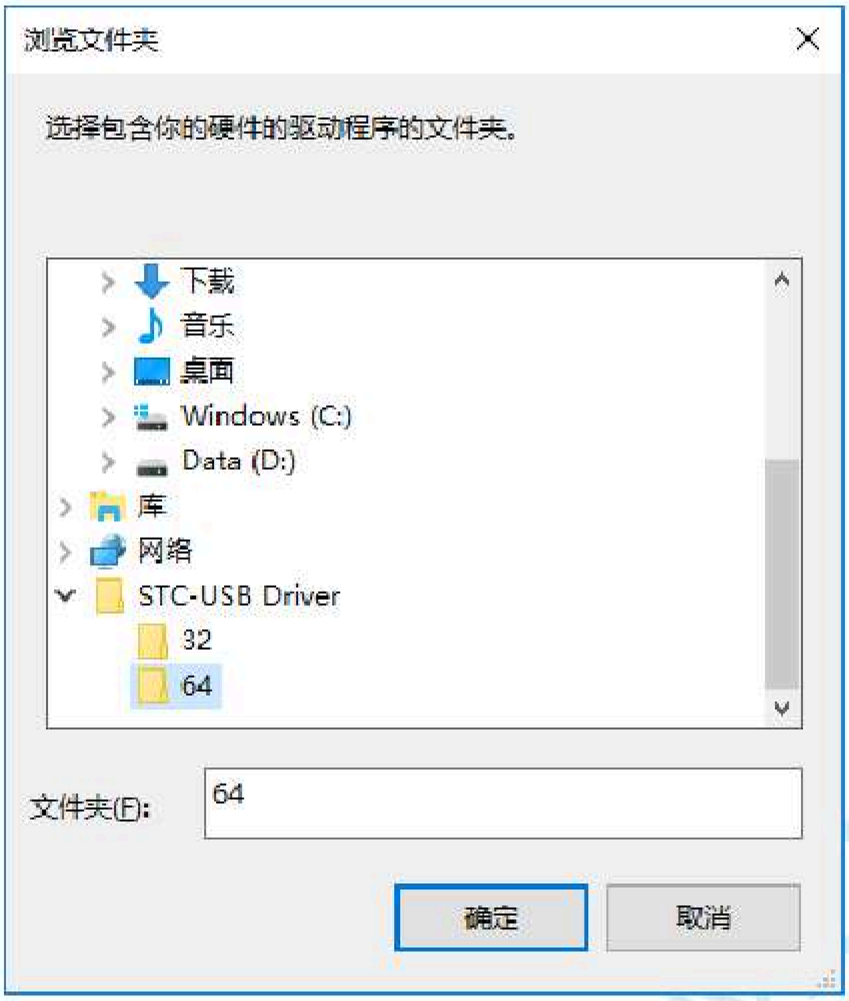
☑ 包括子文件夹(I)

→ 让我从计算机上的可用驱动程序列表中选取(L)

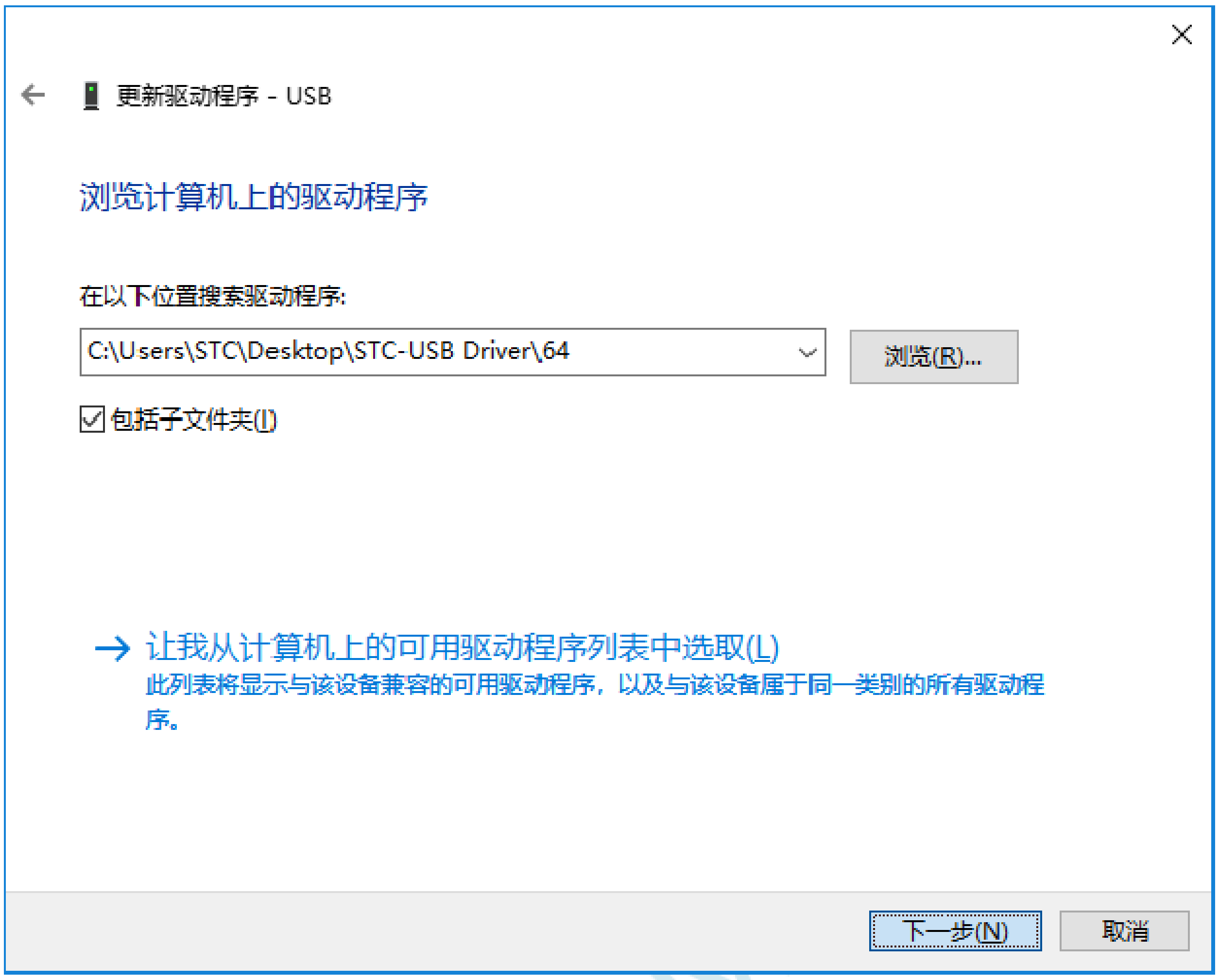
此列表将显示与该设备兼容的可用驱动程序，以及与该设备属于同一类别的所有驱动程序。

下一步(N) 取消

Find the "previously unzipped to the hard disk" "Directory, select "in the directory 64 STC-USB Driver "Table of contents, and determine



Click "Next" to start installing the driver



更新驱动程序 - USB

浏览计算机上的驱动程序

在以下位置搜索驱动程序:

C:\Users\STC\Desktop\STC-USB Driver\64

浏览(R)...

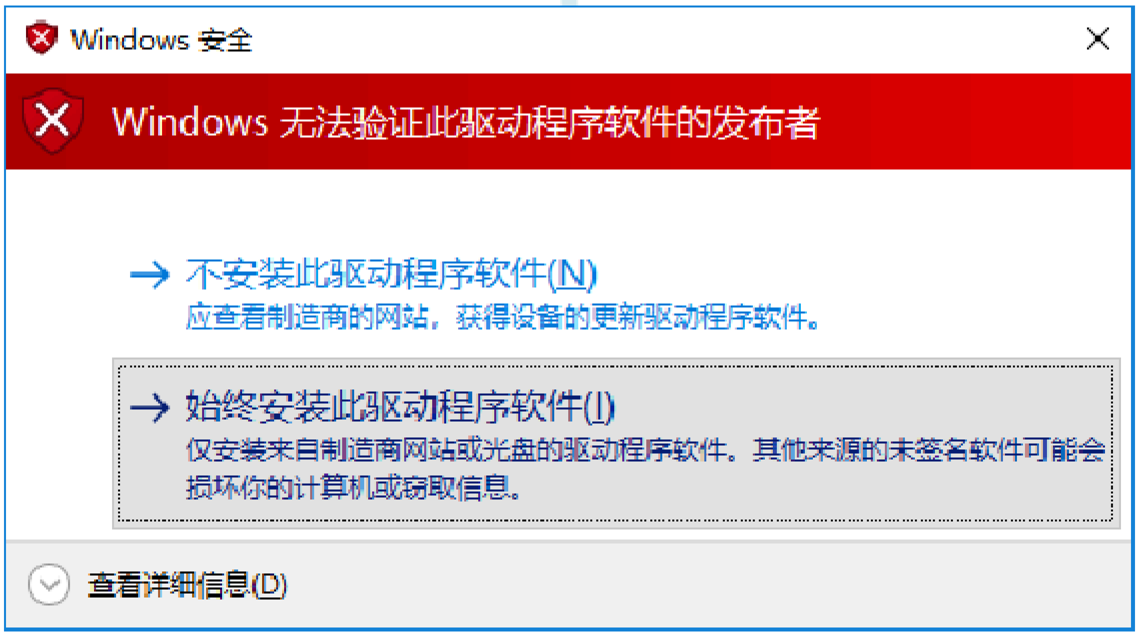
☑ 包括子文件夹(I)

→ 让我从计算机上的可用驱动程序列表中选取(L)

此列表将显示与该设备兼容的可用驱动程序，以及与该设备属于同一类别的所有驱动程序。

下一步(N)　取消

During the driver installation process, the following warning screen will pop up, select "Always install this driver software"



Windows 安全

Windows 无法验证此驱动程序软件的发布者

→ 不安装此驱动程序软件(N)

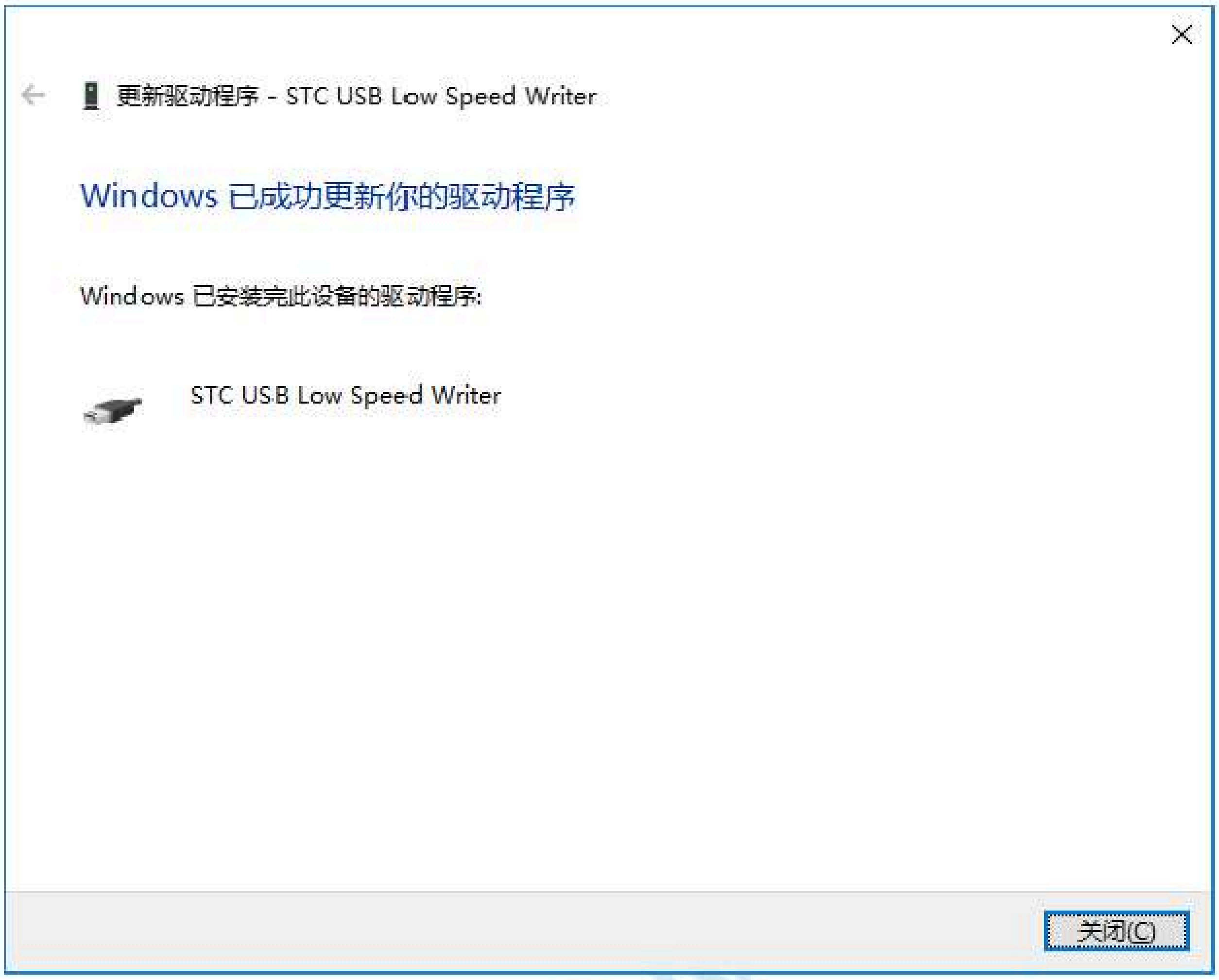
应查看制造商的网站，获得设备的更新驱动程序软件。

→ 始终安装此驱动程序软件(I)

仅安装来自制造商网站或光盘的驱动程序软件。其他来源的未签名软件可能会损坏你的计算机或窃取信息。

查看详细信息(D)

When the following screen appears, the driver is successfully installed



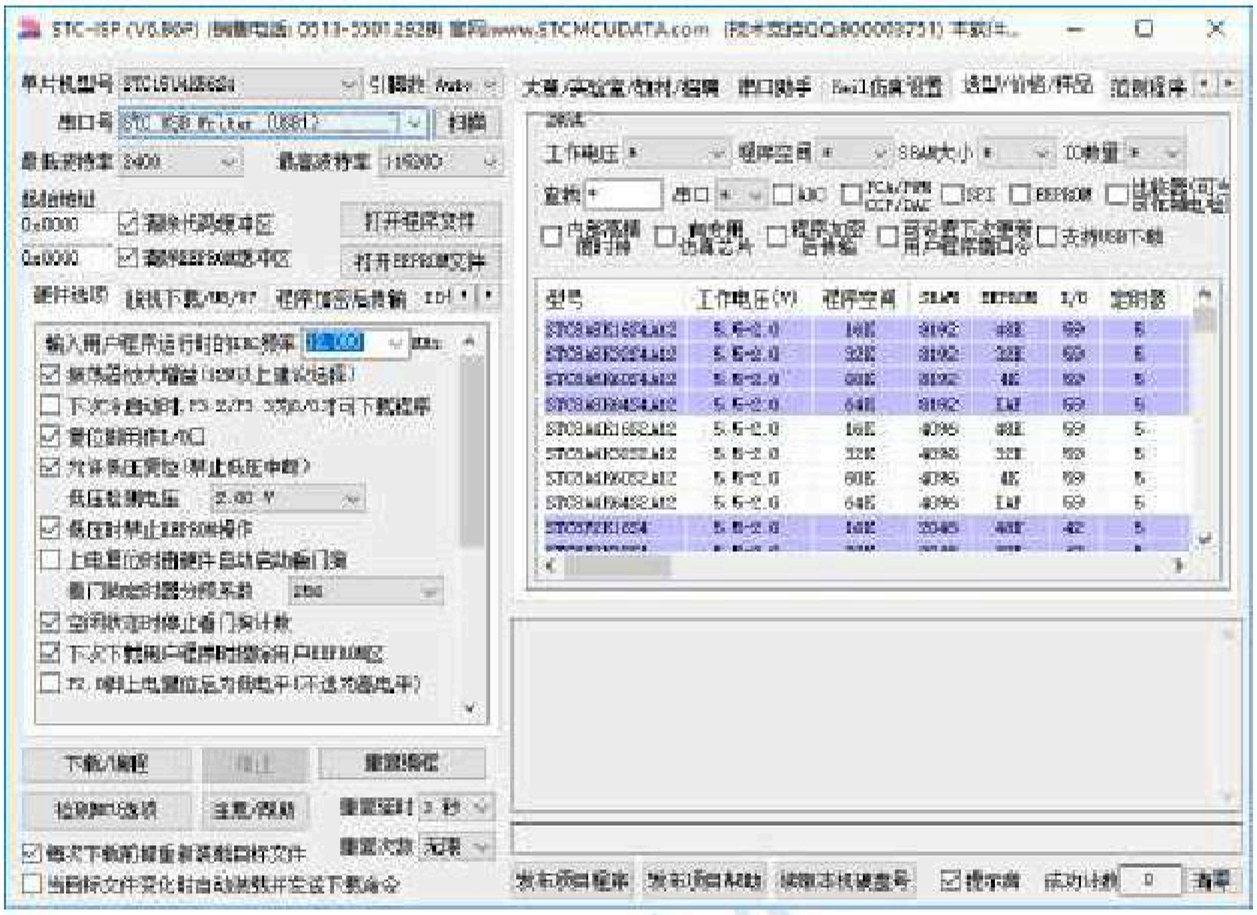
更新驱动程序 - STC USB Low Speed Writer

Windows 已成功更新你的驱动程序

Windows 已安装完此设备的驱动程序:

STC USB Low Speed Writer

Back STC-ISP **Download the software, at this time the "serial port number" has been automatically selected in the drop-down li to "** STC USB Writer (USB1) **", ready to use** USB **Proceed** ISP **downloaded**


STC-ISP (V6.86P) (销售电话: 0513-55012928) 官网www.STCMCUDATA.com (技术支持QQ:800003751) 本软件...

单片机型号 STC15W4K56S4　引脚数 Auto
串口号 STC USB Writer (USB1)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件
硬件选项　脱机下载/U8/U7　程序加密后传输　ID号

输入用户程序运行时的IRC频率 12.000 MHz
☑ 振荡器放大增益(12M以上建议选择)
☐ 下次冷启动时，P3.2/P3.3为0/0才可下载程序
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.00 V
☑ 低压时禁止EEPROM操作
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区
☐ P2.0脚上电复位后为低电平(不选为高电平)

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 3 秒
☑ 每次下载前都重新装载目标文件　重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

范例程序/实验室/资料/视频　串口助手　Keil仿真设置　选型/价格/样品　范例程序

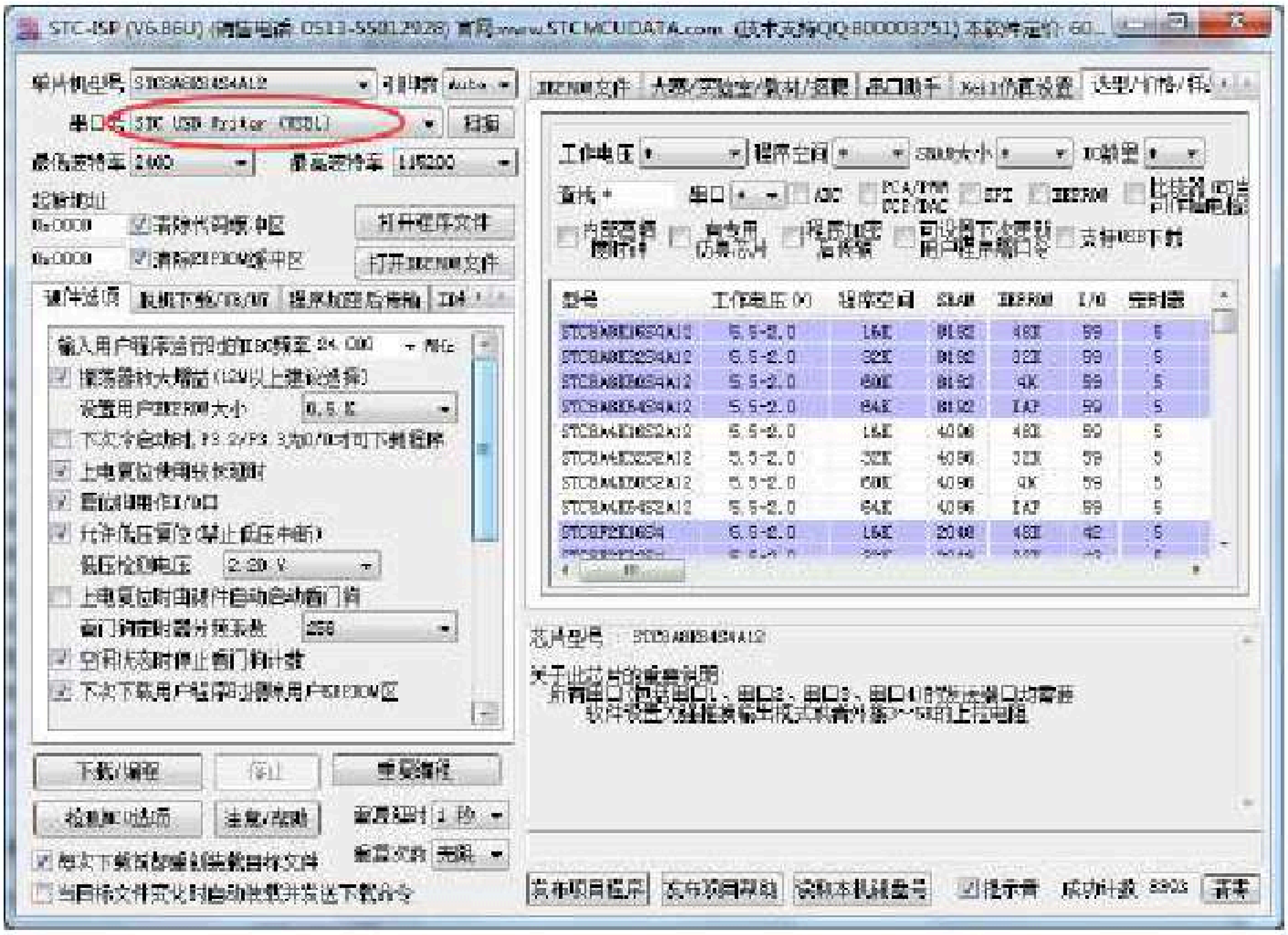
筛选
工作电压 *　程序空间 *　SRAM大小 *　IO数量 *
封装 *　串口 *　☐ ADC　☐ PCA/PWM CCP/DAC　☐ SPI　☐ EEPROM　☐ 比较器(可当ADC使用)
☐ 内部高精准时钟　☐ 可当仿真芯片　☐ 程序加密后传输　☐ 可设置下次更新用户程序需口令　☐ 支持USB下载

型号	工作电压(V)	程序空间	SRAM	EEPROM	I/O	定时器
STC8A8K16S4A12	5.5-2.0	16K	8192	48K	59	5
STC8A8K32S4A12	5.5-2.0	32K	8192	32K	59	5
STC8A8K60S4A12	5.5-2.0	60K	8192	4K	59	5
STC8A8K64S4A12	5.5-2.0	64K	8192	IAP	59	5
STC8A4K16S2A12	5.5-2.0	16K	4096	48K	59	5
STC8A4K32S2A12	5.5-2.0	32K	4096	32K	59	5
STC8A4K60S2A12	5.5-2.0	60K	4096	4K	59	5
STC8A4K64S2A12	5.5-2.0	64K	4096	IAP	59	5
STC8F2K16S4	5.5-2.0	16K	2048	48K	42	5

发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 0　清零

D Appendix USB Download step-by-step demonstration

1, First refer P5.1.5 The application circuit diagram of the chapter is connected to the P3.2 The port is connected to Gnd and then connect the system to the receiving end USB PC microcontroller, and the port of the target chip is connected. open ISP Download the software, you can automatically search for "in the serial port number of the downlo STC USB Writer (USB1)"of USB equipment


STC-ISP (V6.86U) (销售电话: 0513-55012928) 官网: www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60...

单片机型号 STC8A8K64S4A12　引脚数 Auto
串口号 STC USB Writer (USB1)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | ID号

输入用户程序运行时的IRC频率 24.000 MHz
☑ 振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
☐ 下次冷启动时, P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.20 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 1 秒
☑ 每次下载前都重新装载目标文件　重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

EEPROM文件 | 范例/实验室/教材/资料 | 串口助手 | Keil仿真设置 | 选型/价格/样品

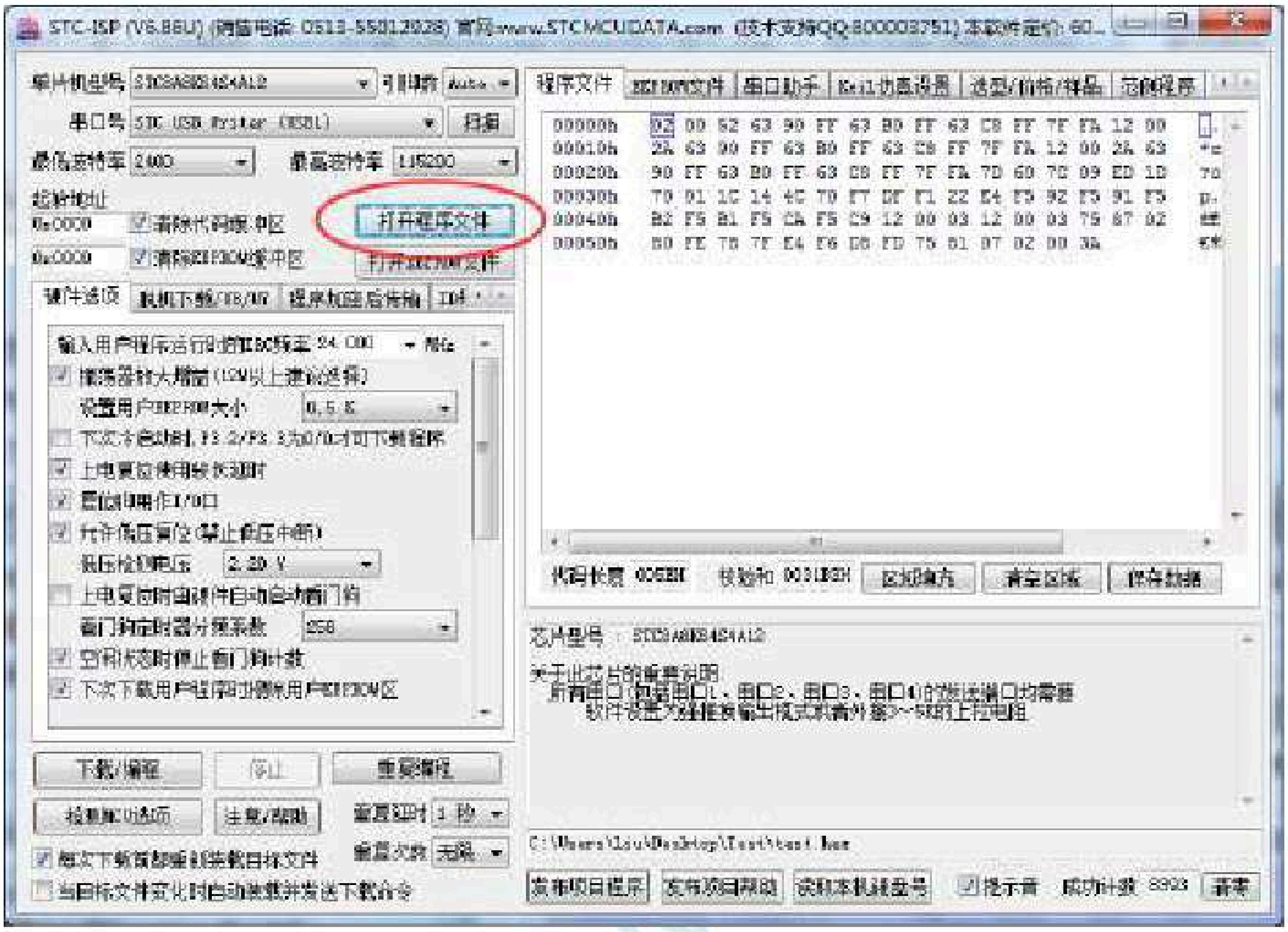
工作电压 *　程序空间 *　SRAM大小 *　IO数量 *
查找 *　串口 *　☐ ADC　☐ PCA/PWM DCP/DAC　☐ SPI　☐ EEPROM　☐ 比较器 可当PIT使用
☐ 内部高精度时钟　☐ 有专用仿真芯片　☐ 程序加密后传输　☐ 可设置下次更新用户程序端口号　☐ 支持USB下载

型号	工作电压(V)	程序空间	SRAM	EEPROM	I/O	定时器
STC8A8K16S4A12	5.5-2.0	16K	8192	48K	59	5
STC8A8K32S4A12	5.5-2.0	32K	8192	32K	59	5
STC8A8K60S4A12	5.5-2.0	60K	8192	4K	59	5
STC8A8K64S4A12	5.5-2.0	64K	8192	IAP	59	5
STC8A4K16S2A12	5.5-2.0	16K	4096	48K	59	5
STC8A4K32S2A12	5.5-2.0	32K	4096	32K	59	5
STC8A4K60S2A12	5.5-2.0	60K	4096	4K	59	5
STC8A4K64S2A12	5.5-2.0	64K	4096	IAP	59	5
STC8F2K16S4	5.5-2.0	16K	2048	48K	42	5

芯片型号 STC8A8K64S4A12
关于此芯片的重要说明
所有串口(包括串口1、串口2、串口3、串口4)的发送和接收端口均需要
软件设置为准双向或强推挽模式并外加3~5K的上拉电阻

发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 8903　清零

2 , Open the user code program



STC-ISP (V6.88U) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60...

单片机型号 STC8A8K64S4A12　引脚数 Auto
串口号 STC USB Writer (USB1)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | ID号

输入用户程序运行时的IRC频率 24.000 MHz
☑ 振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.20 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 1 秒
☑ 每次下载前都重新装载目标文件　重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

程序文件 | EEPROM文件 | 串口助手 | Keil仿真设置 | 选型/价格/样品 | 范例程序

```
00000h  02 00 52 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00
00010h  2A 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00 2A 63
00020h  90 FF 63 B0 FF 63 C8 FF 7F FA 7D 60 7C 09 ED 1D
00030h  70 01 1C 14 4C 70 F7 DF F1 22 E4 F5 92 F5 91 F5
00040h  B2 F5 B1 F5 CA F5 C9 12 00 03 12 00 03 75 87 02
00050h  80 FE 78 7F E4 F6 D8 FD 75 81 07 02 00 3A
```

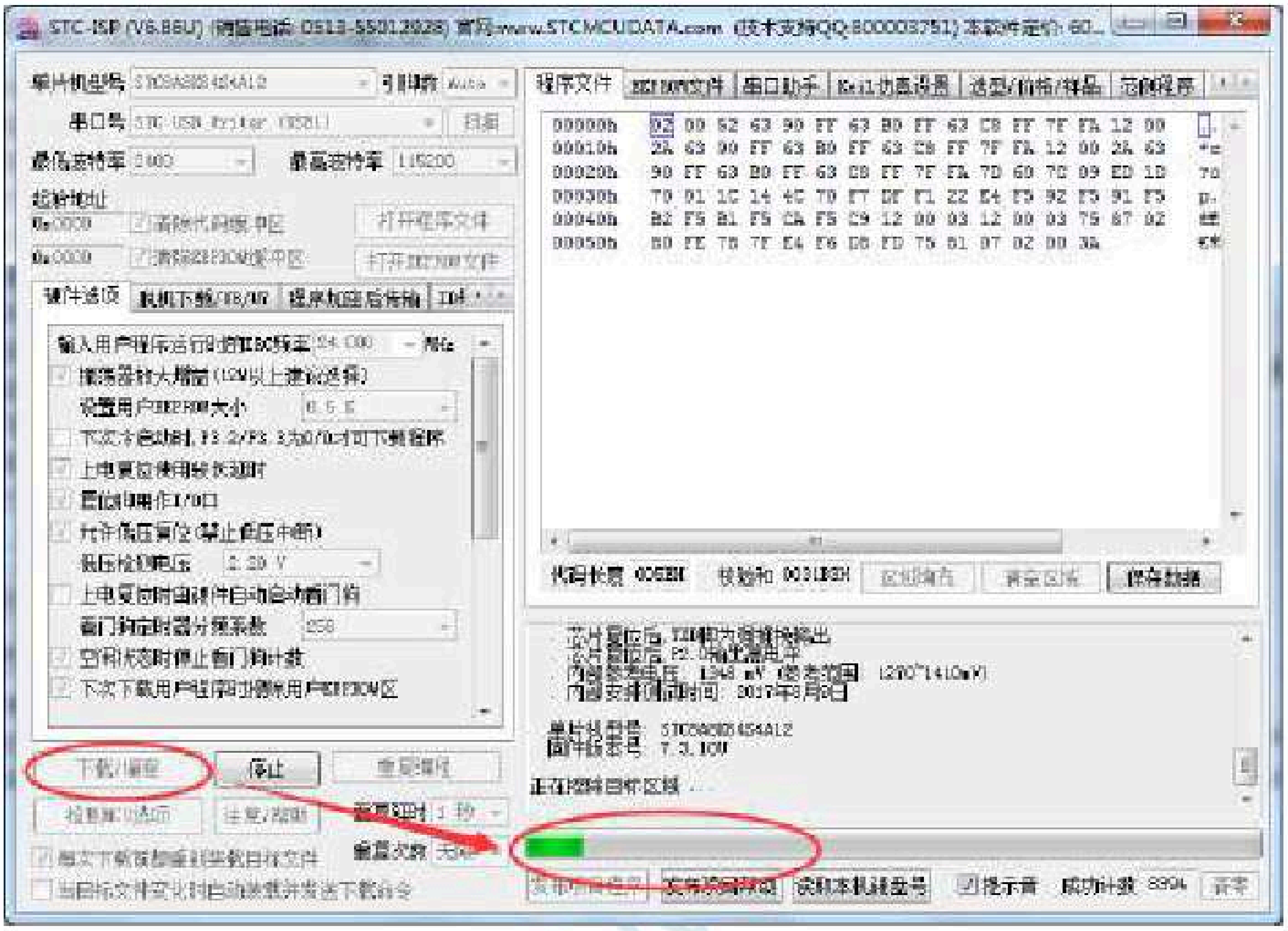
代码长度 005EH　校验和 0031B8H　区域填充　清空区域　保存数据

芯片型号 : STC8A8K64S4A12
关于此芯片的重要说明
所有串口(包括串口1、串口2、串口3、串口4)的发送端口均需要
软件设置为强推挽输出模式或者外接3~5K的上拉电阻

C:\Users\Liu\Desktop\Test\test.hex

发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 8903　清零

3 , Click "Download/The "Program" button starts downloading the user code


STC-ISP (V6.86U) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60...

单片机型号 STC8A8K64S4A12 引脚数 Auto
串口号 STC USB Writer (USB1) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区 打开程序文件
0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | ID号

输入用户程序运行时的IRC频率 24.000 MHz
☑ 振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
☐ 下次冷启动时, P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.20 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区

下载/编程 | 停止 | 重复编程
检测MCU选项 | 注意/帮助 | 重复延时 1 秒
☑ 每次下载前都重新装载目标文件 重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

程序文件 | EEPROM文件 | 串口助手 | Keil仿真设置 | 选型/价格/样品 | 范例程序

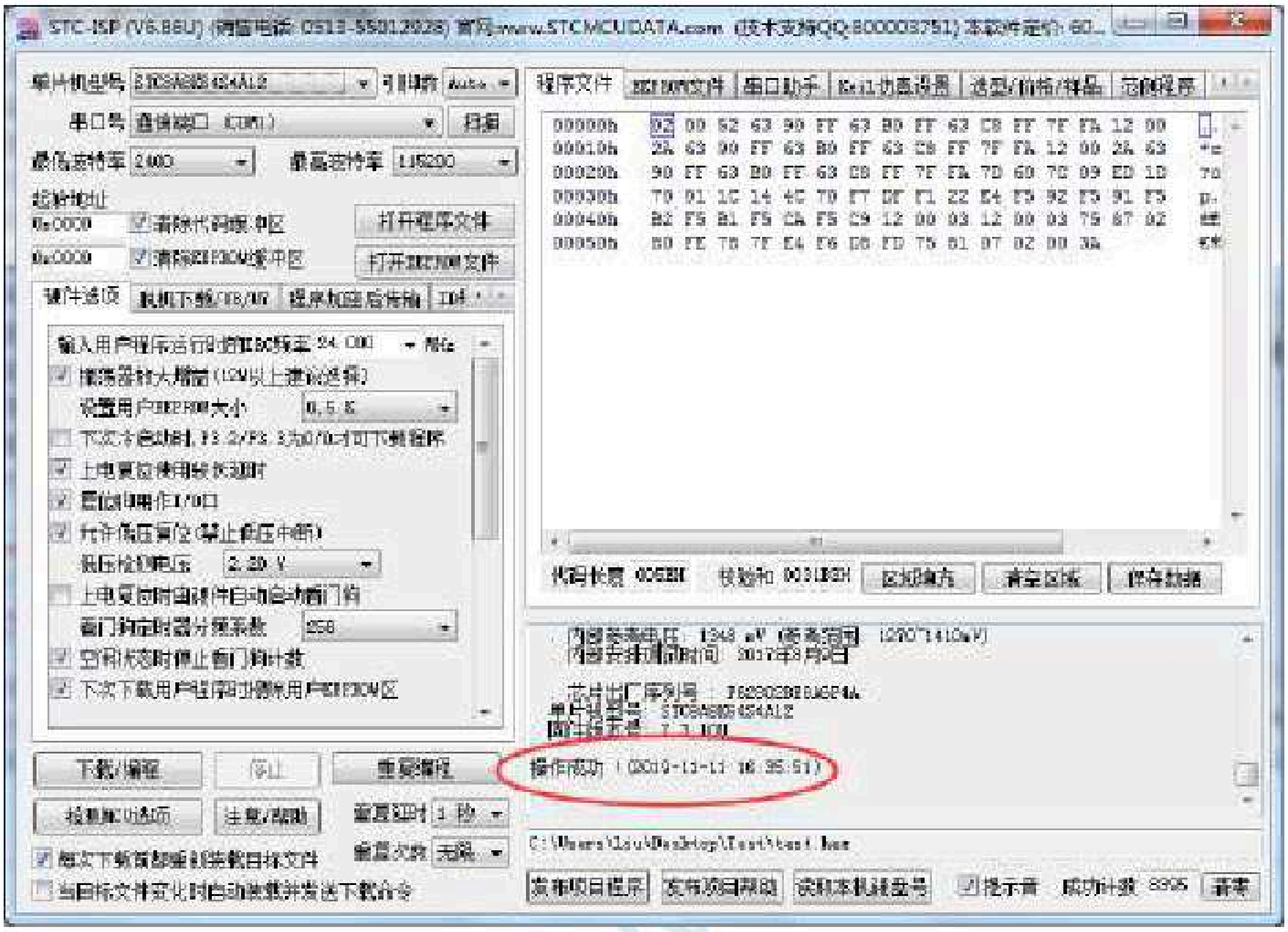
```
00000h  02 00 52 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00
00010h  2A 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00 2A 63
00020h  90 FF 63 B0 FF 63 C8 FF 7F FA 7D 60 7C 09 ED 1D
00030h  70 01 1C 14 4C 70 F7 DF F1 22 E4 F5 92 F5 91 F5
00040h  82 F5 81 F5 CA F5 C9 12 00 03 12 00 03 75 87 02
00050h  80 FE 75 7F E4 F6 D8 FD 75 81 07 02 00 3A
```

代码长度 005EH 校验和 0031BEH 区域填充 清空区域 保存数据

```
  芯片复位后, TxD脚为强推挽输出
  芯片复位后, P2.0脚输出高电平
  内部参考电压: 1348 mV (参考范围: 1290~1410mV)
  内部安排测试时间: 2017年8月8日

 单片机型号: STC8A8K64S4A12
 固件版本号: 7.3.10U

正在擦除目标区域 ...
```

发布项目程序 | 发布项目帮助 | 读取本机硬盘号 | ☑ 提示音 | 成功计数 8904 | 清零

4 , Indicating that the program code download is complete. , Until the prompt "Operation was successful


STC-ISP (V6.88U) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60...

单片机型号 STC8A8K64S4A12　引脚数 Auto
串口号 通信端口 (COM1)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件

硬件选项 | 脱机下载/U8/U7 | 程序加密后传输 | ID号

输入用户程序运行时的IRC频率 24.000 MHz
☑ 振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
☐ 下次冷启动时, P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.20 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 1 秒
☑ 每次下载前都重新装载目标文件　重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

程序文件 | EEPROM文件 | 串口助手 | Keil仿真设置 | 选型/价格/样品 | 范例程序

```
00000h  02 00 52 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00
00010h  2A 63 90 FF 63 B0 FF 63 C8 FF 7F FA 12 00 2A 63
00020h  90 FF 63 B0 FF 63 D8 FF 7F FA 7D 60 7C 09 ED 1D
00030h  70 01 1C 14 4C 70 F7 DF F1 22 E4 F5 92 F5 91 F5
00040h  B2 F5 B1 F5 CA F5 C9 12 00 03 12 00 03 75 87 02
00050h  80 FE 75 7F E4 F6 D8 FD 75 81 07 02 00 3A
```

代码长度 005EH　校验和 0031BDH　区域填充　清除区域　保存数据

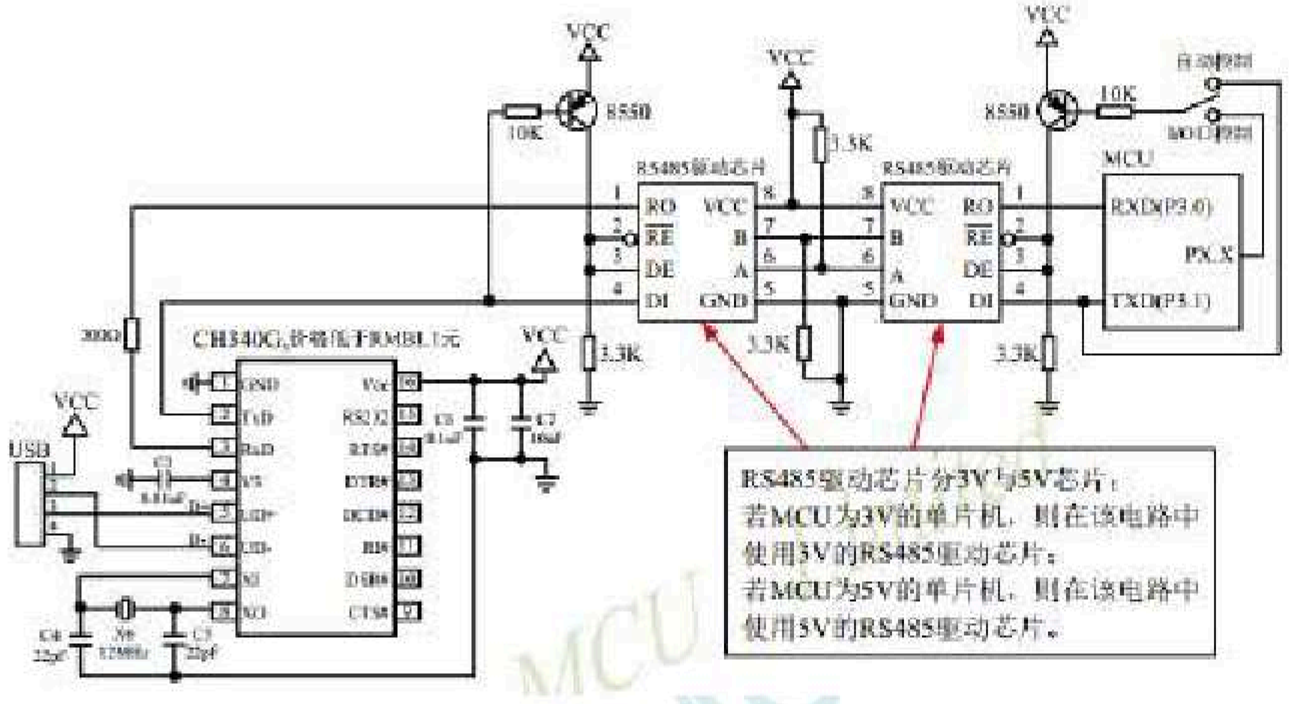
```
内部参考电压: 1348 mV (参考范围: 1290~1410mV)
内部安排测试时间: 2017年8月2日

芯片出厂序列号 : F62802D2D0AGF4A
单片机型号: STC8A8K64S4A12
固件版本号: 7.3.10U
操作成功 ! (2019-11-11 16:35:51)
```

C:\Users\Liu\Desktop\Test\test.hex

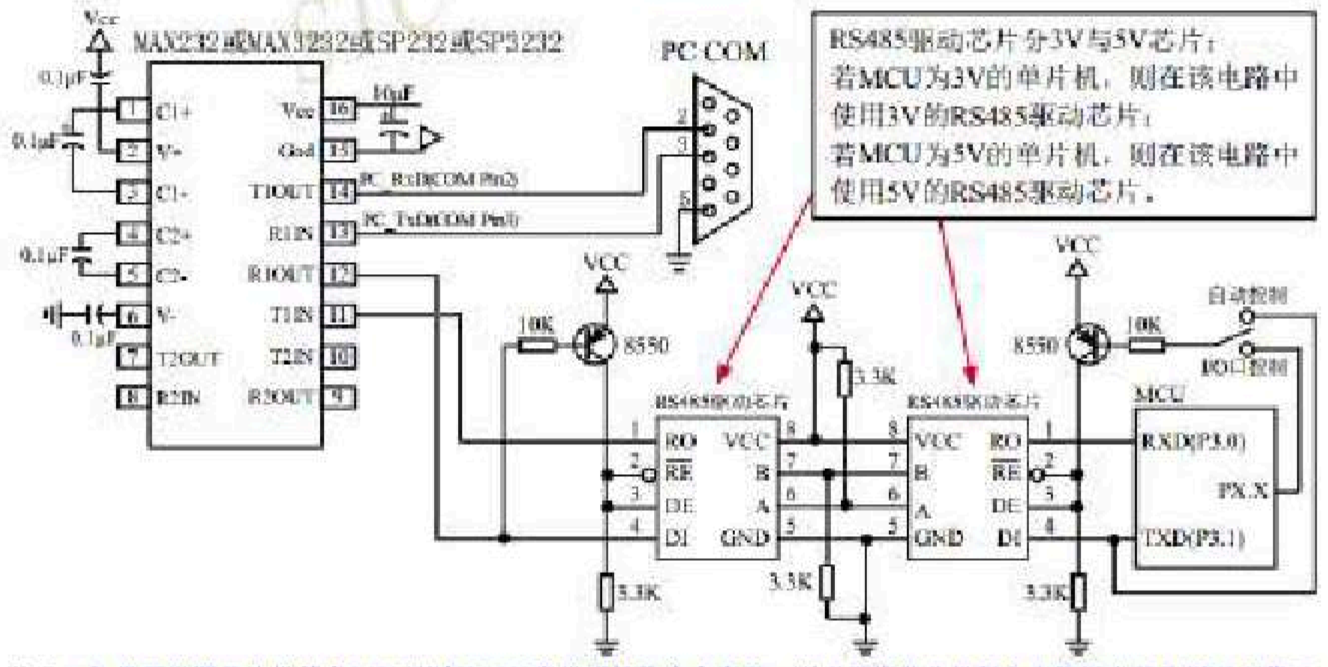
发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 8305　清零

E Appendix RS485 Automatic control or I/O Port control circuit diagram

1、use USB Connect to the computer via serial port RS485 Control download circuit diagram (Automatic control or I/O Port control)



2、use RS232 Connect to the computer via serial port RS485 Control download circuit diagram (Automatic control or I/O Port control)



注意：如果要设置单片机某个I/O口控制RS485发送或接收命令有效，则必须将单片机焊入电路板之前先用U8下载工具结合电脑ISP软件对该单片机进行“RS485控制”设置并烧录一下（如上节所述），否则该单片机实现不了RS485控制功能。

建议用户将本节所述“RS485控制下载线路图(自动控制或I/O口控制)”设计到您的用户板上

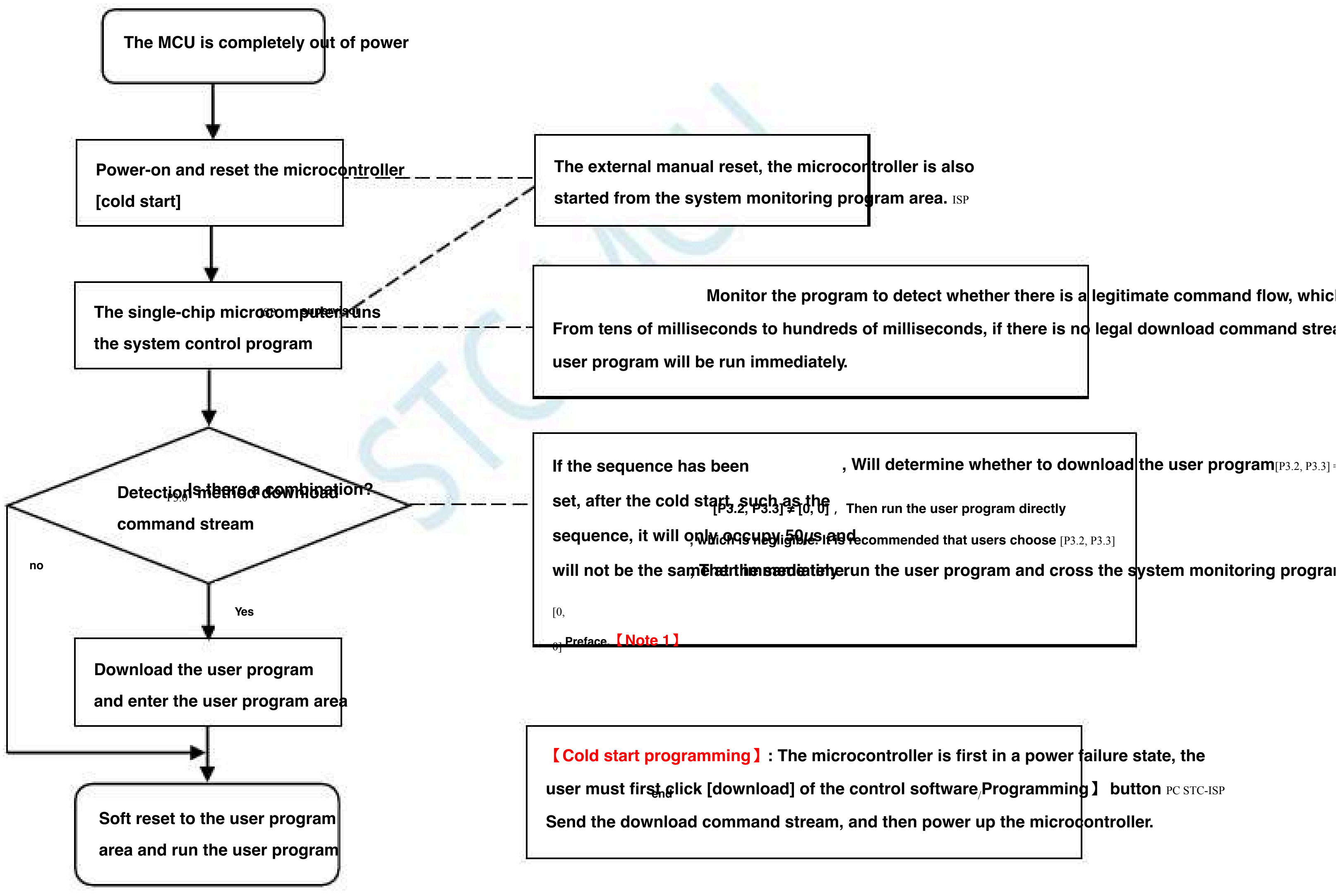
F Appendix STC Tool instruction manual

F. 1 overview

U8W/U8W-Mini It is a series of programming tools that combine online online download and offline download. STC universal USB to serial port tool It is a programming tool that supports online download and online simulation

Tool type	Online download	offline download	Burner download support is	Online simulation needs	price (RMB)
U8W	Support	Support		to set the pass-through mode,	Yuan
U8W-Mini	support	support	not supported, not supported	the pass-through mode	50 Yuan
Universal USB To serial port	support	not support		needs to be set to support	30 Yuan

F. 2 System programmable (ISP) Process description


The MCU is completely out of power
Power-on and reset the microcontroller [cold start]
The single-chip microcomputer runs ISP the system control program
Detection P3.0 method download Is there a combination? command stream
no
Yes
Download the user program and enter the user program area
Soft reset to the user program area and run the user program

The external manual reset, the microcontroller is also started from the system monitoring program area. ISP

Monitor the program to detect whether there is a legitimate command flow, which From tens of milliseconds to hundreds of milliseconds, if there is no legal download command strea user program will be run immediately.

If the sequence has been , Will determine whether to download the user program [P3.2, P3.3] set, after the cold start, such as the [P3.2, P3.3] ≠ [0, 0] , Then run the user program directly sequence, it will only occupy 50μs and , which is negligible. It is recommended that users choose [P3.2, P3.3] will not be the same. Then immediately run the user program and cross the system monitoring progra [0, 0] Preface 【Note 1】

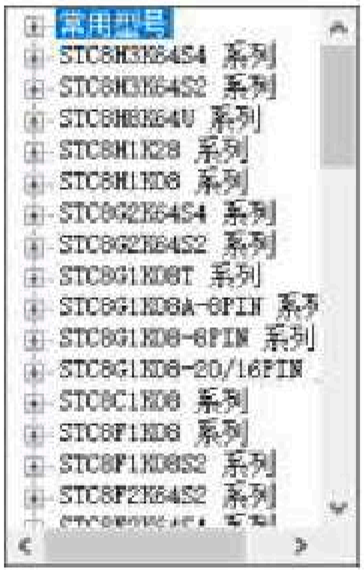
【Cold start programming】: The microcontroller is first in a power failure state, the user must first click [download] of the control software Programming】 button PC STC-ISP Send the download command stream, and then power up the microcontroller.

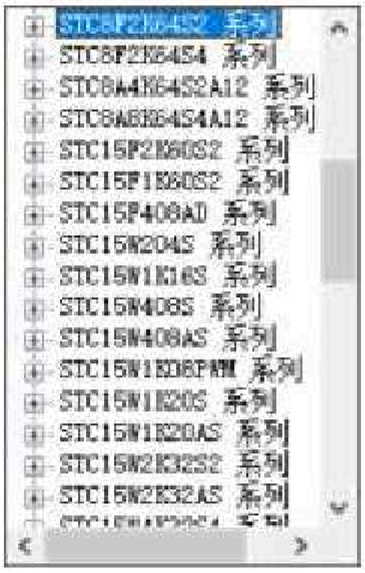
Note: [P3.0, P3.1] Make a download/For simulation (download/Simulation interface is, so it is recommended that users connect the serial p due to P3.6/P3.7 or P1.6/P1.7 , If the user does not want to switch, stick to it If you work or communicate as a serial port, you must When downloading the program, check "Next cold start P3.2/P3.3 on the software for 0/0 You can download the program at that time".

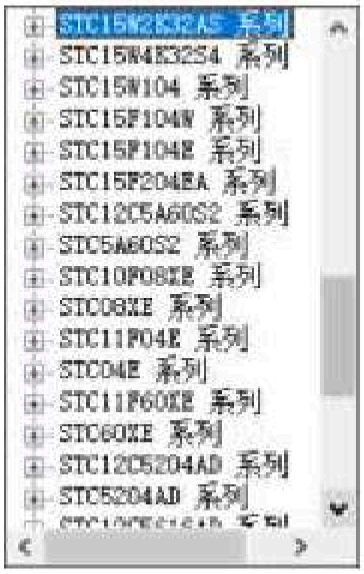
[Note 1]: The burn protection pin of the new chips of STC15, STC8 series and later【Note 1】 is P3.2/P3.3, and the burn protection pin of the earlier chips is P1.0/P1.1.

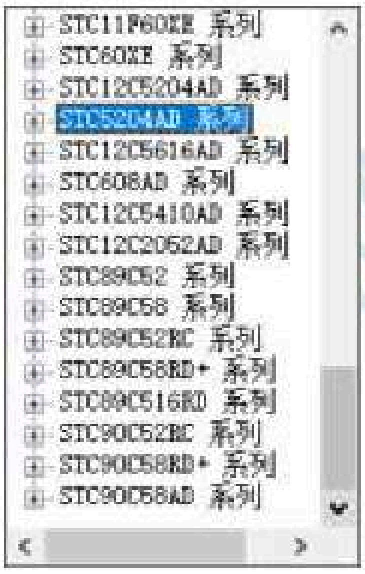
F. 3 USB Type online/Offline download tool U8W/U8W-Mini

U8W/U8W-Min The scope of application can support All current series of STC MCU, Flash Program space and EEPROM Data space is not restricted. Support includes the following and upcoming STC Full range of chips :


常用型号
STC8H3K64S4 系列
STC8H3K64S2 系列
STC8H8K64U 系列
STC8H1K28 系列
STC8H1K08 系列
STC8G2K64S4 系列
STC8G2K64S2 系列
STC8G1K08T 系列
STC8G1K08A-8PIN 系列
STC8G1K08-8PIN 系列
STC8G1K08-20/16PIN
STC8C1K08 系列
STC8F1K08 系列
STC8F1K08S2 系列
STC8F2K64S2 系列


STC8F2K64S2 系列
STC8F2K64S4 系列
STC8A4K64S2A12 系列
STC8A8K64S4A12 系列
STC15F2K60S2 系列
STC15F1K60S2 系列
STC15F408AD 系列
STC15W204S 系列
STC15W1K16S 系列
STC15W408S 系列
STC15W408AS 系列
STC15W1K08PWM 系列
STC15W1K20S 系列
STC15W1K20AS 系列
STC15W2K32S2 系列
STC15W2K32AS 系列


STC15W2K32AS 系列
STC15W4K32S4 系列
STC15W104 系列
STC15F104W 系列
STC15F104E 系列
STC15F204EA 系列
STC12C5A60S2 系列
STC5A60S2 系列
STC10F08XE 系列
STC08XE 系列
STC11F04E 系列
STC04E 系列
STC11F60XE 系列
STC60XE 系列
STC12C5204AD 系列
STC5204AD 系列


STC11F60XE 系列
STC60XE 系列
STC12C5204AD 系列
STC5204AD 系列
STC12C5616AD 系列
STC608AD 系列
STC12C5410AD 系列
STC12C2052AD 系列
STC89C52 系列
STC89C58 系列
STC89C52RC 系列
STC89C58RD+ 系列
STC89C516RD 系列
STC90C52RC 系列
STC90C58RD+ 系列
STC90C58AD 系列

The offline download tool can download work without leaving the computer, and can be used for mass production and remote upgrades. The offline download board can support various functions such as automatic increment, download limit, and transmission after user program encryption. The front and back view of the tool and The picture below shows the front and back pictures : U8W U8W-Mini

U8W正面图 U8W反面图

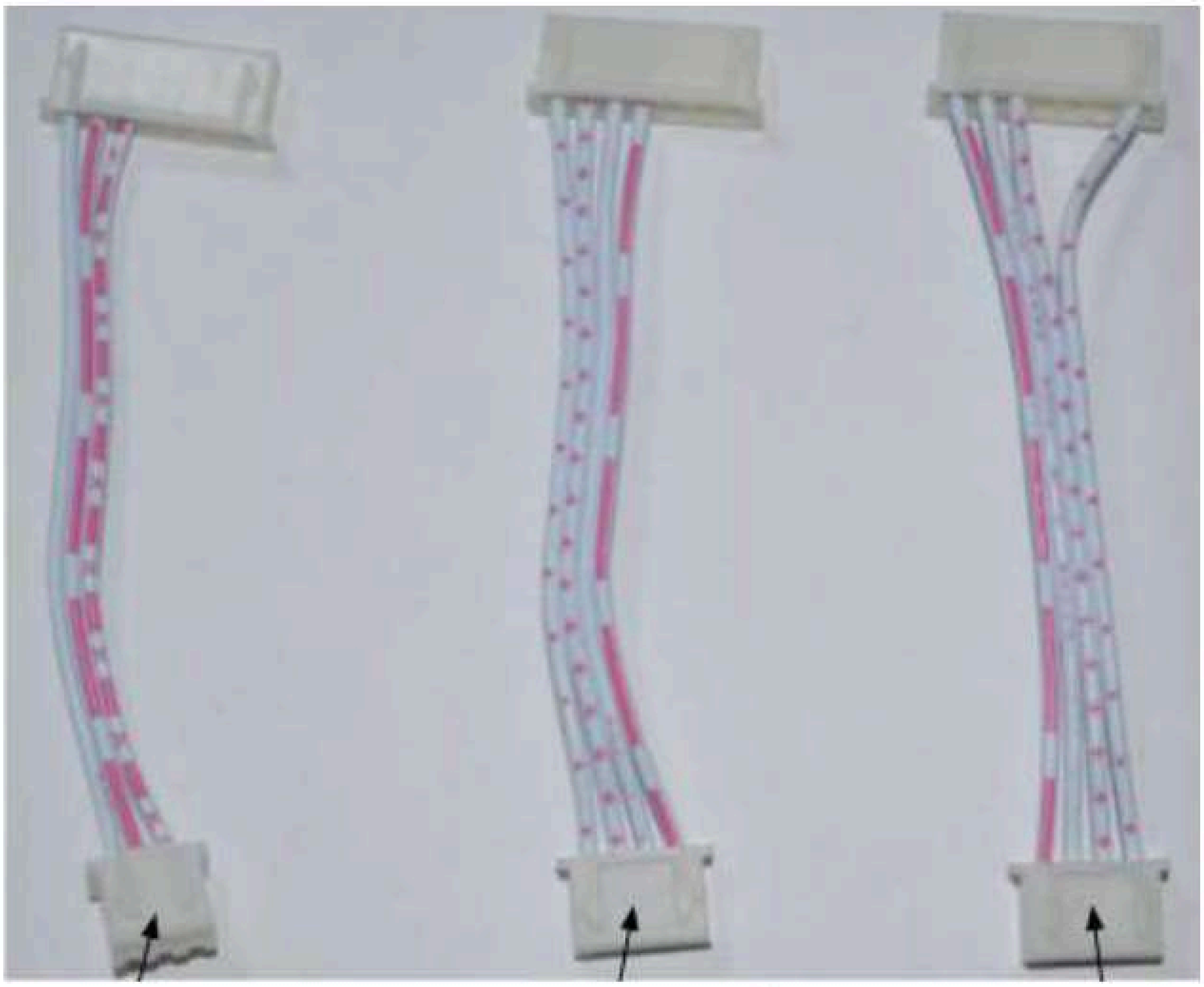
U8W-Mini正反面图

U8W-Mini工具的体积仅有一个U盘大小，其功能与U8W相同，但无锁紧座，价格仅为RMB 50元，欢迎来电订购！

In addition, some of the following wires are used in conjunction with tools, such as: (1) Two male USB cables (shown on the left of the picture below) and USB-Micro cables (shown on the right of the picture below).

Note: This USB Strengthen the line to ensure that the download board is especially customized for our A... Some of the lower-quality male and USB Line, the internal resistance is too large, resulting in a large voltage drop (such as when using 5.0V female ends are connected with inferior wires. USB U8W/U8W-Mini , The voltage to our download board may drop to 4.2V Or lower , As a result, the chip is in a reset state and cannot be successfully downloaded).

(2) The download cable connecting U8W/U8W-Mini to the user system (that is, the cable connecting U8W/U8W-Mini to the target MCU on the user board), as shown in the figure below. :



with U8W/U8W-Mini User systems are independent Power supply cable

give U8W/U8W-Mini Connection to the power supply of the user's system Wiring

The user system gives U8W/U8W-Mini Power supply cable

F. 3.1 install U8W/U8W-Mini driver

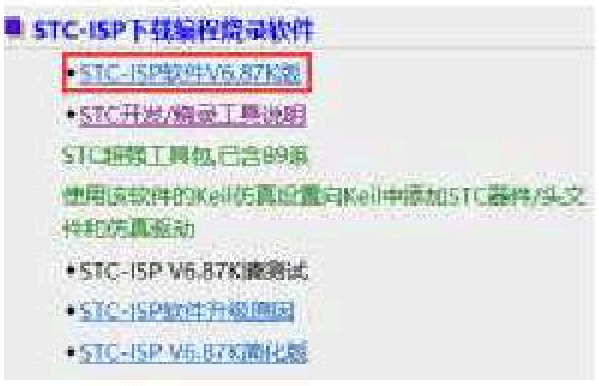
U8W/U8W-Mini One is used on the download board of CH340 USB Go to the serial port universal chip. This saves some power that does not hav

I have to buy an extra one USB It's troublesome to download only by going to the serial port tool. CH340 and others USB The same as the serial port conversion tool, it is i

The driver must be installed before.

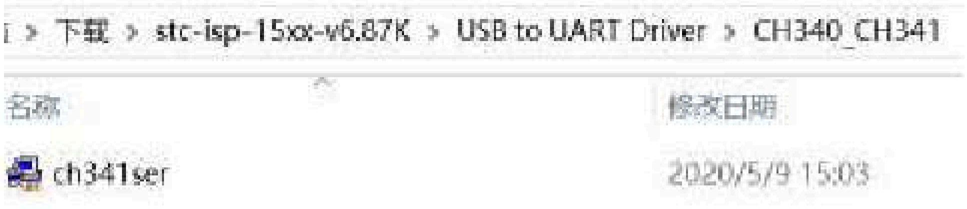
By downloading STC-ISP Package get driver

The following is the official website of STC (www.STCMCUDATA.com) The download location of the STC-ISP software package provided :



■ STC-ISP下载编程烧录软件
• STC-ISP软件V6.87K版
• STC开发/烧录工具说明
STC仿真工具包,已含89系
使用该软件的Keil仿真设置向Keil中添加STC器件/头文件和仿真驱动
• STC-ISP V6.87K版测试
• STC-ISP软件升级原因
• STC-ISP V6.87K简化版

After downloading, unzip it, and the driver installation package path of CH340 is stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341 :



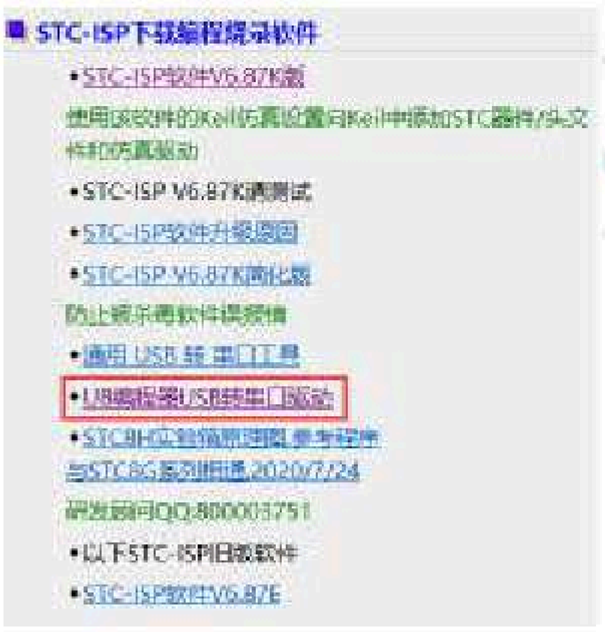
[illegible] > 下载 > stc-isp-15xx-v6.87K > USB to UART Driver > CH340_CH341

名称	修改日期
ch341ser	2020/5/9 15:03

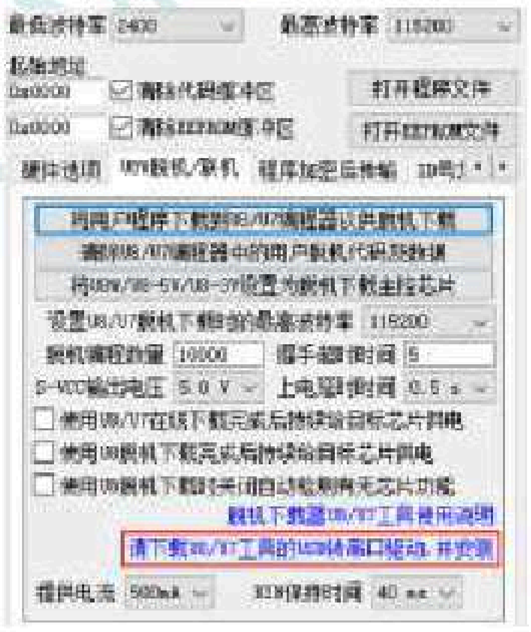
pass STC The official website or in the latest STC-ISP Download the driver manually in the download software

in STC Manually download the driver on the official website or in the latest STC-ISP download software. The download link for the driver is : programming deviceUSB To serial driver (). On the website and on the STC-ISP download software http://www.stcmcu.com/STCISP/CH341SER.exe

The driver address is shown in the figure below :



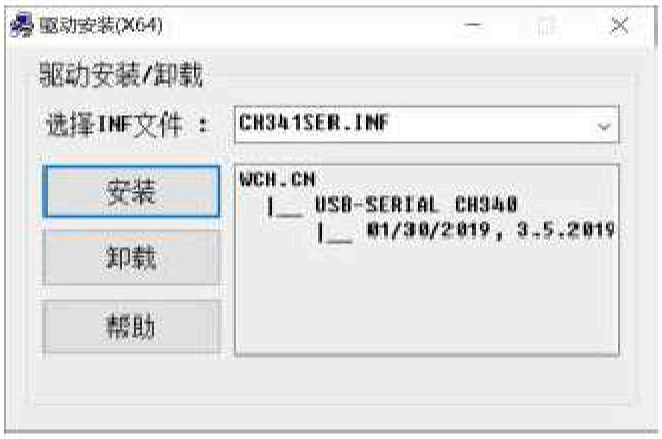
■ STC-ISP下载编程烧录软件
• STC-ISP软件V6.87K版
使用该软件的Keil仿真设置向Keil中添加STC器件/头文件和仿真驱动
• STC-ISP V6.87K版测试
• STC-ISP软件升级原因
• STC-ISP V6.87K简化版
防止杀毒软件误报
• 通用 USB 转 串口工具
• U8编程器USB转串口驱动
• STC8H实验箱原理图,参考程序
与STC8G系列相通,2020/7/24
研发顾问QQ:800003751
• 以下STC-ISP旧版软件
• STC-ISP软件V6.87E



最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件
硬件选项　U8W脱机/联机　程序加密后传输　ID号
将用户程序下载到U8/U7编程器以供脱机下载
清除U8/U7编程器中的用户脱机代码及数据
将U8W/U8-5V/U8-3V设置为脱机下载主控芯片
设置U8/U7脱机下载时的最高波特率 115200
脱机编程数量 10000　握手超时时间 5
S-VCC输出电压 5.0 V　上电延时时间 0.5 s
☐ 使用U8/U7在线下载完成后持续给目标芯片供电
☐ 使用U8脱机下载完成后持续给目标芯片供电
☐ 使用U8脱机下载时关闭自动检测目标芯片功能
脱机下载器U8/U7工具使用说明
请下载U8/U7工具的USB转串口驱动,并安装
提供电流 500mA　[illegible]保持时间 40 ms

install U8W/U8W-Mini The driver

After the driver is downloaded to the machine, double-click the executable program directly and run it. The interface shown in the figure below appears, and click the "Install" button to start automatically installing the driver. :

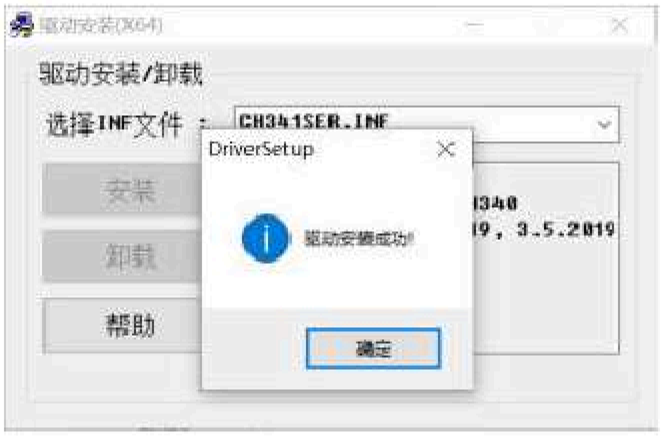

驱动安装(X64)

驱动安装/卸载

选择INF文件 : CH341SER.INF

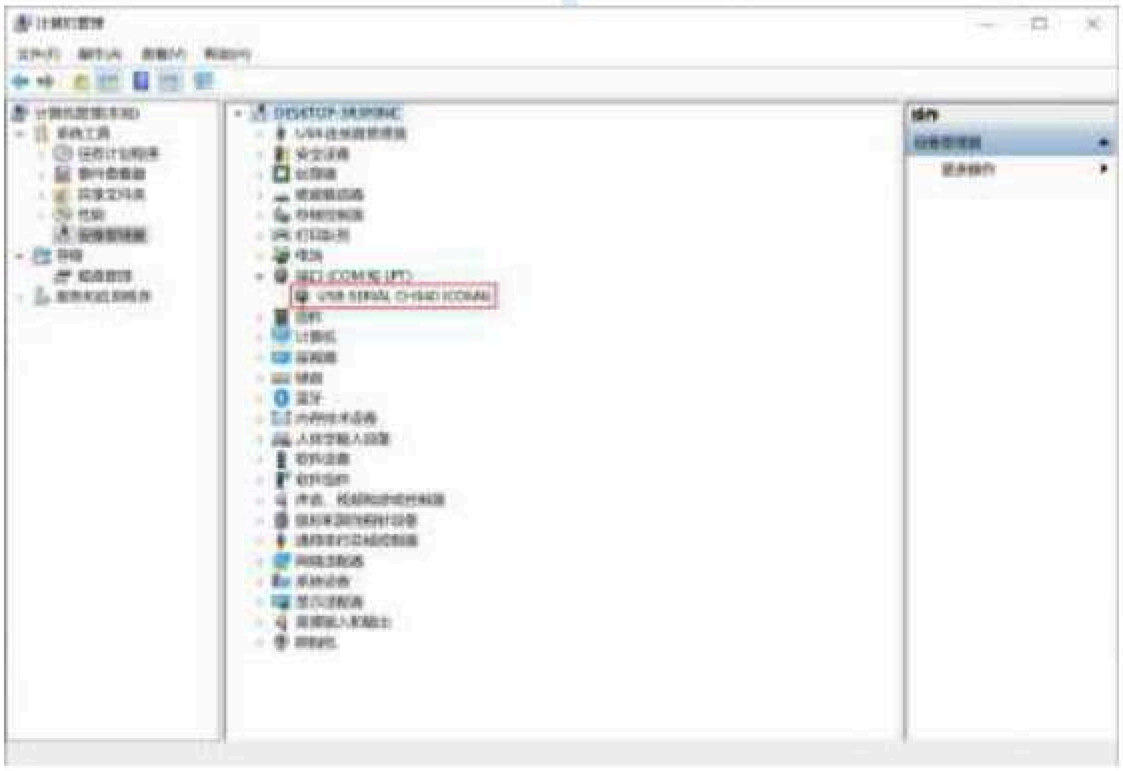
WCH.CN
 |__ USB-SERIAL CH340
  |__ 01/30/2019, 3.5.2019

Then the driver installation success dialog box pops up, click the "OK" button to complete the installation :

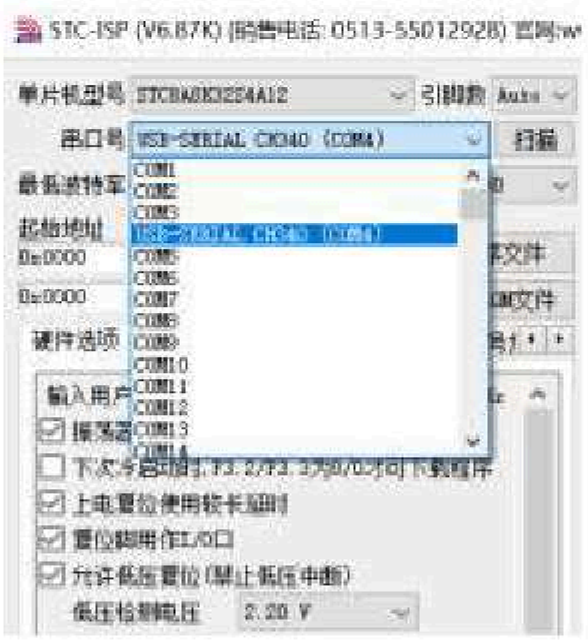

DriverSetup

驱动安装成功!

Then use STC Provided by USB The cable will U8W/U8W-Mini Connect the download board to the computer, open the computer's device Under the port device class, if there is something similar "USB The device of SERIAL CH340 (COMx)" means U8W/U8W-Mini Can be normal Used it. As shown in the figure below (The serial port number may be different for different computers) :



Note: Use it later STC-ISP When downloading the software, the selected serial port number must select the corresponding serial port number, as s


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:w

单片机型号 STC8A8K32S4A12 引脚数 Auto

串口号 USB-SERIAL CH340 (COM4) 扫描

最低波特率

起始地址

0x0000

0x0000

硬件选项

COM1
COM2
COM3
USB-SERIAL CH340 (COM4)
COM5
COM6
COM7
COM8
COM9
COM10
COM11
COM12
COM13
COM14

输入用户

☑ 振荡器

☐ 下次冷启动时, P3.2/P3.3为0/0才可下载程序

☑ 上电复位使用较长延时

☑ 复位脚用作I/O口

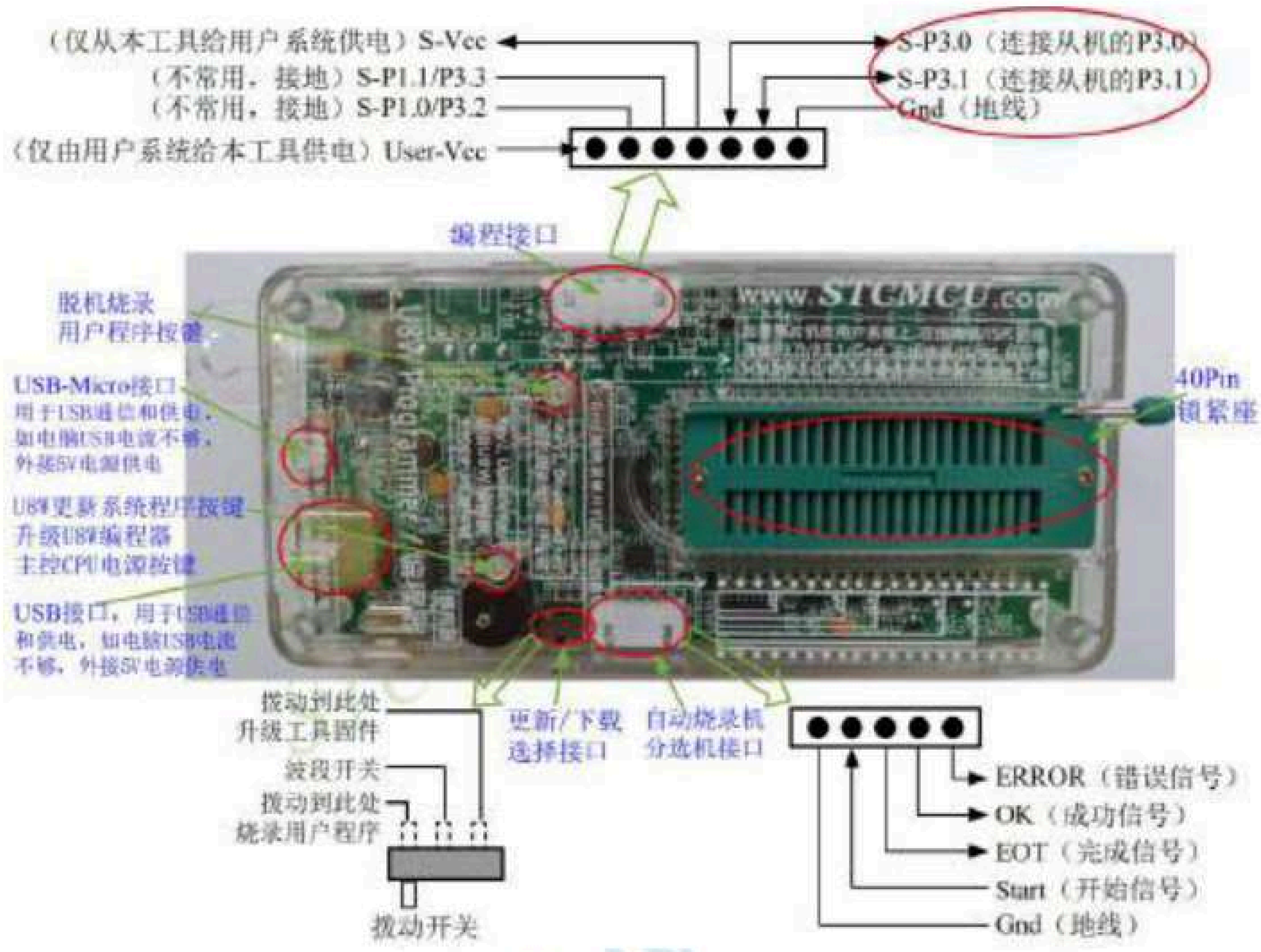
☑ 允许低压复位(禁止低压中断)

低压检测电压 2.20 V

F. 3.2 U8W Introduction to the function of

Described in detail below U8W The main interfaces and functions of the tool :

If the microcontroller is on the user's system, burn it online /ISP When it is necessary to connect, the target P3.0/P3.1/Gnd, Online burning /ISP Honey, don't connect to any other line P3.0/P3.1


(仅从本工具给用户系统供电) S-Vcc
(不常用，接地) S-P1.1/P3.3
(不常用，接地) S-P1.0/P3.2
(仅由用户系统给本工具供电) User-Vcc
S-P3.0 (连接从机的P3.0)
S-P3.1 (连接从机的P3.1)
Gnd (地线)
编程接口
www.STCMCU.com
脱机烧录 用户程序按键
USB-Micro接口 用于USB通信和供电，如电脑USB电流不够，外接5V电源供电
U8W更新系统程序按键 升级U8W编程器 主控CPU电源按键
USB接口，用于USB通信和供电，如电脑USB电流不够，外接5V电源供电
40Pin 锁紧座
拨动到此处 升级工具固件
波段开关
拨动到此处 烧录用户程序
拨动开关
更新/下载 选择接口
自动烧录机 分选机接口
ERROR (错误信号)
OK (成功信号)
EOT (完成信号)
Start (开始信号)
Gnd (地线)

Programming interface: According to different power supply methods, use different Download board and user system. When the

U8W Update system program button: Used for updating U8W tools, when there is a new U8W version of firmware is installed, you need to press this button to pair

The main control chip is updated (Note: You must update it first Download the toggle switch on the selection interface and toggle to the upgra

Offline download user program button: Start offline download button. First download the offline code to the board, and then use the downloa

Connect the user system to U8W the cable, and then press this button to start the offline download (the user code will be downloaded immediately

Update/download selection interface: When you need to upgrade the underlying firmware, you need to toggle this toggle switch to the firmwa

If you need to program the target chip through U8W, you need to dial the toggle switch to the programming user program.

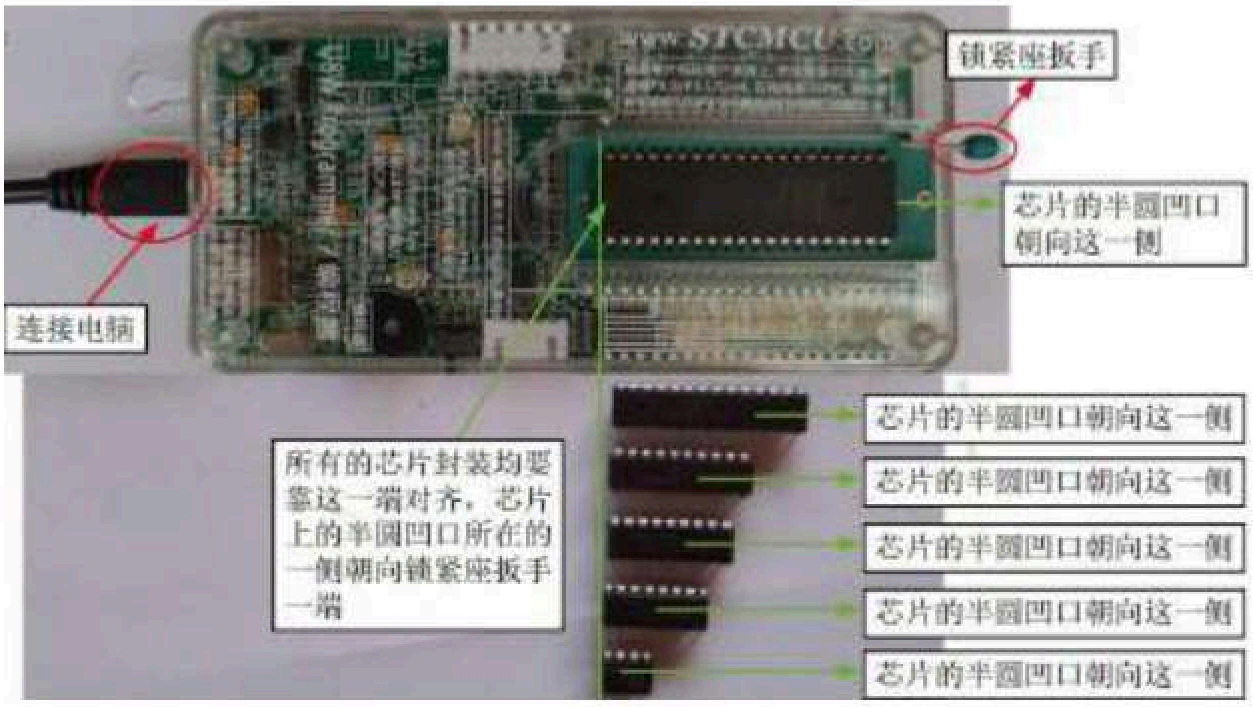
(Please refer to the picture above for the connection method of the toggle switch)

Automatic burner/sorter interface: It is a control interface used to control the automatic burner/sorter for automatic production.

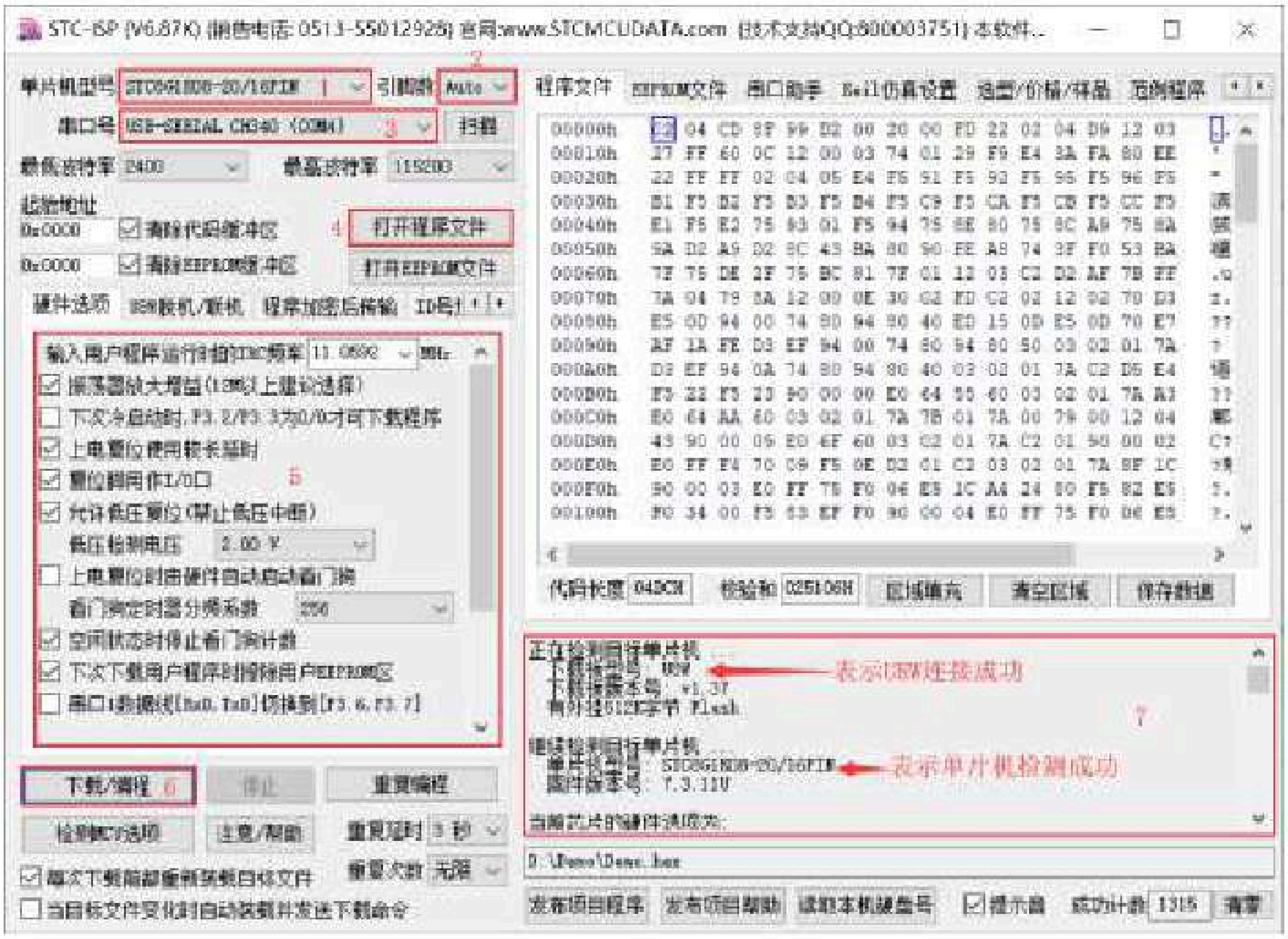
F. 3.3 U8W Online download instructions for use

The target chip is installed in U8W Lock the seat and make it U8W Connect to a computer for online download

First use the USB cable provided by STC to connect the U8W to the computer, and then install the target MCU on the U8W in the direction shown in the figure below. :



Then use STC-ISP Download the software download program, the steps are as follows：


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件...

单片机型号 STC8G1K08-20/16PIN 1
引脚数 Auto 2
串口号 USB-SERIAL CH340 (COM4) 3
最低波特率 2400
最高波特率 115200
起始地址 0x0000 ☑ 清除代码缓冲区
0x0000 ☑ 清除EEPROM缓冲区
4 打开程序文件
打开EEPROM文件

硬件选项 5
输入用户程序运行时的IRC频率 11.0592 MHz
☑ 振荡器放大增益(12M以上建议选择)
☐ 下次冷启动时，P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.00 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区
☐ 串口1数据线[RxD, TxD]切换到[P3.6, P3.7]

下载/编程 6
重复延时 3 秒
重复次数 无限
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

代码长度 042CH 校验和 0251D6H

7
正在检测目标单片机 ...
下载板型号 : U8W
下载板版本号 : v1.37
有外挂512K字节 Flash

继续检测目标单片机 ...
单片机型号 : STC8G1K08-20/16PIN
固件版本号 : 7.3.11U

当前芯片的硬件选项为:

D:\Demo\Demo.hex

成功计数 1315

1 Select the MCU model; 2 select

the number of pins. When the chip is installed directly on the U8W to download, you must pay attention to selecting the correct number of pins, otherwise the download will fail; 3 select the serial port number corresponding to the U8W;

4 Open the target file (HEX format or BIN format);

5 Set the hardware options;

6 clickThe "Download/program" button starts the burning;

7 Displays the step information of the burning process, and the burning is completed with a prompt "The operation is successful!" ".

When the information box has the version number information of the output download board and the plug-in Flash U8W Download the tool. When the corresponding information is obtained, it means that it has been d

In the process of downloading, the download tool 4 LED Will be displayed in marquee mode. After the download is complete, if the download is

U8W Will light up at the same time and turn off at the same time; if the download fails, then a 4 All are on

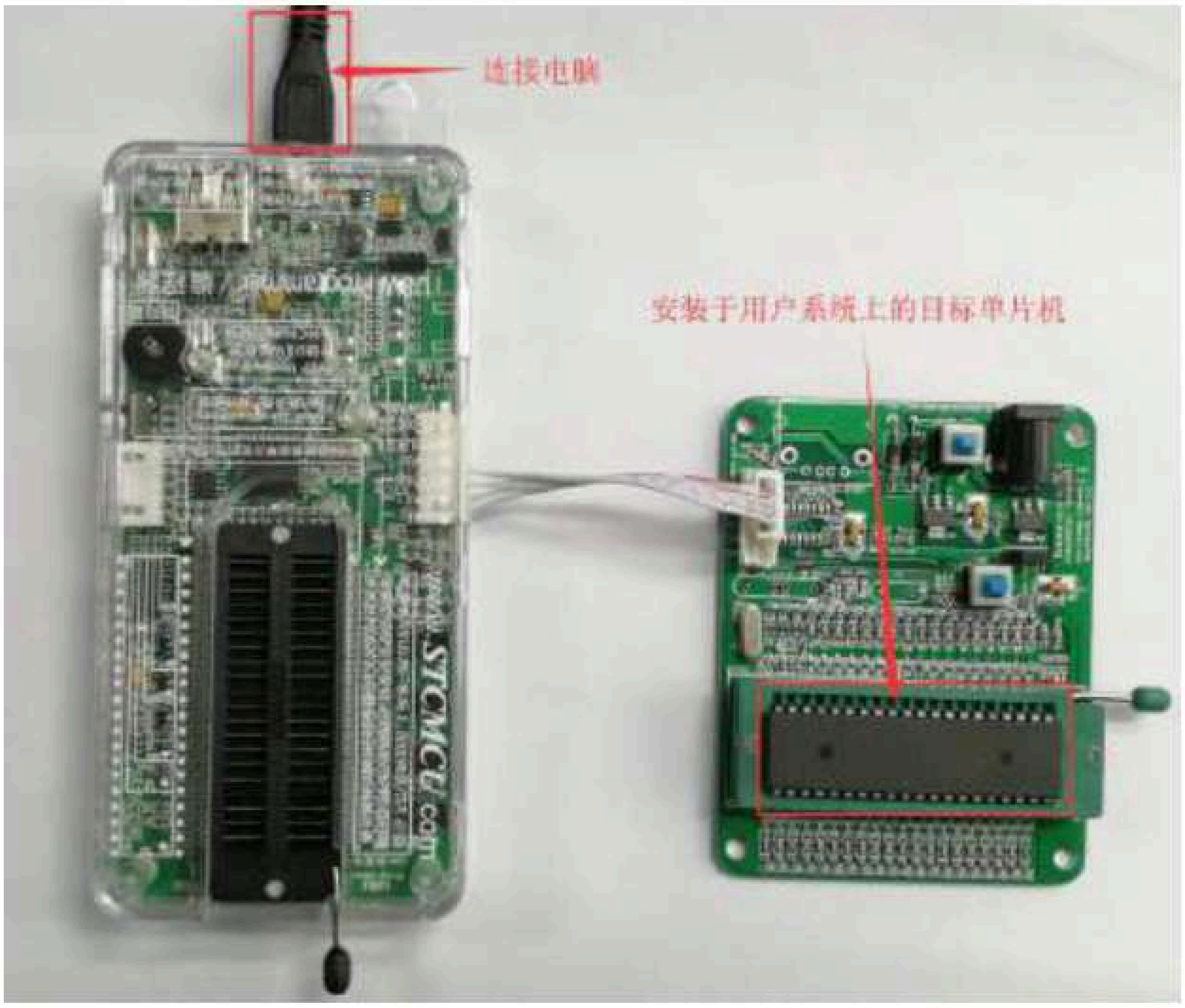
a LED STC-ISP Download the software (please feel free to pay attention Official website www.STCMCUDATA.com Download the

It is recommended that users use To download the software update, it is strongly recommended that users go to the official website STC http://www.STCMCUDATA.com latest version from Zhongzhong the latest version of STC-ISP

Software use).

The target chip is connected through the lead of the user system U8W And then U8W Connect to a computer for online download

First, use the USB cable provided by STC to connect the U8W to the computer, and then connect the U8W to the target monolithic machine of the user's system through the download cable . The connection method is shown in the figure below. :



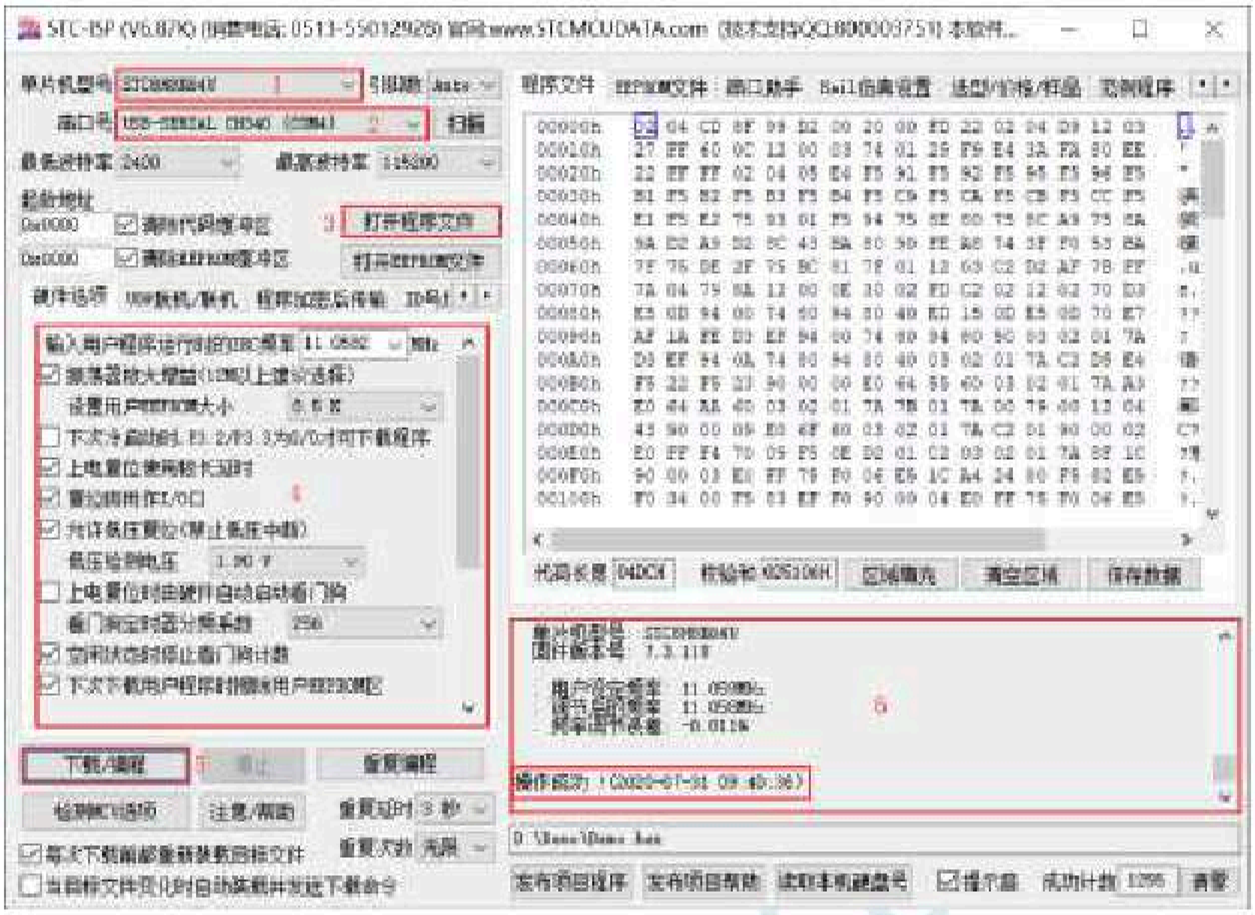
Then use Download the software download program, the steps are as follows : STC-ISP

1. Select the MCU model ;

2. Select the serial port number corresponding to U8W; 3. Open the target file (HEX format or BIN format); 4. Set hardware options;

5. Click the "Download/program" button to start burning; 6.

The step information of the burning process is displayed, and the burning is completed with a prompt "The operation was successful!"


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网 www.STCMCUDATA.com (技术支持QQ:800003751) 本软件...

单片机型号 STC8H8K64U 1 引脚数 Auto
串口号 USB-SERIAL CH340 (COM4) 2 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区 3 打开程序文件
0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 U8W脱机/联机 程序加密后传输 ID号
输入用户程序运行时的IRC频率 11.0592 MHz
☑ 振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小
☐ 下次冷启动时, P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 1.90 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区
4
下载/编程 5 停止 重复编程
检测MCU选项 注意/帮助 重复延时 3 秒
☑ 每次下载前都重新装载目标文件 重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

程序文件 EEPROM文件 串口助手 Keil仿真设置 选型/价格/样品 范例程序
代码长度 04DC4 校验和 025106H 区域填充 清空区域 保存数据

单片机型号 STC8H8K64U
固件版本号 7.3.11U
用户设定频率 11.0592MHz
调节后的频率 11.058MHz
频率调节误差 -0.011%
6
操作成功！(2020-07-31 09:40:36)

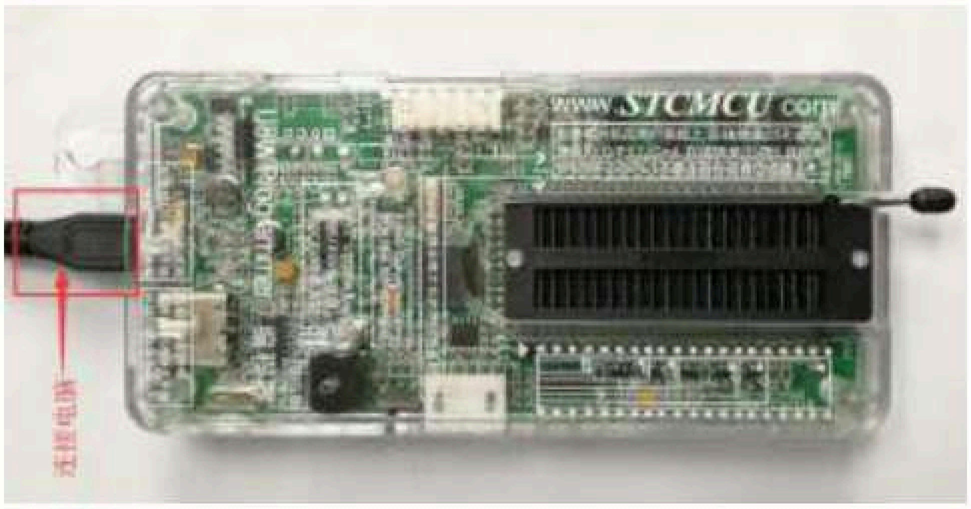
发布项目程序 发布项目帮助 读取本机硬盘号 ☑ 提示音 成功计数 1295 清零

When the information box has the version number information of the output download board and the plug-in, it means that it has been d Download the tool U8W
In the process of downloading， U8W 4 Download the one on the tool Will be displayed in marquee mode. After the download is 4
One will light up and turn off at the same time; if the download fails, if the download is successful, all of them will not light up. LED completed
It is recommended that users use the latest version of the download software (please feel free to pay attention Official website STC-ISP Download the
To download the software update, it is strongly recommended that users go to the official website STC http://www.STCMCUDATA.com latest version from Zhongzhong STC-ISP
Software use).

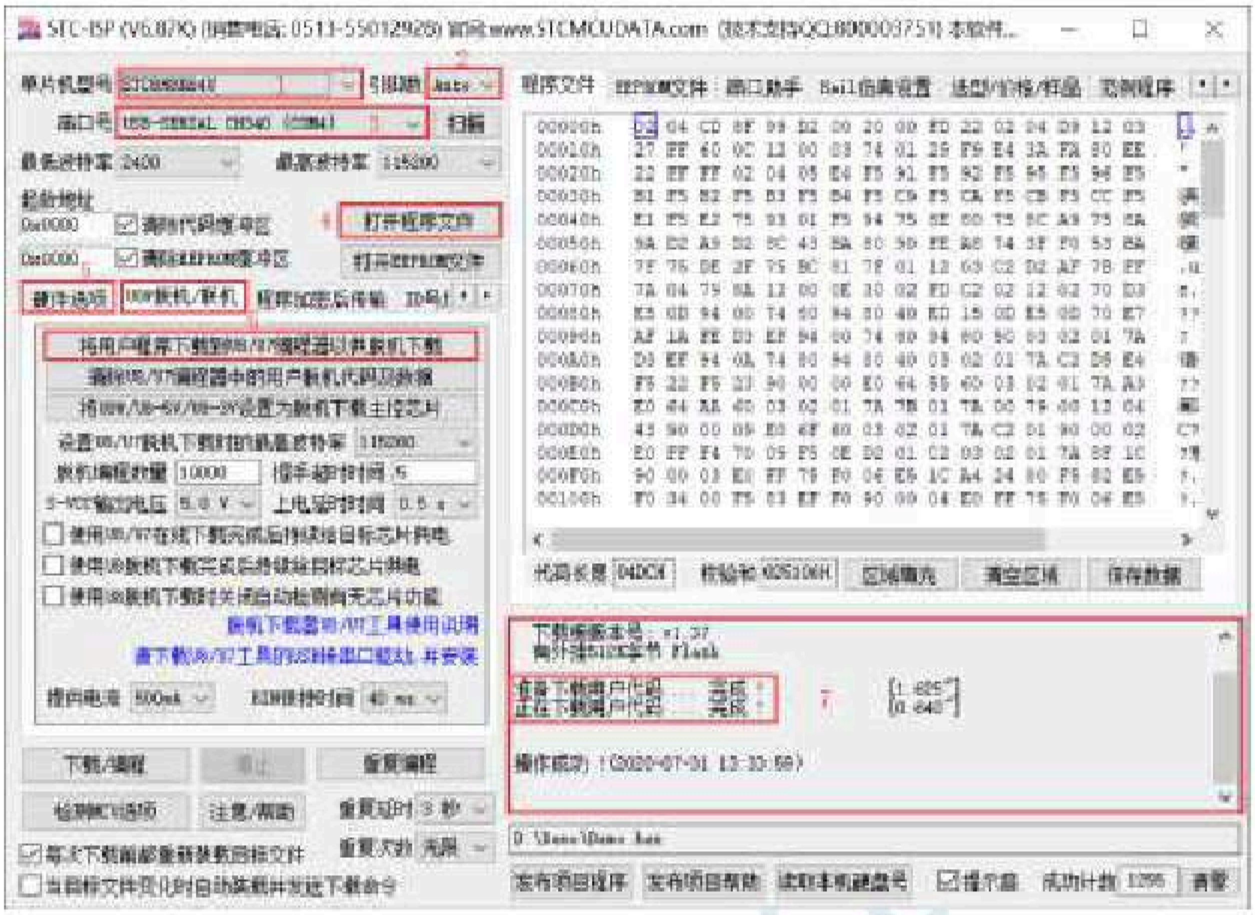
F. 3.4 U8W Offline download instructions for use

The target chip is installed in U8W The seat is locked and passed USB Connect to the computer to U8W Power supply for offline download

The steps to use USB to power the U8W for offline download are as follows: (1) Use the USB cable provided by STC to connect the U8W download board to the computer, as shown in the figure below. :



2() In STC-ISP In the download software, follow the steps shown in the figure below to set it up :



1. Select the MICROCONTROLLER model;

2. Select the number of pins. When the chip is installed directly on the U8W and downloaded, you must pay attention to selecting the correct number of pins, otherwise the download will fail; 3.4 Select the serial port number corresponding to U8W ;

. Open the target file (HEX format or BIN format);

5. Set hardware options; 6. Select the "U8W

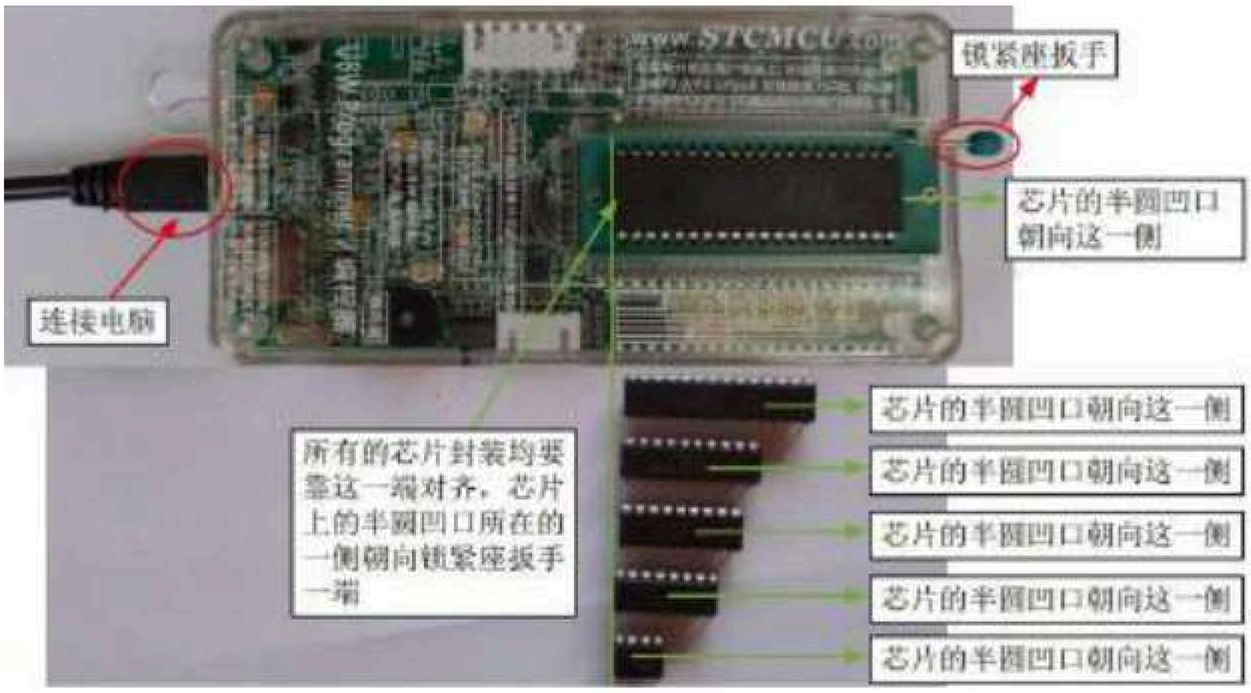
Offline/Online" tab, set the offline programming option, and note that the output voltage of S-VCC matches the operating voltage of the target chip. ;

7. Click the "Download the user program to the U8/U7 programmer for offline download" button; the step information of the setup process is displayed, and the setup is complete with a prompt "The operation is successful!" ".

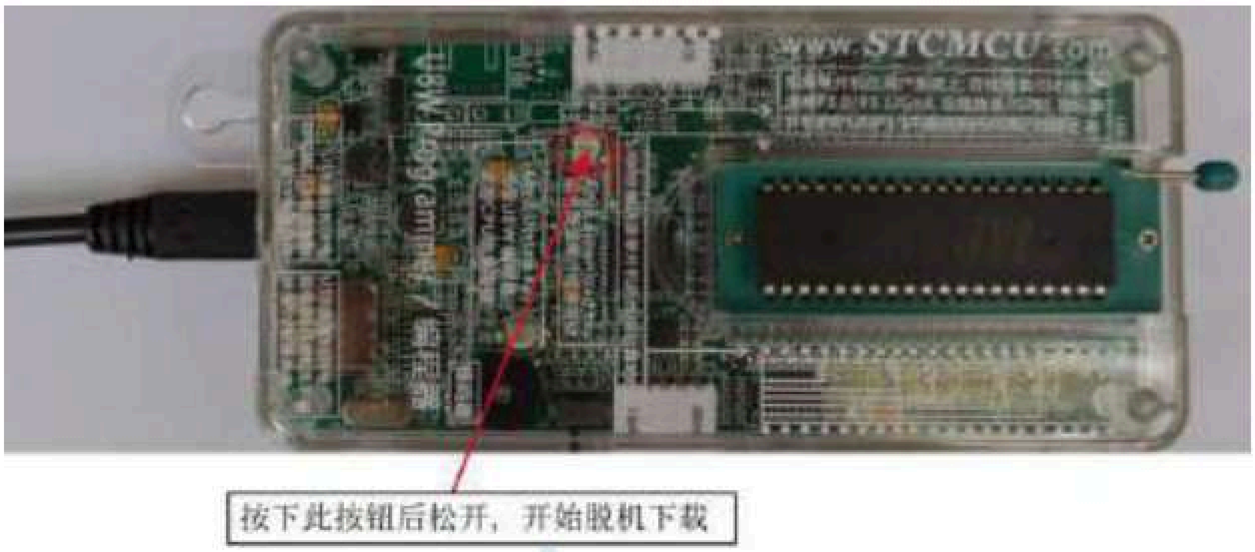
Follow the steps in the figure above. After the operation is completed, if the download is successful, it means that the user code and related s Download work In the tool.

It is recommended that users use the latest version of the download software (please feel free to pay attention to the official website the Download the ISP STC I To download the software update, it is strongly recommended that users go to the official website latest version from Zhongzhong http://www.STCMCUDATA.com STC-ISP Software use)

(3) Then place the target MCU in the U8W download tool in the direction shown in the figure below, as shown in the figure below. :



(4) Then press the button shown in the figure below and release it to start the offline download. :



In the process of downloading， LED 4 Download tool will be displayed in marquee mode. After the download is complete, if the download is successful LED U8W Will light up and turn off at the same time; if the download fails, tool one by one, then not on.

Offline download Plug and play Introduction

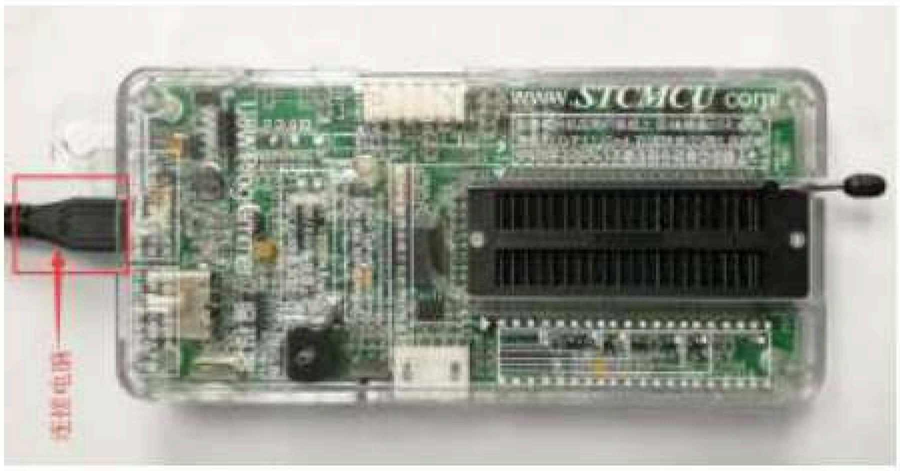
to the burning function: 1. After the above steps (1) and (2) are completed, the U8W is connected to the computer and is in the plug-and-play burning state by default when powered on; 2. Follow the instructions in step (3) to put the chip into the burning seat. While locking the seat wrench, U8W will automatically start burning; 3. Display the burning process and burning results through the indicator light;

4. After the burning is complete, release the seat wrench and remove the chip;

5. Repeat steps 2, 3, and 4 for continuous burning, eliminating the need to press the button to trigger the burning action.

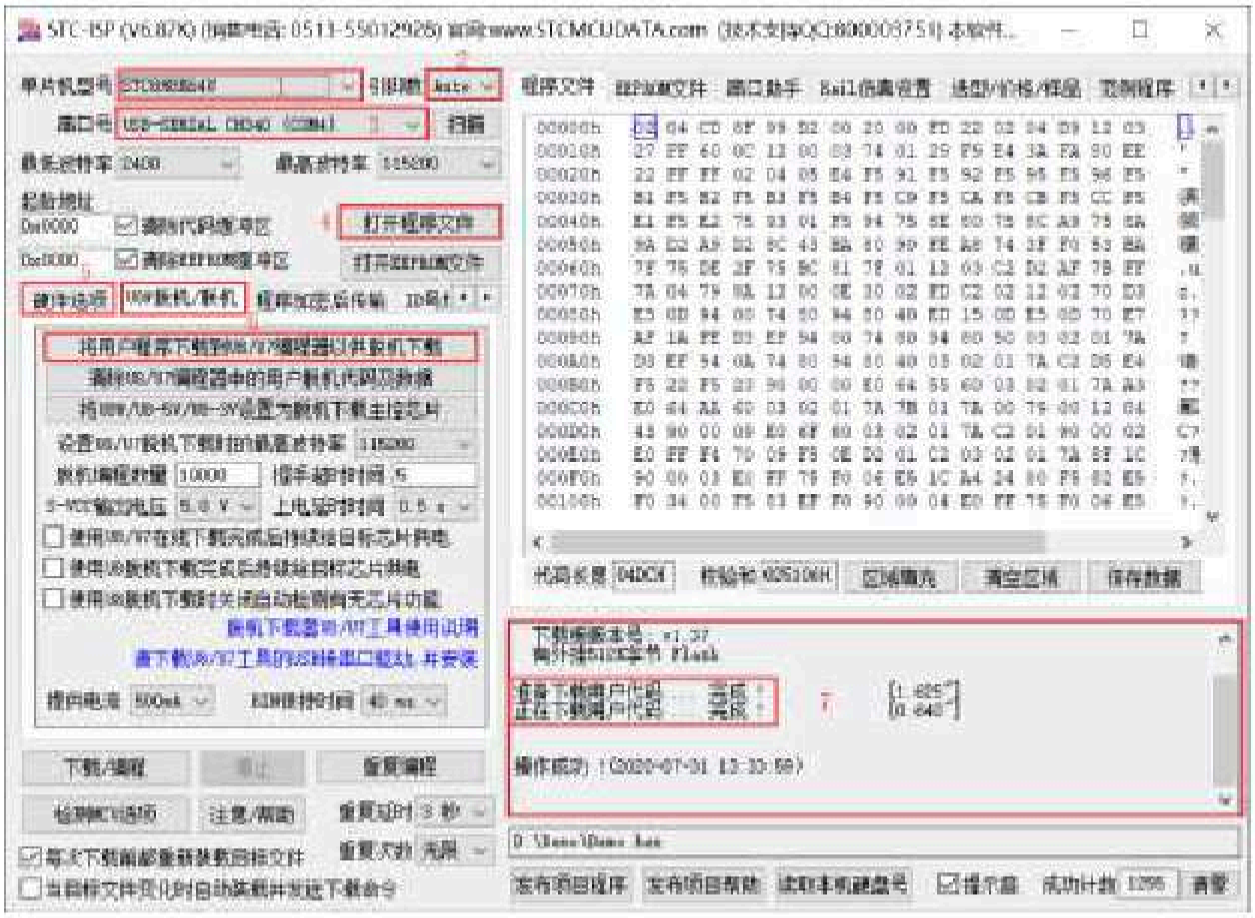
The target chip is connected by the lead of the user system U8W Through pass USB Connect to the computer to U8W Power supply for offline download

The steps to use USB to power the U8W for offline download are as follows: (1) Use the USB cable provided by STC to connect the U8W download board to the computer, as shown in the figure below. :



（2）In STC-ISP In the download software, follow the steps shown in the figure below to set it up：

It is recommended that users STC-ISP Download the software (please feel free to pay attention Official website STC http://www.STCMCUDATA.com Download the use the latest version of STC-ISP To download the software update, it is strongly recommended that users go to the official website latest version from Zhongzhong Software use).


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网www.STCMCUDATA.com (技术支持QQ:800003751) 本软件...

单片机型号 STC8H8K64U　引脚数 Auto
串口号 USB-SERIAL CH340 (COM4)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件
硬件选项　U8W脱机/联机　程序加密后传输　ID号 ◂ ▸
将用户程序下载到U8/U7编程器以供脱机下载
擦除U8/U7编程器中的用户脱机代码及数据
将U8W/U8-5W/U8-5Y设置为脱机下载主控芯片
设置U8/U7脱机下载时的最高波特率 115200
脱机编程数量 10000　相邻编程时间 5
S-VCC输出电压 5.0 V　上电延时时间 0.5 s
☐ 使用U8/U7在线下载完成后持续给目标芯片供电
☐ 使用U8脱机下载完成后持续给目标芯片供电
☐ 使用U8脱机下载时关闭自动检测有无芯片功能
脱机下载器U8/U7工具使用说明
请下载U8/U7工具的USB转串口驱动 并安装
提供电流 500mA　ISP保持时间 40 ms
下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 3 秒
☑ 每次下载前都重新装载目标文件　重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

程序文件　EEPROM文件　串口助手　Keil仿真设置　选型/价格/样品　范例程序

[illegible]

代码长度 04DCH　校验和 0252DH　区域填充　清空区域　保存数据

下载器版本号：v1.37
内部存储512K字节 Flash
正在下载用户代码 ... 完成
正在下载用户代码 ... 完成
[1.625"]
[0.640"]
操作成功！(2020-07-01 13:33:58)

D:\Demo\Demo.hex
发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 1296　清零

1. Select the MICROCONTROLLER model;

2. Select the number of pins. When the chip is installed directly on the U8W and downloaded, you must pay attention to selecting the correct number of pins, otherwiseThe download will fail; 3. Select the serial port number corresponding to U8W;

4.5 Open the target file (HEX format or BIN format)；

. Set hardware options; 6. Select the "U8W

Offline/Online" tab, set the offline programming option, and note that the output voltage of S-VCC matches the operating voltage of the target chip.；

Click the "Download user program to U8/U7 programmer for offline download" button；

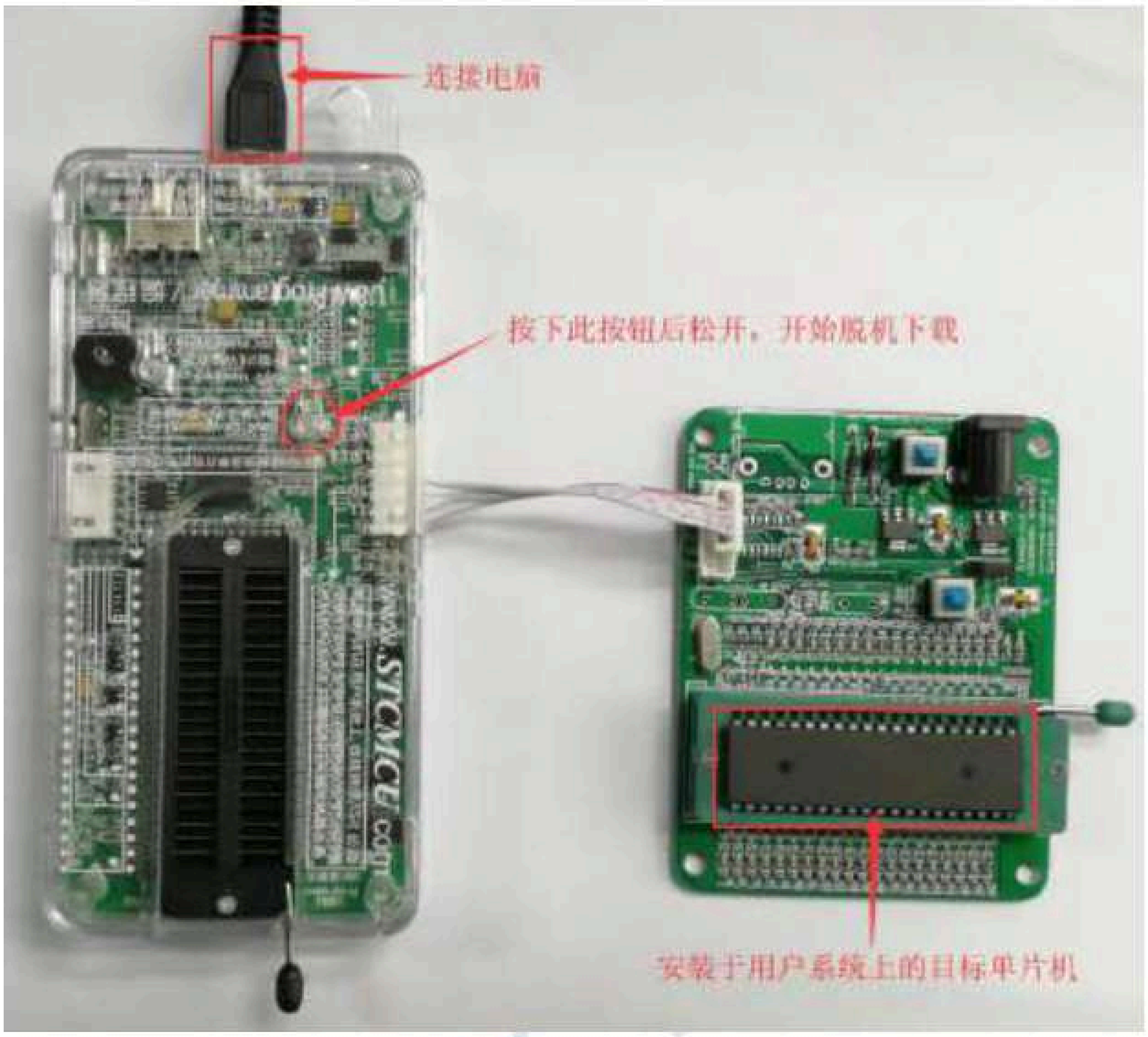
7. The step information of the setup process is displayed, and the setup is complete with a prompt "The operation was successful!" ".

Follow the steps in the figure above. After the operation is completed, if the download is successful, it means U8W that the user code and related s Download work

In the tool. (3) Then use the cable to connect to the computer,

connect the U8W download tool and the user system (target microcontroller) in the manner shown in the figure below

, and press the button shown in the figure and release it to start the offline download.：



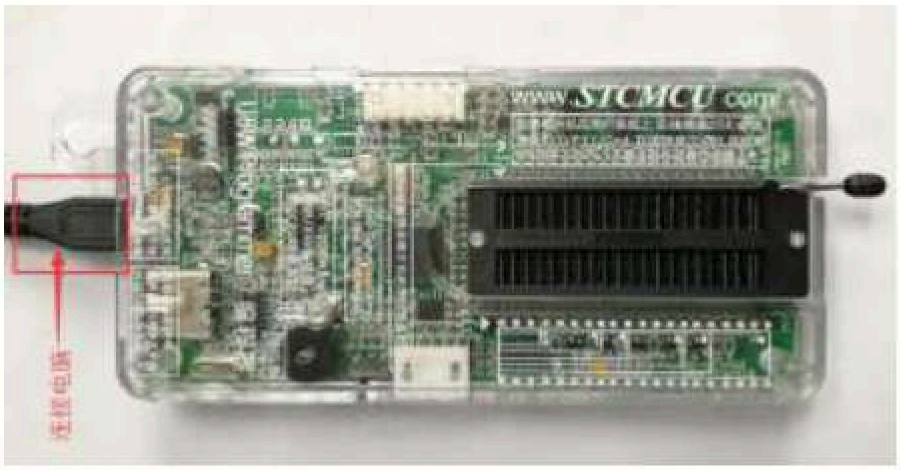
In the process of downloading, Download the one on the tool 4 LED Will be displayed in marquee mode. After the download is complete, if the download is

U8W LED Will light up at the same time and turn off at the same time; if the download fails, then a 4 LED All are not on.

a

The target chip is connected by the lead of the user system U8W And through the user system to U8W Power supply for offline download

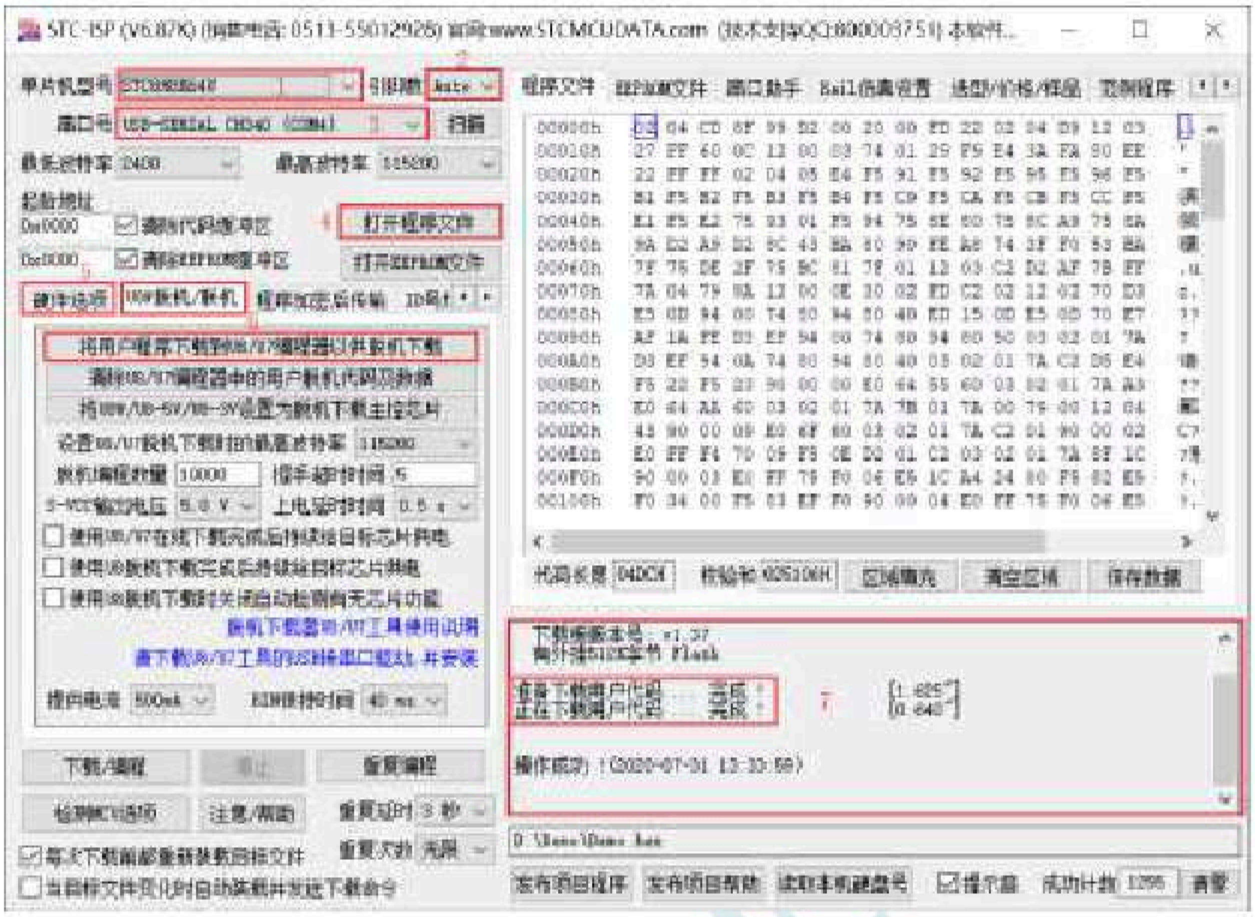
(1) First use the USB cable provided by STC to connect the U8W download board to the computer, as shown in the figure below. :



(2) In STC-ISP In the download software, follow the steps shown in the figure below to set it up :

It is recommended that users use STC-ISP Download the software (please feel free to pay attention Official website STC http://www.STCMCUDATA.com Download the the latest version of STC-ISP To download the software update, it is strongly recommended that users go to the official website STC http://www.STCMCUDATA.com latest version from Zhongzhong

Software use)



```
下载器版本号 : v1.37
剩余空间 : 1024 KB字节 Flash
正在下载用户代码 ... 完成 !  [1.625"]
正在下载用户代码 ... 完成 !  [0.640"]
操作成功 !(2020-07-31 13:33:59)
```

1. Select the MICROCONTROLLER model; 2. Select the number of pins. When the chip is installed directly on the U8W to download, you must pay attention to selecting the correct number of pins, otherwise the download will fail; 3. Select the serial port number corresponding to U8W;

4. Open the target file (HEX format or BIN format);

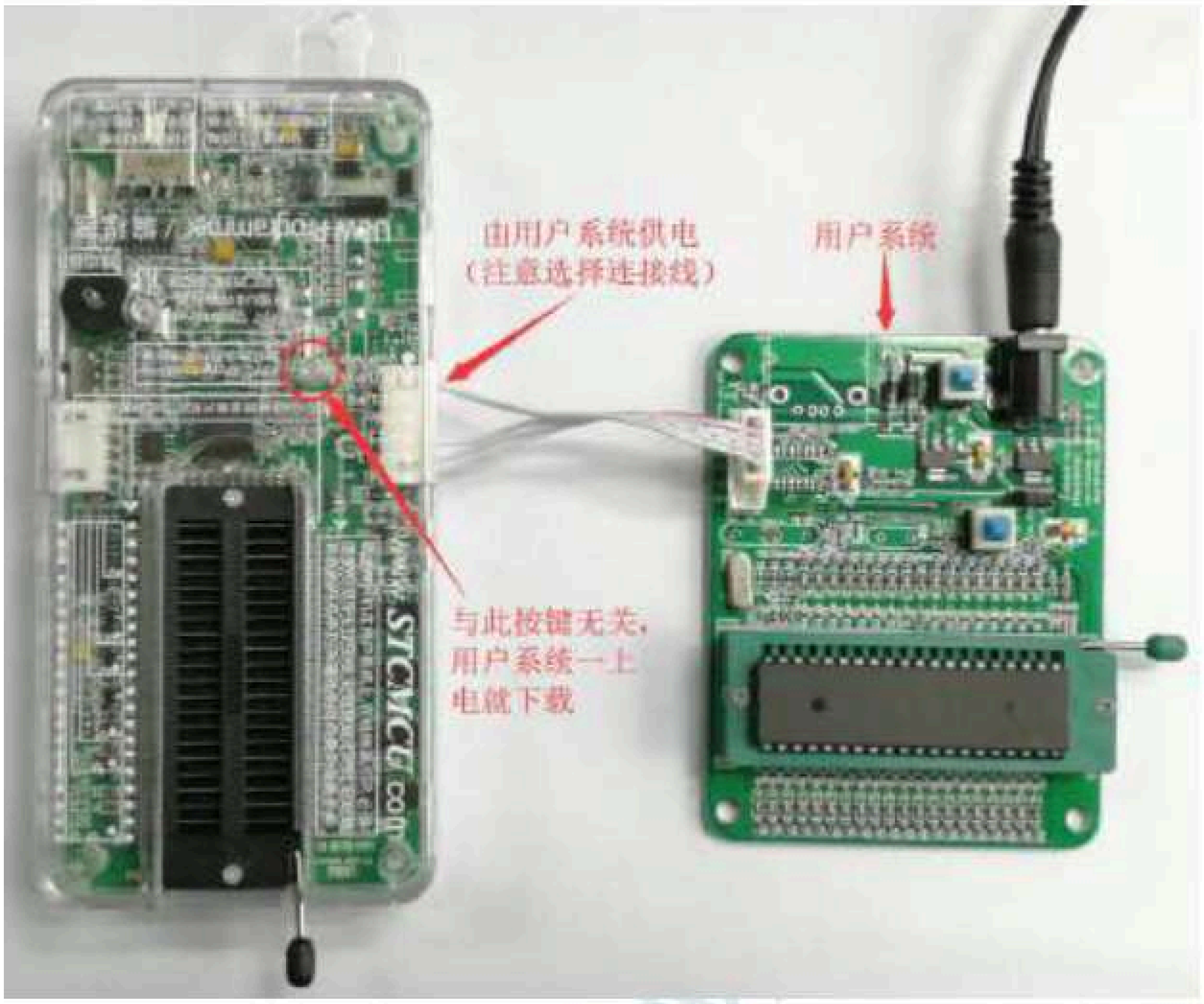
5. Set hardware options; 6. Select the "U8W Offline/Online" tab, set the offline programming option, and note that the output voltage of S-VCC matches the operating voltage of the target chip. ;

Click the "Download user program to U8/U7 programmer for offline download" button ;

7. The step information of the setup process is displayed, and the setup is complete with a prompt "The operation was successful!" ".

Follow the steps in the figure above. After the operation is completed, if the download is successful, it means that the user code and related setting option have been downloaded to the U8W download tool . (3) Then

connect the U8W to the user system in the manner shown in the figure below, supply power to the user system, and you can start the offline download. :



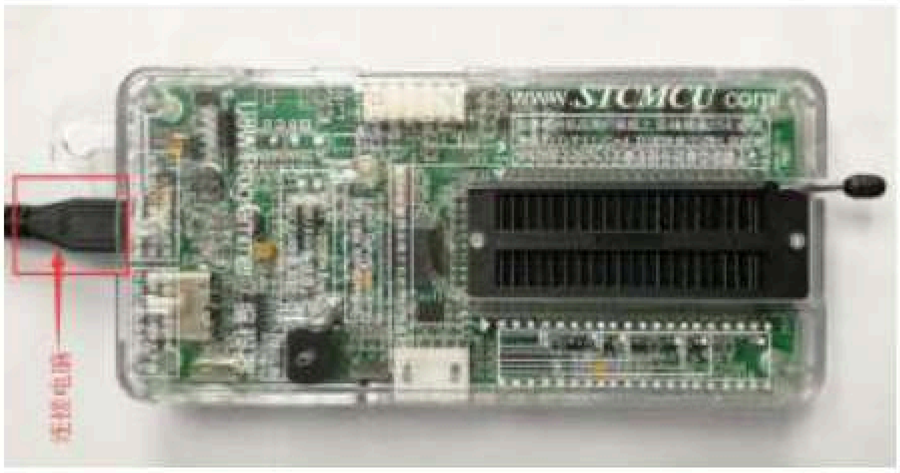
In the process of downloading , Download the one on the tool 4 LED Will be displayed in marquee mode. After the download is complete, if the download is 4

U8W LED Will light up at the same time and turn off at the same time; if the download fails, then a 4 LED All are not on.

a

The target chip is connected by the lead of the user system U8W and U8W Independent power supply with the user system for offline download

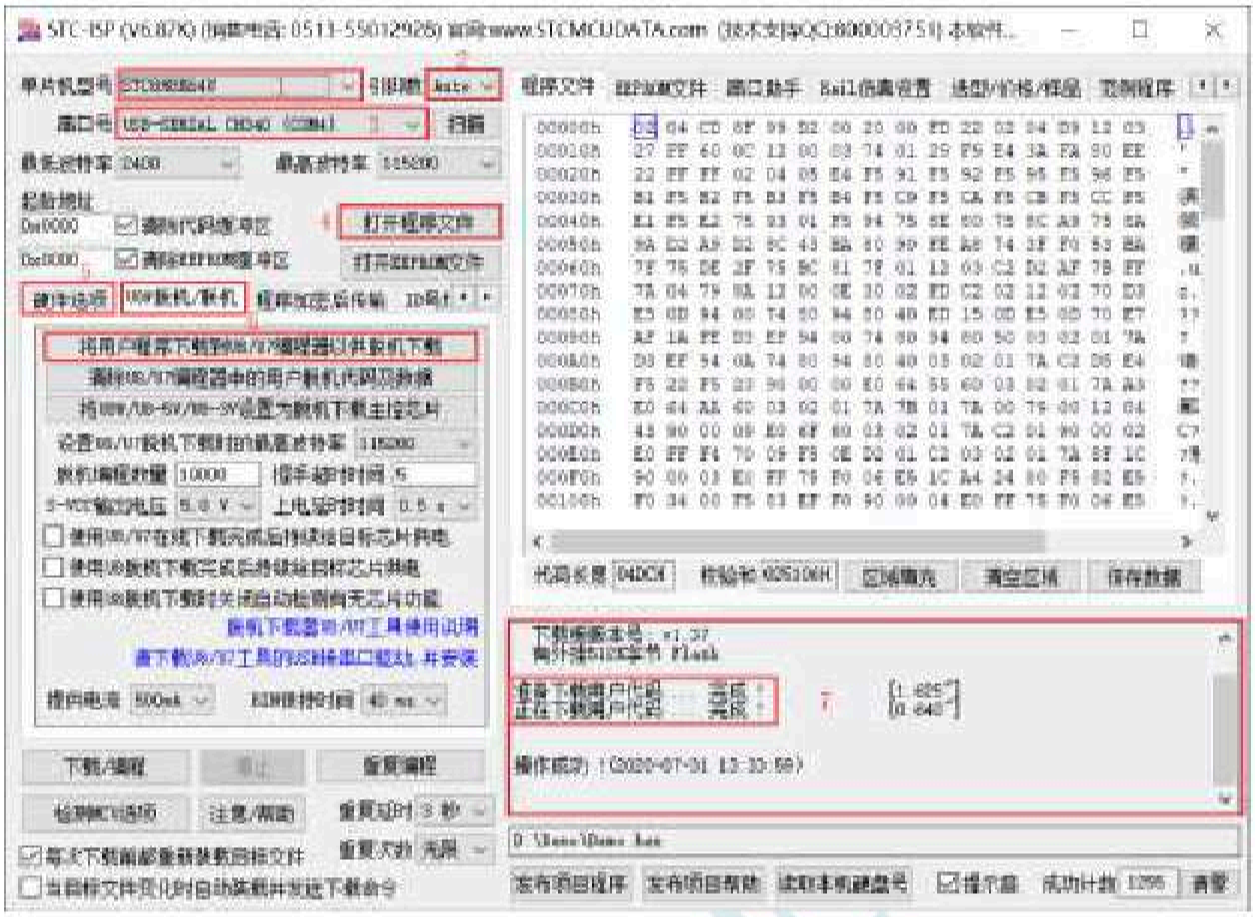
(1) First use the USB cable provided by STC to connect the U8W download board to the computer, as shown in the figure below. :



(2) In STC-ISP In the download software, follow the steps shown in the figure below to set it up :

It is recommended that users use STC-ISP Download the software (please feel free to pay attention Official website STCMCUDATA.com Download the the latest version of STC-ISP the software update, it is strongly recommended that users go to the official website STC http://www.STCMCUDATA.com latest version from Zhongzhong To download

Software use)



```
下载器版本号 : v1.37
擦除下载用户代码 ... 完成 !
正在下载用户代码 ... 完成 !

操作成功 !(2020-07-31 13:33:58)
```

1. Select the MICROCONTROLLER model; 2. Select the number of pins. When the chip is installed directly on the U8W to download, you must pay attention to selecting the correct number of pins, otherwise the download will fail; 3. Select the serial port number corresponding to U8W;

4. Open the target file (HEX format or BIN format);

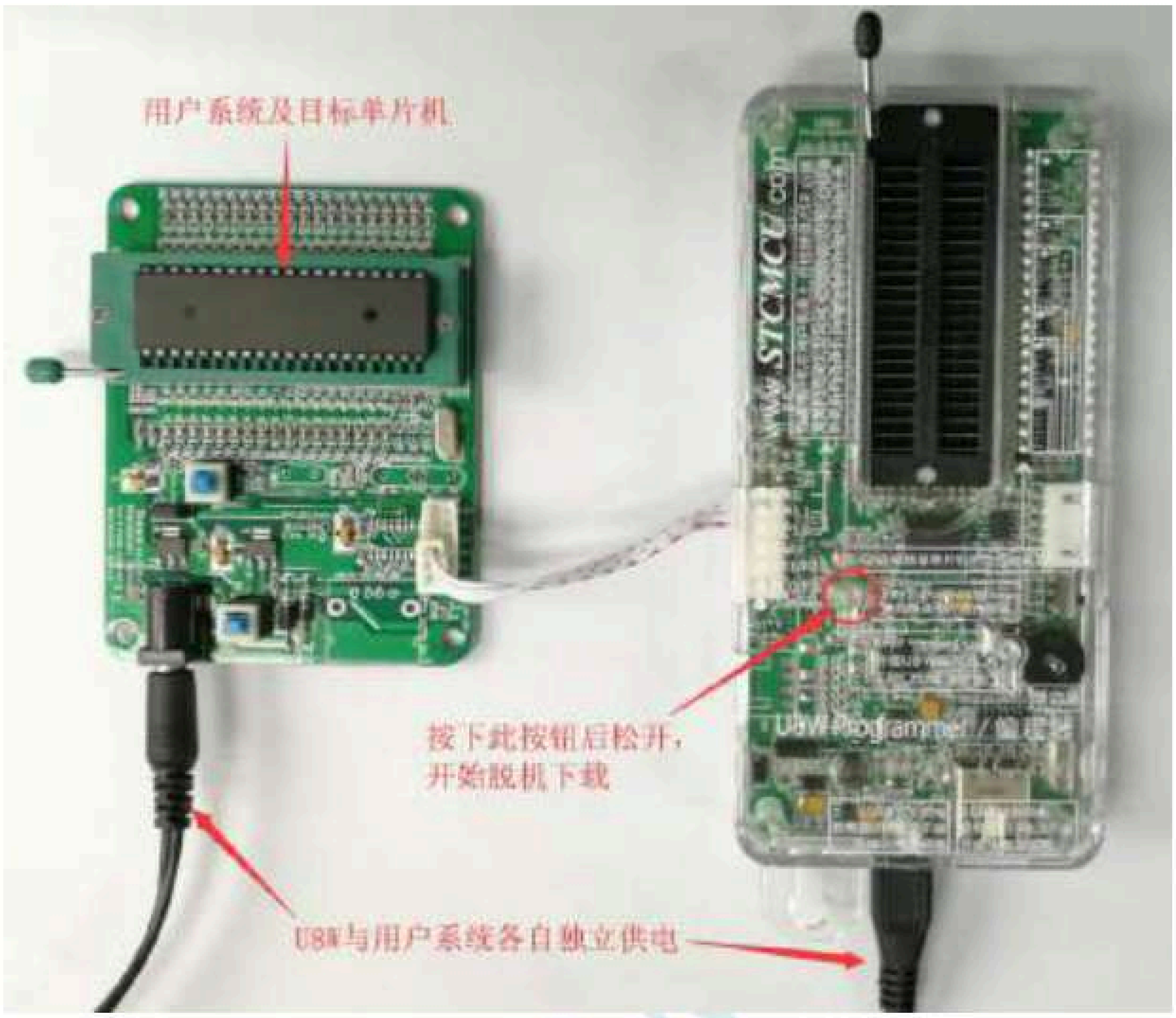
5. Set hardware options; 6. Select the "U8W Offline/Online" tab, set the offline programming option, and note that the output voltage of S-VCC matches the operating voltage of the target chip. ;

Click the "Download user program to U8/U7 programmer for offline download" button ;

7. The step information of the setup process is displayed, and the setup is complete with a prompt "The operation was successful!" ".

Follow the steps in the figure above. After the operation is completed, if the download is successful, it means that the user code and related setting option been downloaded to the U8W download tool . (3) Then connect

the U8W to the user system in the manner shown in the figure, and press the button shown in the figure first and then release it, ready to start the offline d and finally power on/off the user system, download the user program and officially start :



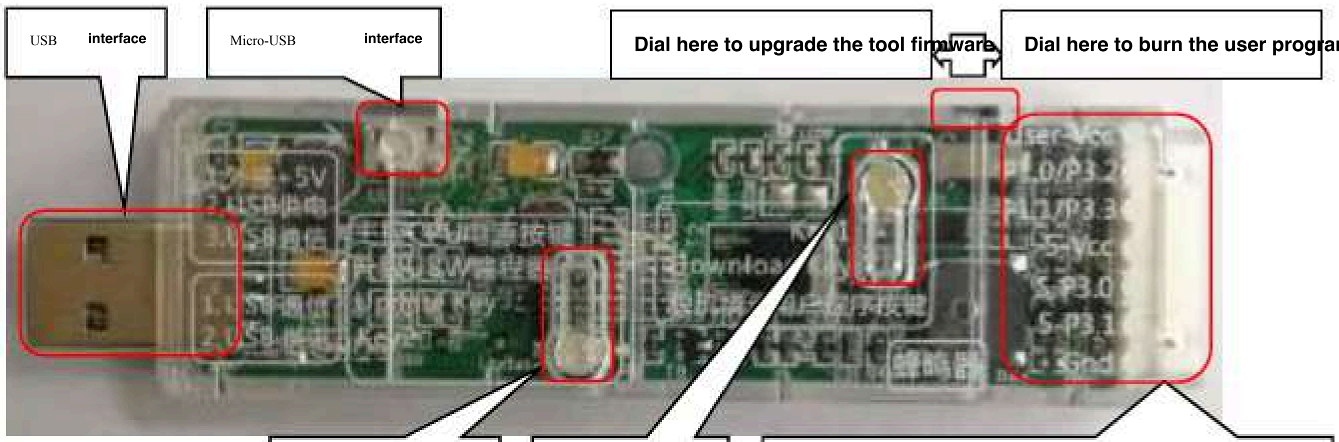
In the process of downloading, Download the one on the tool 4 LED Will be displayed in marquee mode. After the download is complete, if the download is U8W LED Will light up at the same time and turn off at the same time; if the download fails, then a LED All are not on. 4

a

Introduction to the function of F.

3.5 U8W-Mini The main interfaces and functions of the tool :

Described in detail below U8W-Mini


Toggle switch
USB interface
Micro-USB interface
Dial here to upgrade the tool firmware
Dial here to burn the user program

update U8W-Mini System program button

Offline burning user program button

Programming interface User-Vcc: Only power this tool from the user's system to ground

P1.0/P3.2: (used when setting the pin for burning protection) to

P1.1/P3.3: ground (used when setting the pin for burning protection)

S-Vcc: Only power the user's system from this tool

S-P3.0: Connected to the slave P3.0

S-P3.1: Connect the ground P3.1

Gnd: wire of the slave

Programming interface: According to different power supply methods, use different download cables to connect the user system.

U8W-Mini **Update system program button**: Used for U8W-Mini update Tools, when there is a new U8W version of When the firmware is installed, you need to pre

correct U8W-Mini The main control chip is updated (Note: You must update it first, Download the toggle switch on the selection interface and tog

Pieces)

Offline download user program button: Start offline PC Download the offline code to U8W-Mini On, then use the download

download button. First , the cable connects, the user system to U8W-Mini Press this button again to start the offline download (the download will start immediately every ti

User code).

Update/download selection interface: When needed U8W-Mini When upgrading the underlying firmware, you need to toggle this toggle switch to the firmw

Place, when you need to pass U8W-Mini To program the target chip, you need to toggle the toggle switch to the programming user program. (Toggle switch

Please refer to the picture above for connection method)

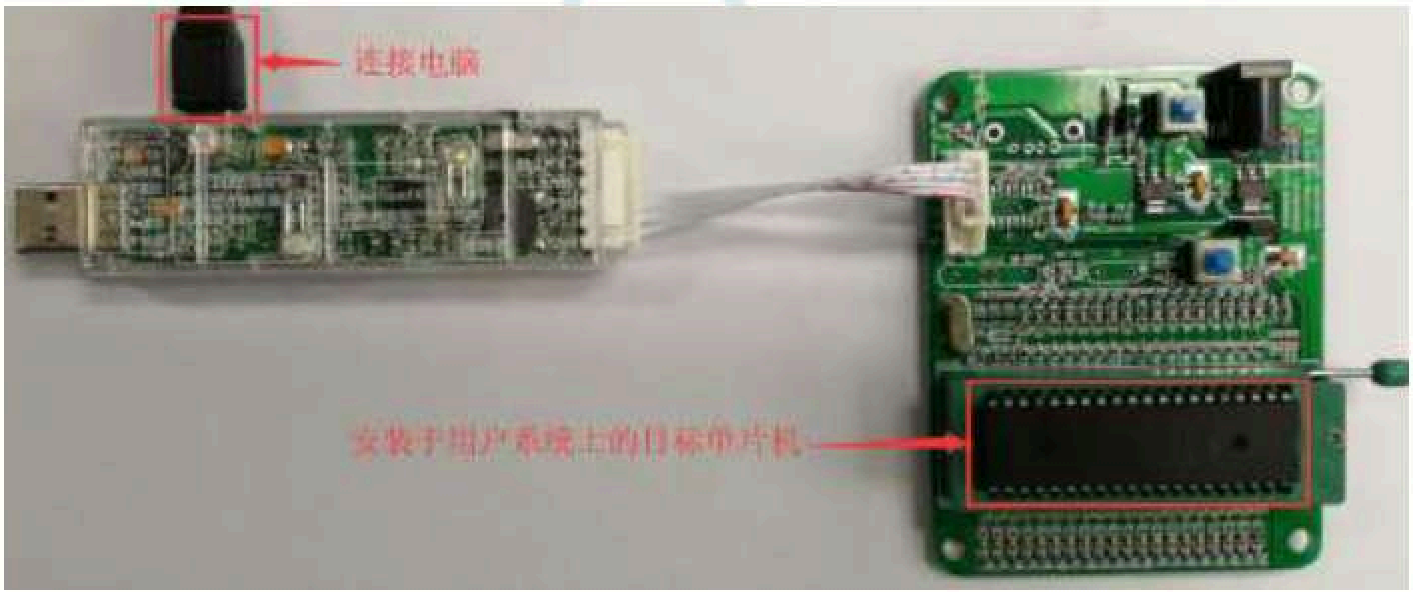
USB **interface** :

The USB interface has the same function as the Micro-USB interface, and the user can connect one of the interfaces to the computer as needed.

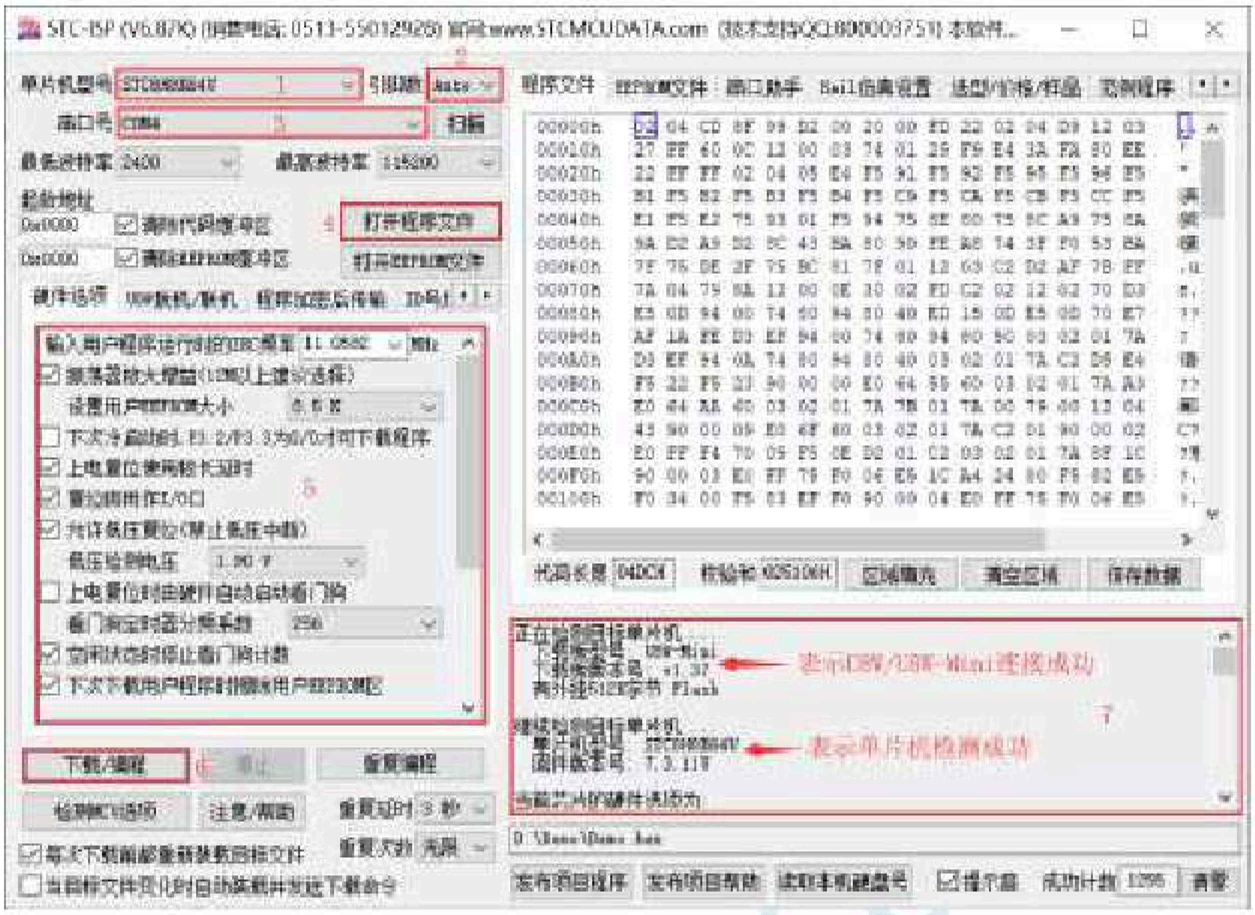
F. 3.6 U8W-Mini Online download instructions for use

The target chip is connected through the lead of the user system U8W-Mini And by U8W-Mini Connect to a computer for online download

First, use the USB cable provided by STC to connect the U8W-Mini to the computer, and then connect the U8W-Mini to the target monolithic machine of the user's system through the download cable . The connection method is shown in the figure below. :



Then use STC-ISP Download the software download program, the steps are as follows :



1. Select the MICROCONTROLLER model; 2. Select the number of pins. When the chip is installed directly on the U8W-Mini and downloaded, you must pay attention to selecting the correct number of pins, otherwise the download will be lost.

defeat ;

3. Select the serial port number corresponding to U8W-Mini;

4. Open the target file (HEX format or BIN format);

5. Set hardware options;

6. Click the "Download/program" button to start burning;

7. The step information of the burning process is displayed, and the burning is completed with a prompt "The operation was successful!"

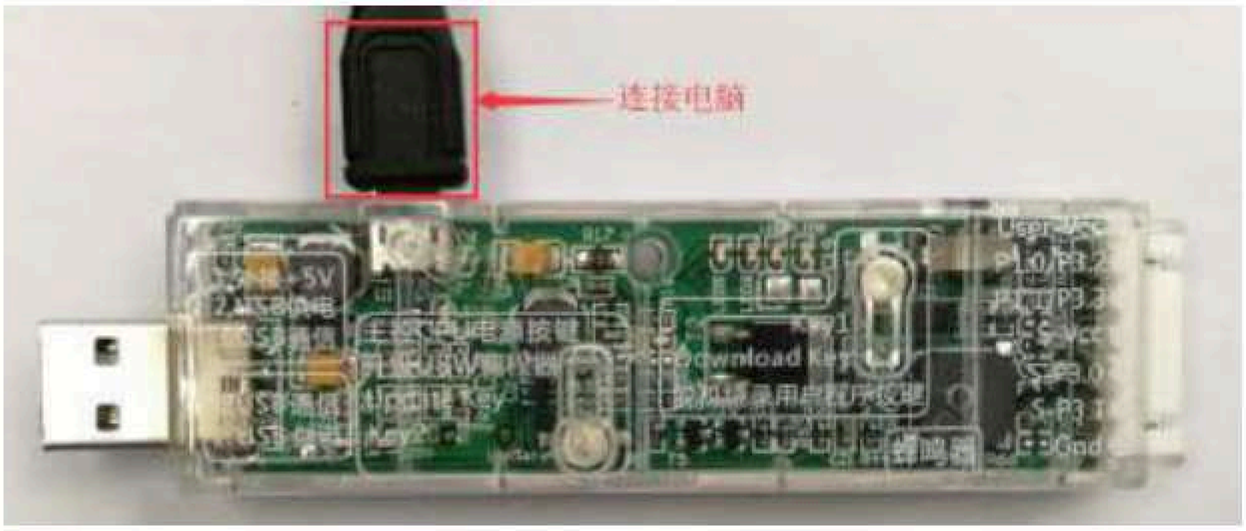
When the information box has the version number information of Flash When the corresponding information is obtained, U8W-Mini it means under that it has been c the output download board and the external mount tool. In the process of downloading , U8W-Mini 4 Download the one on the tool LED Will be displayed in marquee mode. After the download is complete, if the downl Will light up at the same time and turn off at the same time. If the download fails, then a 4 4 Then one LED All are not on.

It is recommended that users use the latest version of the download software (please feel free to pay attention to the official website http://www.STCMCUDATA.com STC-ISP Download the To download the software update, it is strongly recommended that users go to the official website http://www.STCMCUDATA.com latest version from Zhongzhong STC-ISP Software use).

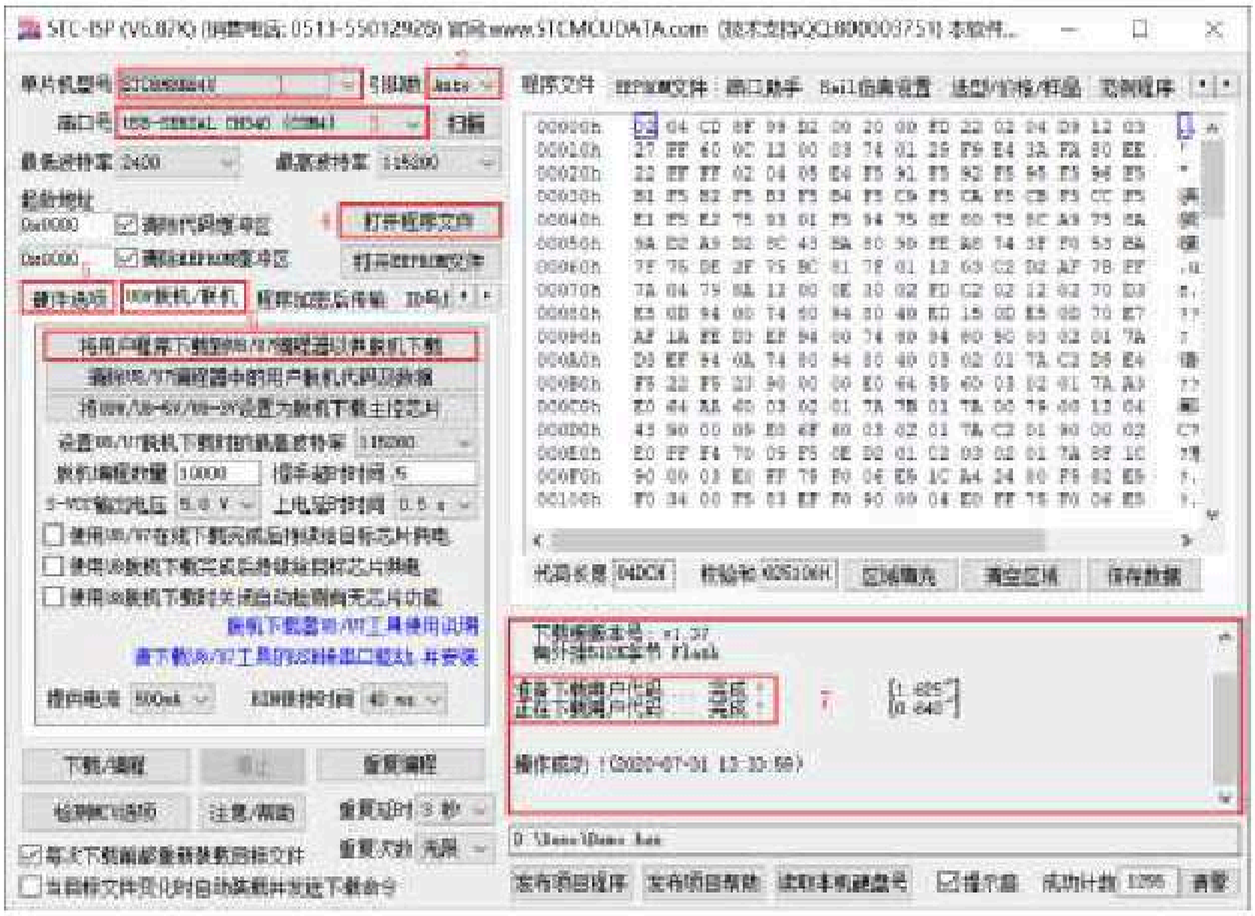
F. 3.7 U8W-Mini Offline download instructions for use

The target chip is connected by the lead U8W-Mini And pass USB Connect to the computer to U8W-Mini Power supply is offline of the user's system to download

The steps to use USB to power the U8W-Mini for offline download are as follows: (1) Use the USB cable provided by STC to connect the U8W-Mini download board to the computer, as shown in the figure below. :



2() In STC-ISP In the download software, follow the steps shown in the figure below to set it up :


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网 www.STCMCUDATA.com

单片机型号 STC8H8K64U 引脚数 Auto
串口号 USB-SERIAL CH340 (COM4) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区 打开程序文件
0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 U8W脱机/联机 程序加密后传输 ID号

将用户程序下载到U8/U7编程器以供脱机下载
清除U8/U7编程器中的用户脱机代码及数据
将U8W/U8-6W/U8-2W设置为脱机下载主控芯片
设置U8/U7脱机下载时的最高波特率 115200
脱机编程数量 10000 插拔延时时间 5
S-VCC输出电压 5.0 V 上电延时时间 0.5 s
☐ 使用U8/U7在线下载完成后持续给目标芯片供电
☐ 使用U8脱机下载完成后持续给目标芯片供电
☐ 使用U8脱机下载时关闭自动检测有无芯片功能
脱机下载器U8/U7工具使用说明
请下载U8/U7工具的USB转串口驱动，并安装
提供电流 500mA 启动延时时间 40 ms

下载/编程 停止 重复编程
检测MCU选项 注意/帮助 重复延时 3 秒
☑ 每次下载前都重新装载目标文件 重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

代码长度 04DCH 校验和 025106H 区域填充 清空区域 保存数据

下载编程器版本号 : v1.37
再外部扩展128K字节 Flash
正在下载用户代码 ... 完成 ! [1.605"]
正在下载用户代码 ... 完成 ! [0.640"]
操作成功 ! (2020-07-01 13:33:59)

D:\Demo\Demo.hex

发布项目程序 发布项目帮助 读取本机硬盘号 ☑ 提示音 成功计数 1295 清零

1. Select the MICROCONTROLLER model; 2. Select the number of pins. When the chip is installed directly on the U8W-Mini and downloaded, you must pay attention to selecting the correct number of pins, otherwise the download will be lost.

defeat ;

3. Select the serial port number corresponding to U8W-Mini; 4. Open the target file (HEX format or BIN format) ;

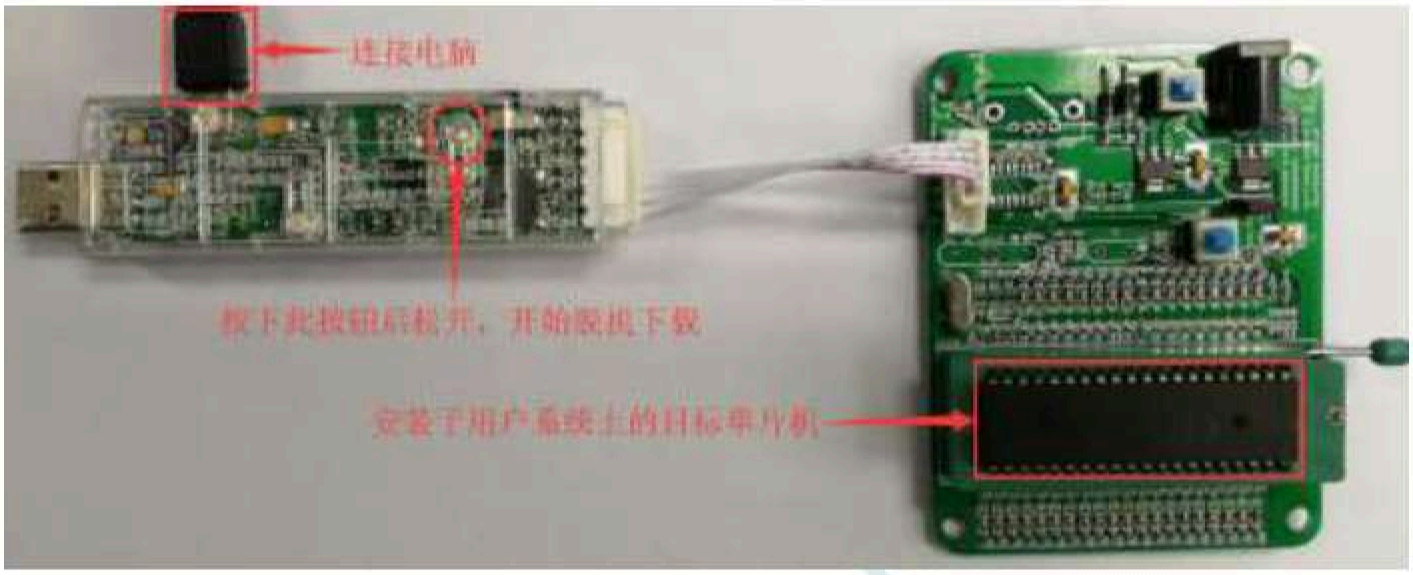
5. Set hardware options; 6. Select the "U8W Offline/Online" tab, set the offline programming option, and note that the output voltage of S-VCC matches the operating voltage of the target chip. ;

Click the "Download user program to U8/U7 programmer for offline download" button ;

7. The step information of the setup process is displayed, and the setup is complete with a prompt "The operation was successful!" ". Follow the steps in the figure above. After the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the download tool.

It is recommended that users STC-ISP Download the software (please feel free to pay attention Official website www.STCMCUDATA.com Download the use the latest version of the software To download STC-ISP update, it is strongly recommended that users go to the official website STC http://www.STCMCUDATA.com Download the latest version from Zhongzhong Software use)

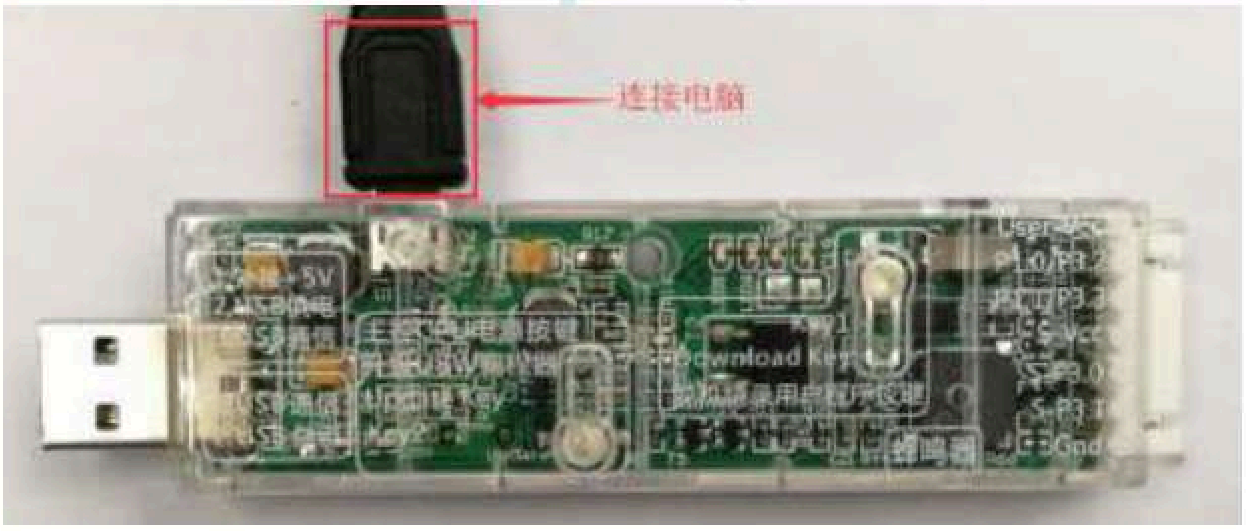
(3) Then use the cable to connect to the computer and connect the U8W-Mini Download the tool and the user system (target MCU) as shown in the figure below Connect upCome, and press the button shown in the figure and release it to start the offline download :



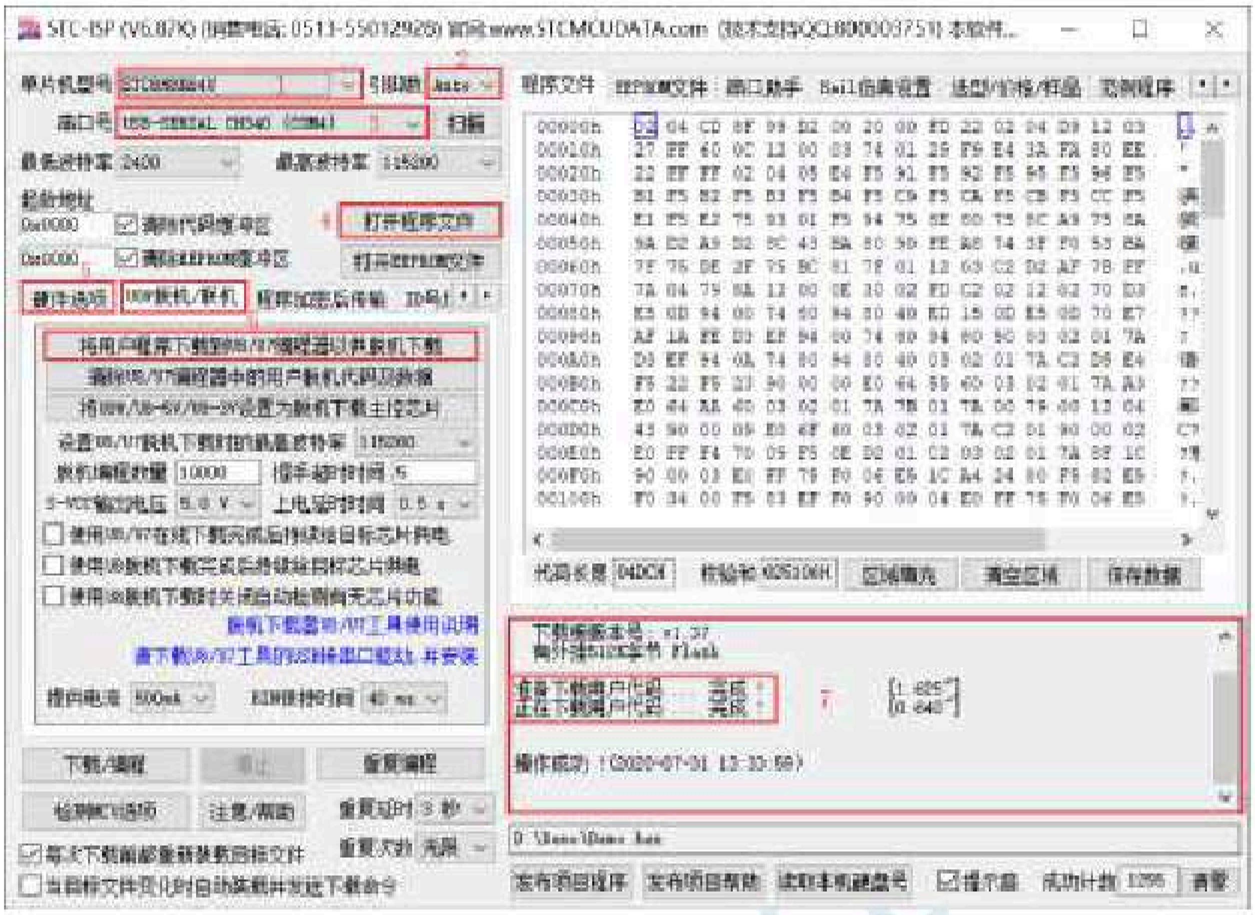
In the process of downloading , 4 Download the one on the tool LED Will be displayed in marquee mode. After the download is complete, if the download U8W-Mini Will light up at the same time and turn off at the same time; if the download fails, then a 4 All are 4 Then one LED

The target chip is connected by the lead of the user system U8W-Mini And through the user system to U8W-Mini Power supply for offline download

(1) First use the USB cable provided by STC to connect the U8W-Mini The download board is connected to the computer, as shown in the figure below



2() In STC-ISP In the download software, follow the steps shown in the figure below to set it up :


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件..

单片机型号 STC8H8K64U 引脚数 Auto
串口号 USB-SERIAL CH340 (COM4) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区 打开程序文件
0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 U8W脱机/联机 程序加密后传输 ID号

将用户程序下载到U8/U7编程器以供脱机下载
清除U8/U7编程器中的用户脱机代码及数据
将U8W/U8-S/U8-3P设置为脱机下载主控芯片
设置U8/U7脱机下载时的最高波特率 115200
脱机编程次数 10000 握手超时时间 5
S-VCC输出电压 5.0 V 上电延时时间 0.5 s
☐ 使用U8/U7在线下载完成后持续给目标芯片供电
☐ 使用U8脱机下载完成后持续给目标芯片供电
☐ 使用U8脱机下载时关闭自动检测有无芯片功能
提供电流 500mA 超时时间 40 ms

下载/编程 停止 重复编程
检测MCU选项 注意/帮助 重复延时 3 秒
重复次数 无限
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

程序文件 EEPROM文件 串口助手 Keil仿真设置 选型/价格/样品 范例程序

```
00000h  02 04 CD 8F 99 D2 00 20 00 FD 22 02 04 D9 12 03
00010h  27 FF 60 0C 12 00 03 74 01 25 F9 E4 3A FA 80 EE
00020h  22 FF FF 02 04 05 E4 F5 91 F5 92 F5 95 F5 96 F5
00030h  B1 F5 B2 F5 B3 F5 B4 F5 C9 F5 CA F5 CB F5 CC F5
00040h  E1 F5 E2 75 93 01 F5 94 75 8E 00 75 8C A9 75 8A
00050h  9A D2 A9 D2 8C 43 8A 80 90 FE 80 74 3F F0 53 8A
00060h  7F 75 DE 2F 75 BC 81 7F 01 12 03 C2 D2 AF 7B FF
00070h  7A 04 79 8A 12 00 0E 30 02 FD C2 02 12 02 70 D3
00080h  E5 0D 94 00 74 80 94 80 40 ED 15 0D E5 0D 70 E7
00090h  AF 1A EE D2 EF 94 00 74 80 94 80 90 03 02 01 7A
000A0h  D0 EF 94 0A 74 80 94 80 40 03 02 01 7A C2 D8 E4
000B0h  F5 22 F5 23 90 00 00 E0 64 55 60 03 02 01 7A A3
000C0h  E0 64 AA 60 03 02 01 7A 7B 01 7A 00 79 00 12 04
000D0h  43 90 00 06 E0 6F 60 03 02 01 7A C2 D1 90 00 02
000E0h  E0 FF F4 70 09 F5 0E D2 01 C2 03 02 01 7A 8F 1C
000F0h  90 00 03 E0 FF 78 F0 06 E5 1C A4 24 80 F8 82 E5
00100h  F0 34 00 F5 83 EF F0 90 00 04 E0 FF 75 F0 04 E5
```

代码长度 04DCH 校验和 00510EH 区域填充 清空区域 保存数据

下载器版本号 : v1.27
脱机代码保存在 Flash
正在下载用户代码 ... 完成 ! [1.625"]
正在下载用户代码 ... 完成 ! [0.640"]
操作成功 ! (2020-07-31 13:33:58)

D:\Demo\Demo.hex

发布项目程序 发布项目帮助 读取本机硬盘号 ☑ 提示音 成功计数 1295 清零

1. Select the MICROCONTROLLER model; 2. Select the number of pins. When the chip is installed directly on the U8W-Mini and downloaded, you must pay attention to selecting the correct number of pins, otherwise the download will be lost.

defeat ;

3. Select the serial port number corresponding to U8W-Mini;

4. Open the target file (HEX format or BIN format);

5. Set hardware options; 6. Select the "U8W Offline/Online" tab, set the offline programming option, and note that the output voltage of S-VCC matches the operating voltage of the target chip. ;

Click the "Download user program to U8/U7 programmer for offline download" button ;

7. The step information of the setup process is displayed, and the setup is complete with a prompt "The operation was successful!" ".

Follow the steps in the figure above. After the operation is completed, if the download is successful, it means that the user code and related s under

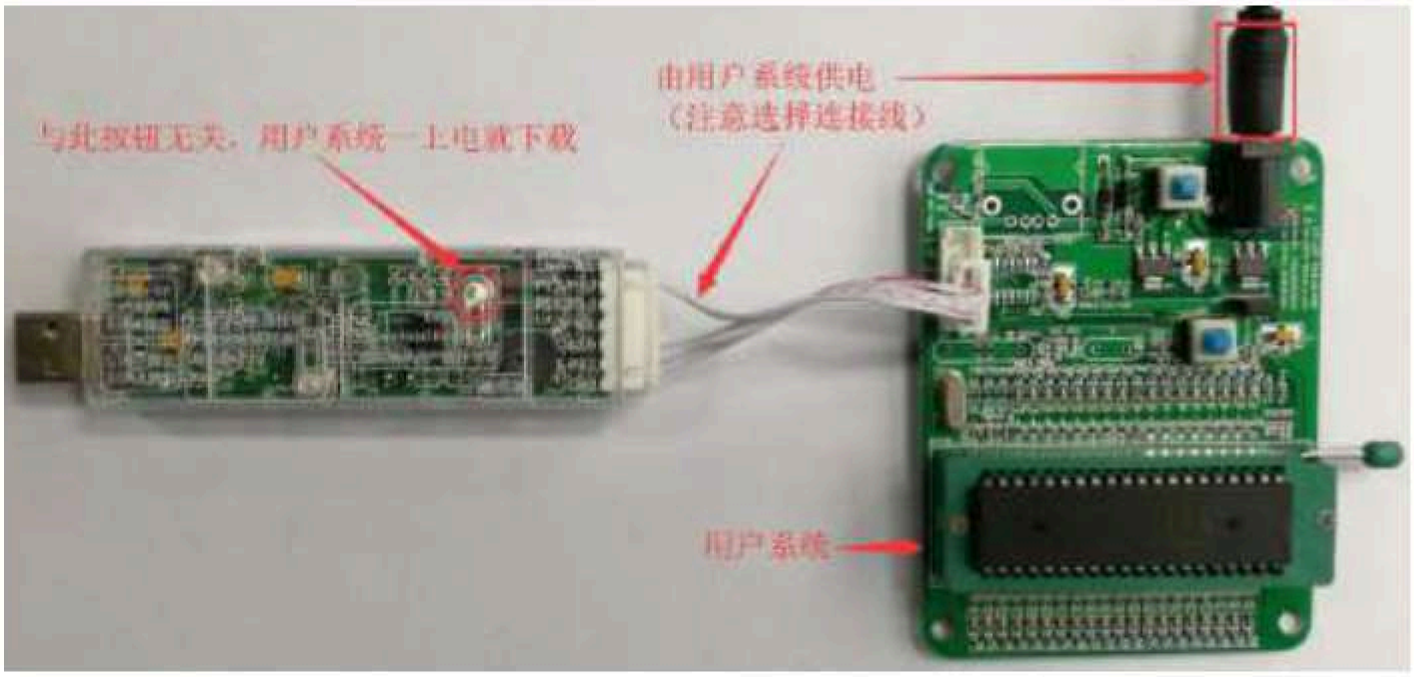
Loading tools.

It is recommended that users use the latest version of the download software (please feel free to pay attention to the official website the ISP STC Download the

To download the software update, it is strongly recommended that users go to the official website https://www.STCMCUDATA.com STC-ISP latest version from Zhongzhong

Software use).

(3) Then connect U8 in the manner shown in the figure below U8W-Mini With the user system, the user system starts offline download as soon as it is powered o



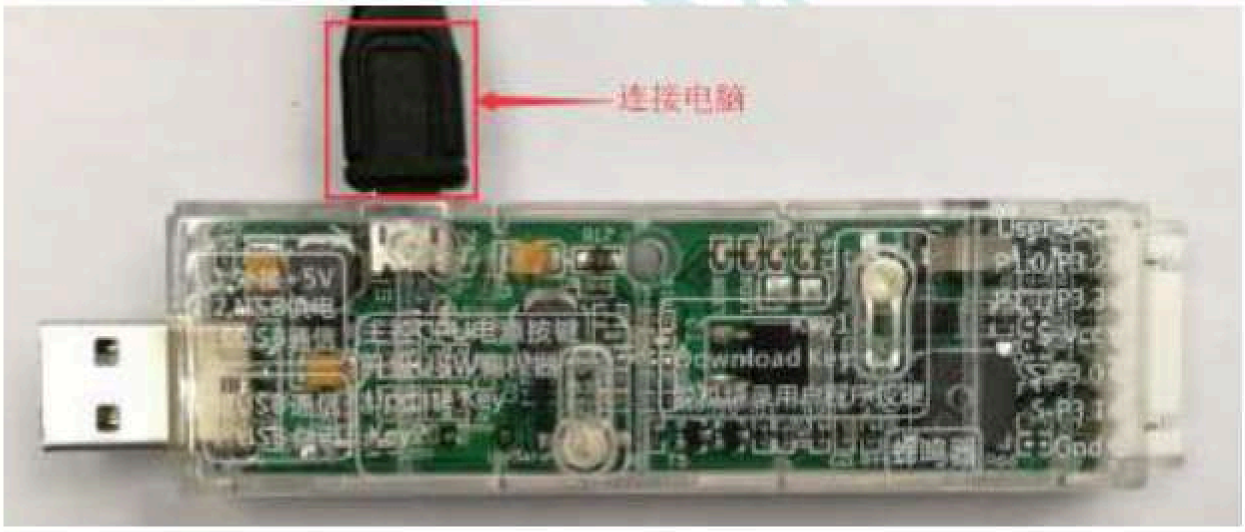
In the process of downloading，4 Download the one on the tool LED Will be displayed in marquee mode. After the download is complete, if the downl

U8W-Mini Will light up at the same time and turn off at the same time; if the download fails, then a 4 All LED are on.

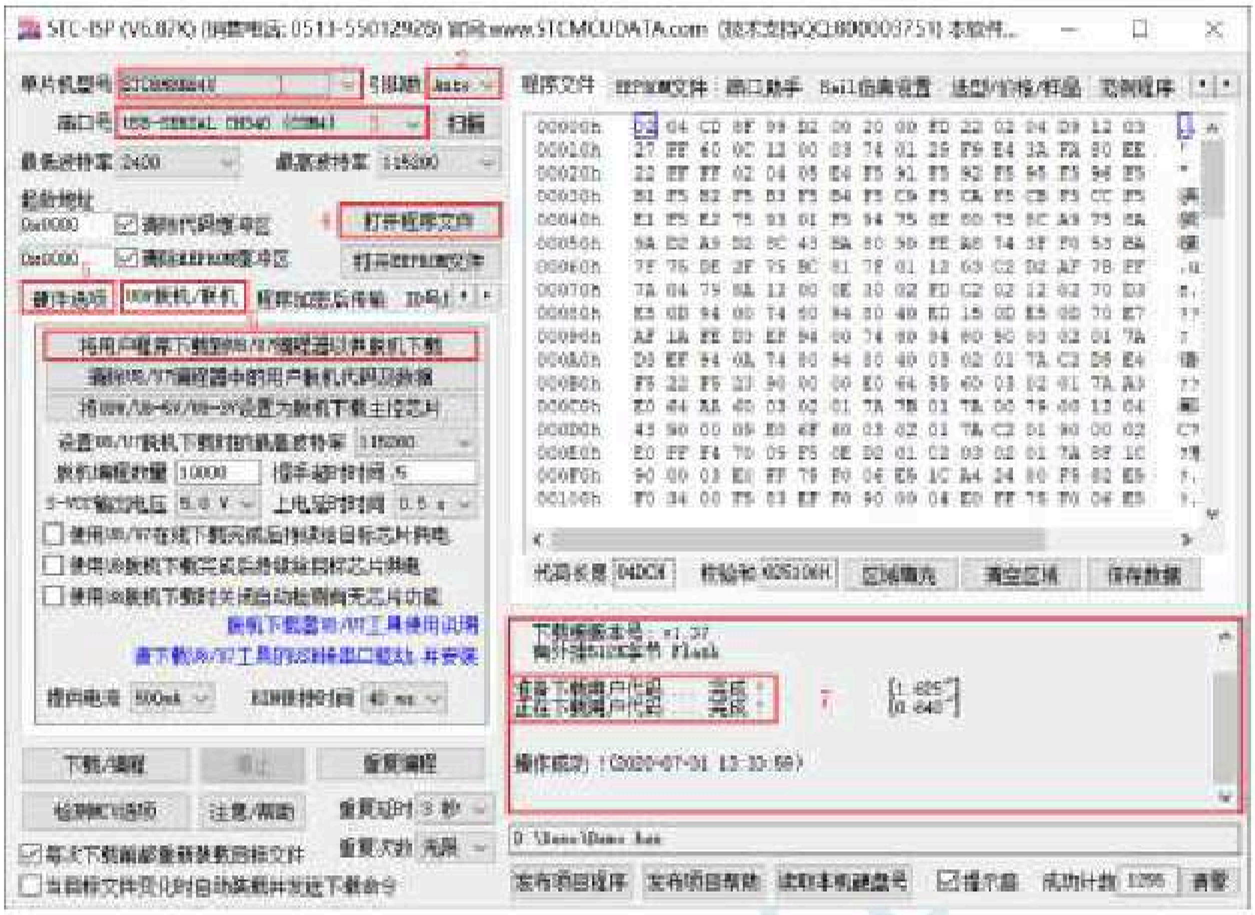
4 Then one LED

The target chip is connected by the lead U8W-Mini and U8W-Mini Independent power supply with the user system for offline operation of the user's system to download

(1) First use the USB cable provided by STC to connect the U8W-Mini The download board is connected to the computer, as shown in the figure below



(2) In STC-ISP In the download software, follow the steps shown in the figure below to set it up：


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件...

单片机型号 STC8H8K64U　引脚数 Auto
串口号 USB-SERIAL CH340 (COM4)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件
硬件选项　U8W脱机/联机　程序加密后传输　ID号1

将用户程序下载到U8/U7编程器以供脱机下载
清除U8/U7编程器中的用户脱机代码及数据
将U8W/U8-6V/U8-3V设置为脱机下载主控芯片
设置U8/U7脱机下载时的最高波特率 115200
脱机编程次数 10000　按手延时时间 5
S-VCC输出电压 5.0 V　上电延时时间 0.5 s
☐ 使用U8/U7在线下载完成后持续给目标芯片供电
☐ 使用U8脱机下载完成后持续给目标芯片供电
☐ 使用U8脱机下载时关闭自动检测有无芯片功能
供电电流 500mA　RESET持续时间 40 ms

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 3 秒
☑ 每次下载前都重新装载目标文件　重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

程序文件　EEPROM文件　串口助手　Keil仿真设置　选型/价格/样品　范例程序
代码长度 04DCH　校验和 0251DAH　区域填充　清空区域　保存数据

下载器版本号 : v1.27
用户程序 ... Flash
正在下载用户代码 ... 完成 [1.625"]
正在下载用户代码 ... 完成 [0.640"]
操作成功 ! (2020-07-31 13:33:59)

发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 1295　清零

1. Select the MICROCONTROLLER model; 2. Select the number of pins. When the chip is installed directly on the U8W-Mini and downloaded, you must pay attention to selecting the correct number of pins, otherwise the download will be lost.

defeat ;

3. Select the serial port number corresponding to U8W-Mini;

4. Open the target file (HEX format or BIN format);

5. Set hardware options; 6. Select the "U8W Offline/Online" tab, set the offline programming option, and note that the output voltage of S-VCC matches the operating voltage of the target chip. ;

Click the "Download user program to U8/U7 programmer for offline download" button ;

7. The step information of the setup process is displayed, and the setup is complete with a prompt "The operation was successful!" ".

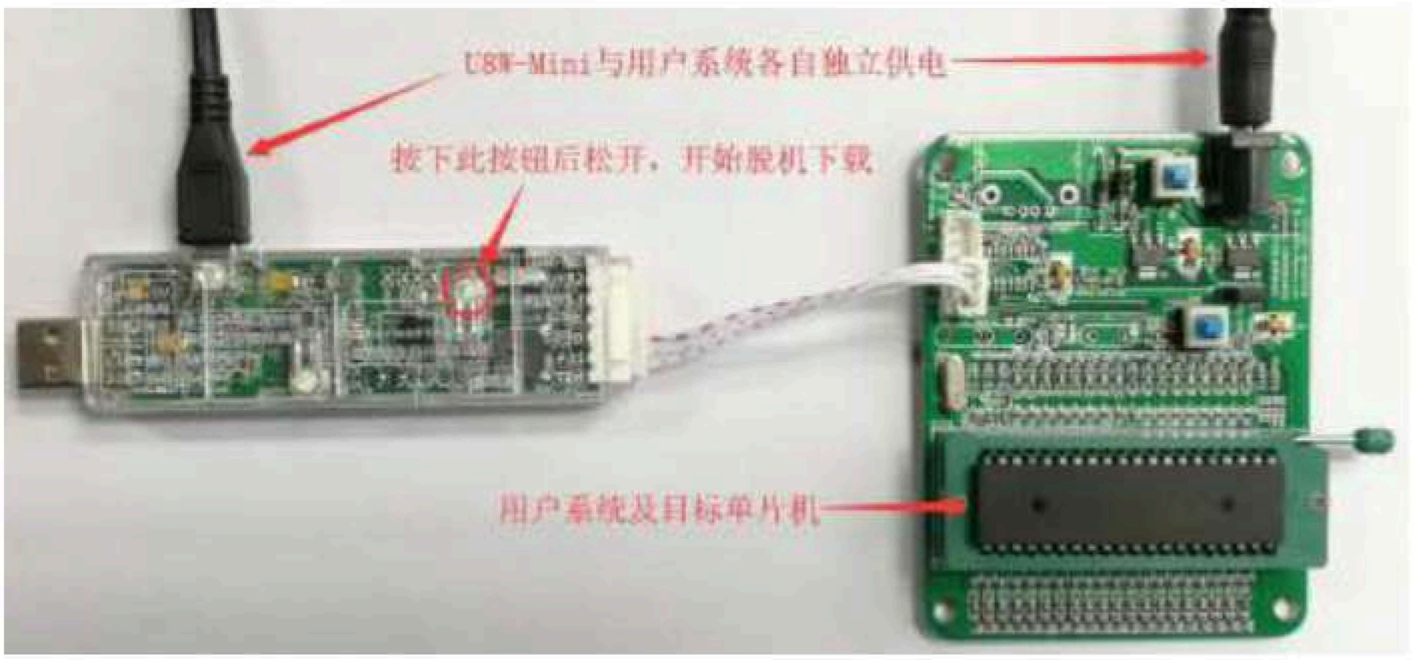
Follow the steps in the figure above. After the operation is completed, if the download is successful, it means that the user code and related s under Loading tools.

It is recommended that users use the latest version of the download software (please feel free to pay attention to the official website the Download ISP STC l

To download the software update, it is strongly recommended that users go to the official website http://www.STCMCUDATA.com STC-ISP latest version from Zhongzhong Software use).

(3) Then connect U8 in the manner shown in the figure below W-Mini System with the user, and press the button shown in the figure first and then release it, r

The machine is downloaded, and finally the user's system is powered on/off, and the download of the user program officially begins. :



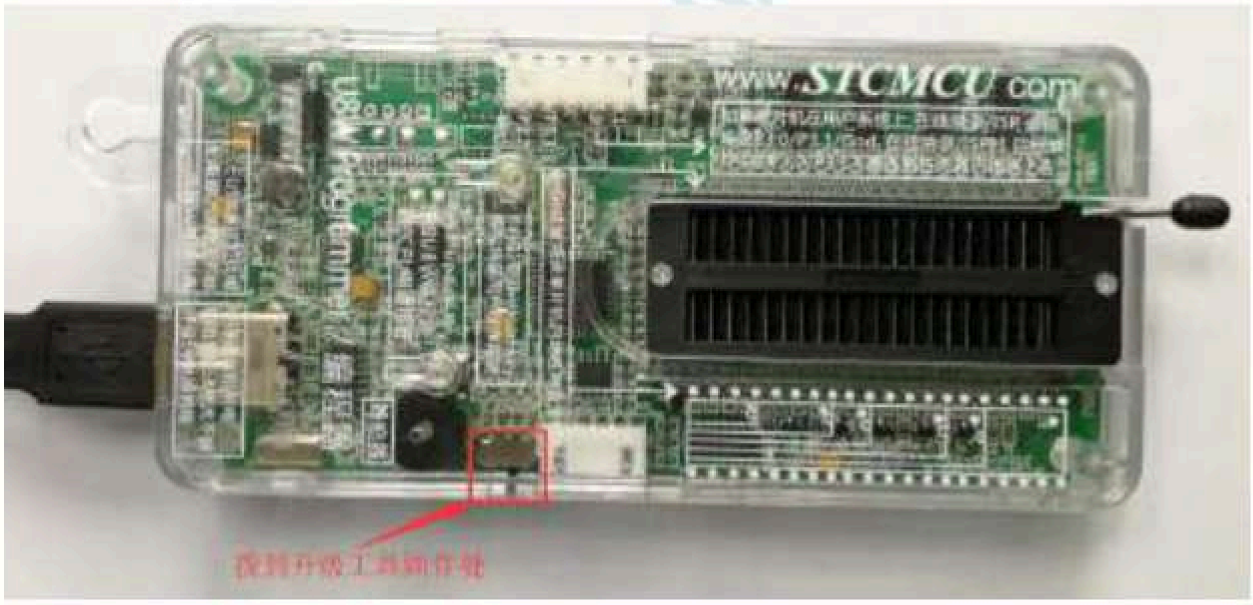
In the process of downloading , 4 Download the one on the tool LED Will be displayed in marquee mode. After the download is complete, if the downl

U8W-Mini Will light up at the same time and turn off at the same time; if the download fails, then a 4 All are not

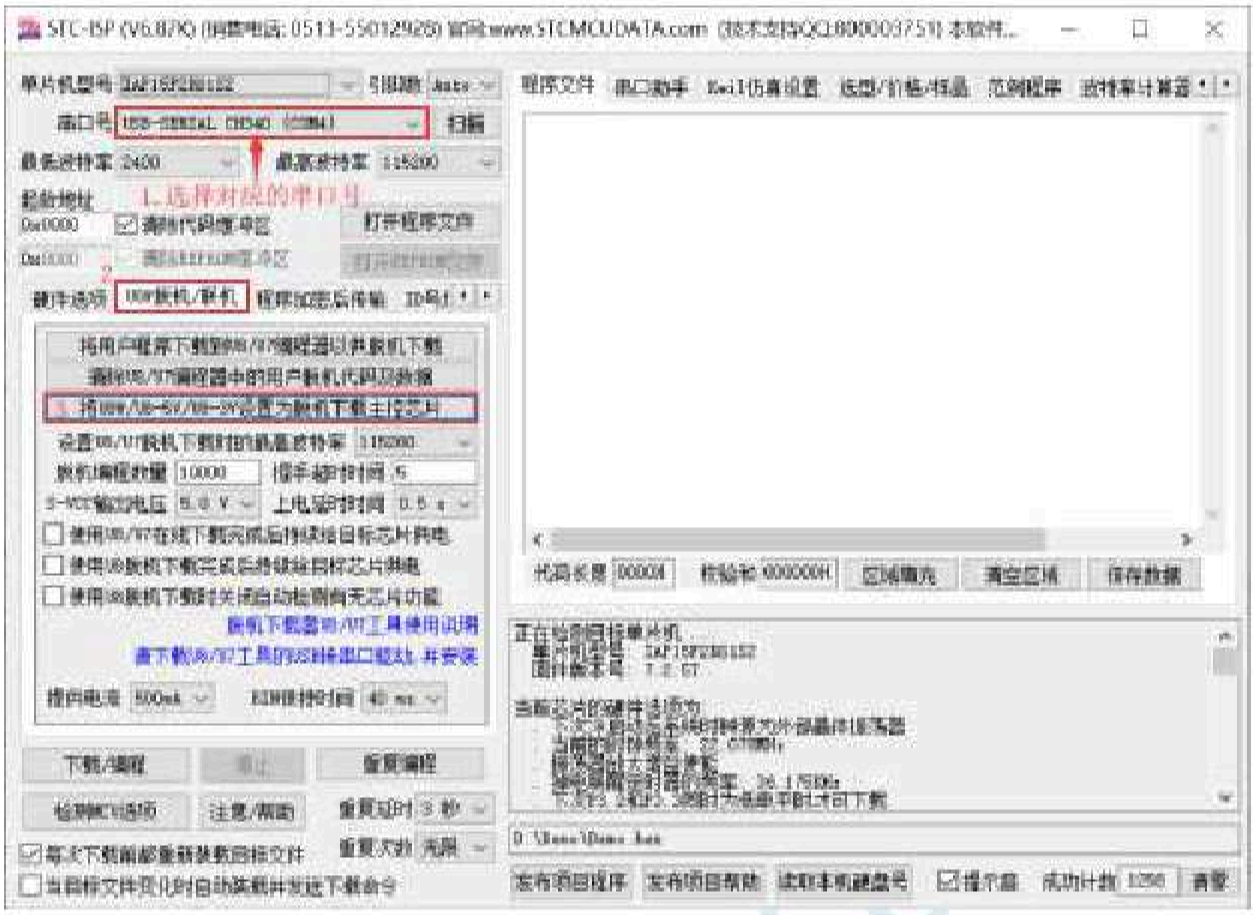
4 Then one LED

F. 3.8 / Production update U8W/U8W-Mini

The process of making a U8W/U8W-Mini download master is similar. In order to save space, the following uses U8W as an example to detail how to make a download master. Before making the U8W download master, you need to dial the "Update/Download selection interface" of the U8W download board to "U tool Firmware", as shown in the figure below. :



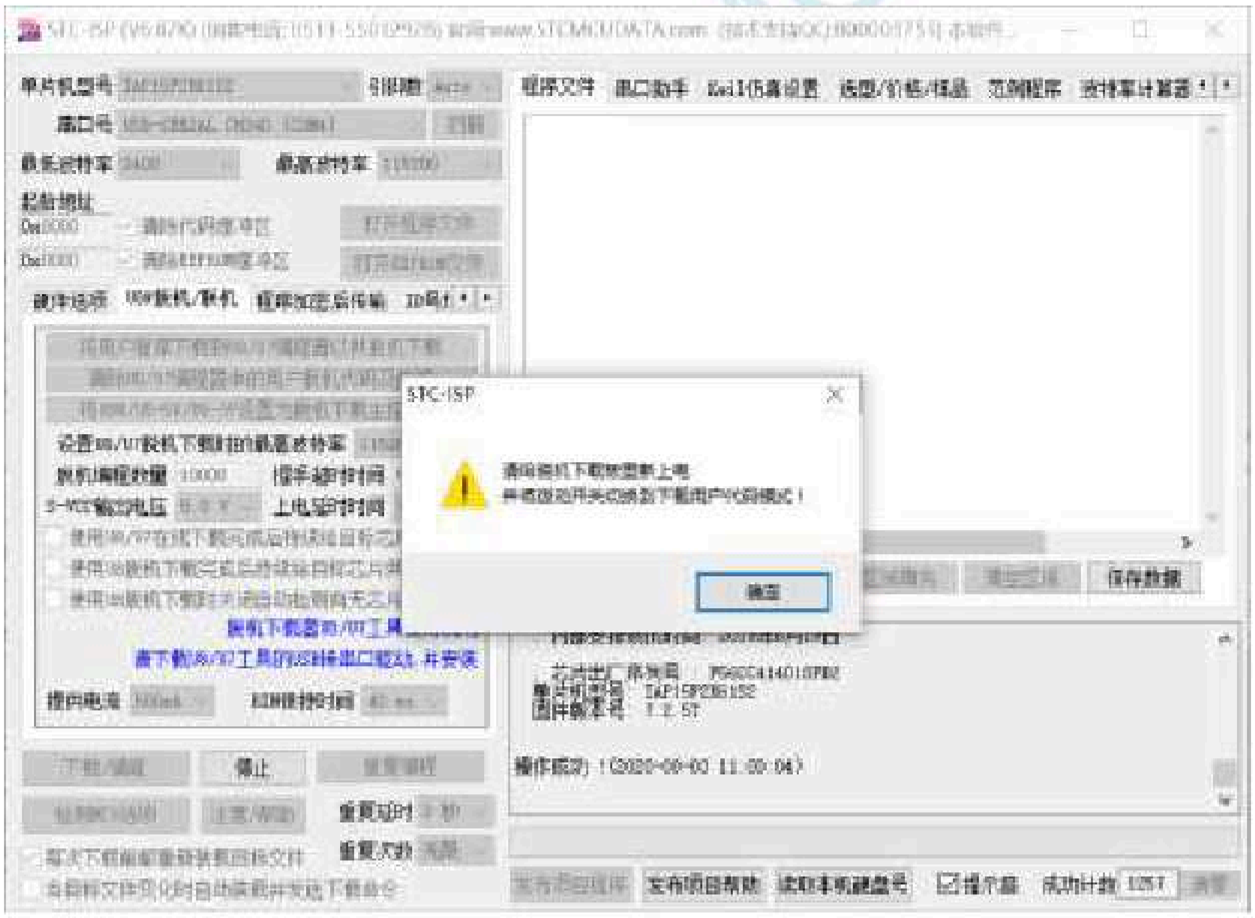
Then in STC-ISP Click "Set U8W/U8-5V/U8-3V as offline download master" on the "U8W Offline/Online" page in the download program Chip" button, as shown below: (Note: Be sure to choose the serial port corresponding to U8W)



STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网 www.STCMCUDATA.com (技术支持QQ:800003751) 本软件...

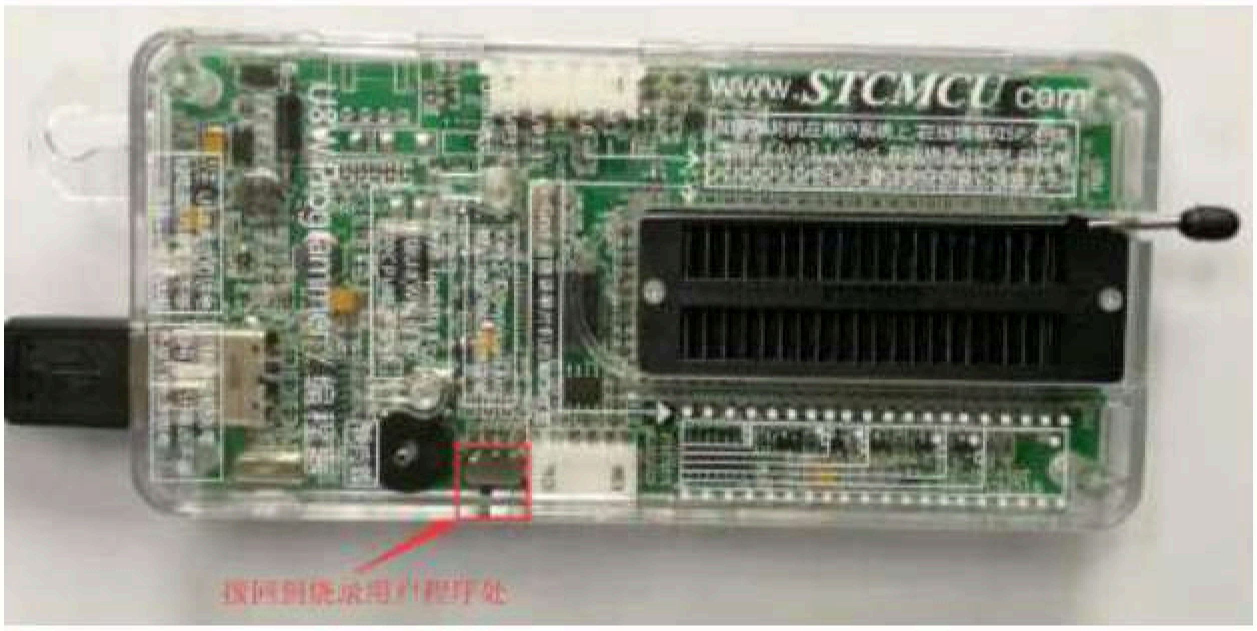
串口号 USB-SERIAL CH340 (COM4)
最低波特率 2400　最高波特率 115200
1. 选择对应的串口号
起始地址 0x0000 ☑ 清除代码缓冲区
硬件选项 U8W脱机/联机
将U8W/U8-Mini/U8-SW设置为脱机下载主控芯片
设置U8/U7脱机下载时的最高波特率 115200
脱机编程数量 10000　握手超时时间 5
上电延时时间 0.5 s
提供电流 500mA
重复延时 3 秒　重复次数 无限
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

The following screen appears to indicate that the U8W control chip is completed：



STC-ISP

After the production is complete, don't forget to dial the "Update/Download selection interface" of U8W back to the "Burn user program" mode, and power up the U8W download tool again, as shown in the figure below. : (Otherwise, it will not be able to burn normally)



F. 3.9 U8W/U8W-Mini Set the pass-through mode (can be used for simulation)

To use For simulation, you must first U8W/U8-Mini Set to pass-through mode. U8W/U8W-Mini real

the current The method of switching to serial port pass-through mode is as follows :

1. First of all, the U8W/U8W-Mini firmware must be upgraded to v1.37 and above; 2. After U8W/U8W-Mini is powered on, it will be in normal download mode. At this time, press and hold the Key1 (download) button on the tool without releasing it, and then press again.

Key2 (power supply) button, then release the Key2 (power supply) button, and then release the Key1 (download) button, U8W/U8W-Mini will enter the USB-to-serial port pass-through mode. (Press Key1 to release Key1, Key2

3. that enters the pass-through mode is just a simple USB to serial port and does not have offline download function. If you need to restore the original function of U8W/U8W-Mini, you only need to press the Key2 (power) button separately again.

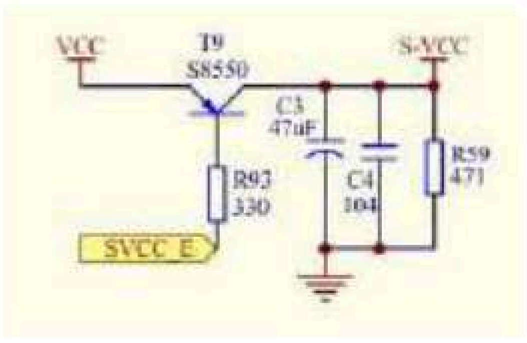
F. 3.10 U8W/U8W-Mini The reference circuit

USB type online/offline download board U8W-Mini Provides users with the following common control interfaces :

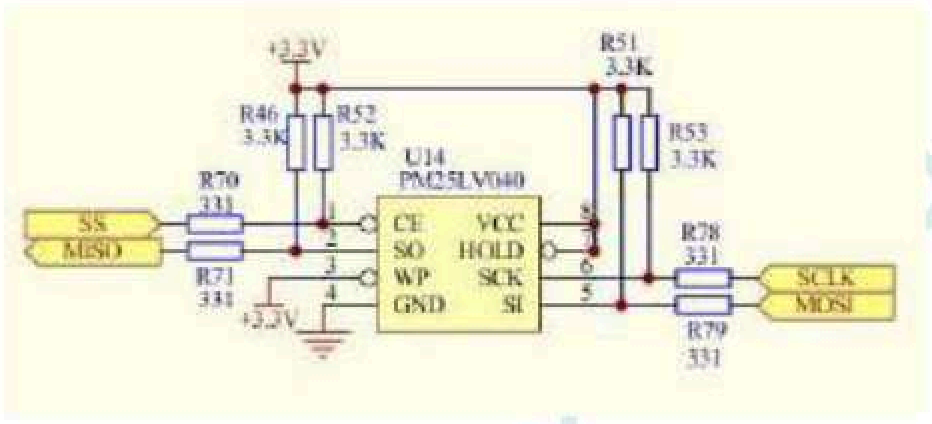
Pin function Power	port	Function description
control pin download	P2.6	Low-bit valid
communication pin	P1.0	serial port RXD, Connected to the target TXD (P3.1)
	P1.1	chip TXD, The serial port RXD (P3.0)
programming button	P3.6	connected to the target
display	P3.2	chip is low effective LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
Plug-in serial Flash Control foot	P2.4	Flash of CE foot
	P2.2	Flash The foot SO
	P2.3	Flash of SI foot
	P2.1	Flash of SCLK foot

Fully automatic burning tool sorter signal	P3.6	Start signal
	P1.5	complete signal
	P5.4	OK Signal (good product signal)
	P3.7	ERROR The signal (defective product signal)
Buzzer (BEEP)	P2.5	is highly effective (high-level sound is emitted)

Refer to the circuit diagram for the control part of the control power supply :

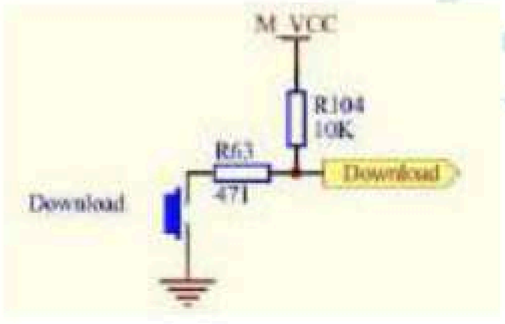


Refer to the circuit diagram for the Flash control part :

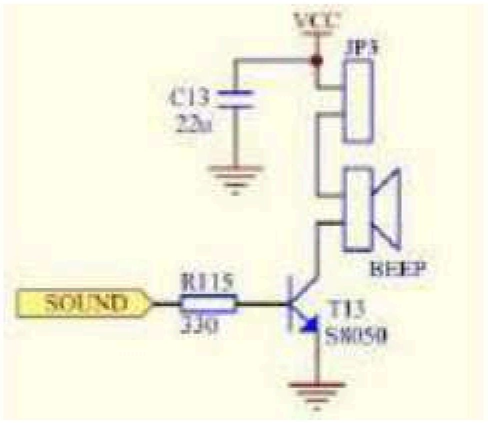


The user program is greater than 4K When you need this Flash memory

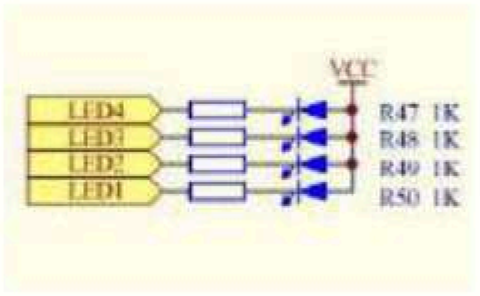
Refer to the circuit diagram for the key part :



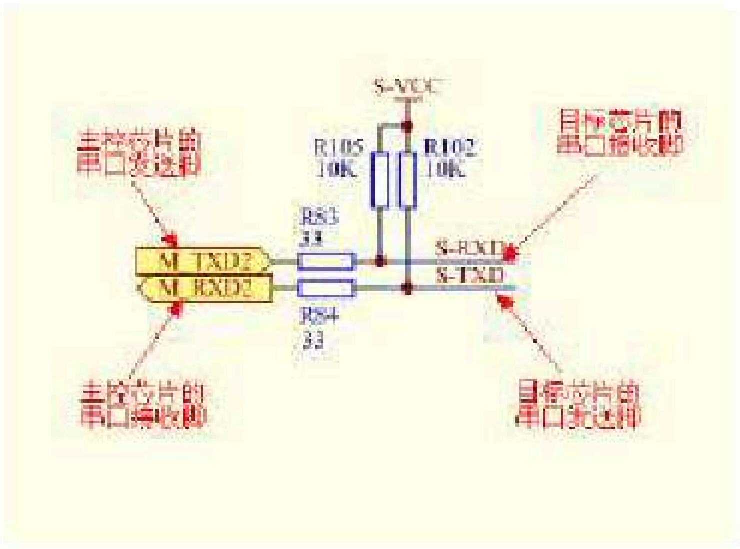
Reference circuit diagram for buzzer part :



Reference circuit diagram for LED display part :



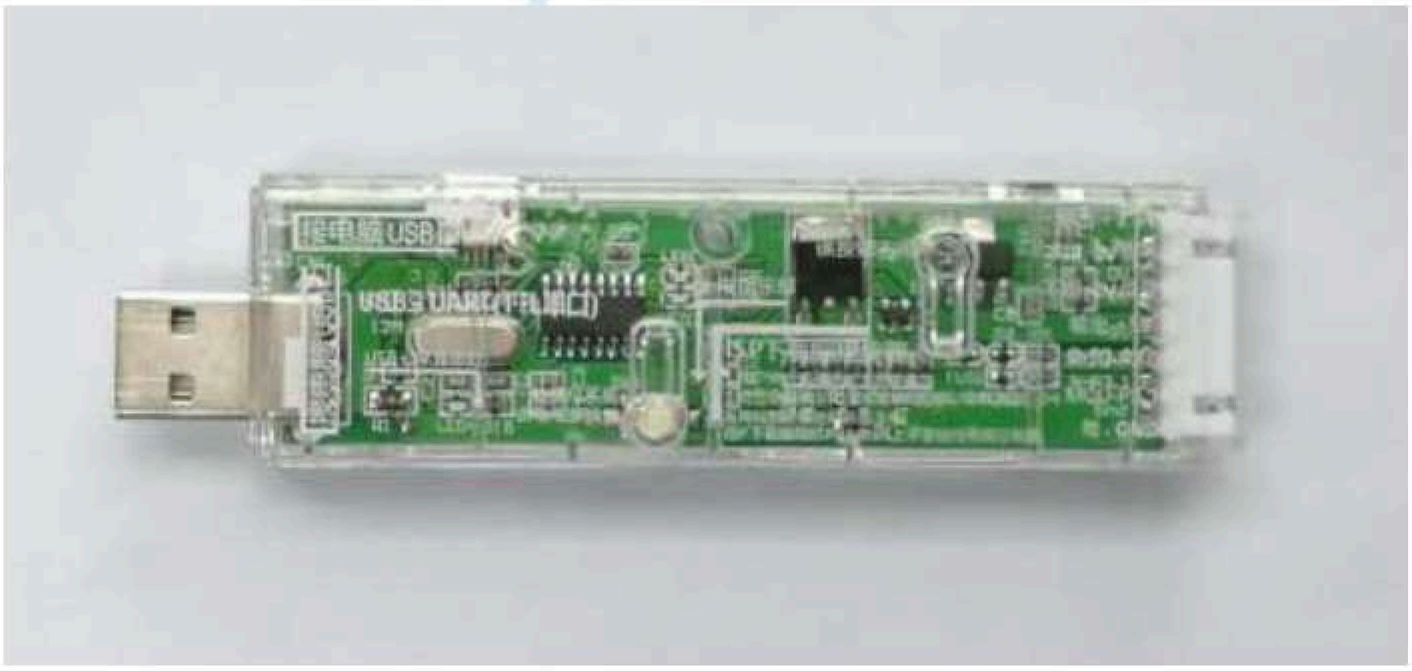
Refer to the circuit diagram for the connection part of the serial communication pin：



F. 4 STC Universal USB To serial port tool

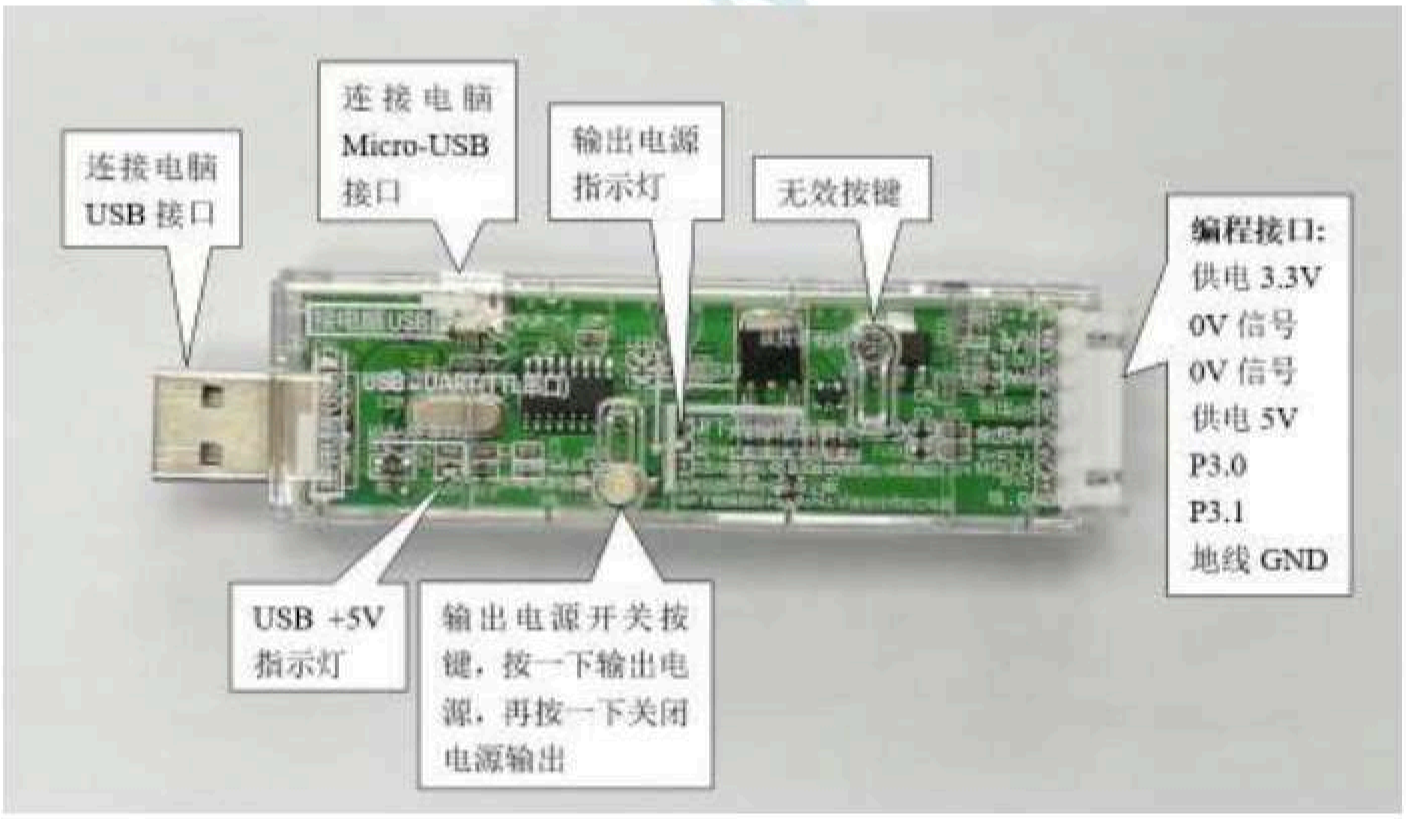
F. 4.1 STC Universal USB External view of the serial port tool

Front：



back：

F. 4.2 STC Universal USB Layout diagram of the serial port tool



Here, the "power switch" needs to be described:

The function of this button is the same as that of a self-locking switch. When the switch button is on time for the first time, the switch is turne

When the switch button is on time for the second time, the switch disconnects the power supply. In view of the fact that the self-locking switc

of circuits that use tact switches to replace the self-locking switch function to improve the service life of the tool.

For downloading, the serial port command must be received during power-on reset before execution will begin. The MCU, if you want to carry out STC ISP ISP

Program, so use STC Go to the serial port tool to download the program to the general USB The correct steps are

1. Use STC Universal USB The serial port conversion tool will be burned MCU Connect to the computer；
2. open STC of ISP Download software；
3. Choose the MCU model;
4. choose STC general purpose The serial port corresponding to the serial port tool；
5. USB Format or BIN Format)；
6. Open the target file (HEX The "Download Programming" button in the download software；
7. click ISP Go to the "power switch" on the serial port tool to MCU Power supply, you can start the download.

Click general STC USB

【Cold start burning】

In addition，USB Interface with Micro-USB The interface is the same function, and the user can connect one of the interfaces to the computer as ne

The download of the 0/1 Inside the signal pin 470 Ohmic resistance to ground, if set P1.0/P1.1=0/0 or P3.2/P3.3=0/0 Time to be able to

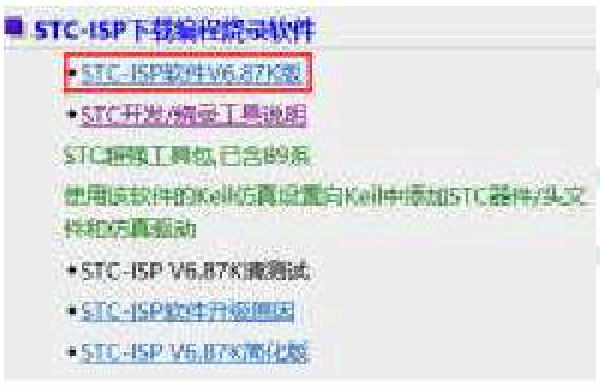
programming interface can be P1.1 P3.2，P3.3 Received 0V Signal foot.

F. 4.3 STC Universal USB To serial port tool driver installation

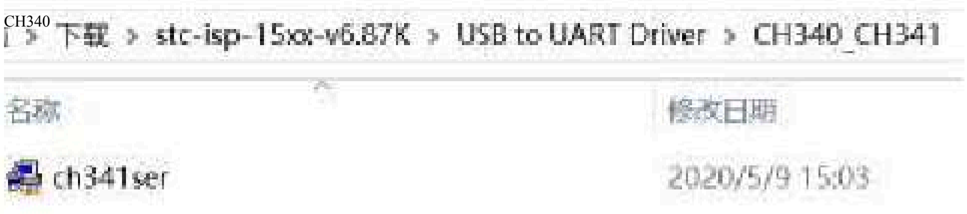
STC Universal USB To serial port tool adopts (You can plug in an external crystal oscillator. More accurate download, the universal CH340，

Just install the serial port driver, the STC To serial chip CH340 www.STCMCUDATA.com) provided CH341SER Serial drive

following is the dynamic download location： USB Official website (


■ STC-ISP下载编程烧录软件
• STC-ISP软件V6.87K版
• STC开发/烧录工具说明
STC仿真工具包,已含89系
使用该软件的Keil仿真设置(向Keil中添加STC器件/头文件和仿真驱动
• STC-ISP V6.87K版测试
• STC-ISP软件升级日志
• STC-ISP V6.87K简化版

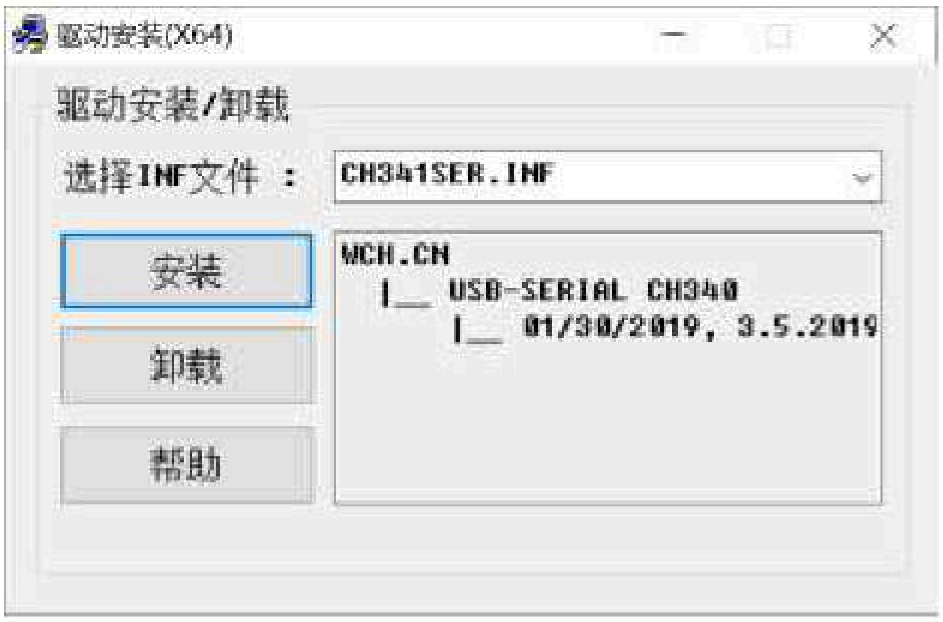
Unzip after downloading， The path of the driver installation package stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341：

CH340

下载 › stc-isp-15xx-v6.87K › USB to UART Driver › CH340_CH341

名称	修改日期
ch341ser	2020/5/9 15:03

Provided by the official website CH341SER Take the serial driver as an example, double-click the Install the package, click "Install" on the main interfac

Install" button to start installing the driver：

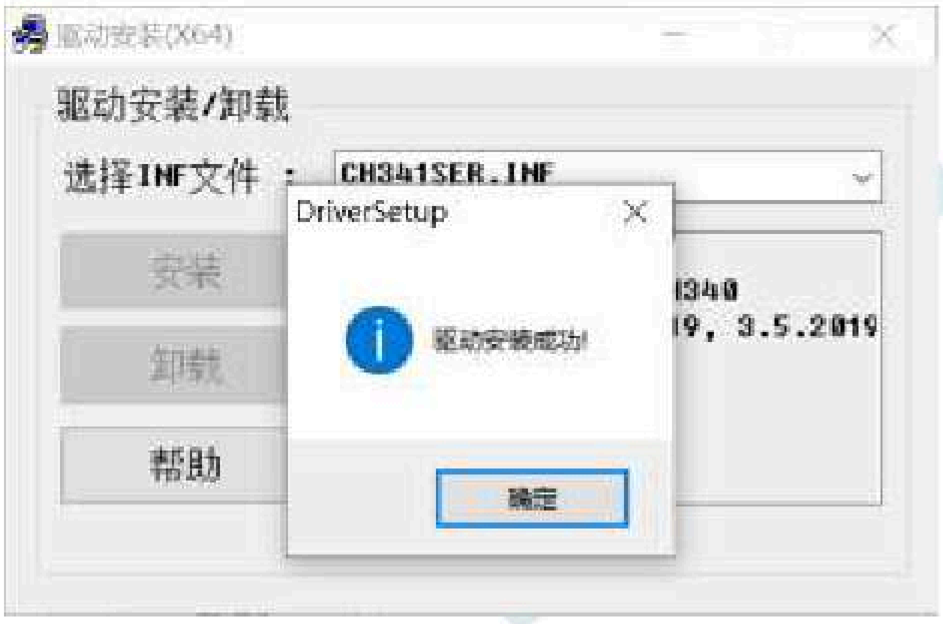

驱动安装(X64)

驱动安装/卸载

选择INF文件 : CH341SER.INF

```
WCH.CN
  |__ USB-SERIAL CH340
      |__ 01/30/2019, 3.5.2019
```

Then the driver installation success dialog box pops up, click the "OK" button to complete the installation :


驱动安装(X64)

驱动安装/卸载

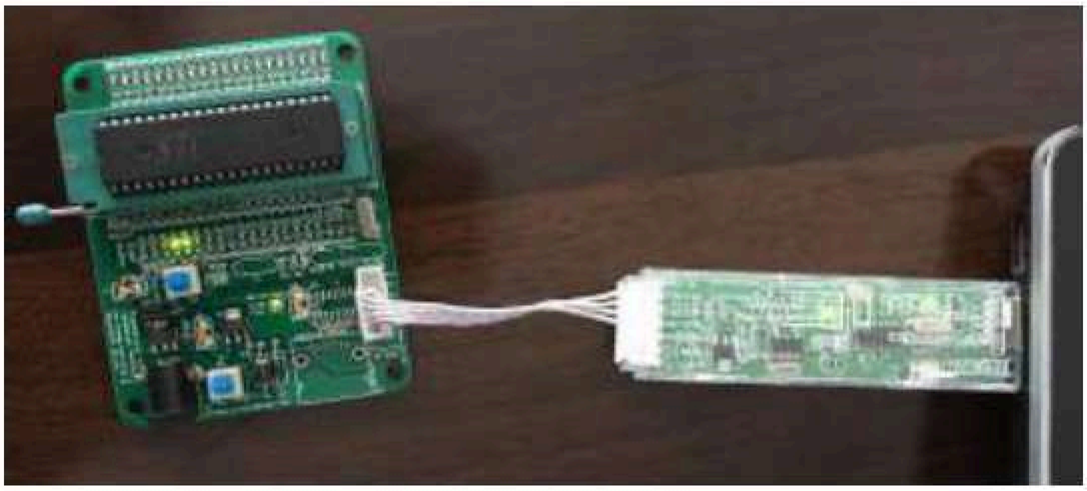
选择INF文件 : CH341SER.INF

DriverSetup

驱动安装成功!

F. 4.4 use STC Universal USB Go to the serial port tool to download the program to MCU

1. use STC Universal USB The serial port conversion tool will be MCU Connect to the computer :



2. software ;

3. STC-ISP Open and select

4. the model corresponding to the burning chip ; choose STC Universal USB The serial port number identified by the STC Universal USB serial port transfer tool (when After the serial port tool is correctly connected to the co

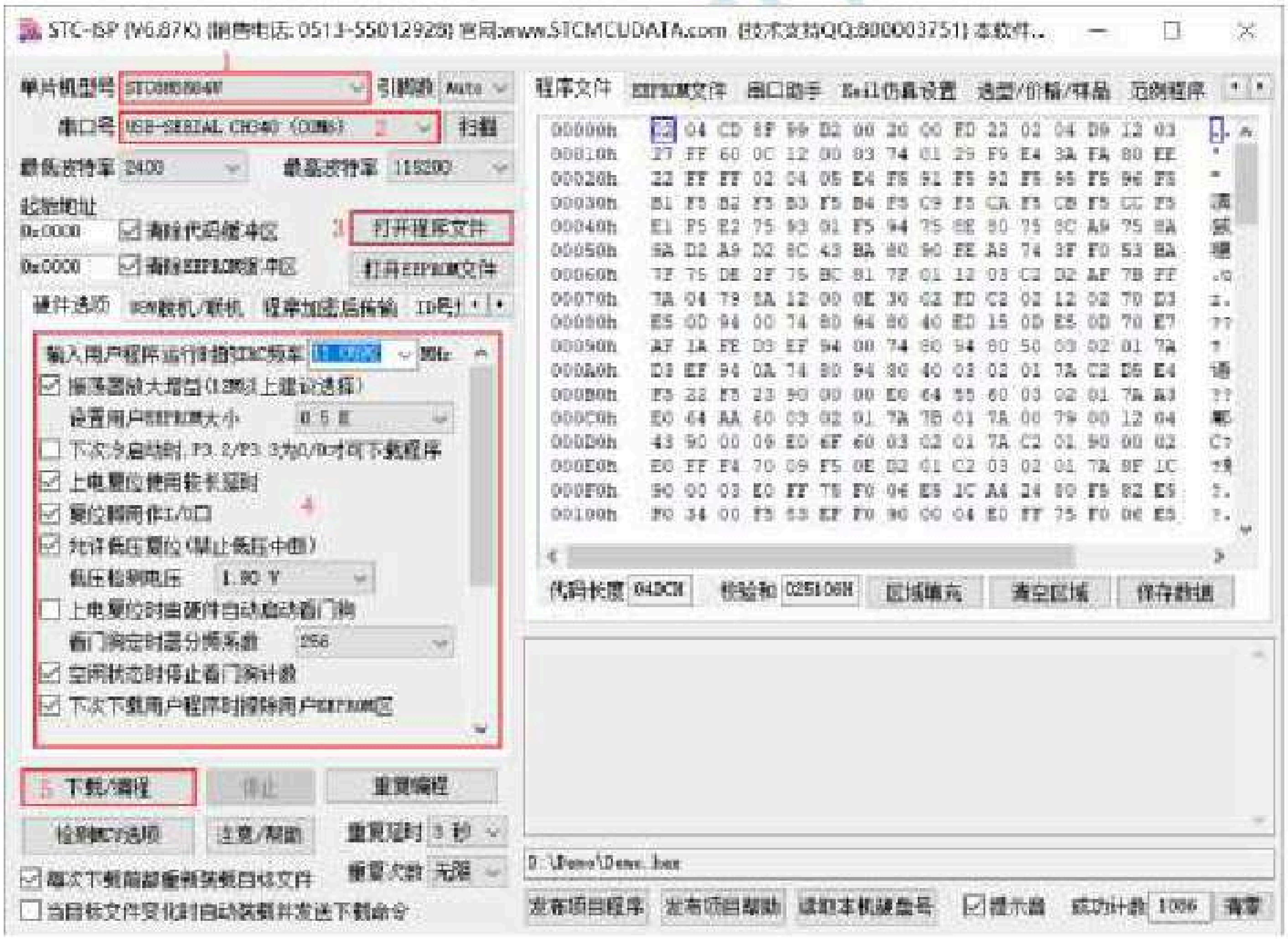
The software will automatically scan and recognize USB-SERIAL CH340 (COMx) "Serial port, when the specific COM The number will vary depending on the co

the name as " different", USB When there are multiple serial-to-serial cable is connected to the computer, you must manually select it. ;

5. Load the burning program;

6. set the burning options; click

7. "Download/"Program" button ;



STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件..

单片机型号 [illegible] 引脚数 Auto

串口号 USB-SERIAL CH340 (COM[illegible]) 扫描

最低波特率 2400 最高波特率 115200

起始地址

0x0000 ☑ 清除代码缓冲区 打开程序文件

0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件

硬件选项 | USB脱机/联机 | 程序加密后传输 | ID号

输入用户程序运行时的IRC频率 [illegible] MHz

☑ 振荡器放大增益(12MHz以上建议选择)

设置用户EEPROM大小 0.5 K

☐ 下次冷启动时, P3.2/P3.3为0/0才可下载程序

☑ 上电复位使用较长延时

☑ 复位脚用作I/O口

☑ 允许低压复位(禁止低压中断)

低压检测电压 1.90 V

☐ 上电复位时由硬件自动启动看门狗

看门狗定时器分频系数 256

☑ 空闲状态时停止看门狗计数

☑ 下次下载用户程序时擦除用户EEPROM区

下载/编程 | 停止 | 重复编程

检测MCU选项 | 注意/帮助 | 重复延时 3 秒

重复次数 无限

☑ 每次下载前都重新装载目标文件

☐ 当目标文件变化时自动装载并发送下载命令

程序文件 | EEPROM文件 | 串口助手 | Keil仿真设置 | 选型/价格/样品 | 范例程序

```
00000h  02 04 CD 8F 99 D2 00 20 00 FD 22 02 04 D9 12 03
00010h  27 FF 60 0C 12 00 83 74 01 29 F9 E4 3A FA 80 EE
00020h  22 FF FF 02 04 05 E4 F5 91 F5 92 F5 95 F5 96 F5
00030h  B1 F5 B2 F5 B3 F5 B4 F5 C9 F5 CA F5 CB F5 CC F5
00040h  E1 F5 E2 75 93 01 F5 94 75 8E 80 75 8C A8 75 8A
00050h  9A D2 A9 D2 8C 43 BA 80 90 FE A8 74 3F F0 53 BA
00060h  7F 75 DE 2F 75 BC 81 7F 01 12 03 C2 D2 AF 7B FF
00070h  7A 04 79 8A 12 00 0E 30 02 FD C2 02 12 02 70 D3
00080h  E5 0D 94 00 74 80 94 80 40 ED 15 0D E5 0D 70 E7
00090h  AF 1A FE D3 EF 94 00 74 80 94 80 50 03 02 01 7A
000A0h  D3 EF 94 0A 74 80 94 80 40 03 02 01 7A C2 D6 E4
000B0h  F5 22 F5 23 90 00 00 E0 64 55 60 03 02 01 7A 83
000C0h  E0 64 AA 60 03 02 01 7A 7B 01 7A 00 79 00 12 04
000D0h  43 90 00 09 E0 6F 60 03 02 01 7A C2 01 90 00 02
000E0h  E0 FF F4 70 09 F5 0E D2 01 C2 03 02 01 7A 8F 1C
000F0h  90 00 03 E0 FF 75 F0 06 E5 1C A4 24 80 F5 82 E5
00100h  F0 34 00 F5 83 EF F0 90 00 04 E0 FF 75 F0 06 E5
```

代码长度 04DCH 校验和 025106H 区域填充 清空区域 保存数据

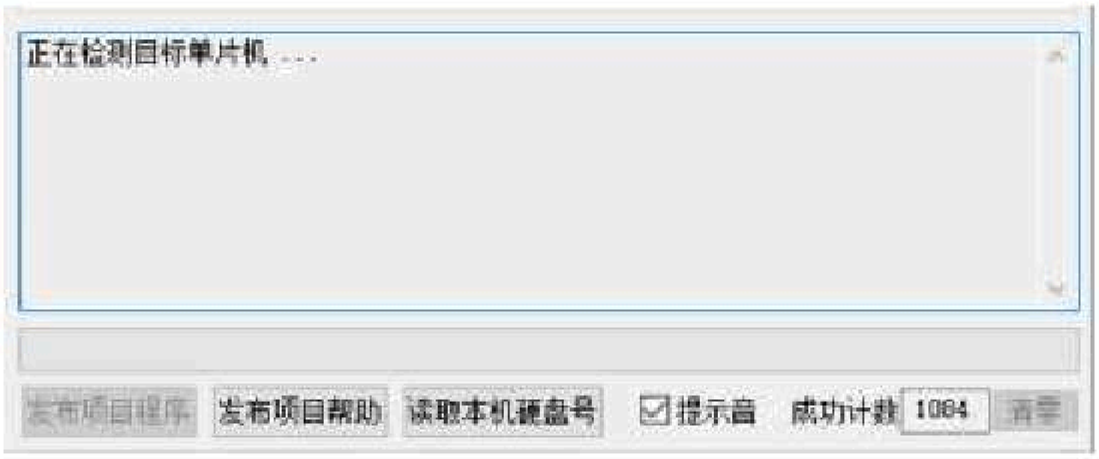
D:\Demo\Demo.hex

发布项目程序 | 发布项目帮助 | 读取本机硬盘号 | ☑ 提示音 | 成功计数 1006 | 清零

The prompt box in the lower right corner shows "The "Click when target MCU is being STC Universal USB detected" 8. Go to "Power on" on the serial port tool

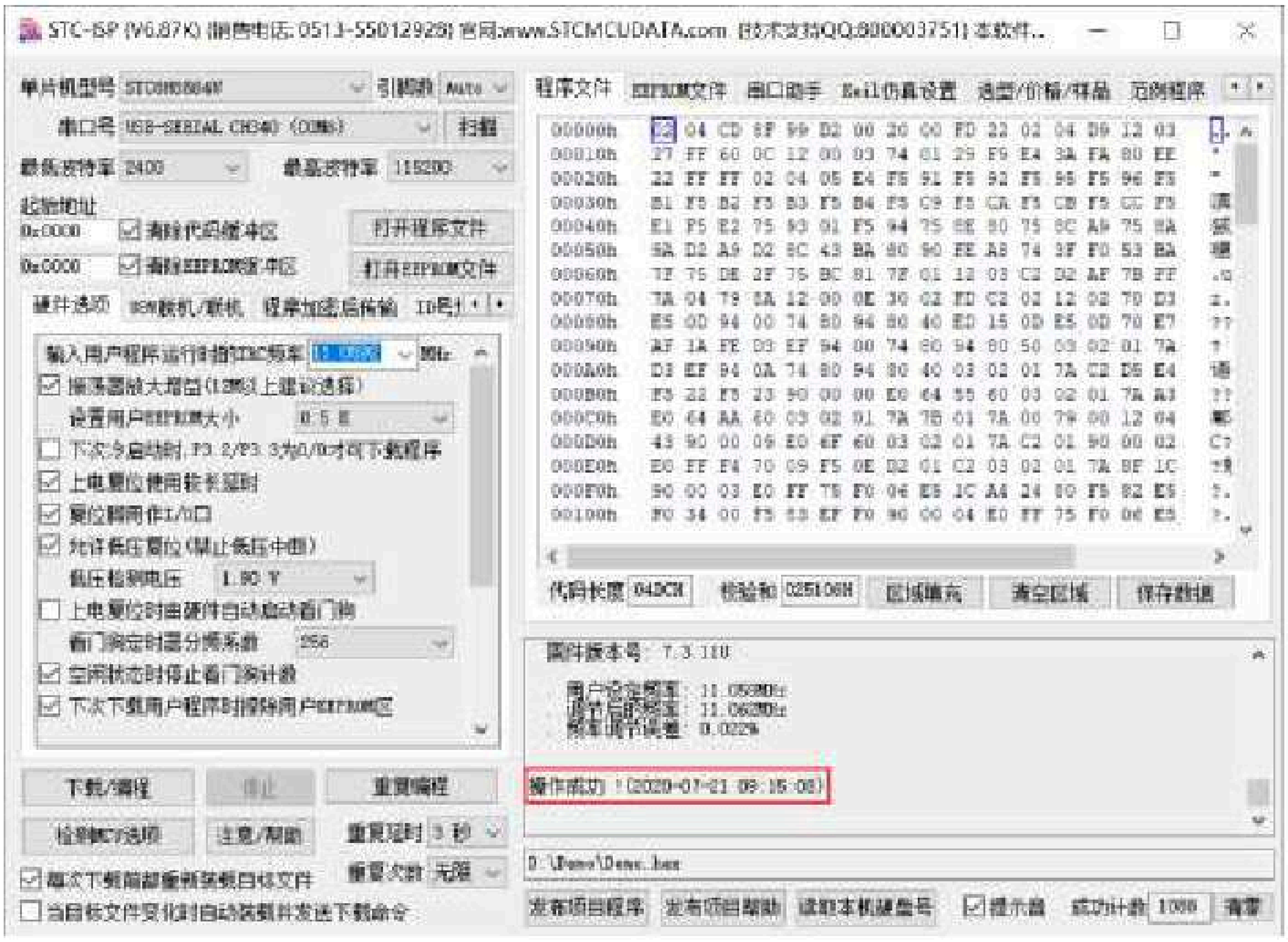
Turn off the "power supply, you can start the download;

【Cold start burning】 MCU


正在检测目标单片机 ...

发布项目程序 发布项目帮助 读取本机硬盘号 ☑提示音 成功计数 1084 清零

9. Wait for the download to end. If the download is successful, the prompt box in the lower right corner will display "Operation successful!"


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件...

单片机型号 STC8H8K64U 引脚数 Auto
串口号 USB-SERIAL CH340 (COM6)
最低波特率 2400 最高波特率 115200
起始地址 0x0000 ☑清除代码缓冲区 0x0000 ☑清除EEPROM缓冲区

输入用户程序运行时的IRC频率 11.0592 MHz
☑ 振荡器放大增益(12MHz以上建议选择)
设置用户EEPROM大小 0.5 K
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 1.90 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区

固件版本号: 7.3.11U
用户设定频率: 11.0592MHz
调节后的频率: 11.062MHz
频率调节误差: 0.022%

操作成功 !(2020-07-21 09:15:08)

D:\Demo\Demo.hex

代码长度 042CH 校验和 025106H

重复延时 3 秒 重复次数 无限
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

成功计数 1088

F. 4.5 use STC Universal USB To serial port tool to simulate user code

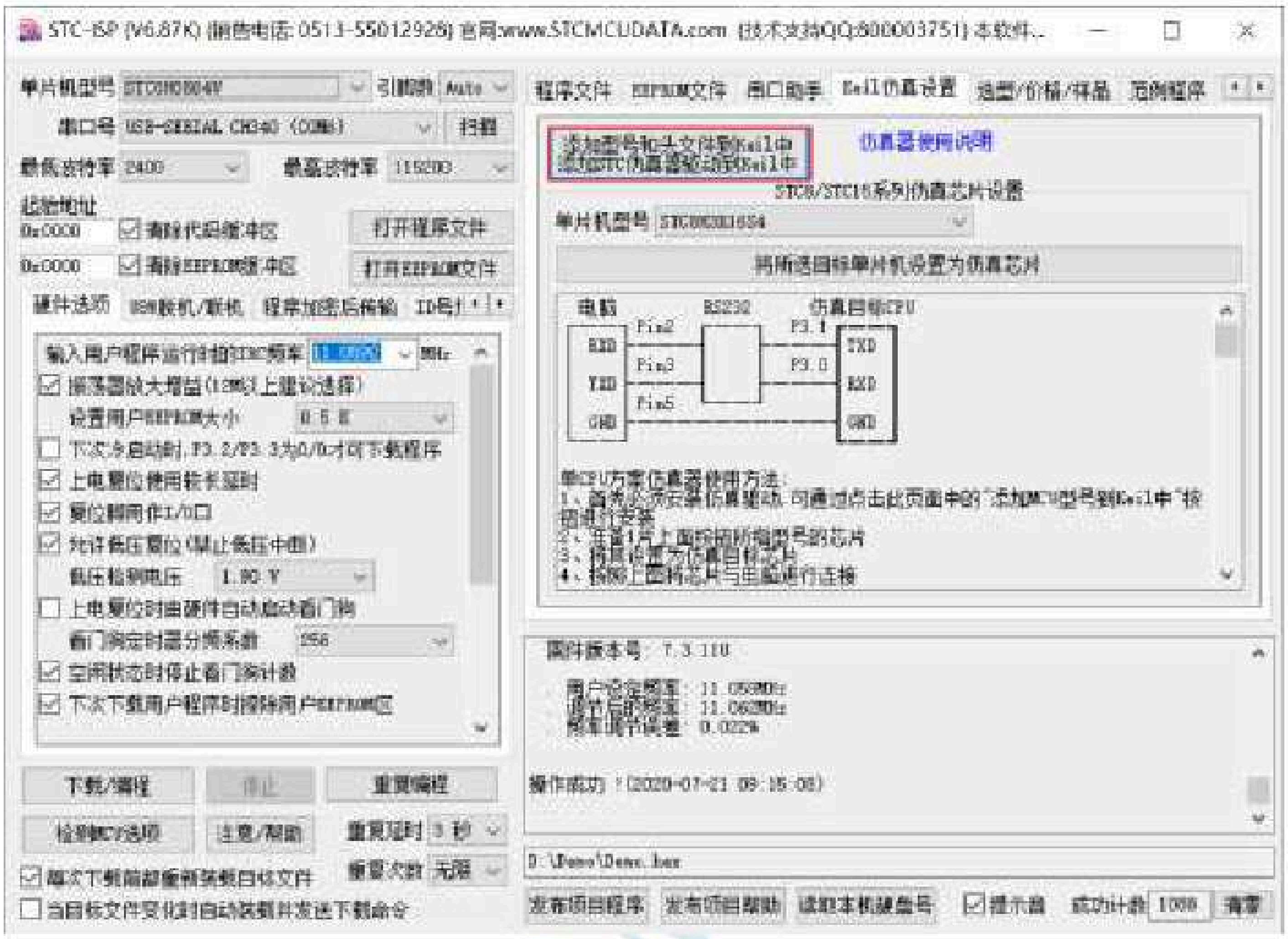
Currently STC The simulation is based on Keil Environmental, so if you need to use STC Universal USB If you go to the serial port tool to simulate the user cod need to be installed Keil software.

Software installation After completion, you also need to install STC Simulation driver Keil The installation steps of the simulation driver are as follows :

First open the download software ; STC-ISP

Then in the functional area on the right side of the software, "Keil Simulation Settings" page, click "Add model and header file to Keil Emulator

Drive to Keil "Medium" button：



STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件...

单片机型号 STC8H8K64U 引脚数 Auto
串口号 USB-SERIAL CH340 (COM6) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区 打开程序文件
0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 USB型联机/脱机 程序加密后传输 ID号

输入用户程序运行时的IRC频率 11.0592 MHz
☑ 振荡器放大增益(12MHz以上建议选择)
设置用户EEPROM大小 0.5 K
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 1.90 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区

下载/编程 停止 重复编程
检测MCU选项 注意/帮助 重复延时 3 秒
☑ 每次下载前都重新装载目标文件 重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

程序文件 EEPROM文件 串口助手 Keil仿真设置 选型/价格/样品 范例程序

添加型号和头文件到Keil中
添加STC仿真器驱动到Keil中
仿真器使用说明

STC8/STC15系列仿真芯片设置
单片机型号 STC8H8K64U
将所选目标单片机设置为仿真芯片

单CPU方案仿真器使用方法:
1、首先必须安装仿真驱动,可通过点击此页面中的"添加MCU型号到Keil中"按钮进行安装
2、在如上图所示的选项中选择芯片
3、将所选芯片设置为仿真芯片
4、按照上图将芯片与电脑进行连接

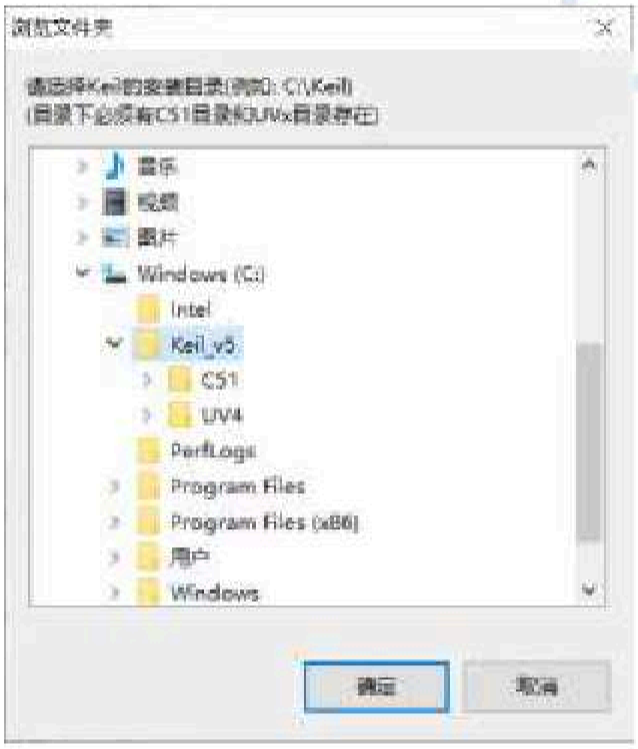
固件版本号: 7.3.11U
用户设定频率: 11.059MHz
调节后的频率: 11.062MHz
频率调节误差: 0.022%

操作成功 !(2020-07-21 09:15:08)

D:\Demo\Demo.hex

发布项目程序 发布项目帮助 读取本机硬盘号 ☑ 提示音 成功计数 1000 清零

After pressing, the following screen will appear：



浏览文件夹

请选择Keil的安装目录(例如: C:\Keil)
(目录下必须有C51目录和UV4目录存在)

Locate the directory Keil to The installation directory of the software, and then determine.

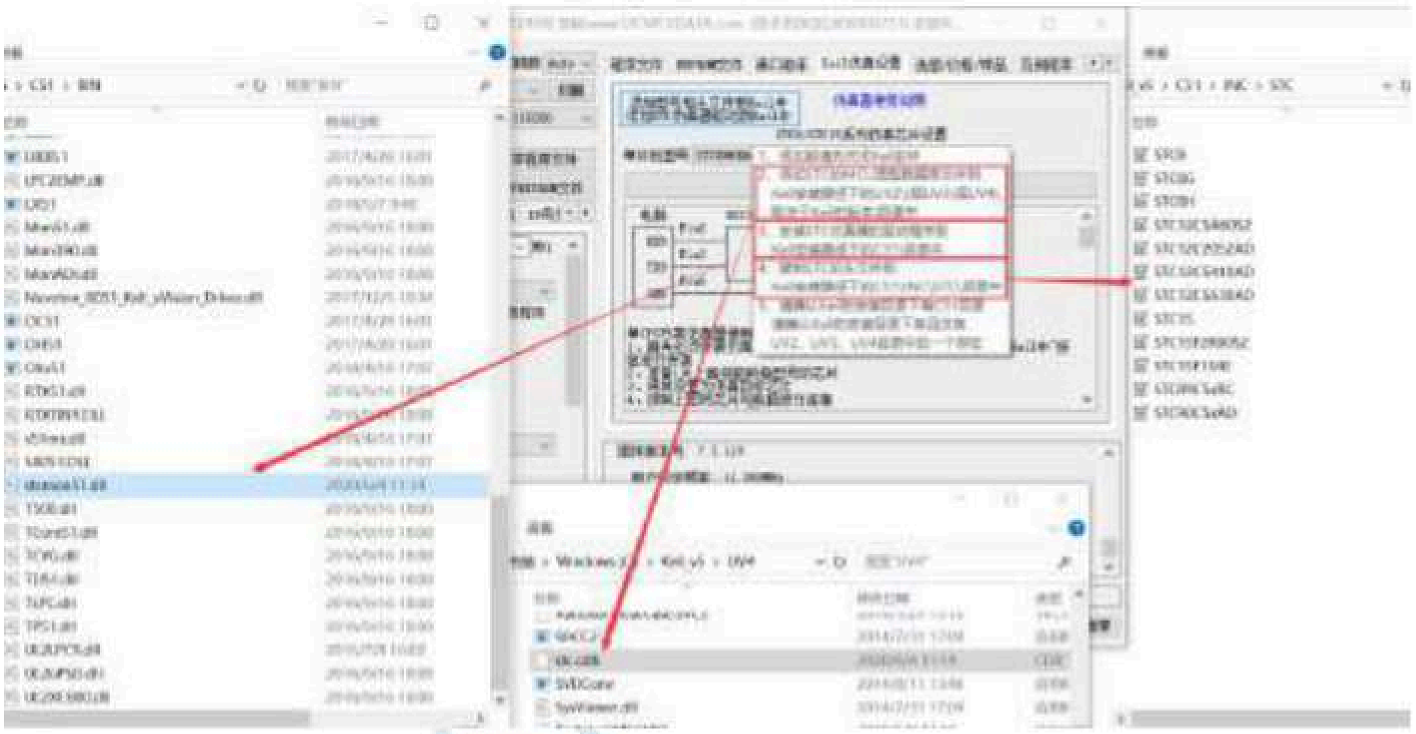
After the installation is successful, the following prompt box will pop up：

STC-ISP

STC MCU型号添加成功！

确定

in Keil The following files can be seen in the relevant directory, which means that the driver is installed correctly.



Since in the default state， STC The main control chip is not a simulation chip and does not have a simulation function,

Also need to STC so if simulation is required, the main control chip is set to a simulation chip.

The steps to make an simulation chip are as follows：

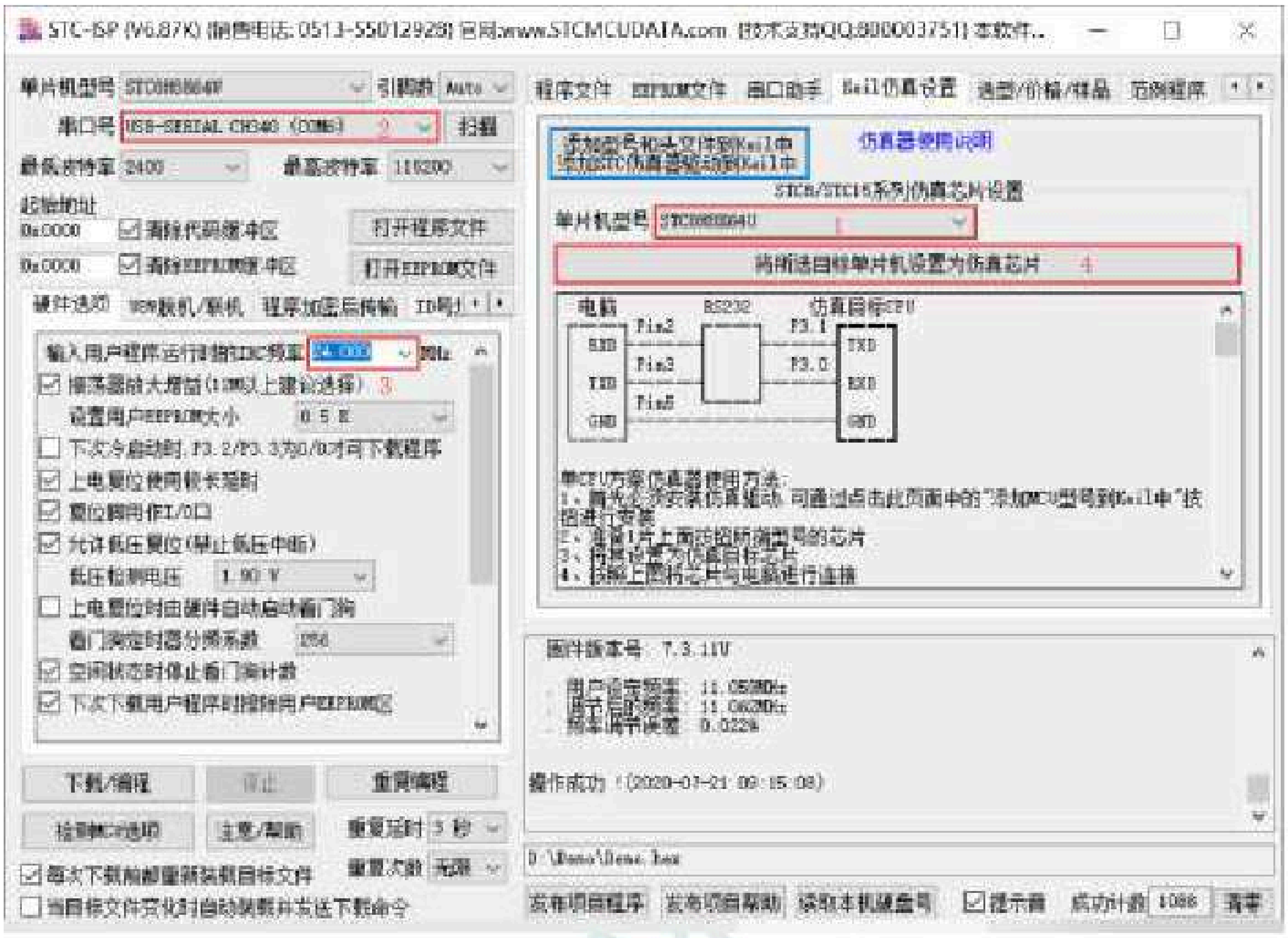
First use STC Universal USB The serial port tool will MCU Connect to the computer；

and then open STC ISP Download the software, and select the serial port number corresponding to the serial

the selection port tool in the drop-down list of serial port number; MCU model；

Select the user program to run IRC Frequency, the frequency selected when making the simulation chip is consistent with the frequency set by the

Can achieve real operating results.

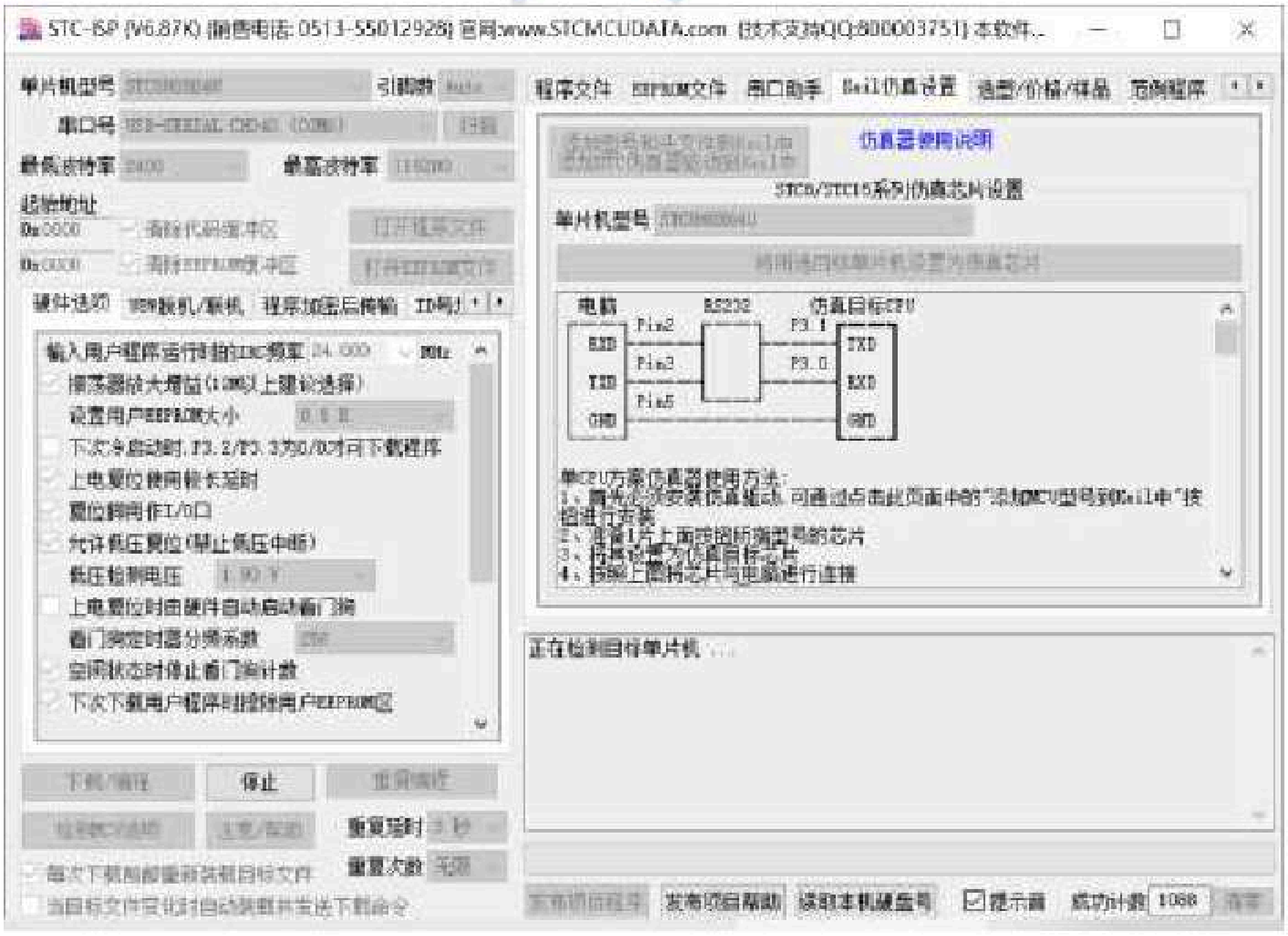

STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件..

单片机型号 STC8H8K64U　引脚数 Auto
串口号 USB-SERIAL CH340 (COM5) 2　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区　打开程序文件
0x0000 ☑ 清除EEPROM缓冲区　打开EEPROM文件
硬件选项　USB联机/脱机　程序加密后传输　ID号!
输入用户程序运行时的IRC频率 24.000 MHz
☑ 振荡器放大增益(12MHz以上建议选择) 3
设置用户EEPROM大小 0.5 K
☐ 下次冷启动时，P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 1.90 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区
下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 3 秒
☑ 每次下载前都重新装载目标文件　重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

程序文件　EEPROM文件　串口助手　Keil仿真设置　选型/价格/样品　范例程序
添加型号和头文件到Keil中 添加STC仿真器驱动到Keil中 1　仿真器使用说明
STC8/STC15系列仿真芯片设置
单片机型号 STC8H8K64U 1
将所选目标单片机设置为仿真芯片 4
电脑 RS232 仿真目标CPU
RXD Pin2 P3.1 TXD
TXD Pin3 P3.0 RXD
GND Pin5 GND
单CPU方案仿真器使用方法:
1、首先必须安装仿真器驱动，可通过点击此页面中的"添加MCU型号到Keil中"按钮进行安装
2、准备1片上面所选所需型号的芯片
3、将其设置为仿真目标芯片
4、按照上图将芯片与电脑进行连接

固件版本号 7.3.11U
用户设定频率: 11.0592MHz
调节后的频率: 11.062MHz
频率调节误差: 0.022%
操作成功 !(2020-07-21 09:15:08)
D:\Demo\Demo.hex
发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 1086　清零

Then in the functional area on the right side of the "Simulation Settings" page, click the "Set the selected target MCU as an simulation chip". After pressing, the following screen will appear :


STC-ISP (V6.87K) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件..

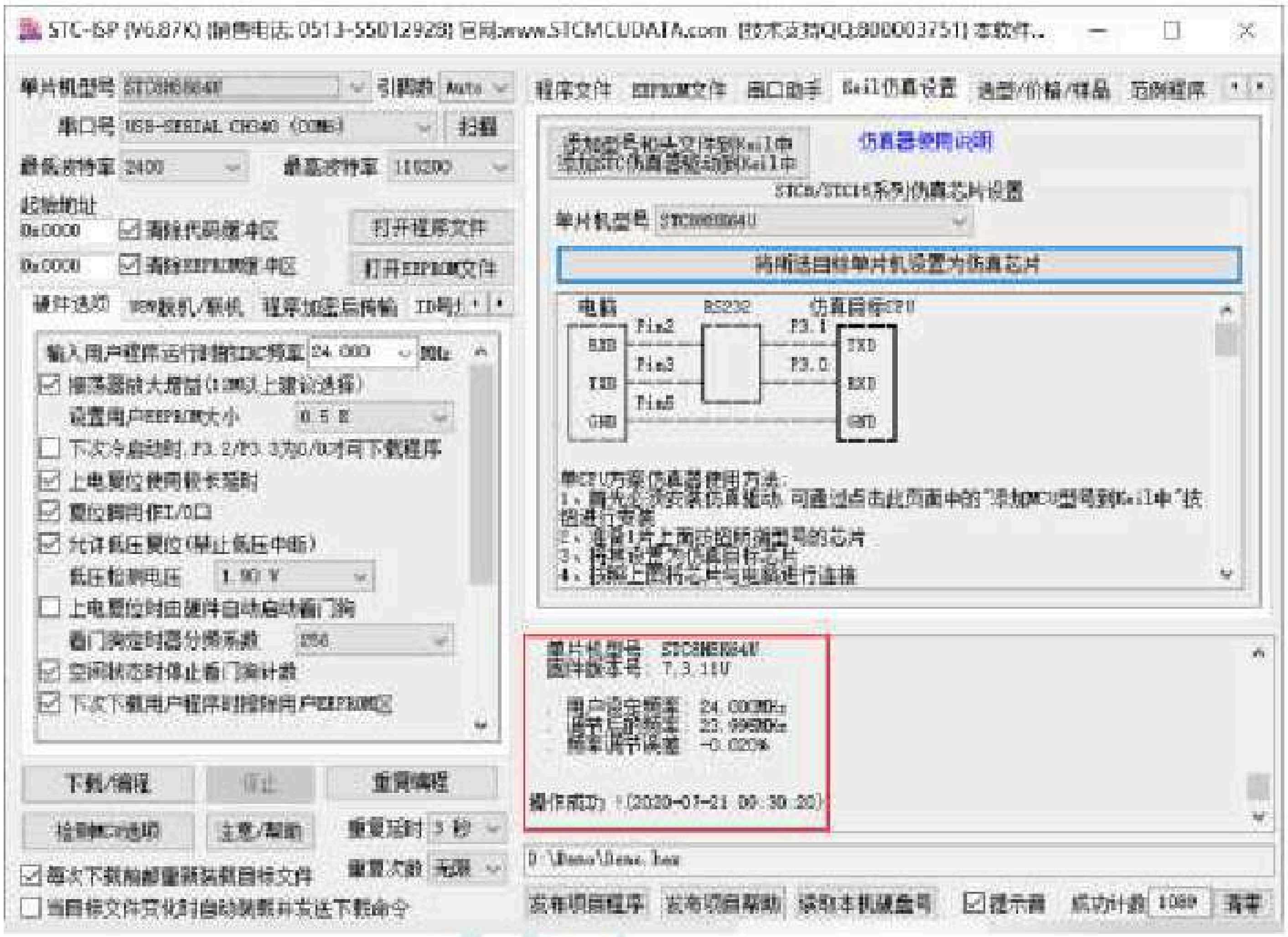
单片机型号 STC8H8K64U　引脚数 Auto
串口号 USB-SERIAL CH340 (COM5)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000 清除代码缓冲区　打开程序文件
0x0000 清除EEPROM缓冲区　打开EEPROM文件
硬件选项　USB联机/脱机　程序加密后传输　ID号!
输入用户程序运行时的IRC频率 24.000 MHz
振荡器放大增益(12MHz以上建议选择)
设置用户EEPROM大小 0.5 K
下次冷启动时，P3.2/P3.3为0/0才可下载程序
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 1.90 V
上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
空闲状态时停止看门狗计数
下次下载用户程序时擦除用户EEPROM区
下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 3 秒
每次下载前都重新装载目标文件　重复次数 无限
当目标文件变化时自动装载并发送下载命令

程序文件　EEPROM文件　串口助手　Keil仿真设置　选型/价格/样品　范例程序
添加型号和头文件到Keil中 添加STC仿真器驱动到Keil中　仿真器使用说明
STC8/STC15系列仿真芯片设置
单片机型号 STC8H8K64U
将所选目标单片机设置为仿真芯片
电脑 RS232 仿真目标CPU
RXD Pin2 P3.1 TXD
TXD Pin3 P3.0 RXD
GND Pin5 GND
单CPU方案仿真器使用方法:
1、首先必须安装仿真器驱动，可通过点击此页面中的"添加MCU型号到Keil中"按钮进行安装
2、准备1片上面所选所需型号的芯片
3、将其设置为仿真目标芯片
4、按照上图将芯片与电脑进行连接

正在检测目标单片机 ...
发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 1088　清零

Next you need to click STC Universal USB Go to the "power switch" on the serial port tool to , You can start making power supply Simulation chip.

If the setting is successful, the following screen will appear :


STC-ISP (V6.87K) [销售电话: 0513-55012928] 官网:www.STCMCUDATA.com [技术支持QQ:800003751] 本软件...

单片机型号 STC8H8K64U 引脚数 Auto
串口号 USB-SERIAL CH340 (COM5) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 ☑ 清除代码缓冲区 打开程序文件
0x0000 ☑ 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 USB脱机/脱机 程序加密后传输 ID号
输入用户程序运行时的IRC频率 24.000 MHz
☑ 振荡器放大增益(12MHz以上建议选择)
设置用户EEPROM大小 0.5 K
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 1.90 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区
下载/编程 停止 重复编程
检测MCU选项 注意/帮助 重复延时 3 秒
☑ 每次下载前都重新装载目标文件 重复次数 无限
☐ 当目标文件变化时自动装载并发送下载命令

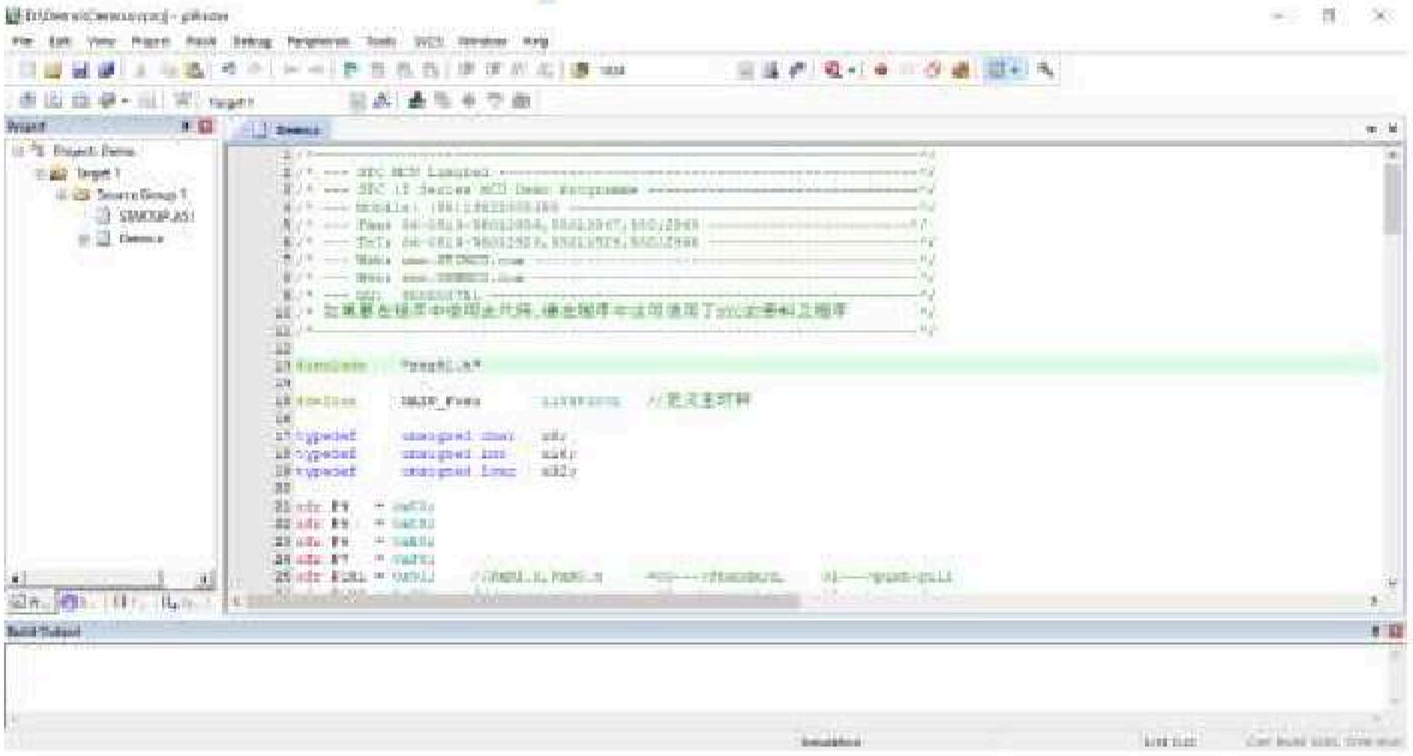
程序文件 EEPROM文件 串口助手 Keil仿真设置 选型/价格/样品 范例程序
添加型号和头文件到Keil中 添加STC仿真器驱动到Keil中 仿真器使用说明
STC8/STC16系列仿真芯片设置
单片机型号 STC8H8K64U
将所选目标单片机设置为仿真芯片
电脑 RS232 仿真目标CPU
RXD TXD GND Pin2 Pin3 Pin5 P3.1 P3.0 TXD RXD GND
单CPU方案仿真器使用方法:
1、首先必须安装仿真驱动。可通过点击此页面中的"添加MCU型号到Keil中"按钮进行安装
2、选择1所上面的选所选型号的芯片
3、将所选芯片为仿真目标芯片
4、按照上图将芯片与电脑进行连接

单片机型号 STC8H8K64U
固件版本号 7.3.11U
用户设定频率 24.000MHz
调节后的频率 23.998MHz
频率调节误差 -0.020%
操作成功 !(2020-07-21 00:30:20)

D:\Demo\Demo.hex
发布项目程序 发布项目帮助 读取本机硬盘号 ☑ 提示音 成功计数 1089 清零

At this point, the simulation chip was successfully produced.

Next we open a project for simulation :


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Project: Demo
Target 1
Source Group 1
STARTUP.A51
Demo.c

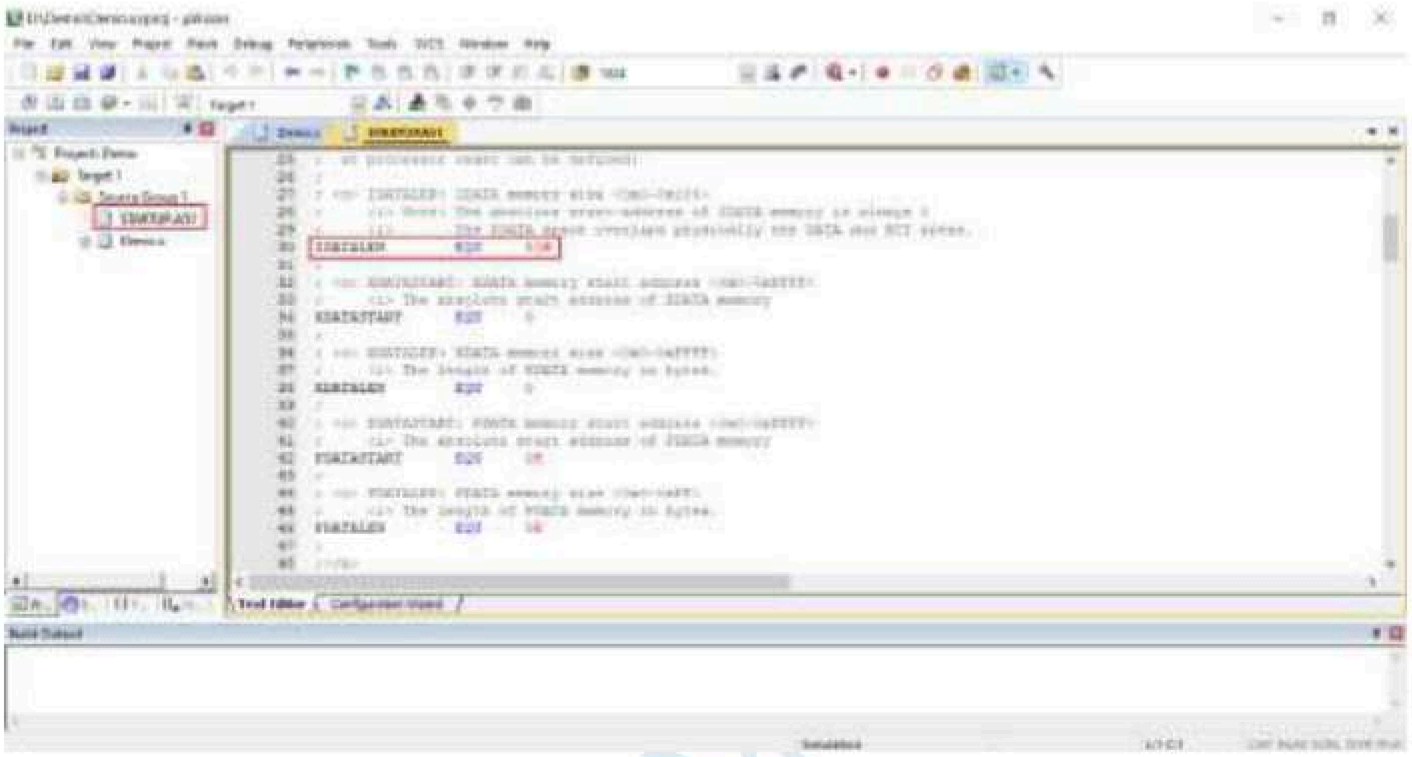
```
/*--- STC MCU Limited ---------------------------------------*/
/*--- STC 1T Series MCU Demo Programme ----------------------*/
/*--- Mobile: [illegible] -----------------------------------*/
/*--- Fax: [illegible] --------------------------------------*/
/*--- Tel: [illegible] --------------------------------------*/
/*--- Web: www.STCMCU.com -----------------------------------*/
/*--- Web: www.GXWMCU.com -----------------------------------*/
/*--- QQ:  800003751 ----------------------------------------*/
/* 如果要在程序中使用此代码,请在程序中注明使用了STC的资料及程序 */

#include    "reg51.h"

#define     MAIN_Fosc       11059200L   //定义主时钟

typedef     unsigned char   u8;
typedef     unsigned int    u16;
typedef     unsigned long   u32;

sfr P0      = 0x80;
sfr P1      = 0x90;
sfr P2      = 0xA0;
sfr P3      = 0xB0;
sfr P1M1    = 0x91;    //P1M1.n,P1M0.n    =00--->Standard,    01--->push-pull
```

Build Output
Simulation

Then make the following project settings :

An additional

note: when it is created C language project, and there is a macro "When added to the project, there is one named STARTUP. A51 is " IDATALEN definition of the startup file " ", which is used to define the IDATA The size of the default value is 128, That is, hexadecimal 80H, When it is also in the startup file that needs of 0 IDATA The same size. So when defined as IDATA 80H, then STARTUP. A51 to be initialized to the code inside, it will be of RAM IDATA Initialized to; similarly, if it is defined as IDATA 0FFH, It will IDATA0 00-7F of 00-FF RAM 0 Initialized to.



```
; <o> IDATALEN: IDATA memory size <0x0-0x100>
;     <i> Note: The absolute start-address of IDATA memory is always 0
;     <i>       The IDATA space overlaps physically the DATA and BIT areas.
IDATALEN        EQU     256
;
; <o> XDATASTART: XDATA memory start address <0x0-0xFFFF>
;     <i> The absolute start address of XDATA memory
XDATASTART      EQU     0
;
; <o> XDATALEN: XDATA memory size <0x0-0xFFFF>
;     <i> The length of XDATA memory in bytes.
XDATALEN        EQU     0
;
; <o> PDATASTART: PDATA memory start address <0x0-0xFFFF>
;     <i> The absolute start address of PDATA memory
PDATASTART      EQU     0H
;
; <o> PDATALEN: PDATA memory size <0x0-0xFF>
;     <i> The length of PDATA memory in bytes.
PDATALEN        EQU     0H
;
; </h>
```

What we chose STC12H The last byte of the series IDATA is The size is 256 Byte (00-7F of DATA and 80H-FFH of IDATA) ,

But because of the RAM microcontrollers is written 17 Number and related test parameters, if the user needs to use this part in the program ID Data, then be sure not to IDATALEN Defined as 256。

Press the shortcut key" Project "in" Alt+F7 Configure the Option for Target 'Target1'"

Option for Target 'Target1' 'In " step project in the "Or Select Menu" dialog box :

one1, Go to the project's settings page and select " Debug Step "Settings page ;

1".2, Select the hardware simulation on the right" Use ...";

step one3, Select "In the simulation driver drop-down STC Monitor-51 Driver "item ;

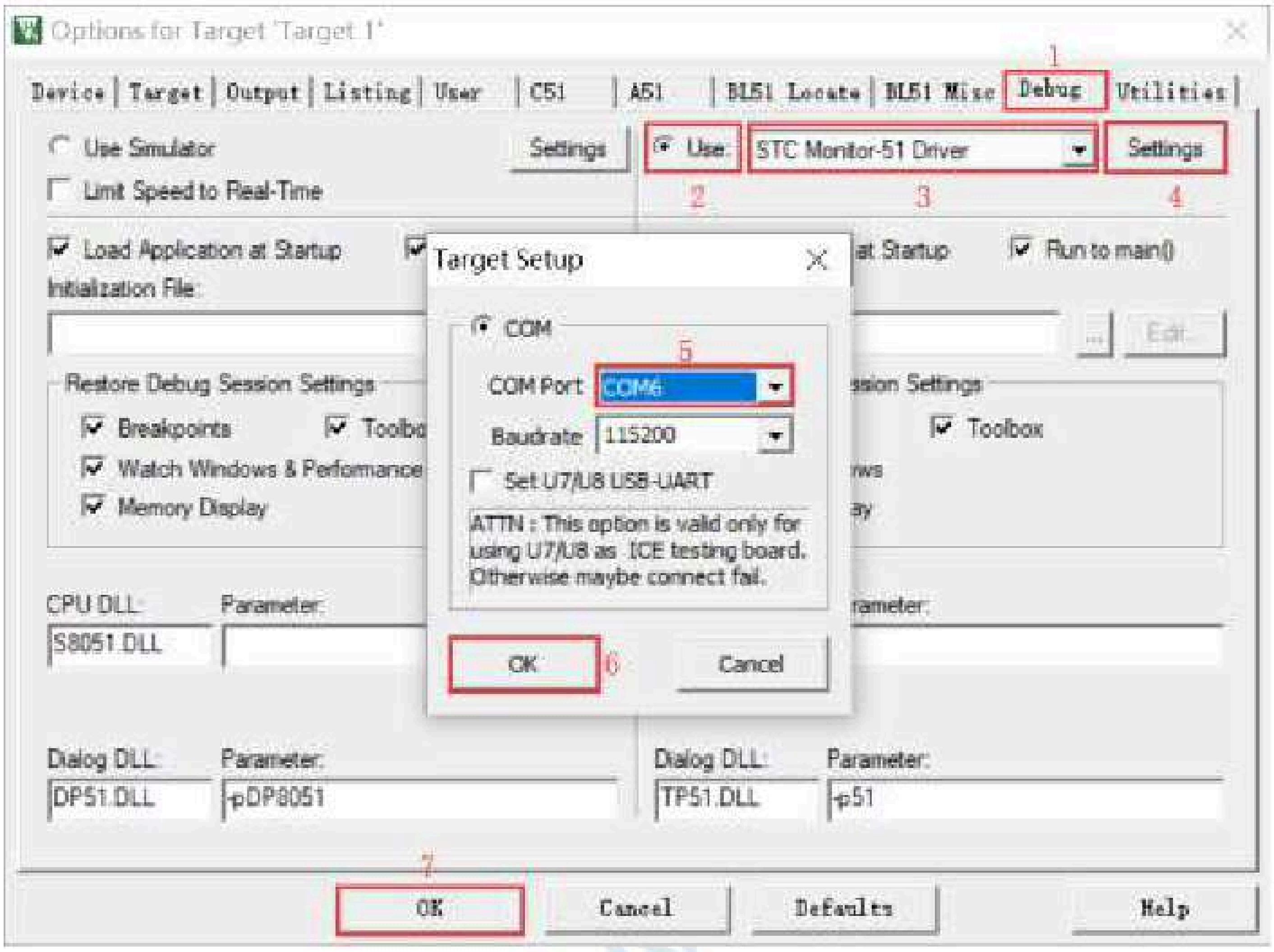
list" 4 Step 1. Click Settings "Button, enter the setting screen of the serial port; step 3. Set the

First 5 port number and baud rate of the serial port, and select the serial port number. Universal USB The serial port corresponding to the serial port to

General choice of baud rate 57600° or 115200

Make sure to complete the simulation settings.

The detailed steps are shown in the figure below :


Options for Target 'Target 1'

Device | Target | Output | Listing | User | C51 | A51 | BL51 Locate | BL51 Misc | Debug | Utilities

Use Simulator — Settings
Limit Speed to Real-Time
Use: STC Monitor-51 Driver — Settings
✓ Load Application at Startup
✓ Run to main()
Initialization File:
Restore Debug Session Settings
✓ Breakpoints ✓ Toolbox
✓ Watch Windows & Performance
✓ Memory Display
CPU DLL: S8051.DLL Parameter:
Dialog DLL: DP51.DLL Parameter: -pDP8051
Dialog DLL: TP51.DLL Parameter: -p51

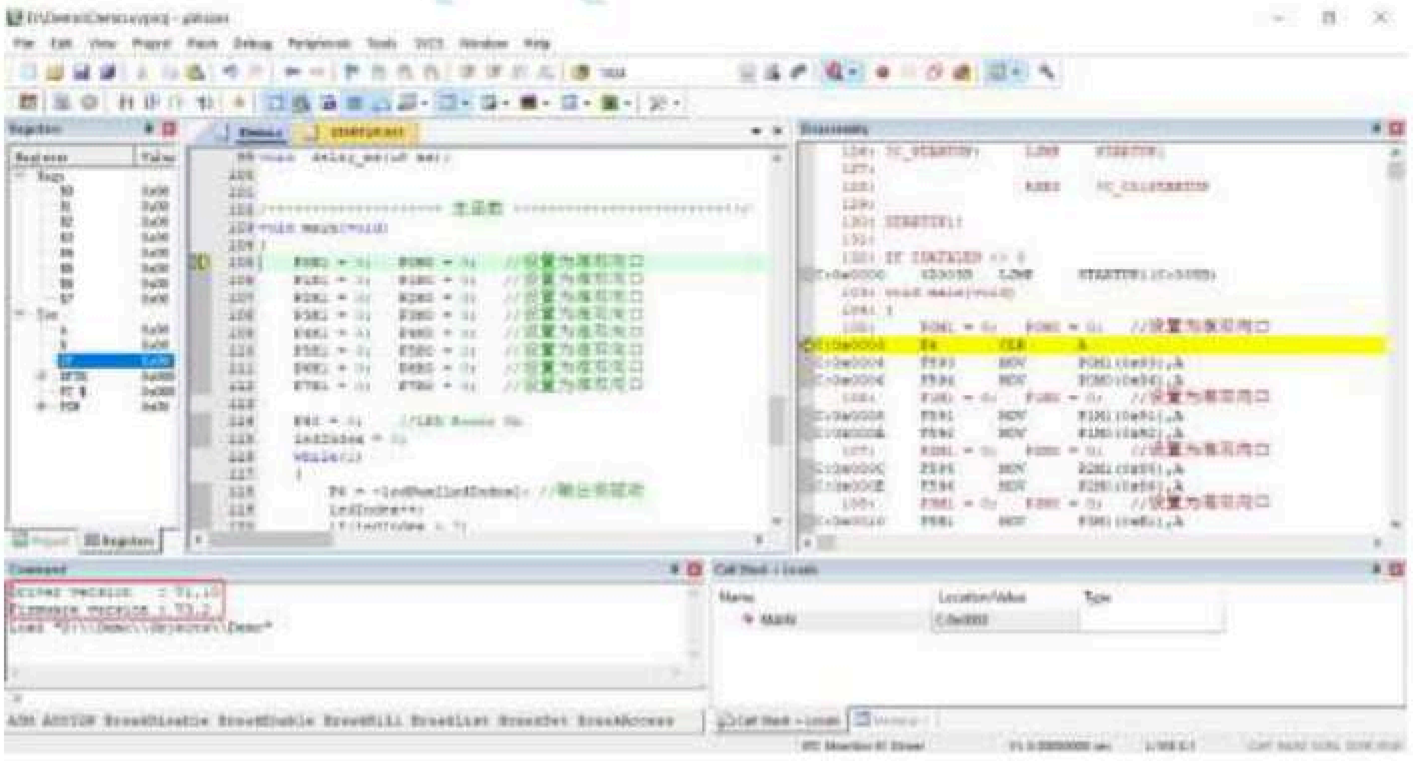
Target Setup
COM
COM Port: COM6
Baudrate: 115200
Set U7/U8 USB-UART
ATTN : This option is valid only for using U7/U8 as ICE testing board. Otherwise maybe connect fail.
OK Cancel

OK Cancel Defaults Help

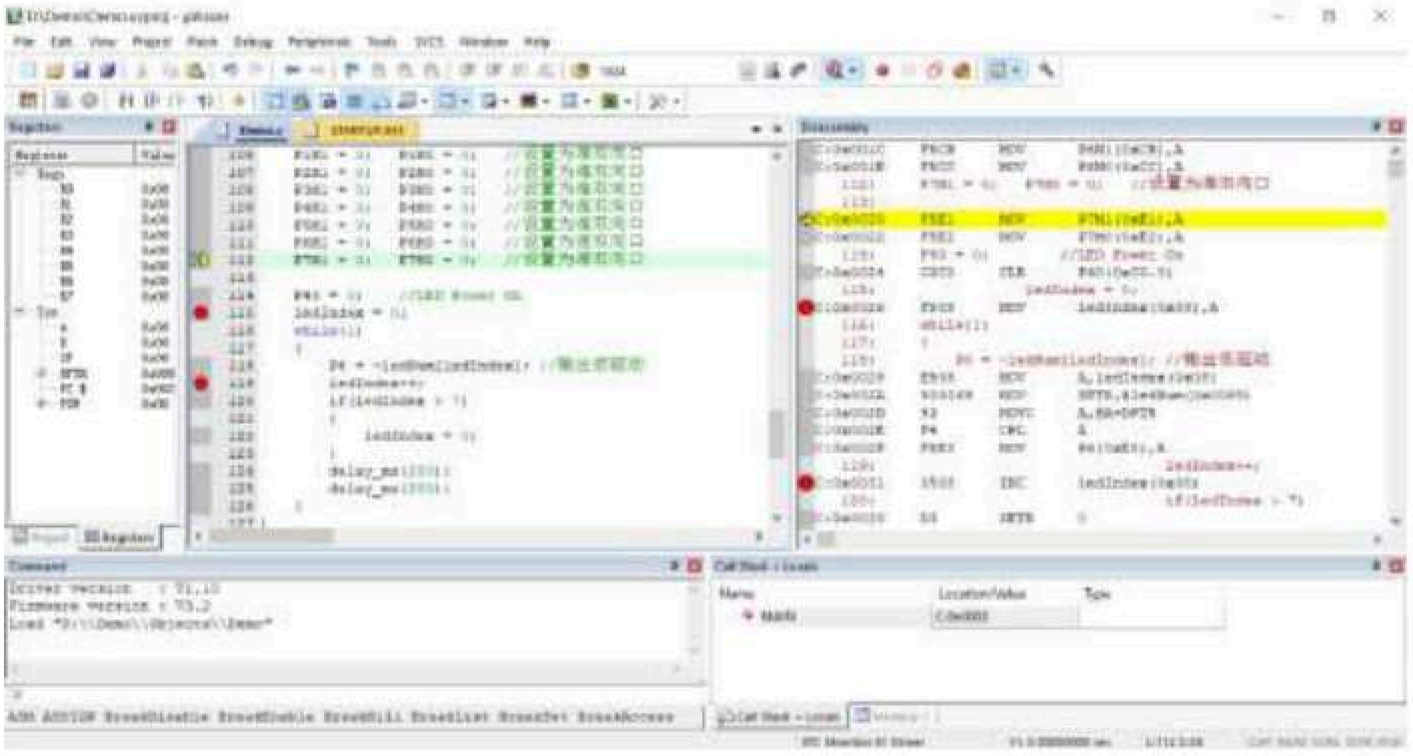
After completing all the above work, you can Press "in the software Ctrl+F5 "Start simulation debugging.

If the hardware connection is correct, it will enter a debugging interface similar to the following, and display the current simulation driver vers

number and the current simulation monitoring code firmware version number in the command output window, as shown in the figure below. :


```
Driver Version   : V1.10
Firmware Version : V3.2
Load "D:\\Demo\\Objects\\Demo"
```

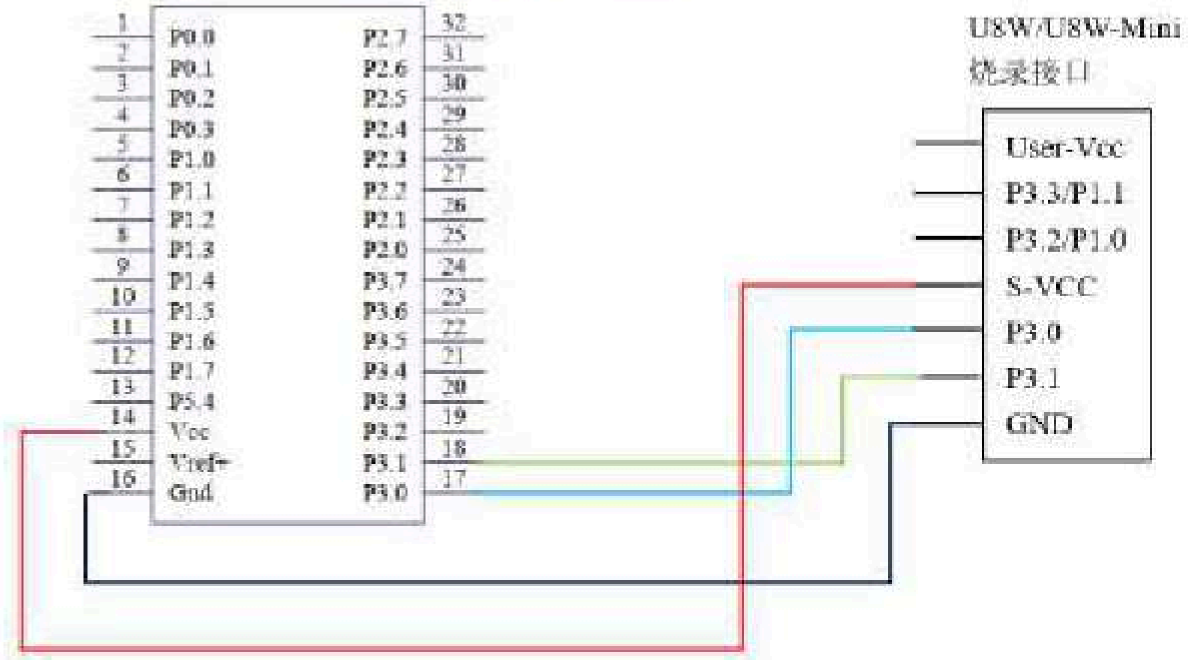
During the simulation debugging process, multiple operations such as reset, full-speed operation, single-step operation, and setting breakpo



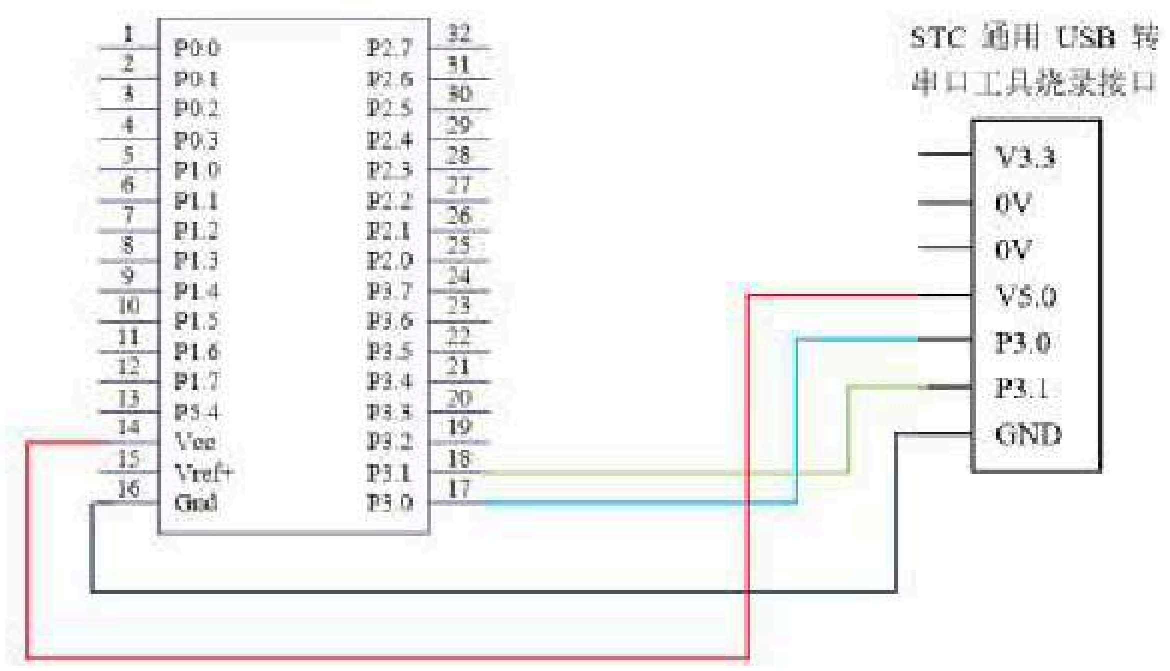
As shown in the figure above, you canSet multiple breakpoints in the program, the maximum allowed before the number of breakpoints is set (in theory, any one can be set, but setting too many breakpoints will affect the speed of debugging)

F. 5 Application circuit diagram

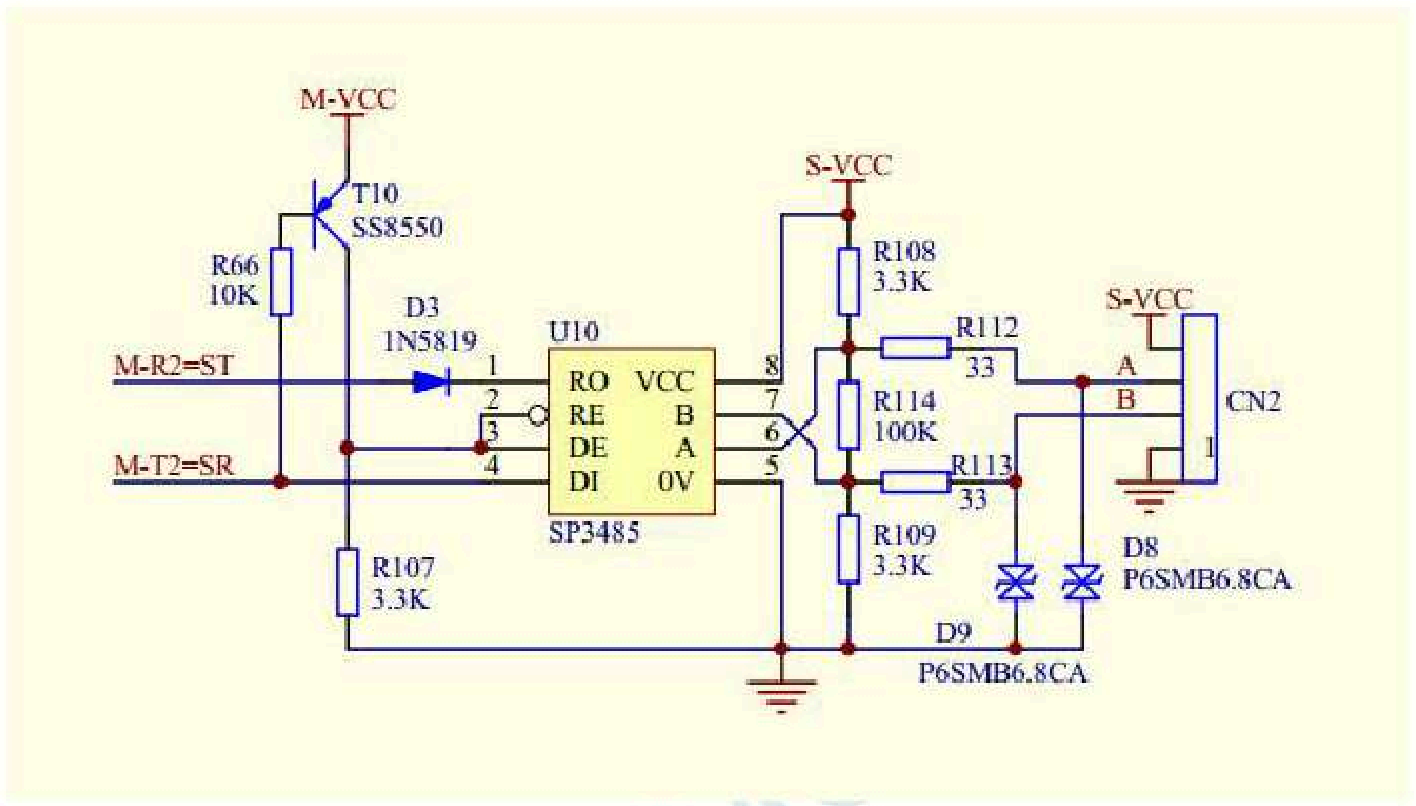
F. 5.1 U8W Tool application reference circuit diagram



F. 5.2 STC Universal USB Refer to the circuit diagram for the application of the serial port tool


1 P0.0
2 P0.1
3 P0.2
4 P0.3
5 P1.0
6 P1.1
7 P1.2
8 P1.3
9 P1.4
10 P1.5
11 P1.6
12 P1.7
13 P5.4
14 Vcc
15 Vref+
16 Gnd
32 P2.7
31 P2.6
30 P2.5
29 P2.4
28 P2.3
27 P2.2
26 P2.1
25 P2.0
24 P3.7
23 P3.6
22 P3.5
21 P3.4
20 P3.3
19 P3.2
18 P3.1
17 P3.0
STC 通用 USB 转
串口工具烧录接口
V3.3
0V
0V
V5.0
P3.0
P3.1
GND

G Appendix U8W Downloading tool RS485 Partial circuit diagram



BOM list :

Label	Model	package	Remarks
U10	SP3485EN	SOP8	RS485 chip
R66	10K	0603	Resistance
R107	3.3K	0603	Resistance
R108	3.3K	0603	Resistance
R109	3.3K	0603	Resistance
R112	33R	0603	Resistance
R113	33R	0603	Resistance
R114	100K	0603	resistance
T10	SS8550	SOT-23	PNP transistor
D3	1N5819	0603	Schottky diode
D8	P6SMB6.8CA	DO-214AA	TVS diode
D9	P6SMB6.8CA	DO-214AA SIP4	TVS diode
CN2			communication interface

Automatically start after receiving the user command w

move ISP (Download without power)

"User-defined download" and "user-defined encrypted download" are two completely different functions. Compared with the function of user-defined encrypted download, the function of user-defined download is simpler.

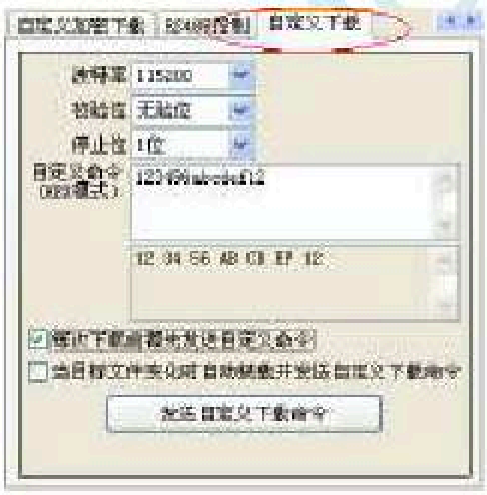
The specific functions are: the computer or offline download board ISP starts to download Before sending the real the programming handshake command, send A string of commands (regarding this string of serial port commands, users can set the baud rate, parity bit , and stop bit according to their serial port settings in the application) and then send it immediately to download the programming handshake command. ISP

The function of "user-defined download" is mainly in the early development stage of the project to achieve uninterruptible power-up (w re-powering the target chip) to download the user code. The specific implementation method is: the user needs to add a piece of code to dete custom command in his own program, and when it is detected, execute a sentence "language The assembly code Or MOV IAP_CONTR,#60H IAP_CONTR = 0x60; "of C code , MCU It will automatically reset to ISP the area to execute the code.

ISP

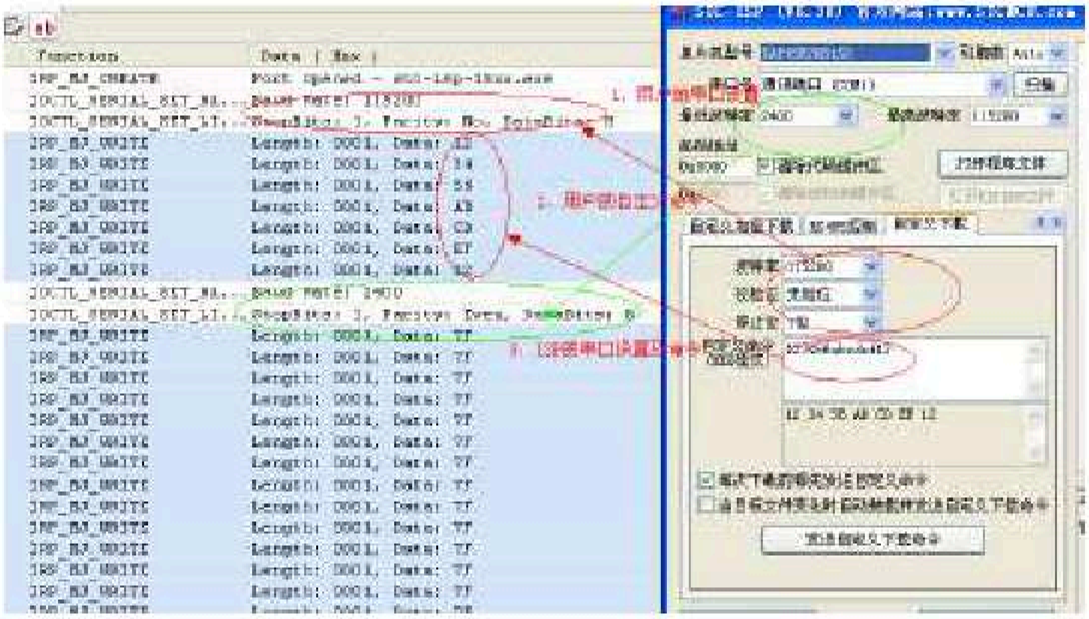
115200

As shown in the figure below, set the custom command to a sequence of commands with a baud rate of, no parity bit, and one stop bit : 0x12、 0x34、 0xAB、 0xCD、 0x56’ 0xEF、 0x12 。 When the option "Send a custom command before each download" is checked, it can be achieved Custom download function



波特率 115200
校验位 无校验
停止位 1位
自定义命令 (HEX格式) 123456abcdef12
12 34 56 AB CD EF 12
[✓] 每次下载前都先发送自定义命令
[ ] 当目标文件变化时自动装载并发送自定义下载命令

Click "Send Custom Download command" or click "Download" in the lower left corner of the interface / With the "Program" button, the application will send the serial port data as shown below



```
Function                        Data | Hex |
IRP_MJ_CREATE                   Port opened - stc-isp-15xx.exe
IOCTL_SERIAL_SET_BA...          Baud Rate: 115200
IOCTL_SERIAL_SET_LI...          StopBits: 1, Parity: No, DataBits: 8
IRP_MJ_WRITE                    Length: 0001, Data: 12
IRP_MJ_WRITE                    Length: 0001, Data: 34
IRP_MJ_WRITE                    Length: 0001, Data: 56
IRP_MJ_WRITE                    Length: 0001, Data: AB
IRP_MJ_WRITE                    Length: 0001, Data: CD
IRP_MJ_WRITE                    Length: 0001, Data: EF
IRP_MJ_WRITE                    Length: 0001, Data: 12
IOCTL_SERIAL_SET_BA...          Baud Rate: 2400
IOCTL_SERIAL_SET_LI...          StopBits: 1, Parity: Even, DataBits: 8
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
IRP_MJ_WRITE                    Length: 0001, Data: 7F
```

12 34 56 AB CD EF 12

I Appendix Use STC of IAP Series of microcontrollers to develop their ow ISP the program

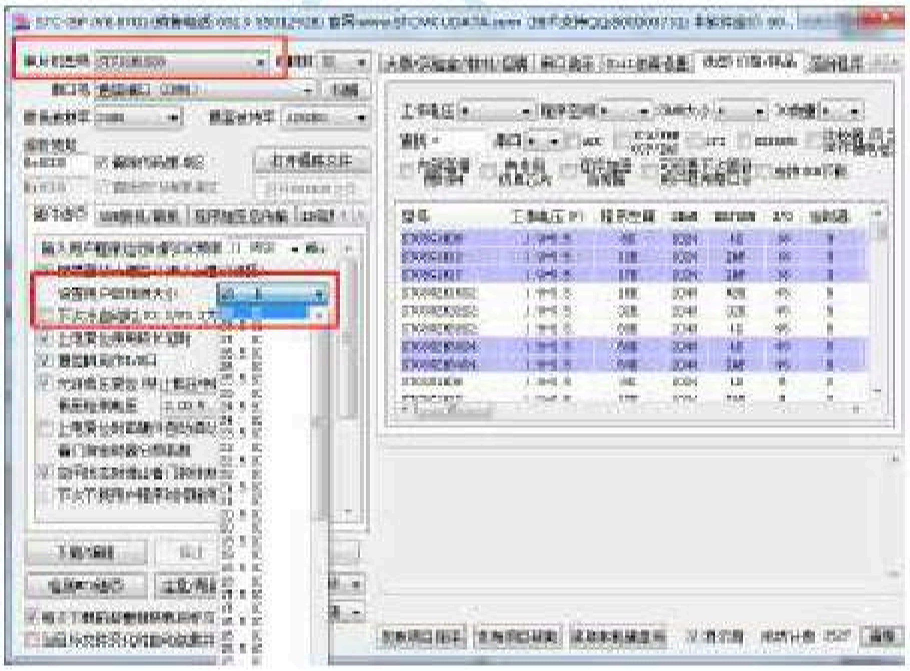
With IAP (In-Application-Programming) The continuous development of technology in the field of microcontrollers has upgrade It has brought great convenience. STC serial port ISP (In-System-Programming) The program is to use IAP Function to the user's program For online upgrades, but for the sake of the security of user code, neither the underlying code nor the upper application are open source. For Out of a series of microcontrollers, that is, the entire space, users can rewrite it in their own programs, which makes it useful. IAP MCU Flash The idea that users need to develop their own programs is realized.

ISP Series list

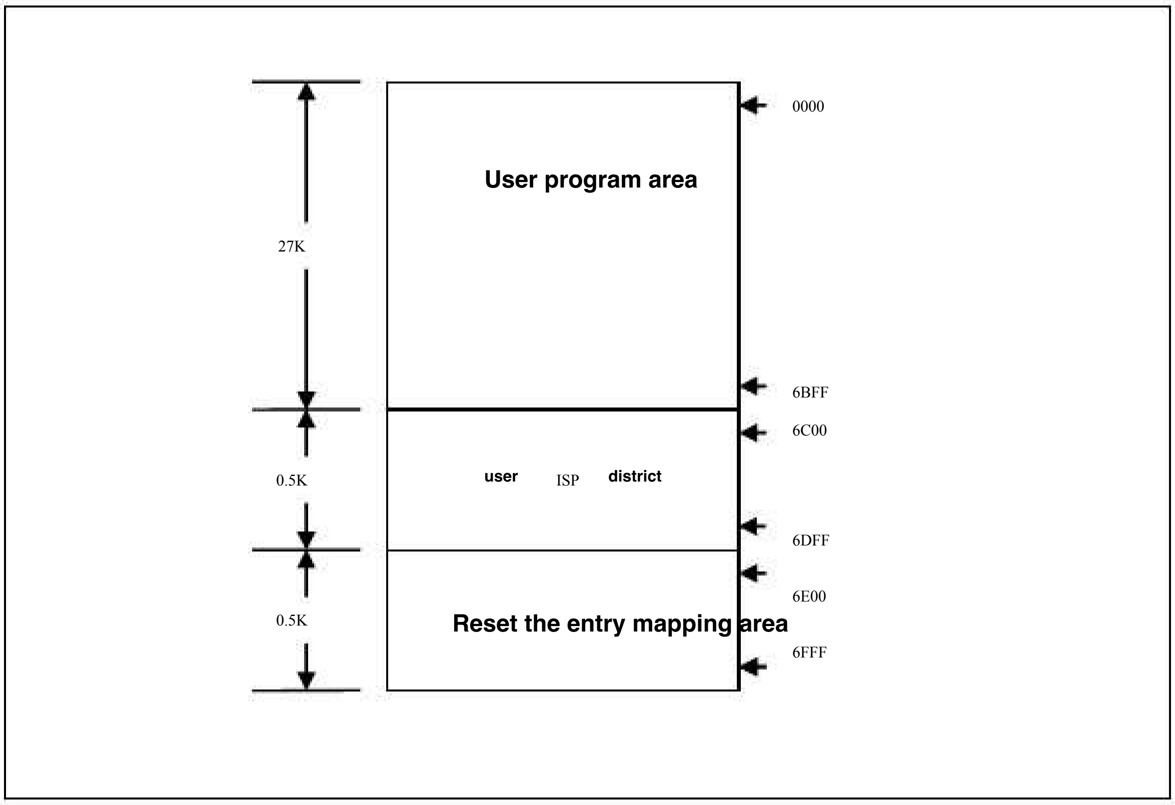
All models in the series of microcontrollers that can be customized by the user at the time of download are This article is based on Film machine. currently The series has the following models of microcontrollers for STC12H STC12H1K33 STC12H1K28 Take an example and explain in detail the use of STC IAP Single chip microcomputer development user's own ISP The method of the program is given based on Keil Compilation of the environment and C Source code

Step 1: Internal FLASH planning

Because I want Model MCU series STC12H IAP The user set it up by himself when downloading, so if the user needs t to realize ISP , Then download the user's own ISP When programming, you need to set all of them to the way shown in the figure below. 28K my own EEPROM , let the user program space and EEPROM The spaces are completely coincident, so that users can modify and modify their own progr update.



The following assumes that the user has divided the entire 28K The program space has been all set to EEPROM Now the entire 28K The program space is made as follo :

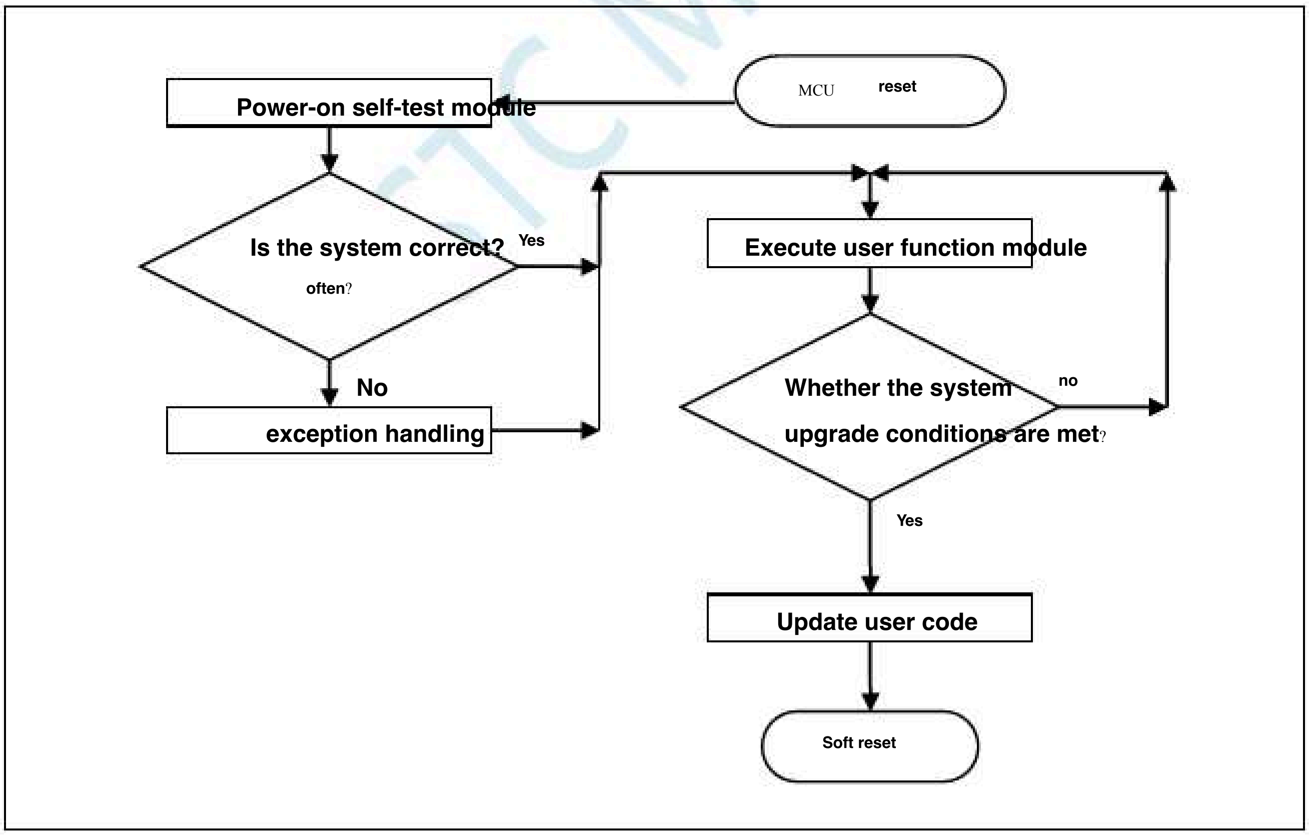


In space, from the address 0000H Beginning of the continuous 27K The byte space is the user program area. When specific download conditions

When the user is required to jump to the user ISP Program area, at this time, the user program area can be erased and rewritten to achieve the purp

The purpose of the sequence.

The second step, the basic framework of the program


MCU reset
Power-on self-test module
Is the system correct? often?
Yes
No
exception handling
Execute user function module
Whether the system upgrade conditions are met?
no
Yes
Update user code
Soft reset

The third step, the firmware program description of the next machine

The firmware program of the next machine consists of two parts : AP Code (User code) and ISP Code (assembly code) ISP

The test operating frequency is 11.0592MHz

```
UARTBAUD        EQU         0FFE8H              ;Define the baud rate of the serial (65536-11059200/4/115200)
                                                port, additional function control register
AUXR            DATA        08EH                ;
WDT_CONTR DATA              0C1H                ;Watchdog control register
IAP_DATA DATA               0C2H                ;IAP    Data register High
IAP_ADDRH DATA              0C3H                ;IAP    address register Low
IAP_ADDRL DATA              0C4H                ;IAP    Address Register
IAP_CMD DATA                0C5H                ;IAP    Command register Command
IAP_TRIG DATA               0C6H                ;IAP    trigger register
IAP_CONTR DATA              0C7H                ;IAP    Control register waiting
IAP_TPS DATA                0F5H                ;IAP    time Control register

ISPCODE         EQU         06C00H              ;ISP                     page), It is also the address of the external interface
APENTRY         EQU         06E00H              ;      Module entry address(1
                                                 Application entry address data(1 )page
                ORG         0000H

                LJMP        ISP_ENTRY           ;System reset entry

RESET:
                MOV         SCON,#50H           ;Set the serial port mode Bit data bit, (8 No parity bit)
                MOV         AUXR,#40H           ;The timer is pattern
                MOV         TMOD,#00H                      1; Timer
                MOV         TH1,#HIGH UARTBAUD  ;Set overload value Working in mode ) Bit reloading 0(16
                MOV         TL1,#LOW UARTBAUD
                SETB        TR1                 ;Start the timer
                                                            1
NEXT1:
                MOV         R0,#16
NEXT2:
                JNB         RI,$                ;Waiting for serial port data
                CLR         RI
                MOV         A,SBUF
                CJNE        A,#7FH,NEXT1                    7F;Determine whether it is
                DJNZ        R0,NEXT2
                LJMP        ISP_DOWNLOAD        ;Jump to the download interface

                ORG         ISPCODE

ISP_DOWNLOAD:
                CLR         A
                MOV         PSW,A                      The module uses the first set
                MOV         IE,A                of registers ;;ISP Turn off all interrupts
                CLR         RI                  ;Clear the serial port reception flag
                SETB        TI                  ;Set the serial port to send the flag
                CLR         TR0
                MOV         SP,#5FH             ;Set the stack pointer

                MOV         A,#5AH              ;return   5A55   to   PC, represents ISP   The erase module is ready
                LCALL       ISP_SENDUART
                MOV         A,#055H
                LCALL       ISP_SENDUART
                LCALL       ISP_RECVACK         ;Receive response data

                MOV         IAP_ADDRL,#0        ; First write at the starting address of the first page 2 instruction
                MOV         IAP_ADDRH,#02H
                LCALL       ISP_ERASEIAP
                MOV         A,#02H
                LCALL       ISP_PROGRAMIAP
                                                ; Programming user code reset vector code
                MOV     A,#HIGH                 ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ; Programming user code reset vector code
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ; Programming user code reset vector code

                MOV     IAP_ADDRL,#0            ; User code address from the beginning 0
                MOV     IAP_ADDRH,#0
                LCALL   ISP_ERASEIAP
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP          Programming user code reset
                MOV     A,#HIGH                 vector code ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ; Programming user code reset vector code
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ; Programming user code reset vector code

                MOV     IAP_ADDRL,#0            ;New code buffer address
                MOV     IAP_ADDRH,#02H
                MOV     R7,#124                 ;erase 62.5K byte
ISP_ERASEAP:
                LCALL   ISP_ERASEIAP
                INC     IAP_ADDRH               ;Destination address +512
                INC     IAP_ADDRH
                DJNZ    R7,ISP_ERASEAP          ;Determine whether the erasure is complete

                MOV     IAP_ADDRL,#LOW APENTRY
                MOV     IAP_ADDRH,#HIGH APENTRY
                LCALL   ISP_ERASEIAP

                MOV     A,#5AH                  ;return 5AA5 to PC, represents ISP The programming module is ready
                LCALL   ISP_SENDUART
                MOV     A,#0A5H
                LCALL   ISP_SENDUART
                LCALL   ISP_RECVACK             ;Receive response data

                LCALL   ISP_RECVUART            ;Receive length high bytes
                MOV     R0,A                    ;Receive length low bytes
                LCALL   ISP_RECVUART
                MOV     R1,A                    ;The total length of the -3
                CLR     C
                MOV     A,#03H
                SUBB    A,R1
                MOV     DPL,A
                CLR     A
                SUBB    A,R0
                MOV     DPH,A                   ;Total length complement length

                LCALL   ISP_RECVUART            Map the user code to reset the
                LCALL   ISP_PROGRAMIAP          entry code to the mapping area ;0000
                LCALL   ISP_RECVUART
                LCALL   ISP_PROGRAMIAP          ;0001
                LCALL   ISP_RECVUART
                LCALL   ISP_PROGRAMIAP          ;0002

                MOV     IAP_ADDRL,#03H          ;User code starting address
                MOV     IAP_ADDRH,#00H
ISP_PROGRAMNEXT:
                LCALL   ISP_RECVUART            ;Receive code data
                LCALL   ISP_PROGRAMIAP          ;Program to the user code area
                INC     DPTR
                MOV         A,DPL
                ORL         A,DPH
                JNZ         ISP_PROGRAMNEXT             ;Length detection

ISP_SOFTRESET:
                MOV         IAP_CONTR,#20H              ;Software reset system
                SJMP        $

ISP_ENTRY:
                MOV         WDT_CONTR,#17H              ;Clear watchdog
                MOV         IAP_CONTR,#80H              ;Enable IAP function
                MOV         IAP_TPS,#11                 ;Set up IAP Waiting time parameters
                MOV         IAP_ADDRL,#LOW ISP_DOWNLOAD
                MOV         IAP_ADDRH,#HIGH ISP_DOWNLOAD
                MOV         IAP_DATA,#00H ;Test data 1
                MOV         IAP_CMD,#1 ;Read command
                MOV         IAP_TRIG,#5AH ;trigger ISP command
                MOV         IAP_TRIG,#0A5H
                MOV         A,IAP_DATA
                CJNE        A,#0E4H,ISP_ENTRY           ; If the data cannot be read out, you need to wait for the voltage to stabilize
                INC         IAP_ADDRL                   ;Test address FC01H
                MOV         IAP_DATA,#45H               ;Test data 2
                MOV         IAP_CMD,#1                  ;Read command
                MOV         IAP_TRIG,#5AH               ;trigger ISP command
                MOV         IAP_TRIG,#0A5H
                MOV         A,IAP_DATA
                CJNE        A,#0F5H,ISP_ENTRY           ; If the data cannot be read out, you need to wait for the voltage to stabilize

                MOV         SCON,#50H                   ;Set the serial port mode (8 Bit data bit, No parity bit)
                MOV         AUXR,#40H                   ;The timer 1 is 1T pattern
                MOV         TMOD,#00H                   ;Timer 0 Working in mode 0(16 Bit reloading)
                MOV         TH1,#HIGH UARTBAUD          ;Set overload value
                MOV         TL1,#LOW UARTBAUD
                SETB        TR1                         ;Start the timer 1
                SETB        TR0

                LCALL       ISP_RECVUART                ; Detect whether there is serial data
                JC          GOTOAP
                MOV         R0,#16
ISP_CHECKNEXT:
                LCALL       ISP_RECVUART                ;Receive synchronization data
                JC          GOTOAP
                CJNE        A,#7FH,GOTOAP               ;Determine whether it is 7F
                DJNZ        R0,ISP_CHECKNEXT
                MOV         A,#5AH                      ;return 5A69 to PC, represents ISP The module is ready
                LCALL       ISP_SENDUART
                MOV         A,#69H
                LCALL       ISP_SENDUART
                LCALL       ISP_RECVACK                 ;Receive response data
                LJMP        ISP_DOWNLOAD                ;Jump to the download interface
GOTOAP:
                CLR         A                           ;will SFR Revert to reset value
                MOV         TCON,A
                MOV         TMOD,A
                MOV         TL0,A
                MOV         TH0,A
                MOV         TL1,A
                MOV         TH1,A
                MOV         SCON,A
                MOV         AUXR,A
                LJMP        APENTRY                 ;Normal operation of user programs

ISP_RECVACK:
                LCALL       ISP_RECVUART
                JC          GOTOAP
                XRL         A,#7FH
                JZ          ISP_RECVACK             ;Skip synchronizing data
                CJNE        A,#25H,GOTOAP           ;Response data detection
                LCALL       ISP_RECVUART
                JC          GOTOAP
                CJNE        A,#69H,GOTOAP           ;Response data detection
                RET

ISP_RECVUART:
                CLR         A
                MOV         TL0,A                   ;Initialize the timeout timer
                MOV         TH0,A
                CLR         TF0
                MOV         WDT_CONTR,#17H          ;Clear watchdog
ISP_RECVWAIT:                                       ;Timeout detection
                JBC         TF0,ISP_RECVTIMEOUT     ;Wait for the reception to
                JNB         RI,ISP_RECVWAIT         complete ;Read serial port
                MOV         A,SBUF                  data ;Clear flag
                CLR         RI
                CLR         C                       ;Receive serial data correctly
                RET
ISP_RECVTIMEOUT:
                SETB        C                       ;Timeout exit
                RET

ISP_SENDUART:
                MOV         WDT_CONTR,#17H          ;Clear watchdog
                JNB         TI,ISP_SENDUART         ; Wait for the previous data to be sent to complete
                CLR         TI                      ;Clear flag
                MOV         SBUF,A                  ;Send current data
                RET

ISP_ERASEIAP:
                MOV         WDT_CONTR,#17H          ;Clear watchdog
                MOV         IAP_CMD,#3              ;Erase command
                MOV         IAP_TRIG,#5AH           ;trigger ISP command
                MOV         IAP_TRIG,#0A5H
                NOP
                NOP
                NOP
                NOP
                RET

ISP_PROGRAMIAP:
                MOV         WDT_CONTR,#17H          ;Clear watchdog
                MOV         IAP_CMD,#2              ;Programming
                MOV         IAP_DATA,A              command ;Send current IAP Data register
                MOV         IAP_TRIG,#5AH           data ;trigger ISP command
                MOV         IAP_TRIG,#0A5H
                NOP
                NOP
                NOP
```

```
            NOP
            MOV             A,IAP_ADDRL                         ;IAP    address+1
            ADD             A,#01H
            MOV             IAP_ADDRL,A
            MOV             A,IAP_ADDRH
            ADDC            A,#00H
            MOV             IAP_ADDRH,A
            RET


            ORG             APENTRY
            LJMP            RESET


            END
```

The code includes the following external interface modules : ISP

ISP_DOWNLOAD : Program download entry address, absolute address: 6C00H Power-on

ISP_ENTRY system self-check program (automatically called by the system)

For user programs, users only need to update the code to the PC The value jumps to ISPPROGRAM (ie. 6C00H of absolute address when the download conditions are met).

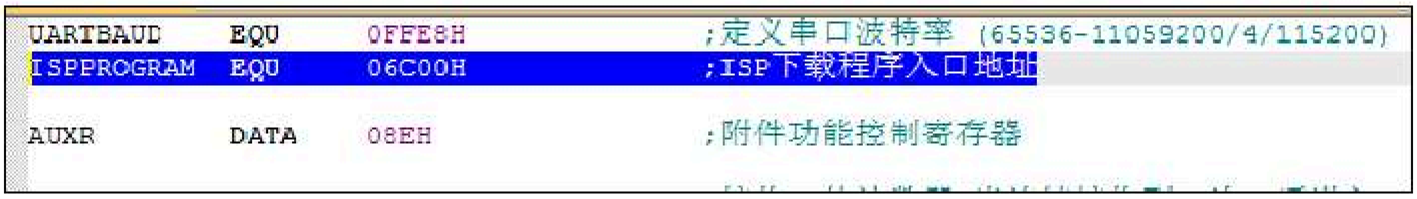
User code (C Language code)

```
// The test operating frequency is 11.0592MHz

#include "reg51. h"

#define         FOSC            11059200L               // System clock frequency
#define         BAUD            (65536 - FOSC/4/115200) // Define the baud rate of the serial port
#define         ISPPROGRAM      0x6c00                  //ISP    Download program entry address

sfr             AUXR            =       0x8e;           // Baud rate generator control register
sfr             P1M0            =       0x92;
sfr             P1M1            =       0x91;

void (*IspProgram)() = ISPPROGRAM;                      // Define pointer function
char cnt7f;                                             //Isp_Check    Variables used internally

void uart() interrupt 4                                 // Serial port interrupt service program
{
        if (TI) TI = 0;                                 // Send completion interrupt
        if (RI)                                         // Receive completion interrupt
        {
                if (SBUF == 0x7f)
                {
                        cnt7f++;
                        if (cnt7f >= 16)
                        {
                                IspProgram();           //              Important statement****)Call the download module(****
                        }
                }
                else
                {
                        cnt7f = 0;
                }
                RI = 0;                                         //Clear receipt completion mark
        }
}
void main()
{
        SCON = 0x50;                                            //Define serial port mode 1, Variable no parity bit
        AUXR = 0x40;
        TH1 = BAUD >> 8;
        TL1 = BAUD;
        TR1 = 1;
        ES = 1;                                                 //Enable serial port interrupt
        EA = 1;                                                 //Turn on the global interrupt switch

        P1M0 = 0;
        P1M1 = 0;
        while (1)
        {
                P1++;
        }
}
```

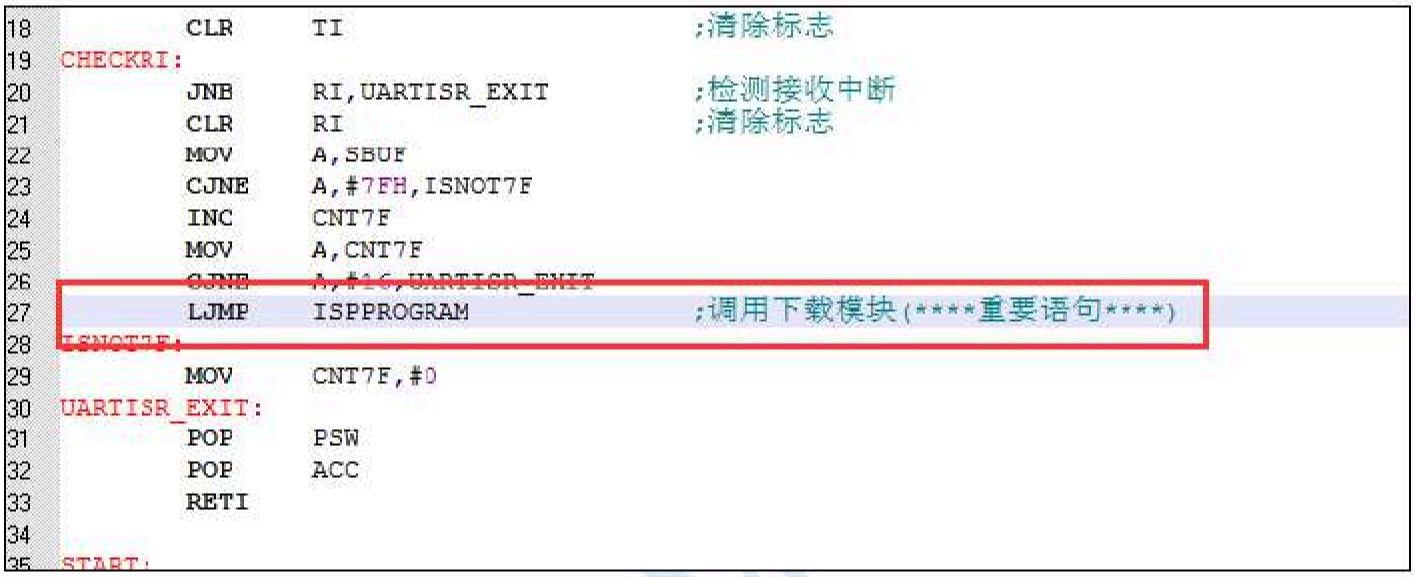
User code (assembly code)

```
;The test operating frequency is 11.0592MHz

UARTBAUD        EQU             0FFE8H                  ;Define the baud rate of the serial port
ISPPROGRAM EQU                  06C00H                  ;ISP Download program entry address

AUXR            DATA            08EH                    ; Accessory function control register

CNT7F           DATA            60H                     ;receive 7F The counter

                ORG             0000H
                LJMP            START                   ;System reset entry

                ORG             0023H
                LJMP            UART_ISR                ;Serial port interrupt entry

UART_ISR:
                PUSH            ACC
                PUSH            PSW
                JNB             TI,CHECKRI              ;Detect transmission interruption
                CLR             TI                      ;Clear flag
CHECKRI:
                JNB             RI,UARTISR_EXIT         ;Detect reception interrupt
                CLR             RI                      ;Clear flag
                MOV             A,SBUF
                CJNE            A,#7FH,ISNOT7F
                INC             CNT7F
                MOV             A,CNT7F
                CJNE            A,#16,UARTISR_EXIT
                LJMP            ISPPROGRAM              ;Call the download module (****Important statement****)
ISNOT7F:
                MOV             CNT7F,#0
UARTISR_EXIT:
                POP             PSW
                POP             ACC
                RETI

START:
                MOV             R0,#7FH                 ;clear RAM
                CLR             A
                MOV             @R0,A
                DJNZ            R0,$-1
                MOV             SP,#7FH                 ;initialize SP

                MOV             SCON,#50H               ;Set the serial port mode (8 Bit variable without parity bit )
                MOV             AUXR,#15H               ;BRT Work in 1T Mode start BRT
                MOV             TMOD,#00H               ;Timer 1 Working in mode 0(16 Bit reloading )
                MOV             TH1,#HIGH UARTBAUD      ;Set overload value
                MOV             TL1,#LOW UARTBAUD
                SETB            TR1                     ;Start the timer 1
                SETB            ES                      ;Enable serial port interrupt
                SETB            EA                      ;Interrupt the main switch

MAIN:
                INC             P0
                SJMP            MAIN

                END
```

User code can be used C Or written in assembly language, but you need to pay attention to one thing about assembly code: it is located Reset 0000H The instruction at the address must be a long jump statement (LJMP START). In the user code, you need to set up the serial port and satisfy when the download condition is similar, the value is jumped to PC (06C00H) Absolute address, in order to achieve code updates. For assembly generated ISPPROGRAM Code, we can use" LJMP 06C00H "The command is called, as shown in the figure below

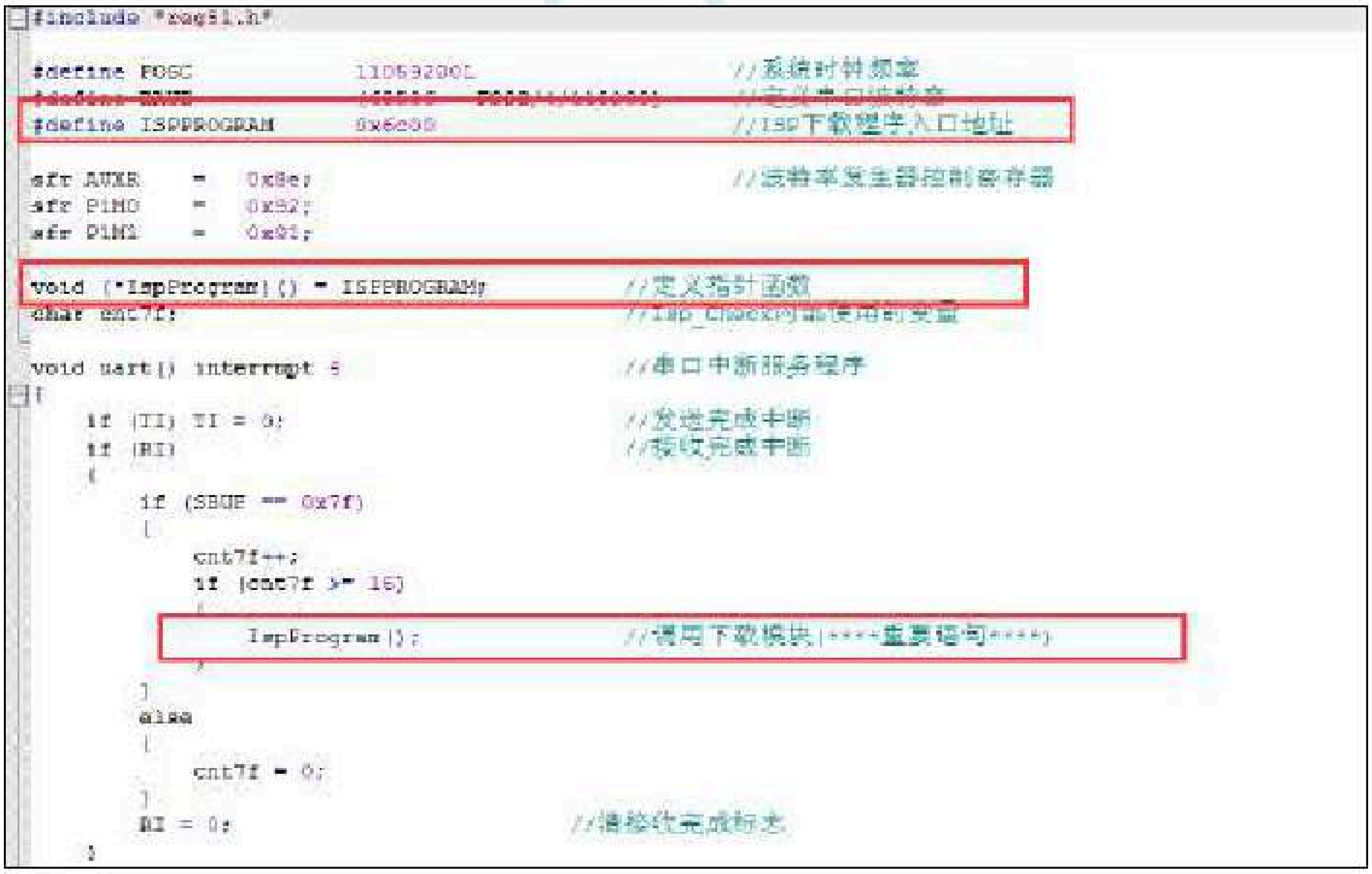


```
UARTBAUD    EQU     0FFE8H                  ;定义串口波特率 (65536-11059200/4/115200)
ISPPROGRAM  EQU     06C00H                  ;ISP下载程序入口地址

AUXR        DATA    08EH                    ;附件功能控制寄存器
```



```
            CLR     TI                      ;清除标志
CHECKRI:
            JNB     RI,UARTISR_EXIT         ;检测接收中断
            CLR     RI                      ;清除标志
            MOV     A,SBUF
            CJNE    A,#7FH,ISNOT7F
            INC     CNT7F
            MOV     A,CNT7F
            CJNE    A,#16,UARTISR_EXIT
            LJMP    ISPPROGRAM              ;调用下载模块(****重要语句****)
ISNOT7F:
            MOV     CNT7F,#0
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

START:
```

in C In the code, you must define a function pointer variable and assign this variable, And then call again, as shown in the figure below



```
#include "reg51.h"

#define FOSC        11059200L                       //系统时钟频率
#define BAUD        (65536 - FOSC/4/115200)         //定义串口波特率
#define ISPPROGRAM  0x6c00                          //ISP下载程序入口地址

sfr AUXR    =   0x8e;                               //波特率发生器控制寄存器
sfr P1M0    =   0x92;
sfr P1M1    =   0x91;

void (*IspProgram)() = ISPPROGRAM;                  //定义指针函数
char cnt7f;                                         //Isp_Check内部使用的变量

void uart() interrupt 4                             //串口中断服务程序
{
    if (TI) TI = 0;                                 //发送完成中断
    if (RI)                                         //接收完成中断
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();                       //调用下载模块(****重要语句****)
            }
        }
        else
        {
            cnt7f = 0;
        }
        RI = 0;                                     //清除接收完成标志
    }
```

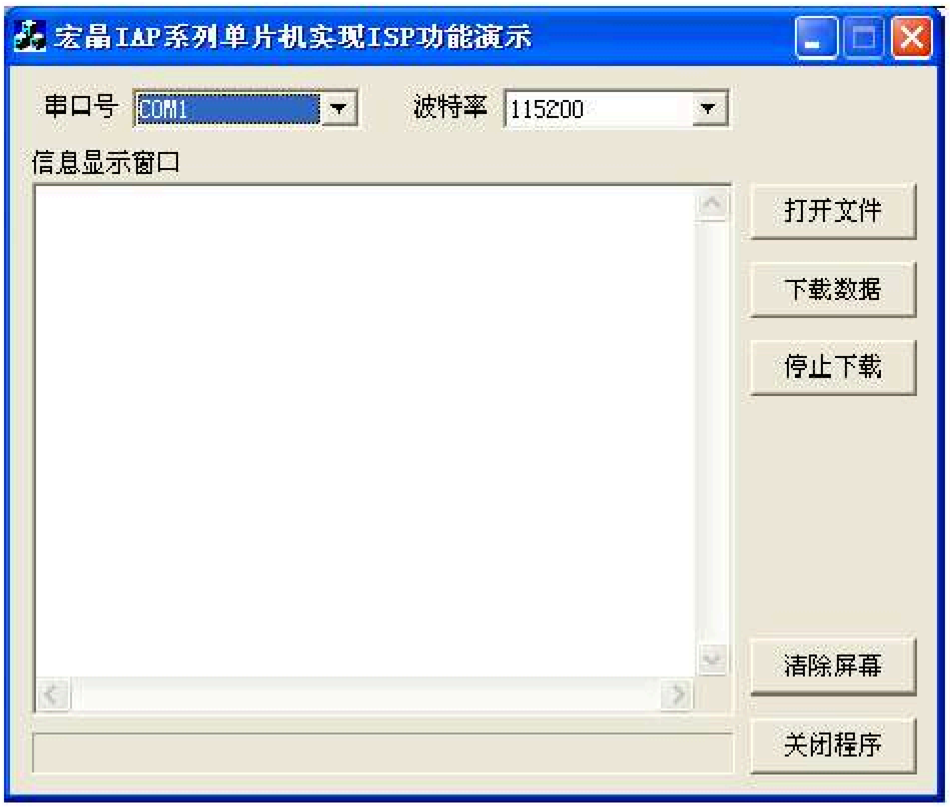
Step 4. Description of the application program of the host computer

The program of the host computer is based on MFC This dialog box item, the access to the serial port is directly called Windows of API Function, but not There are many problems with the use of serial port controls, which eliminates the registration of controls and the incompatible system versi The interface is relatively simple, it just provides a framework for the realization of this function, and other functions and requirements can be

The core module of the host computer program is based on CISPDlg A friend function of" UINT Download(LPVOID pParam);", This function is responsible for communicating with the next machine and sending various communication commands to complete the update of the user program. Users can add commands according to their different needs.

Step 5. How to use the host computer application

Open the upper computer interface, as shown in the figure below



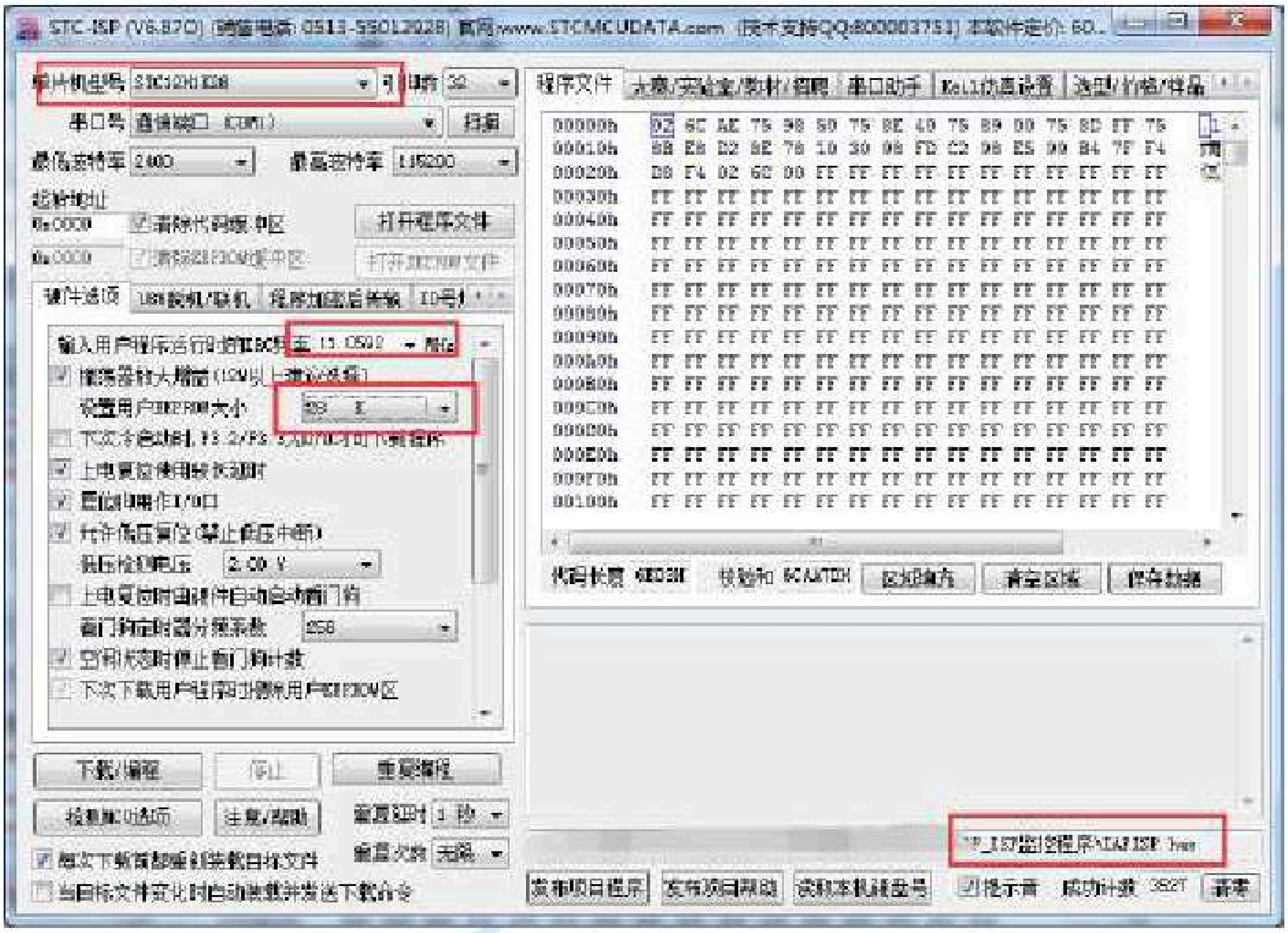
Select the serial port number, set the same serial port baud rate as the next machine , and open the source data file to be downloaded. ,

Bin Or the format can be Intel

hex Click the "Download Data" button to start downloading data

Step 6. How to use the firmware program of the next machine

There are two " download tools" for "IAPISP. hex" and " AP. hex" , For a new single-chip microcomputer, you must use it for the first time the target file of the next machine in the "Download tool" "Write to the chip, as shown in the figure below. If you update it later, you no longer IAPISP. hex "This article tool" in the attachment " AP. hex" is just a template for a user program. When the download conditions are met, PC The value jumps to FA00H the user only needs to send the address to achieve the code update.



STC-ISP (V6.87Q) (销售电话: 0513-55012028) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60..

单片机型号 STC12H1K28　引脚数 32
串口号 通信端口 (COM1)　扫描
最低波特率 2400　最高波特率 115200
起始地址
0x0000　☑ 清除代码缓冲区　打开程序文件
0x0000　☐ 清除EEPROM缓冲区　打开EEPROM文件

硬件选项 | USB脱机/联机 | 程序加密后传输 | ID号

输入用户程序运行时的IRC频率 11.0592 MHz
☑ 振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 [illegible]
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序
☑ 上电复位使用较长延时
☑ 复位脚用作I/O口
☑ 允许低压复位(禁止低压中断)
低压检测电压 2.00 V
☐ 上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
☑ 空闲状态时停止看门狗计数
☑ 下次下载用户程序时擦除用户EEPROM区

下载/编程　停止　重复编程
检测MCU选项　注意/帮助　重复延时 1 秒　重复次数 无限
☑ 每次下载前都重新装载目标文件
☐ 当目标文件变化时自动装载并发送下载命令

程序文件 | 大赛/实验室/教材/编程 | 串口助手 | Keil仿真设置 | 选型/价格/样品

```
00000h  02 6C AE 75 98 50 75 8E 40 75 89 00 75 8D FF 75
00010h  8B E8 D2 8E 78 10 30 98 FD C2 98 E5 99 B4 7F F4
00020h  D8 F4 02 6C 00 FF FF FF FF FF FF FF FF FF FF FF
00030h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00040h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00050h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00060h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00070h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00080h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00090h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000A0h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000B0h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000C0h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000D0h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000E0h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000F0h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00100h  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

代码长度 [illegible]　校验和 [illegible]　区域填充　清空区域　保存数据

...\IAP_ISP.hex

发布项目程序　发布项目帮助　读取本机硬盘号　☑ 提示音　成功计数 3827　清零

The user program is reset to the system area for processing ISP downloaded Method(Non-stop power)

When the project is in the development stage, it is necessary to repeatedly download the user code to the target chip for code verification. For normal downloads, the target chip needs to be re-powered up, which will make the project more cumbersome during the development phase. STC The single-chip microcomputer has added a special function register, when the user writes to this register IAP_CONTR , You can reset the software to the system System area, and then realize that it can be downloaded without power-up. ISP

However, there are two questions about how users can determine whether ISP download? When to register IAP_CONTR write 0x60 Trigger a soft reset? That's it they are in progress. The following are four judgment methods. :

Use the P3.0 port to detect the serial port start signal

Serial port of single chip microcomputer Fixed use P3.0 and STC ISP Two ports, when P3.1 When the download software starts downloading, a handshake The command is sent to the port of the MICROCONTROLLER. P3.0 and P3.1 Is specifically for ISP Download, you can use the port to detect the start of the serial Start the signal to judge the download. ISP

C Language code

```
//The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
sfr        IAP_CONTR        =    0xc7;
sfr P3M0                    =    0xb2;
sfr P3M1                    =    0xb1;
sbit P30                    =    P3^0;
void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
    P30 = 1;
    while (1)
    {
        if (! P30) IAP_CONTR = 0x60;        //P3.0 The low level is the starting signal of the serial port
                                            //The software is reset to the system area

        ...                                 //User code

    }
}
```

Use the falling edge interrupt of the P3.0/INT4 port to detect the serial port start signal

Method of A film method A Similar, the difference lies in A Using the query method, method B Use interrupt mode. because STC single machine P3.0 The mouth is INT4 interrupt port of the method.

C Language code

```
// The test operating frequency is 11.0592MHz

#include "reg51. h"
#include "intrins. h"
sfr         IAP_CONTR        =    0xc7;
sfr INTCLKO                  =    0x8f;
sfr P3M0                     =    0xb2;
sfr P3M1                     =    0xb1;

void Int4Isr() interrupt 16                        //INT4 Interrupt service procedure
{
        IAP_CONTR = 0x60;                          //Serial port start signal trigger INT4 interrupt
                                                   //The software is reset to the system area
}

void main()
{
        P3M0 = 0x00;
        P3M1 = 0x00;
        INTCLKO |= 0x40;                           //Enable INT4 interrupt
        EA = 1;
        while (1)
        {
                ...                                //User code
        }
}
```

Use the serial port of the P3.0/RxD port to receive and detect the 7F sent by the ISP download softwa

Method And Are very simple, but easily disturbed, if B P3.0 If there is any interference signal at the port, it will trigger the softw bit, so method method It is to verify the serial port data.

STC C ISP When downloading, the lowest baud rate will be used first (usually by downloading software). ISP Stop sending handshake commands continuously of A Therefore, the user can set the serial port to bit data bit in the program +2400) Even check 2400 detection 7F, For example, continuously detected 8 7F indicates that it can be determined that it needs to be carried out Download, and then trigger the software reset at this time. Baud rate, and then continue

C Language code

```
// The test operating frequency is 11.0592MHz

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

```
#define         FOSC                    11059200UL
#define         BR2400                  (65536 - FOSC / 4 / 2400)

sfr             IAP_CONTR               =       0xc7;
sfr             AUXR                    =       0x8e;
sfr             P3M0                    =       0xb2;
sfr             P3M1                    =       0xb1;

char cnt7f;

void UartIsr() interrupt 4                                          //Serial port interrupt service program
{
        if (TI)
        {
                TI = 0;
        }
        if (RI)
        {
                RI = 0;
                if ((SBUF == 0x7f) && (RB8 == 1))                   //ISP The even parity bit of the handshake
                                                                    //7F command sent by the download software is 1
                {
                        if (++cnt7f == 8)                           //When a continuous detection
                                IAP_CONTR = 0x60;                   is made //Reset to the system area
                }
                else
                {
                        cnt7f = 0;
                }
        }
}
void main()
{
        P3M0 = 0x00;
        P3M1 = 0x00;
        SCON = 0xd0;                                                //Set the serial port to 9 Bit data bit
        TMOD = 0x00;
        AUXR = 0x40;

        TH1 = BR2400 >> 8;                                          //Set the baud rate of the serial port to
        TL1 = BR2400;
        TR1 = 1;
        ES = 1;
        EA = 1;

        cnt7f = 0;

        while (1)
        {
                ...                                                 //User code
        }
}
```

Use P3.0/RxD serial port to receive and detect user download commands sent by ISP download software

If you need to use a serial port to 3 The Chinese method may not be applicable, you STC can use it at this time

communicate with the interface provided by the software and customize a set of dedicated user download commands (you can specify the baud

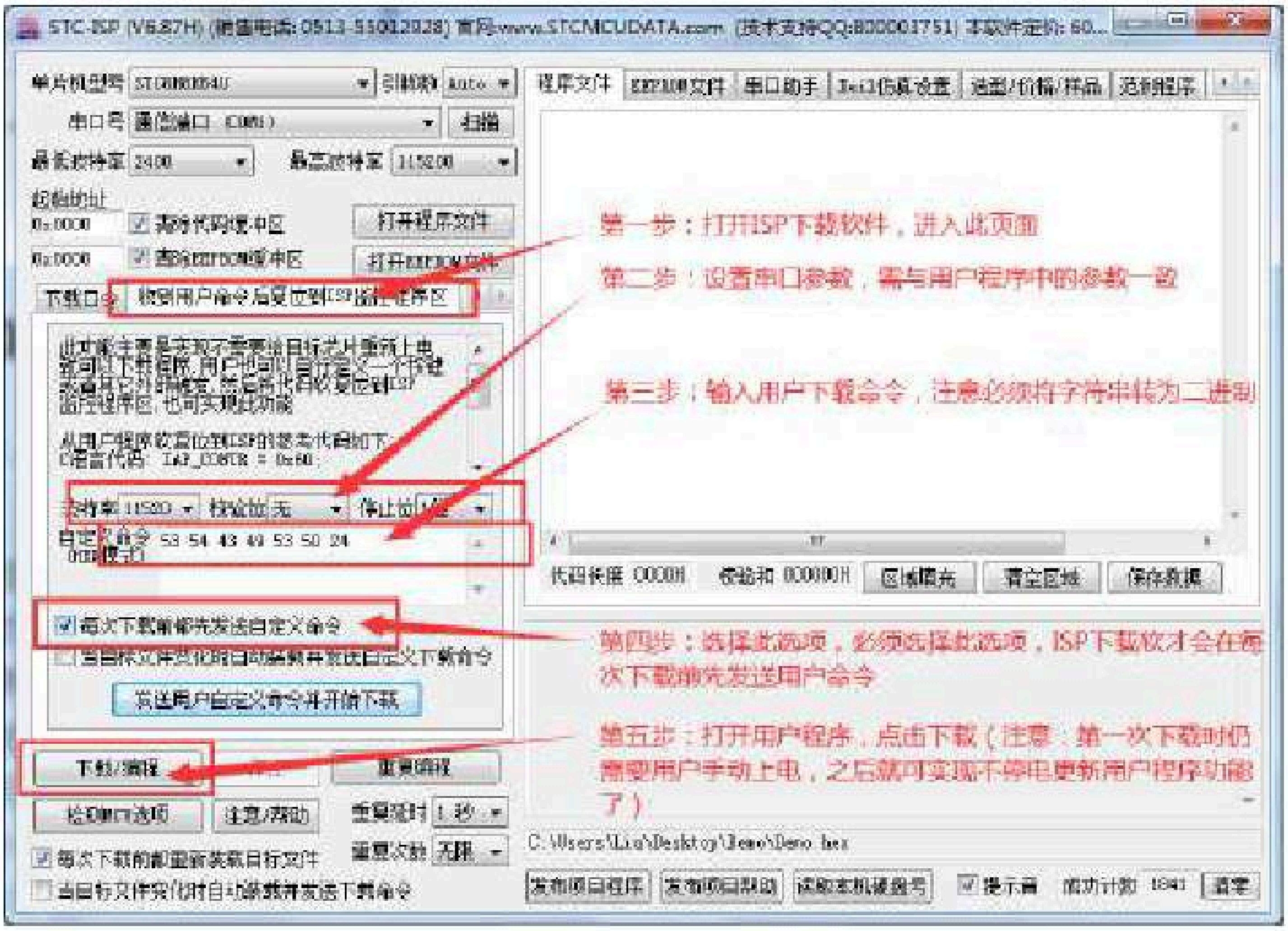
Function, ISP download the software in ISP progress Before downloading, the user download command will be sent using the baud rate, parity bit, and stop b

Then send a handshake command. Users only need to monitor the serial port command sequence in their own code.

When the correct user download command is detected, ISP the function software is reset to the system area to achieve non-stop power-up.

The following assumes that the user download command is a string "STCISP$" The serial port is set to baud rate 115200, No checksum 1 Bit stop bit.

ISP The settings in the downloaded software are as follows：



波特率 115200　校验位 无　停止位 1位

自定义命令 53 54 43 49 53 50 24

☑ 每次下载前都先发送自定义命令

第一步：打开ISP下载软件，进入此页面

第二步：设置串口参数，需与用户程序中的参数一致

第三步：输入用户下载命令，注意必须将字符串转为二进制

第四步：选择此选项，必须选择此选项，ISP下载软件才会在每次下载前先发送用户命令

第五步：打开用户程序，点击下载（注意，第一次下载时仍需要用户手动上电，之后就可实现不停电更新用户程序功能了）

The user sample code is as follows：

C Language code

```
// The test operating frequency is 11.0592MHz

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

#define     FOSC            11059200UL
#define BR115200            (65536 - FOSC / 4 / 115200)

sfr IAP_CONTR       =       0xc7;
sfr AUXR            =       0x8e;
sfr P3M0            =       0xb2;
sfr P3M1            =       0xb1;

char stage;
```

```
void UartIsr() interrupt 4                                    // Serial port interrupt service program
{
        char dat;

        if (TI)
        {
                TI = 0;
        }

        if (RI)
        {
                RI = 0;

dat = SBUF;
switch (stage)
{
case 0:
default:
L_Check1st:
                if (dat == 'S') stage = 1;
                else stage = 0;
                break;
                case 1:
                if (dat == 'T') stage = 2;
                else goto L_Check1st;
                break;
                case 2:
                if (dat == 'C') stage = 3;
                else goto L_Check1st;
                break;
                case 3:
                if (dat == 'I') stage = 4;
                else goto L_Check1st;
                break;
                case 4:
                if (dat == 'S') stage = 5;
                else goto L_Check1st;
                break;
                case 5:
                if (dat == 'P') stage = 6;
                else goto L_Check1st;
                break;
                case 6:
                        if (dat == '$')                       // When the correct user download command is detected
                                IAP_CONTR = 0x60;             // Reset to the system area
                        else goto L_Check1st;
break;
}
        }
}
void main()
{
        P3M0 = 0x00;
        P3M1 = 0x00;
        SCON = 0x50;                                          // Set the user serial port mode to bit data bit 8
        TMOD = 0x00;
        AUXR = 0x40;
        TH1 = BR2400 >> 8;                                  //Set the baud rate of the serial port to
        TL1 = BR2400;
        TR1 = 1;
        ES = 1;
        EA = 1;

        stage = 0;

        while (1)
        {
                ...                                         //User code
        }
}
```

K Appendix Use of third parties MCU correct STC12H Series of microcontrollers Proceed ISP Download sample program

C Language code

// attention:Use When downloading the series of microcontrollers, it must be executed After the code this code to power up the target chip STC12H. Otherwise, the target chip will not be downloaded correctly

```
#include "reg51. h"

typedef         bit                     BOOL;
typedef         unsigned char           BYTE;
typedef         unsigned short          WORD;

// Macro and constant definition
#define         FALSE                   0
#define         TRUE                    1
#define         LOBYTE(w)               ((BYTE)(WORD)(w))
#define         HIBYTE(w)               ((BYTE)((WORD)(w) >> 8))

#define         MINBAUD                 2400L
#define         MAXBAUD                 115200L

#define         FOSC                    11059200L               //Main control chip operating frequency
#define         BR(n)                   (65536 - FOSC/4/(n))    // Main control chip serial port baud rate calculation formula
#define         T1MS                    (65536 - FOSC/1000)     //Main control chip 1ms initial timing value

#define         FUSER                   24000000L               //STC12H Series target chip operating frequency
#define         RL(n)                   (65536 - FUSER/4/(n))   //STC12H Series target chip serial port baud rate calculation formula

sfr             AUXR = 0x8e;
sfr             P3M1 = 0xB1;
sfr             P3M0 = 0xB2;

//Variable definition
BOOL f1ms;                              //1ms Flag position
BOOL UartBusy;                          //Serial port sends busy flag
BOOL UartReceived;                      // Serial port data reception completion flag
BYTE UartRecvStep;                      //Serial port data reception control
BYTE TimeOut;                           // Serial port communication timeout counter
BYTE xdata TxBuffer[256];               // Serial port data transmission buffer
BYTE xdata RxBuffer[256];               // Serial port data receiving buffer
char code DEMO[256];                    //Demo code data

//Function declaration
void Initial(void); void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
BOOL Download(BYTE *pdat, long size);
```

```
// Main function entry
void main(void)
{
        P3M0 = 0x00;
        P3M1 = 0x00;

        Initial();
        if (Download(DEMO, 256))
        {
                /Download successfully/P3
                = 0xff; DelayXms(500);
                P3 = 0x00;
                DelayXms(500);
                P3 = 0xff;
                DelayXms(500);
                P3 = 0x00;
                DelayXms(500);
                P3 = 0xff;
                DelayXms(500);
                P3 = 0x00;
                DelayXms(500);
                P3 = 0xff;

        }
        else
        {
                /Download failed/P3
                = 0xff; DelayXms(500);
                P3 = 0xf3;
                DelayXms(500);
                P3 = 0xff;
                DelayXms(500);
                P3 = 0xf3;
                DelayXms(500);
                P3 = 0xff;
                DelayXms(500);
                P3 = 0xf3;
                DelayXms(500);
                P3 = 0xff;

        }

        while (1);
}

//1ms     Timer interrupt service program
void tm0(void) interrupt 1
{
        static BYTE Counter100;

        f1ms = TRUE;
        if (Counter100-- == 0)
        {
Counter100 = 100;
if (TimeOut) TimeOut--;
        }
}
```

// Serial port interrupt service program

```
void uart(void) interrupt 4
{
        static WORD RecvSum;
        static BYTE RecvIndex;
        static BYTE RecvCount;
        BYTE dat;
        if (TI)
        {
                TI = 0;
                UartBusy = FALSE;
        }

        if (RI)
        {
                RI = 0;
                dat         = SBUF;
                switch (UartRecvStep)
                {
                case 1:
                if (dat ! = 0xb9) goto L_CheckFirst;
                UartRecvStep++;
                break;
                case 2:
                if (dat ! = 0x68) goto L_CheckFirst;
                UartRecvStep++;
                break;
                case 3:
                if (dat ! = 0x00) goto L_CheckFirst;
                UartRecvStep++;
                break;
                case 4:
                RecvSum = 0x68 + dat;
                RecvCount = dat - 6;
                RecvIndex = 0;
                UartRecvStep++;
                break;
                case 5:
                RecvSum += dat;
                RxBuffer[RecvIndex++] = dat;
                if (RecvIndex == RecvCount) UartRecvStep++;
                break;
                case 6:
                if (dat ! = HIBYTE(RecvSum)) goto L_CheckFirst;
                UartRecvStep++;
                break;
                case 7:
                if (dat ! = LOBYTE(RecvSum)) goto L_CheckFirst;
                UartRecvStep++;
                break;
                case 8:
                if (dat ! = 0x16) goto L_CheckFirst;
                UartReceived = TRUE;
                UartRecvStep++;
                break;
L_CheckFirst:
                case 0:
                default:
```

```
                                        CommInit();
                                        UartRecvStep = (dat == 0x46 ? 1 : 0);
                                        break;
                                }
                        }
}
```

System initialization

```
//
void Initial(void)
{
        UartBusy = FALSE;

        SCON = 0xd0;                                    // Serial port data mode must be bit data9,1 8Coupling inspection
        AUXR = 0xc0;
        TMOD = 0x00;
        TH0 = HIBYTE(T1MS);
        TL0 = LOBYTE(T1MS);
        TR0 = 1;
        TH1 = HIBYTE(BR(MINBAUD));
        TL1 = LOBYTE(BR(MINBAUD));
        TR1 = 1;
        ET0 = 1;
        ES = 1;
        EA = 1;
}
```

Delay program//Xms

```
void DelayXms(WORD x)
{
        do
        {
        f1ms = FALSE;
        while (! f1ms);
        } while (x--);
}
```

// Serial port data transmission program

```
BYTE UartSend(BYTE dat)
{
        while (UartBusy);

        UartBusy = TRUE;
        ACC = dat;
        TB8 = P;
        SBUF = ACC;
        return dat;
}
```

// Serial communication initialization

```
void CommInit(void)
{
        UartRecvStep = 0;
        TimeOut = 20;
        UartReceived = FALSE;
}
```

// Send serial communication data packets

```
void CommSend(BYTE size)
```

```
{
        WORD sum;
        BYTE i;

        UartSend(0x46);
        UartSend(0xb9);
        UartSend(0x6a);
        UartSend(0x00);
        sum = size + 6 + 0x6a;
        UartSend(size + 6);
        for (i=0; i<size; i++)
        {
                sum += UartSend(TxBuffer[i]);
        }
        UartSend(HIBYTE(sum));
        UartSend(LOBYTE(sum));
        UartSend(0x16);
        while (UartBusy);

        CommInit();
}
```

Series of chips are carried out ISP Download program

```
ISP BOOL Download(BYTE *pdat, long size)
{
        BYTE arg;
        BYTE offset;
        BYTE cnt;
        WORD addr;

        //Handshake CommInit();
        while (1)
        {
                if (UartRecvStep == 0)
                {
                UartSend(0x7f);
                DelayXms(10);
                }
                if (UartReceived)
                {
                        arg = RxBuffer[4];
                        if (RxBuffer[0] == 0x50) break;
                        return FALSE;
                }
        }
        Setting parameters(Set the parameters such as the highest baud rate used from the chip and the waiting time)
        //
TxBuffer[0] = 0x01;
TxBuffer[1] = arg;
TxBuffer[2] = 0x40;
TxBuffer[3] = HIBYTE(RL(MAXBAUD));
TxBuffer[4] = LOBYTE(RL(MAXBAUD));
TxBuffer[5] = 0x00;
TxBuffer[6] = 0x00;
TxBuffer[7] = 0x97;
CommSend(8);
while (1)
{
```

```
                if (TimeOut == 0) return FALSE;
                if (UartReceived)
                {
        if (RxBuffer[0] == 0x01) break;
        return FALSE;
                }
        }
        /prepare/TH1 =
        HIBYTE(BR(MAXBAUD)); TL1 =
        LOBYTE(BR(MAXBAUD)); DelayXms(10);
        TxBuffer[0] = 0x05;
        TxBuffer[1] = 0x00;
        TxBuffer[2] = 0x00;
        TxBuffer[3] = 0x5a;
        TxBuffer[4] = 0xa5;
        CommSend(5);
        while (1)
        {
                if (TimeOut == 0) return FALSE;
                if (UartReceived)
                {
        if (RxBuffer[0] == 0x05) break;
        return FALSE;
                }
        }
        /erase/DelayXms(10);
        TxBuffer[0] = 0x03;
        TxBuffer[1] = 0x00;
        TxBuffer[2] = 0x00;
        TxBuffer[3] = 0x5a;
        TxBuffer[4] = 0xa5;
        CommSend(5);
        TimeOut = 100;
        while (1)
        {
                if (TimeOut == 0) return FALSE;
                if (UartReceived)
                {
        if (RxBuffer[0] == 0x03) break;
        return FALSE;
                }
        }
        Write user code
        //
        DelayXms(10);
        addr = 0;
        TxBuffer[0] = 0x22;
        TxBuffer[3] = 0x5a;
        TxBuffer[4] = 0xa5;
        offset = 5;
        while (addr < size)
        {
TxBuffer[1] = HIBYTE(addr);
TxBuffer[2] = LOBYTE(addr);
cnt                              = 0;
```

```
        while (addr < size)
        {
        TxBuffer[cnt+offset] = pdat[addr];
        addr++;
        cnt++;
        if (cnt >= 128) break;
        }
        CommSend(cnt + offset);
        while (1)
        {
                if (TimeOut == 0) return FALSE;
                if (UartReceived)
                {
        if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
        return FALSE;
                }
        }
        TxBuffer[0] = 0x02;
    }

    //// Write hardware options
    //// If you do not need to modify the hardware options, This step can be skipped directly, All hardware options at this time
    //// All remain unchanged, MCU The frequency is the last adjusted frequency
    //// If you write hardware options, MCU The interior of IRC The frequency will be fixed and written as 24M, Other options are restored to factory settings
    //// It is recommended to use for the first time STC-ISP Download the software to set up the hardware options from the chip
    //// Use the main chip again in the future, do not write hardware options when downloading programs from the chip
    //DelayXms(10);
    //for (cnt=0; cnt<128; cnt++)
    //{
    //        TxBuffer[cnt] = 0xff;
    //}
    //TxBuffer[0] = 0x04;
    //TxBuffer[1] = 0x00;
    //TxBuffer[2] = 0x00;
    //TxBuffer[3] = 0x5a;
    //TxBuffer[4] = 0xa5;
    //TxBuffer[33] = arg;
    //TxBuffer[34] = 0x00;
    //TxBuffer[35] = 0x01;
    //TxBuffer[41] = 0xbf;
    //TxBuffer[42] = 0xbd;                                  //P5.4 for IO mouth
    ////TxBuffer[42] = 0xad;                                //P5.4 For the reset pin
    //TxBuffer[43] = 0xf7;
    //TxBuffer[44] = 0xff;
    //CommSend(45);
    //while (1)
    //{
    //        if (TimeOut == 0) return FALSE;
    //        if (UartReceived)
    //        {
    //if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
    //return FALSE;
    //        }
    //}
    // download complete
    return TRUE;
}
```

```
char code DEMO[256] =
{
        0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
        0xD9,0xFC,0x22,
};
```

Remarks: If users need to set different operating frequencies, please refer to 7.3.7 and 7.3.8 Sample code of the chapter

Use a third-party application to call STC L Appendix Release project program to download the MICROCONTROLLER ISP

Used STC ISP The release project program generated by the downloaded software is executable EXE Format file, users can directly double-click the published c
A method for calling when the ISP Download, you can also call the release project program in a third-party application for ISP download. The following two are introd
project program is running .

Simple call

In a third-party application, it is just a simple process of creating a publishing project program. All other download operations are carrie
project program . At this time, the third-party application only needs to wait for the publishing project program to complete the operation, and

code VC

```
BOOL IspProcess()
{
        // Define related variables
        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        CString path;

        // Publish the full path of the project program
        path = _T("D:\\Work\\Upgrade. exe");

        // Variable initialization
        memset(&si, 0, sizeof(STARTUPINFO));
        memset(&pi, 0, sizeof(PROCESS_INFORMATION));

        // Set startup variables
        si. cb = sizeof(STARTUPINFO);
        GetStartupInfo(&si);
        si. wShowWindow = SW_SHOWNORMAL;
        si. dwFlags = STARTF_USESHOWWINDOW;

        // Create and publish the project program process
        if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
        {
                // Wait for the release project program operation to complete
                // Since the main process will be blocked here, it is recommended to create a new worker process and wait in the worker process.
                WaitForSingleObject(pi. hProcess,INFINITE);

                // Clean-up work /CloseHandle(pi.
                hThread); CloseHandle(pi. hProcess);

                return TRUE;
        }
        else
        {
                AfxMessageBox(_T("Creation process failed ! "));

                return FALSE;
        }
```

```
}
```

Advanced call

The process of creating and publishing a project program in a third-party application, and performing it in a third-party application, incl

Programming, stop Programming and closing the published project program, etc. Simulate download operation without the need to be bound to the release project program ISP ISP

Surface interaction.

VC code

```
// Define the data structure of the callback function parameters
struct CALLBACK_PARAM
{
        DWORD dwProcessId;                                  ID// Main process
        HWND hMainWnd;                                      // Main window handle
};

// The callback function of the enumeration window, used to obtain the handle of the main window
BOOL CALLBACK EnumWindowCallBack(HWND hWnd, LPARAM lParam)
{
        CALLBACK_PARAM *pcp = (CALLBACK_PARAM *)lParam;
        DWORD id;
        GetWindowThreadProcessId(hWnd, &id);
        if ((pcp->dwProcessId == id) && (GetParent(hWnd) == NULL))
        {
                pcp->hMainWnd = hWnd;

                return FALSE;
        }

        return TRUE;
}

BOOL IspProcess()
{
        // Define related variables
        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        CALLBACK_PARAM cp;
        CString path;

        // Publish some of the                  ID
        controls in the project program const   = 1046;
        UINT ID_PROGRAM const UINT ID_STOP      = 1044;
        const UINT ID_COMPORT                   = 1009;
        const UINT ID_PROGRESS                  = 1044;

        // Publish the full path of the project program
        path = _T("D:\\Work\\Upgrade. exe");

//      Variable initialization
memset(&si, 0, sizeof(STARTUPINFO));
memset(&pi, 0, sizeof(PROCESS_INFORMATION));
memset(&cp, 0, sizeof(CALLBACK_PARAM));
        //Set startup variables
        si. cb = sizeof(STARTUPINFO);
        GetStartupInfo(&si);
        si. wShowWindow = SW_SHOWNORMAL;                 //If this is set to                The release project program will not be displayed SW_
                                                         // The operation interface of ,Operations can be performed in the background
        si. dwFlags = STARTF_USESHOWWINDOW;

        // Create and publish the project program process
        if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
        {
                // Wait for the initialization of the release project program process to complete
                WaitForInputIdle(pi. hProcess, 5000);

                // Get the handle of the main window of the publishing project program
                cp. dwProcessId = pi. dwProcessId;
                cp. hMainWnd = NULL;
                EnumWindows(EnumWindowCallBack, (LPARAM)&cp);
                if (cp. hMainWnd ! = NULL)
                {
                        HWND hProgram;
                        HWND hStop;
                        HWND hPort;

                        // Get the handle of some controls in the main window of the publishing project program
                        hProgram = ::GetDlgItem(cp. hMainWnd, ID_PROGRAM);
                        hStop = ::GetDlgItem(cp. hMainWnd, ID_STOP);
                        hPort = ::GetDlgItem(cp. hMainWnd, ID_COMPORT);

                        // Set the serial port number in the release project program ,The first parameter is 3
                        ::SendMessage(hPort, CB_SETCURSEL, 0, 0);

                        //Trigger the programming button to start
                        ::SendMessage(hProgram, BM_CLICK, 0, 0);

                        //Wait for the programming to complete ,
                        // Since the main process will be blocked here, it is recommended to create a new worker process and wait in the worker process.
                        while (! ::IsWindowEnabled(hProgram));

                        // Close the release project program after the programming is complete
                        ::SendMessage(cp. hMainWnd, WM_CLOSE, 0, 0);
                }

                // Wait for the process to end
                WaitForSingleObject(pi. hProcess,INFINITE);

                //Clean-up work /CloseHandle(pi.
                hThread); CloseHandle(pi. hProcess);

                return TRUE;
        }
        else
        {
                AfxMessageBox(_T("Creation process failed ! "));

                return FALSE;
        }
}
```

M Appendix STC8H Series of orthogonal decoding examples (Chengdu Feifeike

(Provided by friendship)

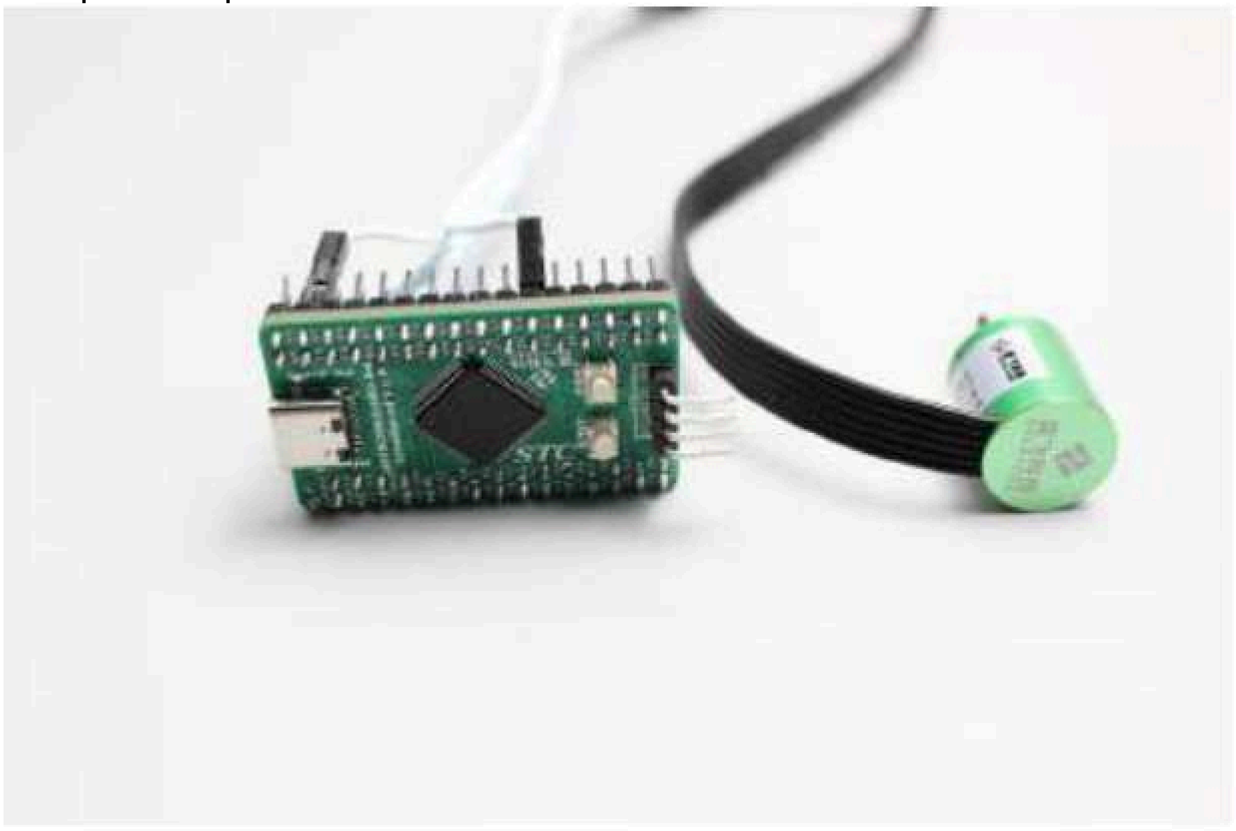
Subject to STC Entrusted, this article is intended for sharing STC8H Enhanced series of microcontrollers PWM The module implements the orthogonal decoding fur
One-step realization of two-way speed measurement of the encoder output of the quadrature encoded signal.

Hardware platform: fly-by-fly STC8H8K-64 + Feet fly by the core board 1024 line mini Quadrature encoder

compilation environment :

keil V9.60

Host computer: serial port assistant



From the previous open source libraries of Yifei Technology, we can understand that there is no orthogonal decoding routine in the oper
Only two PWM Module, if we recommend the use of quadrature coding encoder, it means that one encoder needs to occupy one PWM
Module, however, this year's energy-saving group requires the production of a balanced trolley, which means that there are two motors, so tw
the two modules of the microcontroller will be occupied, but the motor control of the trolley also requires functions, so it is not recommended
The module implements quadrature decoding, but it is recommended that everyone use an encoder with directional output, so that the modul

Of course, you can also have another idea, using one PWM The module implements the speed measurement of one quadrature-coded encoder,
The motor uses an encoder with a directional signal and an ordinary timer to capture the pulse. This scheme is feasible, but there is no need

One more thing to note, use PWM Module counting is not the same as using a timer module to capture pulse counting. PWM
The module captures encoder data through edge counting, which means that this module counts when the rising or falling
edge occurs , and the timer captures the pulse to obtain the number of high and low level flips. Here we will find out
through experiments that we use the same orthogonal coding encoder to rotate The encoder data collected by the module is twice as much as the pulse data captured by the timer, but this
Get taller, just The counting method of the single chip microcomputer has led to a doubling of the results.

The following is STC8H8K64U An example program for collecting quadrature encoded signals and outputting encoders : C the language code used

```
#include "headfile. h"


int16 encoder_data;


//-------------------------------------------------------------------
//@brief          PWMA Module orthogonal decoding initialization
//@param          void
//@return         void
//@since          v1.0
//Sample usage: PWMA_encoder_init();                          // Initialize orthogonal decoding
//@note
//-------------------------------------------------------------------
void PWMA_encoder_init(void)
{

        P_SW2 |= 1<<7;                                        // Enable access XSR
        PWMA_ARR = 0xFFFF;                                    // Set the automatic reload value, When the value of automatic reloading is 0 ,
                                                              // The counter does not work.

        PWMA_CCMR1 |= 1<<0;                                   //IC1 Mapped in TI1FP1 Pin On, that is, use, obtain direction
        PWMA_CCMR2 |= 1<<0;                                   //IC2 mapped in TI2FP2 Pin to, that is, use P22, capture edge jump
        PWMA_SMCR |= 1<<0;                                    // Encoder mode 1     TI1FP1 According to the level,
                                                              // The counter is here TI2FP2 the edge is upward / Count down

        PWMA_CR1 |= 1<<0;                                     PWMA // Enable counter
        PWMA_PS |= 1<<2;                                      //PWMA 1 Channel use P10,PWMB 2 Channel use P22°
}


//-------------------------------------------------------------------
// @brief         PWMA The module obtains the orthogonal decoding value
// @param         void
// @return        void
// @since         v1.0
// Sample usage: encoder_data = PWMA_get_encoder();           // Get orthogonal decoding value
// @note
//-------------------------------------------------------------------
int16 PWMA_get_encoder(void)
{

        int16 res;


        res = PWMA_CNTR;                                      // Save the value of the current counter
        PWMA_CNTR = 0;                                        // Clear the counter
        return res;

}


//-------------------------------------------------------------------
// @brief         Timer 0 5ms Interrupt service function
// @param         void
// @return        void
// @since         v1.0
// Sample usage:
// @note
//-------------------------------------------------------------------
void TM0_Isr() interrupt 1
{

        encoder_data = PWMA_get_encoder();                    // Get the value of the quadrature decoding encoder
}


void main()
```

```
{
        DisableGlobalIRQ();                              //Turn off the total interrupt
        board_init();                                    // Initialize the internal register, do not delete this code.
        pit_timer_ms(TIM_0, 5);                          Execute once //Initialize the timer , 5ms
        PWMA_encoder_init();                             //PWMA   The module is initialized to orthogonal decoding function
        EnableGlobalIRQ();                               //Turn on the total interrupt
        while(1)
        {
                delay_ms(100);                                  Output print information once
                printf("encoder_data = %d \r\n" ,encoder_data); //   100ms //
                                                         Print encoder data per serial port
        }
}
```

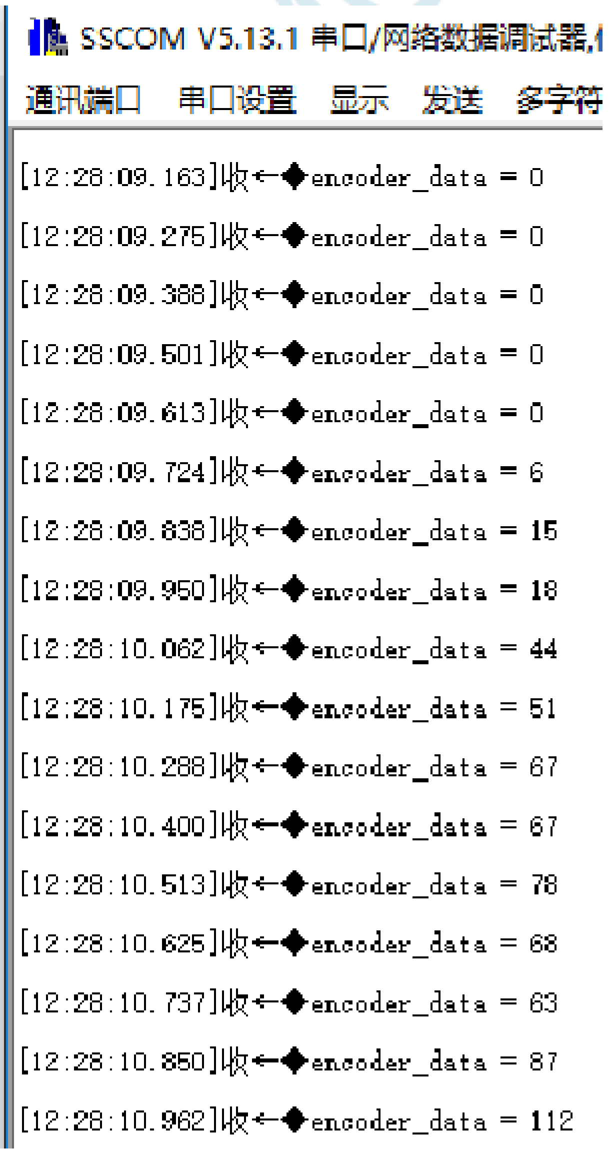
Demo video link：https://www.bilibili.com/video/BV1zT4y177Ht

Video description: We will compile the written routine, then download it to the microcontroller, open the serial port assistant to receive t and the rotary encoder observes the data changes. We found that when the encoder is not rotating, the output data is, and when the encoder output both positive and negative values, the faster the rotation., The greater the absolute value of the value, the positive and negative are us rotation, which direction is positive and which direction is negative can be defined by yourself. At the same time, we can also see from the pro Once, and the data collection is

Once, so the printed data is equivalent to intermittent, and at the same time because the high precision of the line, so the data is observe 1024 The encoder The change is relatively large, but if the motor is used to drive at no load duty cycle, you can see that the data output of the encoder is very stal Driven by the

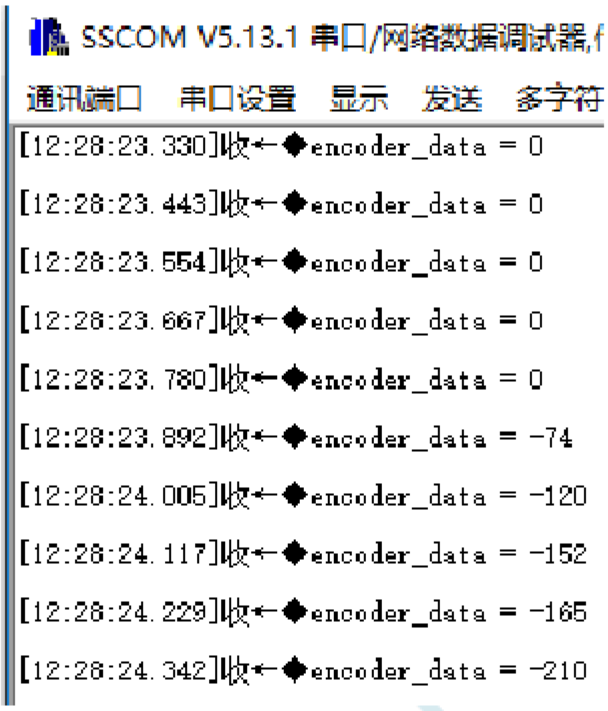
Serial port assistant receives screenshots of data：

The figure below shows the data that the orthogonally encoded encoder rotates clockwise and the angular velocity gradually increases.



```
[12:28:09.163]收←◆encoder_data = 0
[12:28:09.275]收←◆encoder_data = 0
[12:28:09.388]收←◆encoder_data = 0
[12:28:09.501]收←◆encoder_data = 0
[12:28:09.613]收←◆encoder_data = 0
[12:28:09.724]收←◆encoder_data = 6
[12:28:09.838]收←◆encoder_data = 15
[12:28:09.950]收←◆encoder_data = 18
[12:28:10.062]收←◆encoder_data = 44
[12:28:10.175]收←◆encoder_data = 51
[12:28:10.288]收←◆encoder_data = 67
[12:28:10.400]收←◆encoder_data = 67
[12:28:10.513]收←◆encoder_data = 78
[12:28:10.625]收←◆encoder_data = 68
[12:28:10.737]收←◆encoder_data = 63
[12:28:10.850]收←◆encoder_data = 87
[12:28:10.962]收←◆encoder_data = 112
```

The figure below is the data when the orthogonally encoded encoder rotates counterclockwise and the angular velocity gradually increa



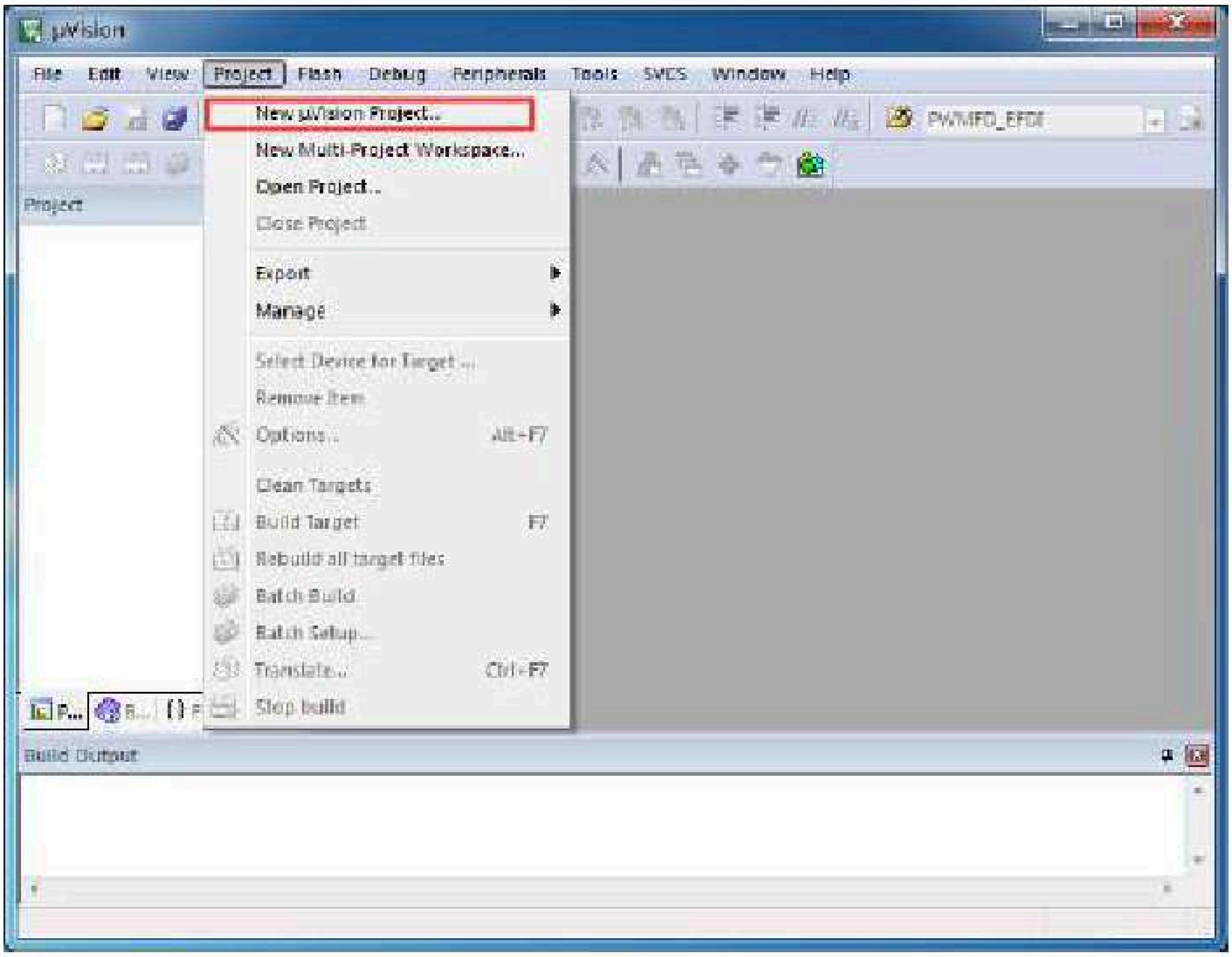
```
[12:28:23.330]收←◆encoder_data = 0
[12:28:23.443]收←◆encoder_data = 0
[12:28:23.554]收←◆encoder_data = 0
[12:28:23.667]收←◆encoder_data = 0
[12:28:23.780]收←◆encoder_data = 0
[12:28:23.892]收←◆encoder_data = -74
[12:28:24.005]收←◆encoder_data = -120
[12:28:24.117]收←◆encoder_data = -152
[12:28:24.229]收←◆encoder_data = -165
[12:28:24.342]收←◆encoder_data = -210
```

N Appendix in Keil How to create a multi-file project in

in Keil In general, relatively small projects have only one source file, but for some slightly complex projects, multiple source files are often n

The method of creating a multi-file project is as follows :

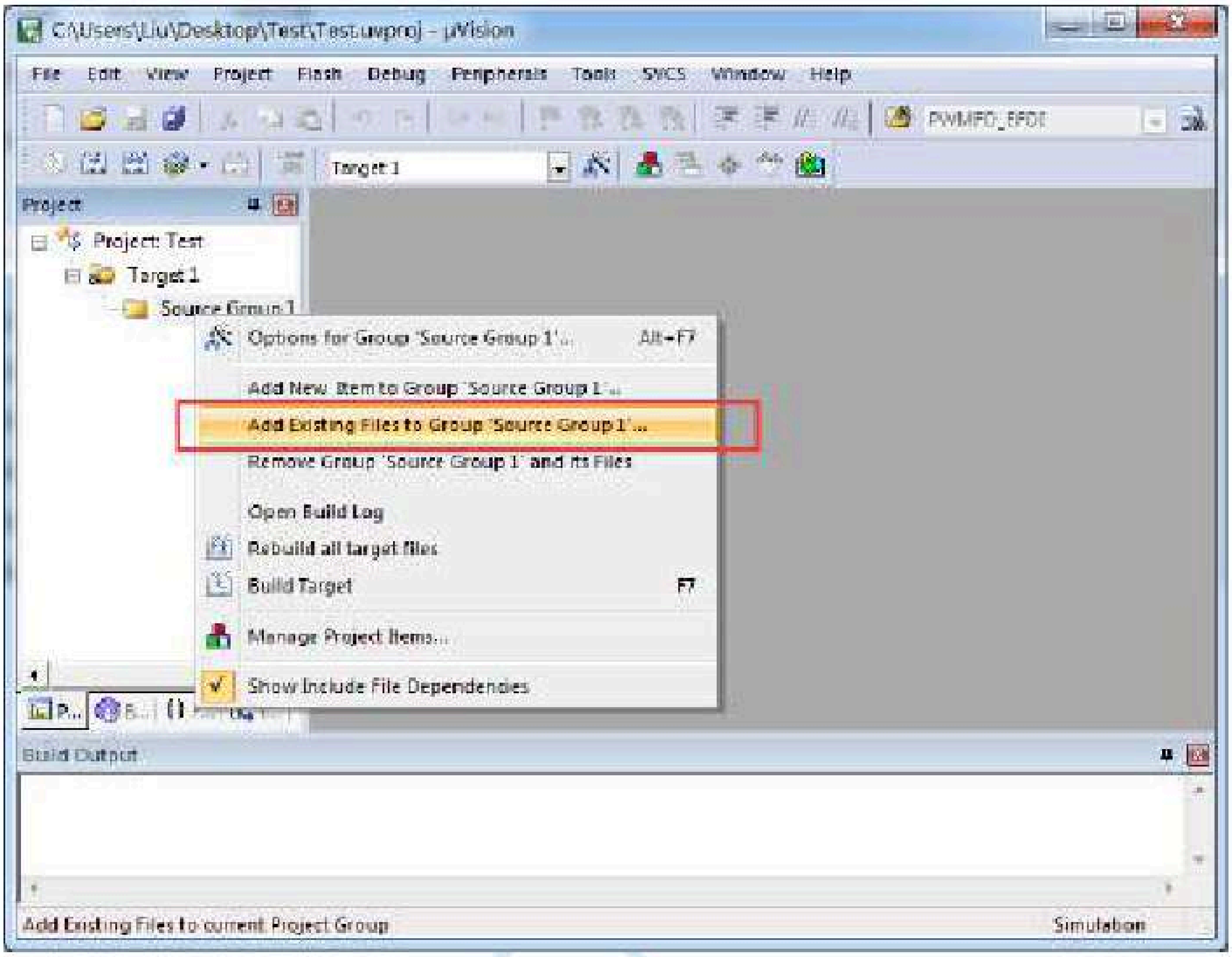
1, Open first Keil, in the menu" Project "Medium selection" New uVision Project . . ."



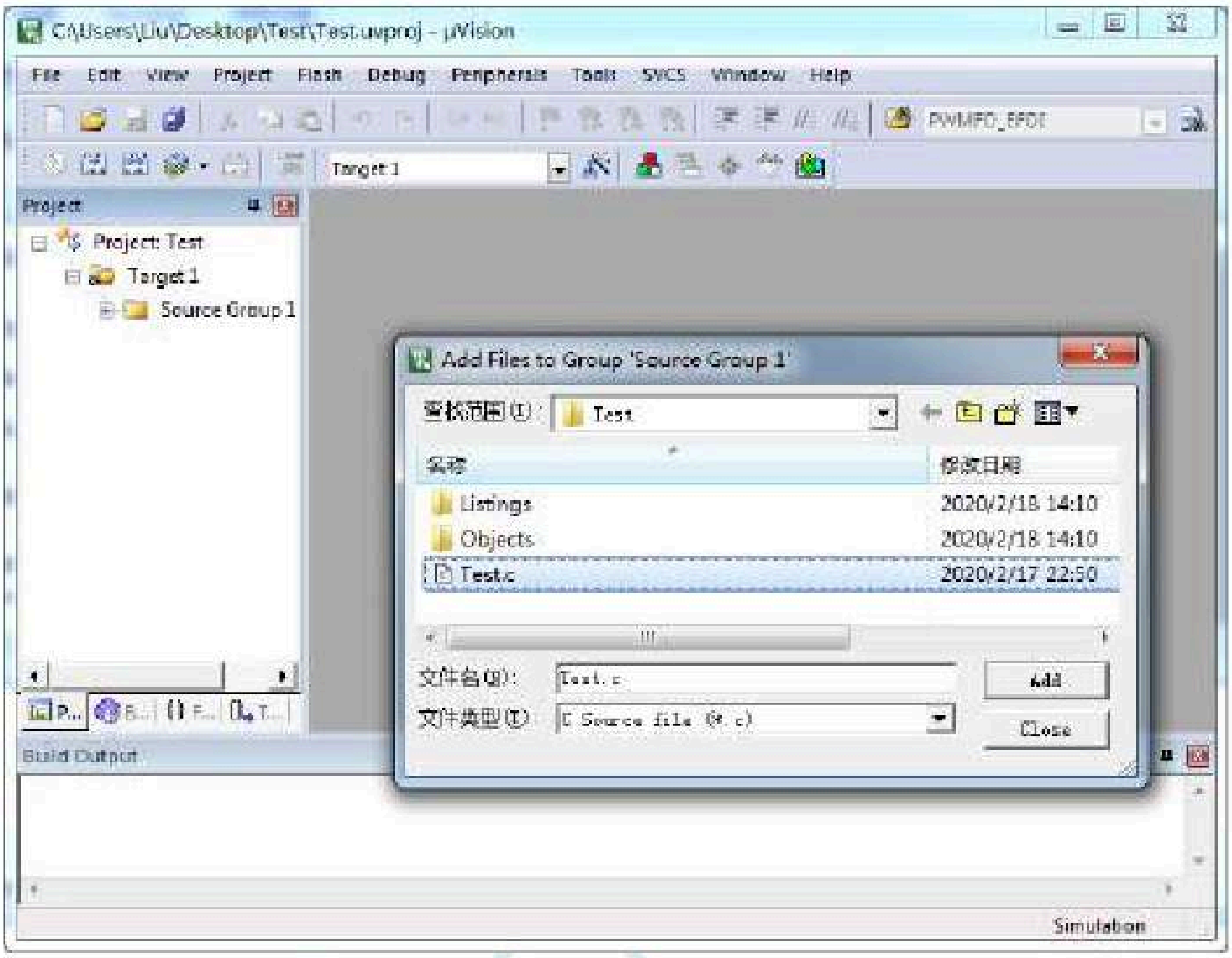
You can complete the establishment of an empty project

2 , In the project tree of the empty project, ", and select "in the context menu Add Existing Files to Source Group 1

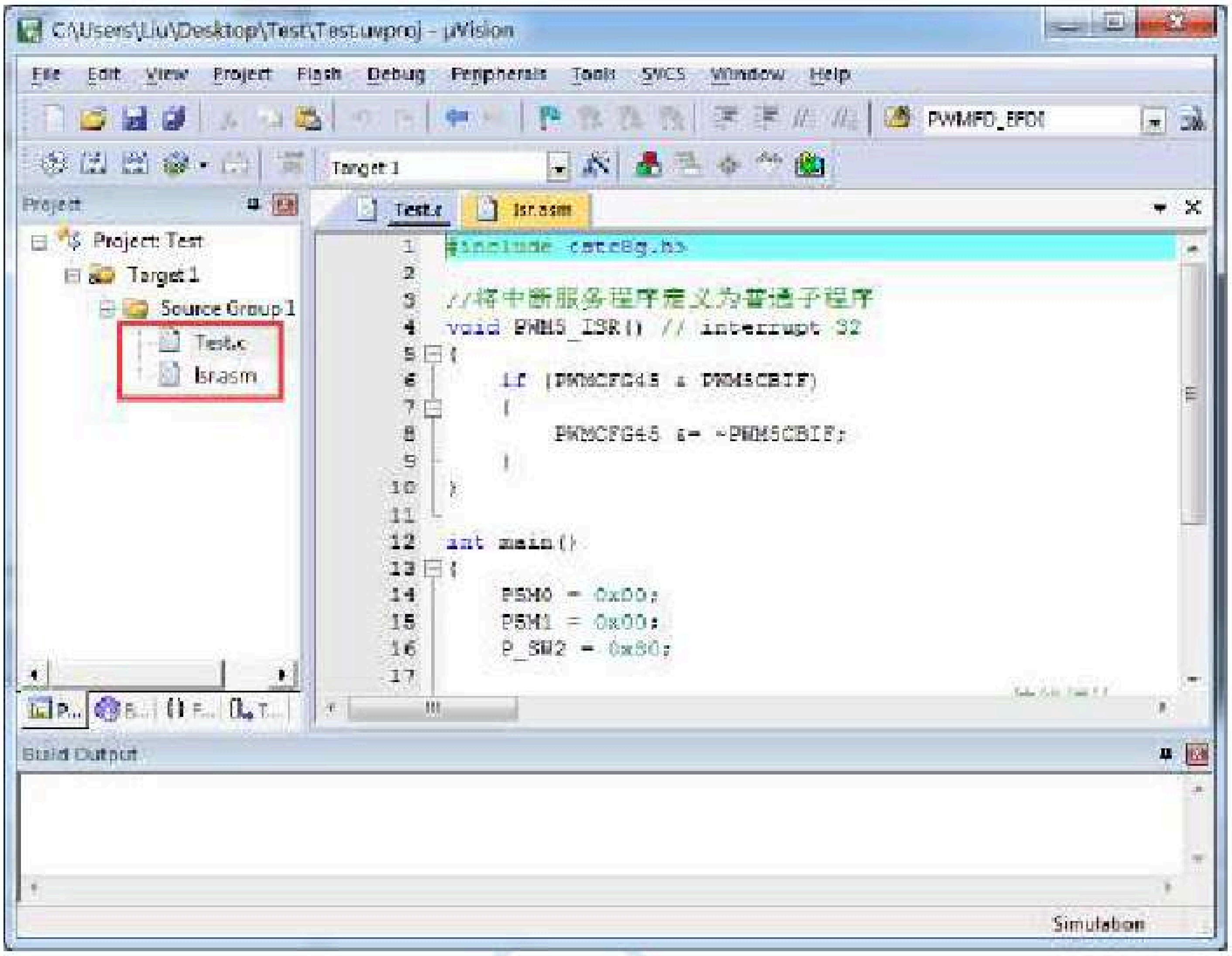
right-click " Group "Source Group 1" . . ."


C:\Users\Liu\Desktop\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
PWMPD_EPDI
Target 1
Project
Project: Test
Target 1
Source Group 1
Options for Group 'Source Group 1'... Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files
Open Build Log
Rebuild all target files
Build Target F7
Manage Project Items...
Show Include File Dependencies
Build Output
Add Existing Files to current Project Group
Simulation

3 、 **In the pop-up file dialog box, add the source file multiple times**



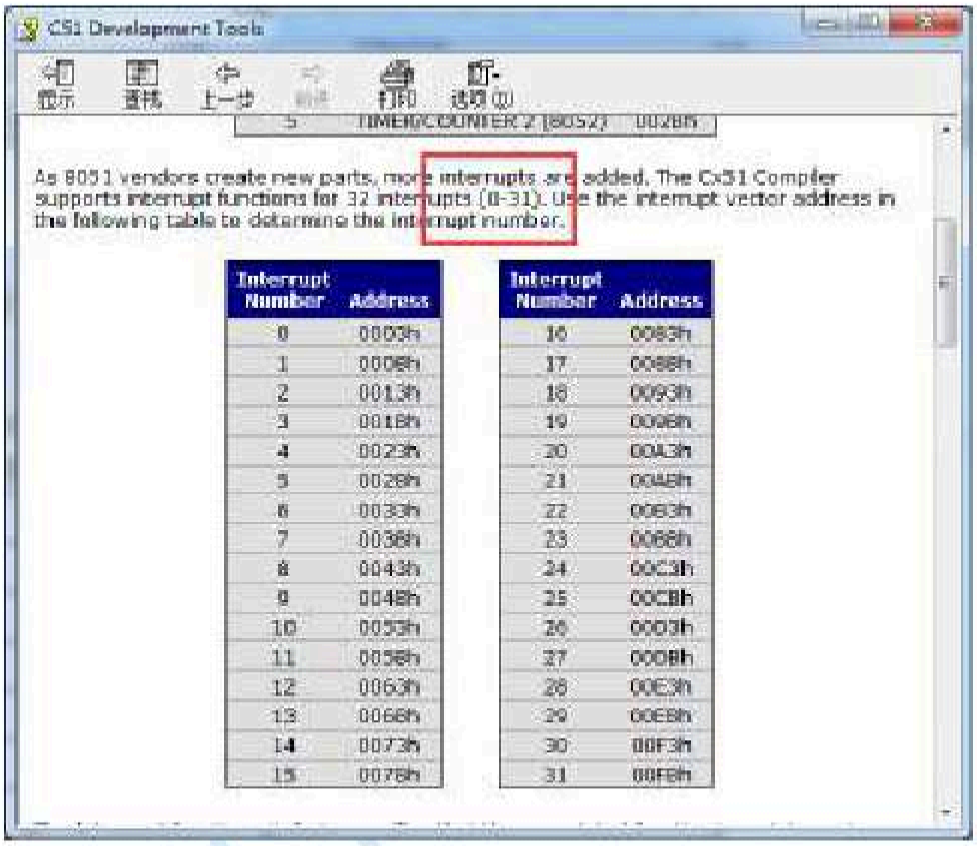
The establishment of a multi-file project can be completed as shown in the figure below


C:\Users\Liu\Desktop\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
PWMFD_EFDI
Target 1
Project
Project: Test
Target 1
Source Group 1
Test.c
Isr.asm
Test.c
Isr.asm
Build Output
Simulation

```
#include <stc8g.h>

//将中断服务程序定义为普通子程序
void PWM5_ISR() // interrupt 32
{
    if (PWMCFG45 & PWM5CBIF)
    {
        PWMCFG45 &= ~PWM5CBIF;
    }
}

int main()
{
    P5M0 = 0x00;
    P5M1 = 0x00;
    P_SW2 = 0x80;
```

O Appendix About the interrupt number is greater than 31 Keil Compilation error Deal with

in Keil of C51 In the compilation environment, the interrupt number must be less than 32, That is, the interrupts supported


C51 Development Tools

5 TIMER/COUNTER 2 (8052) 002Bh

As 8051 vendors create new parts, more interrupts are added. The Cx51 Compiler supports interrupt functions for 32 interrupts (0-31). Use the interrupt vector address in the following table to determine the interrupt number.

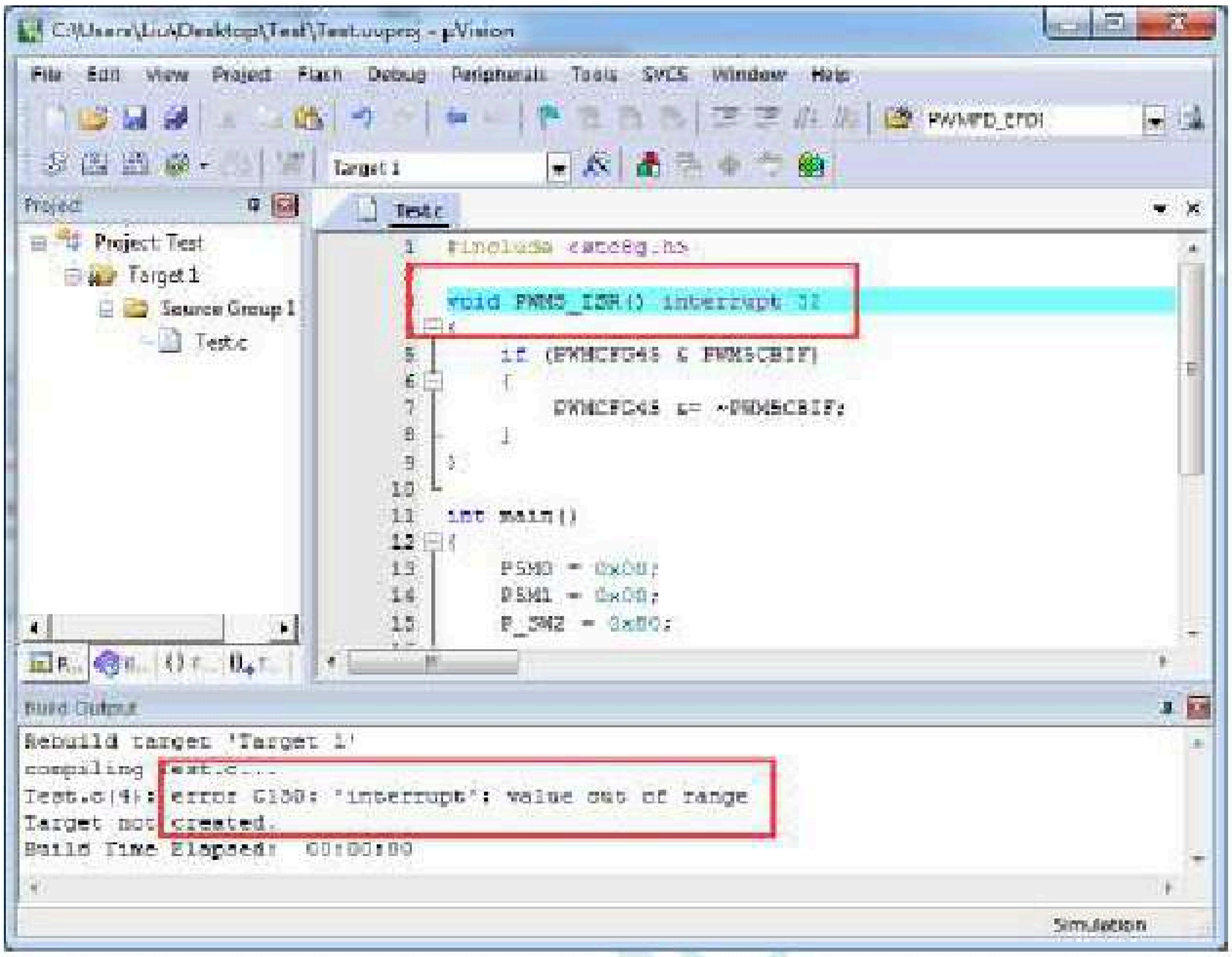
Interrupt Number	Address	Interrupt Number	Address
0	0003h	16	0083h
1	000Bh	17	008Bh
2	0013h	18	0093h
3	001Bh	19	009Bh
4	0023h	20	00A3h
5	002Bh	21	00ABh
6	0033h	22	00B3h
7	003Bh	23	00BBh
8	0043h	24	00C3h
9	004Bh	25	00CBh
10	0053h	26	00D3h
11	005Bh	27	00DBh
12	0063h	28	00E3h
13	006Bh	29	00EBh
14	0073h	30	00F3h
15	007Bh	31	00FBh

The table below is STC List of current interrupts for all series :

Interrupt number	Interrupt vector	Interrupt
0	0003 H	type INT0
1	000B H	Timer 0
2	0013 H	INT1
3	001B H	Timer 1
4	0023 H	serial port 1
5	002B H	ADC
6	0033 H	LVD
7	003B H	PCA
8	0043 H	serial port 2
9	004B H	SPI
10	0053 H	INT2
11	005B H	INT3
12	0063 H	Timer 2
13	006B H	
14	0073 H	
15	007B H	Internal system
		interrupt Internal system interrupt

16	0083 H	INT4
17	008B H	serial port 3
18	0093 H	serial port 4
19	009B H	Timer 3
20	00A3 H	Timer co4mparator
21	00AB H	waveform
22	00B3 H	generator 0
23	00BB H	Abnormal waveform 0 generator
24	00C3 H	I2C
25	00CB H	USB
26	00D3 H	PWMA
27	00DB H	PWMB
28	00E3 H	1 Waveform generator , waveform
29	00EB H	2
30	00F3 H	generator, waveform generator, waveform generator, 3
31	00FB H	waveform generator Waveform generator
32	0103 H	5 , waveform generator, abnormal
33	010B H	waveform generator, 2 abnormal
34	0113 H	4 touch button
35	011B H	
36	0123 H	RTC
37	012B H	P0 Port Interrupt Port
38	0133 H	P1 Interrupt port Interrupt
39	013B H	P2 Port Interrupt Port
40	0143 H	P3 Interrupt Port Interrupt
41	014B H	P4 Port Interrupt
42	0153 H	P5 Port Interrupt
43	015B H	P6 Port Interrupt
44	0163 H	P7 Port Interrupt
45	016B H	P8 Port Interrupt
46	0173 H	P9 Port Interrupt

It is not difficult to find that from 5 The waveform generator, The interrupt begins, and all subsequent interrupt keil service programs will be compilation errors in both, as shown in t



```
#include <stc8g.h>

void PWM5_ISR() interrupt 32
{
    if (PWMCFG45 & PWM5CBIF)
    {
        PWMCFG45 &= ~PWM5CBIF;
    }
}

int main()
{
    P5M0 = 0x00;
    P5M1 = 0x00;
    P_SW2 = 0x80;
```

```
Rebuild target 'Target 1'
compiling Test.c...
Test.c(4): error C130: 'interrupt': value out of range
Target not created.
Build Time Elapsed:  00:00:00
```

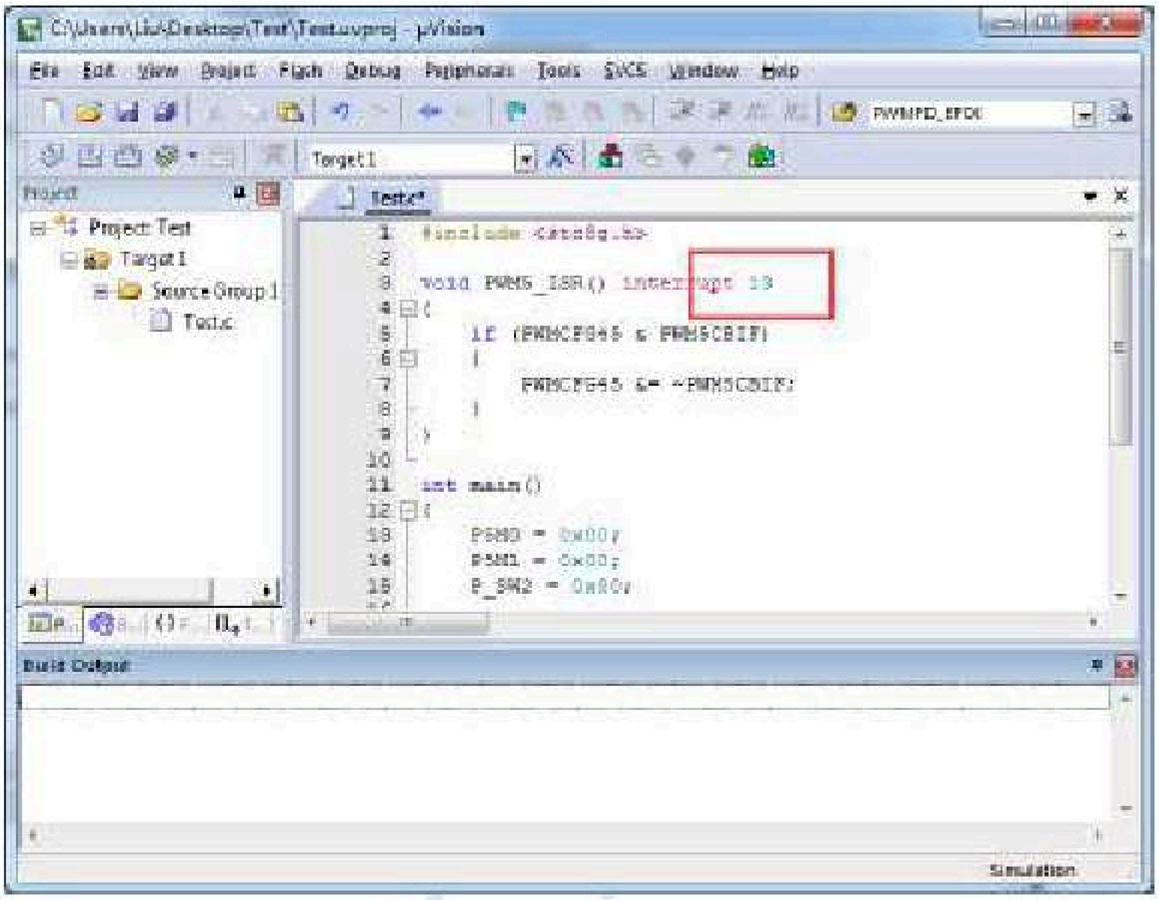
There are three ways to deal with this(All require the help of assembly code, the preferred method is recommended)1

13 1 Method: Borrow interrupt vector

0~31 The signal is interrupted, the first, 13 The number is a reserved interrupt number, we can borrow this interrupt number

The steps are as follows:

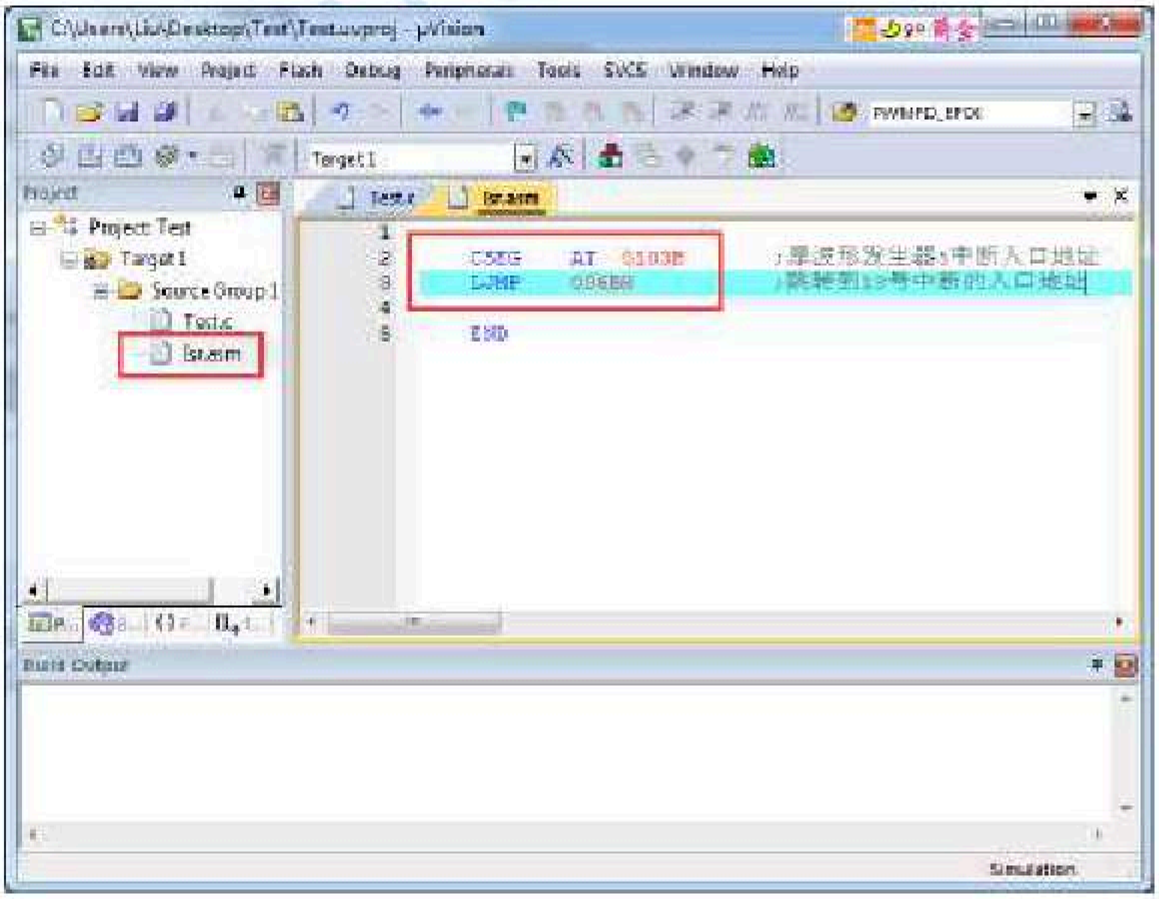
1, Change the interrupt number we reported the error to " 13 ", as shown below：



C:\Users\Liu\Desktop\Test\Test.uvproj - µVision

```
#include <stc8h.h>

void PWM5_ISR() interrupt 13
{
    if (PWMCFG45 & PWM5CBIF)
    {
        PWMCFG45 &= ~PWM5CBIF;
    }
}

int main()
{
    P5M0 = 0x00;
    P5M1 = 0x00;
    P_SW2 = 0x80;
```

2 , Create a new assembly language file, such as " isr. asm ", Add to the project and at the address " 0103H " Add one in the place " LJMP 006BH ", as shown below：

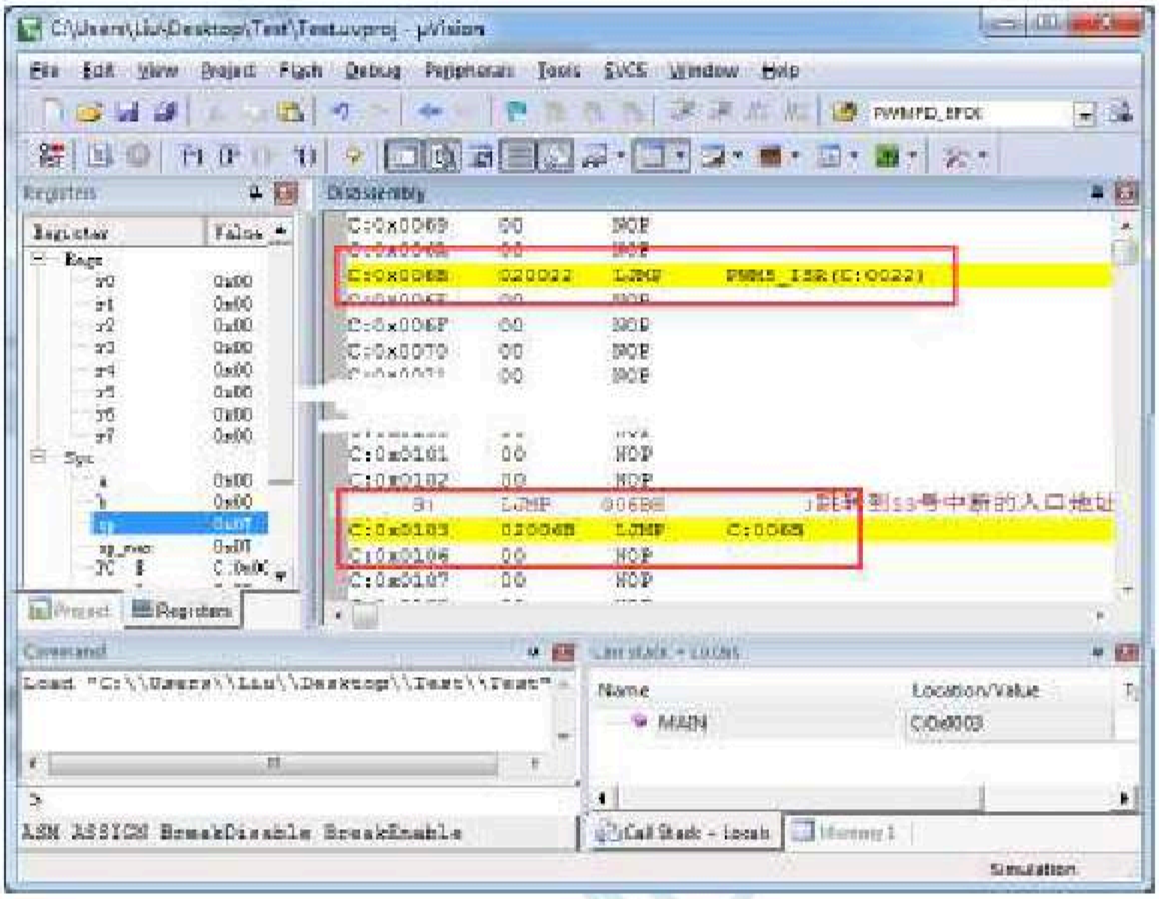


C:\Users\Liu\Desktop\Test\Test.uvproj - µVision

```
        CSEG    AT  0103H           ;[illegible]波形发生器5中断入口地址
        LJMP    006BH               ;跳转到13号中断的入口地址

        END
```

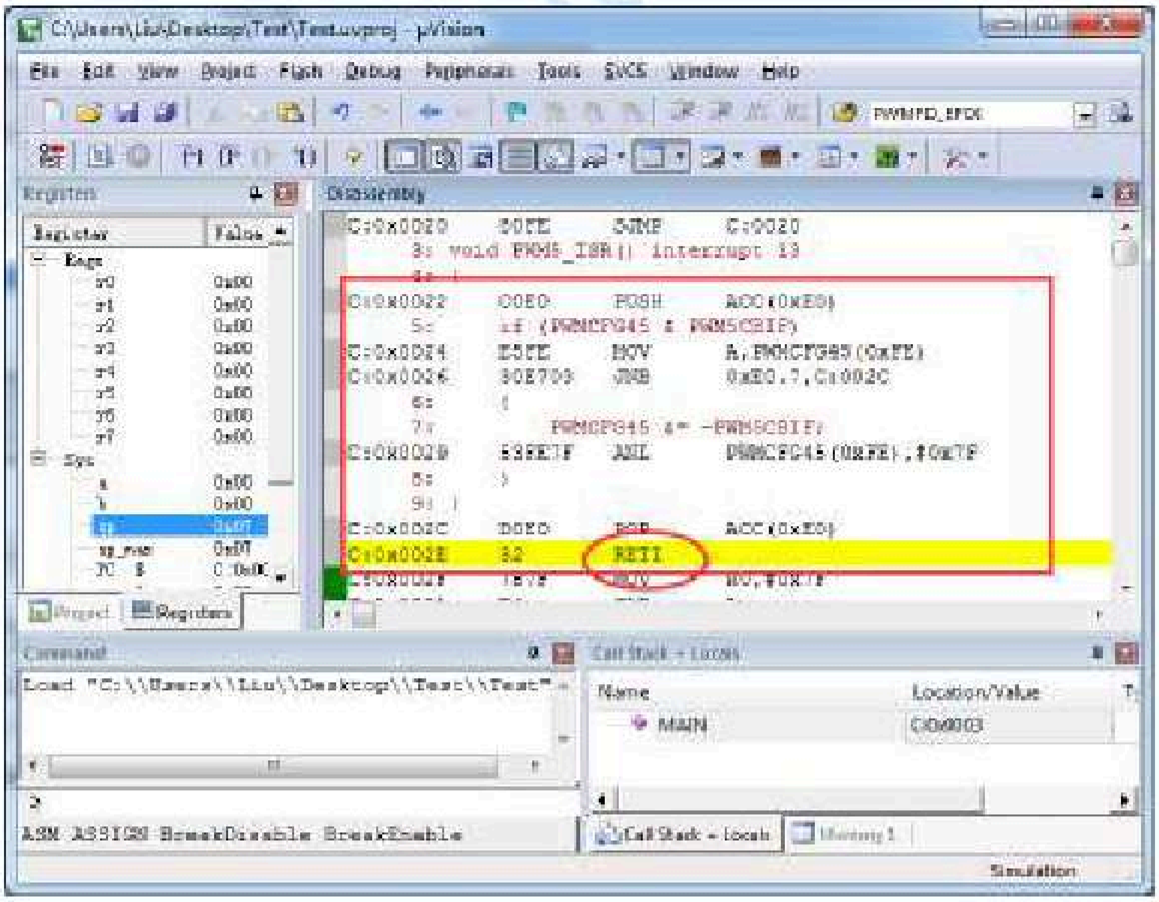
3, Compilation can be passed.

Pass by at After the compiler is compiled, in 006BH There is one here." LJMP PWM5_ISR", in 0103H There is one here this time" LJMP C51 006BH ", as shown below :



```
C:0x0069    00        NOP
C:0x006A    00        NOP
C:0x006B    020022    LJMP     PWM5_ISR(C:0022)
C:0x006E    00        NOP
C:0x006F    00        NOP
C:0x0070    00        NOP
C:0x0071    00        NOP
...
C:0x0101    00        NOP
C:0x0102    00        NOP
      31:   LJMP      006BH          ;跳转到13号中断的入口地址
C:0x0103    02006B    LJMP     C:006B
C:0x0106    00        NOP
C:0x0107    00        NOP
```

When the PWM5 time occurs" LJMP When interrupted, the hardware will automatically jump to 0103H Address execution" LJMP 006BH", and then in 006BH Execute again That is, jump to PWM5_ISR the real interrupt service program, as shown in the figure below :

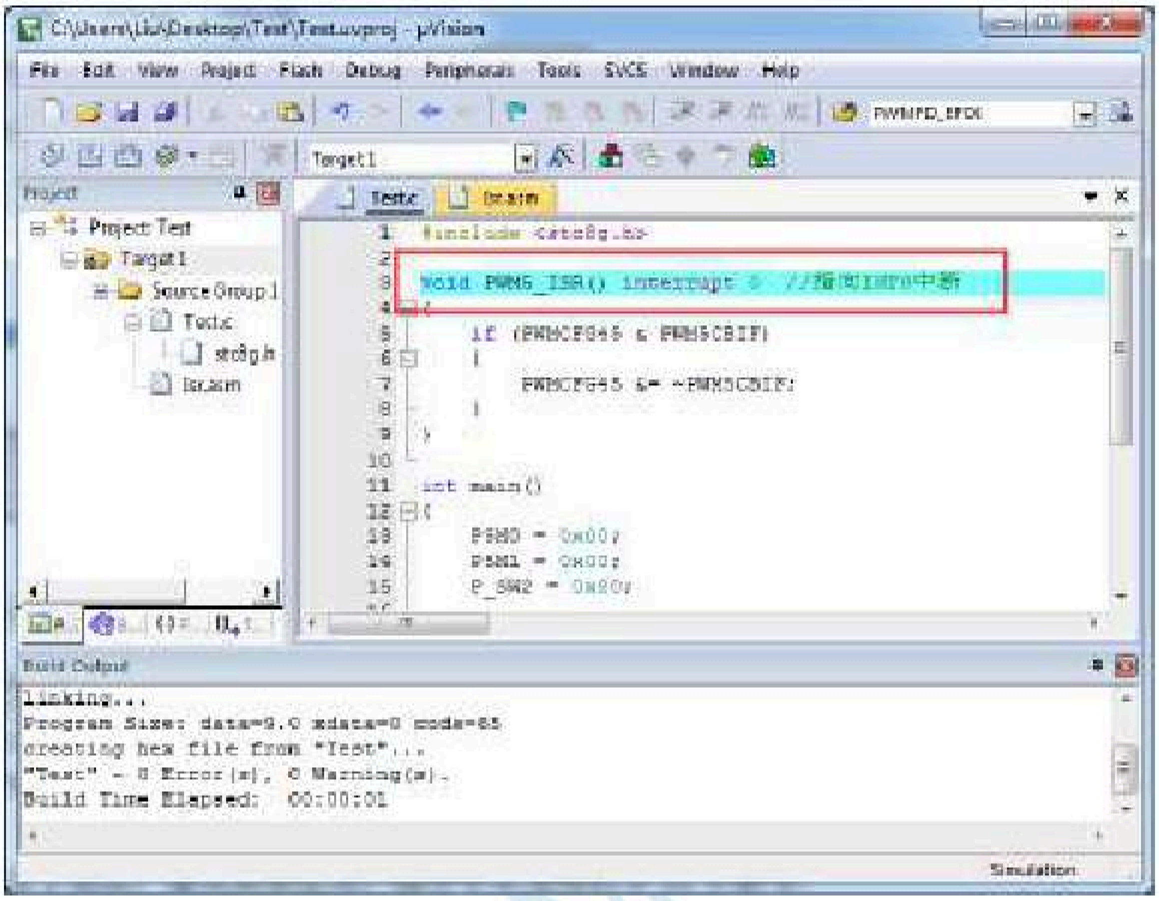


```
C:0x0020    80FE      SJMP     C:0020
     3: void PWM5_ISR() interrupt 13
     4: {
C:0x0022    C0E0      PUSH     ACC(0xE0)
     5:     if (PWMCFG45 & PWM5CBIF)
C:0x0024    E5FE      MOV      A,PWMCFG45(0xFE)
C:0x0026    30E703    JNB      0xE0.7,C:002C
     6:     {
     7:         PWMCFG45 &= ~PWM5CBIF;
C:0x0029    53FE7F    ANL      PWMCFG45(0xFE),#0x7F
     8:     }
     9: }
C:0x002C    D0E0      POP      ACC(0xE0)
C:0x002E    32        RETI
```

After the execution of the interrupt service RETI The command returns. The entire interrupt response process is just one more execution LJMP Statement and program is completed, it will be passed again .

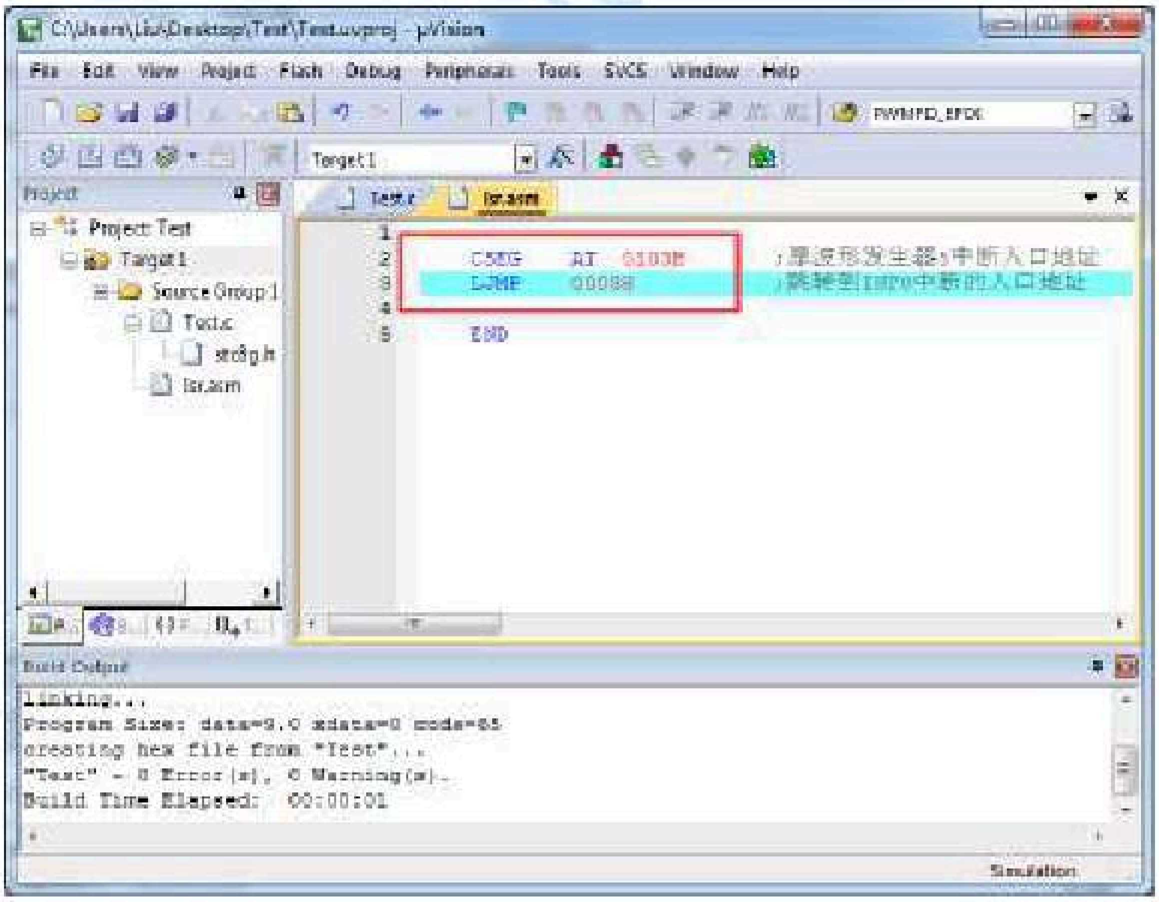
2 Method: and method 1 similarly, borrow unused ones in the user program 0~31 The interrupt number

For example, in the user's code, it is not used INT0 Interrupt, you can use the above code as a similar Modification of :



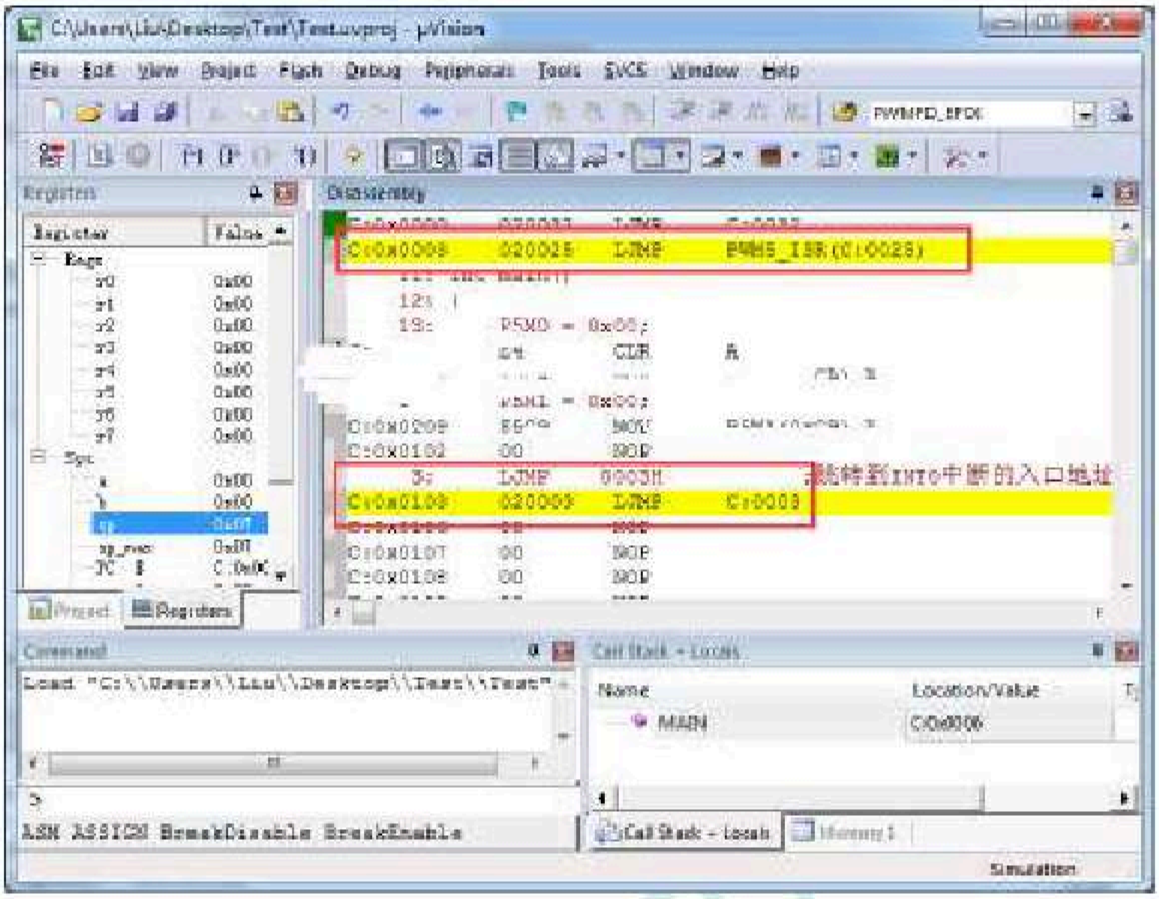
```c
#include <stc8g.h>

void PWM5_ISR() interrupt 0  //借用INT0中断
{
    if (PWMCFG45 & PWM5CBIF)
    {
        PWMCFG45 &= ~PWM5CBIF;
    }
}

int main()
{
    P5M0 = 0x00;
    P5M1 = 0x00;
    P_SW2 = 0x80;
```

```
linking...
Program Size: data=9.0 xdata=0 code=85
creating hex file from "Test"...
"Test" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```



```
        CSEG    AT    0103H         ;波形发生器5中断入口地址
        LJMP    0003H               ;跳转到INT0中断的入口地址

        END
```

```
linking...
Program Size: data=9.0 xdata=0 code=85
creating hex file from "Test"...
"Test" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```



```
C:0x0008    020025    LJMP    PWM5_ISR(C:0028)
    12: {
    13:     P5M0 = 0x00;
            E4        CLR     A
            P5M1 = 0x00;
C:0x0209    F5C9      MOV
C:0x0102    00        NOP
     3:     LJMP      0003H
C:0x0103    020003    LJMP    C:0003
C:0x0107    00        NOP
C:0x0108    00        NOP
```

跳转到INT0中断的入口地址

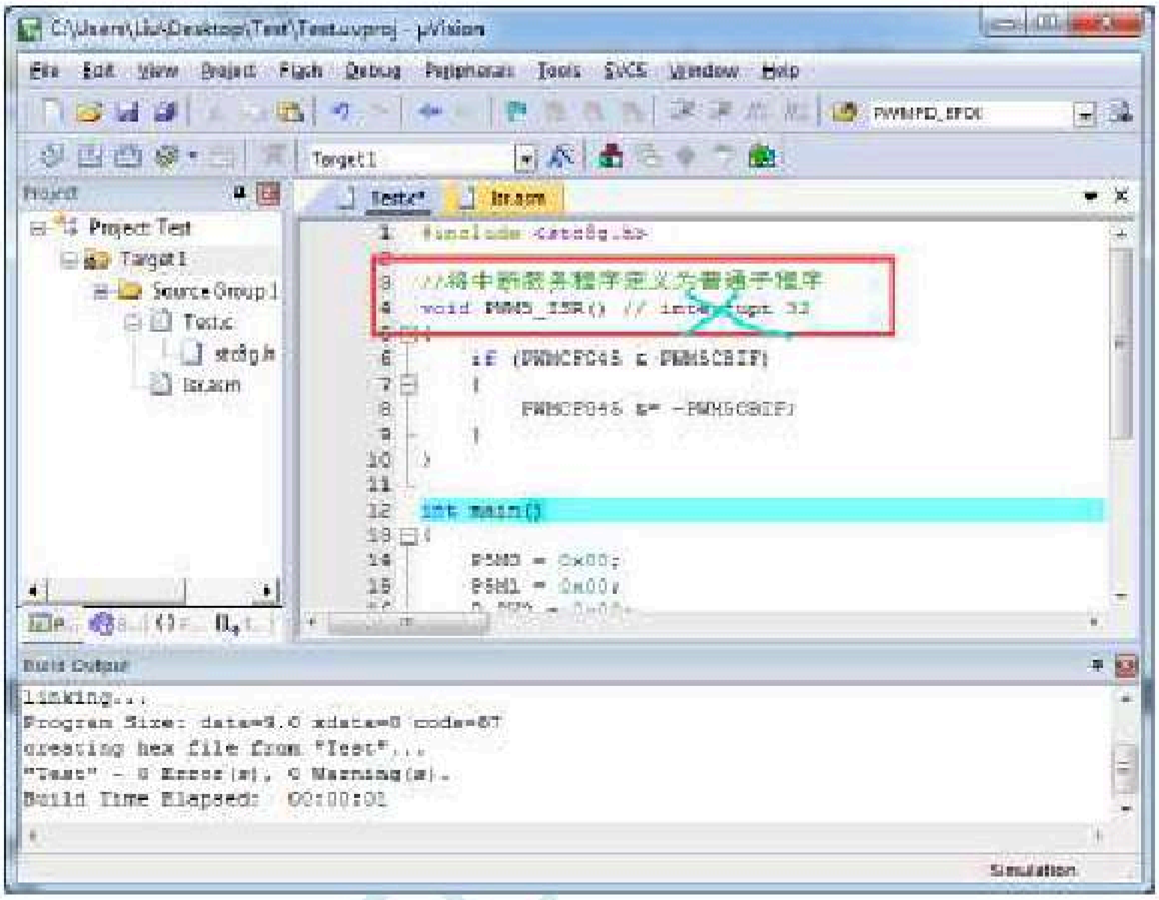
```
Load "C:\\Users\\Liu\\Desktop\\Test\\Test"
```

Execution effect and method The same, this method is suitable for multiple interrupt numbers that need to be remapped. The situation.

Method 3: Define the interrupt service program as a subroutine, and then in the interrupt entry address in the assembly code use LCALL Instruction execution service program

The steps are as follows:

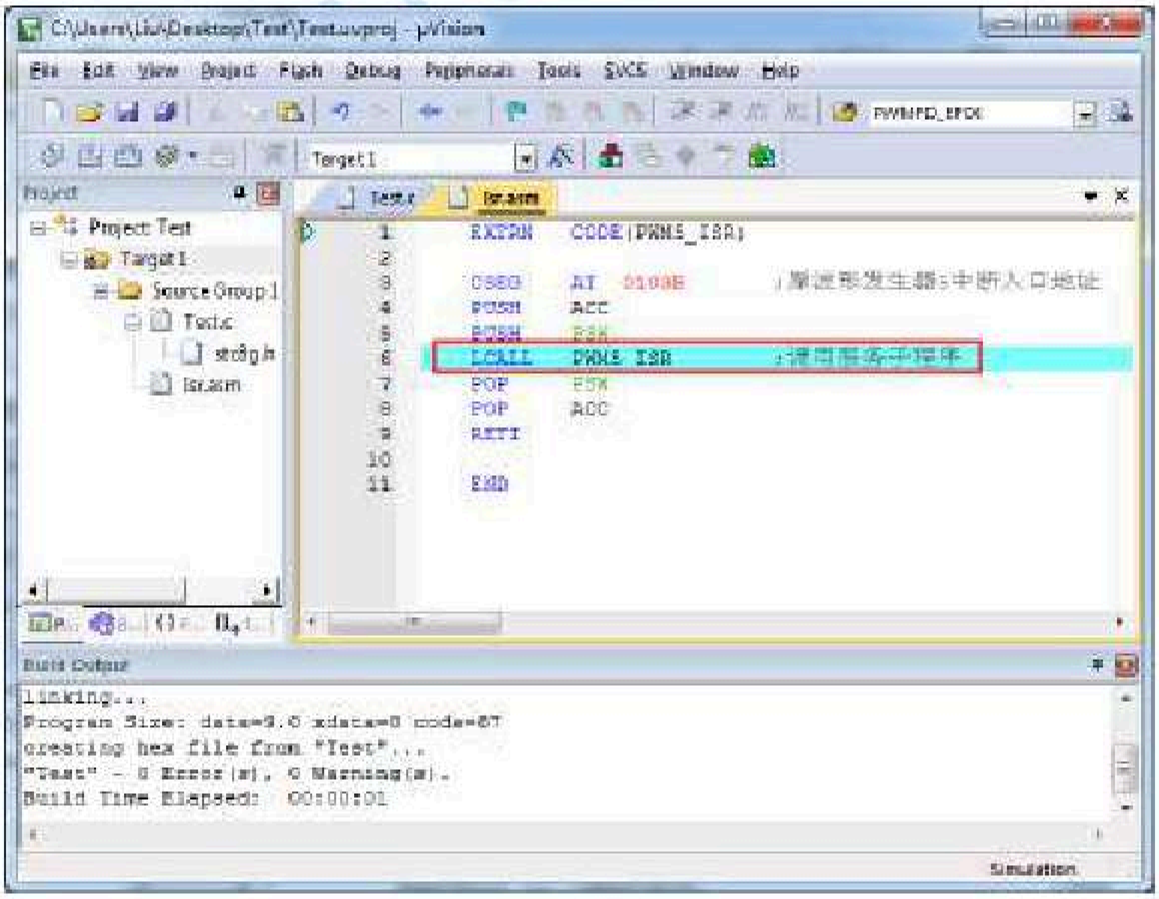
1 **, First remove the interrupt service program" interrupt" Attributes, defined as ordinary subroutines**



```
#include <stc8g.h>

//将中断服务程序定义为普通子程序
void PWM5_ISR() // interrupt 32
{
    if (PWMCFG45 & PWM5CBIF)
    {
        PWMCFG45 &= ~PWM5CBIF;
    }
}

int main()
{
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```
linking...
Program Size: data=9.0 xdata=0 code=87
creating hex file from "Test"...
"Test" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```

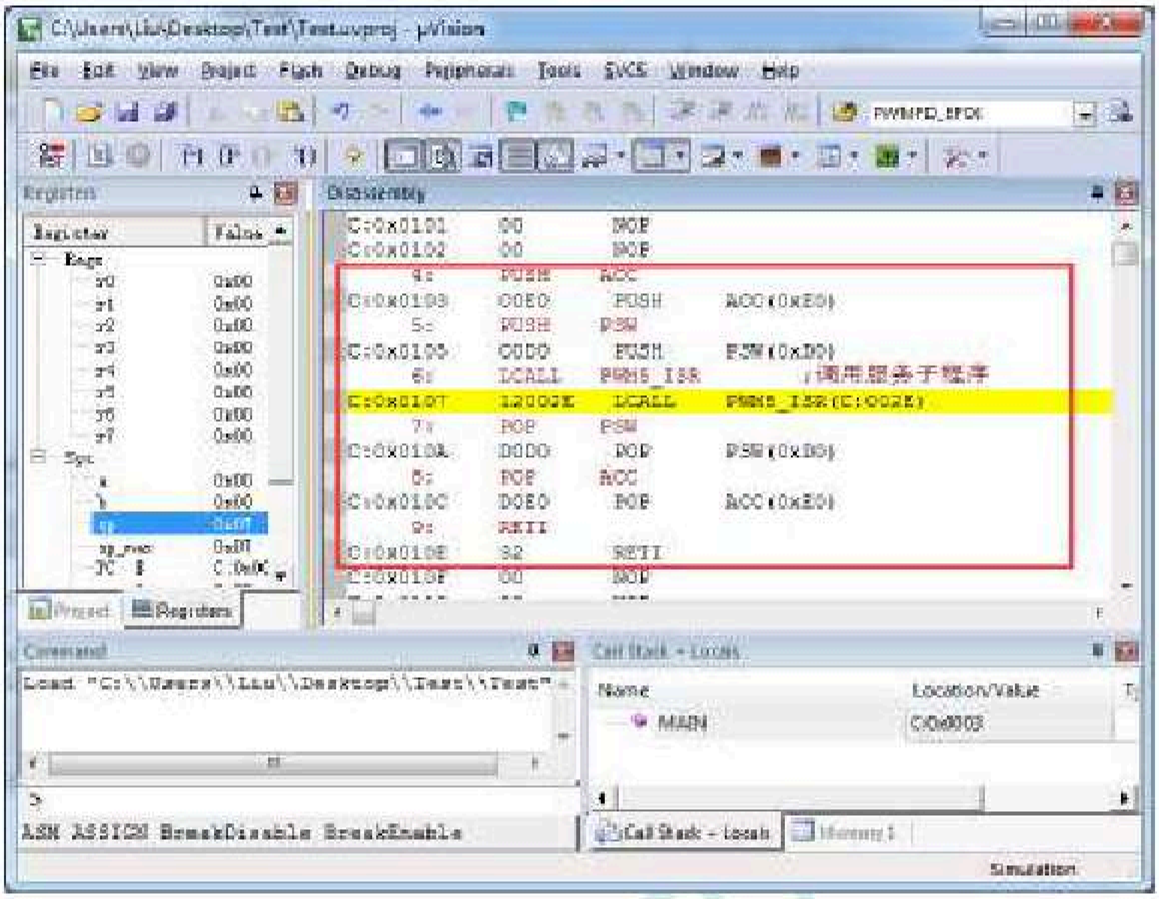
2 **, And then in the compilation of the file Enter the address code as shown in the figure below**



```
        EXTRN   CODE(PWM5_ISR)

        CSEG    AT      0103H           ;波形发生器5中断入口地址
        PUSH    ACC
        PUSH    PSW
        LCALL   PWM5_ISR                ;调用服务子程序
        POP     PSW
        POP     ACC
        RETI

        END
```

```
linking...
Program Size: data=9.0 xdata=0 code=87
creating hex file from "Test"...
"Test" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```

3 **The place of the address is to interrupt the service program, and after the compilation is passed, i**



```
C:0x0101    00        NOP
C:0x0102    00        NOP
     4:     PUSH     ACC
C:0x0103    C0E0      PUSH     ACC(0xE0)
     5:     PUSH     PSW
C:0x0105    C0D0      PUSH     PSW(0xD0)
     6:     LCALL    PWM5_ISR           ;调用服务子程序
C:0x0107    12002E    LCALL    PWM5_ISR(C:002E)
     7:     POP      PSW
C:0x010A    D0D0      POP      PSW(0xD0)
     8:     POP      ACC
C:0x010C    D0E0      POP      ACC(0xE0)
     9:     RETI
C:0x010E    32        RETI
C:0x010F    00        NOP
```

```
Load "C:\\Users\\Liu\\Desktop\\Test\\Test"
>
ASM ASSIGN BreakDisable BreakEnable
```

This method does not require remapping the interrupt entry, but there is a problem with this method. Which registers need to be pressed into the stack in the assembly file, the user needs to view the disassembled code of the program is determined. Generally includes DPH

PSW **In addition to the stack must be pressed, which other registers are used in the user's subroutine, and which registers must be pressed**

P Appendix Electrical characteristics

P.1 Absolute maximum rating

Parameter	Minimum value	Maximum value	unit	description
storage temperature	-55	+150	°C	
Operating temperature	-40	+85	°C	If the operating temperature is higher than 85 °C nearby), due to 85 °C (For example, the freq inside IRC has a large temperature drift at high temperatures, it is recommended to External high temperature clock or crystal oscillator. In addition, the frequency does when the temperature is high . If you must use an internal clock, it is recommended 24M Operating frequency; if the system must run at a higher frequency, please use an external highly reliable active clock. If the operating temperature is -55 Nature°C, the operating voltage cannot be too high Low, highly recommended MCU-VCC The voltage should not be lower than 3.0V. In addition, the rising speed of the power supply must also be as fast as possible, preferably controlled at the millisecond level
Operating voltage	1.9	5.5	V	
VDD Ground voltage	-0.3	+5.5	V	
I/O port ground voltage	-0.3	VDD+0.3	V	

P. 2 DC characteristics (3.3V)

(VSS=0V , VDD=3.3V, Test temperature =25°C)

Label	parameters	Typical range				Test environment
		Minimum value	value	Maximum value	unit	
I_{PD}	Power-down mode, current ,	-	0.4	-	uA	
I_{WKT}	power-down, wake-up timer , low voltage	-	1.5	-	uA	
I_{LVD}	detection module, power consumption	-	10	-	uA	
I_{CMP}	Comparator power consumption		90	-	uA	
I_{IDL}	idle mode current (internal 32KHz)	-	0.48	-	mA	Equivalent to tradition, 8051 0.5M
	Idle mode current (6MHz)	-	0.88	-	mA	Of the equivalent to tradition, 8051 79M
	Idle mode current (Idle 12MHz)	-	1.00	-	mA	equivalent to tradition, 8051 158M equivalent to tradition
	mode current (normal 24MHz)	-	1.16	-	mA	, equivalent to tradition 8051 317M
I_{NOR}	mode current (internal 32KHz)	-	0.48	-	mA	, equivalent to tradition 8051 0.5M
	Normal mode current (500KHz)	-	0.88	-	mA	Equivalent to traditional [illegible]M 8051
	Normal mode current (600KHz)	-	0.88	-	mA	Equivalent to traditional [illegible]M 8051
	Normal mode current (700KHz)	-	0.90	-	mA	Equivalent to traditional [illegible]M 8051
	Normal mode current (800KHz)	-	0.91	-	mA	Equivalent to tradition , 8051 11M
	Normal mode current (900KHz)	-	0.91	-	mA	of , of, of, equivalent to tradition , 8051 12M
	Normal mode current (1MHz)	-	0.94	-	mA	equivalent to tradition , 8051 13M equivalent
	Normal mode current (2MHz)	-	1.05	-	mA	to tradition , equivalent 8051 26M of, of, of, of, of,
	Normal mode current (3MHz)	-	1.17	-	mA	to tradition, equivalent to tradition 8051 40M
	Normal mode current (4MHz)	-	1.26	-	mA	, equivalent to tradition 8051 53M
	Normal mode current (5MHz)	-	1.40	-	mA	, equivalent to tradition 8051 66M of, of, of 158M
	Normal mode current (6MHz)	-	1.49	-	mA	, equivalent to tradition 8051 79M
	Normal mode current (12MHz)	-	2.09	-	mA	, equivalent to tradition of 317M 8051
	Normal mode current (24MHz)	-	3.16	-	mA	Equivalent to traditional 8051
V_{IL1}	Input low level	-	-	0.99	V	Turn on Schmidt trigger
		-	-	1.07	V	Turn off Schmidt trigger
V_{IH1}	Input high level (normal IO)	1.18	-	-	V	Turn on Schmidt trigger
		1.09	-	-	V	Turn off Schmidt trigger
V_{IH2}	input high level (reset pin)	1.18	-	-	V	
I_{OL1}	Output low-level sink current , output	-	20	0.99	mA	Port voltage 0.45V
I_{OH1}	high-level current (bidirectional mode) , output	-	110	-	uA	
I_{OH2}	high-level current (push-pull mode)	-	20	-	mA	Port voltage 2.4V
I_{IL}	Logic 0 Input current	-	-	50	uA	Port voltage 0V
I_{TL}	to logic to 1 The transfer current 0	100	270	600	uA	Port voltage 2.0V
R_{PU}	I/O Port pull-up resistor	5.8	5.9	6.0	KΩ	
I/O speed	I/O High current drive , Fast conversion ,		25		MHz	PxDR=0, PxSR=0
	I/O I/O fast conversion ,		22		MHz	PxDR=1, PxSR=0
	I/O Low current drive , slow conversion		16		MHz	PxDR=0, PxSR=1
	I/O I/O , slow conversion		12		MHz	PxDR=1, PxSR=1
compare device	High current drive Fastest speed		10		MHz	Turn off all analog and digital filtering
	I/O analog filtering time		0.1		us	

Low current drive ,

I/O

	Digital filtering time , power-down mode power		0		System clock	LCDTY = 0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	consumption when the comparator is enabled	-	400	-	uA	
I_{PD3}	Enable LVD Power consumption in power-down mode		470	-	uA	

P. 3 DC characteristics (5.0V)

(VSS=0V , VDD=5.0V, Test temperature =25°C)

Label	parameters	Typical range				Test environment
		Minimum value	value	Maximum value	unit	
I_{PD}	Power-down mode, current ,	-	0.6	-	uA	
I_{WKT}	power-down, wake-up timer , low voltage		4.4	-	uA	
I_{LVD}	detection module, power consumption		30	-	uA	
I_{CMP}	Comparator power consumption		90	-	uA	
I_{IDL}	idle mode current (internal 32KHz)	-	0.58	-	mA	Equivalent to tradition 8051 0.5M
	Idle mode current (6MHz)	-	0.98	-	mA	Of the , equivalent to tradition 8051 ?M
	idle mode current (12MHz)	-	1.10	-	mA	Equivalent to tradition 8051 158M
	Idle mode current (24MHz)	-	1.25	-	mA	, equivalent to tradition 8051 317M
I_{NOR}	Normal mode current (internal 32KHz)	-	0.58	-	mA	, equivalent to tradition 8051 0.5M
	Normal mode current (500KHz)		0.97		mA	Equivalent to traditional 6.6M 8051
	Normal mode current (600KHz)		0.97		mA	Equivalent to traditional 7.9M 8051
	Normal mode current (700KHz)		1.00		mA	Equivalent to traditional 9.2M 8051
	Normal mode current (800KHz)		1.01		mA	Equivalent to tradition , 11M8051
	Normal mode current (900KHz)		1.01		mA	of , of, of, equivalent to tradition , 8051
	Normal mode current (1MHz)		1.03		mA	equivalent to tradition , equivalent 8051
	Normal mode current (2MHz)		1.15		mA	to tradition, equivalent to tradition, of, of, of, of, of, 8051
	Normal mode current (3MHz)		1.27		mA	, equivalent to tradition 8051
	Normal mode current (4MHz)		1.35		mA	, equivalent to tradition 8051
	Normal mode current (5MHz)		1.49		mA	of, of, of 158M , equivalent to tradition 8051
	Normal mode current (6MHz)	-	1.59	-	mA	, equivalent to tradition 8051
	Normal mode current (12MHz)	-	2.19	-	mA	Equivalent to tradition 8051 of 317M
	Normal mode current (24MHz)	-	3.27	-	mA	, equivalent to tradition 8051
V_{IL1}	Input low level	-	-	1.32	V	Turn on Schmidt trigger Turn
		-	-	1.48	V	off Schmidt trigger Turn on
V_{IH1}	Input high level (normal IO)	1.60	-	-	V	Schmidt trigger Turn off Schmidt
		1.54	-	-	V	trigger 12M 13M 26M 40M 53M 66M 79M
V_{IH2}	input high level (reset pin)	1.60	-	1.32	V	
I_{OL1}	Output low-level sink current , output		20	-	mA	Port voltage 0.45V
I_{OH1}	high-level current (bidirectional mode) , output		200	-	uA	
I_{OH2}	high-level current (push-pull mode)	-	20	-	mA	Port voltage 2.4V
I_{IL}	Logic 0 Input current	-	-	50	uA	Port voltage 0V
I_{TL}	to logic to 1 The transfer current 0	100	270	600	uA	Port voltage 2.0V
R_{PU}	I/O Port pull-up resistor	4.1	4.2	4.4	KΩ	
I/O speed	I/O High current drive , Fast conversion ,		36		MHz	PxDR=0, PxSR=0
	I/O I/O fast conversion ,		32		MHz	PxDR=1, PxSR=0
	I/O Low current drive , slow conversion		26		MHz	PxDR=0, PxSR=1
	I/O I/O , slow conversion		22		MHz	PxDR=1, PxSR=1
compare device	High current drive Fastest speed		10		MHz	Turn off all analog and digital filtering
	I/O analog filtering time		0.1		us	

Low current drive ,

I/O

	Digital filtering time , power-down mode power consumption when the comparator is enabled		0		System clock	LCDTY = 0
			n+2			LCDTY=n (n=1~63)
I_{PD2}		-	460	-	uA	
I_{PD3}	Enable LVD Power consumption in power-down mode	-	520	-	uA	

P. 4 Port drive capability (corresponding to the drive current I/O Voltage on I/O)

(VSS=0V , VDD=5.0V, Test temperature =25°C)

Normal I/O push-pull output 1		
	Normal thrust	Strong
10mA	4.50V	thrust
20mA	4.00V	4.72V 4.49 V
30mA	3.40V	4.24 V
40mA	2.31V	3.96 V
50mA	-	3.65 V
60mA	-	3.25 V
70mA	-	2.75 V
80mA	-	1.65 V

Normal I/O Push-pull output 0/quasi-bidirectional port output 0/Open-drain mode 0		
	General thrust	Strong
10mA	0.34V	thrust
20mA	0.66V	0.20V 0.35V
30mA	1.04V	0.52 V
40mA	1.70V	0.68 V
50mA	-	0.88 V
60mA	-	1.14 V
70mA	-	1.44 V
80mA	-	1.93V

P. 5 inside IRC Temperature drift characteristics (reference temperature 25°C)

temperature	Typical range value		
	Minimum value		Maximum value
-40°C ~ 85°C		-1.38% ~ +1.42%	
-20°C ~ 65°C		-0.88% ~ +1.05%	

P. 6 Low voltage reset threshold voltage (test temperature 25°C)

level	voltage

	Minimum value	Typical value	Maximum value
POR		(measured value) (1.69V~1.82V)	
LVR0		2.0V (1.88V~1.99V)	
LVR1		2.4V (2.28V~2.45V)	
LVR2		2.7V (2.58V~2.76V)	
LVR3		3.0V (2.86V~3.06V)	

Q Appendix Application precautions

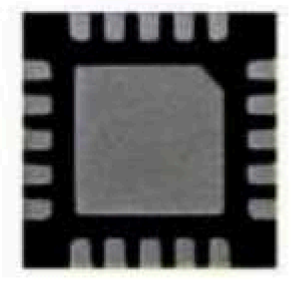
Q. 1 about STC12H series IO Precautions for mouth

1. STC12H Series of chips IO Mouth, except ISP Download port P3.0 and P3.1 Outside, the rest IO The initial after the port is powered o
The modes are all high-impedance input modes, and the user cannot directly output the level, so the user must use two
registers to and initialize the corresponding mode at the place where the program is initialized in order to use it normally. PxM0 PxM1
2. STC12H All series of chips I/O The port can be set as the two-way port mode, strong push-pull output mode, and open-d
I/O Pull-up resistors can be independently enabled internally
Or high impedance input mode. in addition to each I/O
3. STC12H Series chips will not be automatically special Mouth mode, such as ADC Port, serial port、Mouth and SPI
Port, the user must set the corresponding port to the appropriate mode I2C
4. by himself. If enabled P5.4 If the pin is a reset pin, the reset level is low
5. , pay special attention: Due to all the series STC12H I/O (Except ISP Outside) After power-up, it is a high down
Resistance input mode , Go directly to power-down mode/Shutdown mode, which will c
I/O The external level is not fixed at this time, if power-down mode/Before the shutdown mode, there is additional power consumption. The mouth is based on the rea
The actual situation is set up I/O The mode of the mouth, for all unused external floating ones All I/O need to be set as a two-way port ,
And fix the output high level. Especially for those 28 The pin of the chip, because there are some pins and ports inside the
who are not wired to the external pins, so these are I/O It is also in a floating state, and this part also needs to be set to prevail. I/O
to the port and the output is fixed at a high level.
6. STC12H series A Version chip I/O The port is interrupted. After testing, it is found that there is a problem. Please do not

R Appendix QFN/DFN Welding method of packaged components

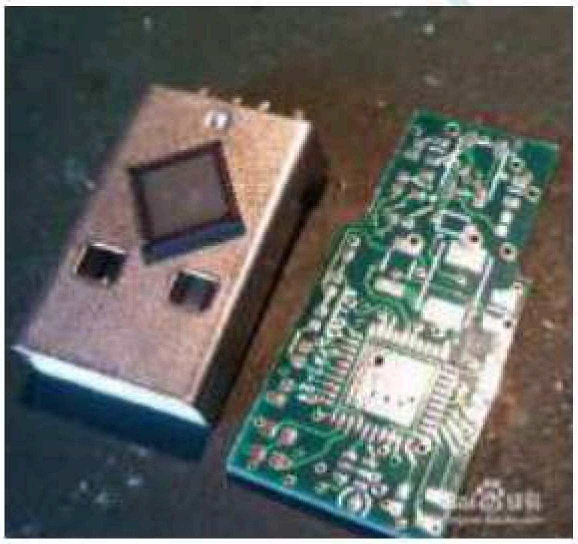
STC In the packaging form of the product, the more popular ones have been added QFN and DFN Encapsulation. Due to this

The pins of the chip in the form of a package are at the bottom of the chip, and it is difficult to weld manually. There are small companies on the market that specialize in welding engineering samples, which can undertake engineerin If the user needs to weld by himself, please refer to the welding method below.

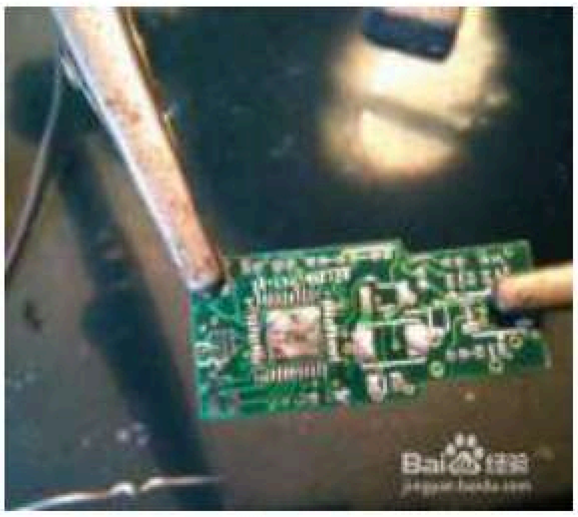


1, First of all, you need to prepare the following tools: electric soldering iron, hot air gun, tweezers, fixing

2、 frame and other tools . The plates and chips that need to be welded are as followsPicture below: PCB

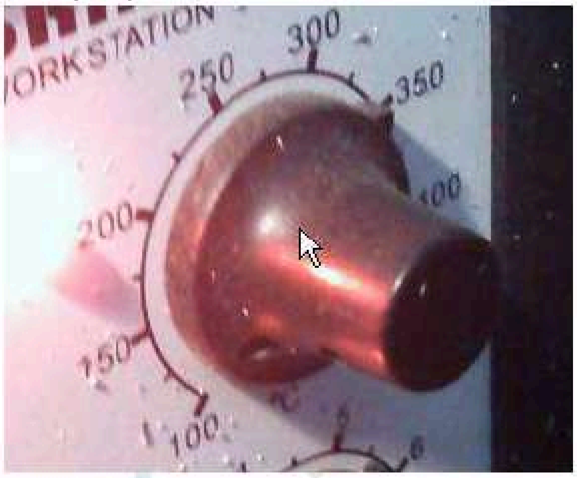


3、 First tin the pad of the chip on the board :

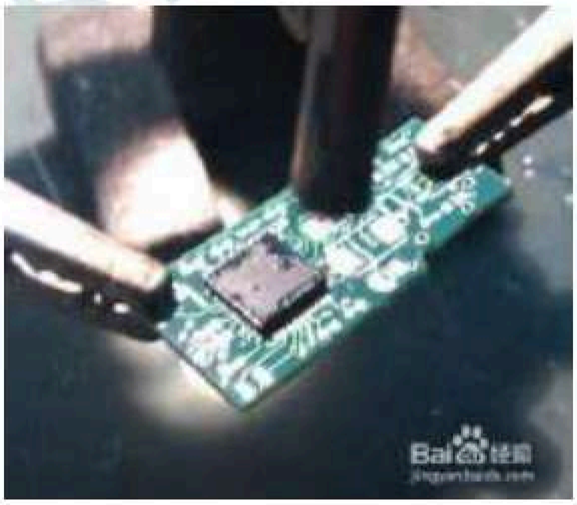


4、 Then put tin on the bottom of the chip. After this is finished, flatten it to minimize the tin, but not without it.

5、 Adjust the temperature of the hot air gun, and the actual 240 About the degree, because the quality of the air gun is not the same, acc
air outlet is probably adjusted in the actual situation.

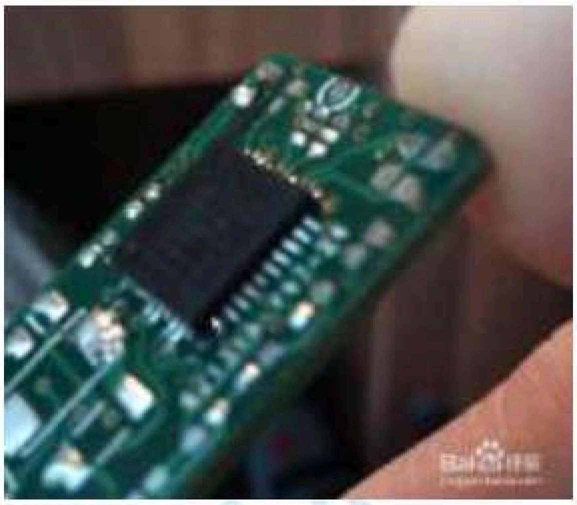


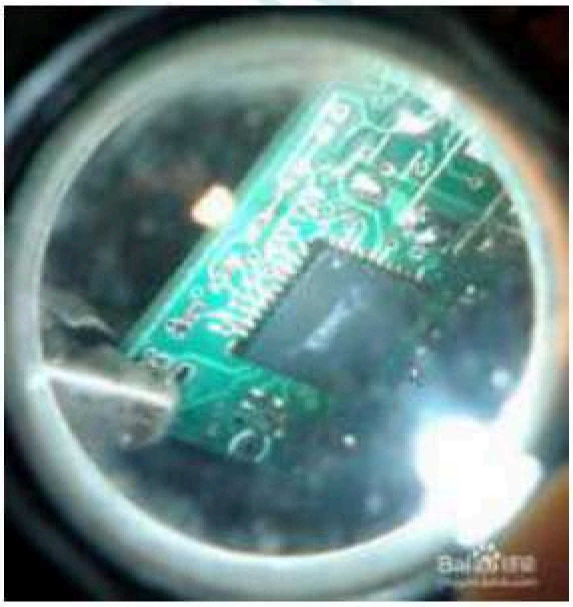
6、 Put the chip on the pad, be sure to put it upright, and then blow it with a hot air gun at an even speed until the tin melts, generally within seconds. 20



7、 Tin the chip side pins with a soldering iron

8、 The effect after the welding is completed





About whether to bake before reflow soldering S Appendix

According to the international moisture sensitivity level (3 MSL3) According to the requirements of the specification, after the SMD components are disa With in hours,

7 Within days, the reflow soldering patch must be 168 completed, and if it is not completed, it must be baked at high temperature again.

Plastic pipe can't withstand 100 a high temperature of more than degrees, after unpacking the vacuum packaging 7 Reflow soldering must be completed within days ,

Otherwise, it will not be able to be removed before reflow soldering. 100 Put the plastic tube with a high temperature of more than one degree 110~125°C in a metal tray and bake it aga 4~8

You can do it in an hour

LQFP/QFN/DFN Pallet capacity 100 At a high temperature of more than degrees, after unpacking the vacuum packaging 7 Reflow soldering must be completed within days ,

Otherwise, it must be re-baked before 110~125°C 4~8 reflow soldering : You can do it in an hour

T Appendix How to use a multimeter to detect the chip I/O Good or bad mouth

According to the international moisture sensitivity level (the requirements of the specification, after the SMD components are disa) According to the requirements of the specification, after the 3 MSL3 When the unpacking,

7 Within days, the reflow soldering patch must be completed, and if it is not completed, it must be baked at high temperature again. If t 168
Reflow soldering, the metal wire inside the chip may be pulled off due to uneven heating inside and outside the chip, and the final phen
Mouth damage.

STC When the microcontroller is designed on the chip, each I/O There are mouths Separately to The protection diode, with Wanhe VCC GND
The measurement can be made with the diode monitoring file of the meter. You can use this I/O The quality of the pins. Use a multimeter to measure
method to make a simple judgment as follows (note: A digital multimeter is used here)

First, adjust the multimeter to the diode detection gear, the chip under test does not need to be powered, and the multimeter's Red watch pen
under test pin, The black meter pen measures each port in turn. If the parameters displayed by the multimeter are left and right, it means
to I/O GND The protection diode is normal, that is, the wire is intact, if the displayed 0V parameters are the wiring inside the chip
Has been pulled off.

The above method is a method of detecting the wiring situation inside the chip.

In addition, if there is no protection circuit on the pins of the MICROCONTROLLER on the user board, overcurrent or overvoltage
Burned out. In order to detect whether the pin is burned out, in addition to using the above method to I/O Mouth to mouth In addition to the protection diode, you also nee
The protection diode detected by the detection port. The method of using the multimeter to detect the protection diode from the port is

First, adjust the multimeter to the diode detection gear, the chip under test should not be powered, and connect the black meter pe
VCC chip under test. pin,
I/O Red watch pen Measure each port in turn. If the parameters displayed by the multimeter are left and right, it means that the inside of the
0.7V The protection diode received is normal. If the displayed parameter is, it means that this port of the chip has been damaged. VCC

Mass production, how to eliminate the need for dedicate
How to have no burning link

Mass production, you will be produced by STC of MCU Before the control board as the main control chip is assembled into the device, you

MCU After the patch is completed to your control board, you must test the quality of your control board. Don't say 100% If there is no problem, then

It is to raise the bar, not to engage in production. As long as the production is carried out, there will be false welding, short circuit, misla

So after the patch comes back, you have to test it before assembling it into STC MCU The quality of the control board ,

the shell . You have to assemble the good ones and repair and rescue the bad ones.

Testing, mass production, there must be a test stand/Connect to our U8W/U8W-Mini/STC-USB

Link1 offline burning tool below , and also connect to other control parts

Pass USER-VCC、 P3.0， P3.1、 GND Connect, ask the workers to turn on the power

by S-VCC、 P3.0、 P3.1、 GND supply every time, don't you turn on the power supply The offline tool automatically powers you

STC

The cost of making a test stand for you outside 500 Below Yuan, there are plexiglass, clamps, and thimbles.

1 A worker management that tests whether your control board is normal 2~3 A test stand

Operation flow：

1, Will your The control board is stuck to the test stand 1 On STC

2, Will your MCU STC MCU The control board is stuck to the test stand 2 On, the test stand The program on has been burned/Can't feel the burn

Recording time

3, Test test stand 1 On the STC Whether the function of the main control board is normal, it is normally placed in the norma

district

4, To the test stand 1 A new untested and unplanned control board on the card

、5 Test the untested control board on the test stand 2 / I don't know when the program was burned

out unknowingly, and the new one has not been tested.

Burning control board 2

、6 Cycle steps to step No need to arrange burning personnel

Appendix V about Keil in software 0xFD Description of the problem

As we all know , Keil Software and 8051 All versions of the compiler have one called The problem is mainly manifested in the wor

on this issue , 0xFDKeil The string cannot contain a band Coded Chinese characters 80251 Keil The software will skip at compile time And garbled code appears.

Otherwise the official response 0xfd this 0xfe The character encoding is For internal use by the compiler, so

If included in the code 0xfd is: when the string , 0xfd Will be automatically skipped by the compiler 3。

The official solution: with Keil Add one after the encoded Chinese characters That is: For example :

printf ("mathematics"); //Keil will display garbled code when printing after

printf ("number")\xfd Learn"); compilation //The display is normal

The "\xfd" here is standard C The escape character in the code, "\x" means 1~2 The characters are hexadecimal numbers "\xfd" make

Show will 16 Hexadecimal 0xfd number the subsequent insertion into the string.

Since the Chinese character encoding of "number" is 0xCAFD Keil At compile time, FD will be added Compile into the target file, then skip, and only

manually supplemented by escape characters, one more is required. 0xfd to the target file to form a complete 0xCAFD add CA , so that it can be displayed normally.

Find the There are many patches on the Internet, basically only for The software is effective. The method of patching is in the executable

key code in 0xFD the old version of [80 FB FD], and modified to [80 FB FF] . The key code found by this modification method is too simple and easy to modify.

To other unrelated places, resulting in inexplicable problems when the compiled object file is running.

Therefore, when the string in the code contains the problem Chinese characters, it is recommended to use Keil The solution provided by the official will be resolved

GB2312 In, including 0xfd Coded Chinese The words are as follows :

In addition to waiting for the spy, Er captured Gengguo, accumulated arrows, embers,

Junkui , the cage, slowly condensing the pipa, driving three liters of water,

she listened to the delusion, the demon, the introduction of Yu ha Zhengzhu Lost there A because B Pi Luo nest 摒 spit

ze Xian Kai obliterate upon brood over WAN Cong Qian Surin

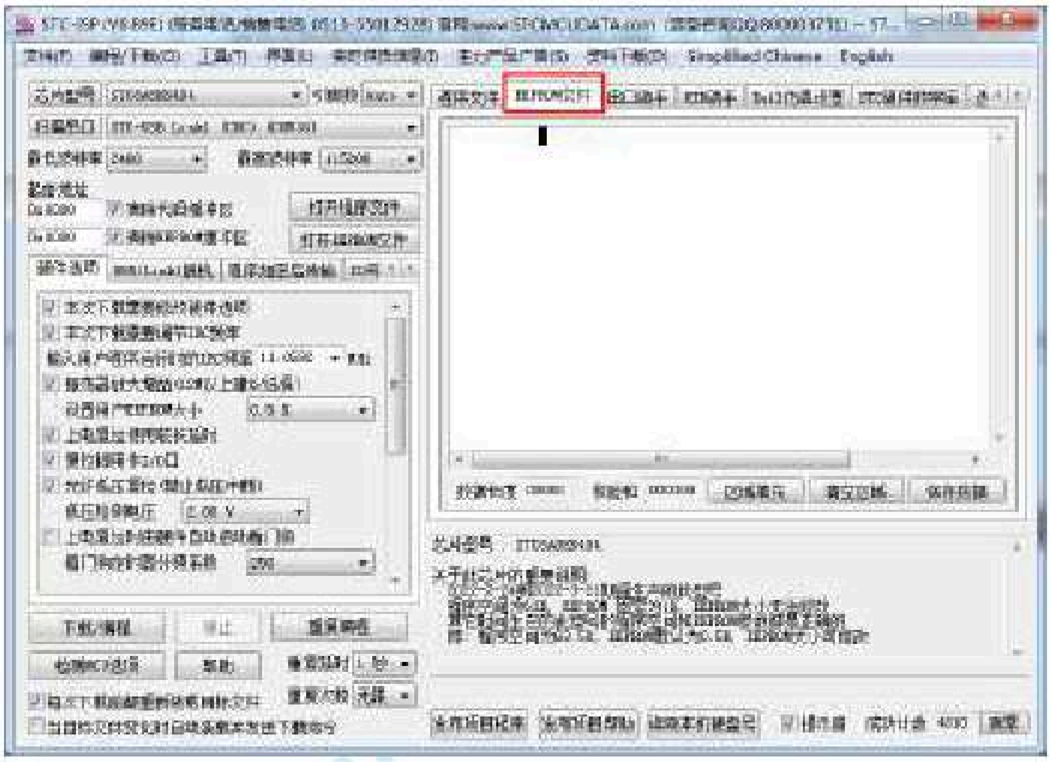
San bashfulness nervous Jair Quan wheat Jia tall mites 齘 elixirs

Goblet of bream snoring In addition , Keil The characters of the project path name cannot contain 0xFD Coded Chinese characters, otherwise Keil The software will not compile this correctly

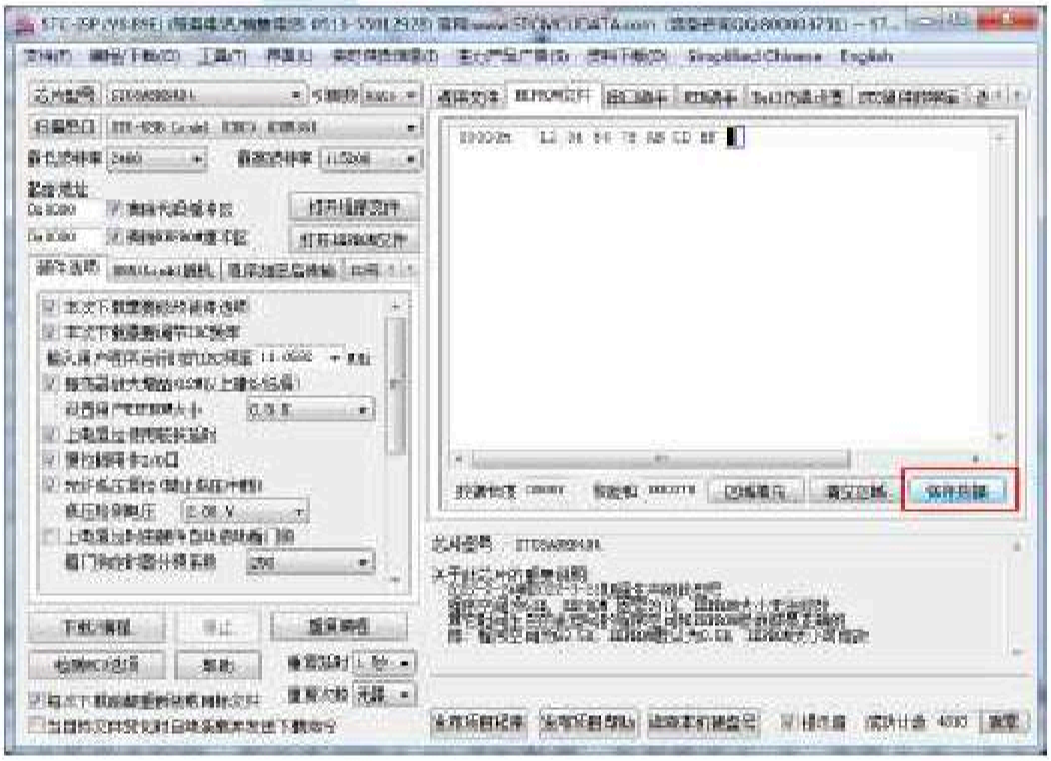
project.

W Appendix How to use STC-ISP Download software production and editing EEPROM file

Open any version STC-ISP Download the software and select "EEPROM" Page, click the data window, as shown below



When a black rectangular cursor appears, you can enter Hexadecimal data, including numbers and letters A~F (Universal case)
After the data input is complete, click the "Save Data" button to save EEPROM data



0000h: 12 34 56 78 AB CD EF

X Appendix Can the MCU provide bare core?

Q : Can the single chip microcomputer provide bare core?

A : Bare core is not available for the time being. If you need a small chip area, you can use it DFN8 QFN20 QFN32 QFN48 And other small volumes package

Y Appendix STC12H Replaced by a series of microcontrollers STC12C56/54 Series of precautions

Single chip microcomputer instructions

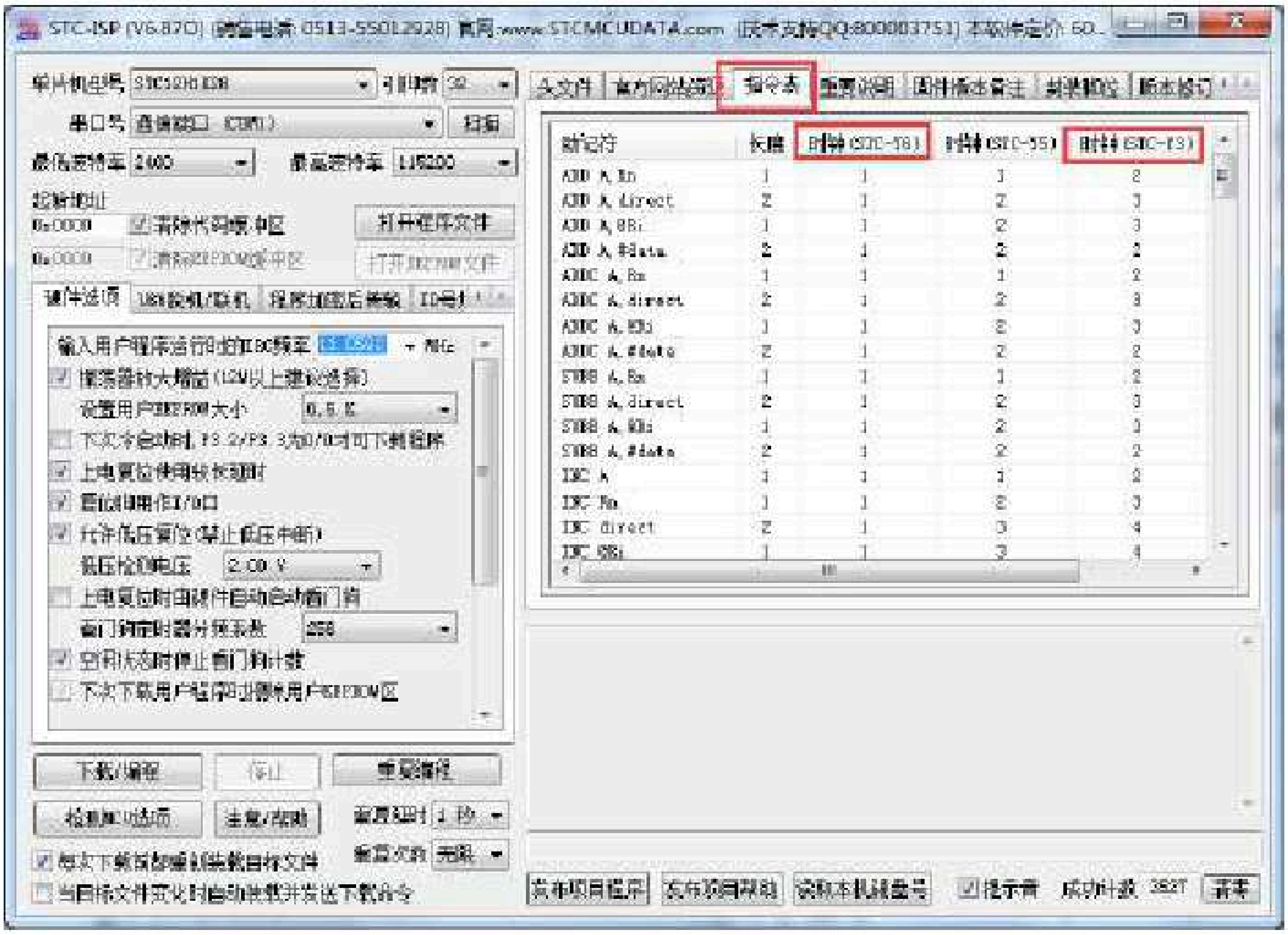
Series of instruction codes and STC12H STC12C56/54 The series is exactly the same, so STC12C56/54 Series of code shifts

Planted to STC12H On, the operation is still correct, but STC12H STC12H The command speed ratio of the series STC12C56/54 The series should be fast ,

STC12C56/54 The command system of the series belongs to Series of instructions, and STC-Y3 STC12H The instructions of the series belong to Series refers to STC-Y6

order , STC-Y6 Most of the instructions in the series only need one to execute CPU clock. If there is an instruction delay code in the user code,

You need to make adjustments. There is a comparison of each instruction for reference STC download the instruction table of the software, as shown in the figure below



助记符	长度	时钟(STC-Y6)	时钟(STC-Y5)	时钟(STC-Y3)
ADD A,Rn	1	1	1	2
ADD A,direct	2	1	2	3
ADD A,@Ri	1	1	2	3
ADD A,#data	2	1	2	2
ADDC A,Rn	1	1	1	2
ADDC A,direct	2	1	2	3
ADDC A,@Ri	1	1	2	3
ADDC A,#data	2	1	2	2
SUBB A,Rn	1	1	1	2
SUBB A,direct	2	1	2	3
SUBB A,@Ri	1	1	2	3
SUBB A,#data	2	1	2	2
INC A	1	1	1	2
INC Rn	1	1	2	3
INC direct	2	1	3	4
INC @Ri	1	1	3	4

Operating voltage

STC12H The series is a wide-voltage chip, and the operating voltage 1.9V ~ 5.5V STC12C56/54 1.9V Series and 5V

series , 3V range is the operating voltage range of the chip. ~ 2.4V The operating voltage range of the chip is series are divided into 3.5V

~ 3.6V , 5V ~ 5.5V3V。

Low voltage detection

STC12H Inside the series The low voltage detection is 4 STC12C56/54 Two levels of series bits are optional.

System clock source

STC12H The series system clock can be internal, high-speed and high-precision IRC, dynamically configure the software, External crystal oscillator and internal STC12C56/54 the series can only be downloaded when the hardware selects the internal IRC Oscillator or external crystal oscillator.

Internal expansion RAM

STC12H Series internal expansion RAM for 1K byte, STC12C56/54 Series internal expansion RAM for 512 byte

I/O mouth

STC12H After the series of microcontrollers are powered on, The pattern and STC12C56/54 The series is different. Series list STC12C56/54 STC12H I/O STC12C56/54 The series is different. STC12C56/54 Series of single chip microcompu have I/O After the series of microcontrollers are powered on Quasi-two-way port mode, and STC12H 8051 Series of microcontrollers I/O In addition to ISP Download foot P3.0/P3.1 I/O The Except for the two-way port mode, all the rest I/O The port is in high impedance input mode after power-up. 8051 And traditional STC12C56/54 mode and port are both after power-on After the series of microcontrollers are powered on, they are in quasi-bidirectional port mode and I/O output a high level. T Motor or Light, so there will be a moment when the microcontroller is powered on, LED Will flash, motor will move or system column I/O After power-up, it is a high-impedance input mode, which can avoid this kind of malfunction of the motor and the motor. LED

Due to STC12H Series of microcontrollers I/O In addition to ISP Download foot P3.0/P3.1 Except for the two-way port mode, the remaining por The ports are all high-impedance input modes after power-up, so when the user needs to STC12H of the series must output signals to the outside world. I/O You must first use the operating mode of the two register pairs to set it. PxM0 PxM1 I/O

Reset foot

series STC12H The mouth is generally regarded as ordinary mouth, when the user is Used by the reset pin. Set up when ISP downloading P5.4 When it is the reset pin function I/O The port is the reset pin of the microcontroller (RESET), at this time When the reset pin is low, the microcontroller is in foot) P5.4 Reset state, when the mcu is high, the reset state is released by the mcu. STC12C56/54 Series, when the reset pin is high, the microcontroller is in the reset state, the microcontroller releases the reset state when the level is low.

So when the user enables P5.4 The reset pin function of the port requires attention to the reset level.

EEPROM

STC12H Series of EEPROM Related special function registers and STC12C56/54 Incompatible. In addition, erasure and programming, The waiting time setting is also different. Use register STC12C56/54 of IAP_CONTR B2-B0 Setting, setting is just a general Frequency range value, A new register has been added to the series IAP_TPS (SFR address: 0F5H), dedicated to setting up EEPROM STC12H Waiting time for erasure and programming, And the user does not need to be based on The current working frequency, fill in directly MCU IAP_TPS need to calculate, that is, the hardware will automatically calculate CPU The operating frequency is 24MHz you only need to IAP_TPS the waiting time. (For example: just fill in the current one) 24

Timer

STC12H The series has 5 a 16 Bit timer, STC12C56/54 There are only two timers.

STC12H The series of timer 0 and timer modes are 0 1 16 Bit automatic reload mode, and STC12C56/54 series The timer 0 And the mode of the timer is the bit does not automatically reload mode. 1 0 13

STC12H Is an unshielded interrupt 16 Bit automatic reload mode, and STC12C56/54 Series of timer 0 The pattern of 3 The timer of the mode series is dual 03 Bit timer mode.

serial port

The series has two serial ports , STC12H only STC12C56/54 one STC12H The highest baud rate of all serial ports in the series is a

Reach the system frequency /4 , STC12C56/54 The fastest system frequency can only be clocked /16。

ADC

Series and STC12C56/54 STC12H Series of ADC_CONTR、 ADC_RESL3 ADC_RES, Register address

Incompatible. STC12H Two new registers have been added to the series : and ADCTIM。

ADCCFG

STC12C56/54 series start ADC Conversion ADC_START Located in the register ADC_CONTR of BIT3, and

STC12H The series is located in ADC_CONTR bit of BIT6

STC12C56/54 Conversion completion flag ADC_FLAG Located in the register ADC_CONTR of BIT4, and

STC12H

The series is located in ADC ADC _CONTR of BIT5

STC12C56/54 The speed control is ADC_SPEED Located in the register ADC_CONTR of BIT6-BIT5, and

STC12H series ADC

The series is located in ADCCFG of BIT3-BIT0

STC12C56/54 Alignment control ADC Alignment control bit of the conversion result Located in the register CLK_DIV of BIT5, and

STC12H bits of series series RESFMT Located in ADCCFG of BIT5

STC12H A more accurate one has been added to the series ADC Conversion timing control mechanism, through the register ADCTIM Set up

PCA/PWM/CCP

STC12H The group of the PCA/PWM/CCP The first 0~2 The address of the special function register of the group module is not the STC12C56/54

Content, but the first series is not compatible , The first 3 and the group register are defined in the area, and The first 3 SFR STC12H Group hosting

The device is defined in area XFR

STC12H Series of PWM Mode can output bits 6 bit /7 /8 bit /10 bit PWM , STC12C56/54 Series can only be fixed output

Out 8 bit.

SPI

STC12H Series of SPI The address and speed control of Incompatible.

Series of STC12H SPI the relevant special function registers /4 /8 /16、 STC12C56/54 STC12C56/54 Series of SPI speed

The control is the system clock /4、 are the system clock, , /16、 /64、 /128

Watchdog

STC12H Series of watchdog-related special function register addresses and STC12C56/54 Incompatible.

Z Appendix Update record

2024/5/13

1. update USB The price of a
2. increase dual-serial chip STC-USB Link1D The production steps of the tool master chip

2024/4/30

1. update STC12H1K08 Series selection price
2. update STC12H1K08 list series pin diagram SOP16

2024/2/2

1. Add interrupt response instructions , add
2. QR code for small shopping malls
3. Add the chip series name to each pin diagramcalled
4. Organize the structure of the document so that the register description is displayed
5. on the same page as much as possible. Correct the clerical errors in the document
6. and add a description of the process of power-up in the clock chapter. MCU

2023/11/24

1. The cover of the manual adds the QR code of the small mall
2. , and the catalog title is added to each package of the MCU
3. series . Update the tool picture in each package picture
4. . Update the description based on precautions.
5. I/O
6. Added to the introduction chapter of dual serial port chip USB Go to dual serial port tool simulation and download schematic diagram
7. USB
8. Increase the use of tools and STC-USB Link1D
9. Update simulation download instructions USB
10. Add a tool instruction chapter for killing two birds with one stone, add a compiler introduction and project settings chapter , and a more selected price list

2023/9/12

1. Add MCU overview chapter
2. STC-ISP ,add download flow chart
A schematic diagram of the download line is added to the minimum system pin diagram 3.

2023/1/10

Add forum link 1.

2. Update reference circuit diagram description

2022/12/23

1. update I2C Slave code sample program (improve code compatibility)

2022/11/14

1. ISP Download the capacitor in the reference circuit diagram and it is recommended to use it uniformly 104) combination

2022/10/31

1. Increase the use and increase STC-USB Link1D Download the reference circuit diagram tool to proceed ISP
2. the software simulation USB Proceed ISP Downloaded reference circuit diagram

2022/9/21

1. Fixed typos in the comparator
2. chapter , updated the official website ,
3. updated the selection price list, added
4. advanced application chapters STC-ISP

Added the chapter on mandatory digital signature instructions for turning off the driver 5.

2022/3/9

1. Correct clerical errors in the document
2. Update the list of special function registers Series only STC12H P1IE and P2IE)
3. Update application precautions regarding port interruption I/O

2021/12/17

1. Correct timer 2/3/4 The timing calculation formula

2021/10/29

1. Correct typos in the document
2. and add pin diagrams SOP16

2021/8/26

1. correction ADC Annotation error in the chapter sample program

2021/6/26

1. Fixed the wrong name of the crystal oscillator pin port

2021/5/10

1. increase ADC Power switch delay description
2. Increase use The first ADC 15 Description of the principle and calculation formula of the input voltage of the external channel of the cha
3. Modify the maximum available for some series Error description of size FLASH
4. Increase timer 2/3/4 Description of the interrupt flag

2021/2/26

1. Add relevant simulation USB Download instructions
2. Add timer, timer 3, timer 4 Bit clock prescaler register description 3 8

2021/2/4

Reset the initial value of the register

CLKDIV correction 1.

Add a description of the initial value of the special function register

3. 2. Add the application reference circuit diagram under the pin diagram

2020/11/25

1. Correct errors in some sample programs
2. , update the sample price list, correct
3. the description error in the document

2020/10/16

Reference price of series of microcontrollers

STC12H update 1.

Description of increasing the load capacitance of the external crystal oscillator circuit 2.

2020/9/23

1. create STC12H Series of single chip microcomputer technical reference manual document

Standard sales contract for this series of products

one. Product quality standards: the goods are brand new and authentic ROHS Quality standards.with

Two. Supplier responsibility: In the case of quality problems of the supplier, after confirmation by both parties, the buyer will return the chip,
for each replacement, and the warranty will be one year. Three. Buyer's responsibility :

A , Acceptance: At the time of express delivery, the buyer confirms that the quantity is correct, no chips are scattered, no pins are defor
Then sign for it. If there is an abnormal buyer who cannot sign for it, the courier company shall bear the responsibility. Once the buye
acknowledges that the supplier has completed the order as required, and there is no longer any other joint and several liability. , Stor

B to the international humidity sensitivity (3 MSL3) According to the requirements of the specification, after the SMD components are disa

168 Within hours , LQFP/QFN/DFN Pallet capacity 150 At a high temperature of more than degrees,
7 Within days, the reflow soldering patch must be completed
The reflow soldering patch must be completed within a few days after the empty packaging. If it is not completed, it must be re-bak 4~8 110~125 °C, reflow sold
must be done within days
soldering: The plastic tube can withstand high temperatures above zero for hours. After disassembling the vacuum packaging, the 7
100 The patch is completed, otherwise it will not be able to be removed before reflow soldering. Put the plastic tube with a high temperature 100 of more than one degree in a meta
110~125°C , 4~8 You can do it in an hour

Since the goods returned by customers often contain products of unknown origin, and the original SMD device is disassembled and va
Complete the reflow soldering patch process within hours and days.
Our company has no production capacity to re-test the returned devices in detail and then re-bake them. We are unable to evaluate the
returned by customers . In order to ensure the interests of all customers, once the products are shipped out of the warehouse, they wil
to ensure quality and ensure the safety of all customers..

Four. Dispute resolution method: If this contract is not exhaustive or there is a dispute, the two parties shall negotiate and resolve it. If the n
supplier's location. V. Other terms: The contract is in duplicate. It will take effect from the signing of both parties. If the supplier is unable to
supplier shall promptly notify the buyer and renegotiate the relevant matters of this contract, and the buyer shall be exempted from the oblig
included in this contract may be included in detail in the attachment to the contract.

Six. This contract can only take effect after the representatives of both parties sign it and the payment is received.

Remarks: In case of special circumstances, the model purchased by the buyer must be replaced with other models, and the supplier also ag
Hourly high temperature 100 baking Yuan , once Turn on 13
2, Boot test RMB500 Once , +0.2 / Meta film

Product authorization letter

To: Jiangsu Guoxin Technology Co., Ltd.

The intellectual property rights of STC12H series products belong to Shenzhen Guoxin Artif Intelligence Co., Ltd . Jiangsu Guoxin Technology Co., Ltd. is now authorized to engage in the pro and sales of STC12H series products in China.

Authorized unit:深圳国芯人工智能有限公司

Authorization time limit：
October 24, 2019-December 31, 2024

Independent property rights, controllable production

Shenzhen Guoxin Artificial Intelligence Co., Ltd. is a wholly-owned enterprise in the mainland of the
of China and operates independently in accordance with Chinese laws and regulations. Its registered add
No. 1 Qianwan 1st Road, Qianhai Shenzhen-Hong Kong Cooperation Zone , Shenzhen.

The devices described in this manual are independently developed in China and have independent intellectual property rig

Product coreR&D is in China, with all design capabilities such as chip design, packaging desi
reliability design, device simulation, and process simulation; the core R&D team members and lea
are all composed of personnel in China, including the leader of the R&D team. The R&D experience
long-term, long-term, long-term, long-term, long-term, long-term, long-term,Stable follow-up suppo

Copyright, etc.

Wafer manufacturing: The wafer manufacturing and processing after the design of this device is completed, in the mainlan

The processing and manufacturing of the domestic FAB is completed, and it is subject to the mana
control of the laws and regulations of the People's Republic of China, and it is completely controlla

Packaging manufacturing: The packaging manufacturing of this device after the design is completed is in the mainland of

The processing of the packaging factory is completed, and it is subject to the management, supervision and control of the laws a

Test: The test after the design of this device is completed, the test will be completed in the mainland of the People's Repub

Subject to the management, supervision and control of the laws and regulations of the People's Republic of China, it is complete

All the key processes of this device are completed on our own production line, and it can be
supplied for a long time without the trouble of being cut off.

Hereby explain。

深圳国芯人工智能有限公司