Shenzhen Guoxin Artificial Intelligence Co., Ltd.

Car gauge AEC-Q100 Grade1 MCU　　Design company

Series of microcontrollers

Technical reference manual

Technical support website：www.STCAI.com

Official technical Forum：www.STCAIMCU.com

Data update date：2024/5/13

Remarks and tags can be added directly to this document,

Authorized commercial city

Table of contents

1 Basic overview of single chip microcomputer

--For users without microcomputer principles, please start learning from this chapter.

The main contents of this chapter are: ①the basic knowledge of arithmetic operations in digital equipment--Number system and coding; ②new , some commonly used logical operations and their graphic symbols in digital circuits. They are the basis for learning the MCU course. For users and do not have the foundation of microcomputer principles , please start learning from this chapter.

1.1 Number system and coding

The number system is a scientific method by which people use symbols to count.

There are many kinds of number systems, and the commonly used number systems are: binary, decimal and hexadecimal.

The carry counting system divides the number into different digits, accumulates them bit by bit, and then adds them to a certain numbe the same time. The carry counting system has three elements: digital symbols, carry laws, and counting cardinality. The following table is a g

Commonly used number system	Representation symbol	Digital symbols	The law of the binary	Counting cardinality
Binary	B	0, 1	system is one in two,	2
decimal	D	0、1、2、3、4、5、6、7、8、9	one in ten, one in ten	10
hexadecimal	H	0、1、2、3、4、5、6、7、8、9、 , B CA,, D E,, F	Every sixteen in one	16

Decimal is generally used for counting in our daily lives. Binary is used in computers because binary has the advantages of simple oper easy to implement and reliable, which provides a favorable way for logic design and saves equipment. In order to distinguish it from other bi numbers, the writing of binary numbers is usually marked with the basic letter at the bottom right of the number, or followed by two possible digits, represents. Each digit in the binary number has only B 0 1 The counting base is. The addition and multiplication operations of binary numbers are as follows :

+ 0 = 0 + 1 = 10

0 × 0 = 0 0 × 1 = 1 × 0 =0 1 × 1 = 1

Because binary numbers are in use, the median is too long It is not easy to remember. In order to facilitate description, hexadecimal is c used as an abbreviation for binary. Hexadecimal is usually represented with a trailing flag or a subscript to show the difference. H 16

1.1.1 Number system conversion

Now let's introduce the conversion between these commonly used number systems. One: Binary to decimal conversion -

Method: Weight the binary number (As follows) Expand, and then add the values of each item to the decimal number to get the corresponding equivalent decimal number.

For example : , then N=(1101.101) What is the corresponding decimal number?

Expand by right N 2 N=1×2 +1×2 +0×2 +1×2 +1×2 +0×2 +1×2 =8+4+0+1+0.5+0+0.125 =13.625 D -3 ( )

Two: Decimal - Binary conversion

Method: It is divided into two parts, namely the integer part and the decimal part.

①Integer part conversion (Cardinality division) :

★ Divide the number we want to convert by the base of the binary (The binary cardinality is 2), Take the remainder as the lowest bit of binary;

★ to divide the quotient obtained last time by the binary cardinality (radix 2), take the remainder as the second lowest bit of binary ;

★ Continue to the previous step, Until the last quotient is zero, The remainder at this time is the highest bit of binary.

②conversion of fractional parts (Cardinality multiplication) :

Multiply the decimal part of the number to be converted by the base of the binary (2), Take the integer part as a binary decimal The highest position of the part ;

★ Multiply the decimal part obtained in the previous step by the binary cardinality (radix 2), Take the integer part as the second-highest part of the binary decimal pa

★ Continue to the previous step until the decimal part becomes zero. Or it is possible to meet the predetermined requirements.

For example: will $(213.8125)_{10}$ **Conversion to a binary number can be carried out as follows：**

First integer part：

2	213	remainder $=1=k_0$
2	106	remainder $=0=k_1$
2	53	remainder $=1=k_2$
2	26	remainder $=0=k_3$
2	13	remainder $=1=k_4$
2	6	remainder $=0=k_5$
2	3	remainder $=1=k_6$
2	1	remainder $=1=k_7$
	0	

So the integer part $(213)_{10}=(11010101)_2$

Regenerate the decimal part：

0.8125 × 2 = 1.6250 ---------- **Integer part** $=1=k_{-1}$

0.6250 × 2 = 1.2500 ---------- **Integer part** $=1=k_{-2}$

0.2500 × 2 = 0.5000 ---------- **Integer part** $=0=k_{-3}$

0.5000 × 2 = 1.0000 ---------- **Integer part** $=1=k_{-4}$

So the decimal part $(0.8125)_{10}=(0.1101)_2$

In summary, decimal numbers $213.8125=(11010101.1101)_2=(11010101.1101)B$

Three: Binary - Hexadecimal conversion

Method: Satisfy between binary and hexadecimal The relationship, so put the binary to be converted from the low to the high, high When the bit is insufficient, add "in front of the valid bit" 0 ", then convert each set of binary numbers into hexadecimal.

For example, the Convert to hexadecimal number : (0101110111110.10110010)B

(0101 1101 1110 . 1011 0010)B

↓ ↓ ↓ ↓ ↓

= (5 D E B 2)H

so , (0101110111110.10110010)B=(5DE. B2)H

Four: Hexadecimal - Binary conversion

Method: Hexadecimal conversion When it is binary, reverse the process of converting binary to hexadecimal above, that is, when converting, you only need to replace each bit of hexadecimal with the equivalent bit of binary.

For example: will (C1B. C6) Convert H to binary number :

1 (C CB. 6)H

↓ ↓ ↓ ↓ ↓

= (1100 0001 1011 1100 0110)B

so , (C1B. C6)H=(110000011011.11000110)B

Five: Hexadecimal - Decimal conversion

Method: Weight the hexadecimal number (As follows) Expand, and then add the values of each item to the decimal number to get the corresponding equivalent decimal number.

For example N=(2A. 7F)H, Then N What is the corresponding decimal number?

expand by right N=2×16 +10×16 +7×16 +15×16 =32+10+0.4375+0.05859375=(42.49609375)D

so , (2A. 7F)H =(42.49609375)D

Six: Decimal - Hexadecimal conversion

Method: When converting a decimal number to a hexadecimal number, you can first convert a decimal number to a binary number, and then convert the resulting binary number to an equivalent hexadecimal number.

1.1.2 Original code, reverse code and complement code

In life, The number is plus or minus, How do you represent the positive and negative symbols of numbers in a computer?

When expressing a number in life, it is generally to add a positive number in front of it."+", add one in front of the negative number"-", But the computer does not recognize these, usually adding a sign bit in front of the binary number. The symbol bit is "0 "Means"+", the symbol bit is "" to indicate"". This form of binary number is called Is its complement.

the original code. If the original code is positive, the inverse code and complement code of the original code are the same as the original code number, the original code is reversed bitwise except for the symbol bits. The resulting new binary number is called the inverse code of the ori

The summary of the three forms of original code, inverse code, and complement code is shown in the following table :

	True value	Original code	Reverse code	complement

Positive	+N	0N	0N	0N
negative	-N	1N	$(2^n-1)+N$	2^n+N

Example 1: Find the +18 and -18 eight-digit original code, inverse code, and complement forms.

True value	Original code	Reverse code	Complement
+18	00010010	00010010	00010010
-18	10010010	11101101	11101110

1.1.3 Common encoding

Specifying a certain set of binary numbers to represent a specified information is called encoding. One: Decimal encoding

Decimal numbers represented by binary codes are called decimal encodings. It has the form of binary and the characteristics of decima inter-representation of the connection between people and the number system. There are many kinds of decimal encodings, the most commo Code, also known as

Below we use a table to list several common decimal encodings:

Encoding type / Decimal number	8421 code (BCD code)	Remaining 3 yards	2421 yards	5211 yards	7321 yards
0	0000	0011	0000	0000	0000
1	0001	0100	0001	0001	0001
2	0010	0101	0010	0100	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0111	0101
5	0101	1000	1011	1000	0110
6	0110	1001	1100	1001	0111
7	0111	1010	1101	1100	1000
8	1000	1011	1110	1101	1001
9	1001	1100	1111	1111	1010
right	8421		2421	5211	7321

Decimal encodings are divided into authorized and unauthorized encodings. A binary code means that each decimal digit is represented binary codes, and each bit of the binary code has a fixed weight. Unencoded code means that there is no fixed weight for each bit in the binar

)code 、 code、 5211 code、 7321 The codes are all coded with the weight, and the authorized to be encoded.

BCD

(Code

is two: parity check code In the process of data access, operation and transmission, errors will inevitably occur. 1 "Wrong" 0 "Or put" 0 "Wrong" 1 "

。

Parity check code is a kind of code that can check for this kind of error. It is divided into two parts; information bits and parity bits. There are there are even numbers" 1 "It is called even check.

1.2 Several commonly used logical operations and their graphic symbols

The operations commonly used in logical algebra (NOT), and non- (AND), or (OR), non- (NOR), With or without (NAND), are: and (AND-NOR), XOR (EXCLUSIVE OR), The three (EXCLUSIVE NOR) etc. Which is related to or non- (AND), or (OR), non- (NOT) most basic operations in the same or operation.

One: And operation and gate and operation: The event only occurs when all the conditions that determine the result of the event are met at the same time.

And logical variables When performing and calculating, it can be written as : Y=A·B

Truth table		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

And gate: A unit circuit that implements logic operations. Graphic symbols with doors :

Two: Or operation and or door

Or operation: As long as any of the conditions that determine the result of the event are met, the event will occur. Logical variables and/or operations can be written as : Y=A+B A B

Truth table		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Or gate: A unit circuit that implements or performs logic operations. Or door graphic symbol :

Three: Non-arithmetic and non-gate

Non-operation: When the conditions are met, the event will not occur; when the conditions are not met, the event will only occur. When a logical variable performs a non-operation, it can be written as : A Y=A'

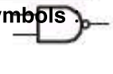
Truth table	
A	Y
0	1
1	0

Non-gate: A unit circuit that implements non-logical operations. Non-gate graphic symbols: Four: and non-Arithmetic and non-graphic symbols

Sum-non-operation: First perform the sum operation, then invert the result, and the final result is the sum-non-operation result. The sum of logical variables and non-operations can be written as : A B Y=(A·B)'

Truth table		
A		Y
0		1
0		1
1		1
1		0

With non-graphic symbols :



Five: Or non-arithmetic and or non-graphic symbols

Or non-operation: Perform the or operation first, then invert the result, and the final result is the or non-operation result. Logical variables and when performing or non-operations can be written as : A B Y=(A+B)'

Truth table

B	A		Y
0			1
1	0		0
0	1		0
1	1		0

Or non-graphic symbols :

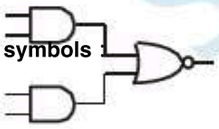
Six: and or non-operation and and or non-graphic symbols

and or non-operation: There are in and or non-logical operations 4 A logical variable, ,. Hypothesis A and B C For a group, and As a group , D A, between and 、 C D They are all related to each other, as long as 、 BA or Any group is at the same time , Y 0 That is. only Yes, when each set of inputs is not all, the output Y 1 That's it.

Logical variables A and B When performing or non-operation, it can be written as : Y=(A·

Truth table				
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

With or without graphic symbols :



Seven: XOR operation and XOR graphic symbol

Xor operation: When 、 for;1and when 、 Y 0 for. Logical variables A and B Make a difference BA At the same time, the output BA At the same time, the output Y or operation can be written as : Y = A ⊕ B = (A· B') + (A'· B)

Truth table			
B	A		Y
0	0		0
1	0		1
0	1		1
1	1		0

XOR graphic symbol :

Eight: The same or operation and the same or

graphic symbol the same or operation: when A, Different times, output Y 0 for; and when A, When the same, output Y 1 for. Logical variables A and B Proceed with the same

Or can be written as : $Y = A \odot B = (A \cdot B) + (A' \cdot B')$

Truth table		
A		Y
0		1
0		0
1		0
1		1

The same or graphic symbol :

1.3 STC12H MCU performance overview

STC12H The series of microcontrollers are microcontrollers that do not require an external crystal oscillator and an external reset. Th
For the 8051 Single chip microcomputer, at the same operating The series of microcontrollers is better than the test (the speed is about
goal of 11.2~13.2 STC12H Series of microcontrollers only need 12, and traditional 8051
You need 1944 Times), execute the complete instructions in sequence A clock. STC12H The series of microcontrollers use a single clock Machine cycle The single-chip microcomputer is a wide voltage
Highly reliable, Low power consumption, Strong antistatic A new generation Single chip microcomputer, super encrypted. The instruction code is fully comp

MCU Internal integrated high precision R/C , Clock at room temperature (+0.3% ~ +1.42% Wen Piao (-40°C ~+85°C ,)
- 0.88% ~ +1.05% Wen Piao (-20°C ~+65°C.) When programming 4MHz~35MHz Wide range can be set (Note: The temperature range is
At°C, the highest frequency must be controlled below 35MHz), can completely eliminate the expensive external crystal oscillator and external reset circu
integrated high ~+85 When programming
Reliable reset circuit , ISP Stage reset threshold voltage is optional 。4

MCU Inside 3 An optional clock source: internal high Clock (can be adjusted up or down appropriately), internal , The points
external there is 4M~35M Crystal oscillator or external clock signal. The user can freely select the clock source in the code, and the clock source can
After the inverter divides the frequency, the clock signal is provided to (such as timer, serial port 、

MCU Provides two low-power modes : SPI Mode and STOP mode. IDLE In mode , MCU Stop giving CPU Provide clock ,
No clock , CPU CPU Stop executing instructions, but all peripherals are still in the working state. At this time, the power consumption is ab
Rate). STOP mode, which is the master clock stop mode, which is the traditional power-down mode. Power failure mode) All peripherals stop working,
All stop working, and the power consumption can be reduced to

Power-down mode can 、 INT0(P3.2) INT1(P3.3)、 INT2(P0.2)、 INT3(P0.3)、 INT4(P3.0)、 T0(P3.4)、 T1(P3.5)、
be used T2(P1.2)、 T3(P0.0)、 T4(P0.2), RXD(P3.0/P3.5/P1.6) , I2C_SDA(P1.6/P2.4/P3.3) RXD2(P1.0/P5.6), , Oral interruption
SPI_SS(P1.4/P2.2/P3.5)、 CCP0(P3.7)、 CCP1(P3.5)、 CCP2(P2.0)、 CCP2(P2.4), all I/O and comparison
The device is interrupted, the low voltage detection is interrupted, and the power-down wake-up timer is awakened.

MCU It provides a wealth of digital peripherals (serial port, timer , comparator) Advanced and) Interface and analog peripherals (ultra I²C、
High speed ADC to meet the design needs of the majority of users. SPI

STC12H The series of microcontrollers integrates an enhanced dual data pointer inside. Through program control, the data pointer ca
The reduction function and the automatic switching function of the two sets of data pointers.

1.4 STC12H Single chip microcomputer product line

Product line	I/O	UART	Timer	ADC	Advanced PWM	PCA CCP PWM	CMP	SPI	I2C	MDU16	I/O interrupt
STC12H1K08 series	30	$2_{(U1/U2)}$	$5_{(T0/T1/T2/T3/T4)}$	$15_{CH}*10_{B}$	•	•	•	•	•	•	•

2 STC12H Series selection introduction, characteristics, price, pin diagram

2.1 STC12H1K08-36I-LQFP32/TSSOP20/SOP20/SOP16 series

2.1.1 Features and prices

Selection price (no external crystal oscillator required, no external reset required, 10 channel)

Header labels, as printed: Monolithic model number; Industrial voltage (V); Flash Process Sequence memory (Ten thousand times); Large data memory (Byte); xdata Internal system / Capacity expansion (Byte); Strong big double DPTR (Can be increased or decreased); EEPROM (Ten thousand times, Byte); I/O ministry Maximum number of mouths; Serial port and can be powered down to wake up; SPI; I2C; MDU16 hardware 16 Bit multiplication and division device; Timing device / Counter (T0-T4 The external pin can also be powered down and Wake up called); 16 Bit high level PWM (The timer complements each other, dead zone control); PCA/CCP/PWM (Can be used as an external interrupt and can wake up after power failure); Dedicated for power-down wake-up timer; high speed A/D (D/A Use); Comparator (internal Can be used for power-down detection, electricity check); Road inspection; Watchdog; reset timer; Internal high reliability reset (optional reset threshold voltage); Internal high-precision clock (Can be adjusted, Frequency tracking); Cheng Ke Under the support Next Pass Lose (Cheng); It can be software imitated online in itself; Support USB download directly; Price packaging and: LQFP32, QFP28, SKDIP28, TSSOP20, SOP20, SOP16; 2020 New product supply information of the year

Monolithic model number	Industrial voltage (V)	Flash					EEPROM		I/O																						Price packaging	2020
STC12H1K08	1.9-5.5	8K	256	1K	2	20K 30		2	have	have	have	5	8	4	have	bit10	have	have	have	4	There are levels	Yes	have	Yes	Yes	Yes	Yes				¥3.0 ¥3.0 ¥3.2 ¥2.7 ¥2.8 ¥3.0 ¥2.7 ¥2.9	Large quantity supply
STC12H1K16	1.9-5.5	16K	256	1K	2	12K 30		2	have	have	have	5	8	4	have	bit10	have	have	have	4	There are levels	Yes	have	Yes	Yes	Yes	Yes				¥3.1 ¥3.1 ¥3.3 ¥2.8 ¥2.9 ¥3.1 ¥2.8 ¥3.0	
STC12H1K24	1.9-5.5	24K	256	1K	2	4K	30	2	have	have	have	5	8	4	have	bit10	have	have	have	4	There are levels	Yes	have	Yes	Yes	Yes	Yes				¥3.2 ¥3.2 ¥3.4 ¥2.9 ¥3.0 ¥3.2 ¥2.9 ¥3.1	
STC12H1K28	1.9-5.5	28K	256	1K	2	IAP 30		2	have	have	have	5	8	4	have	10bit	have	have	have	4	There are levels	Yes	have	Yes	Yes	Yes	Yes				¥3.2 ¥3.2 ¥3.4 ¥2.9 ¥3.0 ¥3.2 ¥2.9 ¥3.1	

kernel

Core (ultra) high speed 1T 8051 more traditional About fast 12 More than times

The instruction code is fully compatible with traditional 8051

Interrupt source, 4

Interrupt priority 26

Support online

simulation of operating

voltage 1.9V ~ 5.5V

Operating temperature

-40 °C ~ 85 °C

(For over-temperature range applications, please refer to the Electrical characteristics section for instructions)

Flash

Maximum

memory bytes 33K FLASH), used to store user code program memory (ROM

Support user configuration EEPROM Byte single page erasure, the number of erasures can reach 1 More than 10,000 times size ,

Support in-system programming (ISP) Update user applications without a dedicated programmer

Support single-chip simulation, no special emulator is required, and the number of theoretical breakpoints is unlimited

SRAM

128 Direct access inside bytes RAM (DATA)

128 Indirect access inside bytes (IDATA)

1024 Byte internal extended RAM (Internal XDATA)

clock control

Internal high-precision, high-stable high-speed IRC (ISP Can be adjusted up and down when programming)

Scan the code to WeChat small mall

Error ±3 (25°C) Temperature drift (full temperature range, -40°C ~ 85°C) 0.3% (At room temperature -1.35% ~ +1.30% 65°C)

Internal and external IRC Temperature drift (temperature range 0°C ~ 70°C) (For low power consumption, temperature compensation and voltage compensation circuits are omitted, and the error is large

crystal oscillator (4MHz ~ 33MHz) Users are free to choose the above 3) and external clock sources

(Chip power-on operation process: power-on reset, Reset foot reset Watchdog reset, When the low-voltage detection is reset, the chip defaults to the slave The system program starts to execute the gen

code, and the internal high-speed is fixed at this time. Clock, when you need to download the user program and reset to the user program area after the downlo

When you want to download and reset directly to the user program area, the default is to use the high-speed adjusted when the user downloaded last time. IRC

External high-speed crystal oscillator, external crystal oscillator or internal low-speed P3.4/XTAL2/MCLKO/RxD_2 needs to start the corresponding clock first, and then pass

Set the register to switch) CLKSEL

reset

Hardware reset

Reset on power-up, the measured voltage value is 1.82V Valid when the low voltage reset function is not enabled on the chip)

The power-on reset voltage is a voltage range composed of an upper limit voltage and a lower limit voltage. When the power-on voltage is reset from the lower limit threshold voltage of the power-on reset, the chip is in a reset state; when the voltage is reset from the lower limit threshold When it rises to the upper limit threshold voltage of the power-on Chip release reset state.

Reset pin reset, the default is port at the factory, ISP When downloading, you can send P5.4 I/O P5.4 The pin is set to the reset pin (Note: When setting When the pin is a reset pin, the reset level is low)

Watchdog overflow reset

Level low voltage detection voltage: low 2.90V ~ 2.4V (Measured as 2.70V, 2.50V), 2.7V (Measured as voltage detection reset, provide (Measured as 3.0V (Measured as 2.90V ~ 3.13V).

Each stage of the low-voltage detection voltage is a voltage range composed of an upper limit voltage and a lower limit voltage. When the opera drops from the lower limit to the lower threshold voltage of the low-voltage detection, the low-voltage detection takes effect; When the voltage Rise to the upper limit threshold voltage of low voltage dete When the low voltage detection takes effect.

Software reset

Software mode writes reset trigger register

interrupt

Interrupt source: INT0 provide 26 (Support rising and falling edge interrupts), INT1 (Support rising and falling edge interrupts), INT2 (Only support Falling edge interrupt), INT3 (Only falling edge interrupts are supported), (Only supports falling edge interrupts), timer, timer, timer 1 2, Timer, timer, serial port, serial port, ADC2 Analog-to-digital conversion, 3 1 PWM2, PCA/CCP/PWM, EXP0, EXP1, EXP2, LVD Low voltage detection, SPI, I²C, Comparator, PWM1, EXP5EXP3, EXP4

Provide 4-level interrupt priority

Interrupt that can wake up in clock stop mode: INT0(P3.2), INT1(P3.3), INT2(P0.2), INT3(P0.3), INT4(P3.0), T0(P3.4), T1(P3.5), T2(P1.2), T3(P0.0), T4(P0.2), RXD(P3.0/P3.5/P1.6), RXD2(P1.0/P5.6), CCP0(P3.7), CCP1(P3.5), CCP2(P2.0), CCP2(P2.4), I2C_SDA(P1.6/P2.4/P3.3) As well as comparator interrupt, low voltage detection interrupt, power-down wake-up Wake up.

Digital peripherals

5 Bit timer: timer, timer, timer, timer, timer, among which the timer 1 2 3 4 a 16 0 The pattern of Has NMI (Non-maskable interrupt) function, the timer is in bit automatic overload mode 0 1 And timer mode 0 16 A high-speed serial port: serial port, serial port, baud rate clock source can be as fast as 2 FOSC/4 2 1

8 PWM , Can realize the control signal with dead zone, and supports the external anomaly detection function, in addition to supporting Bit timer 2 external interrupts, 8 External capture and measurement of pulse width and other functions

SPI : Support host mode and slave mode, as well as host, Automatic switching from slave to machine

:Support host mode and slave mode I²C

Bit multiplication and division method (support MDU16 Divide by bits 16 bit、16 Bit multiplication Bits, data shifts, and data Hardware standardization and other operations 16)

I/O Mouth interrupt: all All support interrupts, each group Interrupts have independent interrupt entry addresses, allowing 4 Interrupts can be supported in species Interrupt mode: high-level interrupt, low-level interrupt, rising edge interrupt, falling edge interrupt. After testing, it is found that there is a problem, ple

Analog peripherals

Ultra-high speed ADC, support Bit high Channel (channel ~ channel) Analog-to-digital conversion , 500K (Every second 50 Ten thousand times ADC precision conversion 10 15) Used The fastest speed can be achieved

ADC The channel 15 to test the interior 1.19V Reference signal source (when the chip is shipped, the internal reference signal source has been adjusted t

Comparator, a set of comparators (The positive end of the comparator can be selected CMP+ Input port, so the comparator can be used as a multiplexer Comparator for time-sharing multiplexing)

DAC : 8 Lu Senior PWM 8 The timer can be used as the read PCA/CCP/PWM 4 Can be used as a read

GPIO

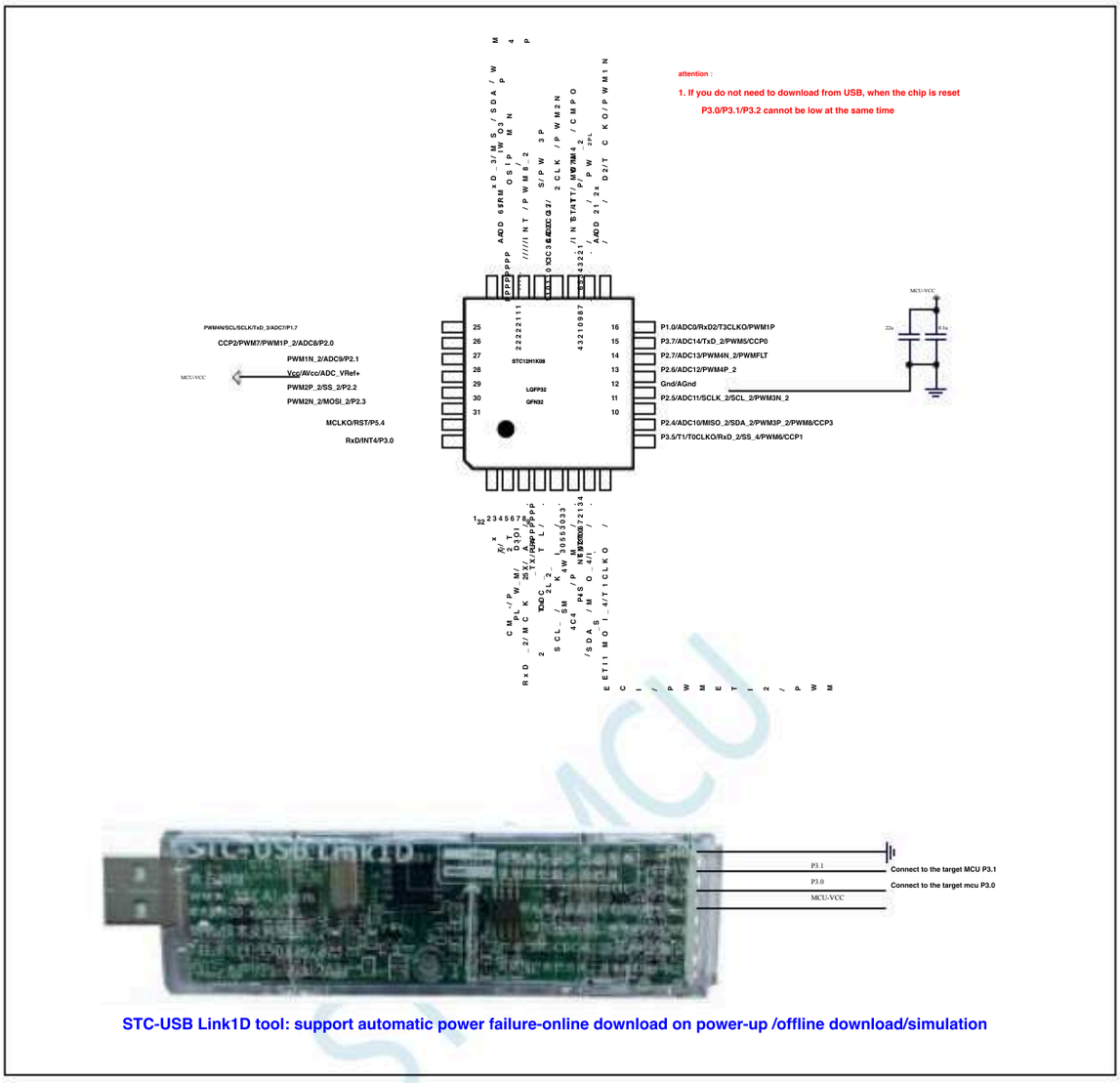
Up to one 30 GPIO : P0.0~P0.3、P1.0~ P1.7、P2.0~P2.7、P3.0~P3.5、P3.7、P5.4、P5.6~P5.7

All of GPIO Three modes: quasi-two-way port mode, strong push-pull output mode, open-drain mode, high-impedance input mode In addition to and P1.0 All support the following In addition, the state of another ports after power-on is a high impedance input state, and the user must first set it when using the po 10 mouth mode. In addition each Can be independently enabled Pull-up resistor

package

LQFP32 (Spot)、QFN32、SOP28、SKDIP28、TSSOP20、SOP20、DIP20、SOP16、DIP16

2.1.2 Pin diagram, minimum system (LQFP32/QFN32)



STC-USB Link1D tool: support automatic power failure-online download on power-up /offline download/simulation

Looking at the small dot at the bottom left of the chip screen printing, the first foot is looking at the bottom line of the chip screen printing, and the last letter is the chip version number.

It is recommended to add a power decoupling capacitor nearby between Vcc and Gnd , Can remove power cord noise and improve anti-interference ability

ISP Download steps :

1 , According to the connection STC-USB Link1D Connect to the target chip method shown in the figure with 2 ISP , In the download software "Download" click "Program" button

Download (Note: if you use 3 STC-USB Link1D ISP start Power supply to the target system, the total current of the target system is n 200mA , Otherwise it will cause the download to fail.

about I/O Precautions :

1、 P3.0 P3 The state of the port after power-up is weak pull-up / Quasi-two-way port mode

2, In addition to and The state of the port after power-on is a high-impedance input state, and the user is using it. P3.0

Except that it must be set before the mouth, all the rest P3.1 IO Mouth mode IO this 3 a I/O Mouth can't USB

、 It is low at the same time, otherwise it will ISP The user code cannot be run in download

3 If you don't need to use the download when the chip is powered P3.1 mode and is low at the same time , P3.2 , The port will switch from the high impedance input state in a short

4 When the chip is powered on, if and P3.0

To two-way port mode P3.2 To determine whether you need to enter the external USB state of the download mode

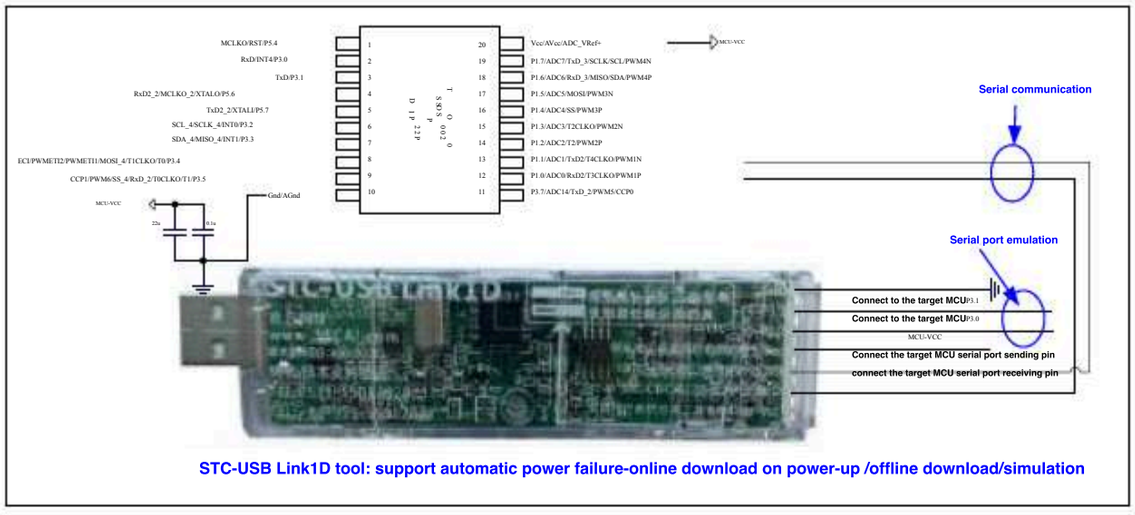
for reading when used 3 When used as a reset pin, the internal one of this port P5.4 The pull-up resistor will always be turned on, but 4K Do general P5.4

When speaking, based on this I/O Pass I/O The port and the reset pin share the special considerations of the pin, the internal Pull-up

The door will still open approximately 6.5 Milliseconds, and then automatically turn off (When the user's circuit design needs to be used P5.4 port use

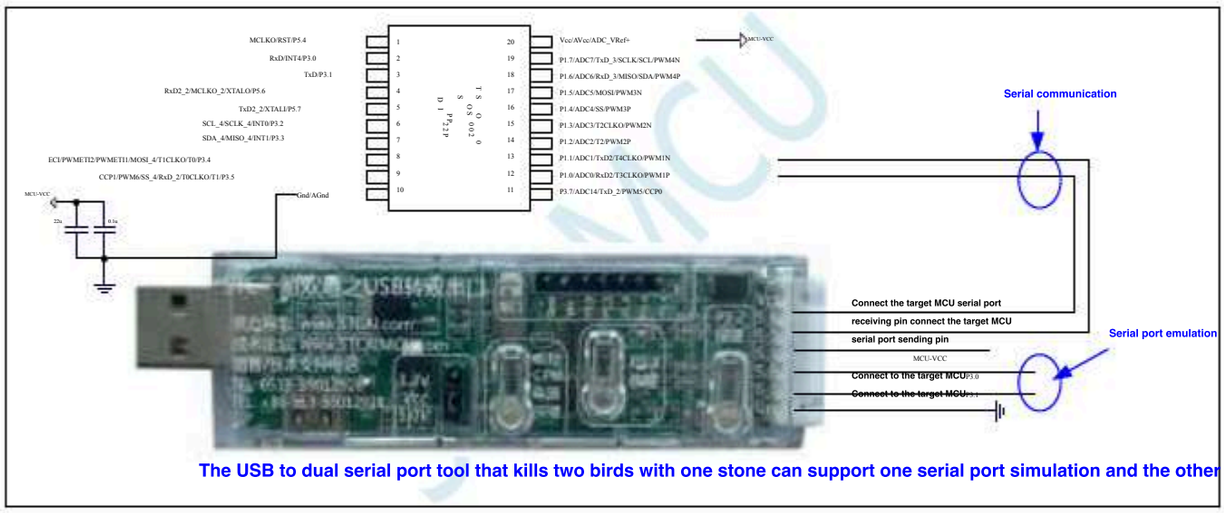
When driving an external circuit, be sure to consider that there will be at the moment of power-up. 6.5 The problem of high levels in milliseconds)

2.1.3 use STC-USB Link1D correct STC12H Series for serial port simulation+Serial commun



STC-USB Link1D tool: support automatic power failure-online download on power-up /offline download/simulation

2.1.4 use USB Turn to dual serial port tool pair STC12H departmentColumn for serial port simulation+Seria



The USB to dual serial port tool that kills two birds with one stone can support one serial port simulation and the other serial port communicati

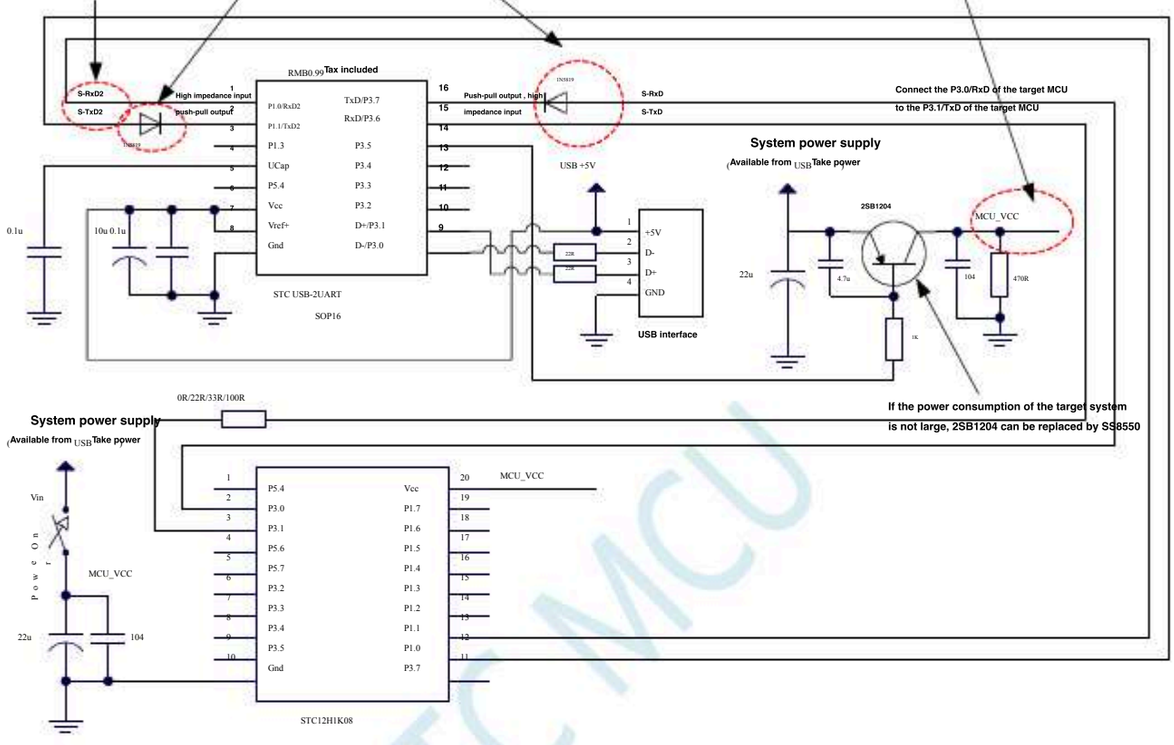
2.1.5 Use universal USB Turn to dual serial port chip pair STC12H Series for serial port simulation + Serial comm

The role of the diodes and resistors here is to prevent the USB device from powering the target chip. Since the target chip cannot be powered off during the power-off process, the USB device cannot be powered off. The transmission pin of the USB device is generally a strong push-pull output, which can work through the I/O power supply chip , so when using USB to serial port TTL to download the program to the STC microcontroller, the isolation diode here cannot be saved.

When the chip is powered on, the P3.5 output is low, and MCU_VCC is in the power supply state. When it detects that an ISP download is required, MCU_VCC will automatically control the power failure for 0.5s and then power on for ISP download. Power will continue to be powered after the download is complete

These two ports are the second set of serial ports, the first set is used for download or communication, and the second set of serial ports is used for serial communication.

Note: The latest USB to dual serial software can automatically add the second

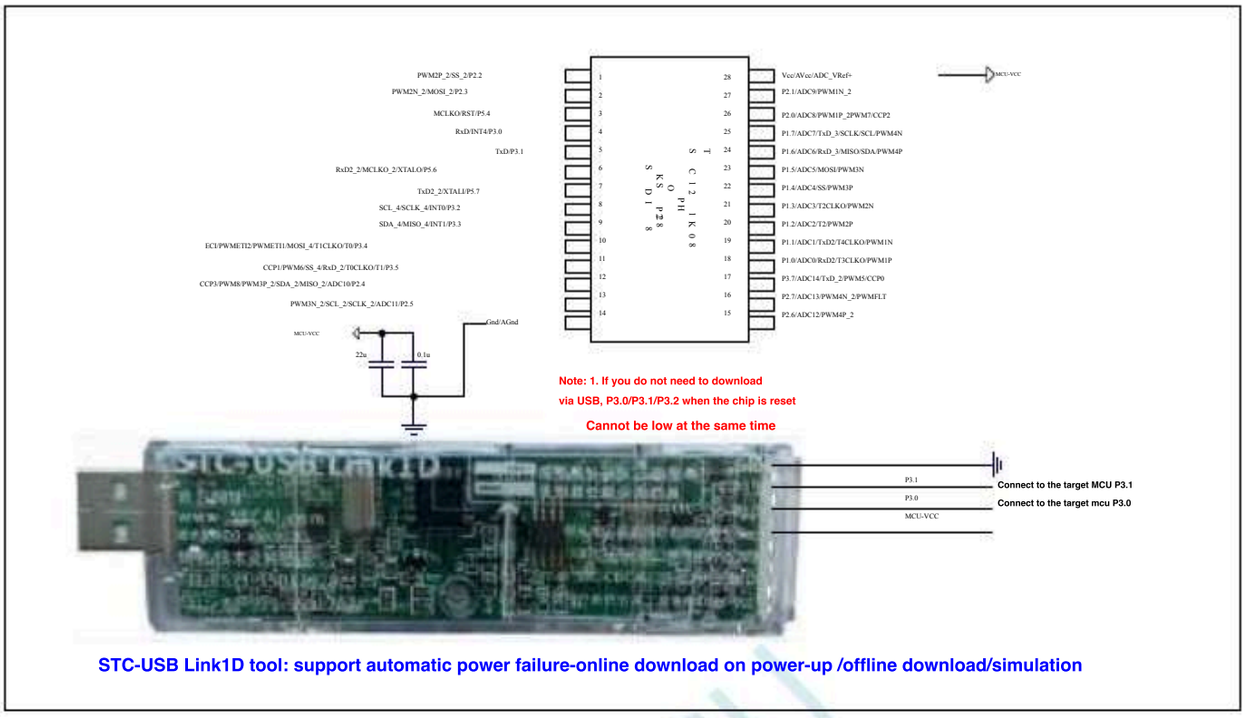


USB to dual serial port chip can choose the following models:

STC8H2K08U-45I-TSSOP20/SOP16, RMB0.99 (tax included)	The factory comes with a USB to dual serial port program , the
STC8H2K12U 2CDC+HID-TSSOP20/SOP16, RMB1.1 (tax included)	factory comes with a USB to dual serial port+HID program ,
STC8H8K64U-45I-TSSOP20/SOP16, RMB1.4 (tax included)	the factory comes with a USB to dual serial port+HID program ,
STC USB-2UART-TSSOP20/SOP16, RMB1.4 (Tax included)	the factory comes with a USB to dual serial port+HID program

The latest version of the above comes with a USB-to-dual serial port program at the factory, which supports automatic power failure and power-on burning, which can save isolation diodes.

2.1.6 Pin diagram, minimum system (SOP28/SKDIP28)



STC-USB Link1D tool: support automatic power failure-online download on power-up /offline download/simulation

Looking at the small dot at the bottom left of the chip screen printing,
the first foot is looking at the bottom line of the chip screen printing, and the
last letter is the chip version number. It is recommended in Vcc/Gnd and 0.1uF , Add power decoupling capacitors nearby , Can remove power cord noise and improve anti-interference ability

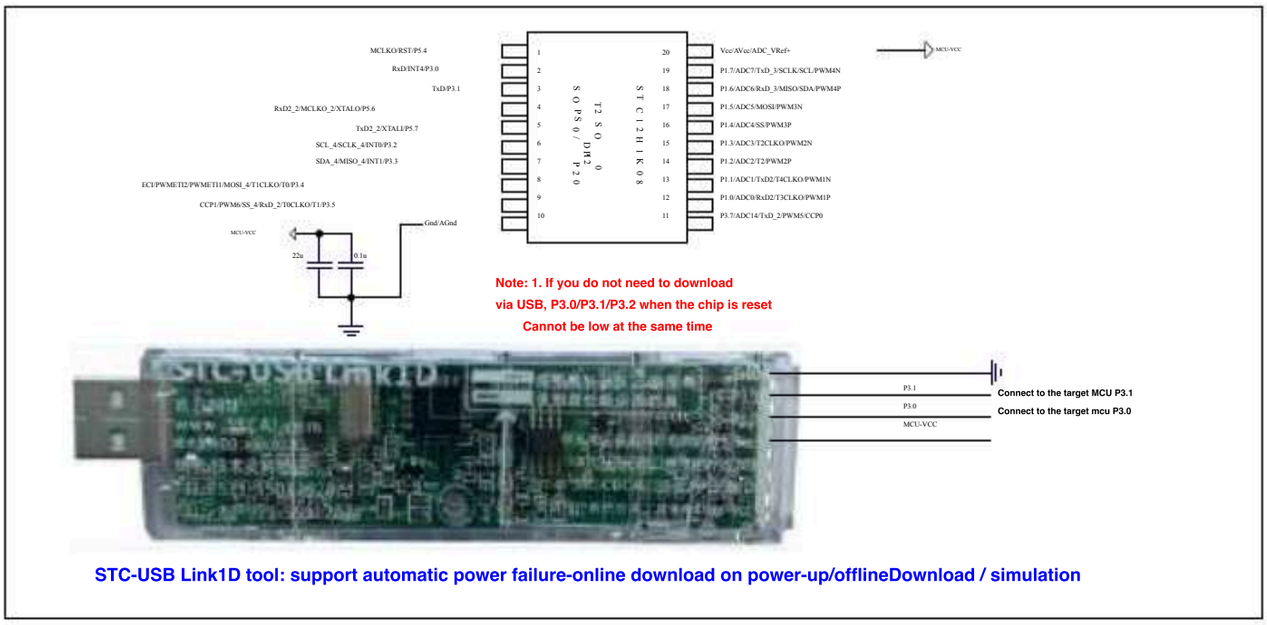
ISP Download steps :

1 , According to the connection STC-USB Link1D Connect to the target chip
method shown in the figure with 2 , In the download software "Program" button
"Download" click
3, start Download (Note: if you use ISP STC-USB Link1D To supply power to the target system, the total current of the target system c
200mA , Otherwise it will cause the download to fail.

about I/O Precautions :

1、 P3.0 P3 The state of the port after power-up is weak pull-up / Quasi-two-way port mode
2, In addition to and and The state of the port after power-on is a high-impedance input state, and the user is using it.
P3.0
Except that it must be set before the mouth, all the rest P3.1 IO Mouth Mouth IO this 3 a I/O Mouth can't
、 It is low at the same time, otherwise it will be unable to run user code in download mode USB
3 If you don't need to use the download when the chip is powered At the same time low level , P3.2 USB ISP Proceed
4 When the chip is powered on, if and P3.0 P3.1 To determine whether you need to enter the external state of the mouth
for reading When used as a reset pin, the internal one of this port pull-up resistor will always be turned on, but 4K
when used
When speaking, based on this The port and the reset pin share the special considerations of the pin, the interna
The door will still open approximately Milliseconds, and then automatically turn off (When the user's circuit design needs to be used
When driving an external circuit, be sure to consider that there will be at the moment of power-up. The problem of high levels in milliseconds)

2.1.7 Pin diagram, minimum system (TSSOP20/SOP20/DIP20)



STC-USB Link1D tool: support automatic power failure-online download on power-up/offlineDownload / simulation

Looking at the small dot at the bottom left of the chip screen printing,
the first foot is looking at the bottom line of the chip screen printing, and the
last letter is the chip version number. It is recommended in Vcc/Gnd and 0.1uF Add power decoupling capacitors nearby , Can remove power cord noise and improve anti-interference ability

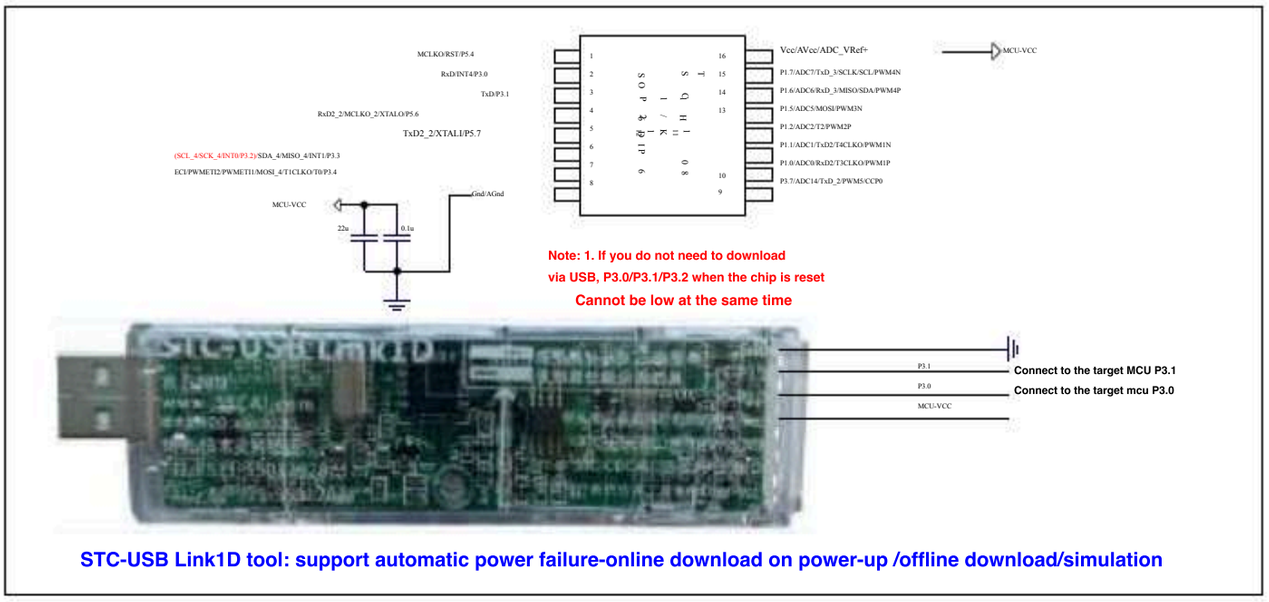
ISP Download steps :

1 , According to the connection STC-USB Link1D Connect to the target chip
method shown in the figure with 2 ISP STC-ISP "Download" click the download software "Program" button
Download (Note: if you use 3 STC-USB Link1D ISP start To supply power to the target system, the total current of the target system c
200mA , Otherwise it will cause the download to fail.

about I/O Precautions :

1、 P3.0 P3 The state of the port after power-up is weak pull-up / Quasi-two-way port mode
2, In addition to and and The state of the port after power-on is a high-impedance input state, and the user is using it.
P3.0
Except that it must be set before the mouth, all the rest P3.1 IO Mouth mode IO this 3 a I/O Mouth can't
、 It is low at the same time, otherwise it will The user code cannot be run in download
3 If you don't need to use the download mode and use it as an ordinary mode and set it at the same time , P3.2
4 When the chip is powered on, if and P3.2 P3.0 To determine whether you need to enter the external state of the mouth
for reading When used as a reset pin, the internal one of this pin The pull-up resistor will always be turned on, but 4K
When speaking, based on this The port and the reset pin share the special considerations of the pin, the interna
The door will still open approximately Milliseconds, and then automatically turn off (When the user's circuit design needs to
When driving an external circuit, be sure to consider that there will be at the moment of power-up. The problem of high levels in milliseconds)

2.1.8 Pin diagram, minimum system (SOP16/DIP16)



Note: 1. If you do not need to download via USB, P3.0/P3.1/P3.2 when the chip is reset Cannot be low at the same time

STC-USB Link1D tool: support automatic power failure-online download on power-up /offline download/simulation

Remarks : STC12H1K08 Series of SOP16 Package form , P3.2/P3.3 Inside the chip is 2 An independent I/O But the package is straight Connect the shorted together to make it easy for everyone to choose.

Looking at the small dot at the bottom left of the chip screen printing, the first foot is looking at the bottom line of the chip screen printing, and the last letter is the chip version number.

It is recommended to add a power decoupling capacitor nearby between Vcc and Gnd , Can remove power cord noise and improve anti-interference ability

ISP Download steps :

1 , According to the connection method shown in the figure with STC-USB Link1D Connect to the target chip

2 、 First STC-USB Link1D "Download" click the download software "Program" button

Download (Note: if you use STC-USB Link1D ISP start To supply power to the target system, the total current of the target system c 200mA , Otherwise it will cause the download to fail.)

about I/O Precautions :

1、 P3.0 P3 The state of the port after power-up is weak pull-up / Quasi-two-way port mode

2, In addition to P3.0 and The state of the port after power-on is a high-impedance input state, and the user is using it. I/O

Except that it must be set before the USB mouth, all the rest P3.1 I/O Mouth mode P3.0 I/O this 3 a I/O Mouth can't

、 It is low at the same time, otherwise it will ISP The user code cannot be run in download

3 If you don't need to use the download when the chip is powered on, and P3.1 mode and is low at the same time , P5.2 To two-way port mode

4 When the chip is powered on, if and P3.0 To determine whether you need to enter the external state of the mouth USB Do the mouth for reading when used as a reset pin, the internal one of this port pull-up resistor will always be turned on, but when used

When speaking, based on this I/O The port and the reset pin share the special considerations of the pin, the interna The door will still open approximately Milliseconds, and then automatically turn off (When the user's circuit design needs to be used Pull-up When driving an external circuit, be sure to consider that there will be at the moment of power-up. The problem of high levels in milliseconds)

2.1.9 Pin description

number				name	type	description
LQFP32 QFN32	SOP28 SKDIP28	TSSOP20 SOP20	SOP16			
1	5	3	3	P3.1	I/O	standard IO The sending
				TxD	O	serial port 1 foot of the mouth
2				P0.0	I/O	standard IO mouth
				T3	I	Timer External clock input 3
				PWM5_3	I/O	Capture input and pulse output PWM5
				CMP-	I	Comparator negative input
3	6	4	4	P5.6	I/O	mouth
				RxD2_2	I	IO Standard string Receiving foot of the mouth 2
				XTALO	O	The output pin of the external crystal oscillator , the master clock divider output
				MCLKO_2	O	
4	7	5	5	P5.7	I/O	mouth
				TxD2_2	O	IO The sending pin of the standard serial port 2
				XTALI	I	External crystal oscillator / Input pin of external clock
5	8	6	6	P3.2	I/O	standard IO mouth
				INT0	I	External interrupt 0
				SCLK_4	I/O	The clock foot SPI
				SCL_4	I/O	I2C
6				P0.1	I/O	standard IO mouth The clock line
				PWM6_3	I/O	PWM6 Capture input and pulse output
				CMP+	I	Comparator positive input
7	9	7	6	P3.3	I/O	standard IO mouth
				INT1	I	External interrupt
				MISO_4	I/O	Host input slave output SPI
				SDA_4	I/O	I2C
8	10	8	7	P3.4	I/O	Standard data cable IO
				T0	I	Timer External clock input , clock
				T1CLKO	O	Timed divider output
				MOSI_4	I/O	SPI 10
				PWMETI	I	Host output slave input PWM External trigger input pin
				PWMETI2	I	PWM External pulse input of
				ECI	I	PCA external trigger input pin

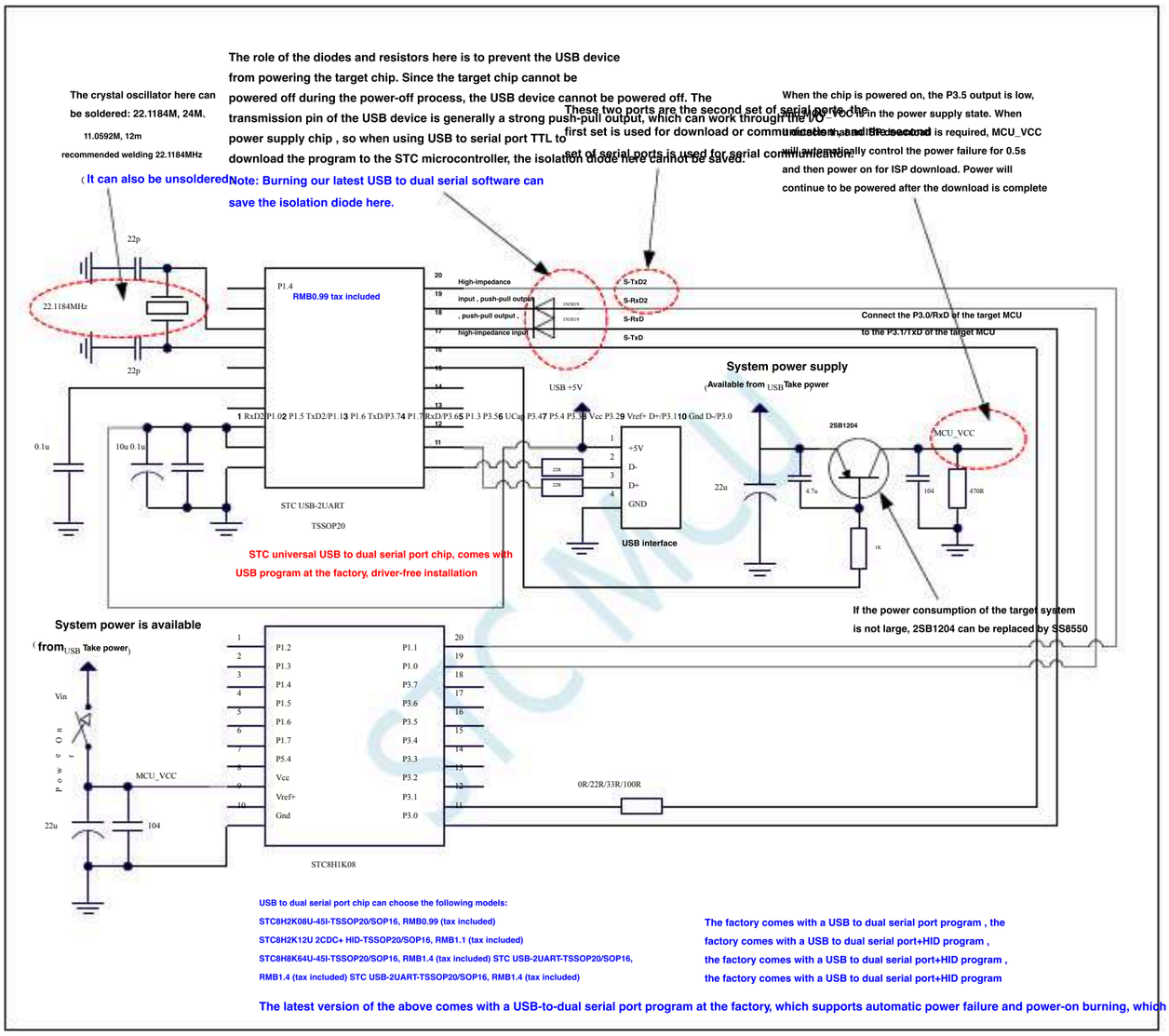
number				name	type	description
LQFP32 QFN32	SOP28 SKDIP28	TSSOP20 SOP20	SOP16			
9	11	9		P3.5	I/O	standard IO mouth
				T1	I	Timer external 1
				T0CLKO	O	clock input timer 0
				RxD_2	I	clock divider The receiving foot
				SS_4	I/O	output serial port SPI Slave selection
				PWM6	I/O	PWM6 Capture input and pulse output
				CCP1	I/O	PCA capture input, pulse output and PWM output
10	12			P2.4	I/O	standard IO Port analog input
				ADC10	I	ADC channel 10
				MISO_2	I/O	SPI Host input slave output
				SDA_2	I/O	Data cable I2C
				PWM3P_2	I/O	PWM3 Capture input and pulse output Capture input
				PWM8	I/O	PWM8 and pulse output of the positive electrode
				CCP3	I/O	PCA Capture input, pulse output and PWM output
11	13			P2.5	I/O	standard mouth IO
				ADC11	I	ADC The 11
				SCLK_2	I/O	SPI clock line of the clock pin of
				SCL_2	I/O	I2C the analog input channel
				PWM3N_2	I/O	PWM3 The capture input and pulse output are negative
12	14	10	8	Gnd	Gnd	Ground
				AGnd	Gnd	wire ADC Ground wire
13	15			P2.6	I/O	standard mouth IO
				ADC12	I	ADC Analog input channel 12
				PWM4P_2	I/O	PWM4 Capture input and pulse output positive electrode
14	16			P2.7	I/O	standard mouth IO
				ADC13	I	ADC Analog input channel 13
				PWM4N_2	I/O	PWM4 Capture inputAnd the external anomaly detection
				PWMFLT	I	PWM1 pin of the negative electrode of the pulse output
15	17	11	9	P3.7	I/O	standard IO Port analog input
				ADC14	I	ADC channel 14
				TxD_2	O	serial port 1 The sending foot
				PWM5	I/O	PWM5 Capture input and pulse output
				CCP0	I/O	PCA capture input, pulse output and PWM output

number				name	type	description
LQFP32 QFN32	SOP28 SKDIP28	TSSOP20 SOP20	SOP16			
16	18	12	10	P1.0	I/O	standard mouth IO
				ADC0	I	ADC Analog input channel 0
				RxD2	I	serial port 2 The receiving foot
				T3CLKO	O	Timer 3 Clock divider output
				PWM1P	I/O	PWM1 Capture input and pulse output positive electrode
17	19	13	11	P1.1	I/O	standard IO Port analog input
				ADC1	I	ADC channel 1
				TxD2	O	serial port 2 The sending foot
				T4CLKO	O	Timer 4 Clock divider output
				PWM1N	I/O	PWM1 The capture input and pulse output are negative
18	20	14	12	P1.2	I/O	standard IO Port analog input
				ADC2	I	ADC channel 2
				T2	I	Timer 2 External clock input
				PWM2P	I/O	PWM2 Capture input and pulse output positive electrode
19				P0.2	I/O	standard IO mouth
				INT2	I	External interrupt
				T4	I	4 External clock
				PWM7_2	I/O	input timer PWM7 Capture input and pulse output
				CMPO	O	Comparator output
20	21	15		P1.3	I/O	standard IO mouth
				ADC3	I	Analog input channel ADC
				T2CLKO	O	3 Timer clock divider output
				PWM2N	I/O	PWM2 The capture input and pulse output are negative
21	22	16		P1.4	I/O	Standard port IO
				ADC4	I	ADC Analog input 4
				SS	I/O	SPI channel slave
				PWM3P	I/O	selection PWM3 Capture input and pulse output positive electrode
22				P0.3	I/O	Standard port
				INT3	I	External interrupt 3
				PWM8_2	I/O	PWM8 Capture input and pulse output
23	23	17	13	P1.5	I/O	standard IO Port analog input
				ADC5	I	ADC channel 5
				MOSI	I/O	Host output slave input SPI
				PWM3N	I/O	PWM3 The capture input and pulse output are negative

number				name	type	description
LQFP32 QFN32	SOP28 SKDIP28	TSSOP20 SOP20	SOP16			
24	24	18	14	P1.6	I/O	standard mouth IO
				ADC6	I	ADC Analog input channel 6
				RxD_3	I	serial port 1 The receiving foot
				MISO	I/O	SPI The data cable output from
				SDA	I/O	I2C the host input to the slave
				PWM4P	I/O	PWM4 Capture input and pulse output positive electrode
25	25	19	15	P1.7	I/O	standard mouth IO
				ADC7	I	ADC Analog input channel 7
				TxD_3	O	serial port 1 The clock line of the
				SCLK	I/O	SPI clock pin of the transmitting pin
				SCL	I/O	I2C
				PWM4N	I/O	PWM4 The capture input and pulse output are negative
26	26			P2.0	I/O	standard mouth IO
				ADC8	I	ADC Analog input channel 8
				PWM1P_2	I/O	PWM1 Capture input and pulse output Capture input
				PWM7	I/O	PWM7 and pulse output of the positive electrode
				CCP2	I/O	PCA Capture input, pulse output and PWM output
27	27			P2.1	I/O	standard mouth IO
				ADC9	I	ADC Analog input channel 9
				PWM1N_2	I/O	PWM1 The capture input and pulse output are negative
28	28	20	16	Vcc	Vcc	Power pin
				AVcc	Vcc	ADC power supply
				ADC_VRef+	I	ADC standard External reference voltage source input pin
29	1			P2.2	I/O	SPI mouth
				SS_2	I/O	PWM2 IO
				PWM2P_2	I/O	Slave selection standard Capture input and pulse output positive electrode
30	2			P2.3	I/O	SPI IO mouth
				MOSI_2	I/O	PWM2 Host output slave input
				PWM2N_2	I/O	standard The capture input and pulse output are negative
31	3	1	1	P5.4	I/O	mouth
				RST	I	IO
				MCLKO	O	Reset pin master clock divider output
32	4	2	2	P3.0	I/O	mouth
				RxD	I	IO 1 Receiving pin of standard serial port
				INT4	I	External interrupt

2.2 Universal USB Turn to dual serial port chip STC USB-2UART, TSSOP20/SOP16

2.2.1 Universal automatic power failure and power-on USB to dual serial port chip STC USB-2UART-45I-TSSOP20



2.2.2 Universal to dual serial port chip STC USB-2UART-TSSOP20, Manual power failure and power-on

The role of the diodes and resistors here is to prevent the USB device from powering the target chip. Since the target chip cannot be powered off during the power-off process, the USB device cannot be powered off. The transmission pin of the USB device is generally a strong push-pull output, which is used for downloading the program through the I/O power supply chip, so the sending USB TTL to STC, the transmission pin of the USB device is generally used for serial communication through the I/O power

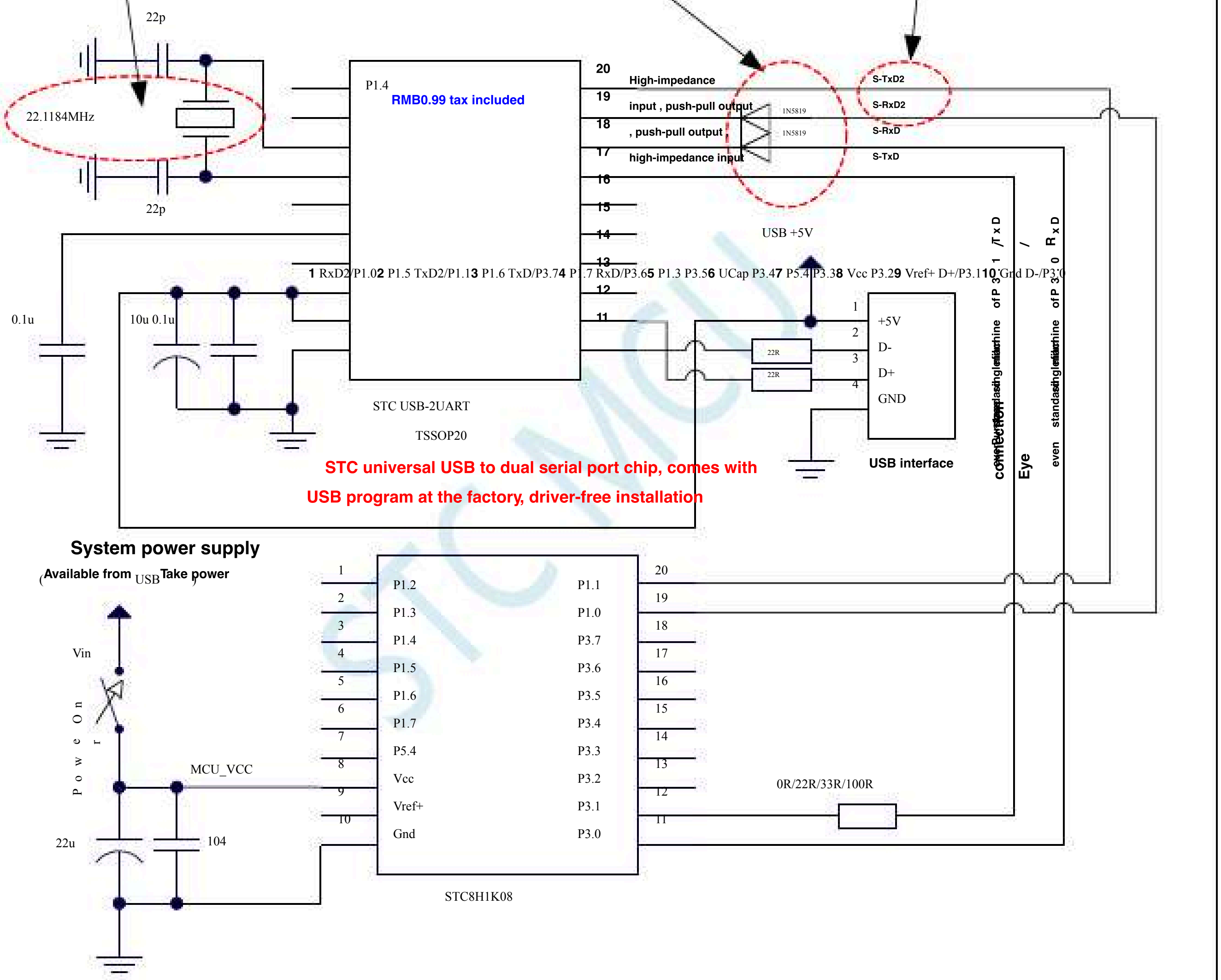
These two ports are the second set of serial ports, the first serial port for download and power supply, and the second set of serial ports is used for serial communication.

The crystal oscillator here can be soldered: 22.1184M, 24M、11.0592M, 12M

It is recommended to weld the isolation diode here when downloading the program from the 22.1184mhz MCU, which cannot be saved.

（It can also be unsoldered）

Note: Burning our latest USB to dual serial software can save the isolation diode here.



USB to dual serial port chip can choose the following models:

- STC8H2K08U-45I-TSSOP20/SOP16, RMB0.99 (tax included) — The factory comes with a USB to dual serial port program ,
- STC8H2K12U 2CDC+HID-TSSOP20/SOP16, RMB1.1 (tax included) — the factory comes with a USB to dual serial port+HID program ,
- STC8H8K64U-45I-TSSOP20/SOP16, RMB1.4 (tax included) — the factory comes with a USB to dual serial port+HID program ,
- STC USB-2UART-TSSOP20/SOP16, RMB1.4 (Tax included) — the factory comes with a USB to dual serial port+HID program

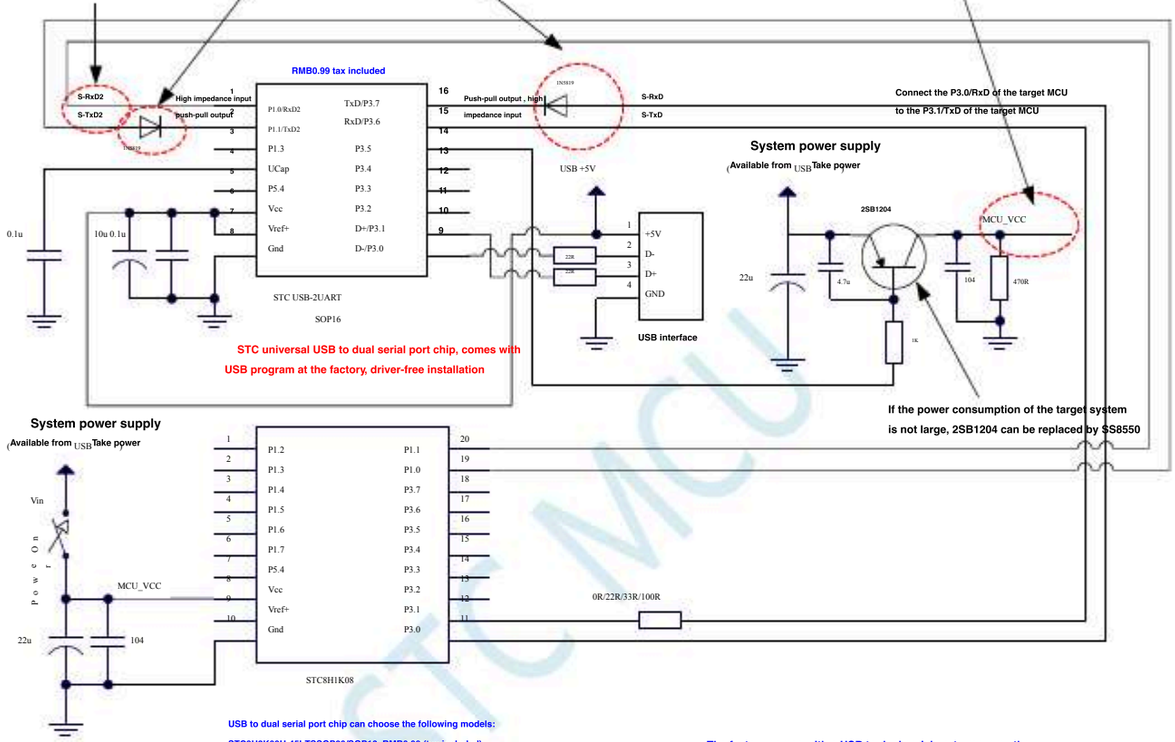
The latest version of the above comes with a USB-to-dual serial port program at the factory, which supports automatic power failure and power-on

2.2.3 Universal to dual serial port chip STC USB-2UART-TSSOP20/SOP16, since Dynamic power failure and power-on

The role of the diodes and resistors here is to prevent the USB device from powering the target chip. Since the target chip cannot be powered off during the power-off process, the USB device cannot be powered off. The transmission pin of the USB device is generally a strong push-pull output, which can work through the I/O power supply chip , so when using USB to serial port TTL to download the program to the STC microcontroller, the isolation diode here cannot be saved.

When the chip is powered on, the P3.5 output is low, and MCU_VCC is in the power supply state. When it detects that an ISP download is required, MCU_VCC will automatically control the power failure for 0.5s and then power on for ISP download. Power will continue to be powered after the download is complete

These two ports are the second set of serial ports Note: Burn our latest USB to dual serial software, you can The first group is used for download or communication To save energy-saving isolation diodes The second set of serial ports is used for serial communication



USB to dual serial port chip can choose the following models:

STC8H2K08U-45I-TSSOP20/SOP16, RMB0.99 (tax included)

STC8H2K12U 2CDC+ HID-TSSOP20/SOP16, RMB1.1 (tax included)

STC8H8K64U-45I-TSSOP20/SOP16, RMB1.4 (tax included) STC USB-2UART-TSSOP20/SOP16, RMB1.4 (tax included) STC USB-2UART-TSSOP20/SOP16, RMB1.4 (tax included)

The factory comes with a USB to dual serial port program , the factory comes with a USB to dual serial port+HID program , the factory comes with a USB to dual serial port+HID program , the factory comes with a USB to dual serial port+HID program

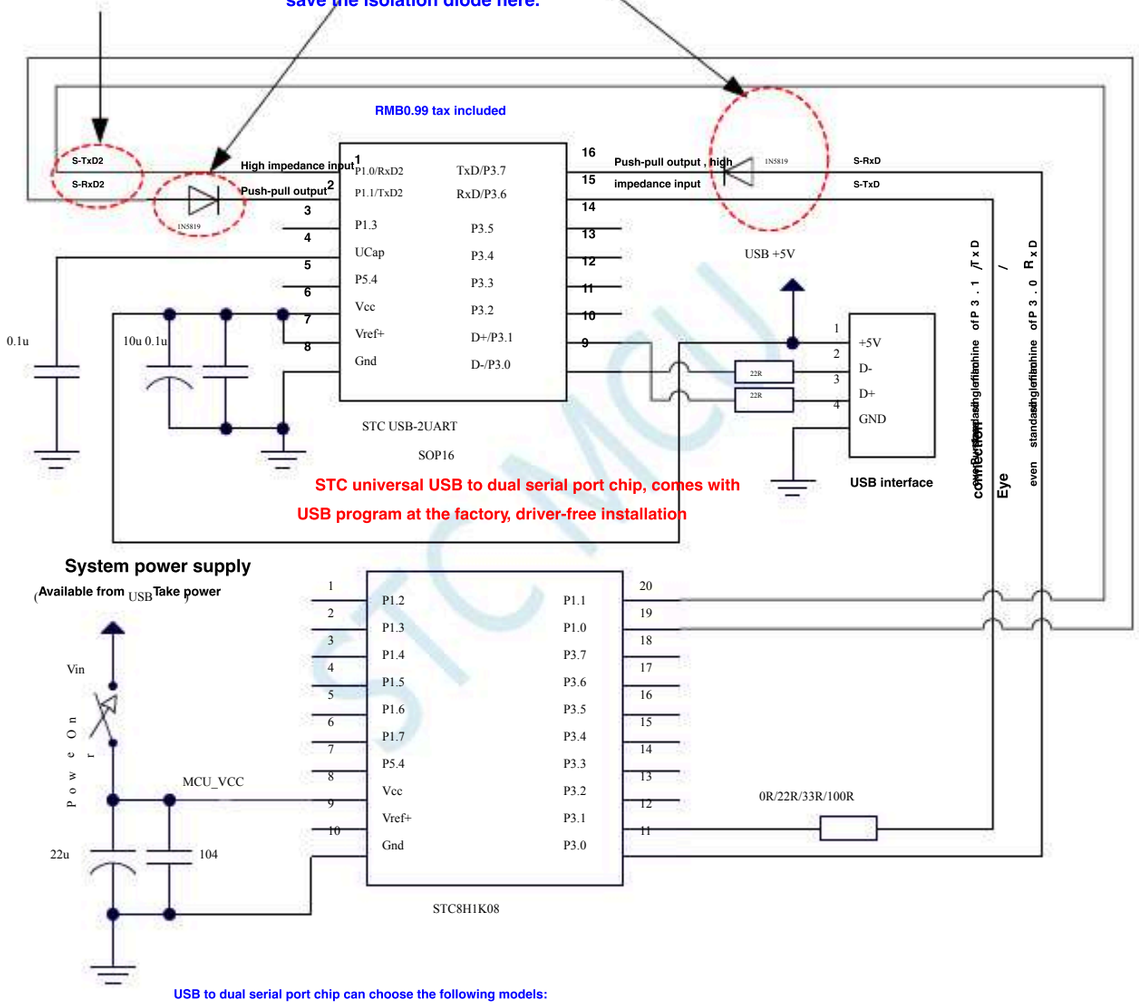
The latest version of the above comes with a USB-to-dual serial port program at the factory, which supports automatic power failure and power-on burning, which can save isolation diodes.

2.2.4 Universal to dual serial port chip STC8H1K08 SOP16, hand Dynamic power failure and power-on

The role of the diodes and resistors here is to prevent the USB device from powering the target chip. Since the target chip cannot be powered off during the power-off process, the USB device cannot be powered off. The transmission pin of the USB device is generally a strong push-pull output, which can work through the I/O power supply chip , so when using USB to serial port TTL to download the program to the STC microcontroller, the isolation diode here cannot be saved.

Note: Burning our latest USB to dual serial software can save the isolation diode here.

These two ports are the second set of serial ports, the first set is used for download or communication, and the second set of serial ports is used for serial communication.



USB to dual serial port chip can choose the following models:

STC8H2K08U-45I-TSSOP20/SOP16, RMB0.99 (tax included)

STC8H2K12U 2CDC+ HID-TSSOP20/SOP16, RMB1.1 (tax included)

STC8H8K64U-45I-TSSOP20/SOP16, RMB1.4 (tax included) STC USB-2UART-TSSOP20/SOP16, RMB1.4 (tax included) STC USB-2UART-TSSOP20/SOP16, RMB1.4 (tax included)

The factory comes with a USB to dual serial port program , the factory comes with a USB to dual serial port+HID program , the factory comes with a USB to dual serial port+HID program , the factory comes with a USB to dual serial port+HID program

The latest version of the above comes with a USB-to-dual serial port program at the factory, which supports automatic power failure and pow

Function foot switch

Change 3 Special peripheral serial port of a series of microcontrollers, In order to realize time-sharing multiplexing of a peripheral as multiple devices. And the bus control pin can be in multiple I/O Quasi directly STC12H

3.1 Function pin switch related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW1	Peripheral port switch register	A2H	S1_S[1:0]		-	-	SPI_S[1:0]		0	-	nnxx,000x
P_SW2	Peripheral port switch register	BAH	EAXFR	-	I2C_S[1:0]		-	-	-	S2_S	0x00,xxx0

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
MCLKOCR	Master clock output control register	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
PWMA_PS	Switch register PWMA	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMB_PS	PWMB Switch register	FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMA_ETRPS PWMA	of ETR Select register	FEB0H						BRKAPS	ETRAPS[1:0]		xxxx,x000
PWMB_ETRPS PWMB	of ETR Select register	FEB4H						BRKBPS	ETRBPS[1:0]		xxxx,x000

3.1.1 Peripheral port switching control register (1P_SW1), serial port 、 1SPI switch

symbol	address		B6 B7	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		-	-	SPI_S[1:0]		0	-

S1_S[1:0]: Serial port 1 Function foot selection bit

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

SPI_S[1:0] : SPI Function foot selection bit

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

Peripheral port switching control register 3.1.2) serial port 2/3/4、 I2C、

Comparator output switching

symbol	address	B7	B6	B4 B5	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]	-	-	-	S2_S

) Access control register XFR

Area special Extended RAM EAXFR

: Access is prohibited. 0 function register (XFR

: Enable access. 1 XFR

When you need to access it, you must first Set it to berright EAXFR Perform normal reading and writing

I2C_S[1:0] : I² C Function foot selection bit

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

S2_S : Serial port function pin

selection bit S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P5.6	P5.7

3.1.3 Clock selection register (MCLKOCR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	FE05H	MCLKO_S	MCLKODIV[6:0]						

: Master clock output pin selection bit MCLKO_S

MCLKO_S	MCLKO
0	P5.4
1	P5.6

3.1.4 Advanced PWM Select register (PWMx_PS)

symbol	address		B6		B4	B3	B2		B0
PWMA_PS	FEB2H	B7 C4PS[1:0]		B5 C3PS[1:0]		C2PS[1:0]		B1 C1PS[1:0]	
PWMB_PS	FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	

C1PS[1:0]: Advanced Channel output pin selection bit PWM

C1PS[1:0]	PWM1P PWM1N	
00	P1.0	P1.1
01	P2.0	P2.1
10	-	-
11	-	-

C2PS[1:0]: Advanced PWM channel 2 Output pin selection bit PWM2N

C2PS[1:0]	PWM2P	
00	P1.2	P1.3
01	P2.2	P2.3
10	-	-
11	-	-

C3PS[1:0]: Advanced PWM channel 3 Output pin selection bit PWM3N

C3PS[1:0]	PWM3P	
00	P1.4	P1.5
01	P2.4	P2.5
10	-	-
11	-	-

C4PS[1:0]: Advanced PWM channel 4 Output pin selection bit PWM4N

C4PS[1:0]	PWM4P	
00	P1.6	P1.7
01	P2.6	P2.7
10	-	-
11	-	-

C5PS[1:0]: Advanced PWM channel 5 Output pin selection bit

C5PS[1:0]	PWM5
00	P3.7
01	P0.0
10	-
11	-

C6PS[1:0]: Advanced PWM channel 6 Output pin selection bit

C6PS[1:0]	PWM6
00	P3.5
01	P0.1
10	-
11	-

C7PS[1:0]: Advanced PWM channel 7 Output pin selection bit

C7PS[1:0]	PWM7
00	P2.0
01	P0.2
10	-
11	-

C8PS[1:0]: Advanced PWM Channel output pin selection bit 8

C8PS[1:0]	PWM8
00	P2.4
01	P0.3
10	-
11	-

3.1.5 Advanced PWM Function pin selection register (PWMx_ETRPS)

symbol	address	B7	B6	B5	B4	B3	B2		B0
PWMA_ETRPS	FEB0H						BRKAPS	B1 ETRAPS[1:0]	
PWMB_ETRPS	FEB4H						BRKBPS	ETRBPS[1:0]	

ETR1PS[1:0]: Advanced PWM1 External trigger foot ERI1 Select bit

ETR1PS [1:0] PWMETI	
00	P3.4
01	-
10	-
11	-

ETR2PS[1:0]: Advanced PWM2 External trigger foot PWMETI2 ERI2 Select bit

ETR2PS [1:0]	
00	P3.4
01	-
10	-
11	-

BRK1PS: Advanced PWM The brake foot PWMFLT Select bit

BRK1PS	PWMFLT
0	P2.7
1	Output of the comparator

BRK2PS: Advanced The brake foot PWMFLT2 Select bit

BRK2PS	PWMFLT2
0	P2.7
1	Output of the comparator

3.2 Sample program

3.2.1 1 Serial port switching C Language code

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

#include "reg51.h"

sfr     P_SW1   =   0xa2;

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;
    P_SW1 = 0x00;                   //RXD/P3.0, TXD/P3.1
//  P_SW1 = 0x40;                   //RXD_2/P3.6, TXD_2/P3.7
//  P_SW1 = 0x80;                   //RXD_3/P1.6, TXD_3/P1.7
    while (1);

}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

P_SW1   DATA    0A2H

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_SW1,#00H              ;RXD/P3.0, TXD/P3.1
;               MOV     P_SW1,#40H              ;RXD_2/P3.6, TXD_2/P3.7
;               MOV     P_SW1,#80H              ;RXD_3/P1.6, TXD_3/P1.7

                SJMP    $

                END
```

2 Serial port switching 3.2.2

C Language code

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


#include "reg51.h"

sfr     P_SW2   =   0xba;

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

```
sfr     P5M0    =   0xca;

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 = 0x00;                   //RXD2/P1.0, TXD2/P1.1
//  P_SW2 = 0x01;                   //RXD2_2/P5.6, TXD2_2/P5.7
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

P_SW2   DATA    0BAH

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,#00H          ;RXD2/P1.0, TXD2/P1.1
;       MOV     P_SW2,#01H          ;RXD2_2/P5.6, TXD2_2/P5.7

        SJMP    $

        END
```

3.2.3 SPI switch

C Language code

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

#include "reg51.h"

sfr     P_SW1   =   0xa2;

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;
    P_SW1 = 0x00;               //SS/P1.4, MOSI/P1.5, MISO/P1.6, SCLK/P1.7
//  P_SW1 = 0x04;               //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
//  P_SW1 = 0x0c;               //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2
    while (1);

}
```

Assembly code

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

```
P_SW1           DATA    0A2H

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_SW1,#00H      ;SS/P1.4, MOSI/P1.5, MISO/P1.6, SCLK/P1.7
;               MOV     P_SW1,#04H      ;SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
;               MOV     P_SW1,#0CH      ;SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

                SJMP    $

                END
```

3.2.4 I2C switch

C Language code

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

#include "reg51.h"

sfr     P_SW2       =   0xba;

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
```

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

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 = 0x00;                   //SCL/P1.7, SDA/P1.6
//  P_SW2 = 0x10;                   //SCL_2/P2.5, SDA_2/P2.4
//  P_SW2 = 0x30;                   //SCL_4/P3.2, SDA_4/P3.3
    while (1);

}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

P_SW2   DATA    0BAH

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,#00H              ;SCL/P1.7, SDA/P1.6
;       MOV     P_SW2,#10H              ;SCL_2/P2.5, SDA_2/P2.4
;       MOV     P_SW2,#30H              ;SCL_4/P3.2, SDA_4/P3.3

        SJMP    $

        END
```

3.2.5 Master clock output switching

C Language code

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

#include "reg51.h"

#define CLKOCR (*(unsigned char volatile xdata *)0xfe00)

sfr P_SW2 = 0xba;

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

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;
```

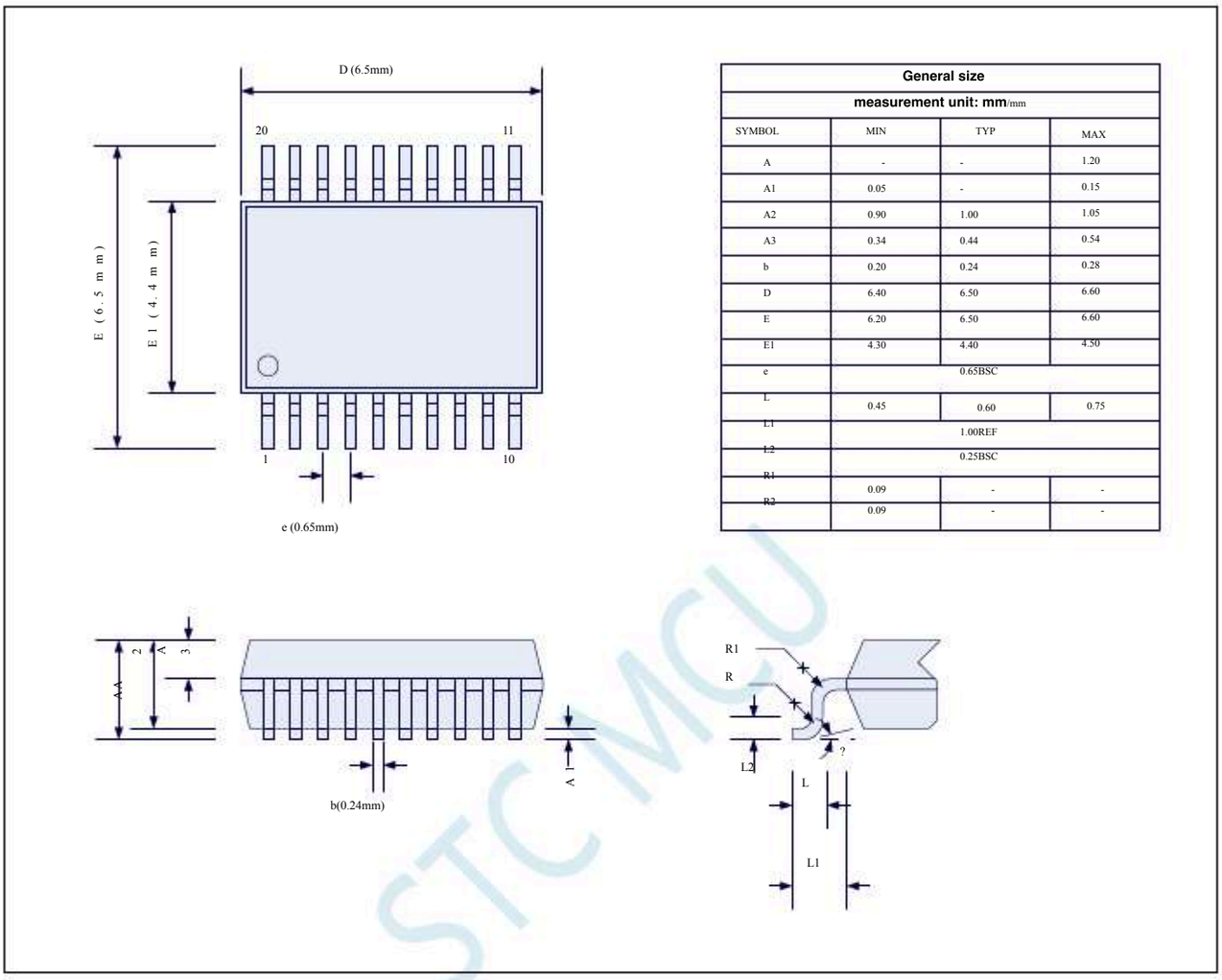
```
    P_SW2 = 0x80;
    CLKOCR = 0x04;                  //IRC/4 output via MCLKO/P5.4
//  CLKOCR = 0x84;                  //IRC/4 output via MCLKO_2/P5.6
    P_SW2 = 0x00;
    while (1);
}
```

Assembly code

```
; The test operating frequency is 11.0592MHz

P_SW2       DATA    0BAH

CLKOCR      EQU     0FE05H

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,#04H          ;IRC/4 output via MCLKO/P5.4
;           MOV     A,#84H          ;IRC/4 output via MCLKO_2/P5.6
            MOV     DPTR,#CLKOCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            SJMP    $

            END
```

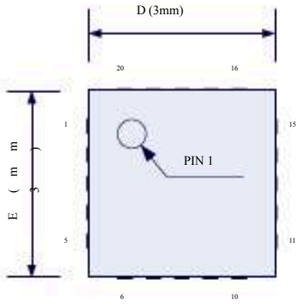
Package size diagram

44.1 TSSOP20 Package size diagram

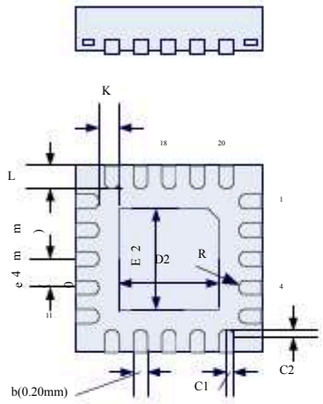


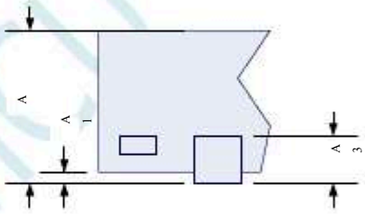
General size			
measurement unit: mm/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.20
A1	0.05	-	0.15
A2	0.90	1.00	1.05
A3	0.34	0.44	0.54
b	0.20	0.24	0.28
D	6.40	6.50	6.60
E	6.20	6.50	6.60
E1	4.30	4.40	4.50
e	0.65BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.09	-	-
R2	0.09	-	-

4.2 QFN20 Package size diagram (3mm*3mm)



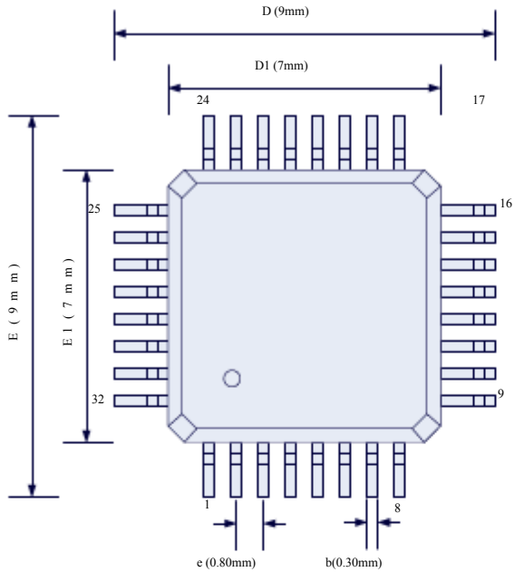
General size			
measurement unit: mm/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	2.90	3.00	3.10
E	2.90	3.00	3.10
D2	1.40	1.50	1.60
E2	1.40	1.50	1.60
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K	0.35REF		
R	0.085	-	-
C1	-	0.07	-
C2	-	0.07	-



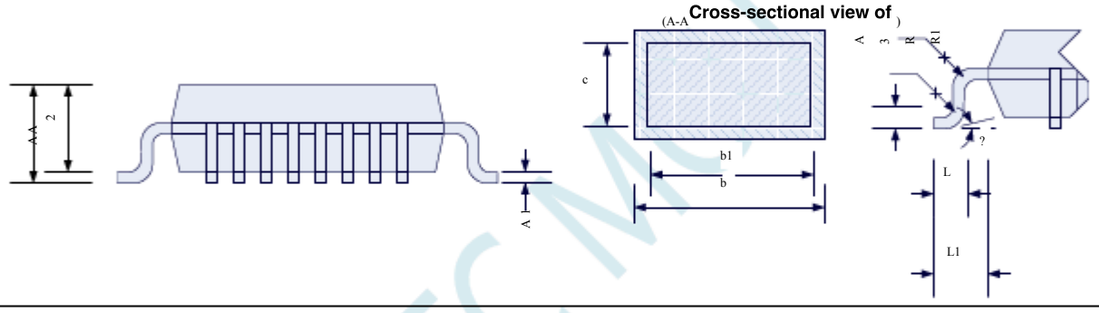


STC existing QFN20 The metal sheet (substrate) on the back of the encapsulated chip is not grounded inside the chip, and can be grounded or not on the board, It can also be ungrounded, which will not affect the performance of the chip

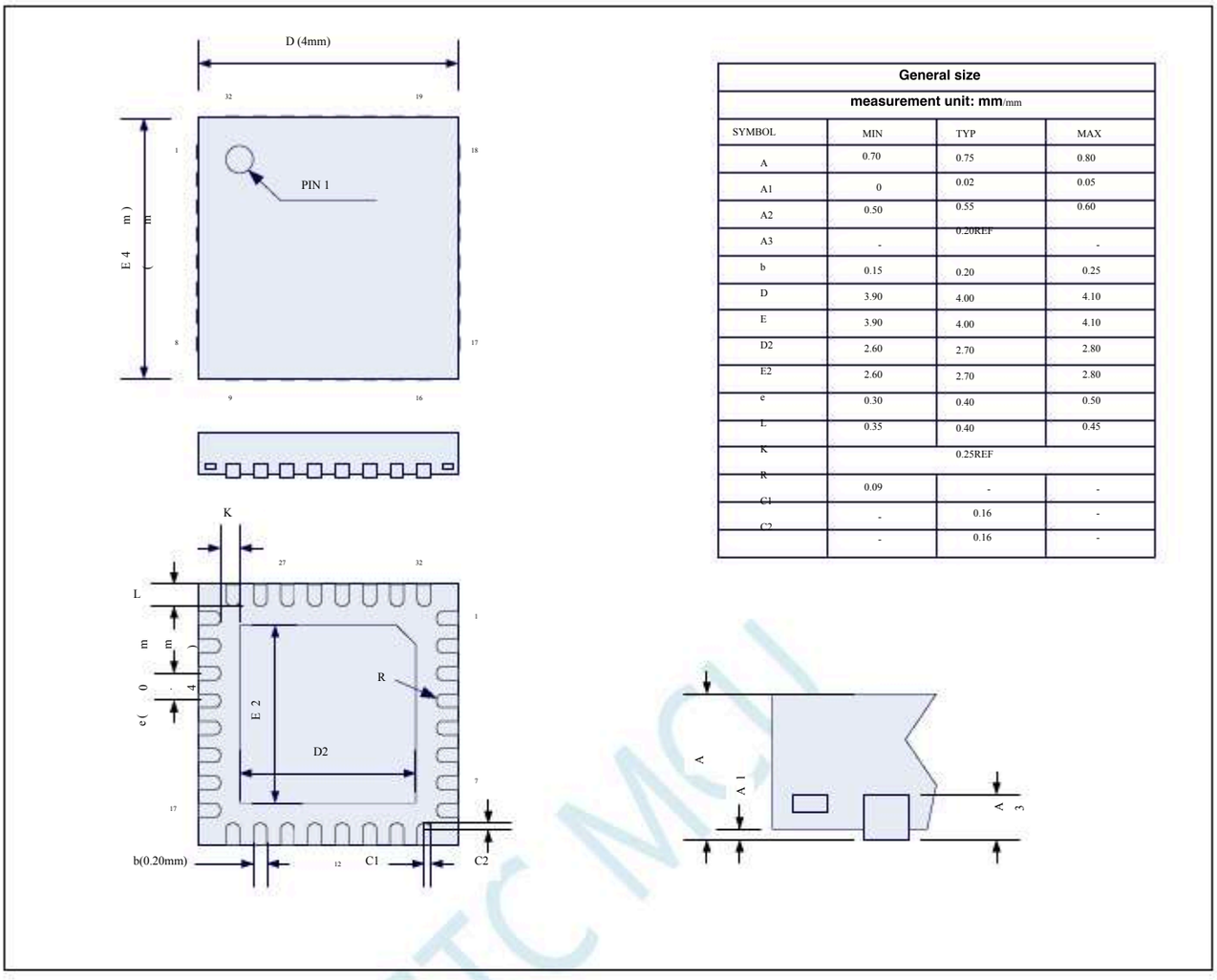
4.3 LQFP32 Package size diagram (9mm*9mm)



General size			
measurement unit: mm/mm			
SYMBOL	MIN	TYP	MAX
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
b1	0.30	0.35	0.40
b	0.31	0.37	0.43
c	-	0.127	-
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.70	0.80	0.90
L	0.43	-	0.71
L1	1.00REF		
R	0.25BSC		
R1	0.1	-	0.25
	0.1	-	-
?	0°	-	10°



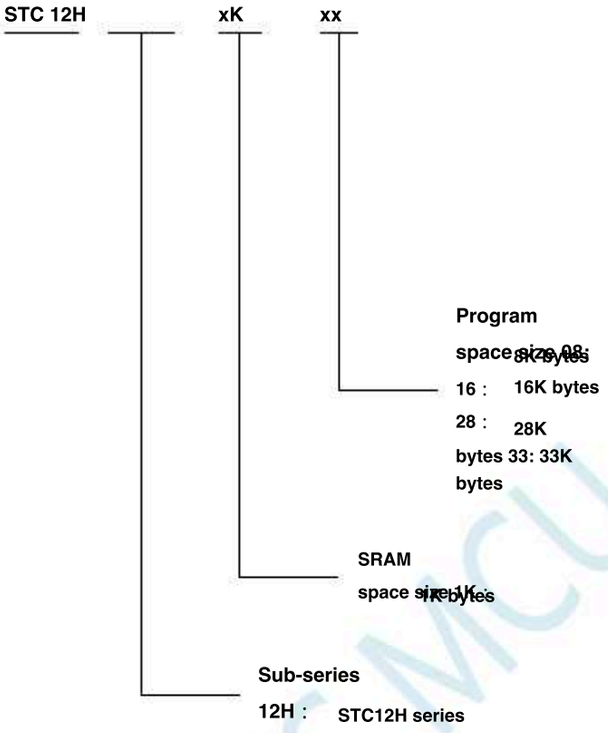
4.4 QFN32 Package size diagram (4mm*4mm)



General size			
measurement unit: mm/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.60	2.70	2.80
E2	2.60	2.70	2.80
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K	0.25REF		
R	0.09	-	-
C1	-	0.16	-
C2	-	0.16	-

STC existing QFN32 The metal sheet (substrate) on the back of the encapsulated chip is not grounded inside the chip, and can be grounded on the board, It can also be ungrounded, which will not affect the performance of the chip

4.5 STC12H Naming rules for series of microcontrollers



5 Establishment of compilation and simulation development environment and download of STC MCU

5.1 Install Keil

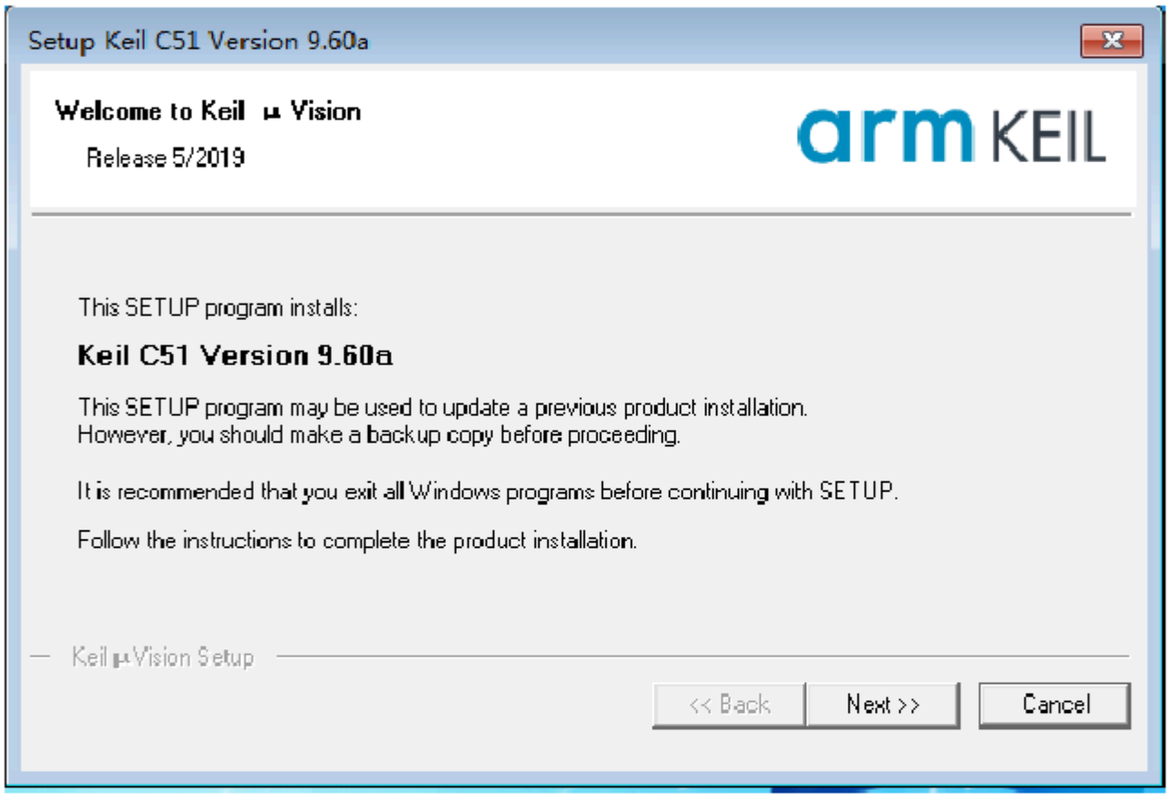
5.1.1 install Compilation environment C51

Log in first Keil Official website, download the latest version C51 The installation package, the download link is as follows :

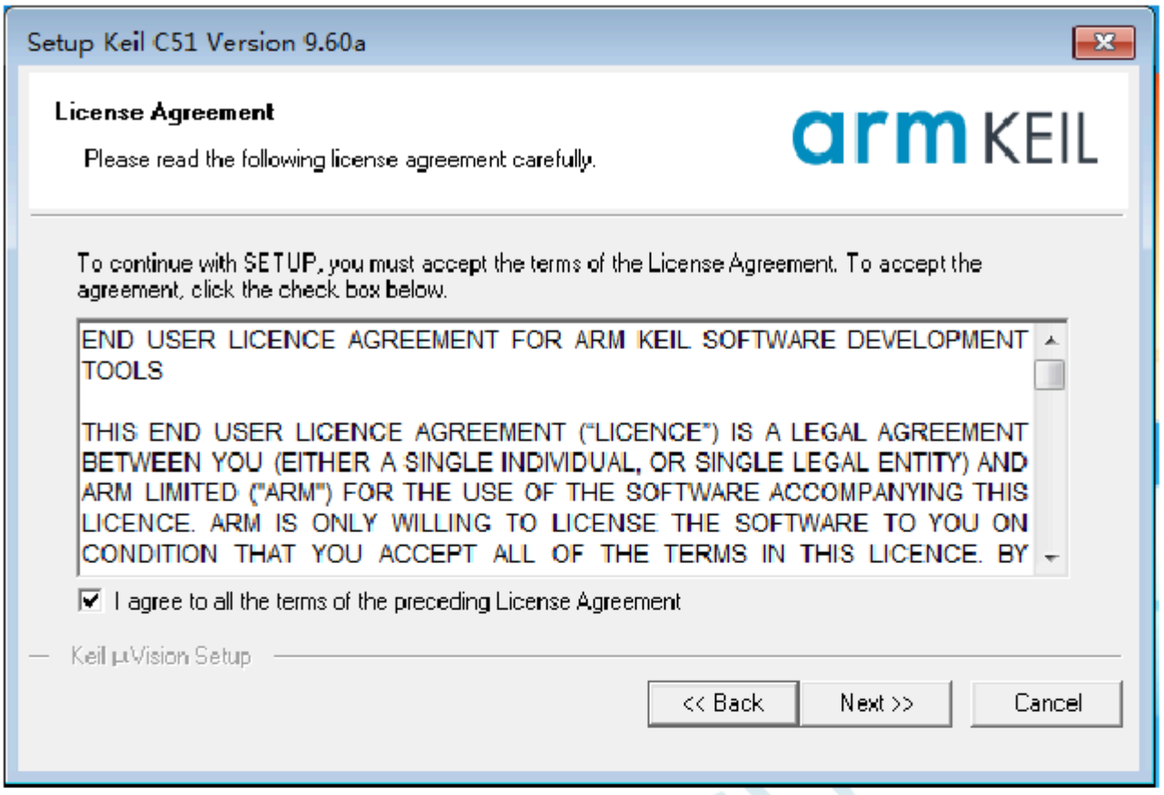
Keil Product Downloads

Fill in the information casually, click OK and enter the download page to download.

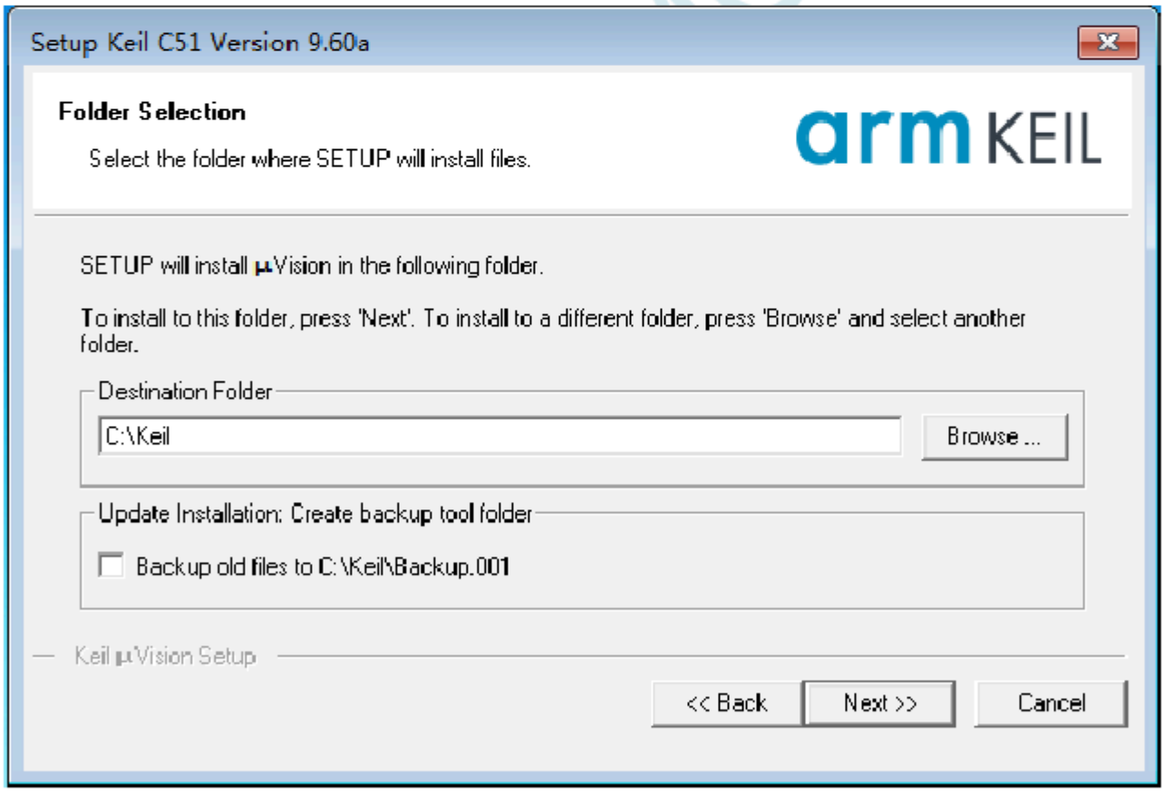
Double-click the downloaded installation package to start the installation and click " Next" :



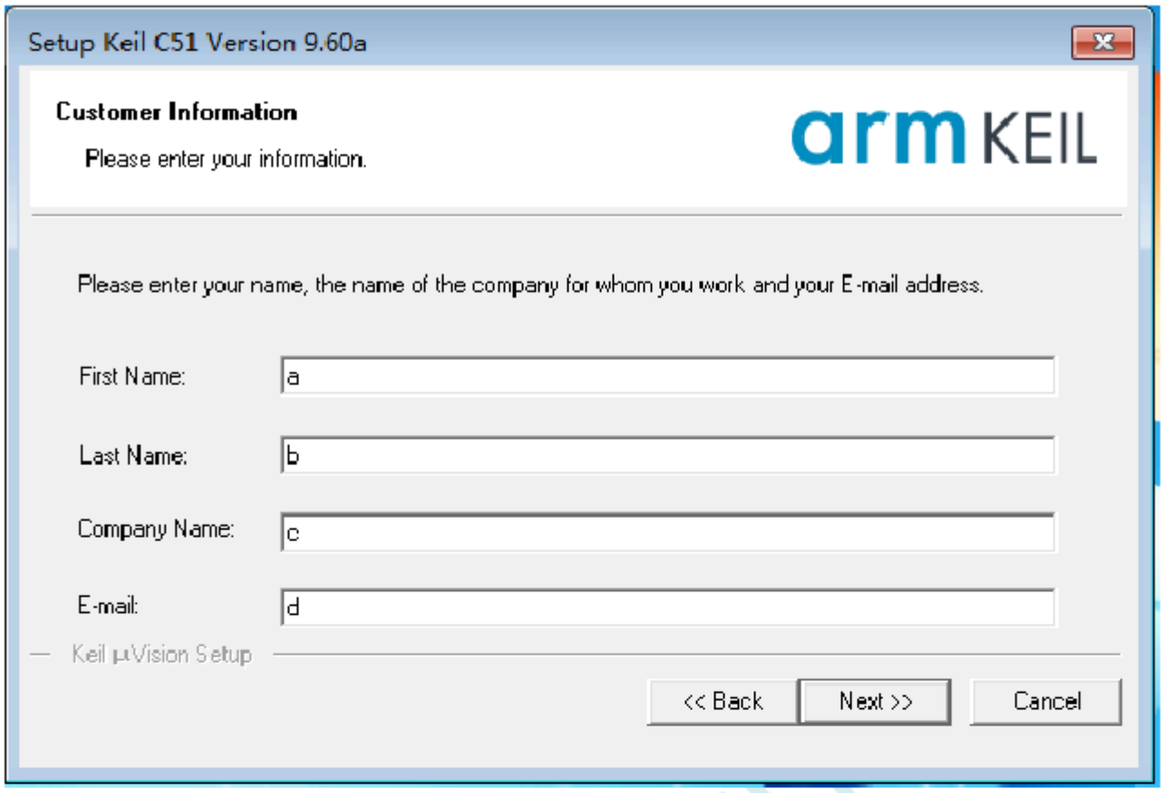
I Check" agree to all the terms of the preceding License Agreement "Then click" , Next" :



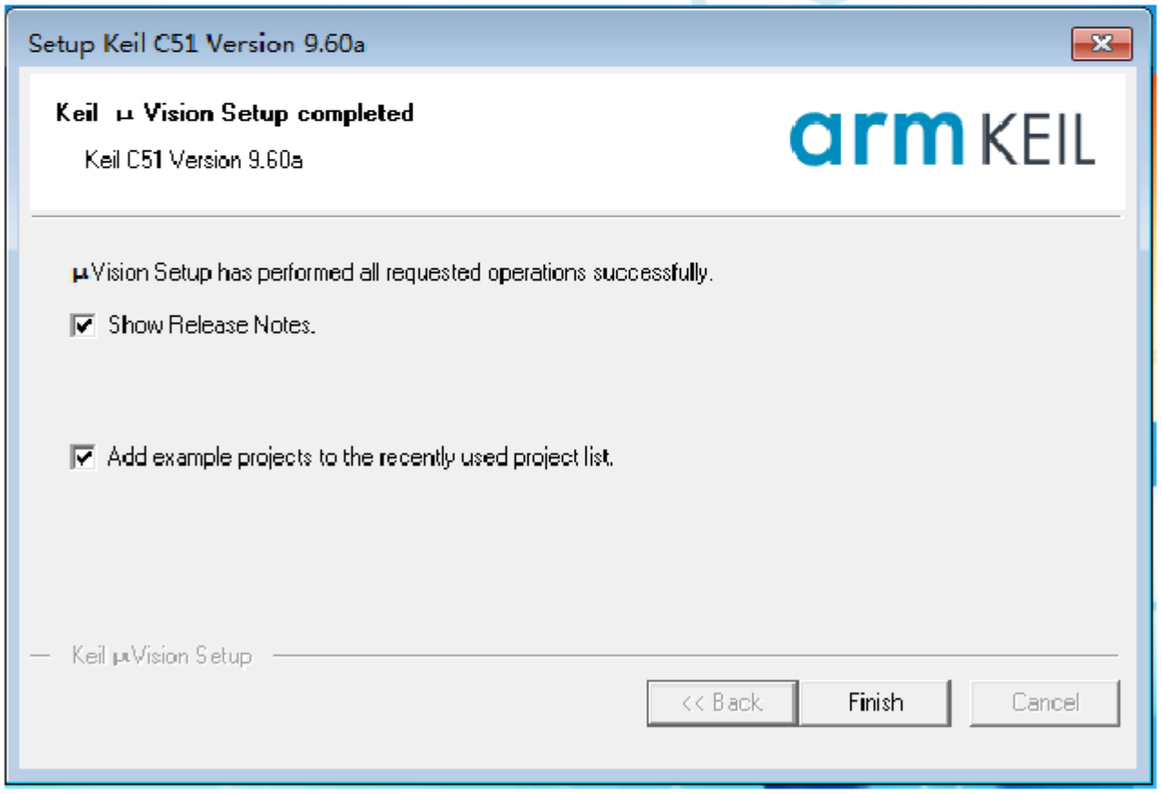
Select the installation directory and click " Next"



Fill in your personal information and click " Next"

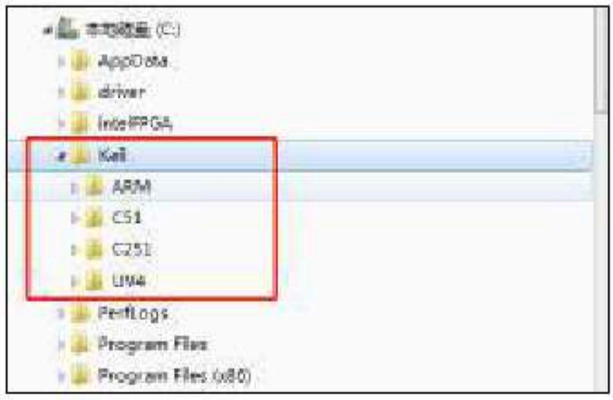


After the installation is complete, click "

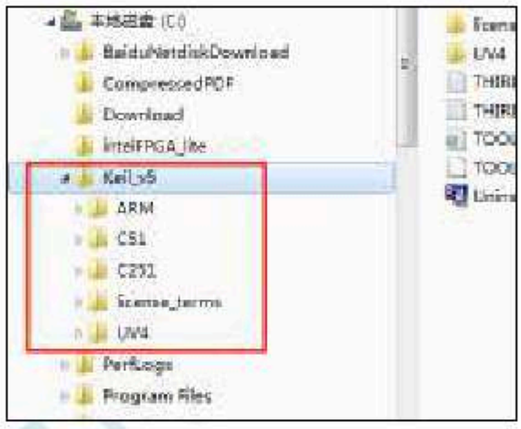


How to install at the same time Keil of C51 and C251 MDK 5.1.2 C251

Old version of Keil The installation directory of the software defaults to C:\Keil , C51、C251 and MDK Will be installed separately in C:\Keil Under the table of contents of C51、C251 and ARM In the directory, as shown in the figure below.



Under the new Keil The installation directory of the software defaults to C:\Keil_v5 In the C51 and MDK Will be installed separately in C:\Keil_v5 version of the directory C51 and ARM directory, as shown in the figure below.



Whether it is a new version or an old version , C51、MDK and It is installed in a different directory, and there will be no conflicts. Software ha also 3 Each software is carried out separately, and the software that has been installed and set up before will not be changed because of the So when installing, you only need to install it in the default way. The software will handle it automatically.

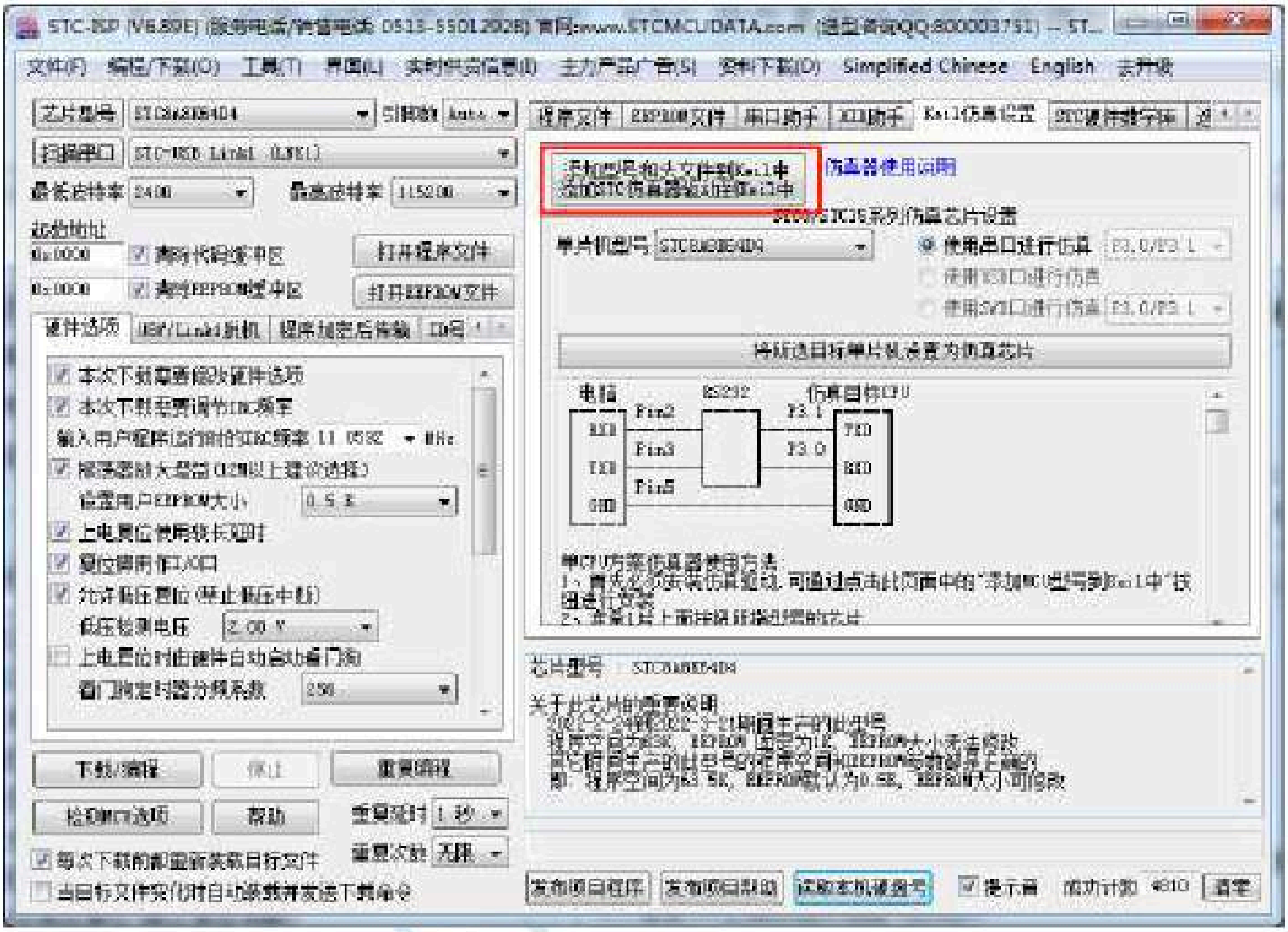
Keil

5.2 Add model and header files to Keil

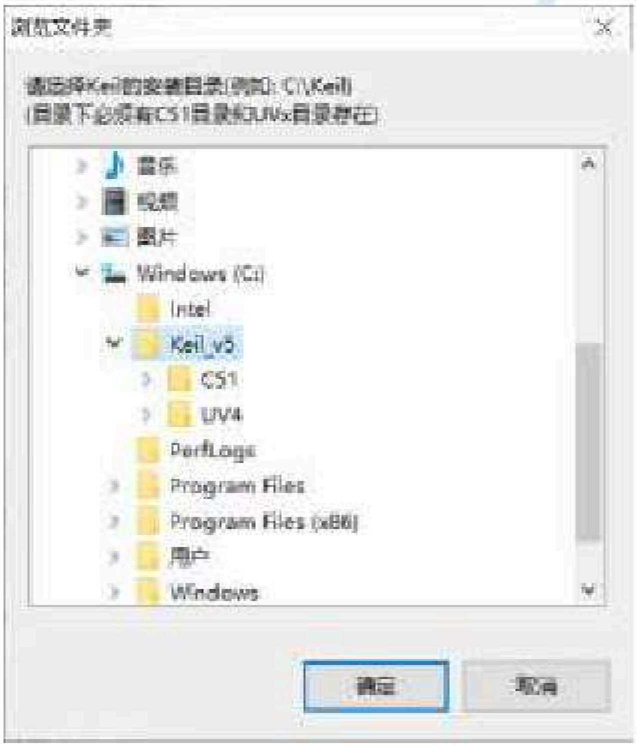
Use to open Keil Need to install before STC Simulation driver. The installation steps of the simulation driver are as follows：

the file STC of ISP Download the software, and then in the "On the right side of the software" Click "Add model and header file to Keil" on the "Simulation Settings" page

first to Keil Add in STC The emulator is driven to "Medium" button：



After pressing, the following screen will appear：



Locate the directory to Keil The installation directory of the software, and then determine. After the installation is successful, the following prompt

It means that the driver is installed correctly

The header file is copied to Under the installation directory "In the catalog use in the code by default" "or"

```
#include <STC32G. H>
```

```
C:\KEIL\C51\INC\STC
```

```
#include "STC32G. H"
```

"All inclusive can be used correctly

5.3 STC How to use header files in MCU programs

c In language include usage

#include A command is a kind of preprocessing command. The preprocessing command can insert other source code content into the speci

There are two ways to specify the insertion header file :

#include <File name .h>

#include "File name .h"

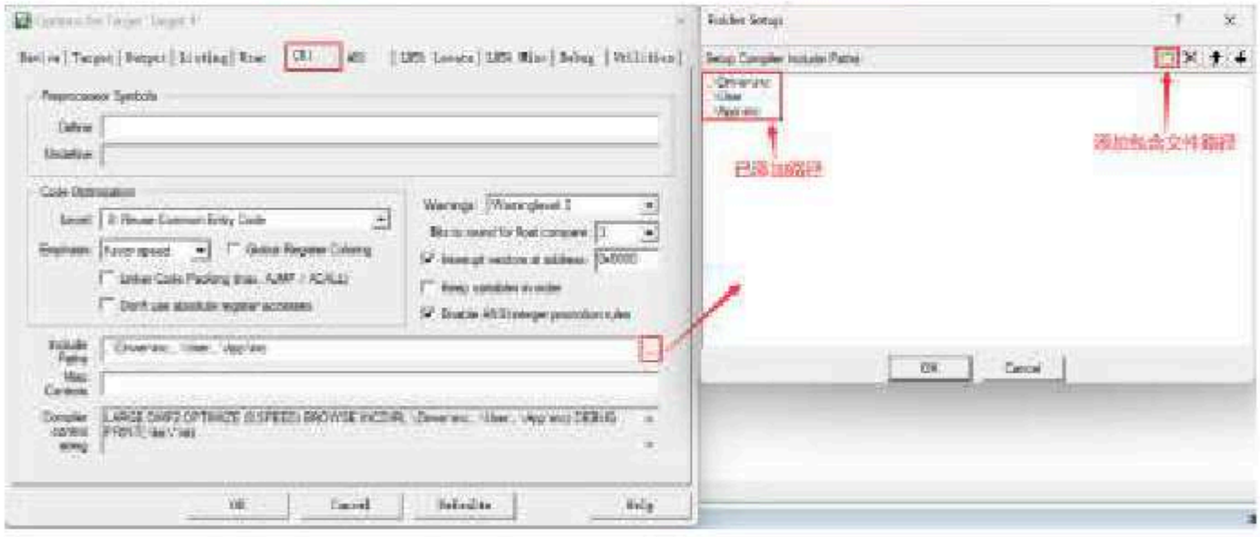
Use angle brackets "" The difference is that the search path for the header file is different : <> And

Use angle brackets double quotation marks, the compiler will look for the header file under the system path ;

Use double quotation marks , The compiler first looks for the header file in the current directory, and if it is not found, look for it in the system path.

1 Path setting method :

pass keil Set up the interface and add the path containing the file :



After adding, directly use the #include "File name .h" You can include the required files, and the compiler will automatically go to the above following to find the included files

when calling. In this case, using double Contains the header file, the compiler first looks for the header file in the current directory, if it is not found, quotation marks will go to Set the page to find, if you don't have it yet, go to the system path (Note: The system path is the compiler installation location to store the keil

of the catalog)

2 Path setting method :

Add an absolute path before including the file name, for example :

#include "E:\xxxx\xxxx\File name .h"

#include "E:/xxxx/xxxx/File name .h"

3 Path setting method :

Add a relative path before including the file name, for example :

#include ". . \comm\File name . h"

#include ". . /comm/File name . h"

Among them ". " Refers to the upper-level directory, and the above path refers to the upper-level directory that contains the file in the current direct

In assembly language The usage is similar to the language, will with brackets Include file :

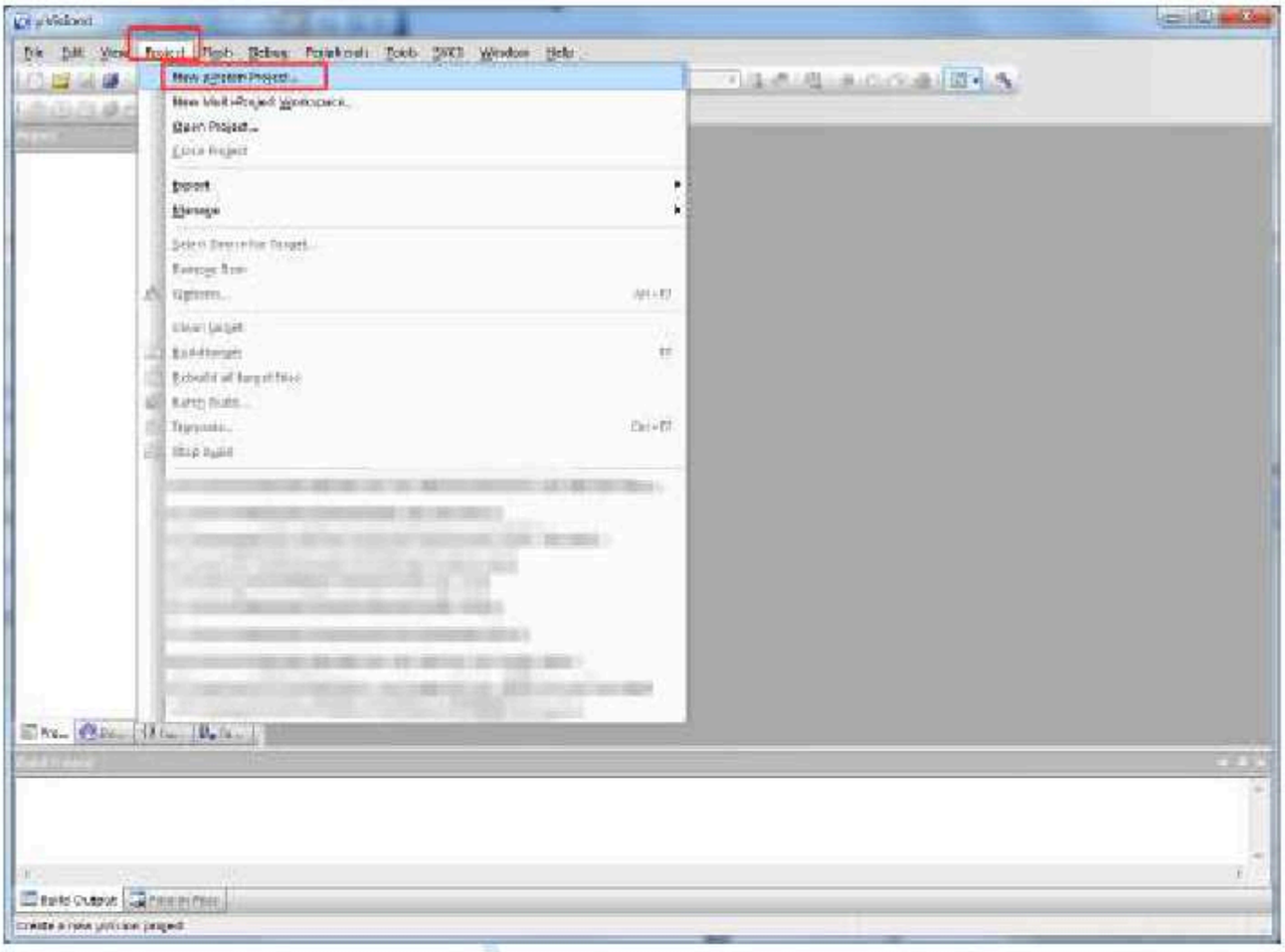
$include (. . /. . /comm/STC8H. INC)

The above command indicates the file to be included The upper-level directory of the upper-level directory of the current directory

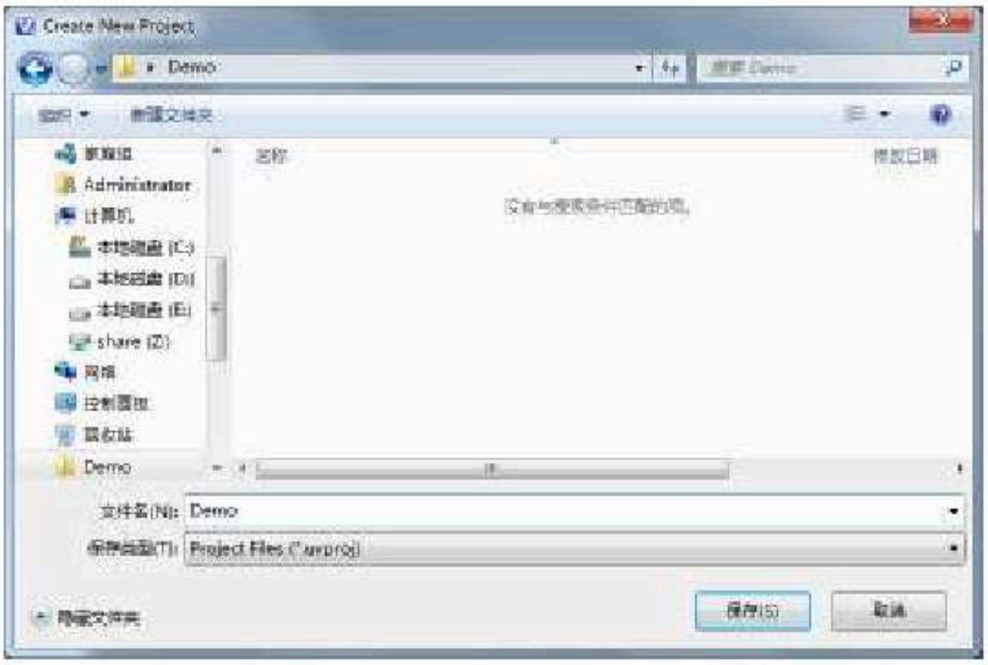
5.4 Create a new project and project settings

5.4.1 set the project path and project name

open Keil Software, and click " Project "In the menu" New uVision Project . . . "item

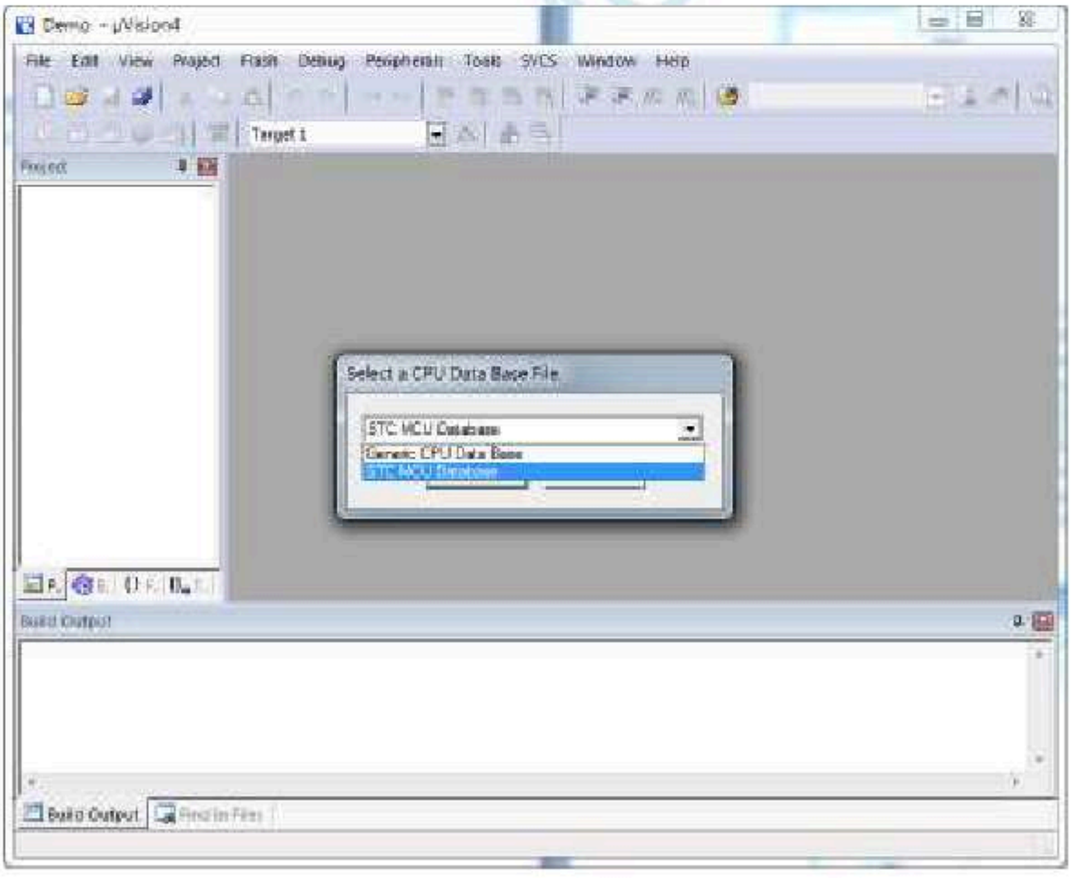


Locate the directory in the prepared project folder and enter the project name (for example： Demo)

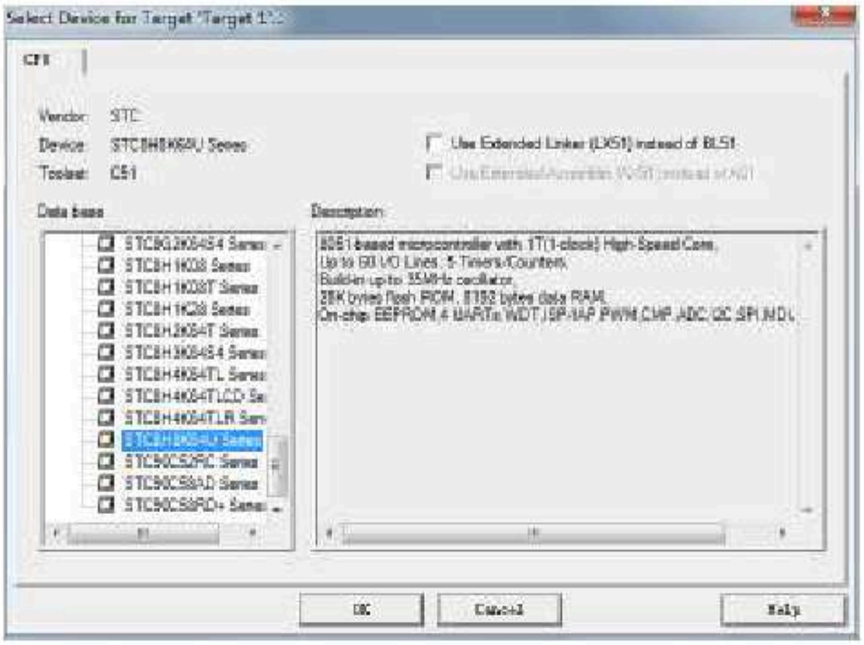


5.4.2 Select the target MCU (STC8H8K64U)

In the pop-up "Select a CPU Data Base File" model (Select a CPU Data Base File **"Select in the window"** STC MCU Database **"**

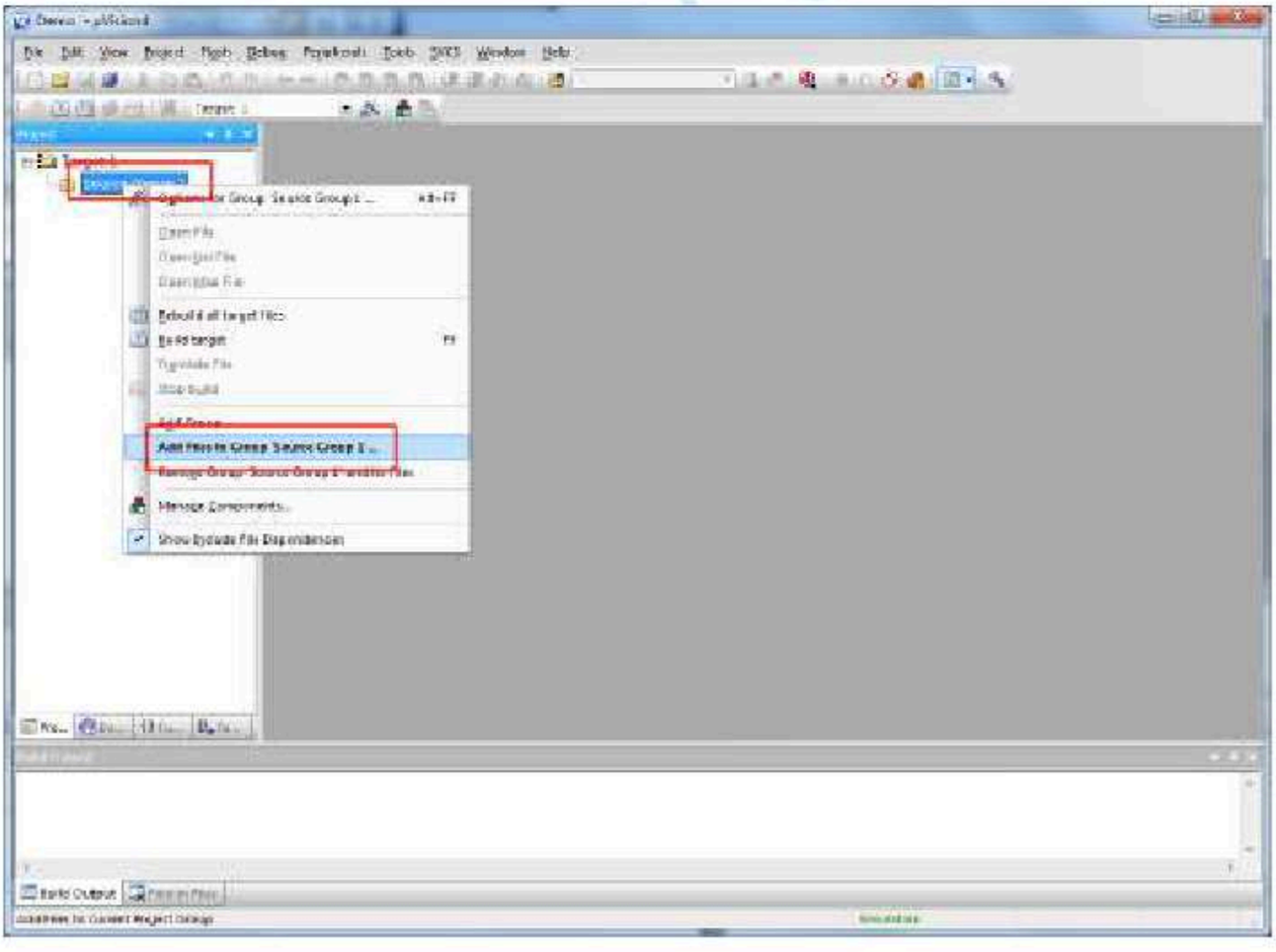


in" Select Device for Target . . . "Select the correct target MCU model in the window (for example : STC8H8K64U)

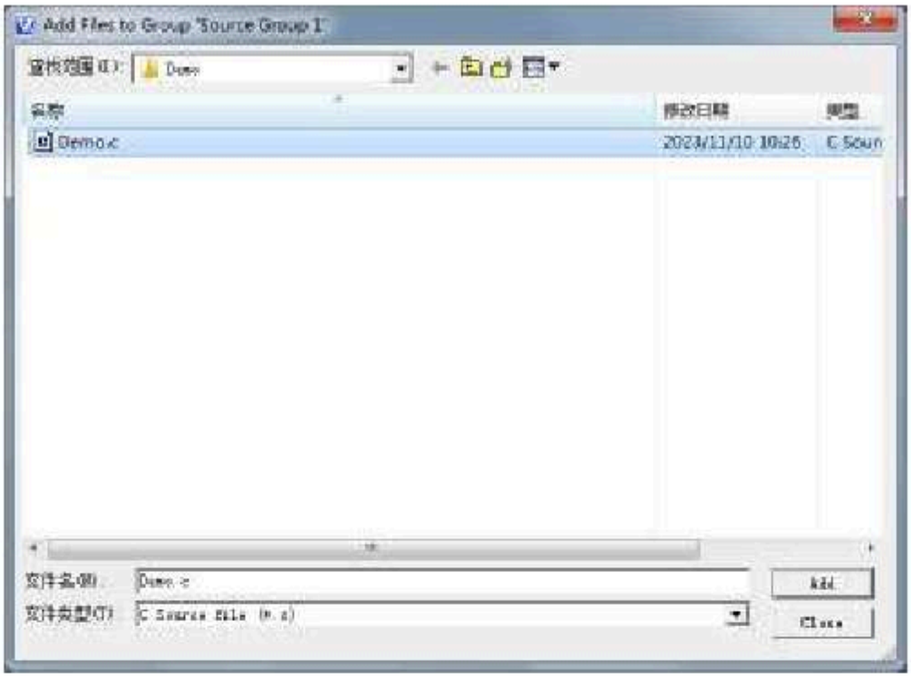


5.4.3 Add source code files to the project

As shown in the figure below, Click the right mouse button on the icon where "is located, and select "in the context menu" in " ' Source Group 1'. . ."

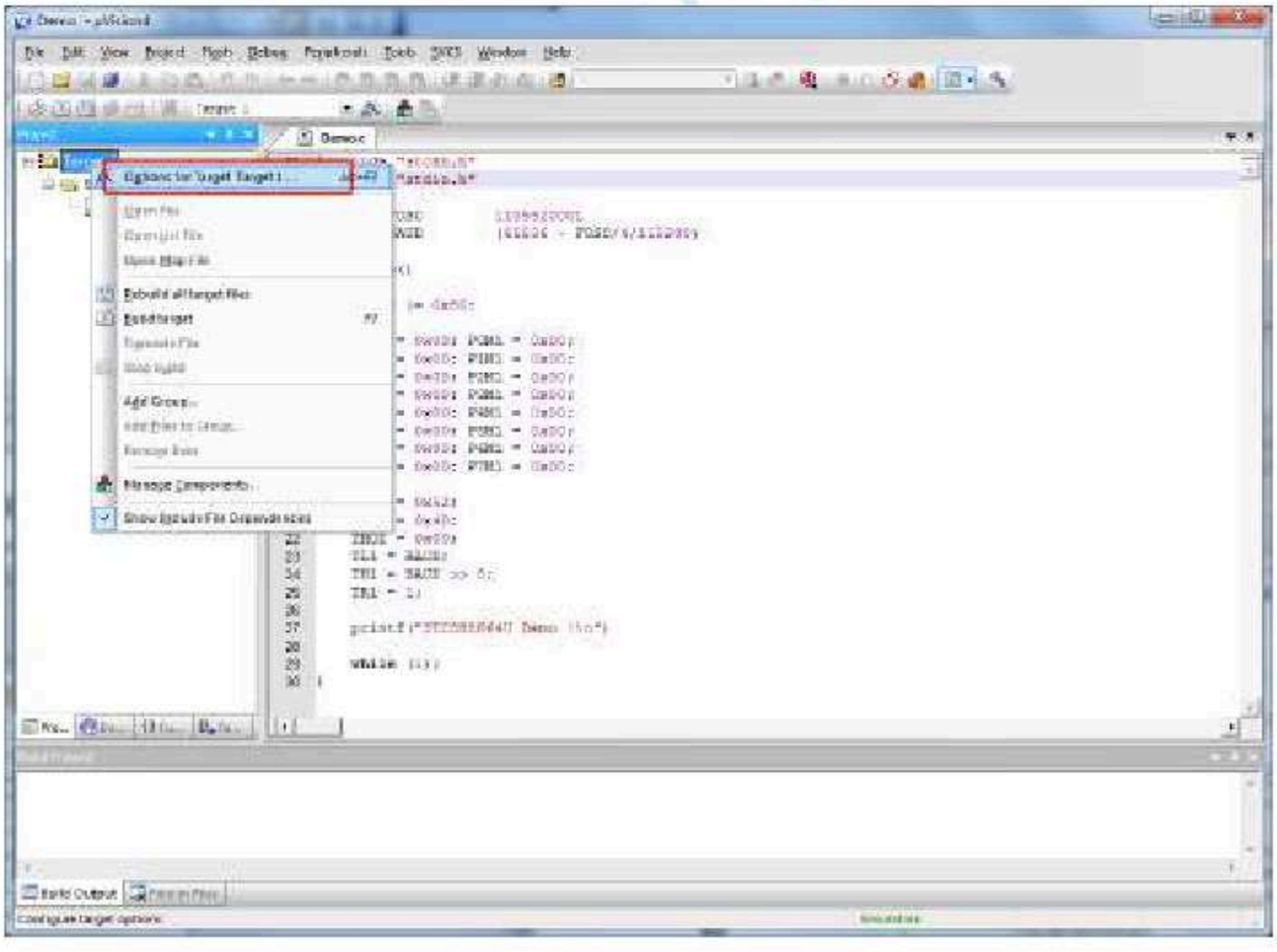


Select the edited code file to add to the project



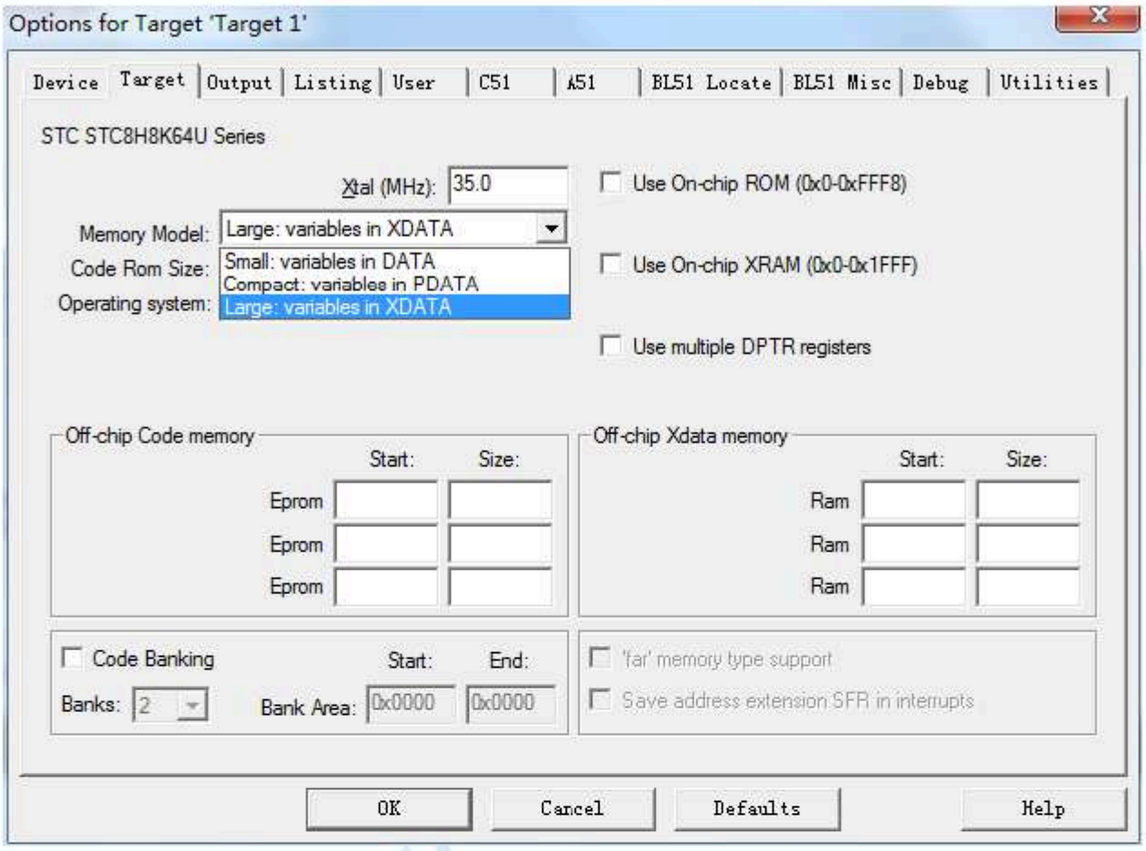
5.4.4 Setup items (settings" Memory Model")

As shown in the figure below, Right-click the icon where it is located and select "Options for Target 'Target 1'..." (Options for Target 1)



" Optional" that Options for Target ' Target 1' "Select in the window" Target "Options page, in" Memory Model "In the drop-down option pops up Small "Mode or" Large "mode.

in Keil In the software" Memory Model "There are the following Warning



The comparison of various modes is shown in the table below :

Memory Model	Default variable type (data storage)	Memory size	Address range
Small pattern	data	Byte	D:00 ~ D:7F
Compact pattern	pdata	byte	X:0000 ~ X:00FF
Large pattern	xdata	64K Bytes (theoretical value)	X:0000 ~ X:FFFF

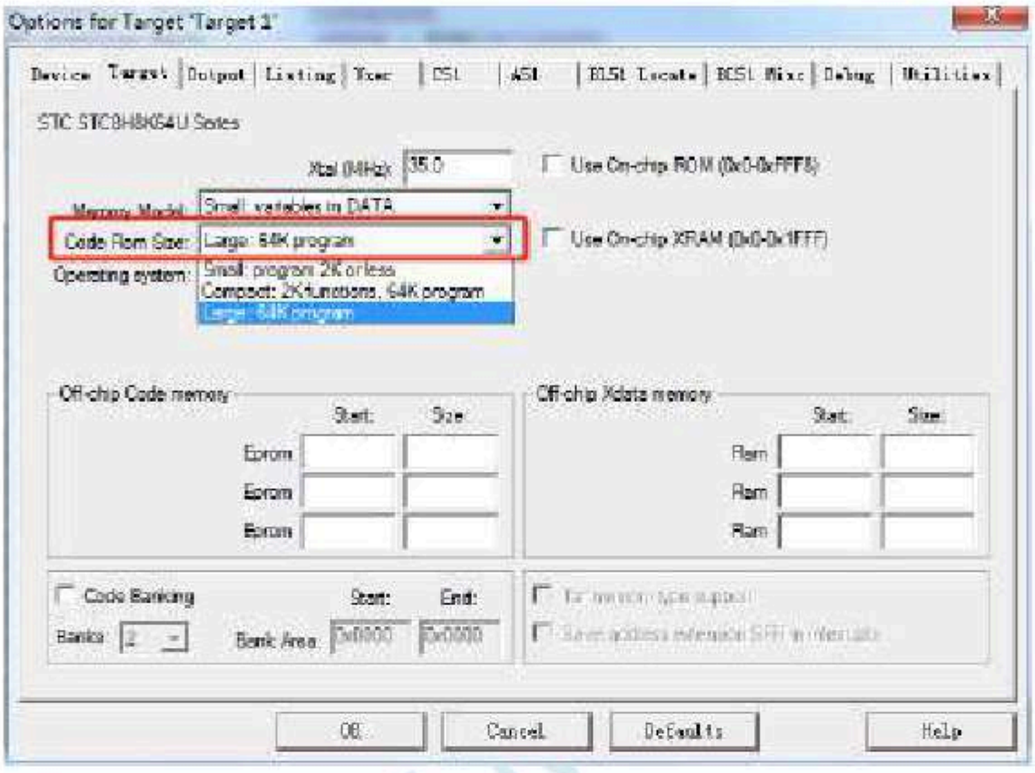
In order to achieve relatively high efficiency, it is general Mode, when the compiler appears "

TOO LARGE "When there is an error, you need to manually pass some relatively large arrays Area (example such as : char xdata buffer[256];)

3 Setup items ("Code Rom Size" Choose Large) 5.4.5

Code Rom Size in "Select from the drop-down options of" "mode" Large

8051 of Code size mode, in Keil In the environment, there are the modes shown in the figure below：



The comparison of various modes is shown in the table below：

Code Rom Size	/Jump to call instruction	Code size limit for a single function/module/The code size of the total code size of the file	
Small pattern	AJMP/ACALL	2K	2K
Compact pattern	Use of internal module code AJMP/ACALL Use of external module code LJMP/LCALL	2K	64K
Large pattern	LCALL/LJMP	64K	64K

5 Setup items (HEX 5.4.6 File format settings)

" Options for Target ' Target 1' "Select in the window" "On the options page, check "Output Create HEX File "Option.

After completing the above settings, click the compile button as shown in the figure below with the mouse. If there is no error in the code, it c HEX file

5.5 How to Specify absolute addresses for variables, table data, and fun

5.5.1 Keil C51 C51 In, how do variables specify absolute addresses

The syntax is as follows:

Data type storage type variable name] _at_ ;Absolute address

Example of specifying an absolute address variable in a region :

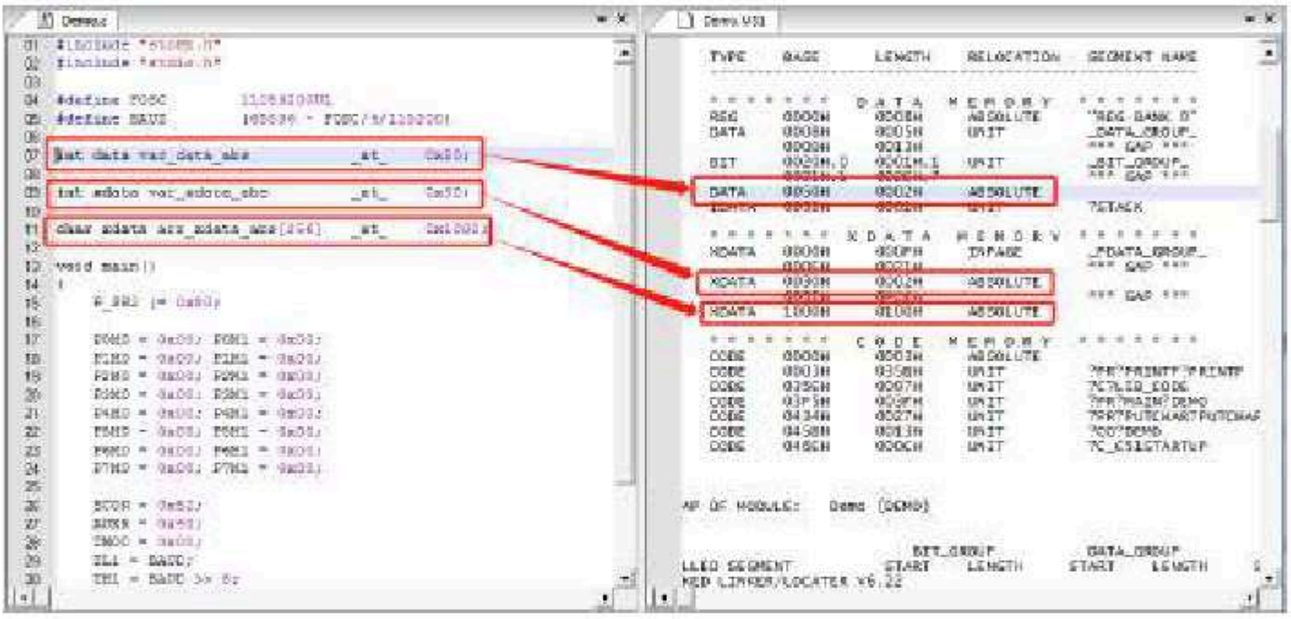
```
data

    data var_data_abs
int _at_ 0x50;
```

Example of specifying an absolute address variable in a region :

```
    xdata
xdata
data
int var_xdata_abs _at_ 0x30;
char xdata arr_xdata_abs[256] _at_ 0x1000;
```

Example of specifying an absolute address variable in a region :

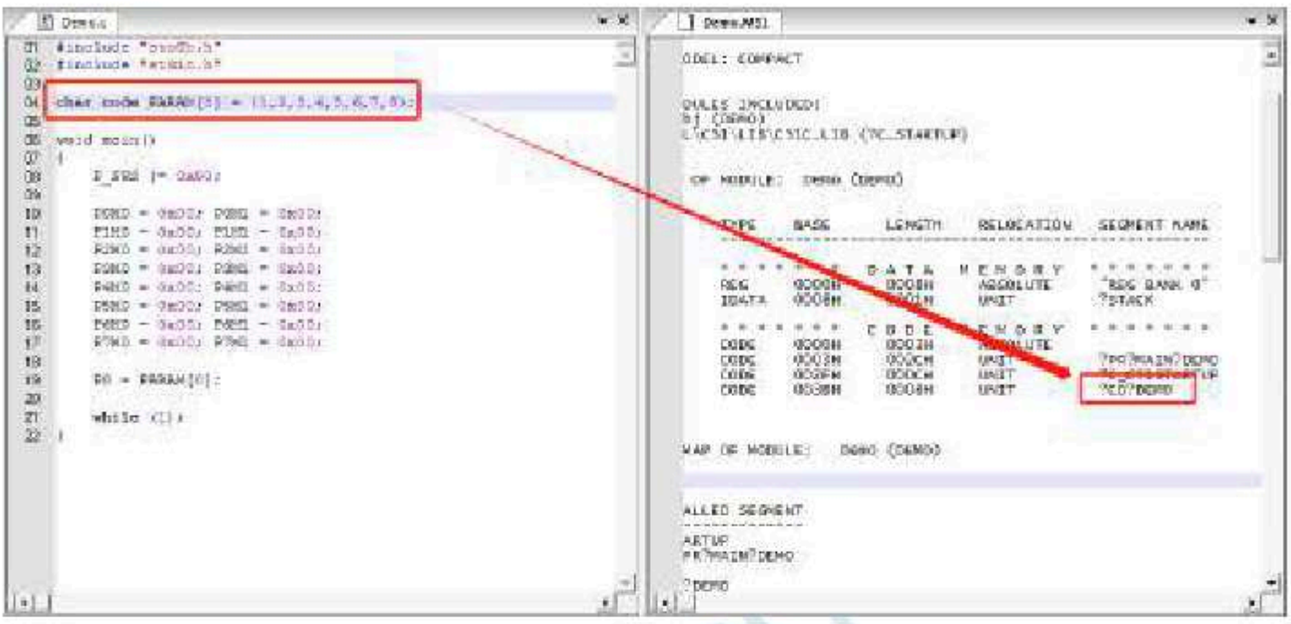
After the compilation is complete, the address is assigned as shown in the figure below :



In, how to specify the absolute address of the table data

C51 **Keil C51 5.5.2**

First define the table data in the program, and after the compilation is successful, get the link symbol of the table (as pictured below is) It is impossible to specify the absolute address of the table data directly in the program, it needs to be achieved by the following method



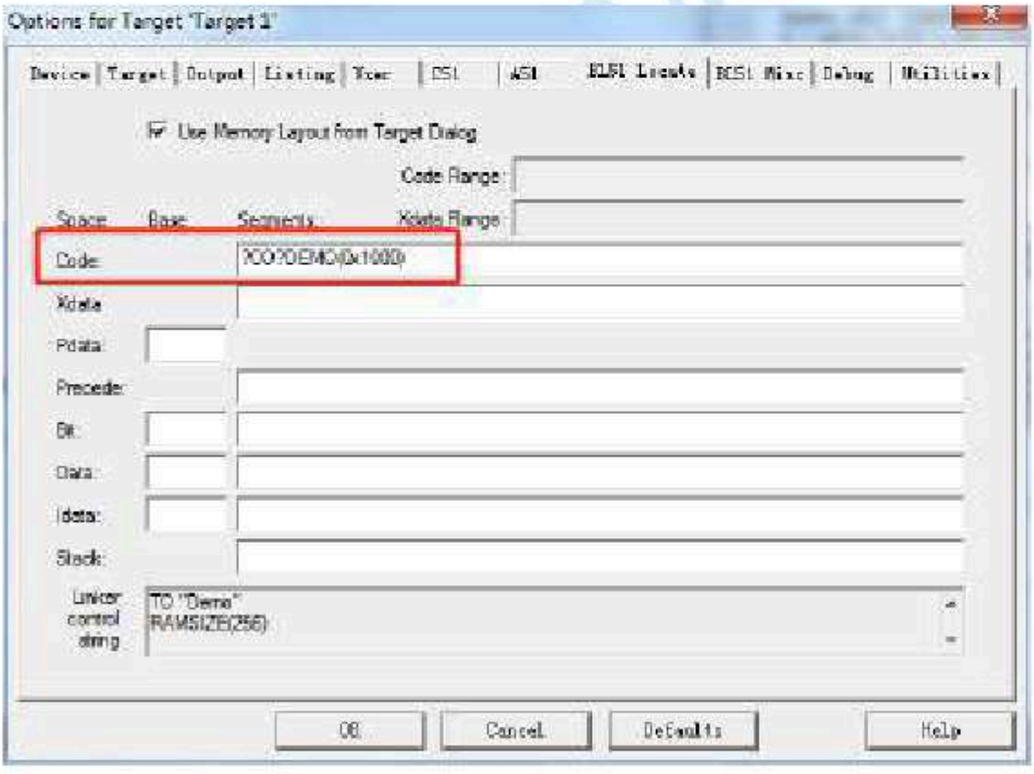
Next, open "in" in the project settings item "Settings page

C51 Locate

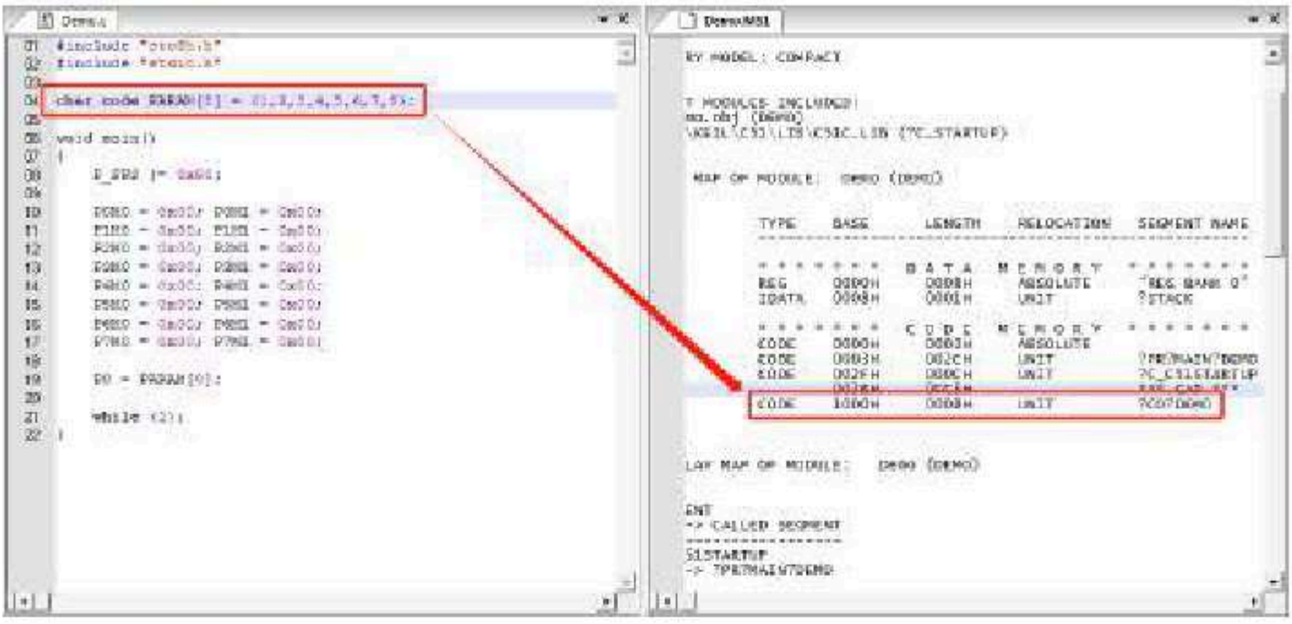
code **In one column, follow : (Format, enter the absolute address**

As shown in the figure below, assign the table data to the code area of the absolute address

Link (Range Absolute Address)



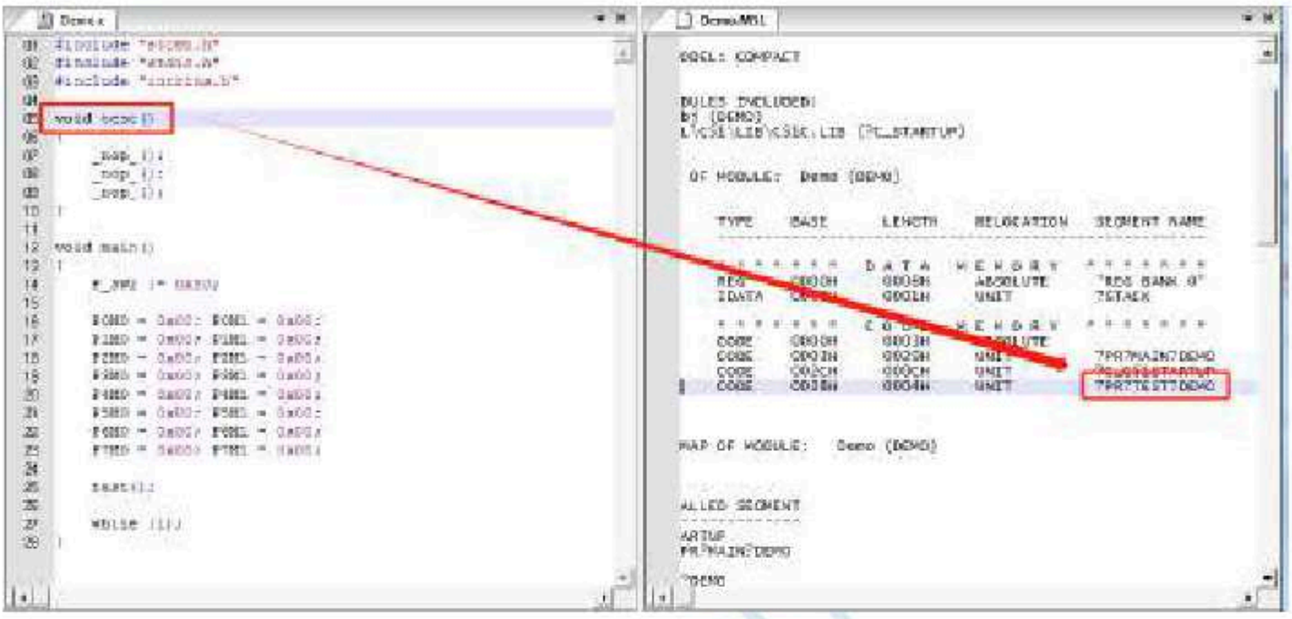
After the setup is complete, compile again, and the table data can be linked to the specified absolute address, as shown in the figure below. :


Demo.c
#include "reg51.h"
#include "intrins.h"
char code PARAM[5] = {1,2,3,4,5,6,7,8};
void main()
{
P_SW2 |= 0x80;
P0M0 = 0x00; P0M1 = 0x00;
P1M0 = 0x00; P1M1 = 0x00;
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
P4M0 = 0x00; P4M1 = 0x00;
P5M0 = 0x00; P5M1 = 0x00;
P6M0 = 0x00; P6M1 = 0x00;
P7M0 = 0x00; P7M1 = 0x00;
P0 = PARAM[0];
while (1);
}
Demo.M51
BY MODEL : COMPACT
T MODULES INCLUDED:
Demo.obj (DEMO)
\KEIL\C51\LIB\C51C.LIB (?C_STARTUP)
MAP OF MODULE: DEMO (DEMO)
TYPE BASE LENGTH RELOCATION SEGMENT NAME
* * * * * * * * D A T A M E M O R Y * * * * * * * *
REG 0000H 0008H ABSOLUTE "REG BANK 0"
IDATA 0008H 0001H UNIT ?STACK
* * * * * * * * C O D E M E M O R Y * * * * * * * *
CODE 0000H 0003H ABSOLUTE
CODE 0003H 002CH UNIT ?PR?MAIN?DEMO
CODE 002FH 0009H UNIT ?C_C51STARTUP
CODE 0038H 0008H UNIT ?CO?DEMO
LAY MAP OF MODULE: DEMO (DEMO)
?PR?MAIN?DEMO
?C_C51STARTUP

In, how does the function specify the absolute address?

C51 **Keil C51 5.5.3**

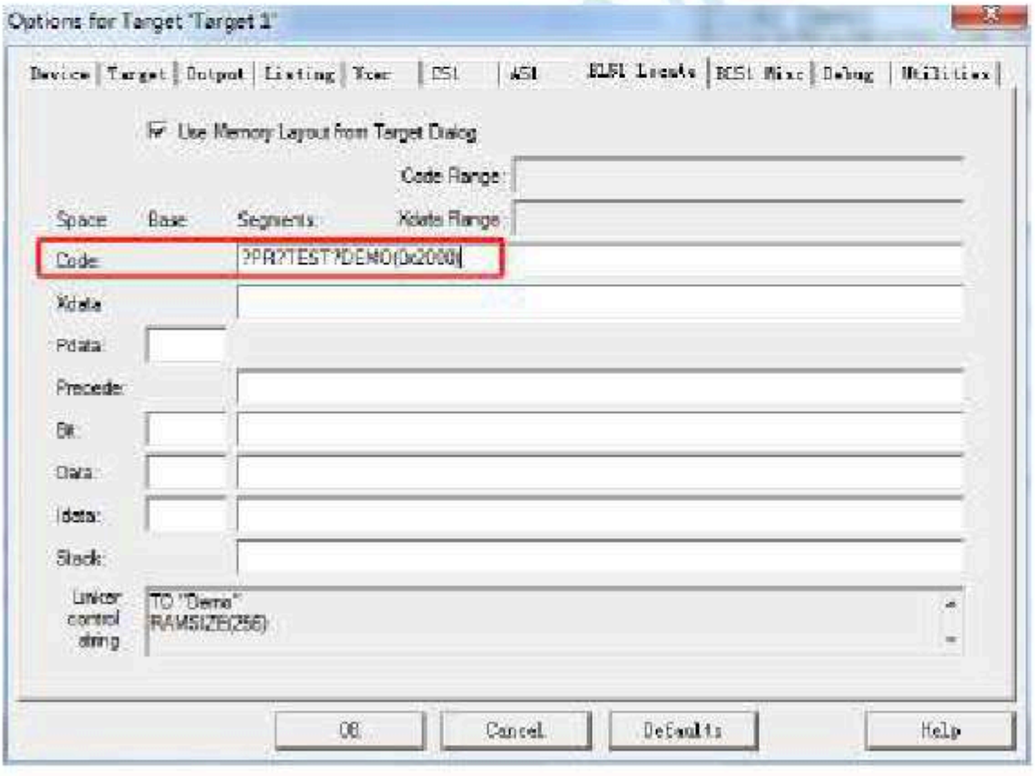
First, after the program is written, the function code is defined, and after the compilation is successful, it is impossible to specify the absolute address of the function correctly in the program. It needs to be implemented in the following ways:



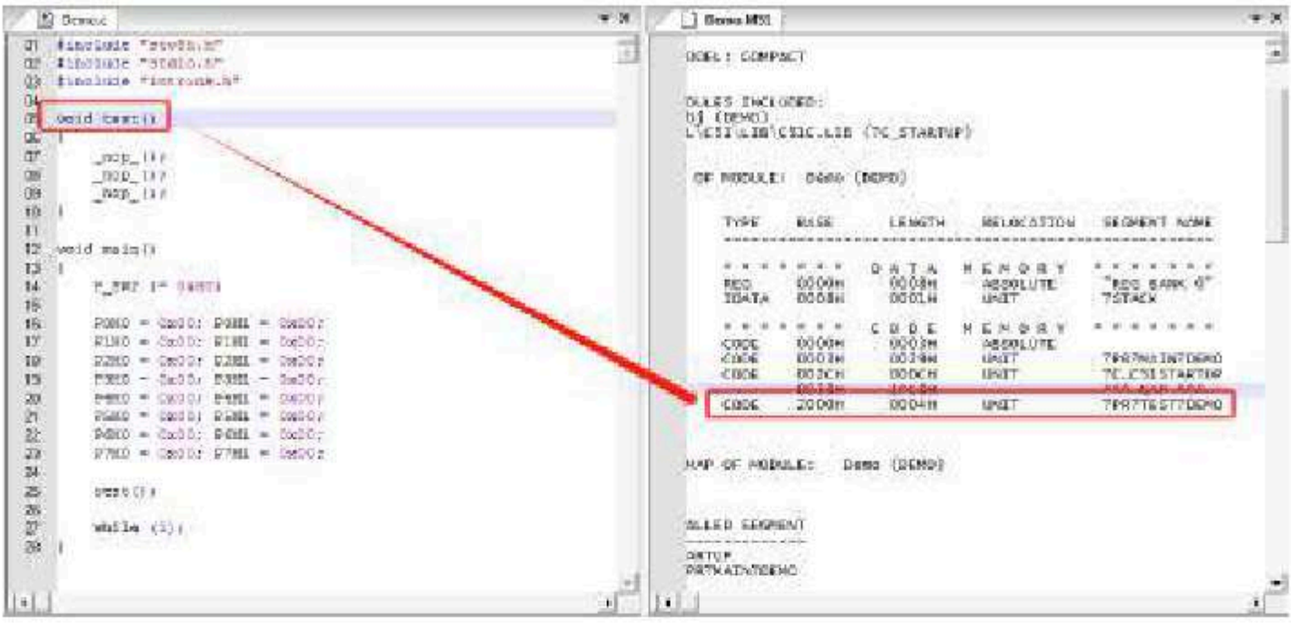
Next, open "in" in the project settings item "Settings page

the project settings item follow : Format, enter the absolute address

As shown in the figure below, assign the function of the absolute address

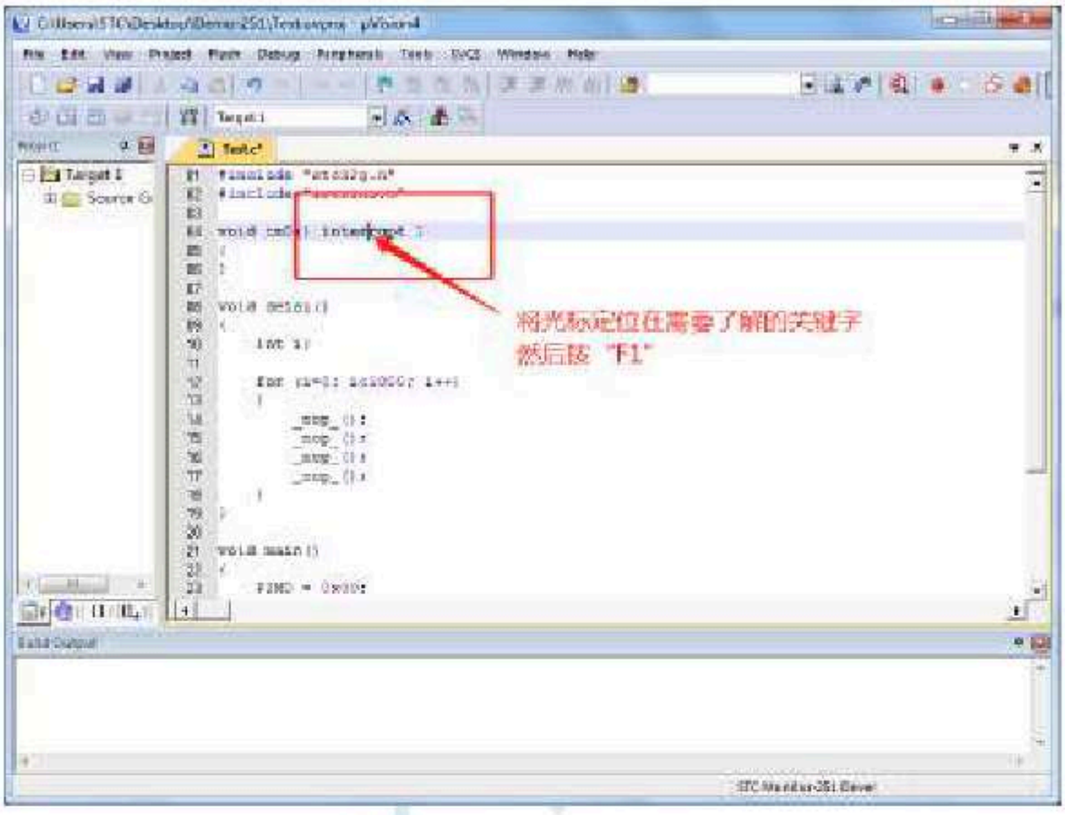


After the setup is complete, compile again, and the function can be linked to the specified absolute address, as shown in the figure below :

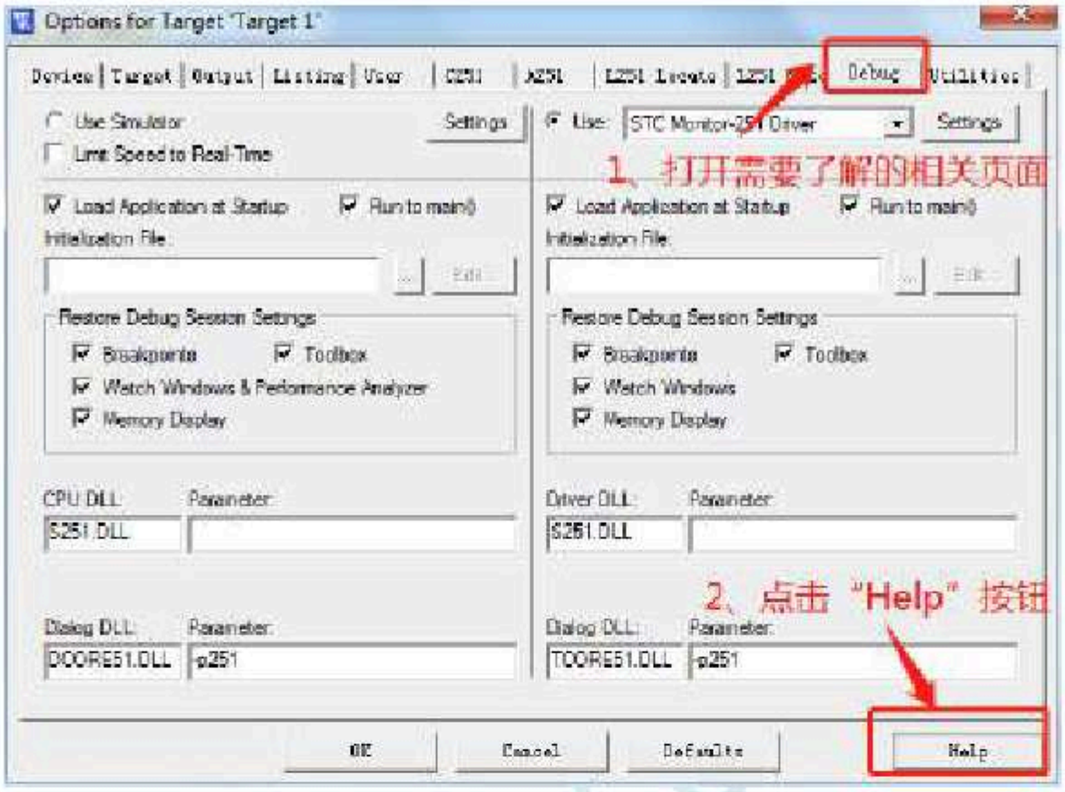

#include "reg51.h"
#include "STC15.H"
#include "intrins.h"
void Delay();
nop();
nop();
nop();
void main()
{
P0M0 = 0x00; P0M1 = 0x00;
P1M0 = 0x00; P1M1 = 0x00;
P2M0 = 0x00; P2M1 = 0x00;
P3M0 = 0x00; P3M1 = 0x00;
P4M0 = 0x00; P4M1 = 0x00;
P5M0 = 0x00; P5M1 = 0x00;
P6M0 = 0x00; P6M1 = 0x00;
P7M0 = 0x00; P7M1 = 0x00;
Delay();
while (1);
}
MODEL: COMPACT
RULES ENCLOSED:
0] (DEMO)
L:\KEIL\C51\LIB\C51C.LIB (?C_STARTUP)
OF MODULE: Demo (DEMO)
TYPE BASE LENGTH RELOCATION SEGMENT NAME
* * * * * * * * D A T A M E M O R Y * * * * * * * *
REG 0000H 0008H ABSOLUTE "REG BANK 0"
IDATA 0008H 0001H UNIT ?STACK
* * * * * * * * C O D E M E M O R Y * * * * * * * *
CODE 0000H 0003H ABSOLUTE
CODE 0003H 0024H UNIT ?PR?INT0DEMO
CODE 0027H 0006H UNIT ?C_C51STARTUP
CODE 2000H 0004H UNIT ?PR?T6ST7DEMO
MAP OF MODULE: Demo (DEMO)
BLOCK SEGMENT
DATA?P
?PR?KAIMUDEMO

5.6 Keil An easy way to get help in the software

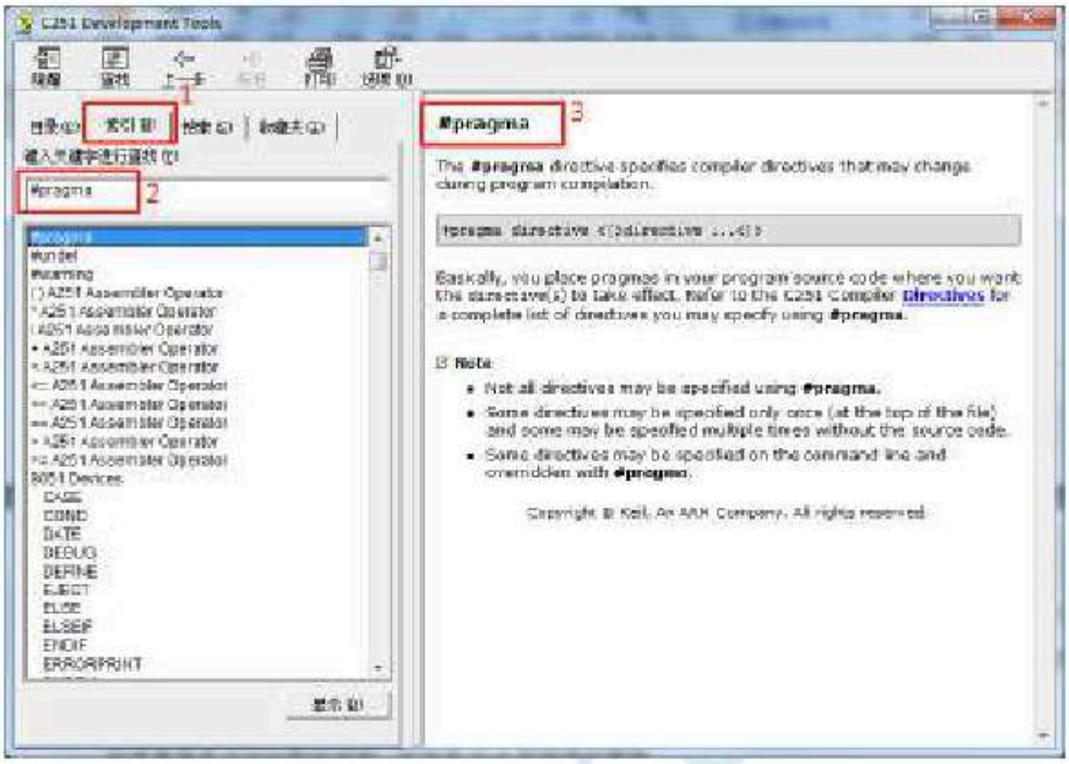
The software provides a very complete help file. For general software use and programming issues, the help of the software is basically Can be resolved. As shown below :



If you need to understand the relevant settings in the project settings, you can get help as shown in the figure below.



In addition, you can directly enter the content you want to know in the help window. For example, if you need to know how to set a special pragma in the program, you can enter "In the search box as shown in the figure below.



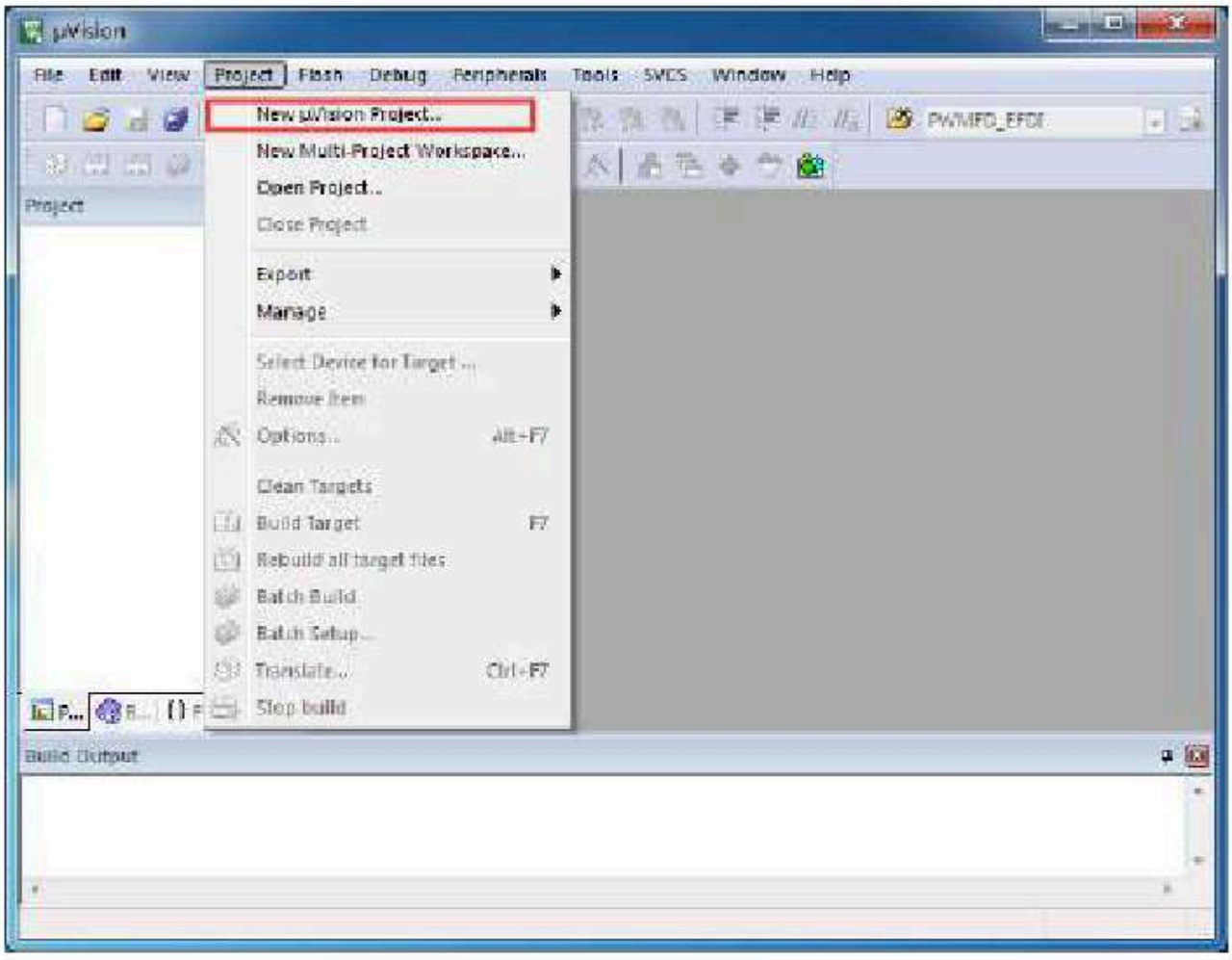
If you need more detailed help, you can log in Keil Make an inquiry on the official website

5.7 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

The method of creating a multi-file project is as follows：

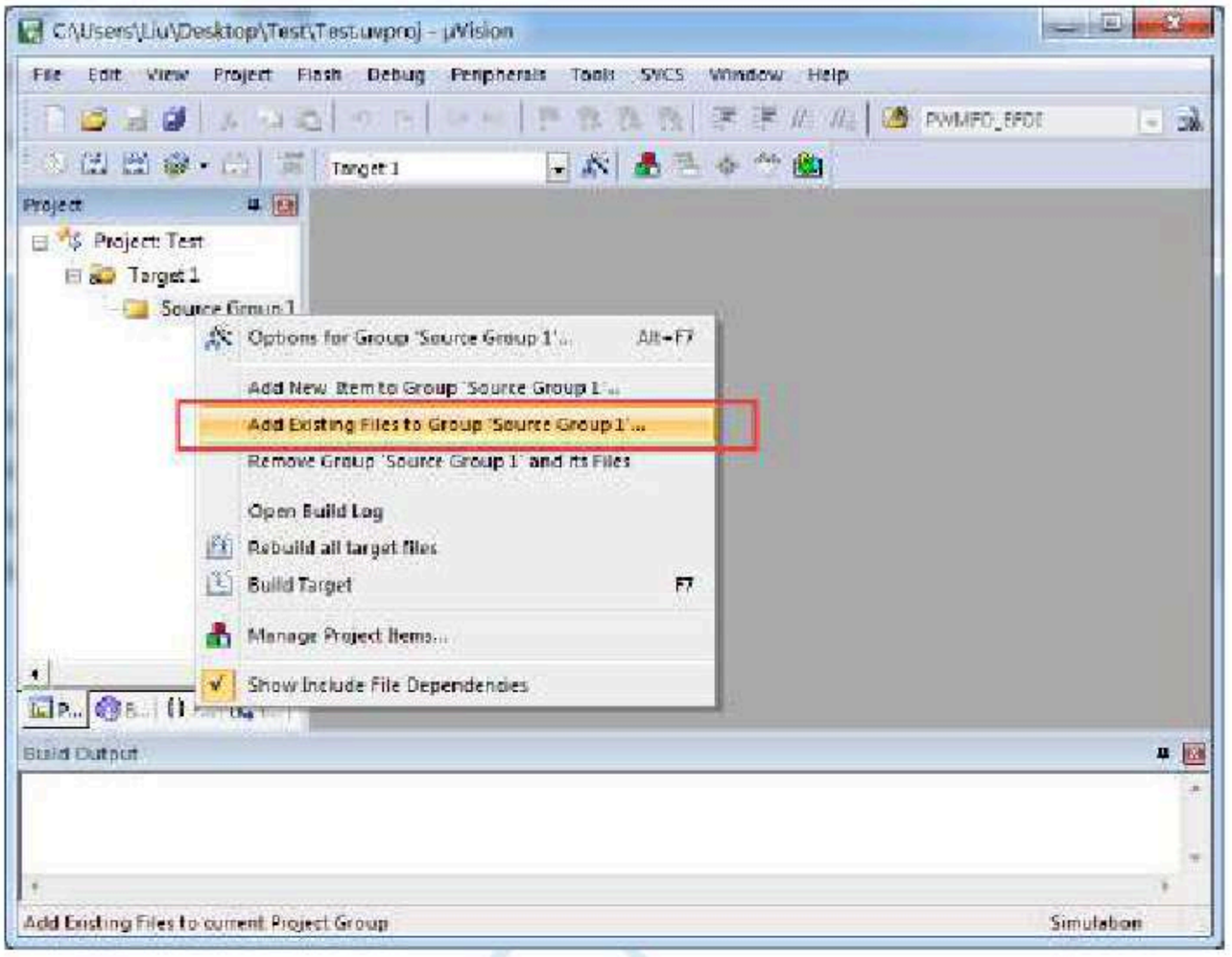
1, Open first Keil, in the menu" Project "Medium selection" New µVision Project . . ."



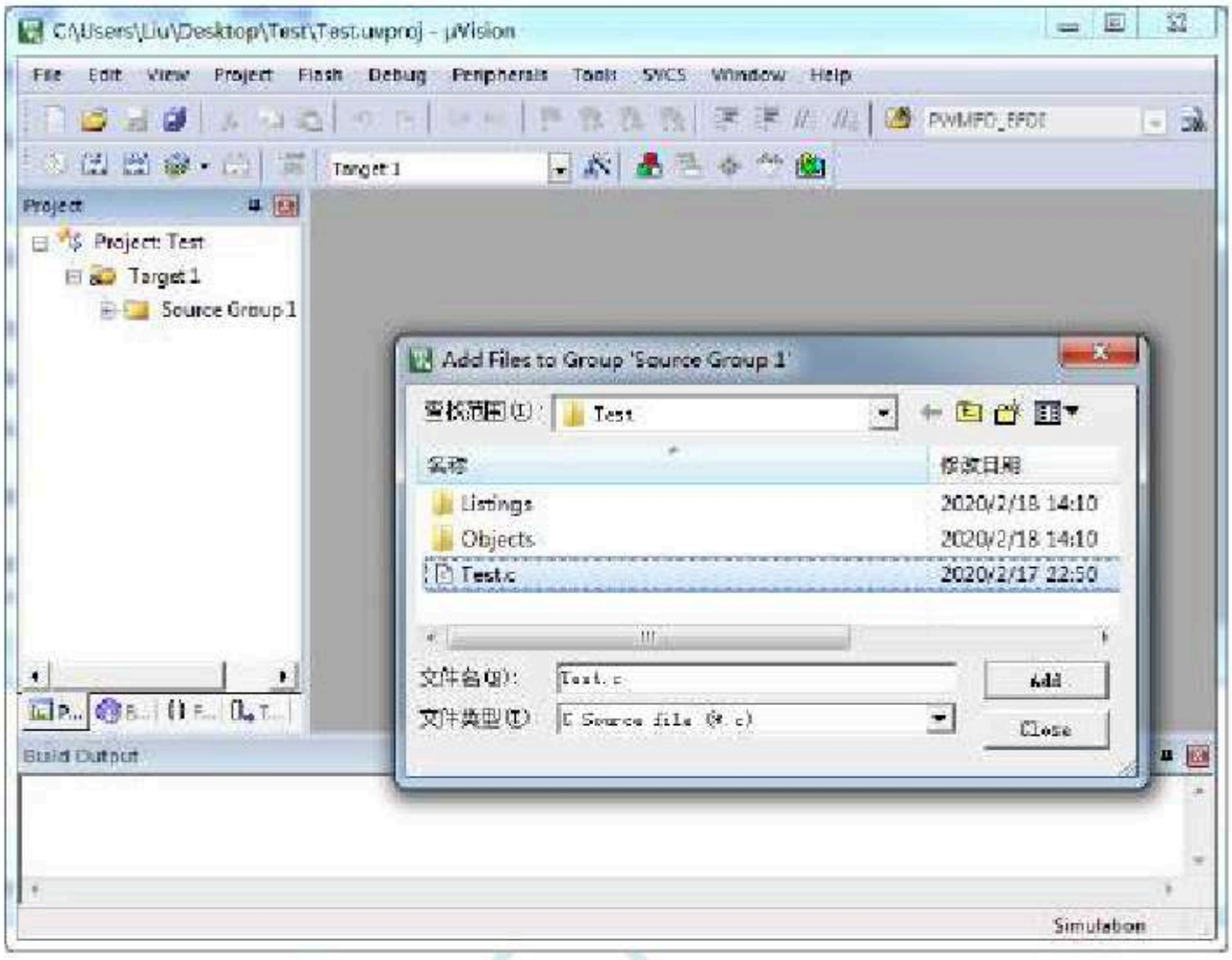
You can complete the establishment of an empty project

2 , In the project tree of the empty project, , and select "in the context menu" Add Files to Source Group 1"

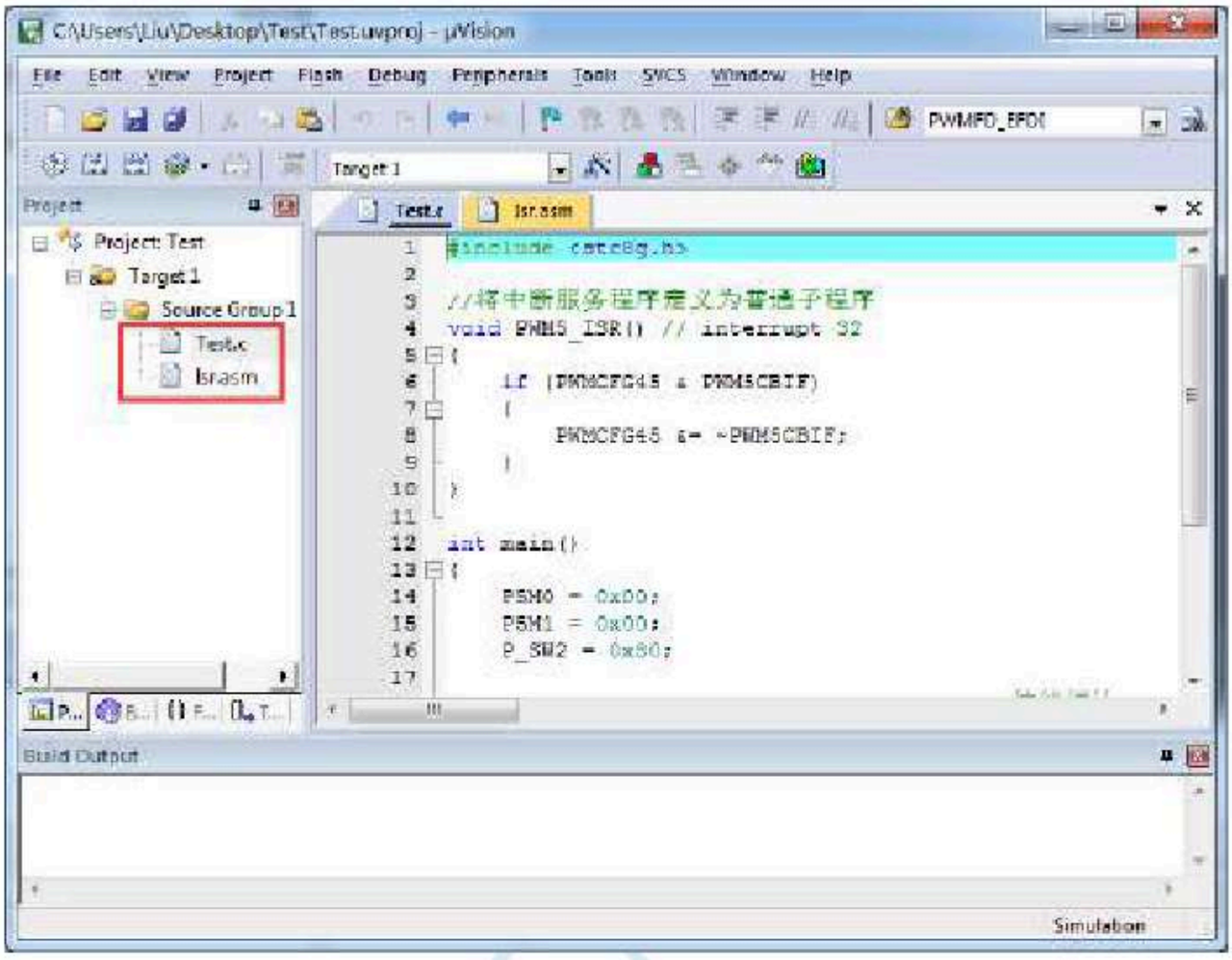
right-click " Group "Source Group 1" . . ."



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
Target 1
Project
Project: Test
Target 1
Source Group 1
Test.c
Isr.asm
Test.c
Isr.asm
1 #include <stc8g.h>
2
3 //将中断服务程序定义为普通子程序
4 void PWM5_ISR() // interrupt 32
5 {
6 if (PWMCFG45 & PWM5CBIF)
7 {
8 PWMCFG45 &= ~PWM5CBIF;
9 }
10 }
11
12 int main()
13 {
14 P5M0 = 0x00;
15 P5M1 = 0x00;
16 P_SW2 = 0x80;
17
Build Output
Simulation

5.8 About the interrupt number is greater than 31 in Keil Handling of compilation errors in

Note: Currently Keil Each version of C51 and C251 The compiler only supports interrupt number (0~31), through our company Multi-party consultation and discussion, the company promised to add our company's version of the interrupt number in the subsequent Keil version, and we can only use the methods in this chapter to resolve it temporarily.

5.8.1 Use the popular interrupt number extension tool on the Internet

Enthusiastic netizens have provided a simple expansion tool that can expand the interrupt number to 254, The tool interface is as follows：

Click the "Open" button to navigate to the Keil After the installation directory, click "OK".

The version is constantly being updated, but there are too many earlier versions, and it is impossible to collect them all. Here is a C51. EXE C251. EXE passed the test due to the version and version： C51. EXE

V6.12.0.1

V8.8.0.1

V9.0.0.1

V9.1.0.1

V9.53.0.0

V9.54.0.0

V9.57.0.0

V9.59.0.0

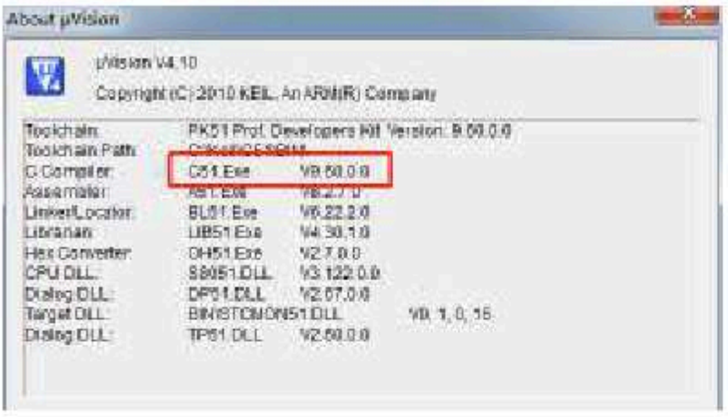
V9.60.0.0

Tested and passed C251. EXE version：

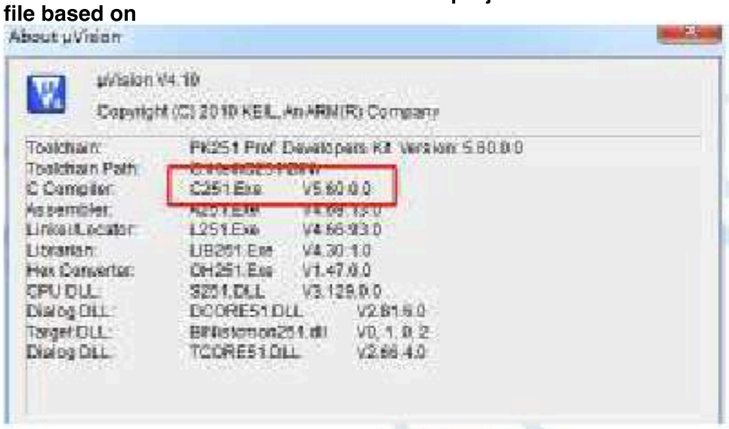
V5.57.0.0

V5.60.0.0

C51. Version method: View EXE in keil Open one based on STC8 Series or STC15 The project of the series of microcontrollers, in Software menu, in items "Open in" About uVision. . ."

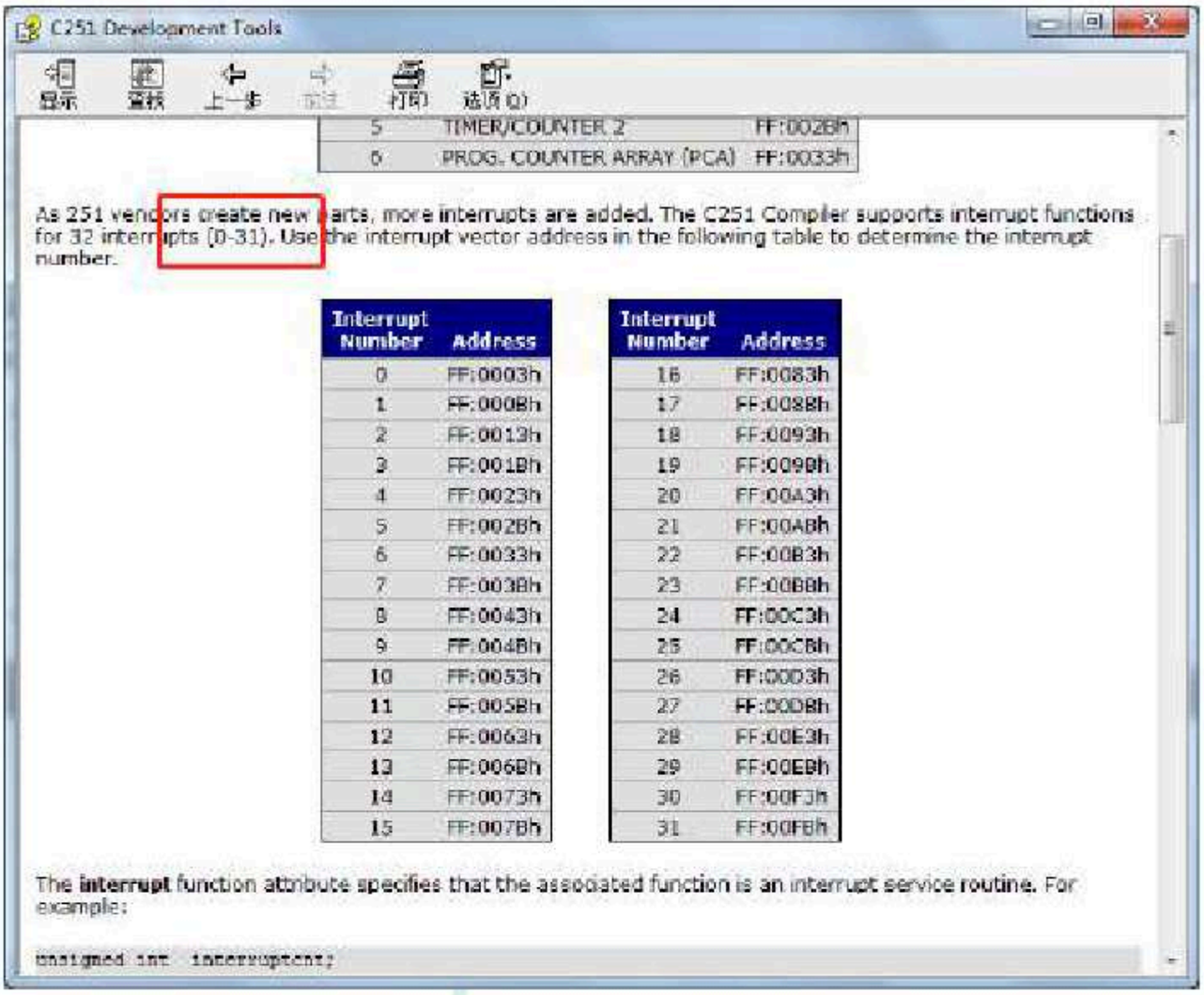


view C251. EXE Version method : Open a file based on keil STC32G The project of the series of microcontrollers, in Software menu, in items "Open in" About uVision. . ."



5.8.2 Use reserved interrupt number for transit

in Keil of C251 In the compilation environment, the interrupt number is only supported , That is, the interrupt vector must be less than



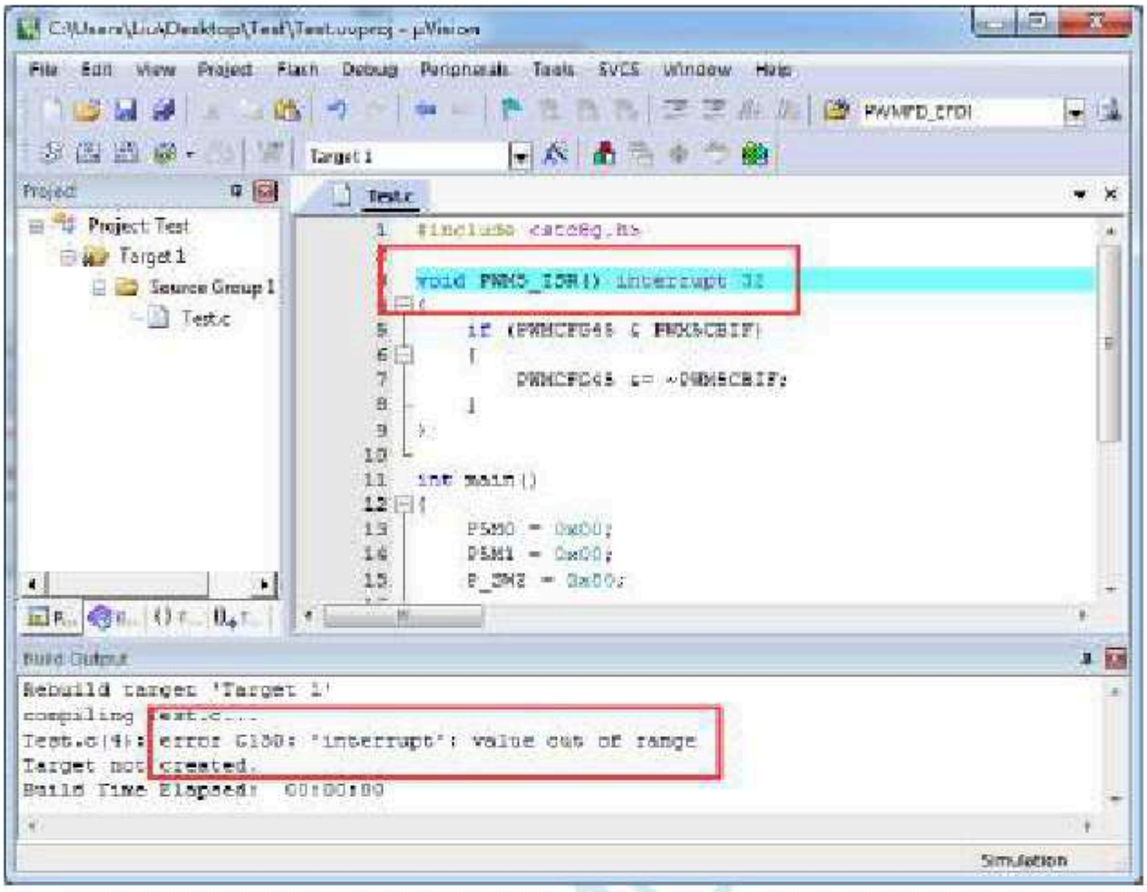
The table below is STC List of current interrupts for all series :

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
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 4
21	00AB H	comparator I2C
24	00C3 H	USB
25	00CB H	PWMA
26	00D3 H	PWMB
27	00DB H	CAN1
28	00E3 H	CAN2
29	00EB H	LIN
30	00F3 H	RTC
36	0123 H	
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
41	014B H	P4 Interrupt Port
42	0153 H	P5 Interrupt Port
43	015B H	P6 Interrupt Port
44	0163 H	P7 Interrupt Port Interrupt
45	016B H	P8 Port Interrupt
46	0173 H	P9 Port Interrupt
47	017BH	M2M DMA Interrupt
48	0183H	ADC DMA interrupt
49	018BH	SPI DMA interrupt
50	0193H	U1T DMA
51	019BH	U1R DMA Interrupt Interrupt Interrupt
52	01A3H	U2T DMA
53	01ABH	U2R DMA Interrupt Interrupt Interrupt
54	01B3H	U3T DMA
55	01BBH	U3R DMA Interrupt Interrupt Interrupt
56	01C3H	U4T DMA
57	01CBH	U4R DMA Interrupt Interrupt Interrupt
58	01D3H	LCM DMA
59	01DBH	interrupt LCM
60	01E3H	I2CT DMA Interrupt
61	01EBH	I2CR DMA interrupt
62	01F3H	interrupt I2S
63	01FBH	I2ST DMA Interrupt
64	0203H	I2SR DMA interrupt

Not difficult to find ,The interrupt begins, and all subsequent interrupt service programs will be compilation errors in both, as shown in the figure bel

RTC



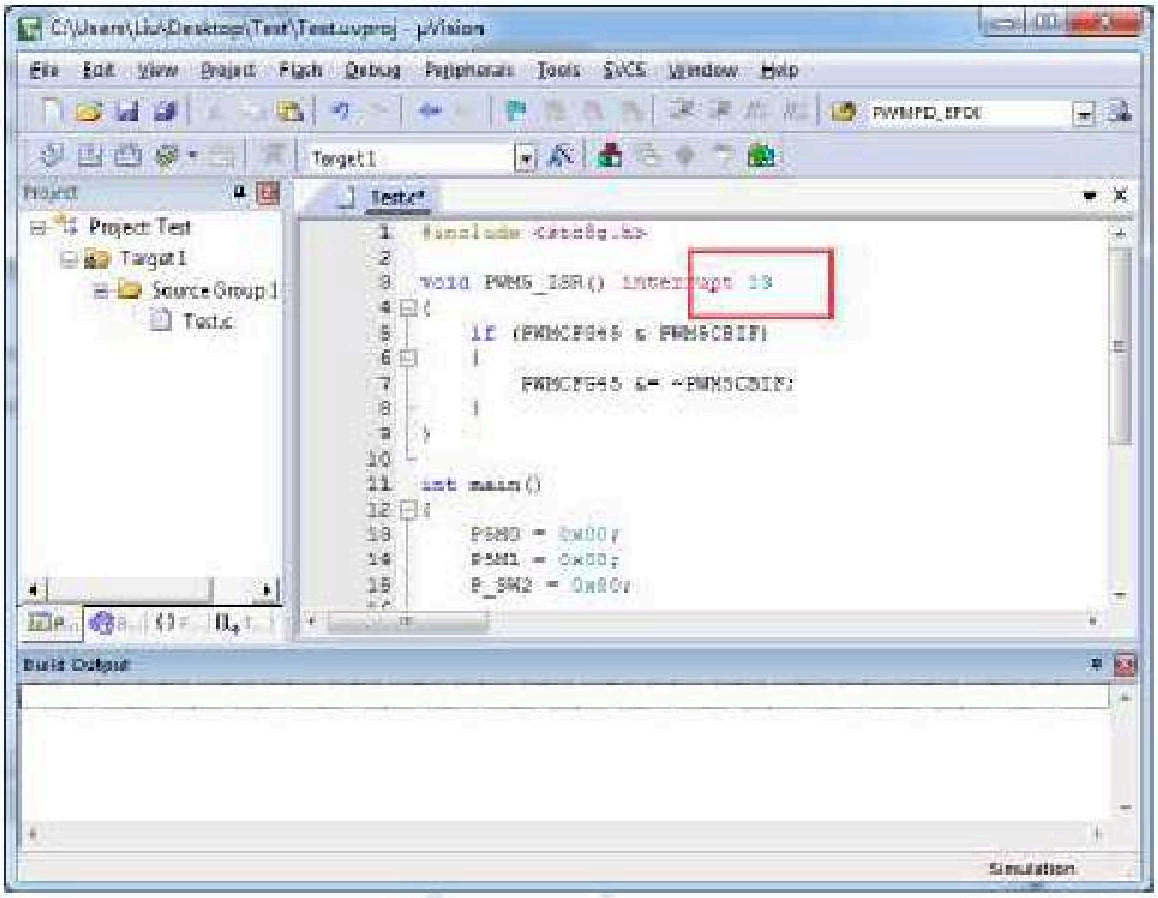
There are three ways to deal with this error (all require the help of assembly code, the preferred method is recommended)[1]

13.1 Method: Borrow interrupt vector

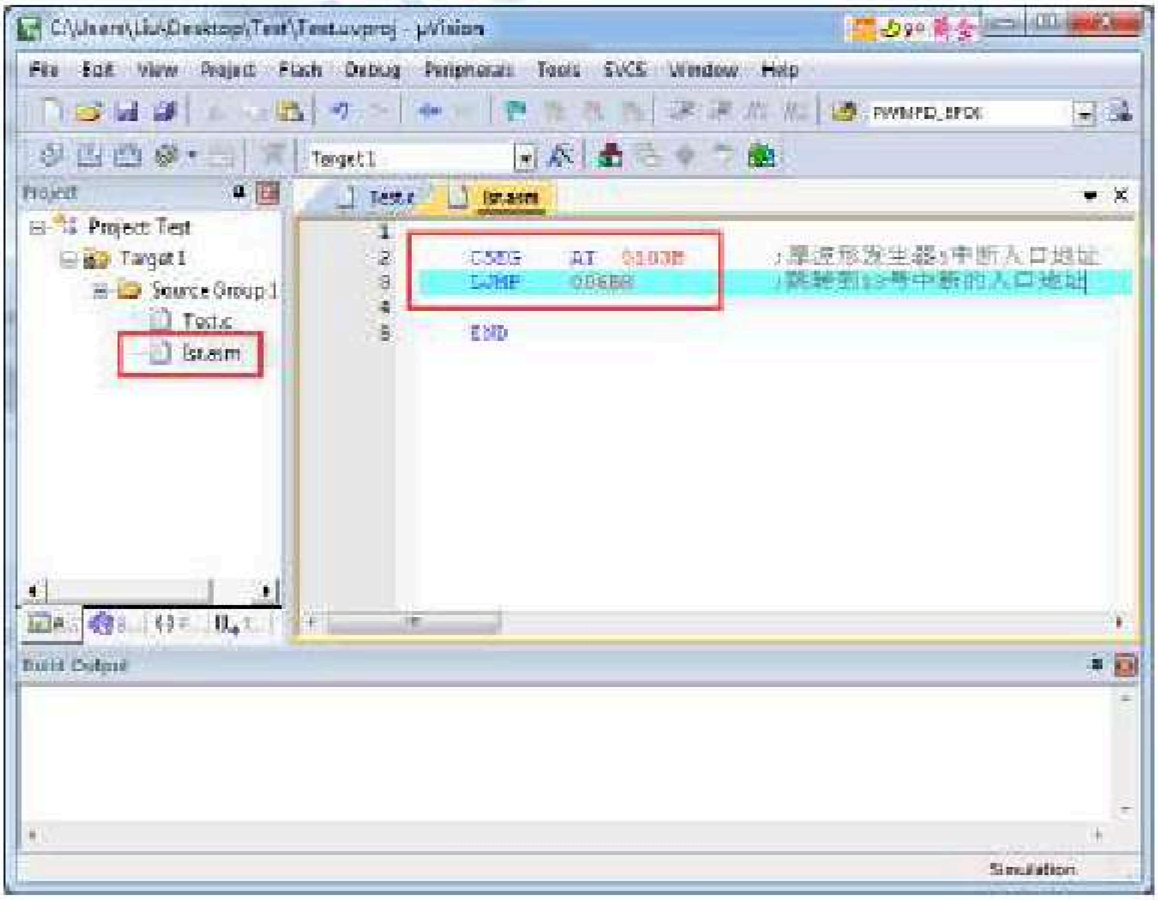
0~31 The signal is interrupted, the first 13 The number is a reserved interrupt number, we can borrow this interrupt number

The steps are as follows:

1, Change the interrupt number we reported the error to "13", as shown below:

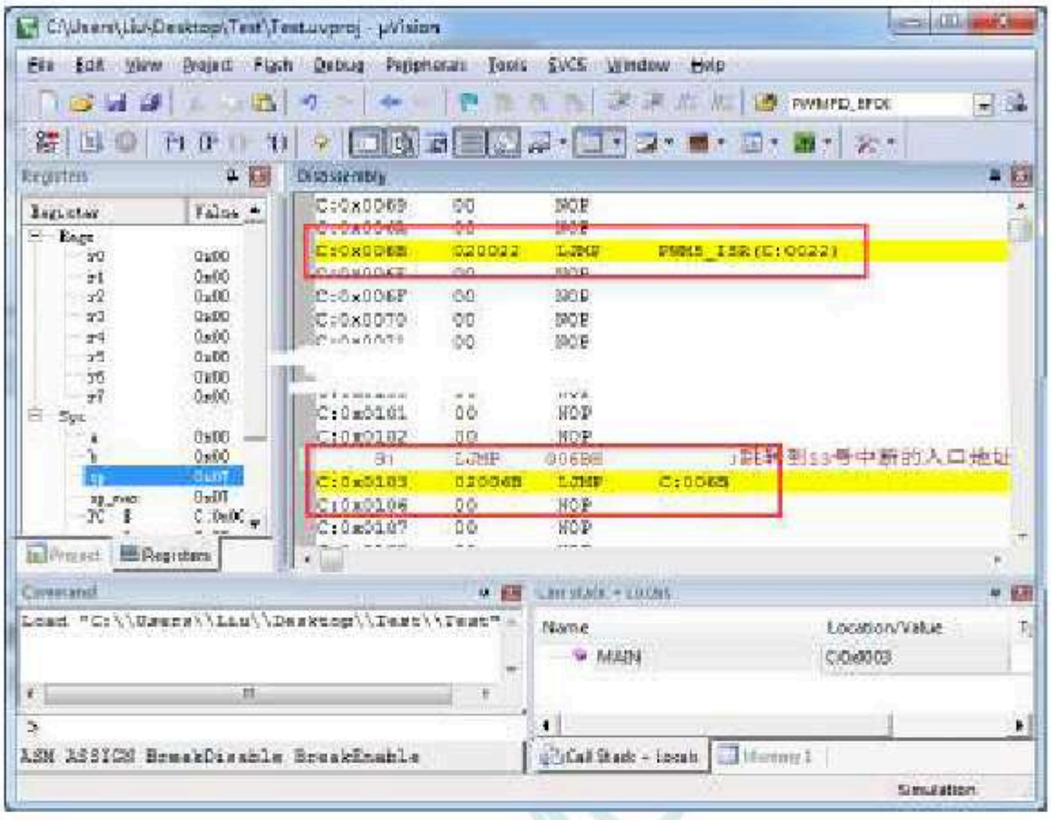


2, Create a new assembly language file, such as "isr.asm", Add to the project and at the address "0103H" Add one in the place "LJMP 006BH", as shown below:

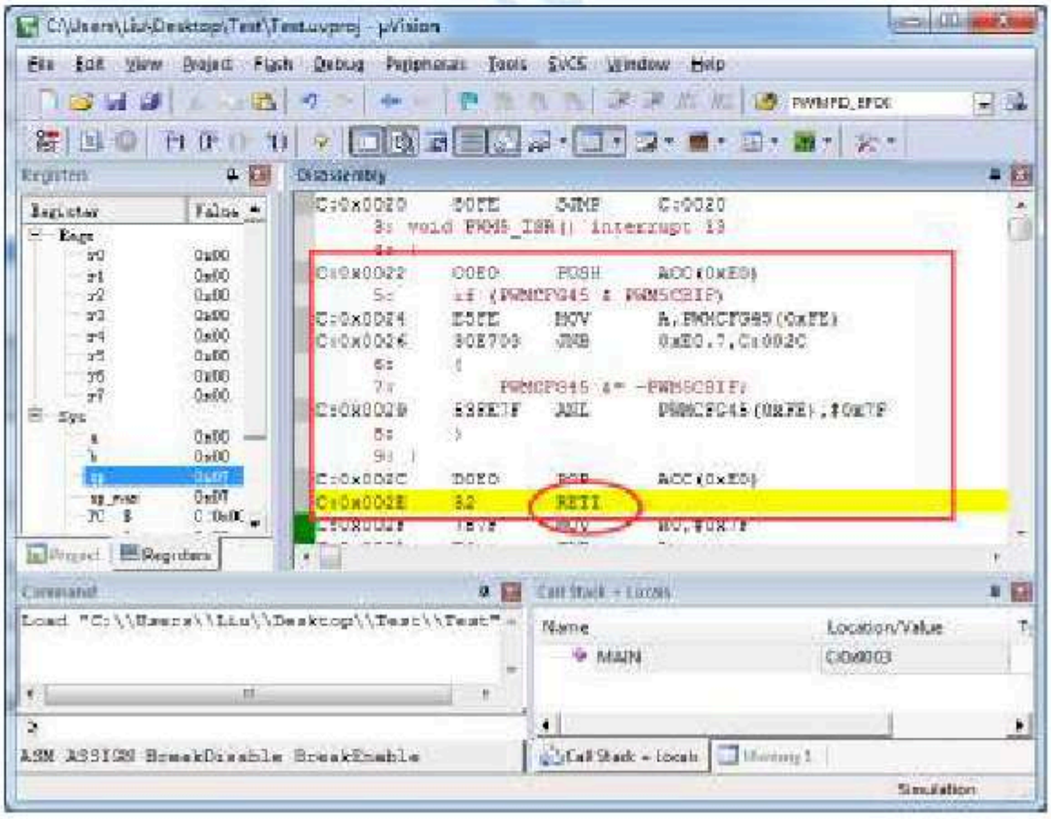


3, Compilation can be passed.

Pass by at After the compiler is compiled, there is There is one here."LJMP PWM5_ISR", in 0103H There is one here this time"LJMP C51 006BH", as shown below :



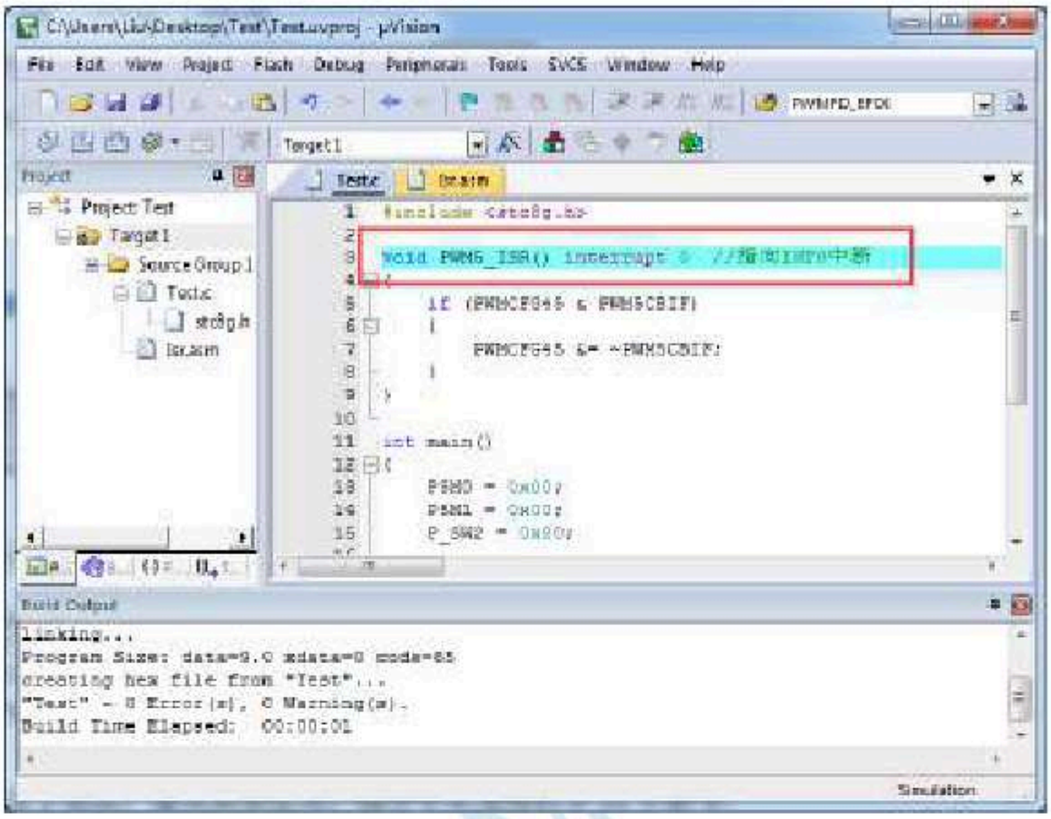
When the line PWM5 When interrupted, the hardware Address execution"LJMP 006BH", and then in 006BH Execute again occurs"LJMP You can jump to PWM5_ISR will automatically jump to the real interrupt service program, as shown in the figure below. :

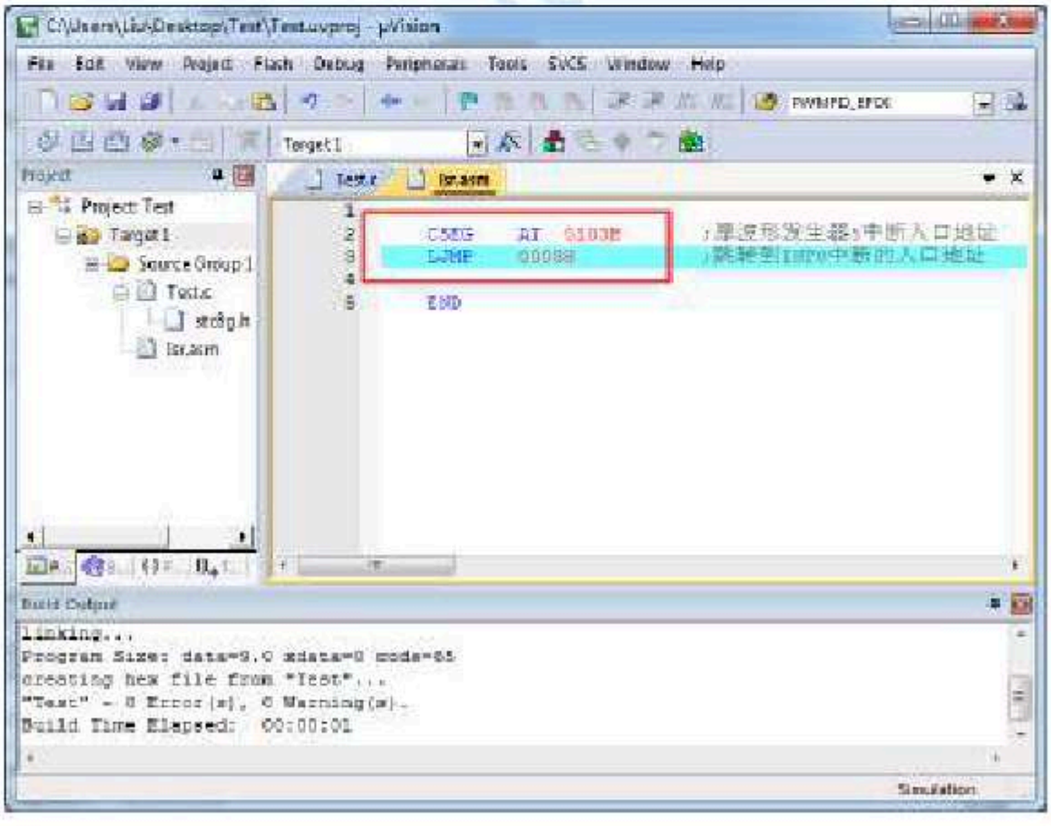


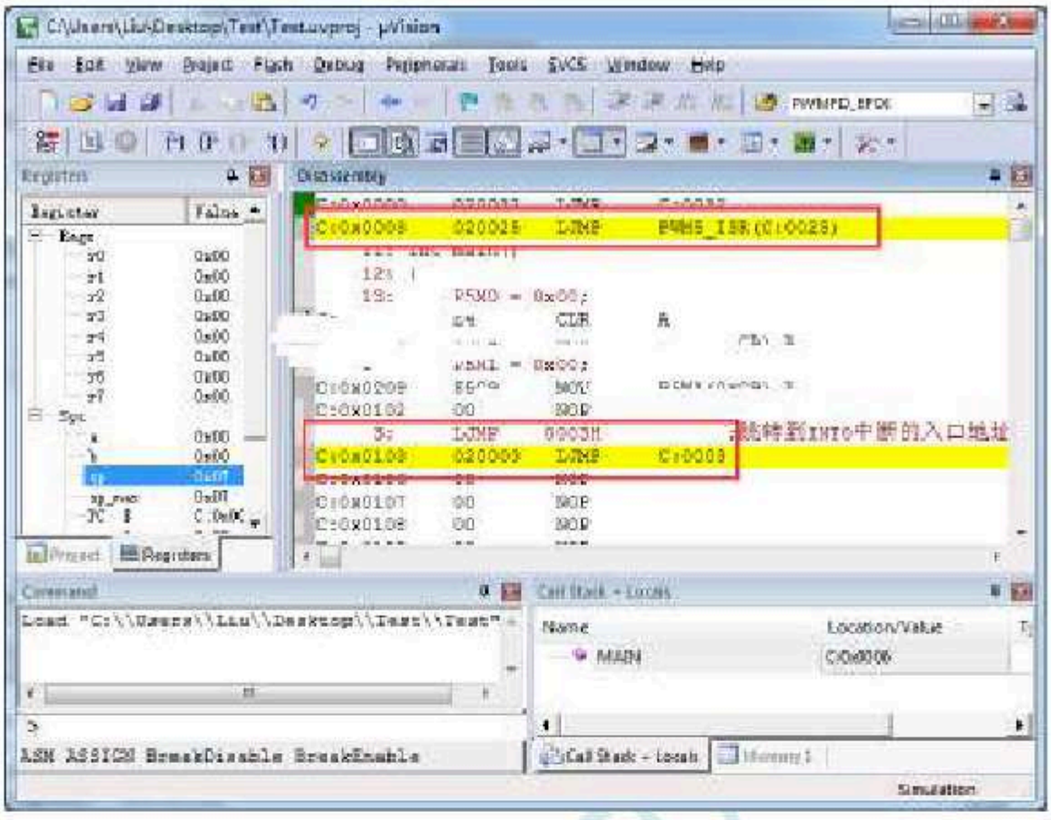
After the execution of the interrupt service RETI The command returns. The entire interrupt response process is just one more execution program is completed, it will be passed again . LJMP Statement and

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 Modified tion of :





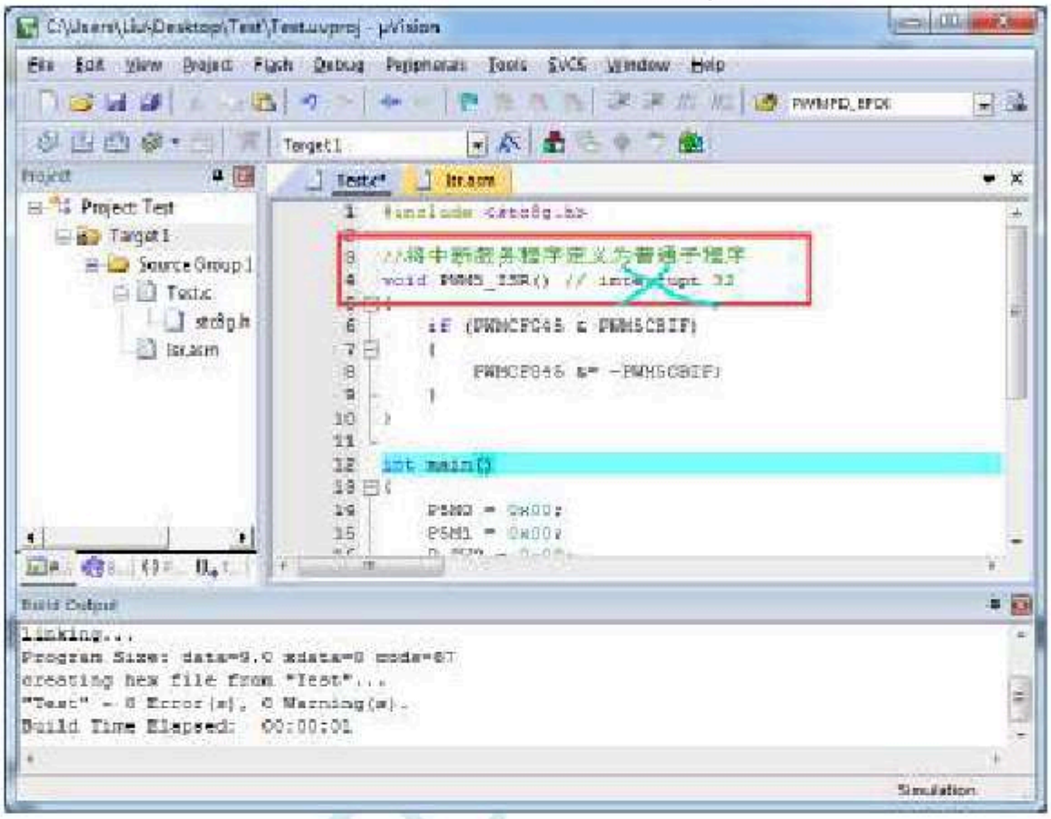


Execution effect and method: The same, this method is suitable for multiple interrupt numbers that need to be remapped. The instruction

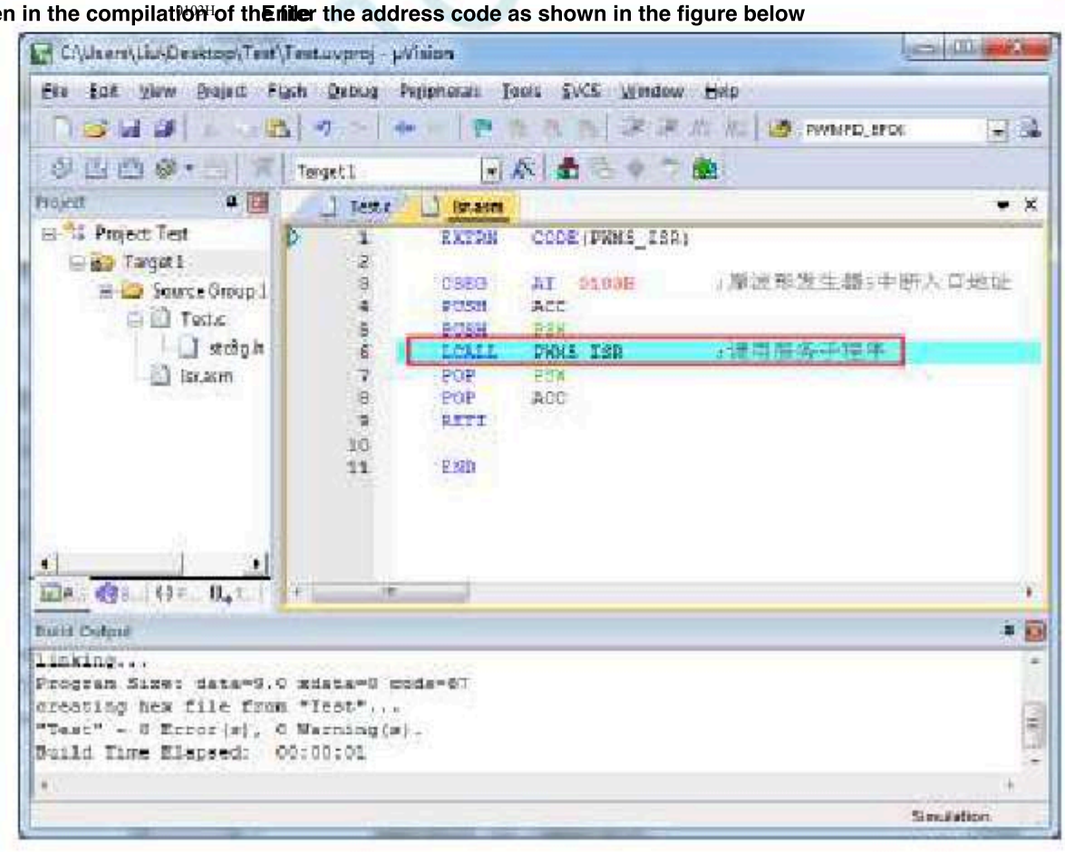
Method 3: Define the interrupt service program as a subroutine, and then in the interrupt entry address in the assembly code use LCALL Instruction execution service program

The steps are as follows：

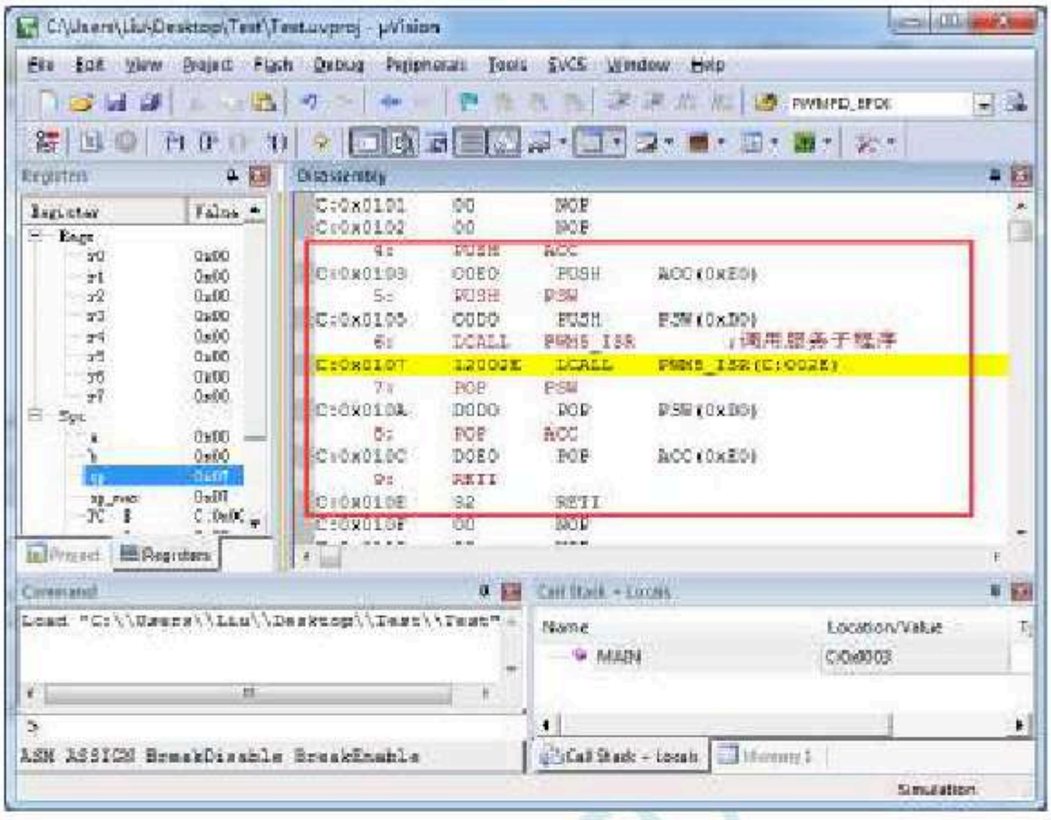
1 **, First remove the interrupt service program" "Attributes, defined as ordinary subroutines**



2 **, And then in the compilation of the file Enter the address code as shown in the figure below**



3 **The place of the address is to interrupt the service program, and after the compilation is passed, i**



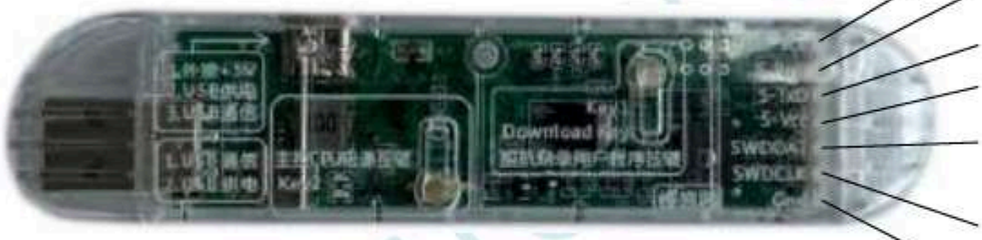
This method does not require remapping the interrupt entry, but there is a problem with this method. Which registers need to be pressed into the stack in the assembly, the disassembled code of the program is determined. Generally includes DPH PSW In addition to the stack must be pressed, which other registers are used in the user's subroutine, and which registers must be pressed

5.9 STC-USB Link1D Precautions for tool use

5.9.1 Tool interface description

STC-USB Link1D The tool is a serial STC-USB Link1D The upgraded version and functions are in STC-USB Two more have been added on the basis of STC-CDC port, which can be used as USB Link1 Go to the serial port to use. tools a universal STC-USB Link1D Please refer to the Appendix chapter for precautions for use

Front view of the tool



User-Vcc, External power supply to the tool

2 STC-CDC Serial port send pin , S-RxD, The first group is connected to the receiving pin of the serial port of the user's MCU

2 STC-CDC Serial port receiving pin , S-TxD, The first group is connected to the sending pin of the serial port of the user's MICROCONTROLLER

S-Vcc, This tool supplies power to the external system

Hardware simulation data cable, Link1S-P3.0

Downloaded serial port sending pin, Link1 Proceed

1 Group 1 STC-CDC Serial port send pin

Hardware simulation clock line, Link1S-P3.1

Downloaded serial port receiving pin, Link1 Proceed

1 Group 1 STC-CDC Serial port receiving pin

Gnd, Ground wire

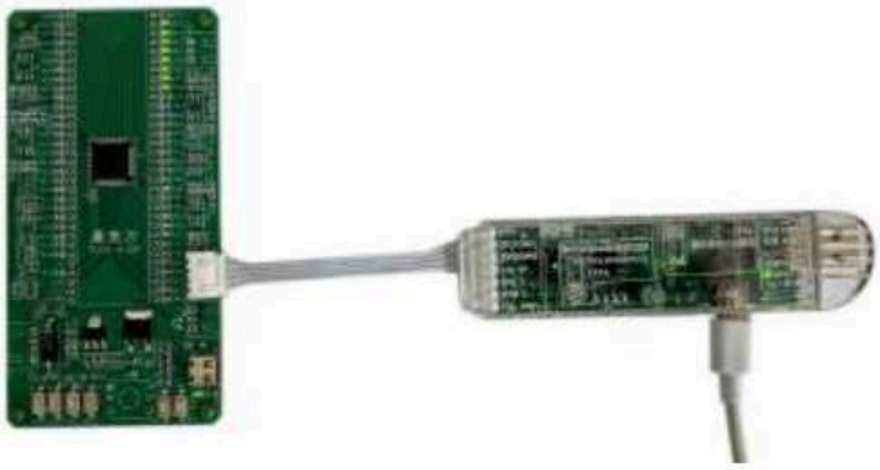
Reverse side view of the tool

Pin number	Interface name	Interface function
1	User-Vcc	Only the user system supplies
2	S-RxD	power to this tool 2 Group 1 The sending pin of the serial port, the receiving pin of the serial port connected to
3	S-TxD	Group 1 2 STC-CDC the user's MCU, the receiving pin of the serial port, and the sending pin of the serial port
4	S-Vcc	Only power the user's system
5	S-P3.0	from this tool Use in progress ISP The serial port sending pin when downloading, connected to the target MCU P3.0 Use in progress Link1D SWD The data pin during hardware simulation, connected to the target MCU SWDIO 1 Group 1 STC-CDC The sending pin of the serial port is connected to the receiving pin of the serial port of the
6	S-P3.1	Use in progress Link1D ISP The serial port receiving pin when downloading, connected to the target MCU P3.1 Use in progress Link1D SWD The clock pin during hardware simulation, connected to the target MCU SWDCLK Group 1 1 STC-CDC The receiving pin of the serial port is connected to the sending pin of the serial port of the
7	Gnd	Ground wire

5.9.2 STC-USB Link1D Practical application

1, use STC-USB Link1D Tool pair Series of microcontrollers for hardware simulation

In the manner shown in the figure below, the SWDGND S-P3.0、S-P3.1、SWDGND Connect to the target MCU separately, and then refe
tool's P3.0（SWDDAT）、P3.1（SWDCLK）, and the settings GND to the hardware simulation steps in the previous chapter.
can be used for hardware simulation SWD

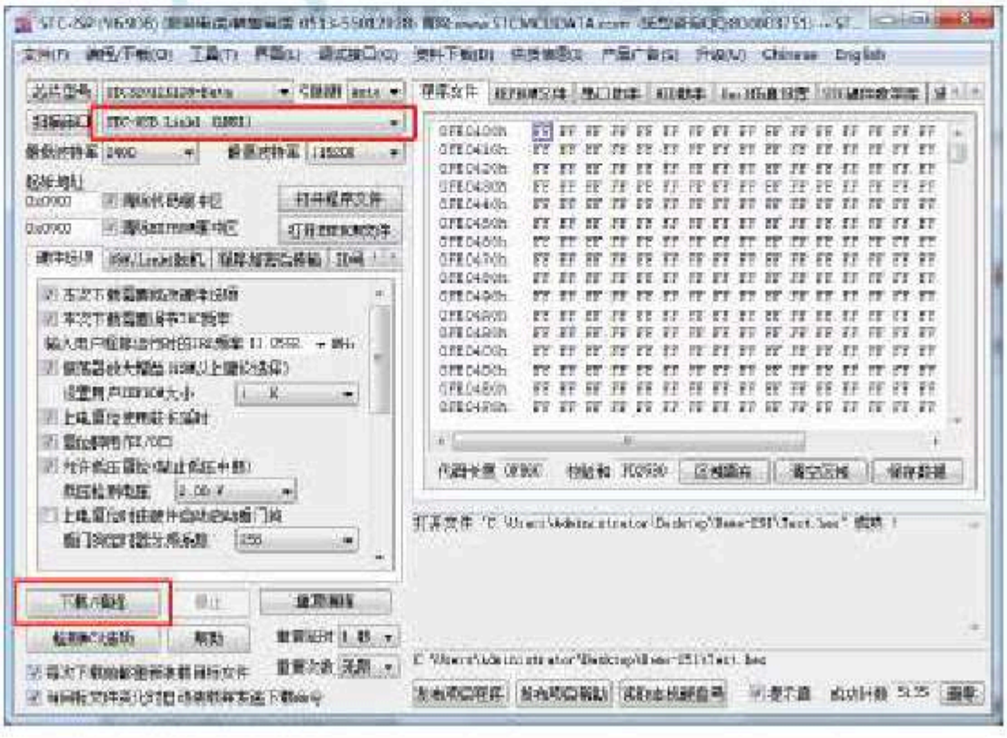


STC-USB Link1D 2’ Tool pair Series for serial port simulation and STC15

The use of tools SWDGND S-P3.0、S-P3.1、GND STC8 Respectively with the target MCU P3.1（TxD）,
GND Connect, and then Select from simulation settings （RxD）, the corresponding STC15/STC8
serial port number, and then refer to
The simulation steps and settings in the direct serial port simulation chapter in the series of data manuals can be used for ser

STC-USB Link1D 3’ Tool pair A full range of microcontrollers can be downloaded out

The use of tools SWDGND S-P3.0、S-P3.1、STCGND Select the serial port number in the download P3.1（TxD）、
GND Connected, in STC-ISP software of the target MCU separately", STC-USB Link1 (LNK1) ", open the program
File and set related hardware options, and then click "Download/The "Program" button can be downloaded out

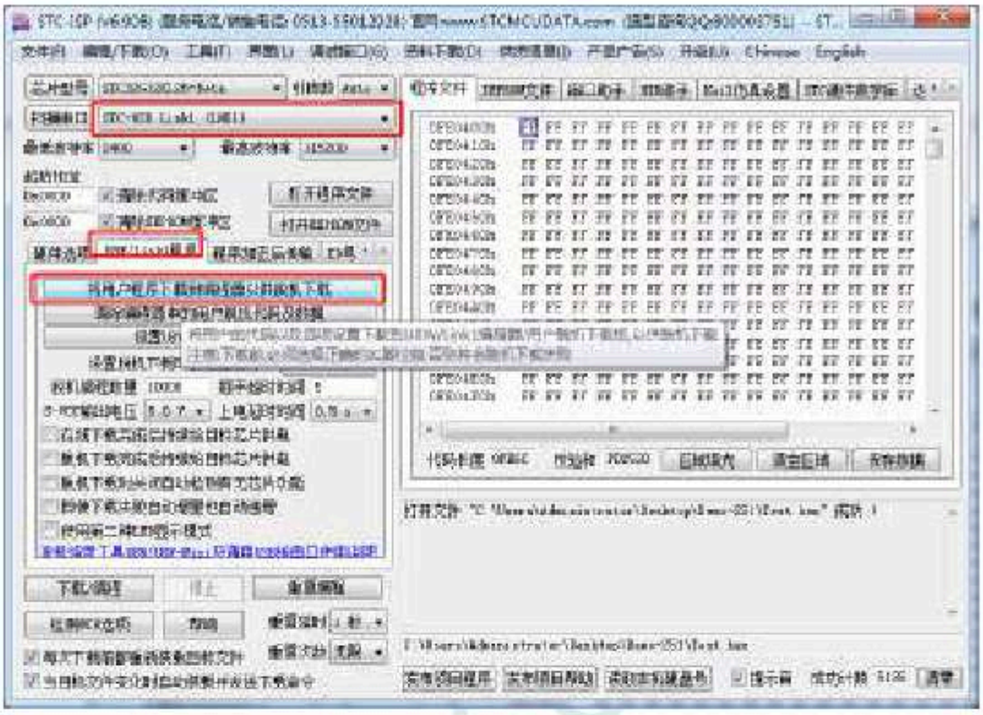


4, use Tool pair STC-USB Link1D Download the full range of microcontrollers offline for pr

Select "in" the serial port number in the downloaded software , Open the program file and set the phase

Turn off the hardware options, and then "Download the user program to the programmer for offline download" on the

Download" button to download the user code and related settings to In the memory on the tool.



The tool's S-Vcc、S-P3.0、S-P3.1、GND Respectively with the target MCU), ("Press the button to

GND Connect, and then press "on the tool" download the target chip offline (that is, no need PC

Terminal control, independent Download)

5、 STC-USB Link1D The tool provides two Designed for use with tools

STC-USB Link1D as a general tool STC-CDC Serial port, can be Designed for use with tools, due to USB

The first serial port CDC1 With hardware simulation download to share and Port, and the second serial port CDC2

It is an independent serial and S-P3.0 As the simulation and Download and use, when you need to use a general purpos

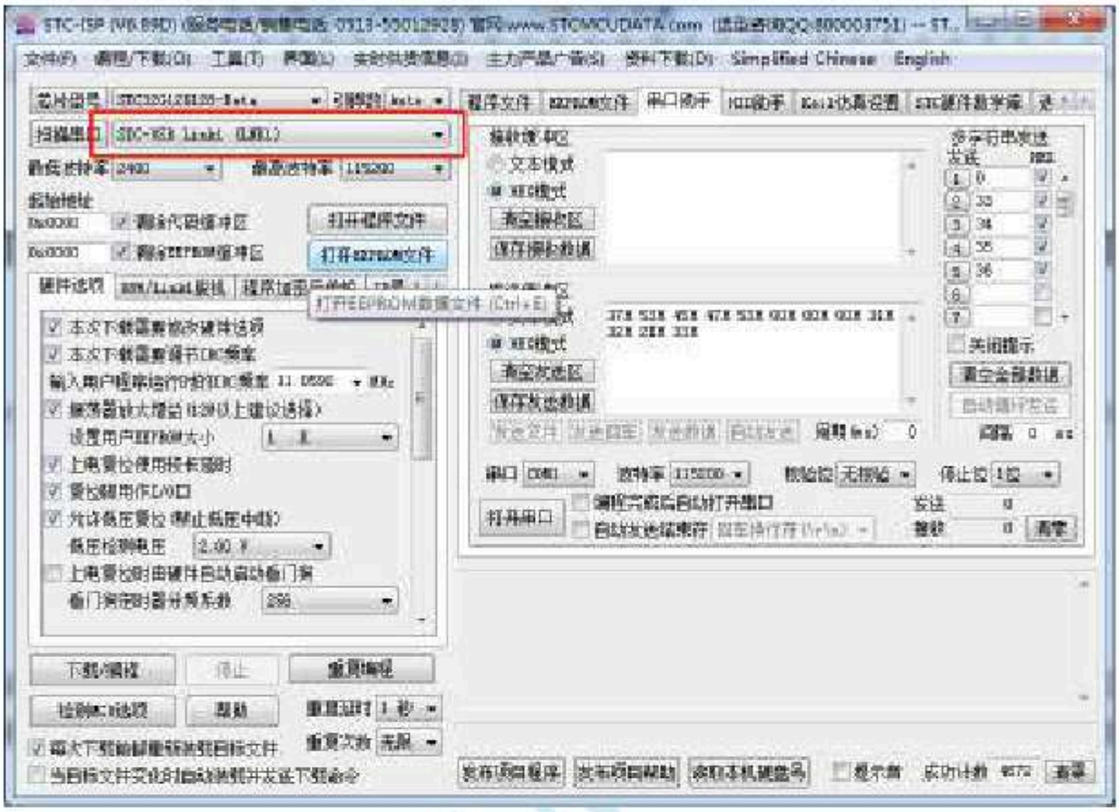
port, so it is recommended to and S-RxD the corresponding CDC2 (Note: In the absence of a conflict of use , CDC1

use the serial port tool and Can be used independently as a general Specially designed for use with tools)

Correct tool identification 5.9.3

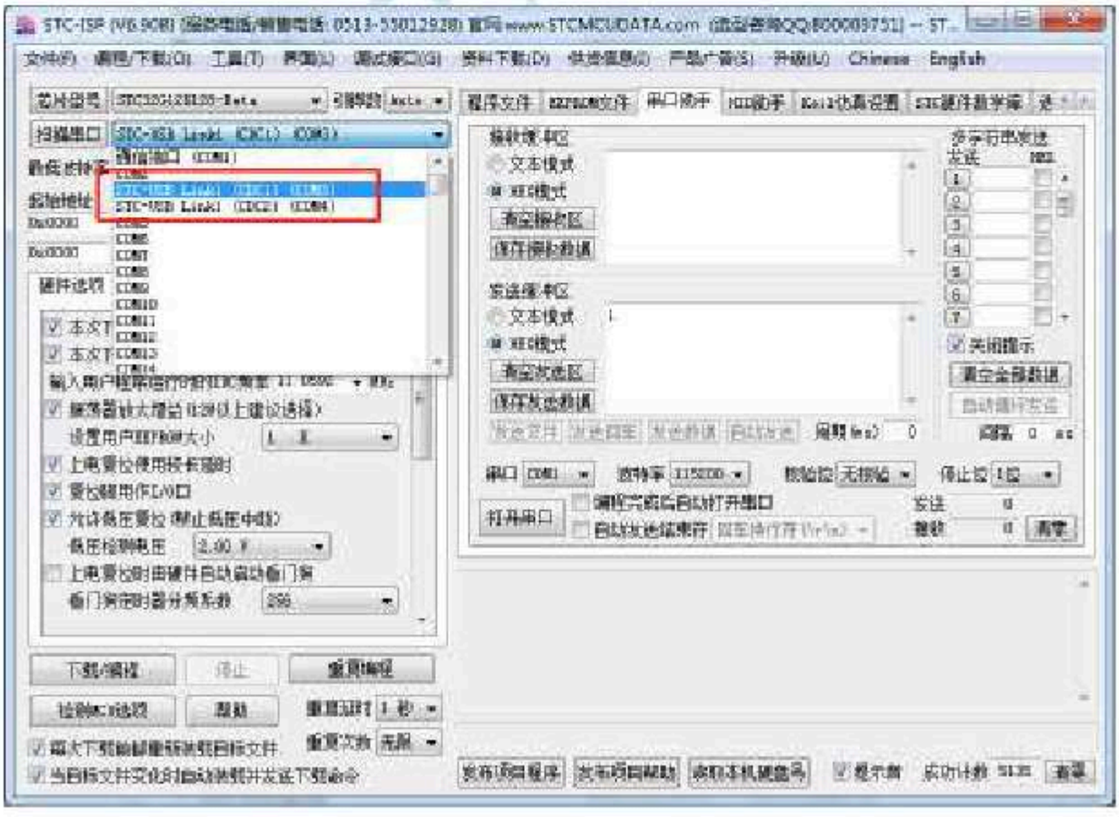
STC-USB Link1D When the tool is out of the factory, the main control chip has been burned The control program. Under normal circumstanc

After the tool is connected to the computer, it will be recognized immediately in the downloaded software " as shown in the figure below



After it is correctly identified, it can be used Go online Download or offline ISP ISP download.

After the driver is successfully installed, two more will be automatically recognized Serial port, as shown in the figure below :



Can be regarded as universal Use the serial port tool.

5.9.4 production STC-USB Link1D Main control chip

STC-USB Link1D When the tool is out of the factory, our company's operators have completed the production of the main control ch

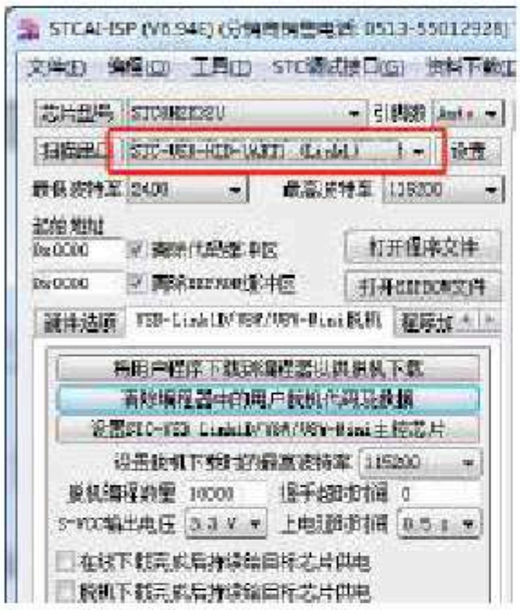
You don't need to make it again manually after that The main control chip of the tool. Only when the customer replaces the main control c

The following steps are required for a brand new chip.

STC-USB Link1D1' Mouth, if the tool is inserted into the computer's The list of the downloaded software is not shown STCAI

The steps of (Link1) "The equipment (as shown below) ,It means that the main control chip of the tool is empty and needs to continu

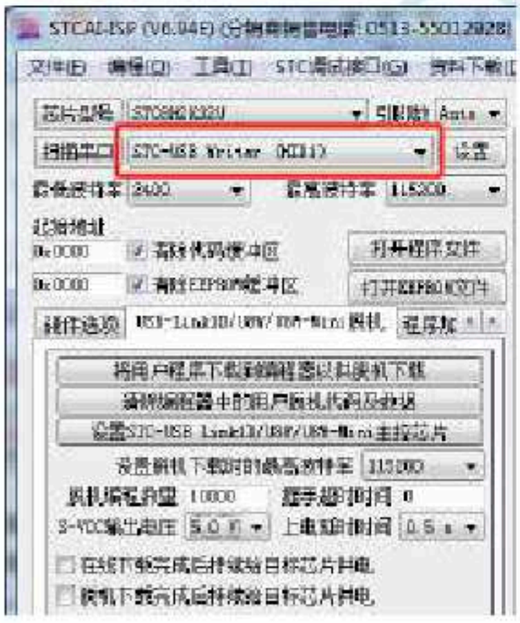
"will " STC-USB-HID-UART1



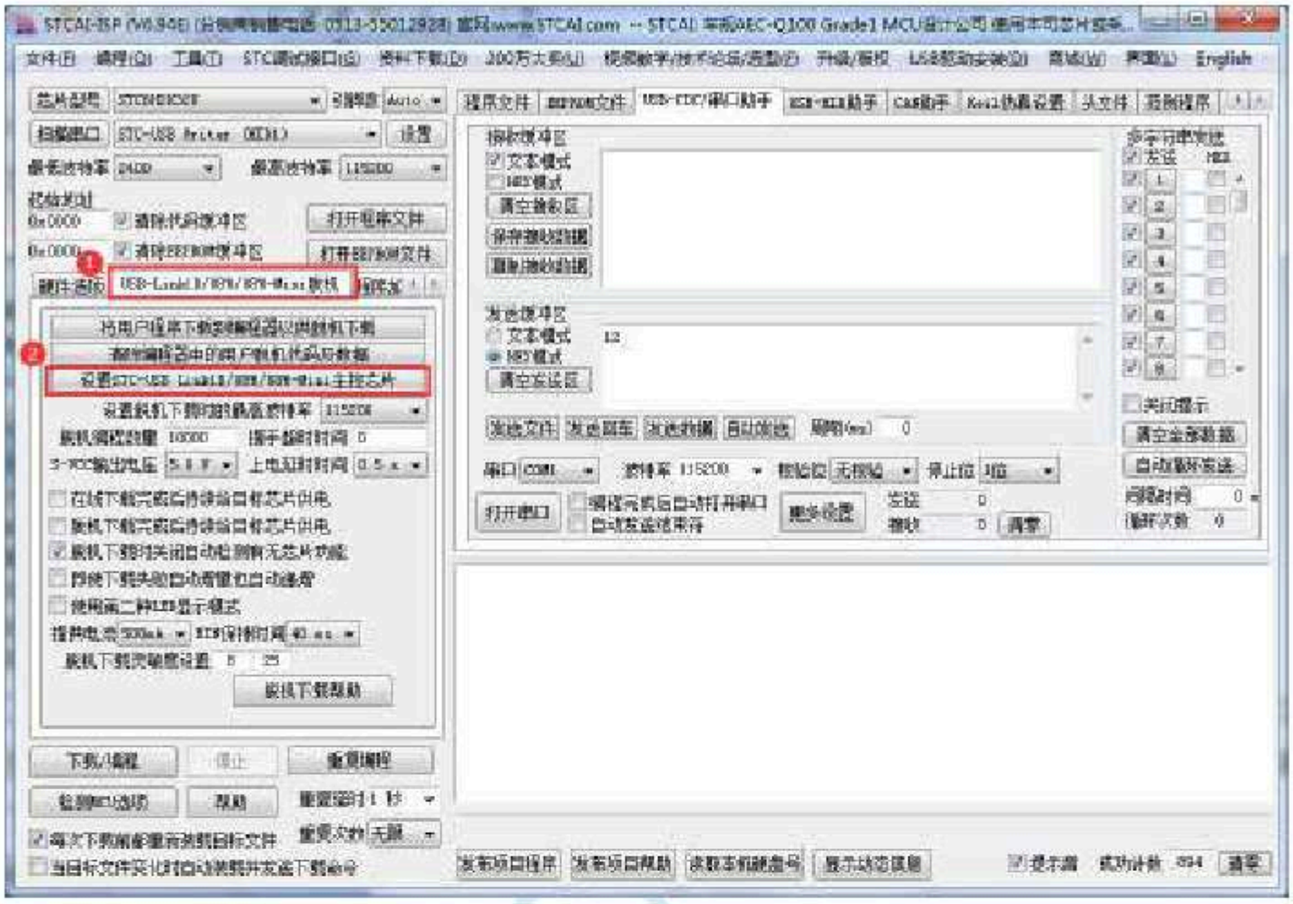
2, Press and hold the tool's" Don't release the button, tap again" "After pressing the button, release the port Button (Key2 need to save at this time Key2

The button has been pressed all the time), wait for the STC-ISP list of the downloaded software and it shows " (HID1)"of

device (as shown below), and then release it key Key1



, click STCAI-ISP 3 **Downloading the software** Link1D/U8W/U8W-Mini **Offline "page" settings** STC-USB Link1D/U8W/U8W-Mini **Main control chip " button**

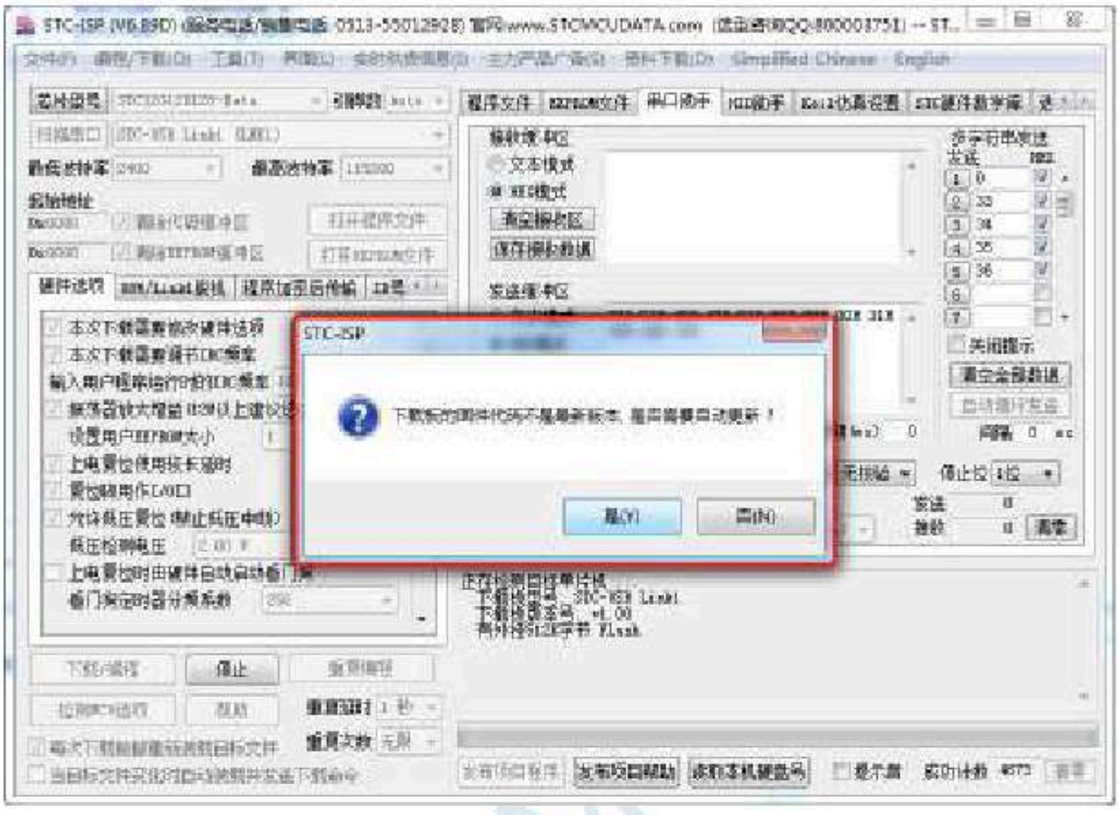


4, **Pop-up** Link1D **The tool settings window can be set according to actual needs, and the default options can be maintained if there are n**

5、Next, the software will automatically start production The main control chip of the tool. After the production is completed, it is recommended to unplug the USB line. Re-plug and unplug once (especially for a brand new chip, you must re-plug and unplug it once when making the main control chip for the first time). When the main control chip is made and reinserted into the computer again, the port list of the downloaded software is displayed STCAI-ISP ""Equipment (as shown below) STC-USB-HID-UART1 (Link1) , It means that the main control chip of the tool is successfully made.

Automatic tool firmware upgrade 5.9.5

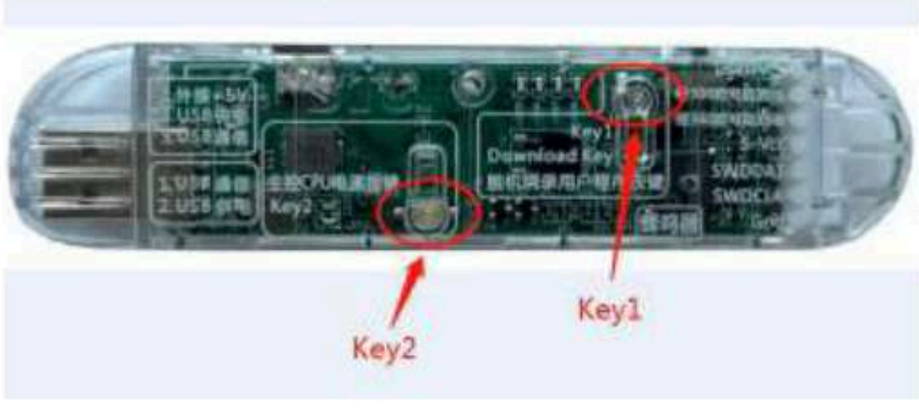
When using tools to carry out ISP When downloading, the following screen pops up in the software, indicating that the firmware of the tool needs



Click the "yes" button and the tool will automatically start upgrading.

5.9.6 Enter the method of updating the firmware

Use first [illegible] computer with a cable, and then first press and release, the tool Key2, etc. to be STC-ISP The downloaded software recognizes "Then release then click Key1。Key1

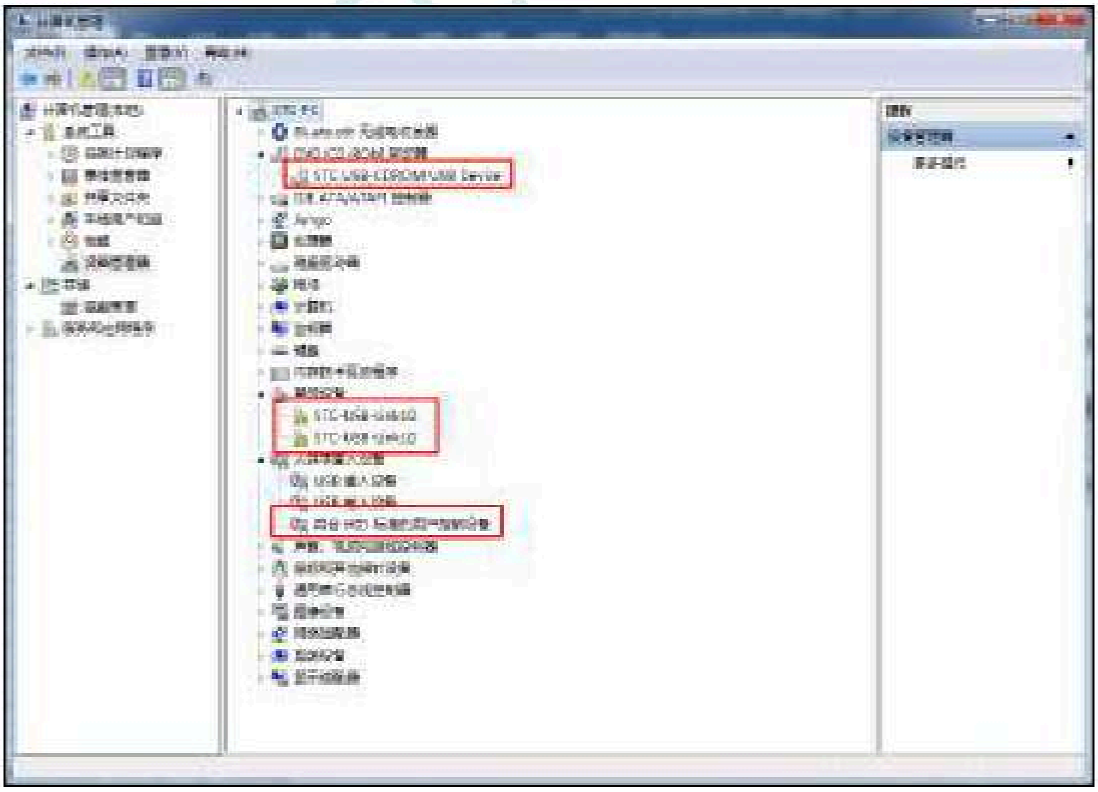


STC-USB Link1D 5.9.7 Driver installation steps

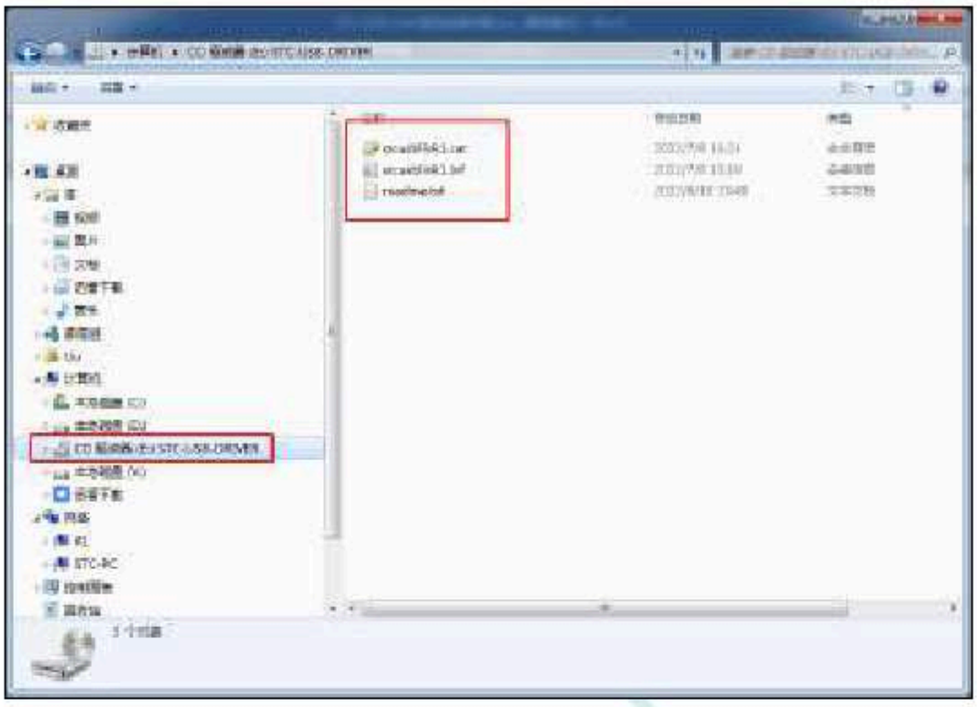
1, will STC-USB Link1D The tool is inserted into the computer USB The computer will display the following screen

2 . After the automatic installation of the driver is completed, in the device manager of the computer, it STC-USB Link1D In the device HID Pick up shows that the port and the optical drive interface have been automatically recognized, but the two CDC The interface has serial port with a yellow exclamation mark, which means that the driver of t

CDC To install manually. As shown in the figure below

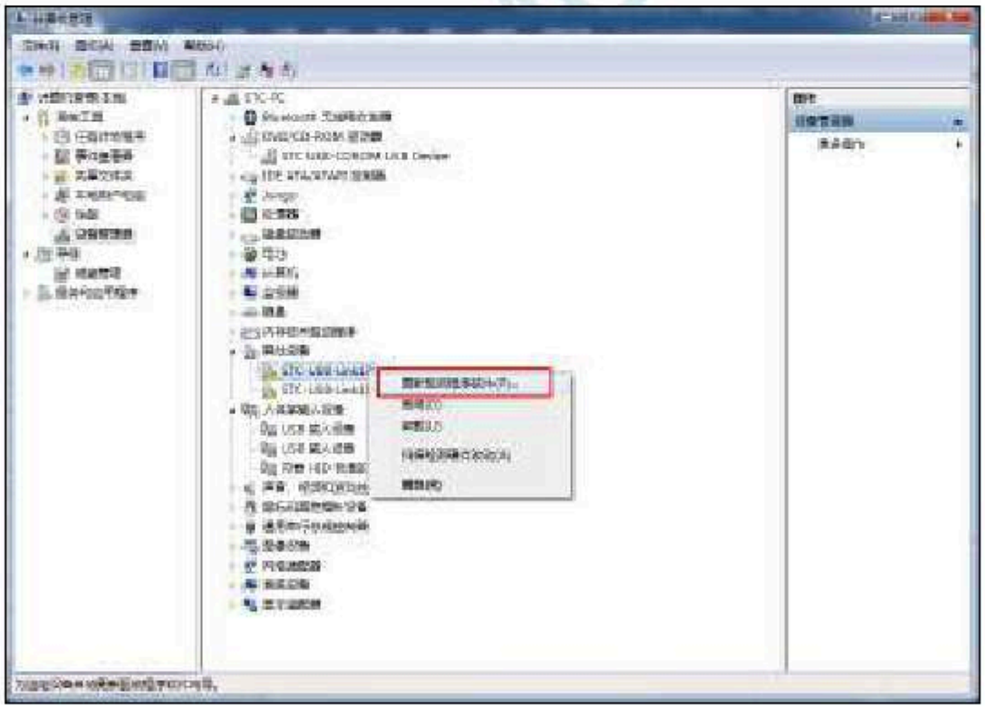


in Windows In the explorer, open the automatically recognized USB Optical drive, there is a virtual serial port driver inside.



3 , The steps to manually install the driver of the virtual serial port are as follows: First, find the first "With a yellow exclamation mark" in the d

Link1D ", and click the right mouse button to select "Update Driver Software" in the context menu (P). . ."

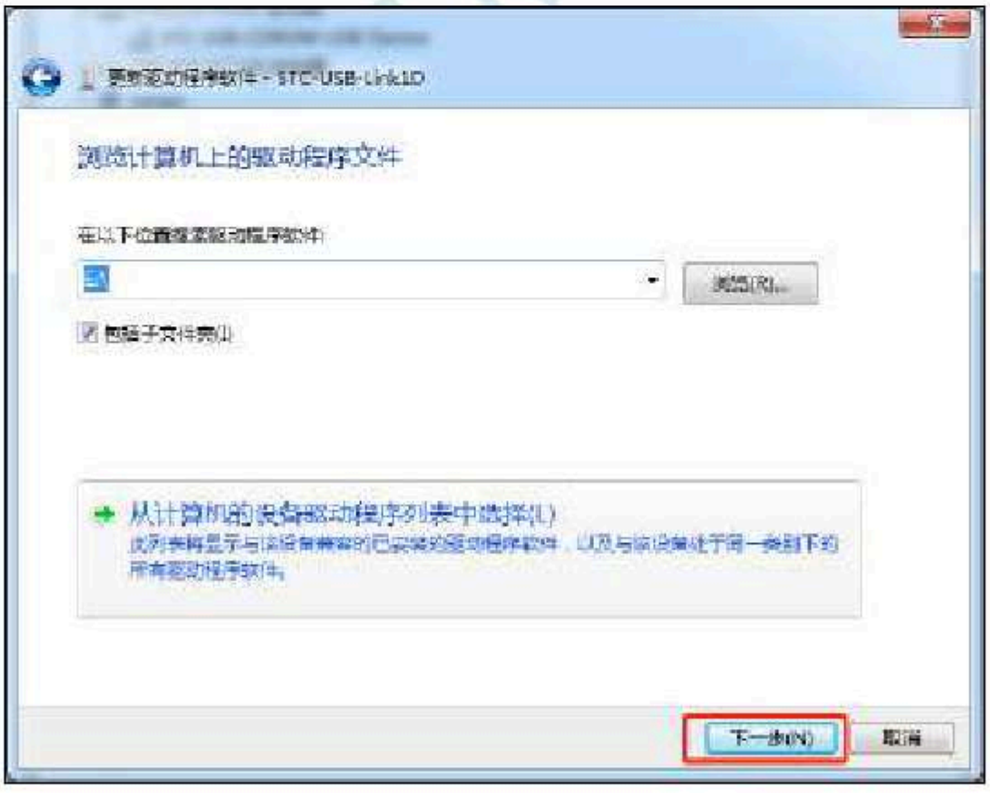


4 , In the pop-up "Update Driver software" window, click "Browse computer to find driver software"

5 , Click the "Browse" button in the following screen

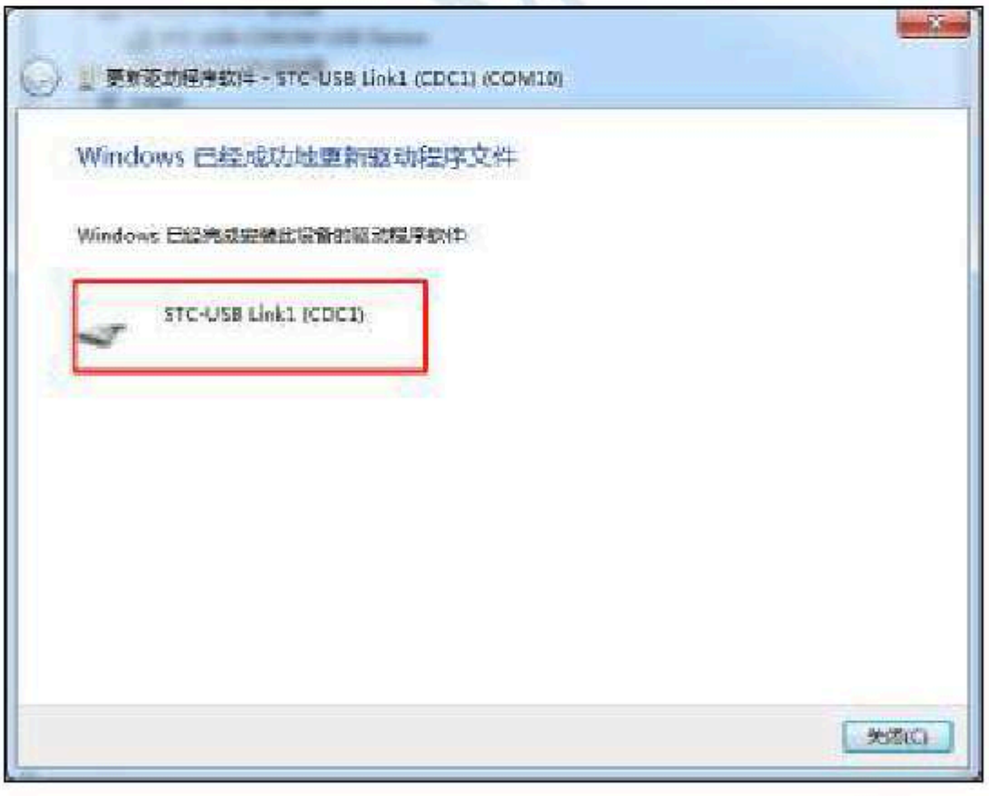
6 , In the browse folder window, select " STC-USB-DRIVER "Optical drive, and confirm

7 , As shown below, click the "Next" button to start installing the driver

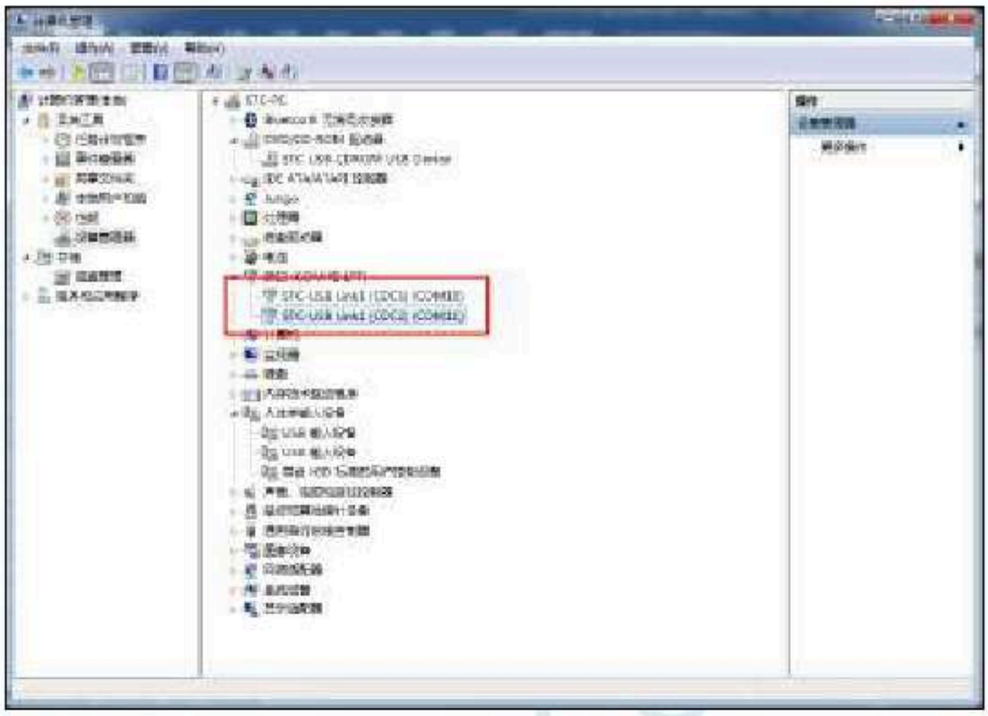


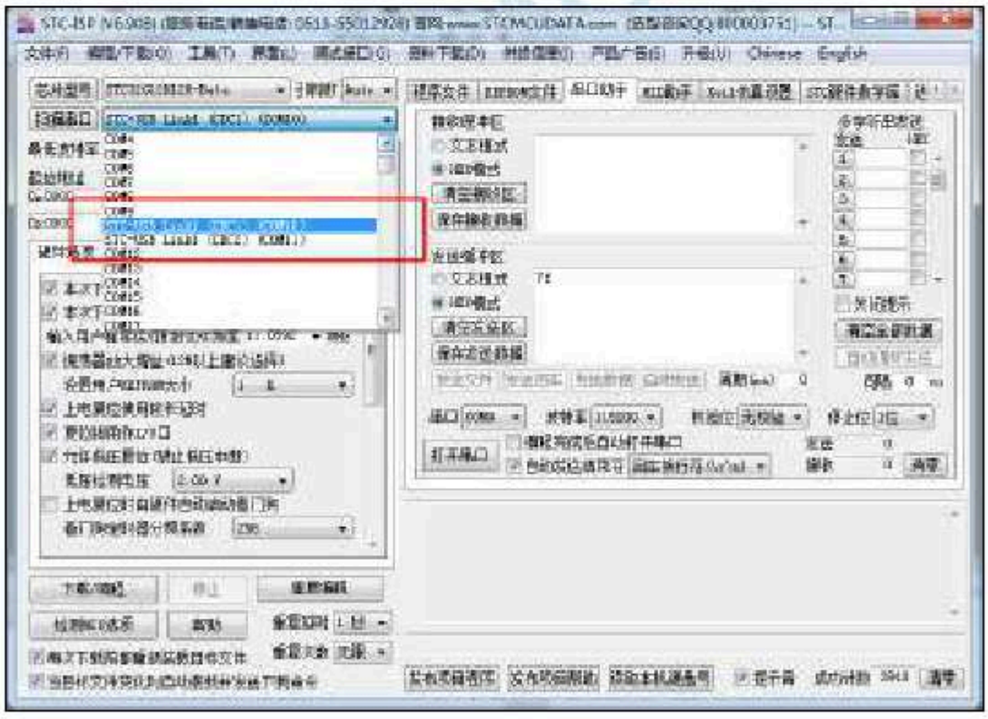
8 , It will pop up during installation "Windows Security" pop-up window, click "Always install this driver software"

9 , After the driver is successfully installed, the following screen will be displayed



10, The second CDC The installation method of the virtual serial port driver is similar to the first one. When the drivers of the two virtual seri
backup manager and Installed drivers can be found in the software STC-USB Link1D Virtual serial port. (Supports Downloading the software
You may need to click the "Scan serial port" button to rescan to find the serial port)





5.10 ISP Download a description of related hardware options

Hardware options	When will the option take effect?
☑ 选择使用内部IRC时钟(不选为外部时钟)	It needs to be powered up again to take effect
输入用户程序运行时的IRC频率 11.0592 MHz	Dynamic adjustment, effective immediately
☑ 振荡器放大增益(12M以上建议选择)	It needs to be powered up again to take effect
☑ 使用快速下载模式	Only related to this download
设置用户EEPROM大小 0.5 K	It needs to be powered up again to take effect
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序	Valid next time you download
☑ 上电复位使用较长延时	It needs to be powered on again to take effect
☑ 复位脚用作I/O口	, it needs to be powered on again to take effect
☑ 允许低压复位(禁止低压中断)	, it needs to be powered on again to take effect
低压检测电压 2.20 V	, it needs to be powered on again to take effect
☑ 低压时禁止EEPROM操作	, it needs to be powered on again to take effect
☐ 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	It needs to be powered up again to take effect
☑ 下次下载用户程序时擦除用户EEPROM区	Valid next time you download
☐ P2.0脚上电复位后为低电平(不选为高电平)	It needs to be powered on again to take effect
☐ 串口1数据线[RxD,TxD]切换到[P3.6,P3.7]	, it needs to be powered on again to take
☐ TxD脚是否直通输出RxD脚的输入电平	effect , it needs to be powered on again to take
☐ P3.7是否为强推挽输出	effect , it needs to be powered on again to take
☐ 芯片复位后是否将PWM相关的端口设置为开漏模	effect , it needs to be powered on again to take
☐ 在程序区的结束处添加重要测试参数	effect Write together every time you download

Need to be powered up again to take effect: After the option is modified, the target chip needs to be powered off once (power failure), and then powered on again before the new setting takes effect. This ISP download is valid

Dynamic adjustment, effective immediately: This option is only related to this ISP download. It does not affect the next download . After the option is modified, it will only take effect when the next download is made. The modification is invalid for this ISP download.

Only related to this download: Valid next time you download:

Write together every time you download : After selecting this option, the additional data will be written together during this download, which has nothing to do with the next downl

5.11 The user program is reset to USB pattern ISP How to download (not the system area for power failure)

When the project is in the development stage, it is necessary to repeatedly download the user code to the target chip for code verification, a Pattern pair
For normal download of the microcontroller, you need to short circuit the port to, And then power up the target chip again, from
It will make the burning steps of the project in the development phase more cumbersome. For this purpose, a special function register has be
When the user writes to this register, the software can be reset to the system area, and then it can be downloaded without power-up. 0x60 ISP

Note: When the user program is soft reset to the system area, if it will Already with the computer If the port is connected, the system code automatically enter the download mode and wait for download, it is not required to this time. USB ISP

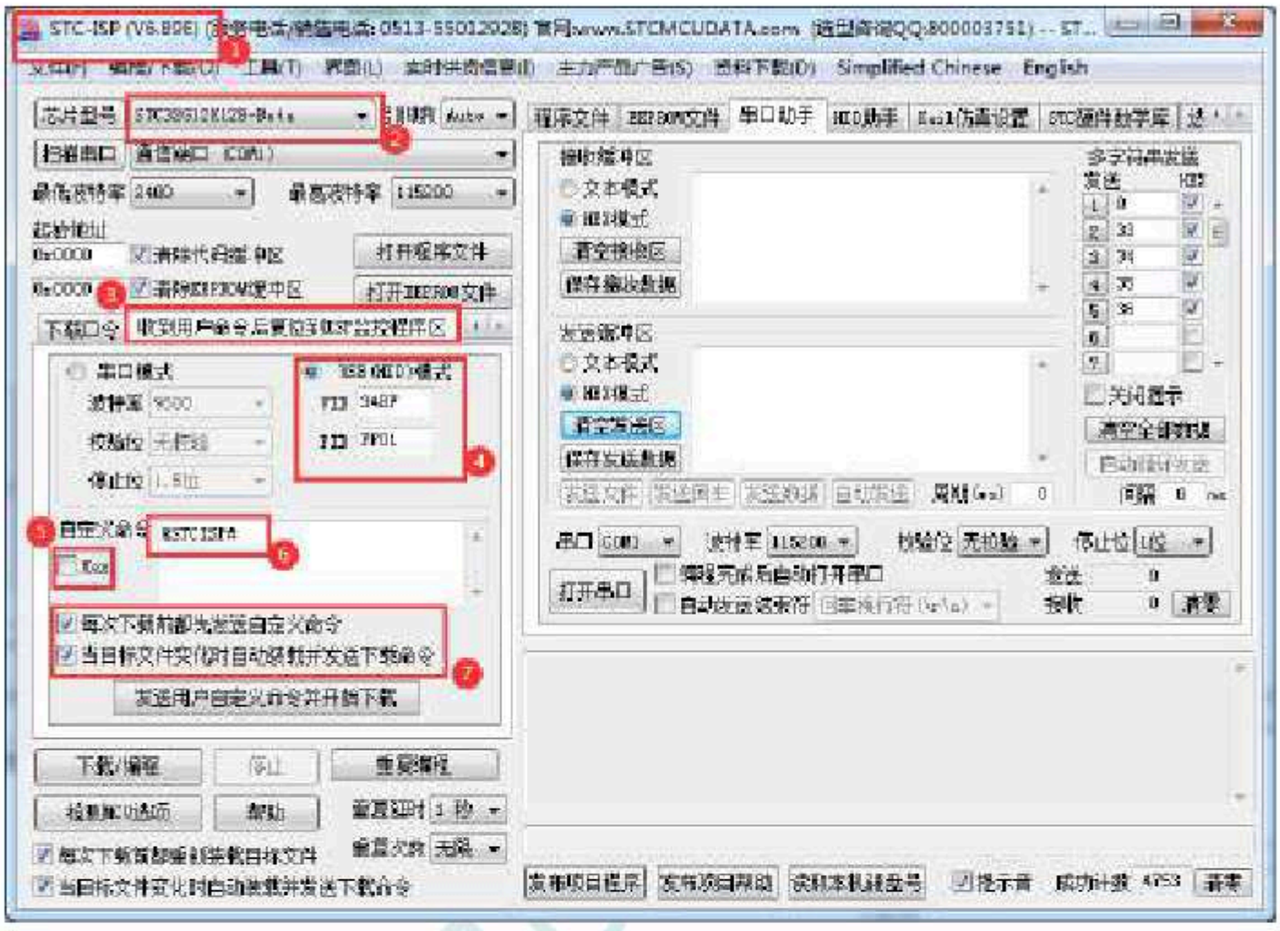
The following two methods are described below：

1. Use the buttons of the P3.2 port (non-USB items)

Use here P3.2 The button of the port triggers a soft reset and then power up the target chip again".
The method is different. In the main loop of the user program, The port is shorted to
it is determined that it can be downloaded in the system area, and when the port level is detected, a software reset is triggered. P3.2 0 P3.2
USB ISP Need to be reset to proceed When downloading, just manually click When the button is released, the level read by the user program from the port is, wh
Judgment in the program on the level of the sample program as follows：

```
//The test operating frequency is

//#include "stc8h. h"
#include "stc32g. h"                                    // See header fileDownload software

void main()
{
    EAXFR = 1;                                          // Enable access
    CKCON = 0x00;                                       // Set the external data bus speed to the fastest
    WTST = 0x00;                                        // Set the program code waiting parameters ,
                                                        //Assign a value to The speed of executing the program is set to the fastest

    P3M0 = 0x00;
    P3M1 = 0x00;
    P32 = 1;
    while (1)
    {
        if (! P32) IAP_CONTR = 0x60;                    // When detected When the level is low
                                                        //The software is reset to the system area
        ...                                             //User code
    }
}
```

2. Use the user download command sent by the STC-ISP download software (USB project)



1, Download the latest version of Download software

2, Select the correct MCU model , turn

on "Reset to after receiving the user command" Monitor Program Area "Options page

4, Select " USB(HID) Mode", and set Equipment of USB VID and PID , STC In the provided example VID for" 34BF" ,

for" PID FF01" Mode or

5, choose HEX text mode

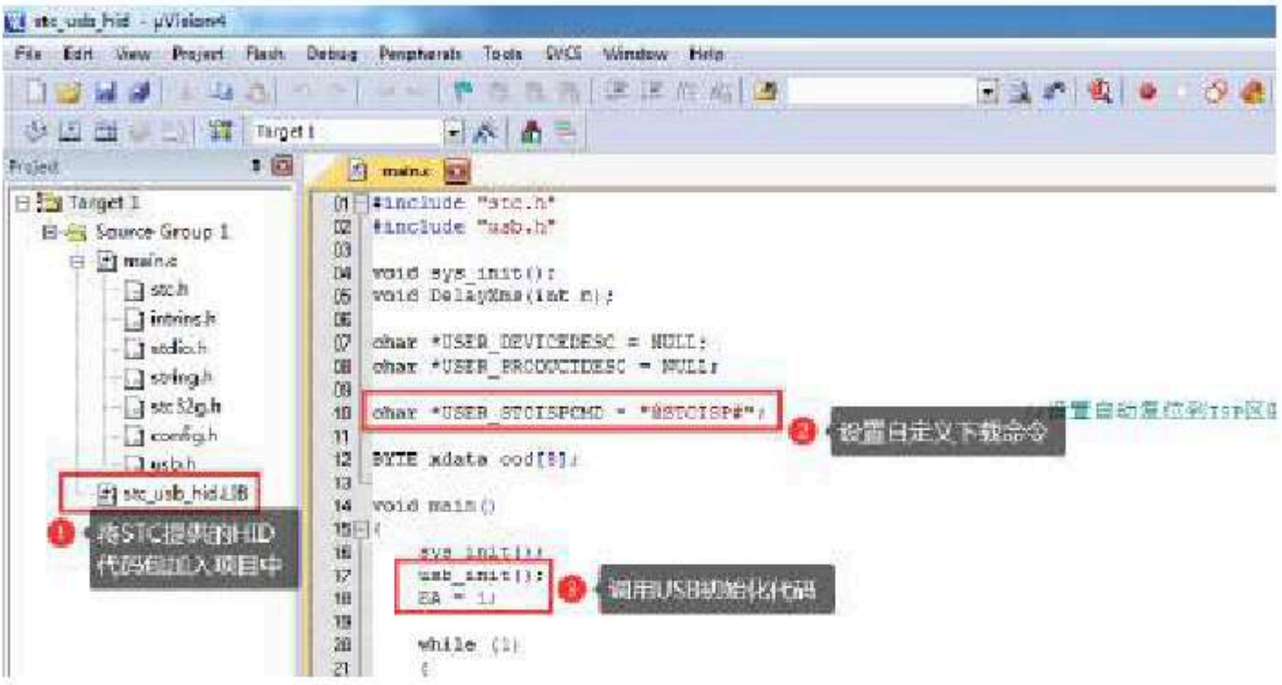
6 , Set up a custom download command, it needs to be consistent with the custom

7 command in the code , select these two items, when the target code is reprogrammed After downloading the software, a reset command will be automatically s

STC-USB Pattern of ISP download

Note: If you need to use this mode, you must STC Provided"

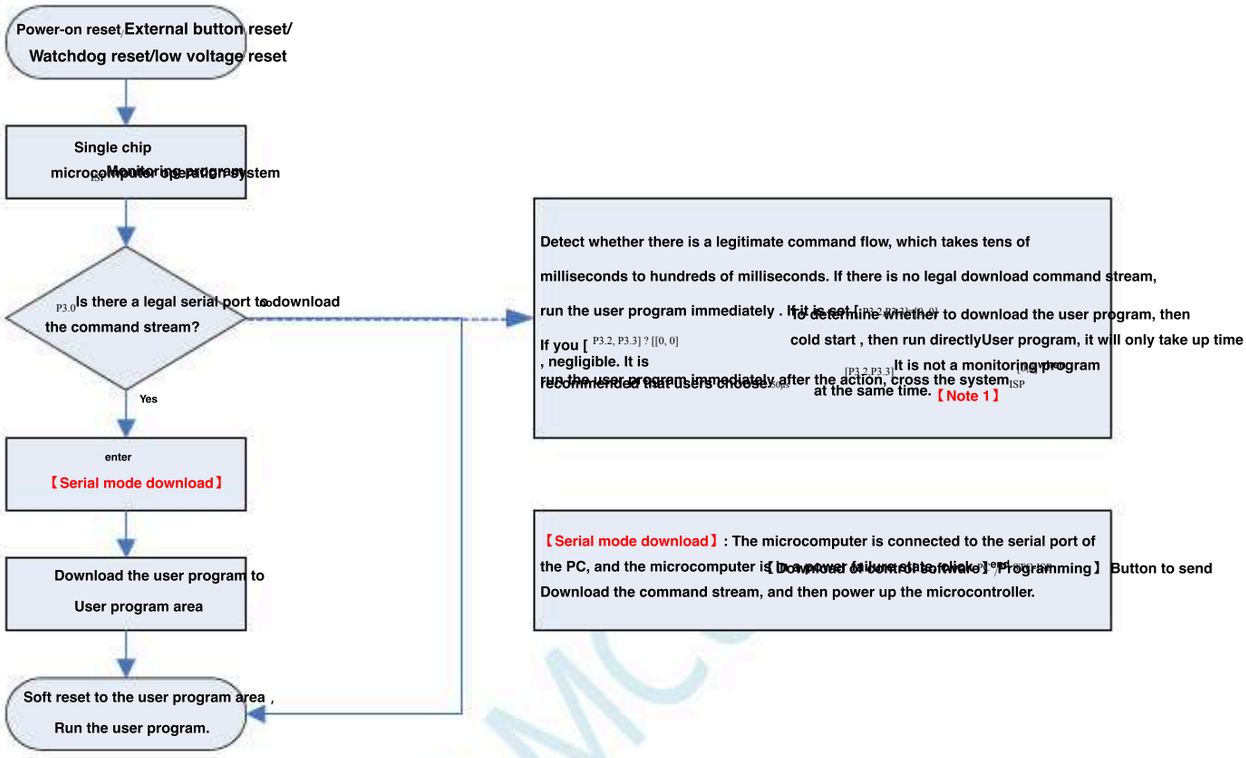
Set up a custom download command in the way shown in the figure. "Add the code base to the project and follow the steps below stc_usb_hid. li



For detailed codes, please refer to "on the Experimental board demonstration program in the package of printing data information For debugging"

5.12 ISP Download process and typical application circuit diagram

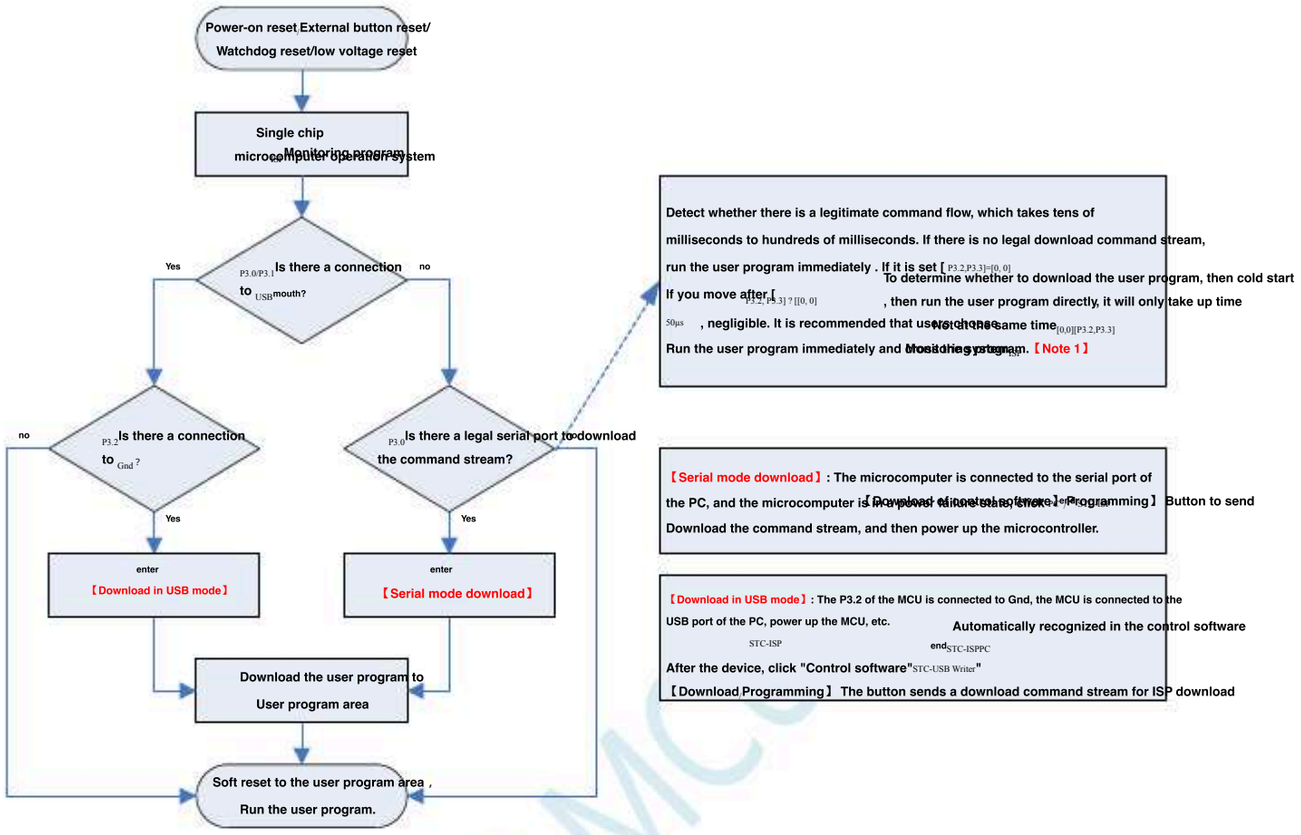
5.12.1 ISP Download flow chart (serial port download mode)



Note: [P3.0, P3.1] Make a download, For simulation (download, Simulation interface is, only available recommended that users put the serial port in
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,
When downloading the program, check "Next cold start" on the software, you must download the program at that time."

[Note 1]: The burn protection pin of the new chips of STC15, STC8 series and later is P3.2/P3.3, and the burn protection pin of the earlier chips is P1.0/P1.1.

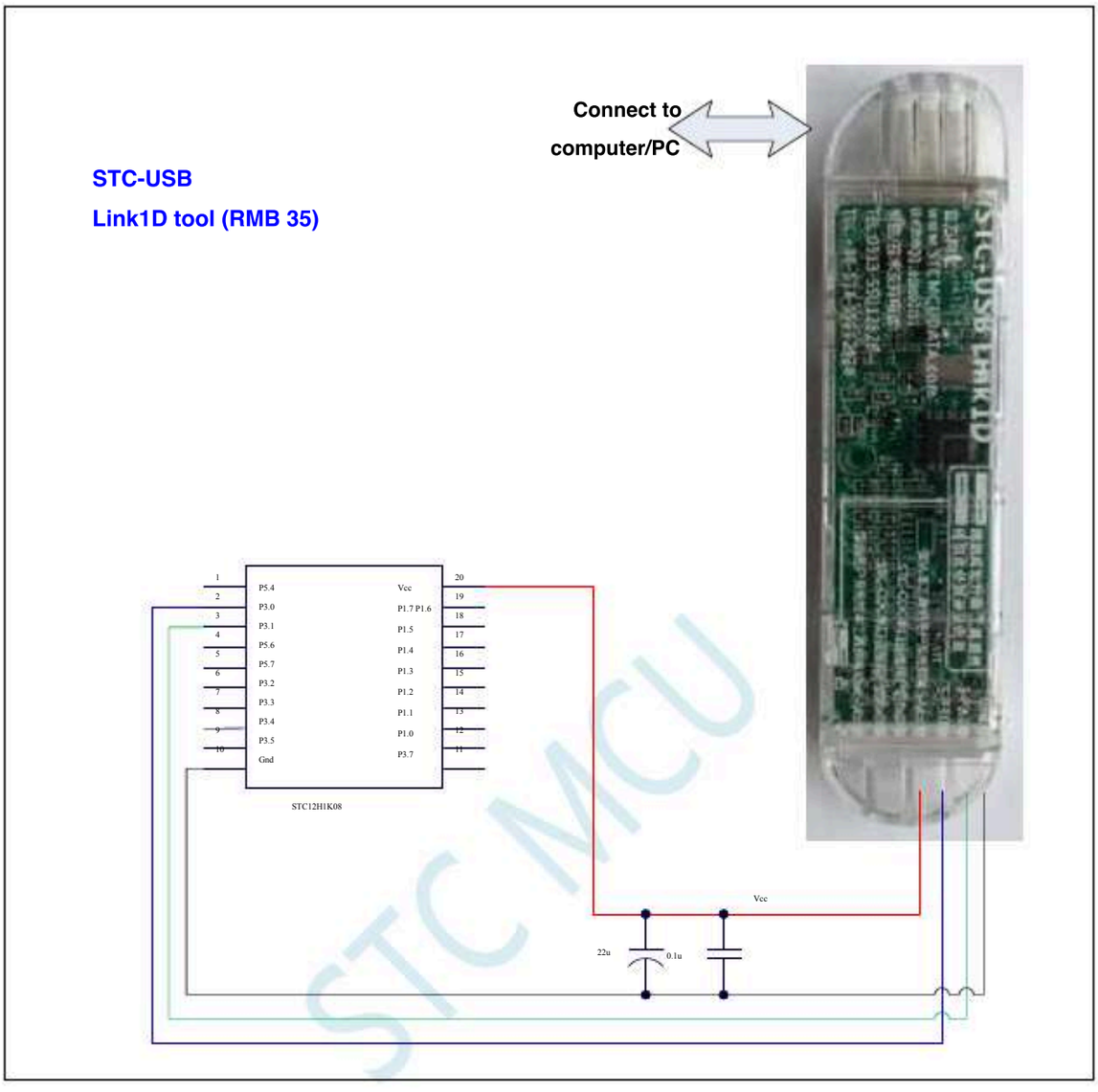
5.12.2 ISP Download flow chart (hardware/Software simulation USB+ Serial port mode)



Note: [P3.0, P3.1] Make a download/For simulation (download/Simulation interface is, only available recommended that users put the serial port in
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,
When downloading the program, check "Next cold start" on the software P3.0/P3.1 for 0/0 you must download the program at that time."

[Note 1]: The burn protection pin of the new chips of STC15, STC8 series and later is P3.2/P3.3, and the burn protection pin of the earlier chips is P1.0/P1.1.

5.12.3 use STC-USB Link1D Tool download, support online and offline download

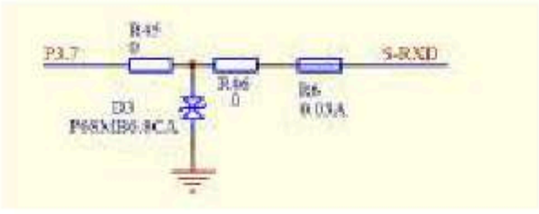


ISP Download steps：

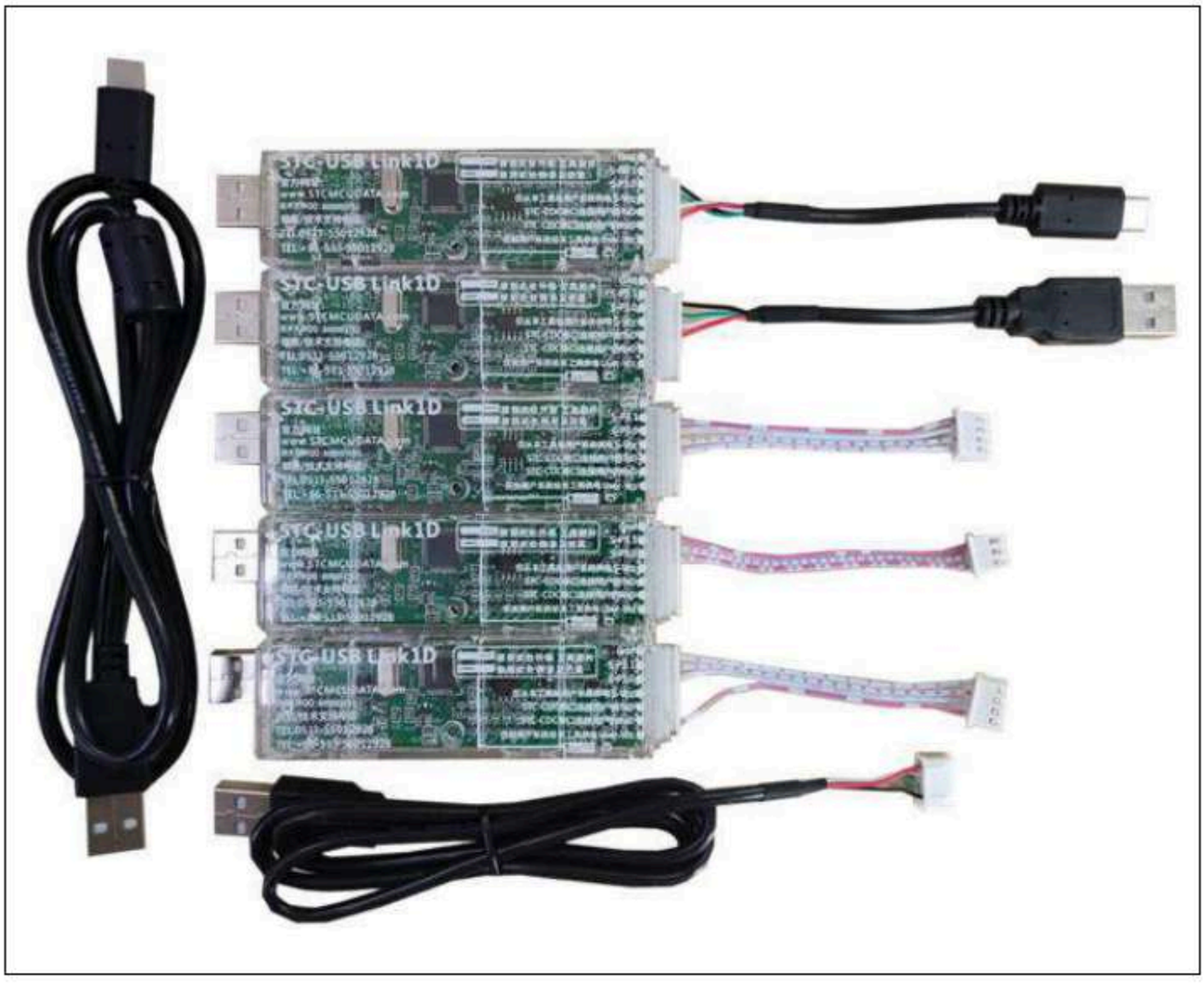
、1 According to the connection method TC-USB Link1D Connect to the target chip shown in the figure, 2' "Download" in the download software "Program" button click download 3' start ISP

Note: If the use STC-USB Link1D To supply power to the target system, the total current of the target system cannot be greater than 200mA Otherwise it will lead to the following fails, the download fails.

Note: It is currently found to be used USB Line power supply When downloading, due to the line is too thin, ISP The pressure drop on the line is too large, to ISP The power supply is insufficient when downloading, supply USB When downloading, be sure to Strengthen the line.



If the serial port on the user board receives There is a strong pull-up or a strong pull-down on the mouth (for example, in the receiving state). P3.0 the pin STC-USB Link1D May not be able to download, the user can send the fuse on the tool STC-USB Link1D 30mA R6 use 0 Ohm electric Resistance replacement (actual measurement 20mA The static resistance value of the fuse is 10~15 Ohm)



above RMB35 It is equipped with all the above lines, and it subsidizes everyone at a loss.

STC-USB Link1D Tool interfacedescription

STC-USB Link1D The tool is a serial The upgraded version and functions are in STC-USB Two more have been added on the basis of STC-CDC port, which can be used as USB Link1 Go to the serial port to use.

tools a universal STC-USB Link Please refer to the Appendix chapter for precautions for use

Front view of the tool

User-Vcc, External power supply to the tool

2 STC-CDC Serial port send pin , S-RxD, The first group is connected to the receiving pin of the serial port of the user's MCU

Serial port receiving pin , , the first 2 group STC-CDC S-TxD Connect to the sending pin of the serial port of the user's MCU

, This tool supplies power to the external system S-Vcc

Hardware simulation data cable, , Link1 S-P3.0 Downloaded serial port sending pin, Link1 Proceed 1 Group 1 STC-CDC Serial port send pin

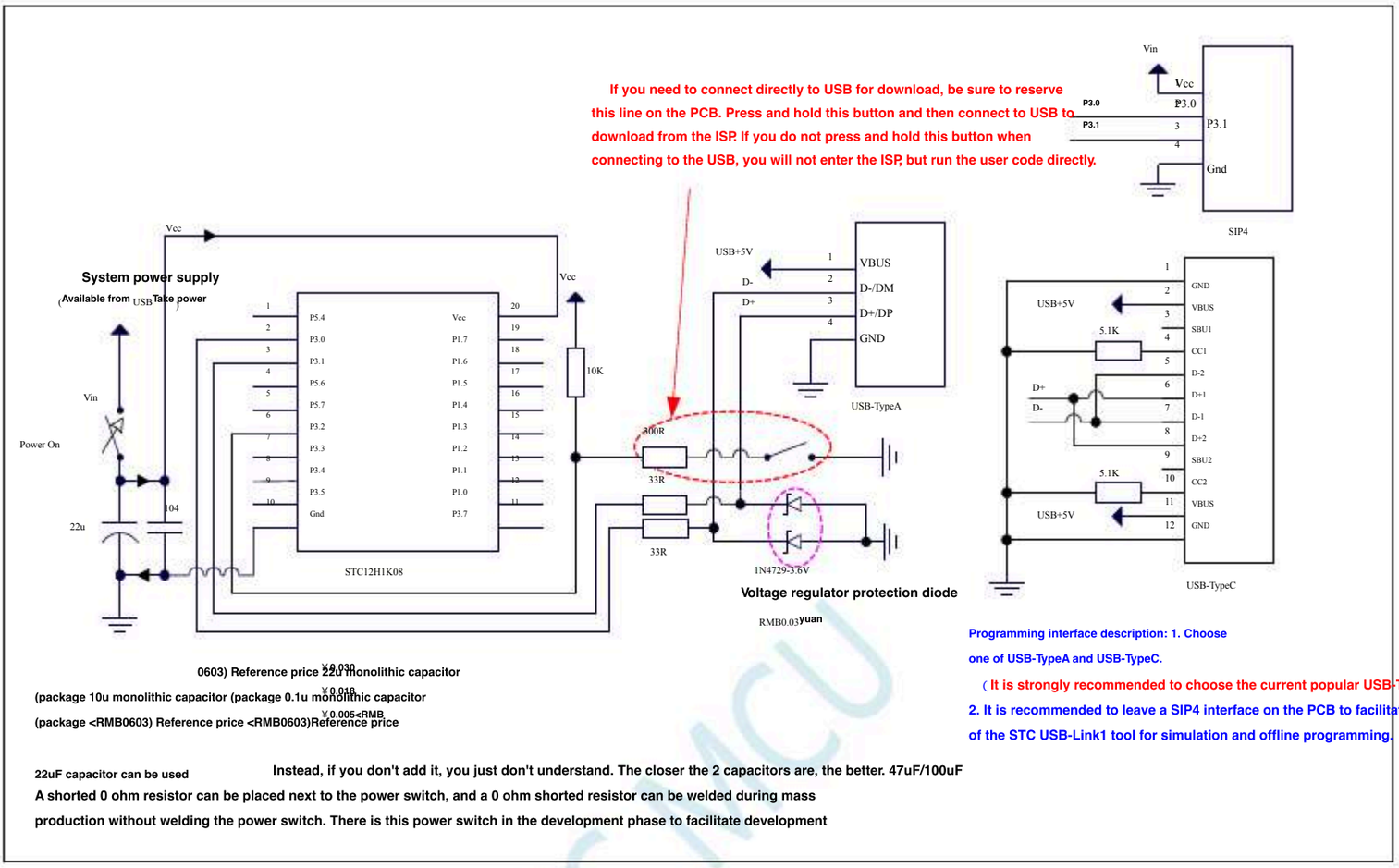
Link1 Hardware simulation clock line, S-P3.1 Link1 proceed ISP Downloaded serial port 1 Group 1 receiving pin, STC-CDC Serial port receiving pin

Gnd Ground wire

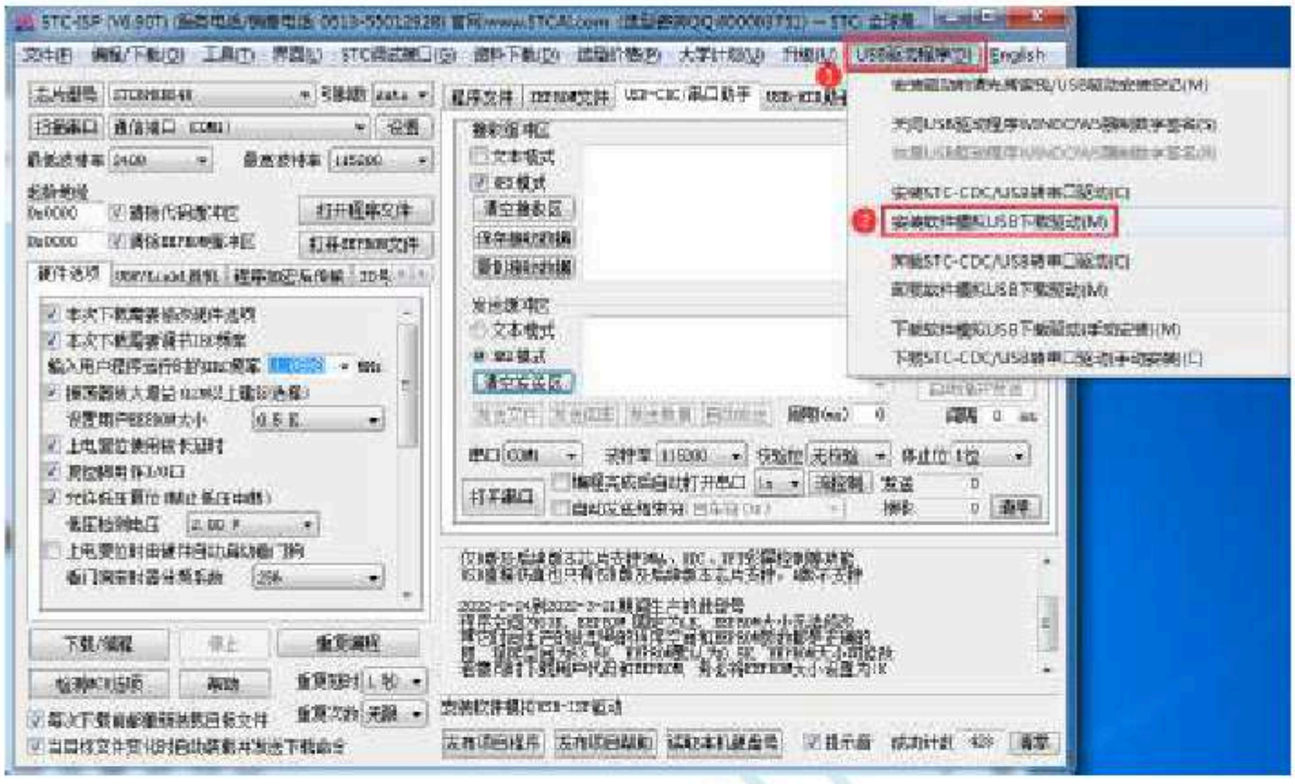
Reverse side view of the tool

Pin number	Interface name	Interface function
1	User-Vcc	Only the user system supplies
2	S-RxD	power to this tool 2 Group 1 STC-CDC The sending pin of the serial port, the receiving pin of the serial port connected to
3	S-TxD	2 Group 1 STC-CDC the user's MCU, the receiving pin of the serial port, and the sending pin of the serial port
4	S-Vcc	Only power the user's system
5	S-P3.0	from this tool Use in progress Link1D ISP The serial port sending pin when downloading, connected to the target MCU P3.0
		Use in progress Link1D SWD The data pin during hardware simulation, connected to the target MCU SWDDAT
		1 Group 1 STC-CDC The sending pin of the serial port is connected to the receiving pin of the serial port of the
6	S-P3.1	Use in progress Link1D ISP The serial port receiving pin when downloading, connected to the target MCU P3.1
		Use in progress Link1D SWD The clock pin during hardware simulation, connected to the target MCU SWDCLK
		1 Group 1 STC-CDC The receiving pin of the serial port is connected to the sending pin of the serial port of the
7	Gnd	Ground wire

5.12.4 Software simulation (USB direct ISP Download, it is recommended to try, simulation is not supported system)



When Unlocking STC USB of STC8G/STC8H of MCU , Basically support software simulation Download the user program, because now is STC-ISP Communication protocol, regardless of any version of the operating system, the driver does not need to be installed. Download the software and run it as shown below Install the driver in the place shown.



ISP **Download steps:**

1、 D-/P3.0, D+/P3.1 **The port is connected and**

2、**will** P3.2 **with** GND **shorted**PC-USB、**On the board of the experimental box, press the button**

3、**Power up the target chip again. If the target chip has been powered off, it can be powered on directly; if the target chip is powered on, it ne**

The chip is powered on after a power failure (cold start), automatically recognized in the download software" STC-ISP **Automatically recognized in the download software"** STC USB Writer (HID1) **"Identify**

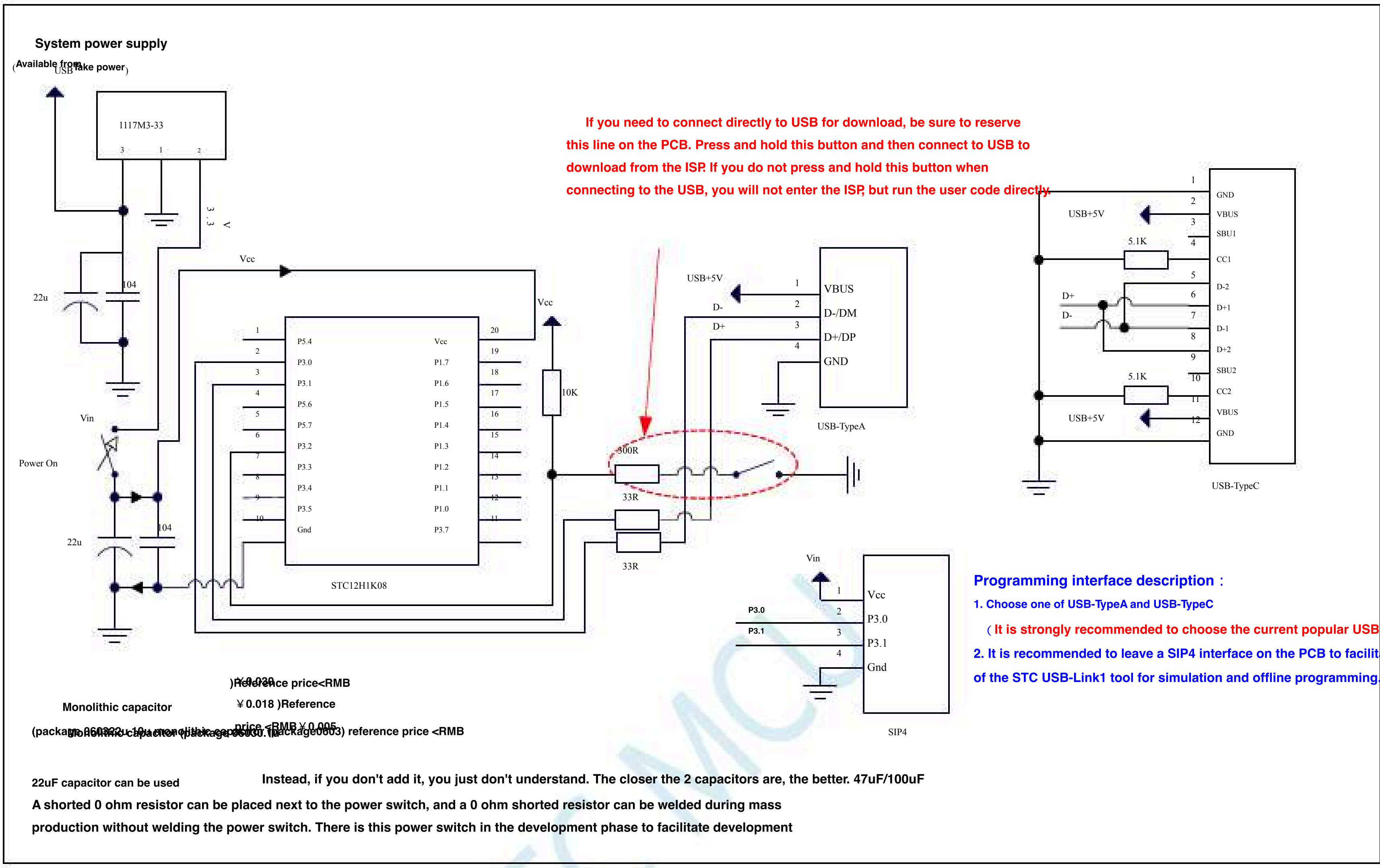
After coming with 3、**The status has nothing to do with it.**

4、**Click "Download" in the download software The "Program" button (**/ **The operation sequence of the download is different from that of the serial**

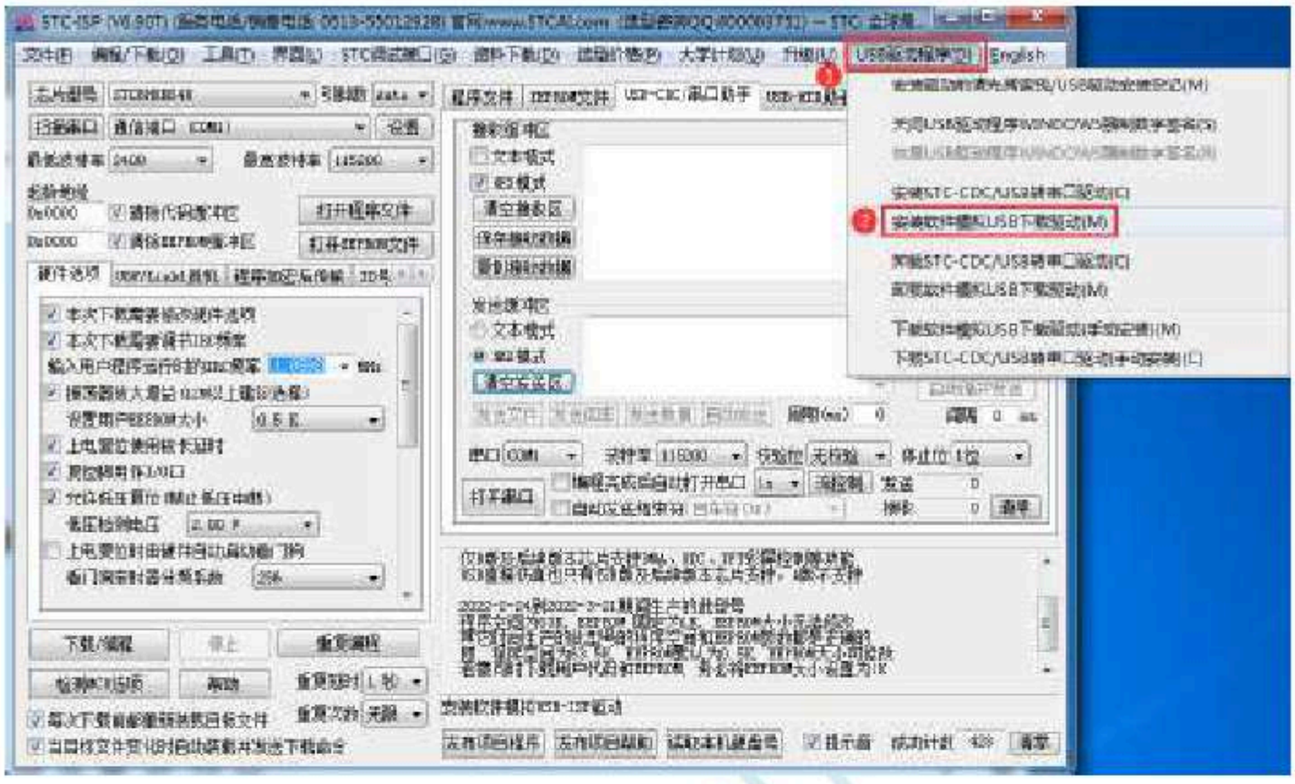
USB **Click the download button first, and be sure to** STC USB Writer (HID1) **"After the device is installed, you can click the downloa**

wait until the computer recognizes " Download".)

5.12.5 Software simulation USB direct ISP Download, it is recommended to try, simulation system) is not supported



What can be drawing STC USB of STC8G/STC8H of MCU , Basically support software simulation Download the user program, because now is STC-ISP Communication protocol, regardless of any version of the operating system, the drivers that are installed as shown be Install the driver in the place shown.



ISP **Download steps:**

1、 D-/P3.0, D+/P3.1 **The port is connected and**

2、**will** P3.2 **with** GND **shorted**PC-USB, **On the board of the experiment box, press the** Reset button

3 **, Power up the target chip again. If the target chip has been powered off, it can be powered on directly; if the target chip is powered on, it ne**

The chip is powered on after a power failure (cold start), automatically recognized in the downloaded software" STC USB Writer (HID1) **"Identify**

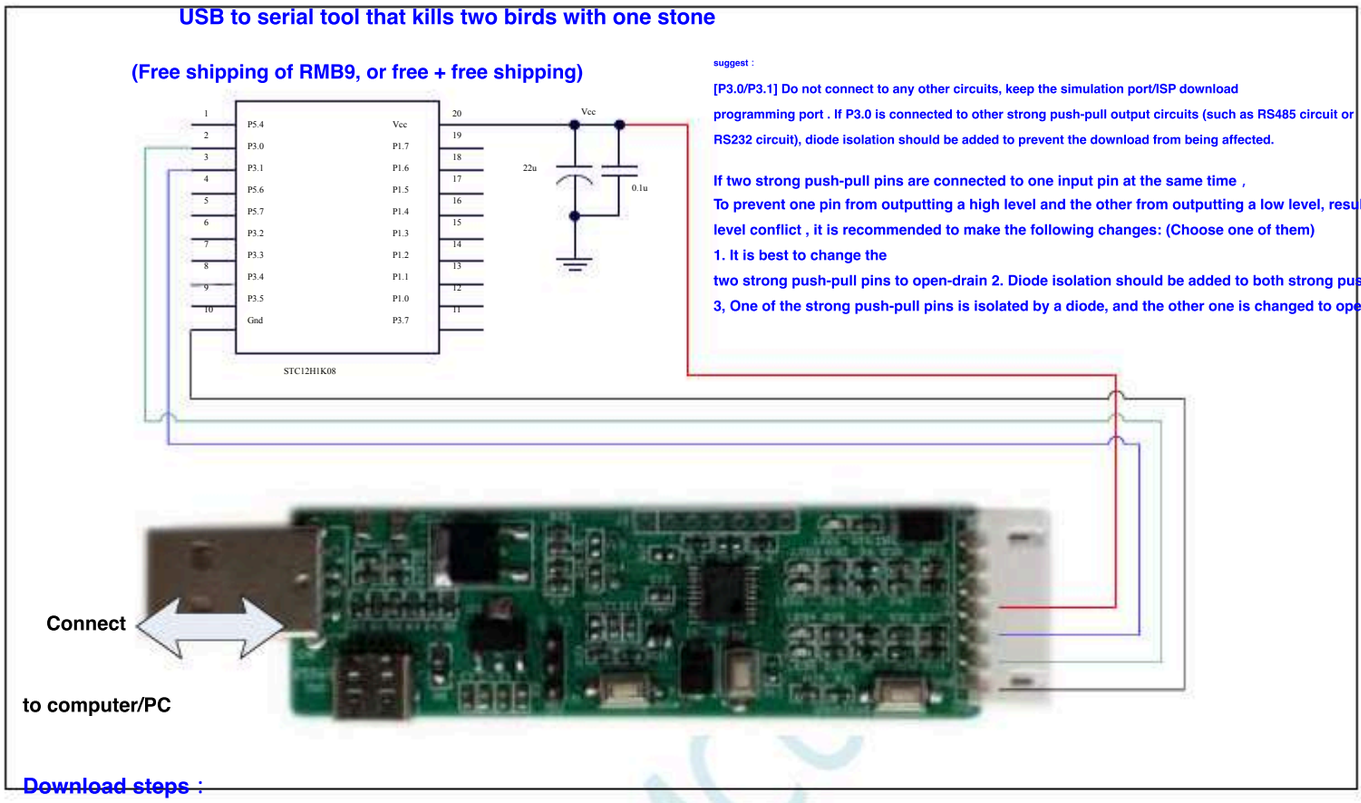
After coming with **The status has nothing to do with it.**

4 **, Click "Download" in the download software The "Program" button (**The operation sequence of the download is different from that of the serial

USB **Click the download button first, and be sure to** STC USB Writer (HID1) **"After the device is installed, you can click the downloa**

wait until the computer recognizes " Download".)

5.12.6 Use one stone to kill two birds with one stone USB Download the serial port tool



Download steps :

ISP

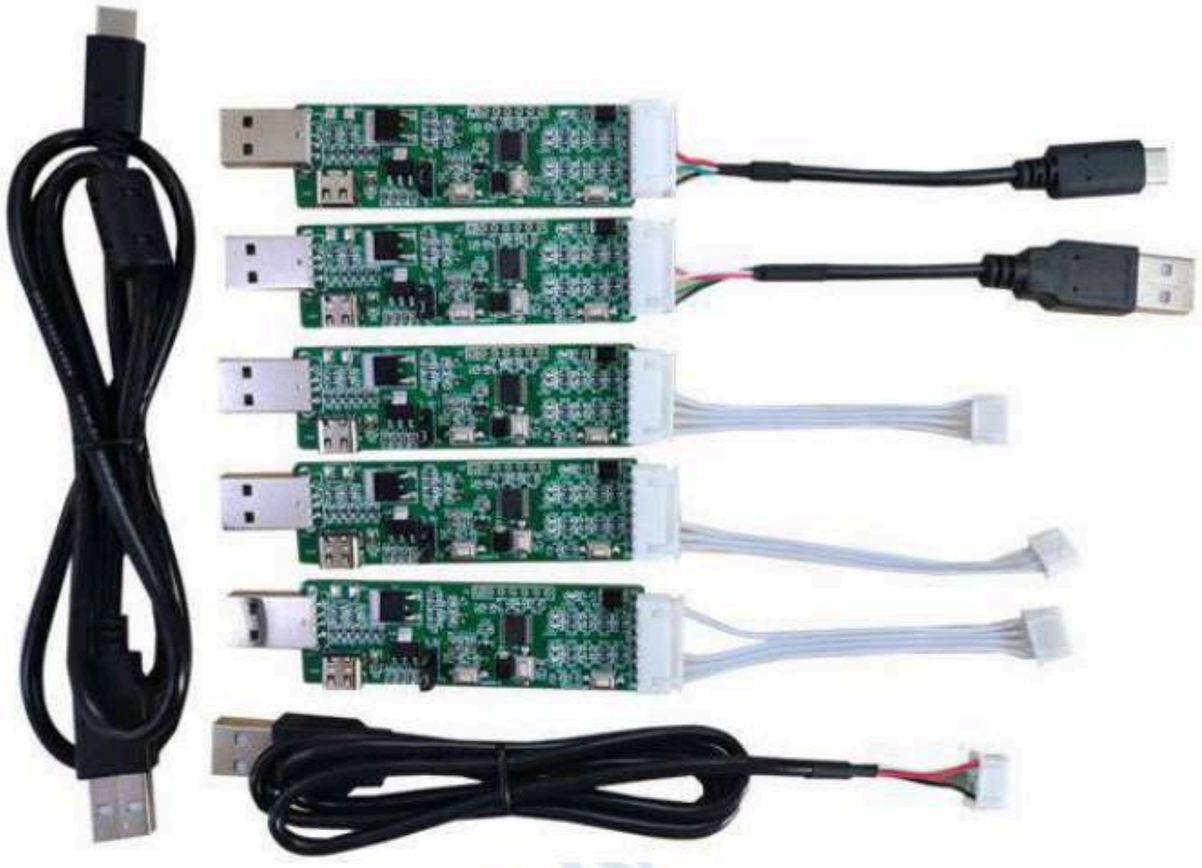
¦ According to the connection method shown in the figure Connect to the serial port tool and the target chip USB

Download click the download software "Program" button

download 3, start ISP

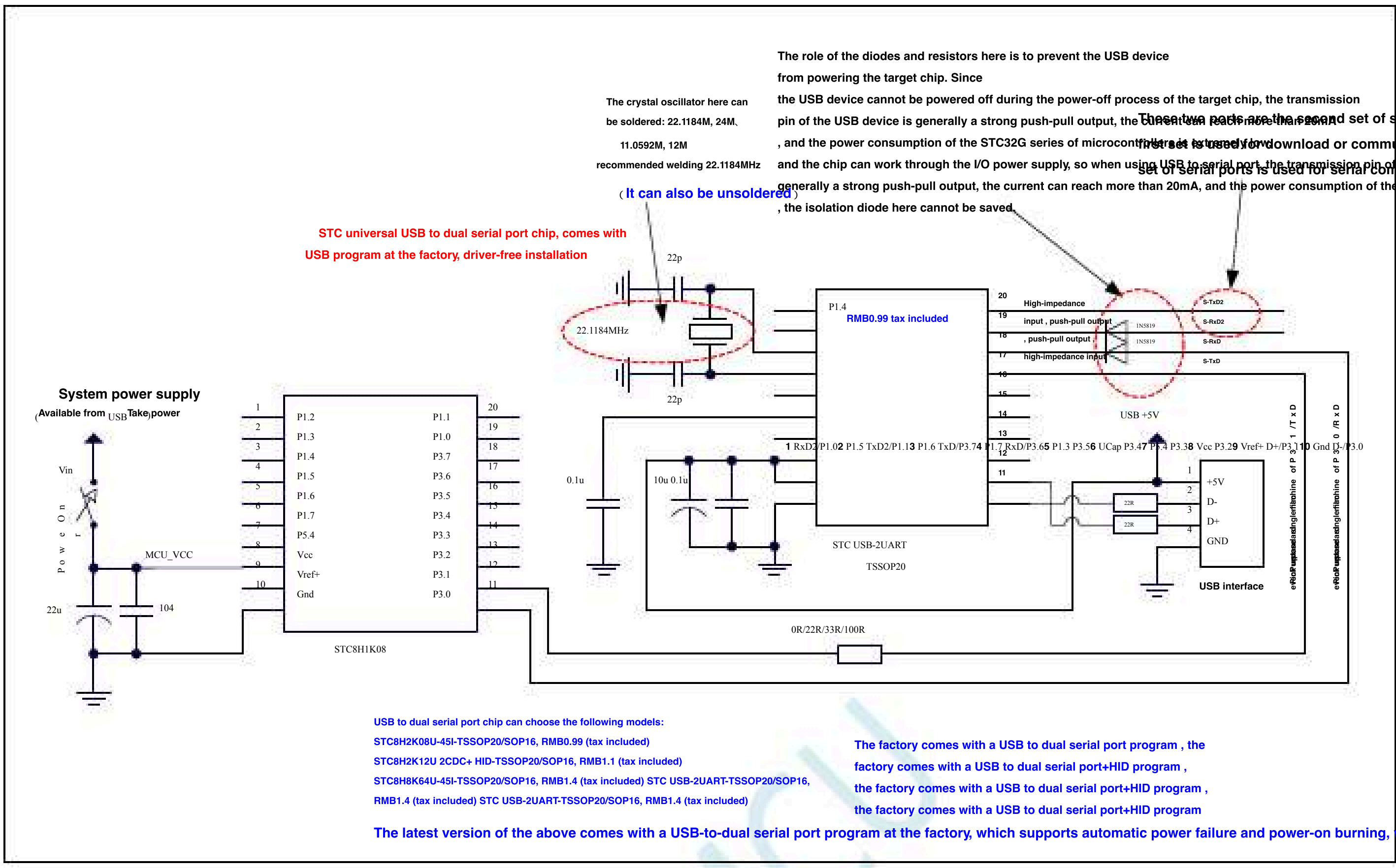
Note: It is currently found to be used ISP Long power supply When downloading, due to the line is too thin, in The pressure drop on the line is too large,

to ISP The power supply is insufficient when downloading, USB power supply When downloading, the USB Strengthen the line.



A USB-to-serial port tool that kills two birds with one stone (free shipping for RMB 9, only one SIP7-SIP4 cable is included, and everyone will be subsidized at a loss)

5.12.7 use USB To dual serial port/TTL Download (with external crystal oscillator)

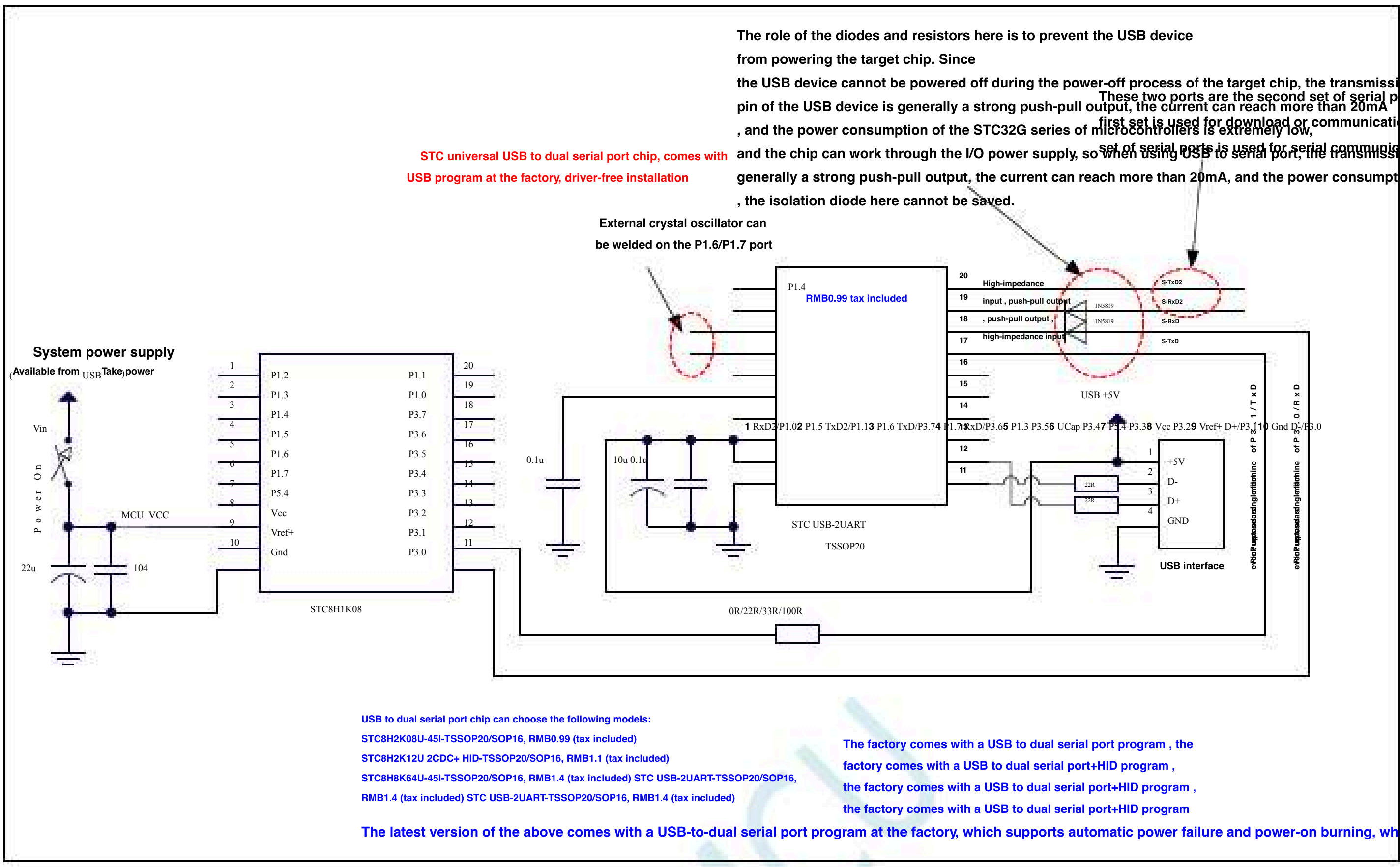


ISP **Download steps :**

1. Power off the target chip, be careful STC USB-2UART "Chip power failure" The transmission pin of the not to give it"
2. Due to " STC USB-2UART chip is a strong push-pull output, which must be on the target chip. P3.0 Kouhe" STC USB-2UART "A diode is connected in series between the transmitting pins, otherwise the target chip cannot be completely powered of The goal.
3. click STC-ISP "Download" in the download software/"Program" button
4. Power up the target chip start ISP

Note: It is currently found to be used USB the power supply is too long when downloading, due to the line is too thin, the pressure drop on the line is too large, re ISP The power supply is insufficient when downloading, so please be sure to use USB power supply, when downloading, strengthen the line.

5.12.8 use USB To dual serial port/TTL Download (no external crystal oscillator)



ISP **Download steps :**

、1 **Power off the target chip, be careful** STC USB-2UART **"Chip power failure" The transmission pin of the not to give it"** 2、 **Due to "** STC USB-2UART **chip is a strong push-pull output, which must be on the target chip.** P3.0 **Kouhe"** STC USB-2UART **"A diode is connected in series between the transmitting pins, otherwise the target chip cannot be completely powered of The goal.**

STC-ISP 3、 **click Download" in the download software "Program" button**

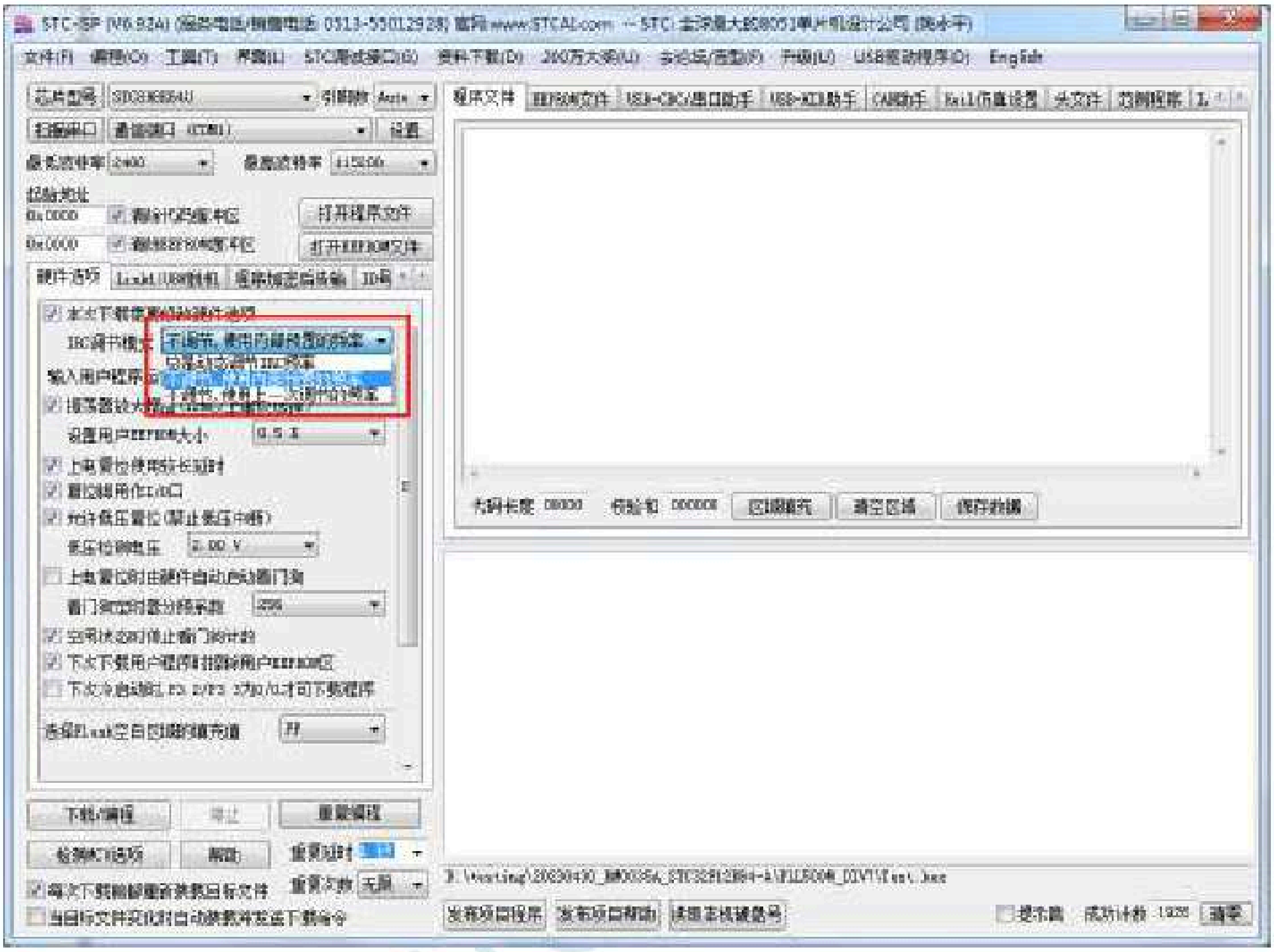
、

4 **Power up the target chip start** ISP

Note: It is currently found to be used the power supply is carried out When downloading, due to the line is too thin, the pressure drop on the line is too large, re

ISP **The power supply is insufficient when downloading, so please use USB power supply at the time of download, be sure to use Strengthen the line.**

Note: If you use one without an external crystal oscillator USB To dual serial port TTL When downloading, it is highly recommended Download options ISP "Select IRC Adjustment mode "
Select the option "Do not adjust, use the internally preset frequency", which can avoid substituting the frequency error of the tool itself USB to dual serial port TTL
of the tool itself without an external crystal oscillator into the target chip when adjusting the frequency. As shown below :



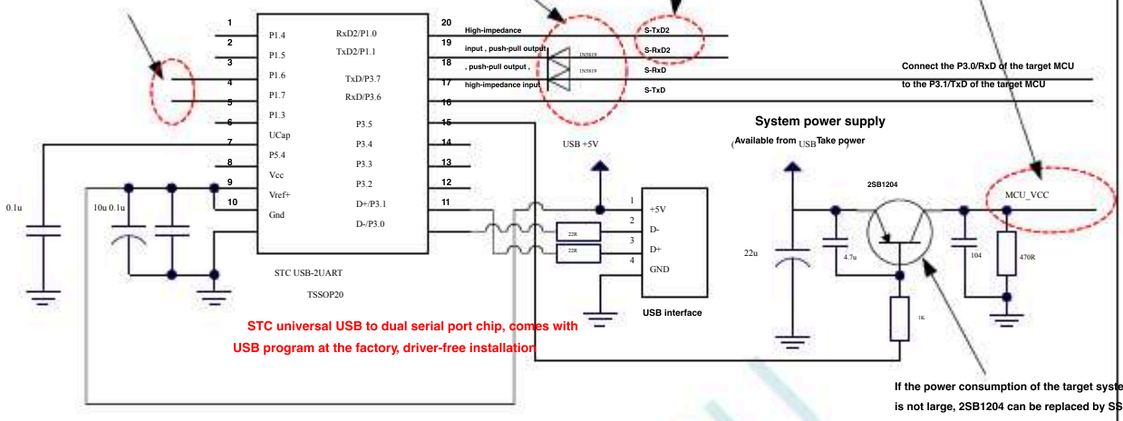
5.12.9 use USB To dual serial port/TTL Download (automatic power failure/Power on)

The role of the diodes and resistors here is to prevent the USB device from powering the target chip. Since the USB device cannot be powered off during the power-off process of the target chip, the transmission pin of the USB device is generally a strong push-pull output, the current can reach more than 20mA , and the power consumption of the STC32G series of microcontrollers is extremely low, and the chip can work through the I/O power supply, so when using USB to serial port, the transmission pin of the USB device is generally a strong push-pull output, the current can reach more than 20mA, and the power consumption of the STC32G series of microcontrollers is extremely low.When TTL downloads the program to the chip is powered on, the P3.5 output is low, and MCU_VCC is in the power supply state. When it detects that an ISP download is required, MCU_VCC will automatically control the power failure for 0.5s and then power on for ISP download. Power will continue to be powered after the download is complete , the isolation diode here cannot be saved.

These two ports are the second set of serial ports, the first set is used for download or communication , and the second set of serial ports is used for serial communication.

External crystal oscillator can be welded on the P1.6/P1.7 port

RMB0.99 tax included



If the power consumption of the target system is not large, 2SB1204 can be replaced by SS8550

STC universal USB to dual serial port chip, comes with USB program at the factory, driver-free installation

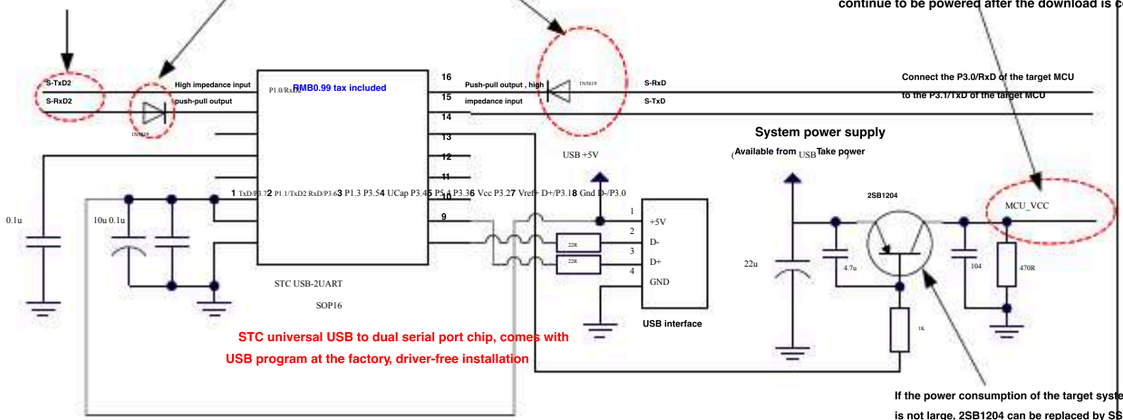
USB to dual serial port chip can choose the following models:

STC8H2K08U-45I-TSSOP20/SOP16, RMB0.99 (tax included)	The factory comes with a USB to dual serial port program , the
STC8H2K12U 2CDC+HID-TSSOP20/SOP16, RMB1.1 (tax included)	factory comes with a USB to dual serial port+HID program ,
STC8H8K64U-45I-TSSOP20/SOP16, RMB1.4 (tax included)	the factory comes with a USB to dual serial port+HID program ,
STC USB-2UART-TSSOP20/SOP16, RMB1.4 (Tax included)	the factory comes with a USB to dual serial port+HID program

The latest version of the above comes with a USB-to-dual serial port program at the factory, which supports automatic power failure and power-on burning, which can save isolation diodes.

The role of the diodes and resistors here is to prevent the USB device from powering the target chip. Since the USB device cannot be powered off during the power-off process of the target chip, the transmission pin of the USB device is generally a strong push-pull output, the current can reach more than 20mA , and the power consumption of the STC32G series of microcontrollers is extremely low, and the chip can work through the I/O power supply, so when using USB to serial port, the transmission pin of the USB device is generally a strong push-pull output, the current can reach more than 20mA, and the power consumption of the STC32G series of microcontrollers is extremely low.When TTL downloads the program to the chip is powered on, the P3.5 output is low, and MCU_VCC is in the power supply state. When it detects that an ISP download is required, MCU_VCC will automatically control the power failure for 0.5s and then power on for ISP download. Power will continue to be powered after the download is complete , the isolation diode here cannot be saved.

These two ports are the second set of serial ports, the first set is used for download or communication , and the second set of serial ports is used for serial communication.



If the power consumption of the target system is not large, 2SB1204 can be replaced by SS8550

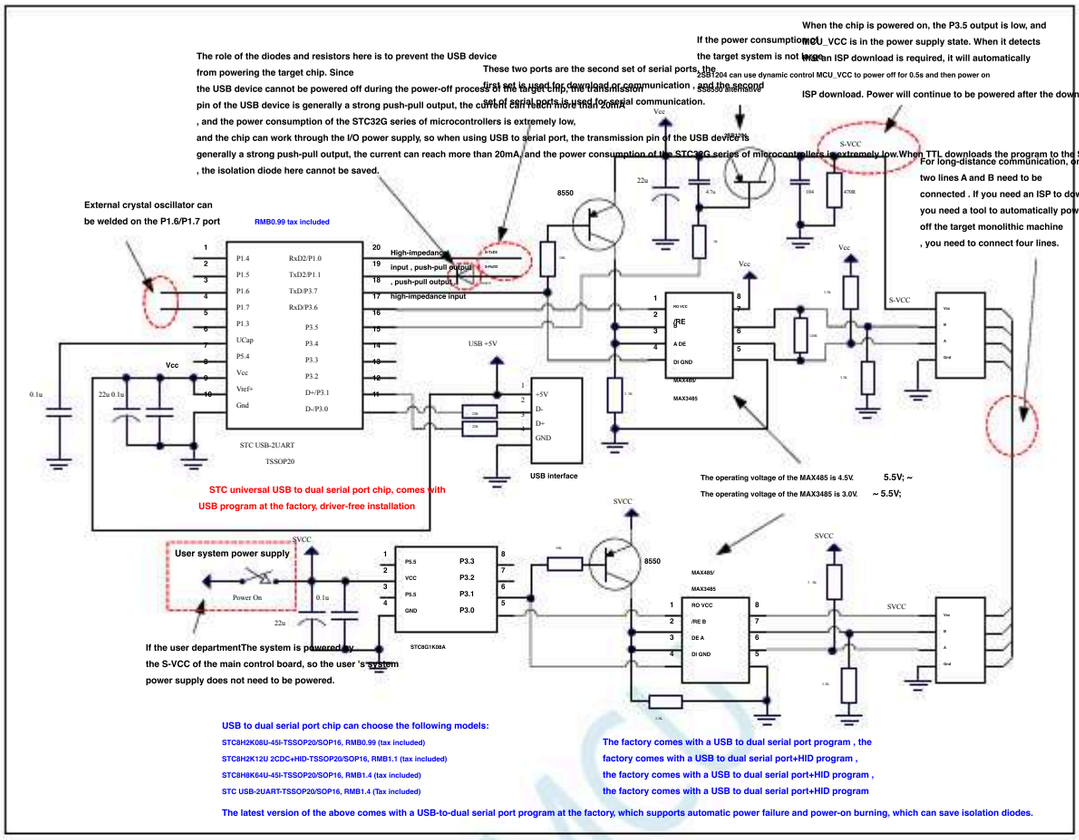
STC universal USB to dual serial port chip, comes with USB program at the factory, driver-free installation

USB to dual serial port chip can choose the following models:

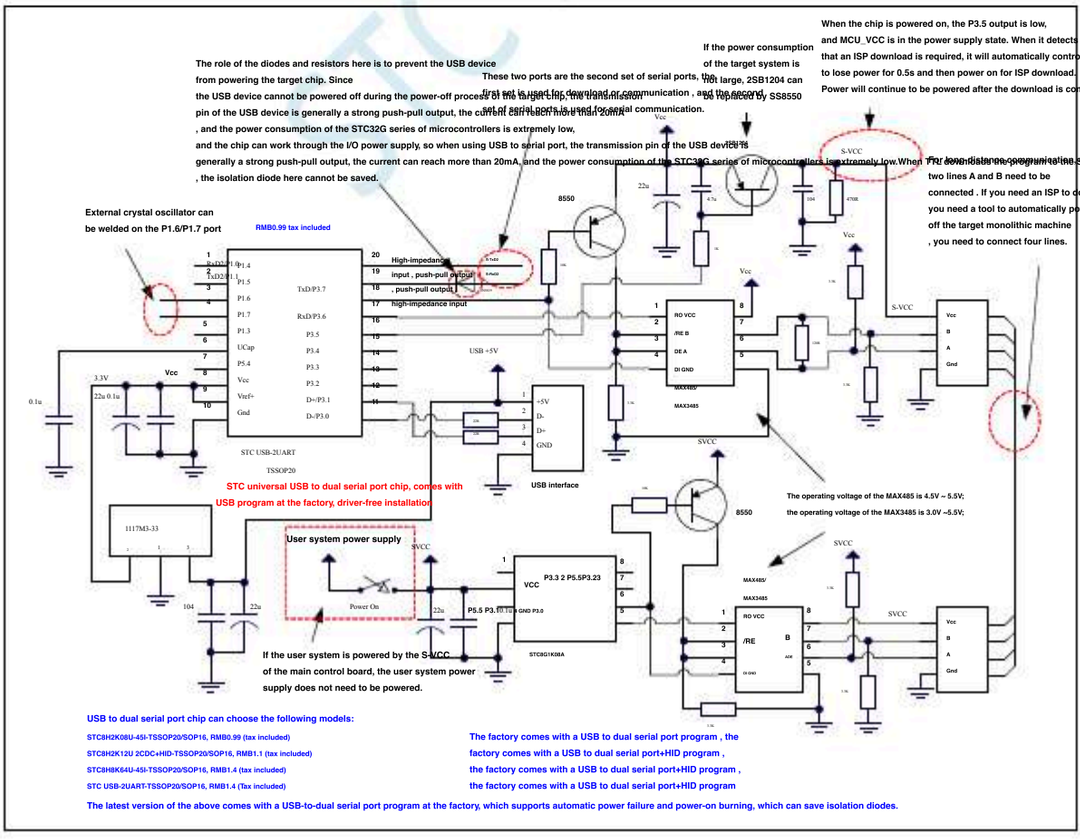
STC8H2K08U-45I-TSSOP20/SOP16, RMB0.99 (tax included)	The factory comes with a USB to dual serial port program , the
STC8H2K12U 2CDC+HID-TSSOP20/SOP16, RMB1.1 (Tax included)	factory comes with a USB to dual serial port+HID program ,
STC8H8K64U-45I-TSSOP20/SOP16, RMB1.4 (tax included)	the factory comes with a USB to dual serial port+HID program ,
STC USB-2UART-TSSOP20/SOP16, RMB1.4 (tax included)	the factory comes with a USB to dual serial port+HID program

The latest version of the above comes with a USB-to-dual serial port program at the factory, which supports automatic power failure and power-on burning, which can save isolation diodes.

5.12.10 use USB To dual serial port/RS485 Download (5.0V)



5.12.11 use USB To dual serial port/RS485 Download (3.3V)



STC-ISP RS485 **Downloading the software The relevant setting interface is as shown below :**

The detailed description of the settings

:Setting control RS485 Receiving control effective level

is as follows : "Receive control

:Setting control RS485 , sending control effective level control function":

settings" , "Send control settings

RS485 Set the target mcu next time ISP Enable when downloading RS485 Control (special

", "Enable the target chip next time

Don't pay attention, if the user product needs to Function, you need to check this option every time you download)

you download " This time use RS485 Make a control download" Through STC-ISP Download software to use The mode downloads the target MCU.

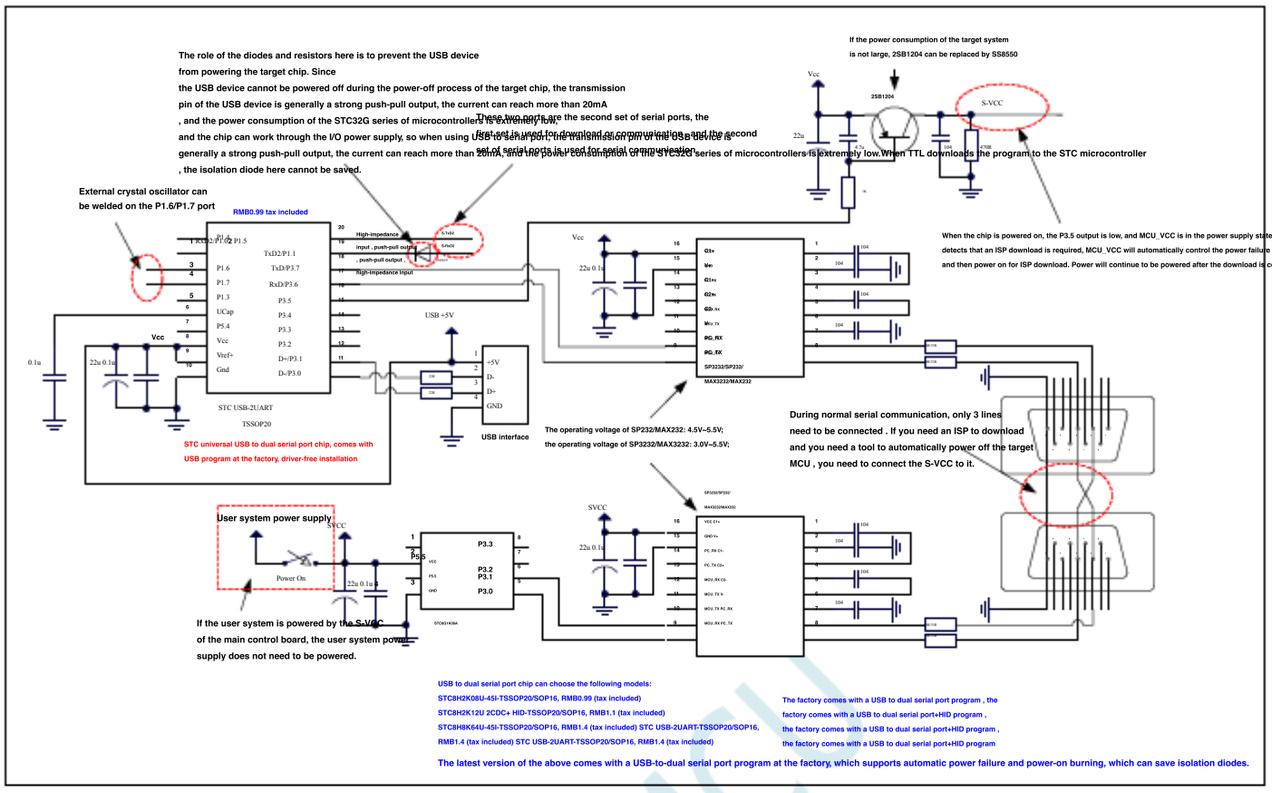
7.3. x

For microcontrollers with firmware version, microcontrollers with firmware version RS485

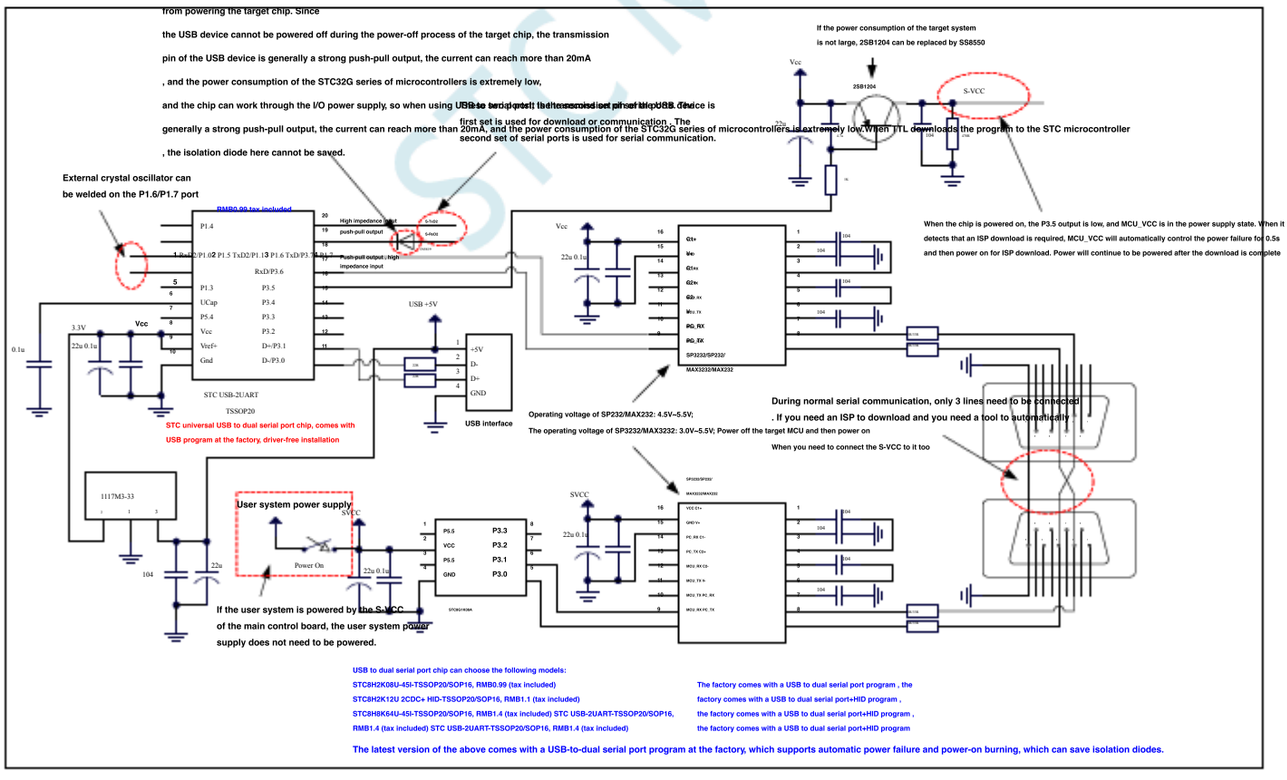
7.4. x

equal to or greater than the firmware version are required to be well supported. RS485

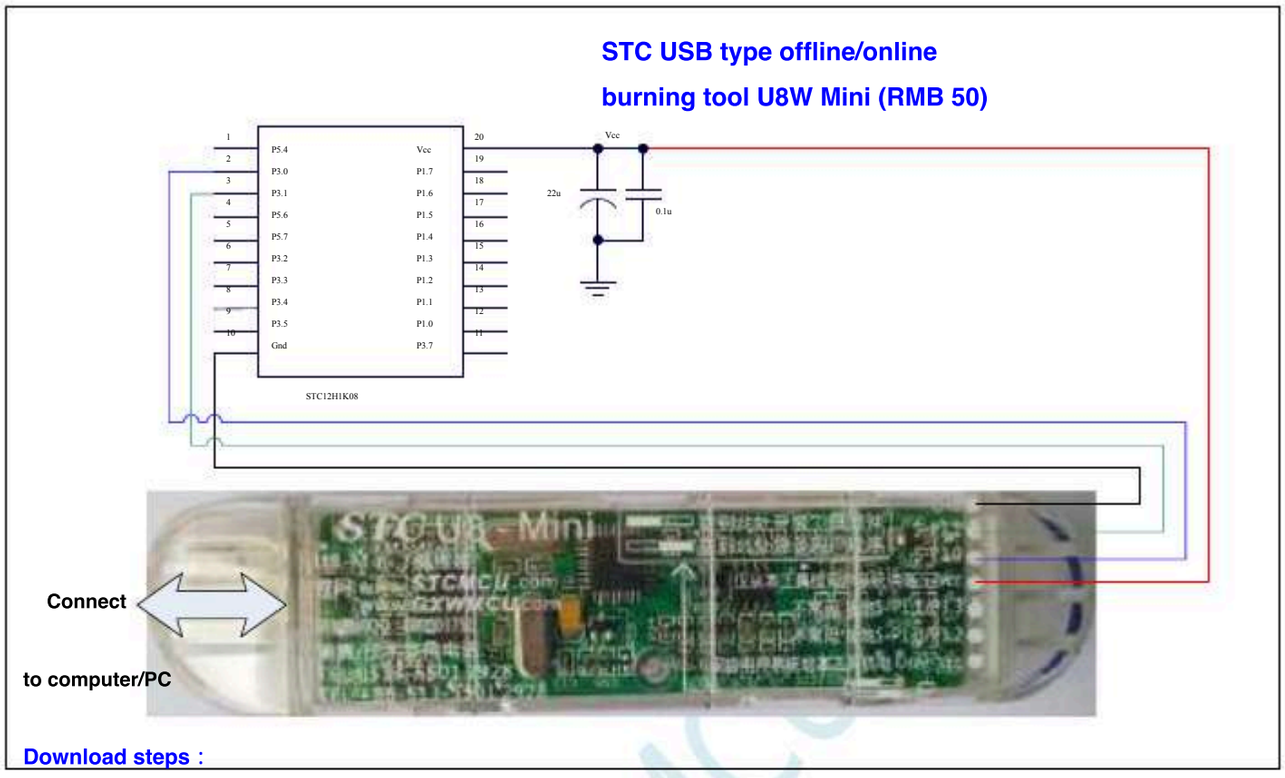
5.12.12 use USB To dual serial port/RS232 Download (5.0V)



5.12.13 use USB To dual serial port/RS232 Download (3.3V)



5.12.14 use U8-Mini Tool download, supporthold ISP Online and offline downloads, simulation can also be supported



Download steps：

ISP

、4 According to the connection method 8-Mini Connect to the target chip

shown in the figure, 5 click "Download" in the download software "Program" button

download 6, start ISP

Note: If you use U8-Mini To supply power to the target system, the total current of the target system will not exceed the specified value, otherwise it will cause the download to fail.

Note: It is currently found to be used USB power supply When downloading, due to the line is too thin, in The pressure drop on the line is too large,

to ISP The power supply is insufficient when downloading, please use USB power supply. When downloading, please make sure to use Strengthen the line.

To use U8-Mini For simulation, you must first 8-Mini Set to pass-through mode. U8W/U8W-Mini realize USB To serial port

The method of pass-through mode is as follows：

1, first U8W/U8W-Mini The firmware must be upgraded to Add above versions

of all U8W/U8W-Mini

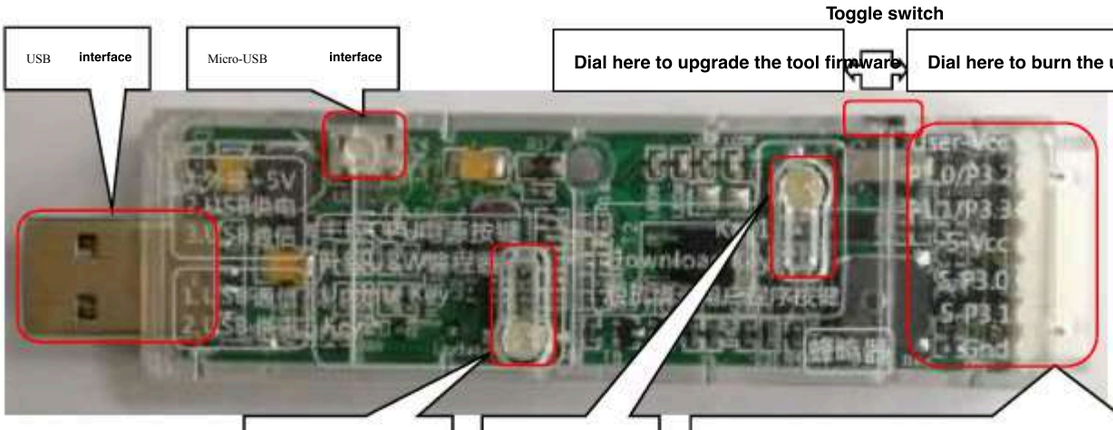
2、 After power-up, it is in normal download mode. At this time (power-on) Don't release the button, click again

Key2 the (power) button on the tool, then release the (power) button. Release again (Download) Button, U8W/U8W-Mini will

enter USB Go to serial port pass-through mode. (Press release, release Key2 Key1)

、 U8W/U8W-Mini The tool is just simple SB The serial port does not have offline download function, if you need

3 Enter pass-through mode U8W/U8W-Mini The original function, just click separately again (Power supply) Just press the button



updateU8W-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)**

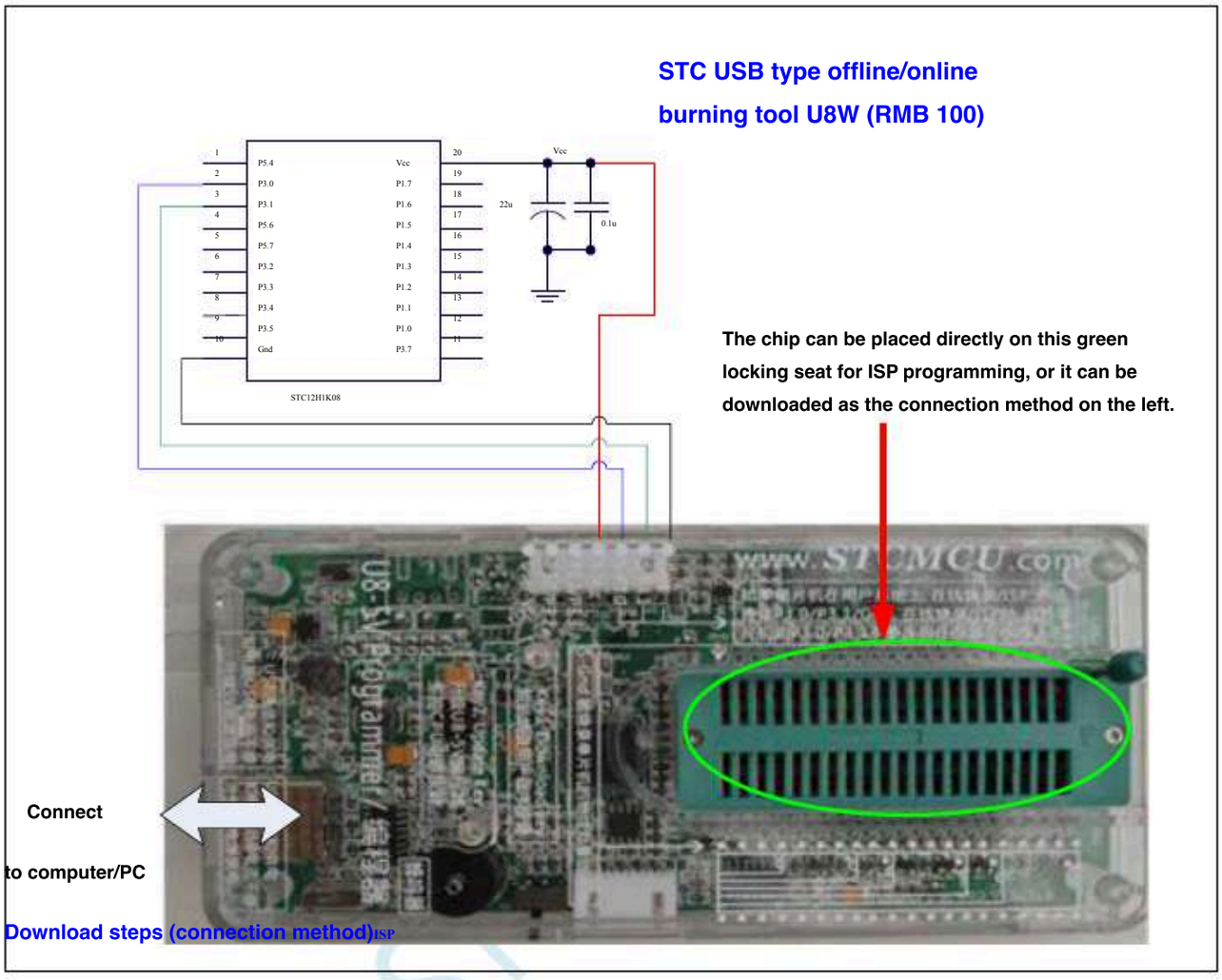
S-Vcc: **Only power the user's system from this**

S-P3.0: **tool and connect the slave to the machine** P.

S-P3.1: 30 **Connected to the slave** P. 31

Gnd: **Ground wire**

5.12.15 use U8W Tool download, support ISP Online and offline downloads, simulation can also be supported



Download steps (connection method) ISP

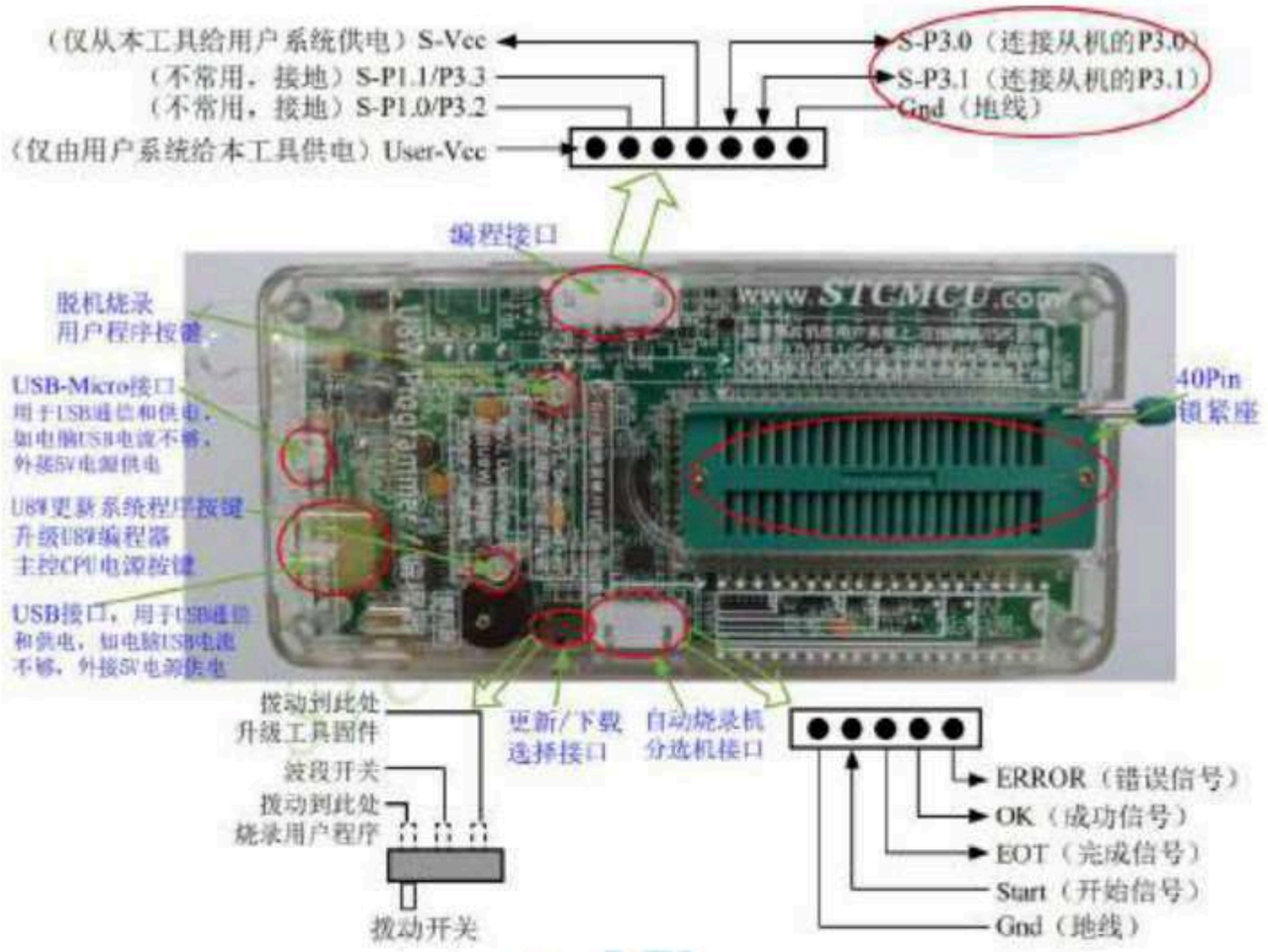
、1 According to the connection method shown in the figure, 2, click STC-ISP "Download" in the download software, "Program" button download 3, start ISP Connect to the target chip U8W

Note: If you use U8W To supply power to the target system, the total current of the target system should not exceed 200mA. Otherwise, it will cause the download to fail.

ISP Download steps (in-board mode)

、1 Place the chip in the direction where the feet are close to the locking wrench and the feet are flush downwards. Place the target chip in the seat, 2, click "Download" in the download software, "Program" button 3, start ISP download

Note: It has been found that USB Line power supply is insufficient when downloading, due to the line is too thin, the pressure drop on the line is too large, the use is caused by the power supply is insufficient when downloading, supply voltage is too low, it is recommended to strengthen the line.


（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
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
U8/更新系统程序按键
升级U8/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）

To use U8W For simulation, you must first U8W Set to pass-through mode. U8W/U8W-Mini realize USB To serial port pass-through mode

The formula

is as follows: U8W/U8W-Mini The firmware must be upgraded to Above versions

first of all U8W/U8W-Mini After power-up, it is in normal download mode. At this time (power-on) Don't release the button, click again

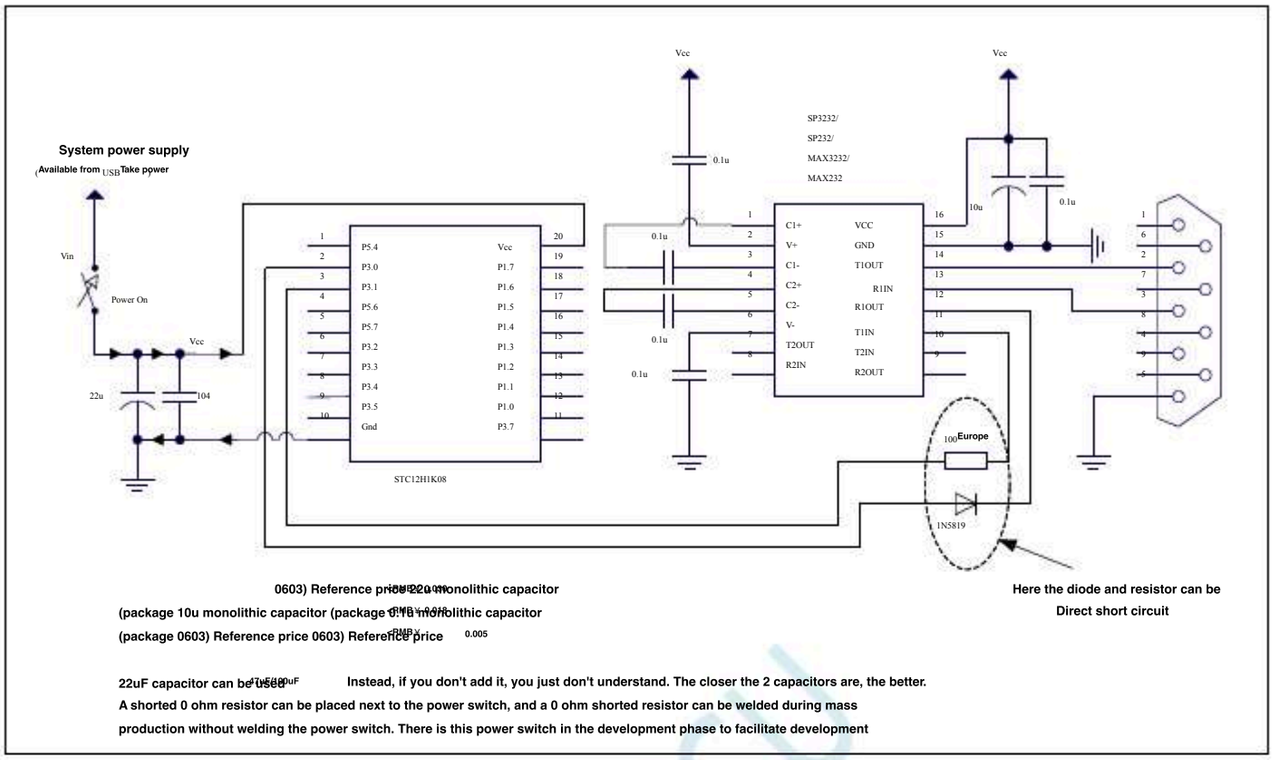
Key2 the (power) button on the tool, then release the (power) button. Release again (Download) Button, U8W/U8W-Mini will

enter USB Go to serial port pass-through mode. (Press Key2 release, release Key2 Key1)

、 U8W/U8W-Mini The tool is just simple USB The serial port does not have offline download function, if you need

3 Enter pass-through mode U8W/U8W-Mini The original function, just click separately again (Power supply) Just press the button Key2

5.12.16 use RS-232 Converter download, can also support simulation

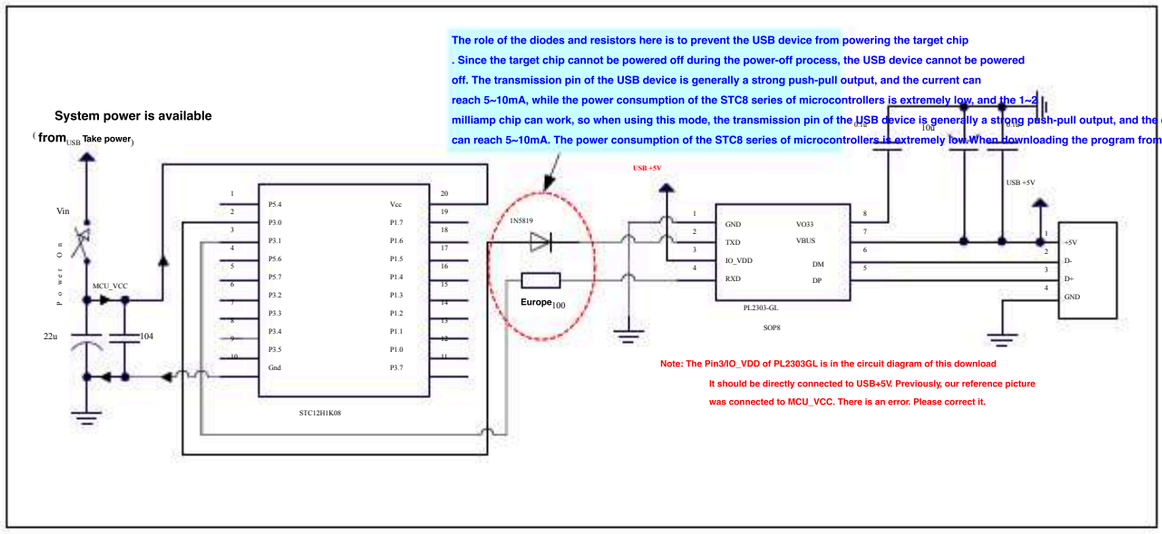


Download steps : ISP

、 1 Power failure to the target chip 2, click STC-ISP "Download" in the download software "Program" button

、 3, Power up the target chip 4, Start

5.12.17 use PL2303-GL Download, simulation can also be supported



ISP Download steps：

、6 Power failure to the target USB Power failure to serial port chip (such as： CH340、 etc.) PL2303-GL

chip, be careful not to give The transmission pin Due to of the serial chip to the serial port is generally a strong push-pull output, which must be on the target c

A diode is connected in series between the transmitting pins, otherwise the target chip cannot be completely powered off, and the target

STC-ISP 8, click Download" in the download software "Program" button

、

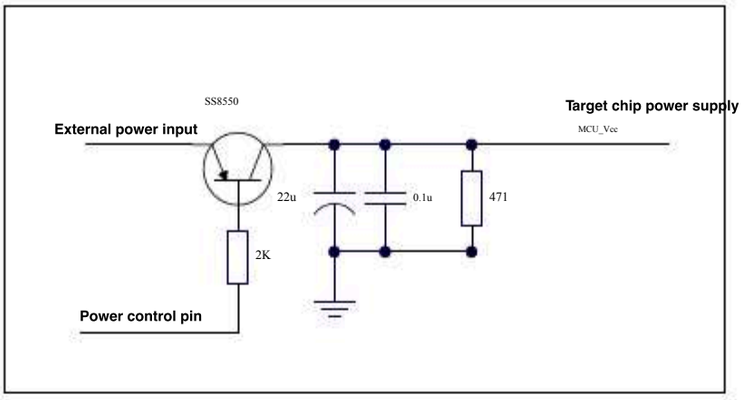
10 start ISP download

9 Power up the target chip

Note: It has been found that USB Line power supply When downloading, due to the line is too thin, in The pressure drop on the line is too large,

the use is caused by The power supply is insufficient when downloading, power supply. When downloading, be sure to use then the line.

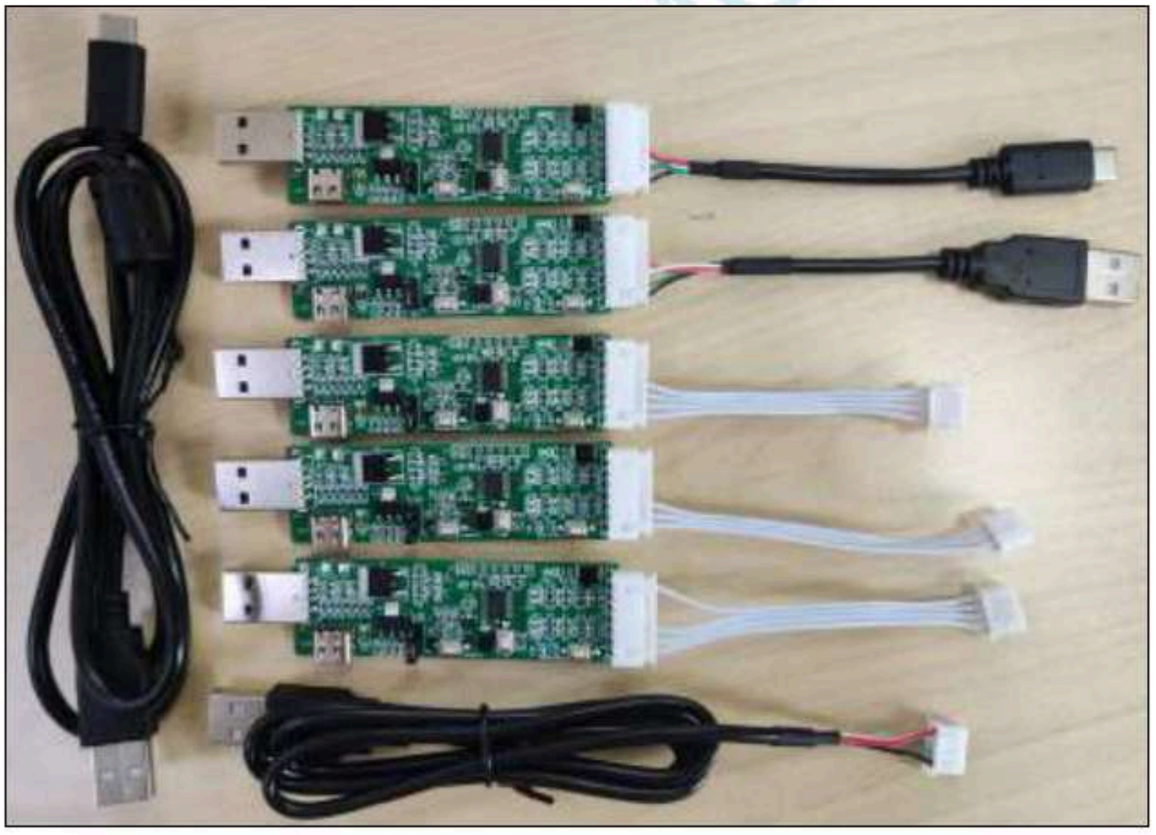
5.12.18 Single chip microcomputer power control reference circuit



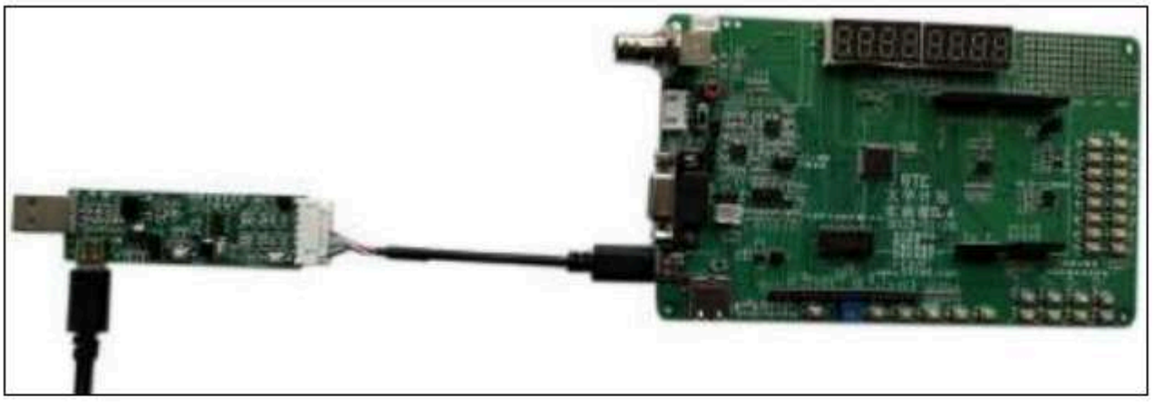
5.13 use STC Kill two birds with one stone USB Turn to dual serial port simulation series STC8 MCU

Let's briefly introduce the USB-to-dual-serial tool that kills two birds with one stone. 1. The appearance diagram of the USB-to-dual-serial tool that kills two birds with one stone with one stone. :

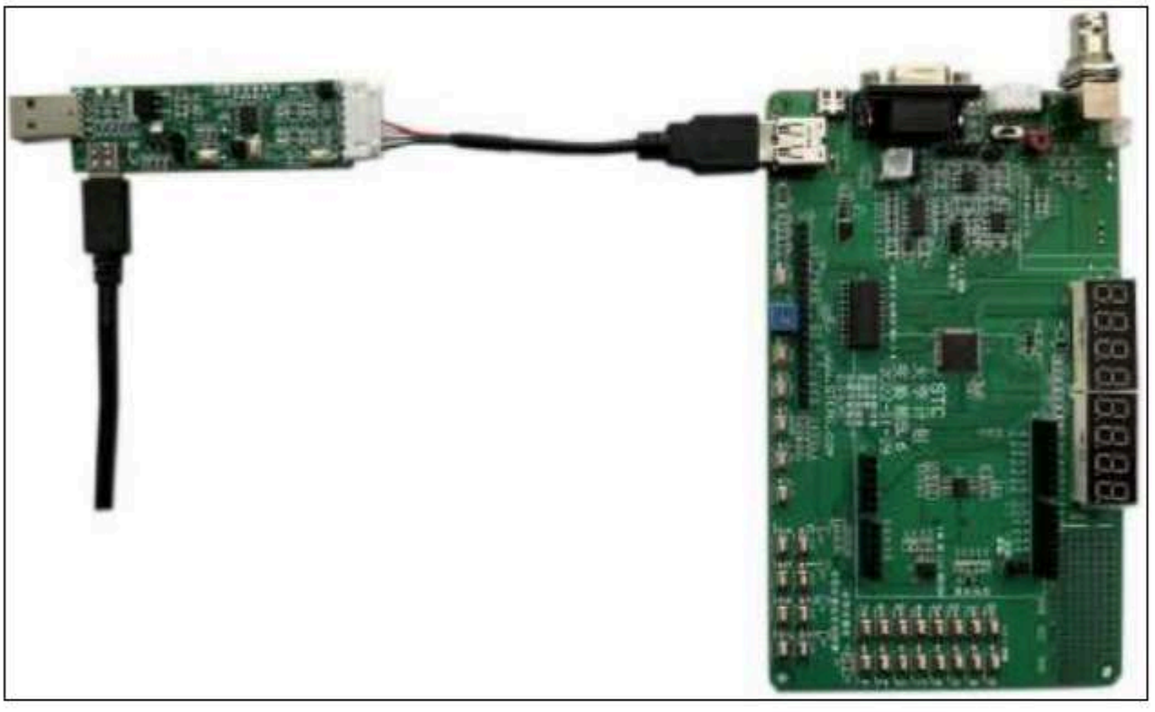
2. Several USB-to-dual serial port tools for killing two birds with one stoneCommonly used connection lines :



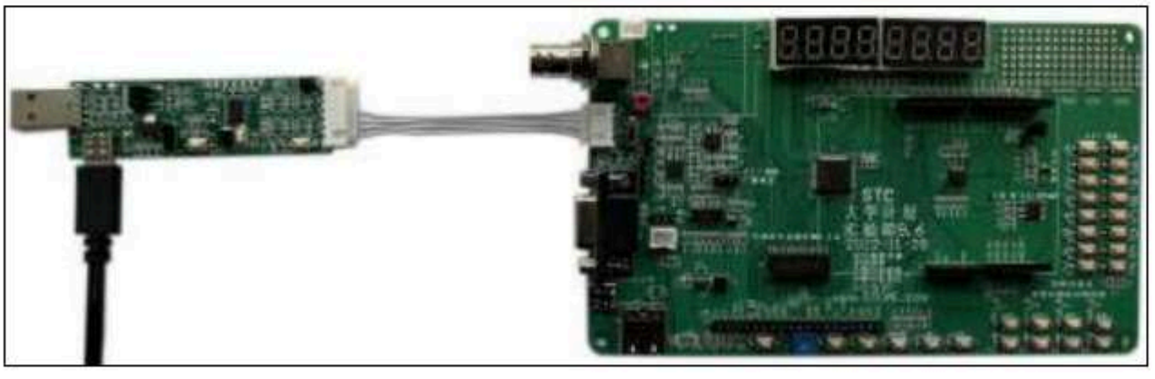
3. Kill two birds with one stone with SIP7-USB-TypeC to simulate/burn the STC8 series MCU. The hardware connection diagram is as follows：



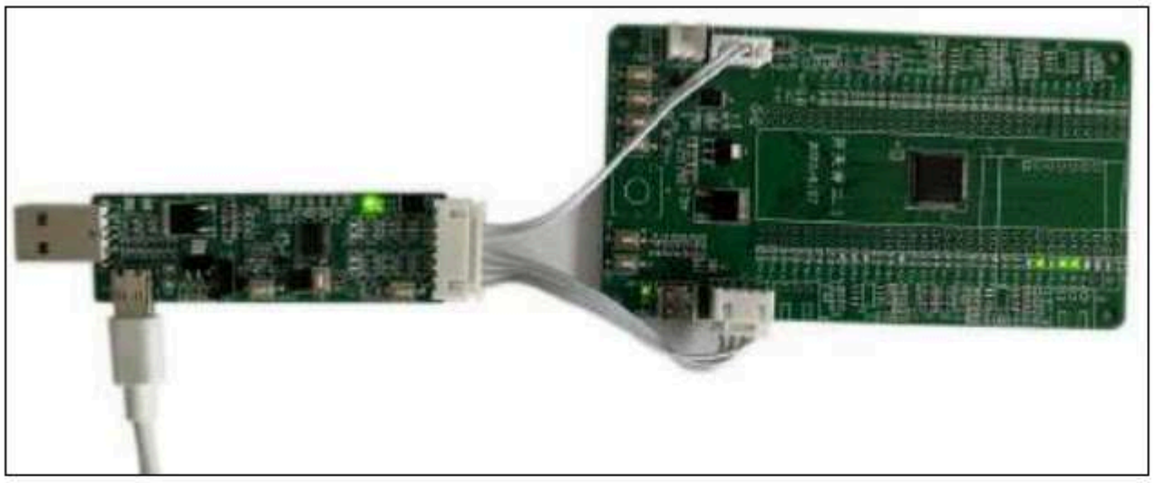
4. Kill two birds with one stone with SIP7-USB-TypeA to simulate/burn the STC8 series MCU. The hardware connection diagram is as follows：



5. Kill two birds with one stone with SIP7-SIP4/2.54mm ordinary socket to simulate/burn the STC8 series MCU. The hardware connection diagram is as follows：

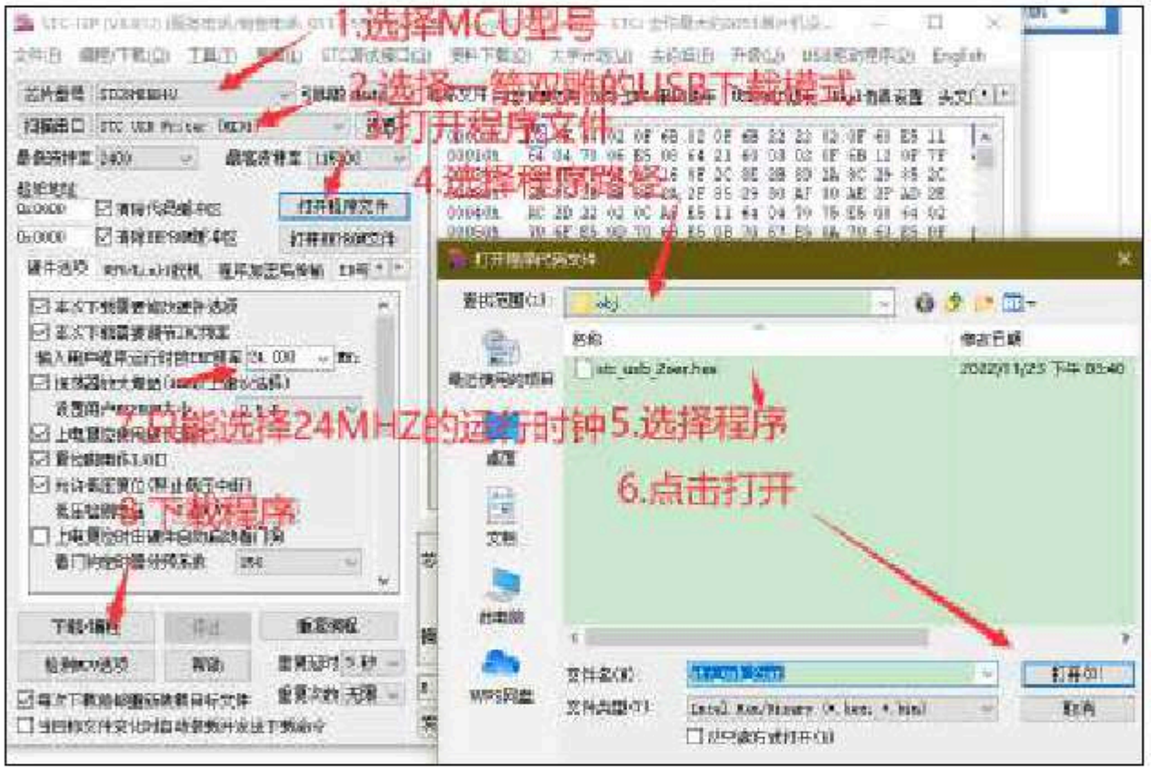


6. Kill two birds with one stone, the USB extended USB-CDC serial port 1 emulates; the extended USB-CDC serial port 2 communicates with other serial ports, and the hardware connection diagram is as follows:



To set up killing two birds with one stone as an ordinary download tool, you can refer to this post on the forum of this official website :
, 7https://www.stcaimcu.com/forum.php?mod=viewthread&tid=240&highlight=%E4%B8%80%E7%AE%AD%E

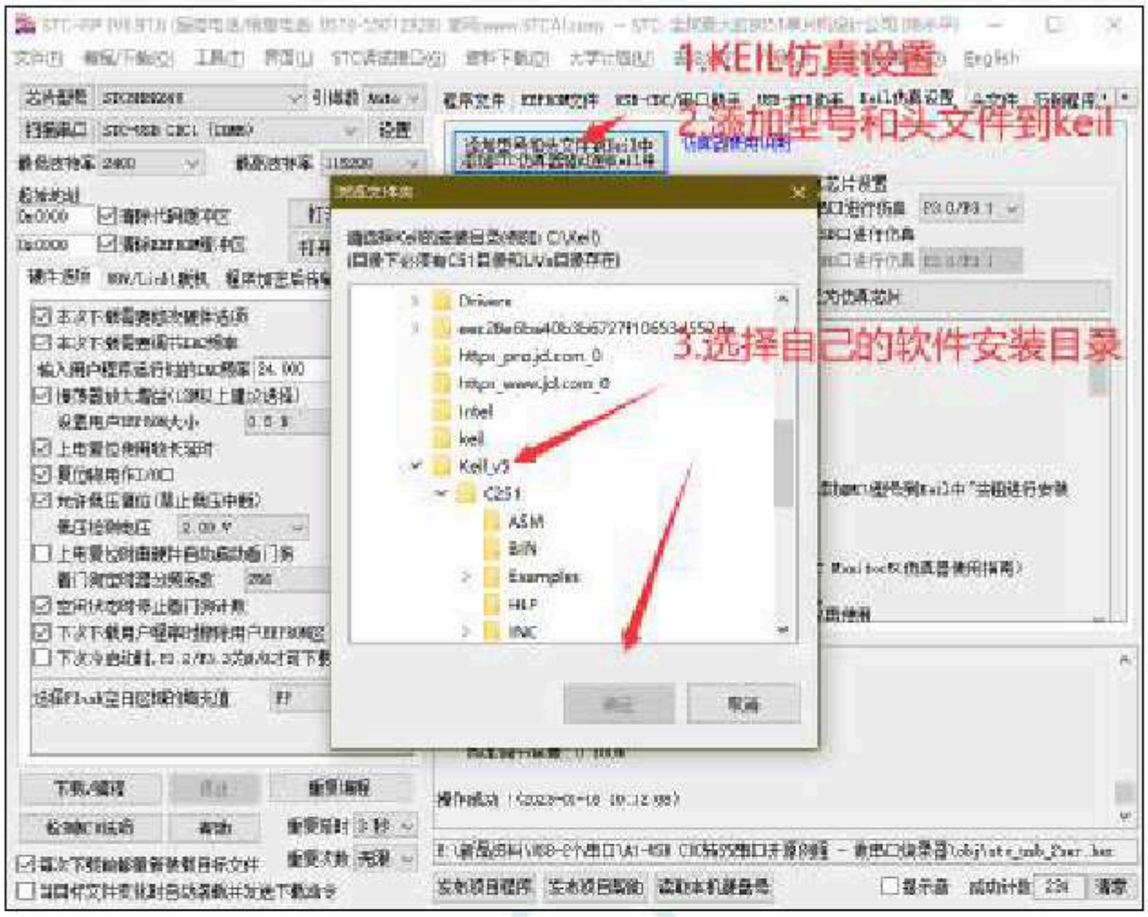
5%8F%8C%E9%9B%95

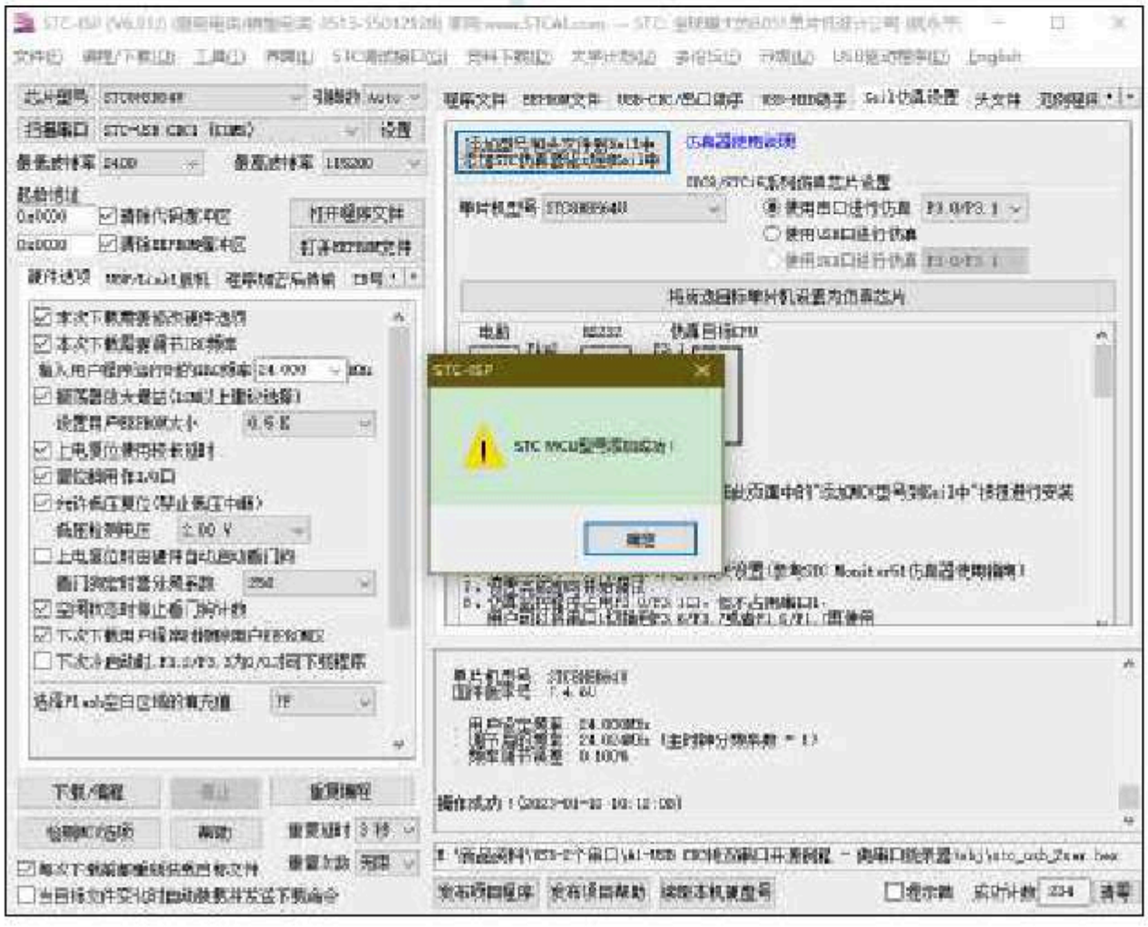


The software settings are as follows: 8. First go to the official website to download the latest STC-ISP software. As of now, the latest version is STC-ISP (6.91J), especially for simulation. The stcmon51 simulation driver version of STC-ISP (6.91J) has been updated to v1.18. After internal repetition , the stcmon51 simulation driver version of STC-ISP (6.91J) has been updated to v1.18. The test has been very stable. (Download address: Tool Software-Shenzhen Guoxin Artificial Intelligence Co., Ltd)

9. Update the STC resources in KEIL: add the firmware and chip model of the STC emulator to KEIL.

(This step is recommended to be done every time you download the software update to ensure that the simulation drivers are all up-to-date)





10. Now start the port and For the simulation steps, first STC8H8K64U Set to emulation chip ; STC8H8K64U Currently only supports strings direct simulation. USB put KEIL (P3.0/3.1 is selected here) As an emulation port, so there can be no occupation in the program. P3.0 and P3.1 quote

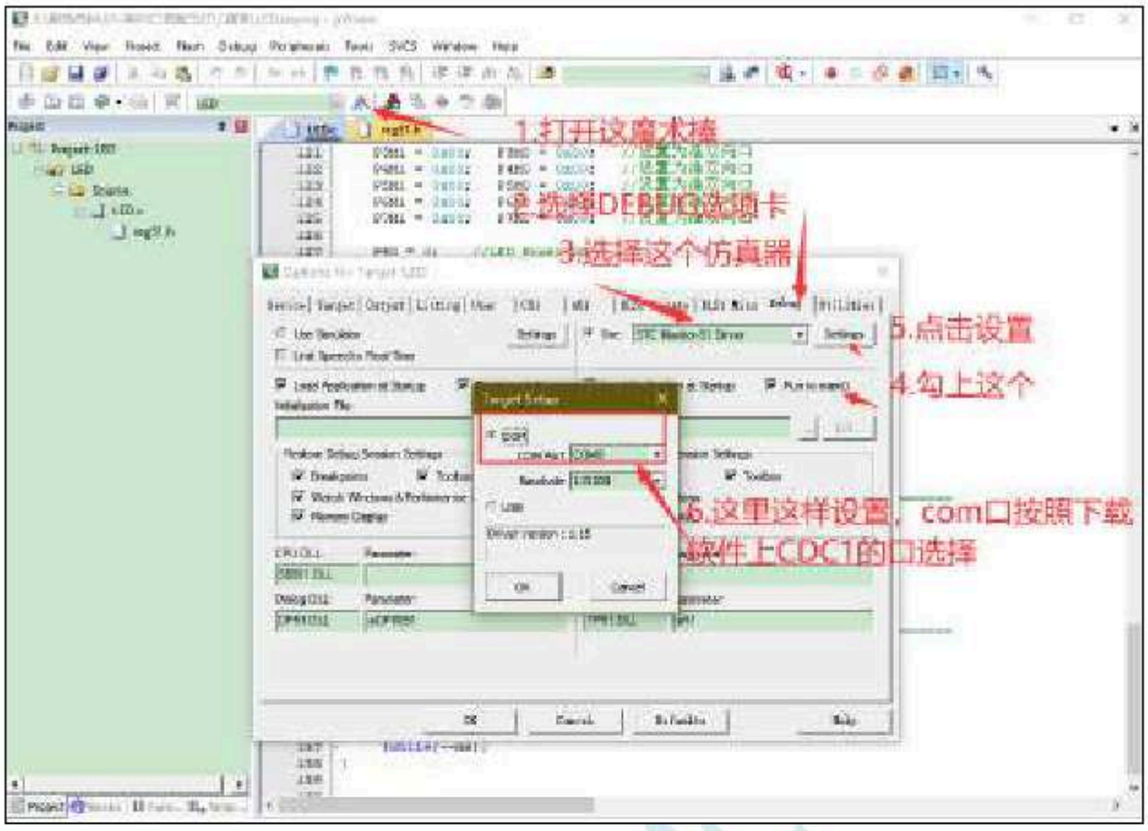
The function of the foot, the simulation precautions will also be explained in the program. [2] The program is tested, it is easier to observe the results!

Light The chip, and then perform the following settings to set the Kaitian Axe as the simulated main control chip.

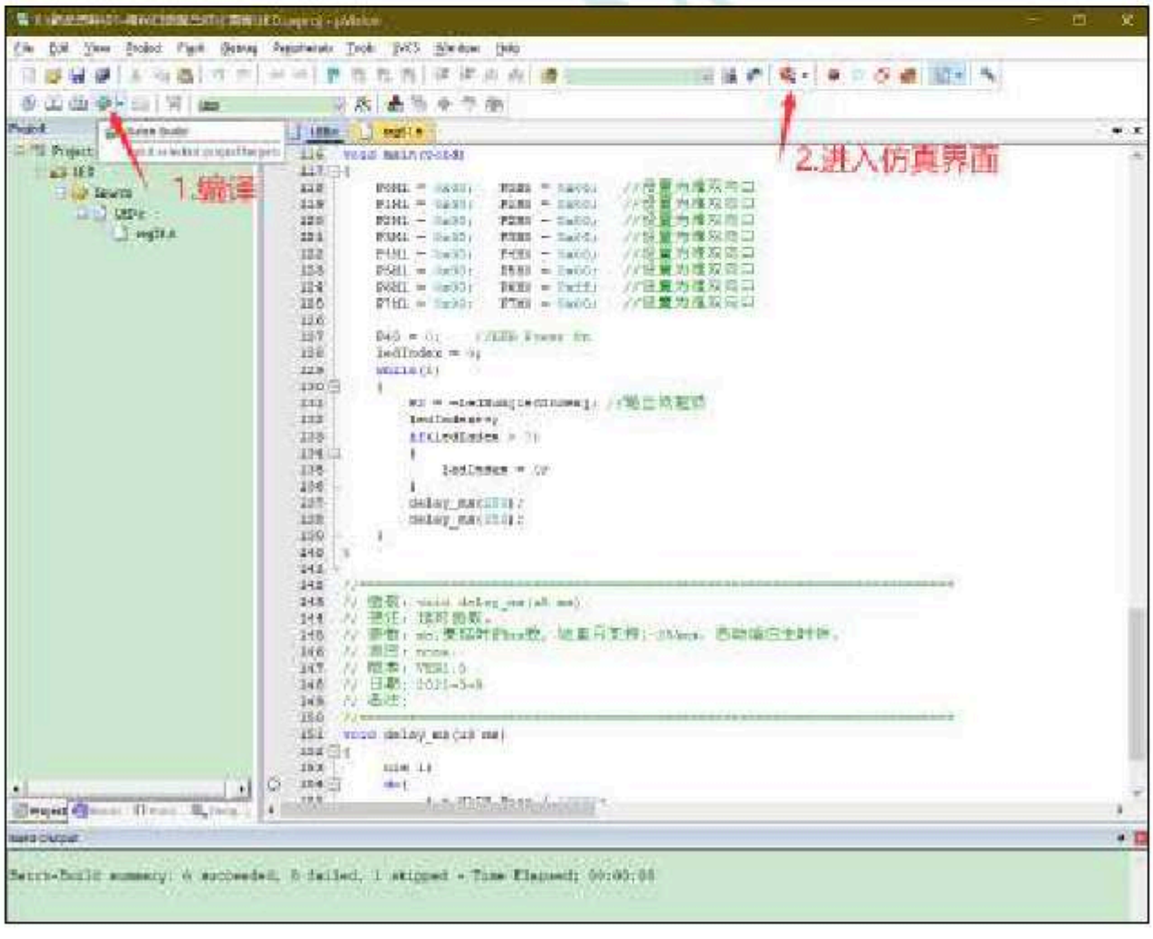
up one first At this time connect the IRC The frequency must be the same as the master clock set in the program!

Pay attention to what's here

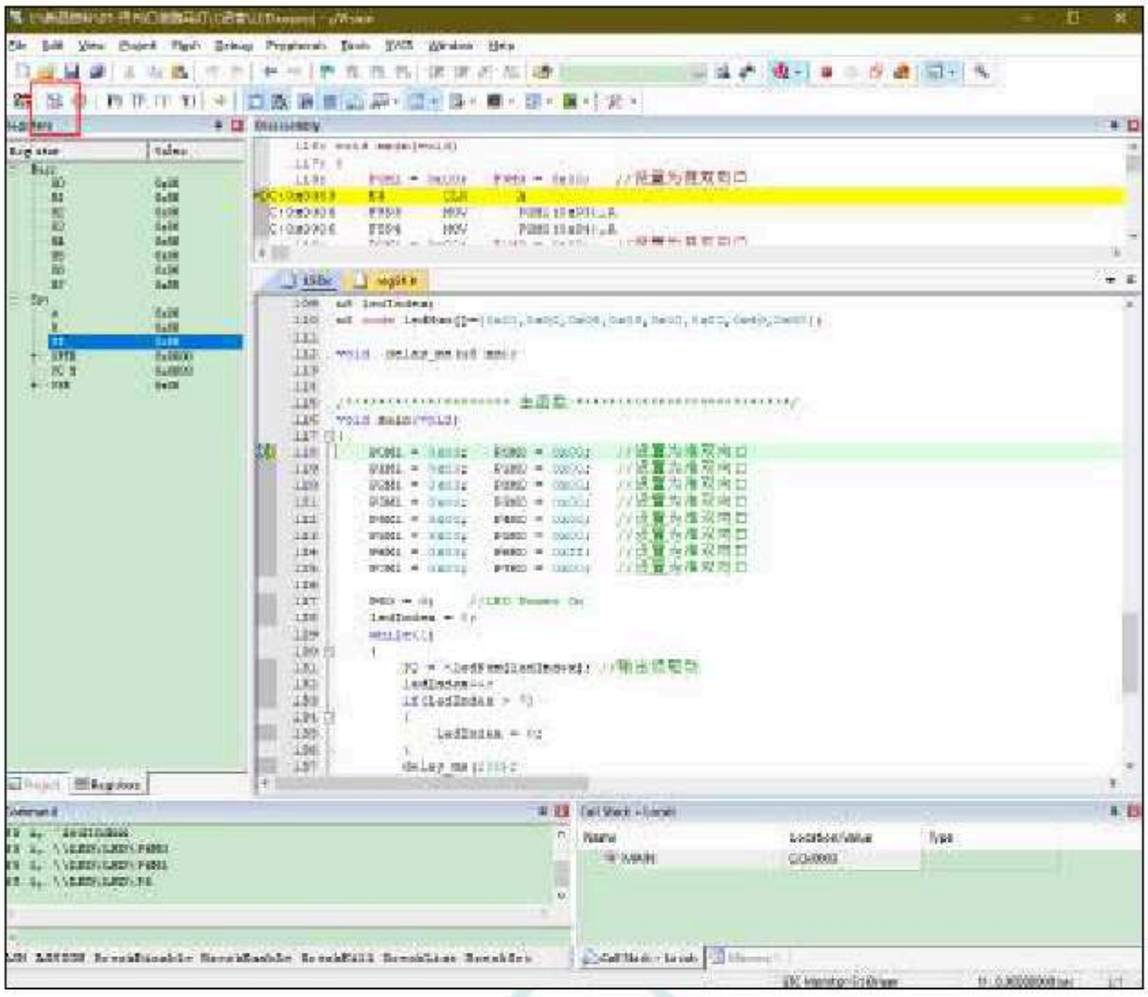
11 , Kaitian Axe is set to the simulation chip, open For projects under construction, go directly to the path of the simulation in.



12 , So that it can be compiled and debugged.



13 , The following interface appears, indicating that the simulation mode has been successfully entered, and then you can use the functions o



5.14 STC-ISP Download software advanced application

5.14.1 release project program

The function of publishing the project program is mainly to package the user's program code and related option settings into a program that can directly download and program the target chip. Super simple executable file of the user's own interface。

Regarding the interface, users can customize it by themselves (users can modify the title, button name, and help information of the publishing project program by themselves) , and at the same time, users can also specify the hard disk number of the target computer and the system of the target chip . The publishing application can only run on the specified computer.

Number, after specifying the hard disk number of the target compu

(Prevent the burner from easily stealing the program from the computer If sent via the interne

Pan bake away Can't prevent it Of course there is no way to steal your computer The offline download tool is better than electricity Brain burning saf

Number of burning chips Let the front desk clerk burn You can let the boss burn it)

Copied to another computer, the application cannot run. The same ,

When the number of the target chip is specified, the user code can only be downloaded to the corresponding chip (Only the target chip of the number (For a device to

Products that sell tens of millions are particularly useful The tank can be sent to the customer to upgrade by themselves No need to risk your life to go to war-torn Ira

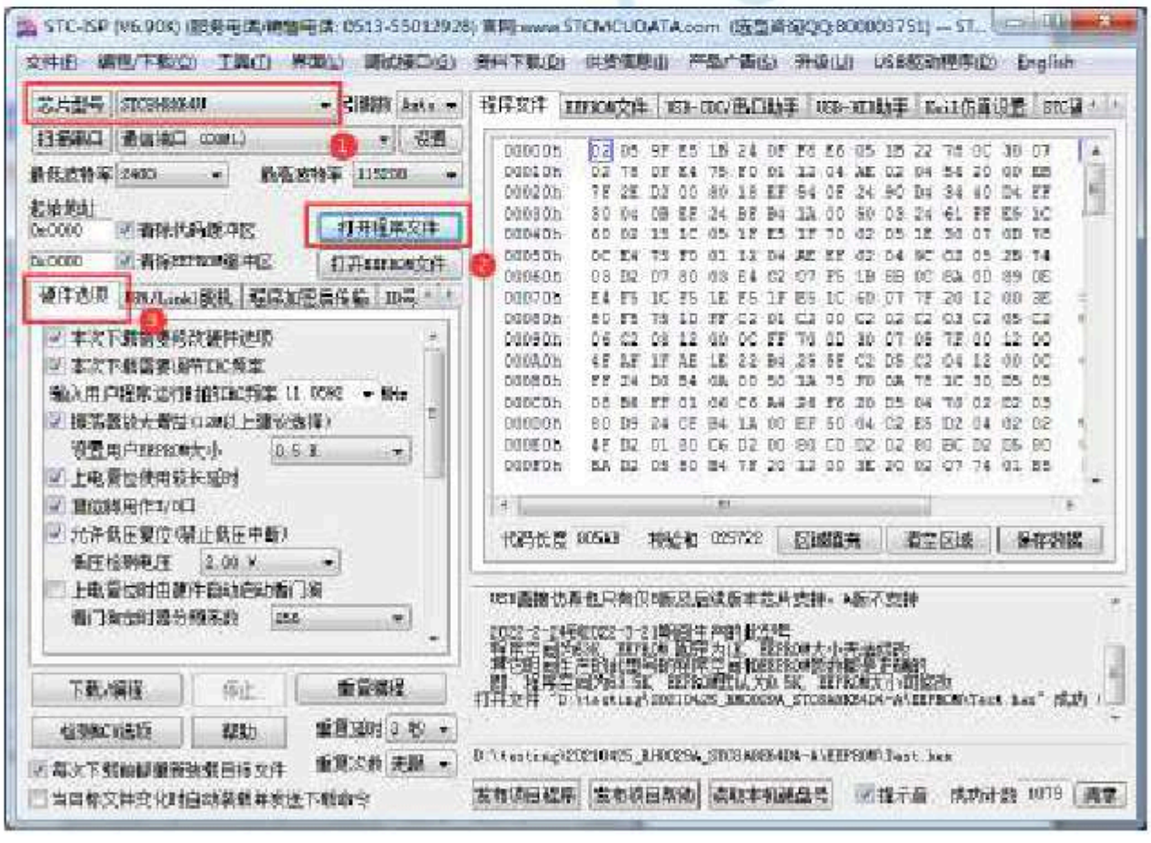
for Software) Other chips with inconsistent numbers cannot be downloaded and programmed. ID

The detailed steps for publishing the project program are as follows：

1 , First select the model of the target

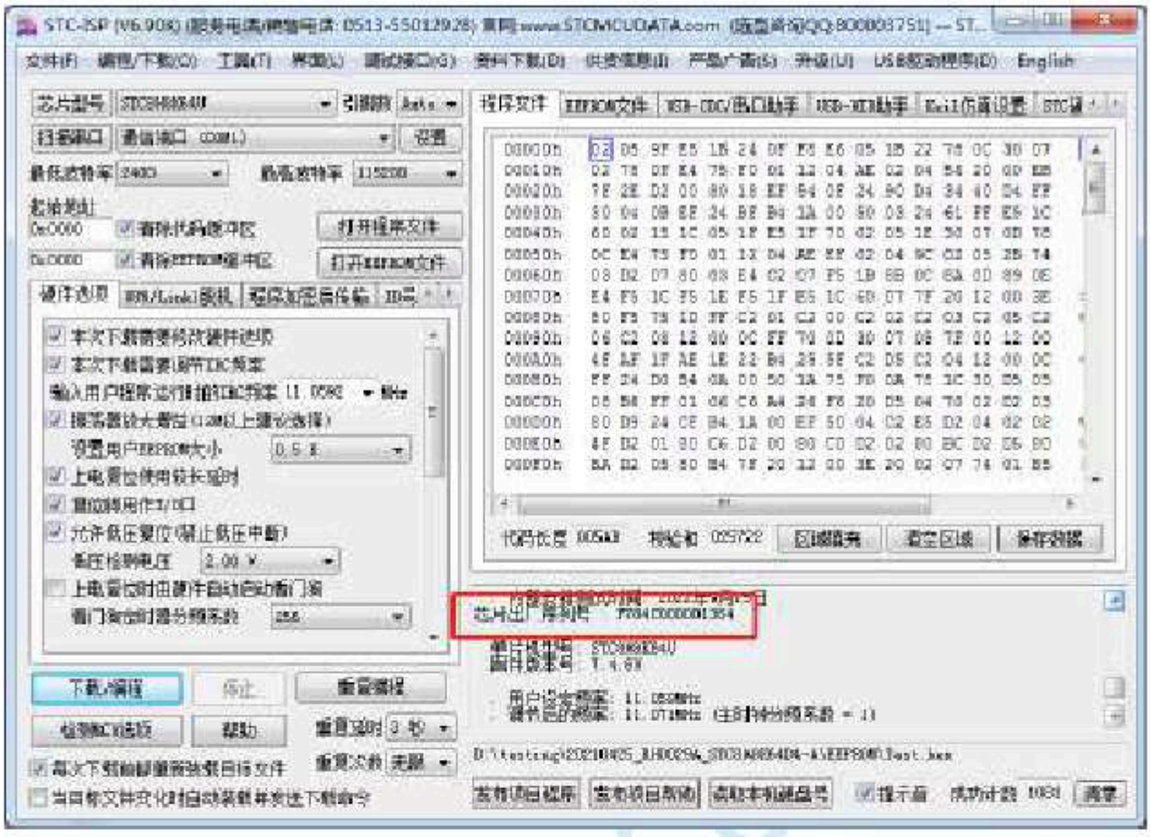
2 chip , open the program code file , and set

3 the corresponding hardware options

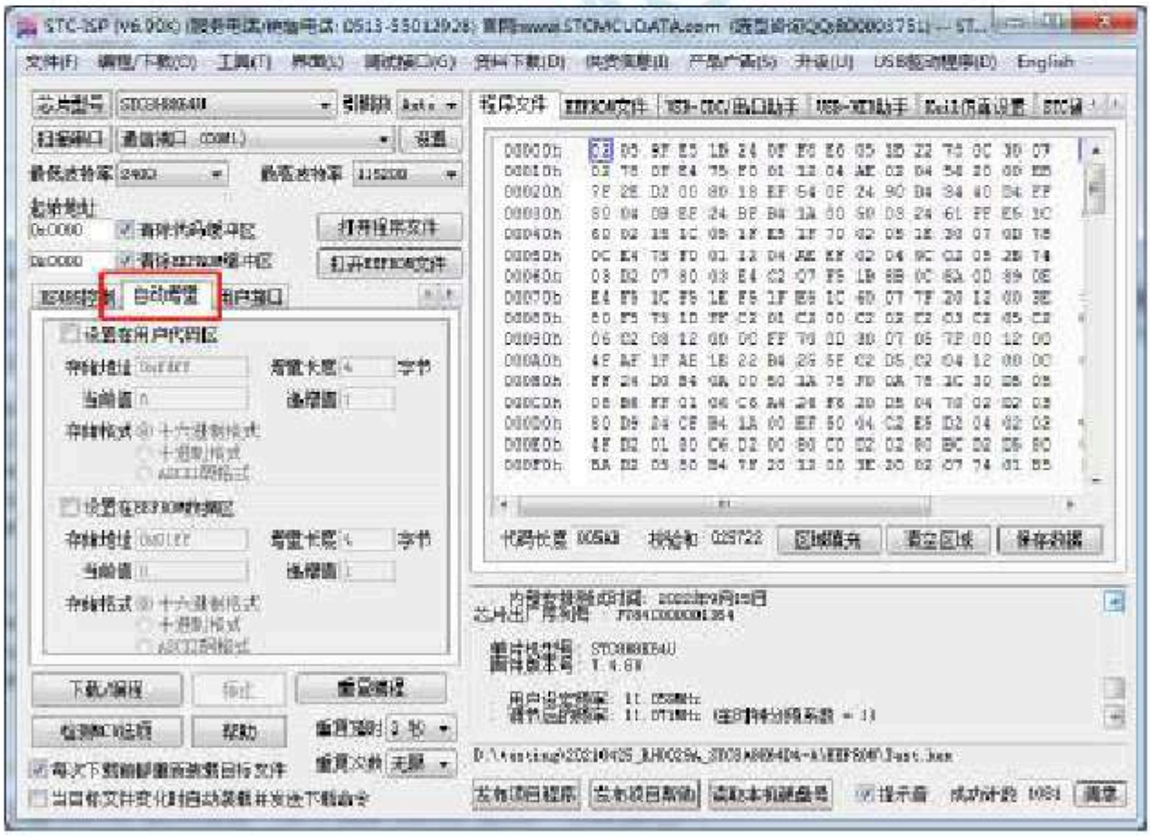


4 , Try to burn the chip, and write down the target chip number, as shown in the figure below, The chip ID number is "

(If the target chip is not required Verify the number, you can skip this step)



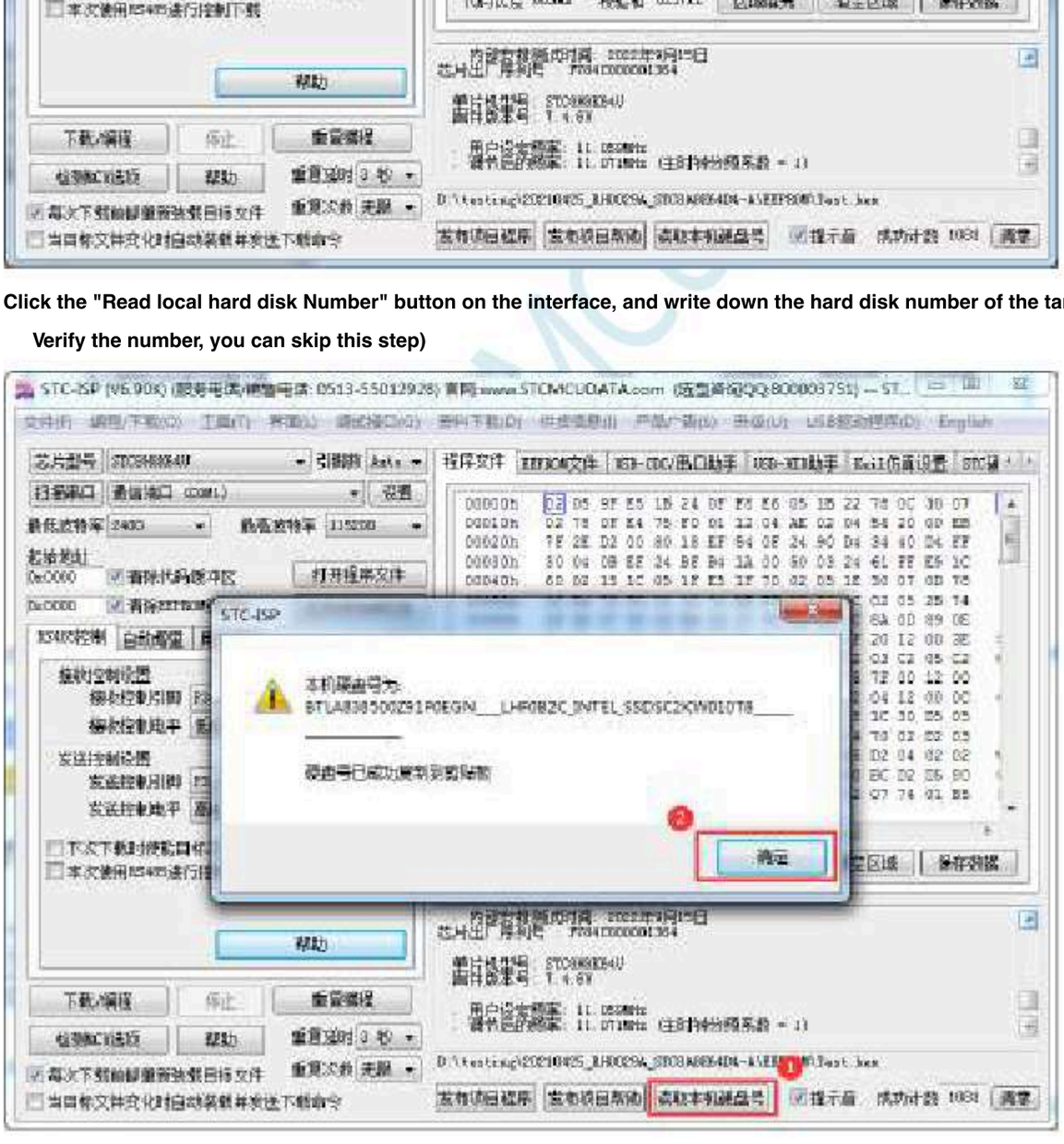
5 , Set the automatic increment (if you do not need the automatic increment, you can skip this step)



6, Set up RS485 Control information (if not required RS485 Control, you can skip this step)

7 , Click the "Read local hard disk Number" button on the interface, and write down the hard disk number of the target computer (if the tar

Verify the number, you can skip this step)



8 , Click the "Publish Project program" button to enter the settings interface of the publish application.

9 , According to their respective needs, modify the title of the published software, the name of the download button, the name of the repea

and the name of the automatic increment

Name and help information

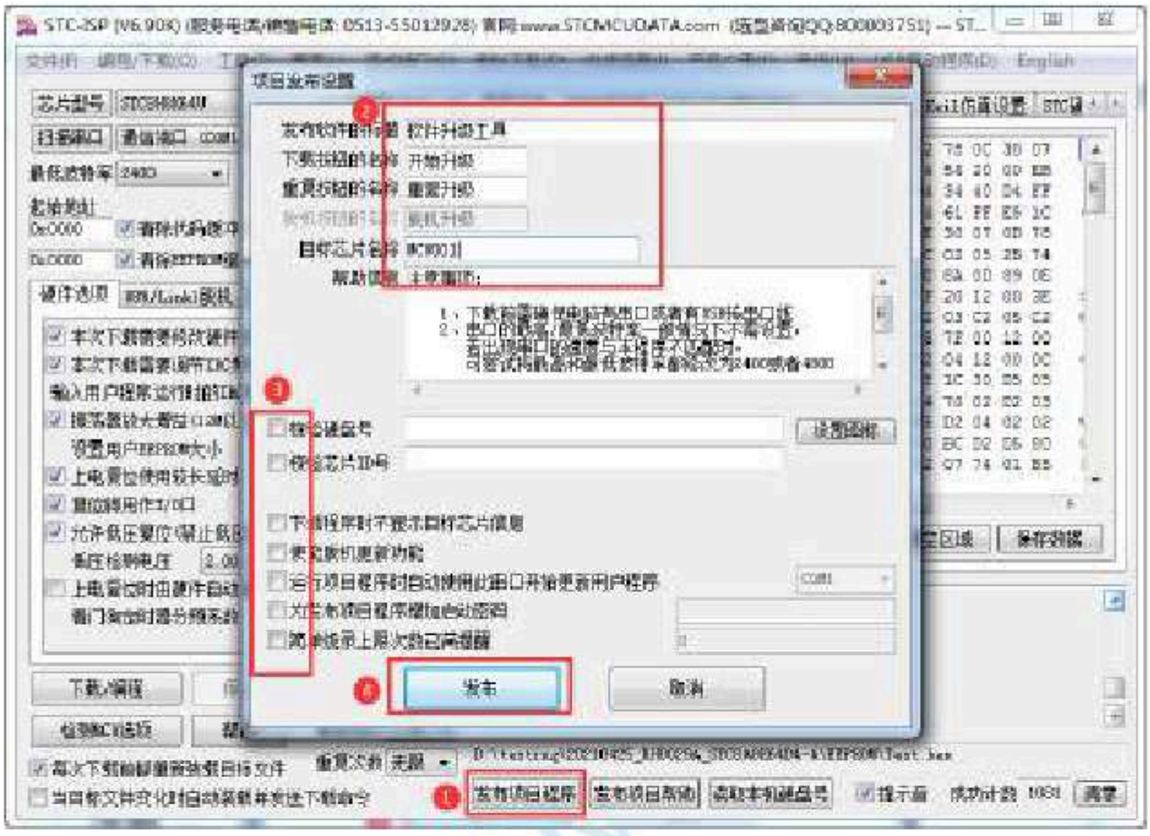
, if you need to verify the hard disk number of the target computer You need to check "Verify hard disk number" and enter the previous nc

at the back. 10

The hard disk number of the target computer

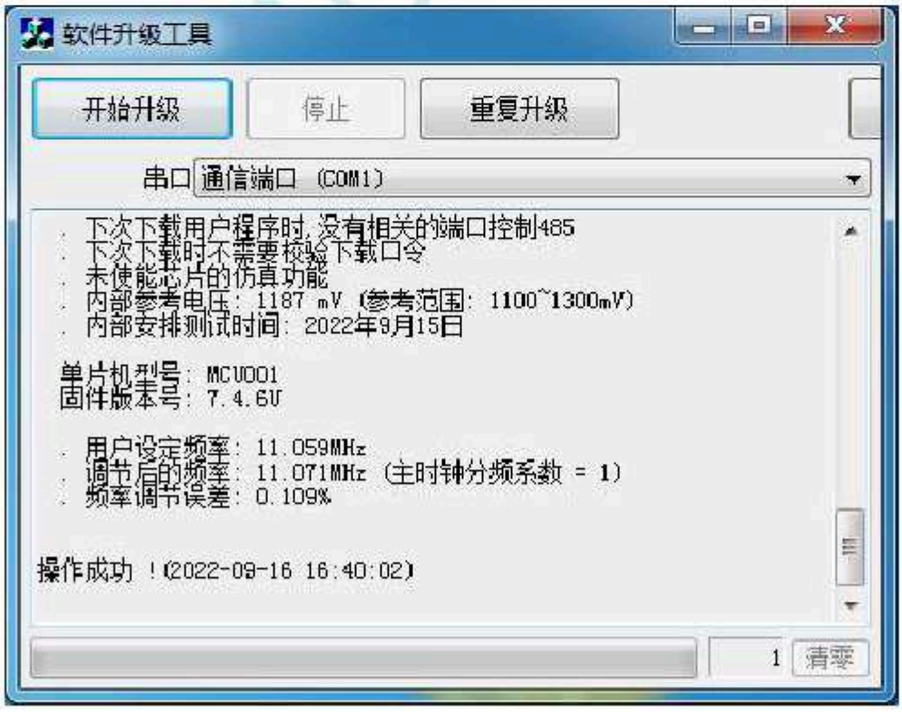
11 , if you need to verify the target chip ID number, you need to check ID Number", and enter the front in the text box at the back

The recorded target chip ID "Verify chip number"



12 , Finally click the publish button, save the project publishing program, and you can get the corresponding executable file. The publish

The picture below is as follows



5.14.2 The program is encrypted and transmitted (the serial port analyzes the program during anti-burning

Currently, all ordinary serial port downloads are burnedRecording programming is usedClear code communication(When the computer communicates with the target chip), problem: If the programmer downloads the data of the serial port communication during programming ar draw a line on the serial port during programming, and analyze the actual data by analyzing the serial port communication.User program code It's better to download a board burning program than to burn a program with a computer

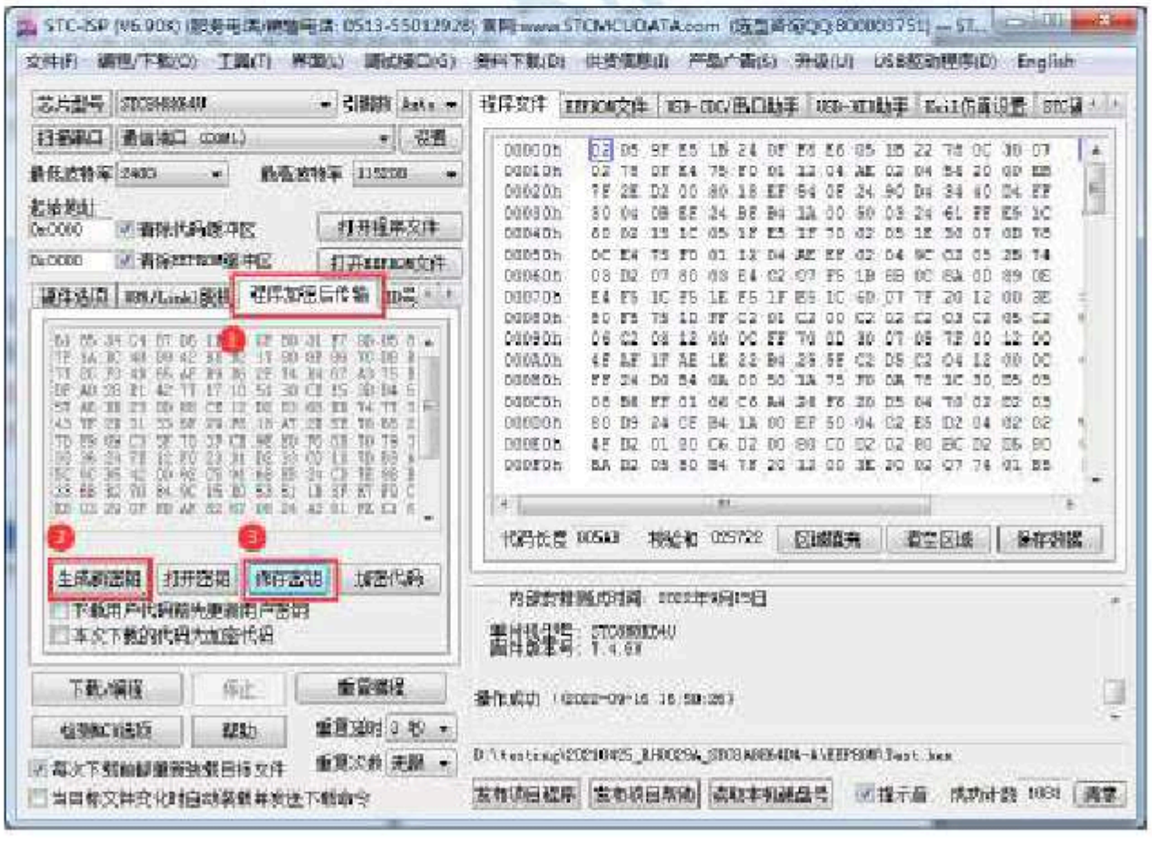
（Prevent the burner from easily stealing the program from the computer, such as sending it over the Internet, such as through The offline download tool is safer than computer burning, so tha It's hard to prevent it from being baked away. Of course, if you steal your computer, there's no way to do that, so STC Even if it is Miss desk clerk can burn it, let the boss burn it.) The world's first offline download tool, which is important to prevent the illegal distribution o In the process of offline download tool burning, the actual user program code is analyzed by analyzing the data of serial communication, but The program encryption and transmission function provided by the single-chip microcomputer to meet the requirements, which requires the use of the latest STC

After the program is encrypted, the transmission and download is that the user first encrypts the program code through his own set of through the serial port. At this time, the encrypted file is downloaded and transmitted, and the encrypted garbled code is analyzed through th into your company, you will not be able to download the encrypted file.Stealing the encryption key in your computer has no value, and it can s code by monitoring the serial port when burning the program .

The use of the transmission function after the program is encrypted requires the following steps：

1, Generate and save a new key

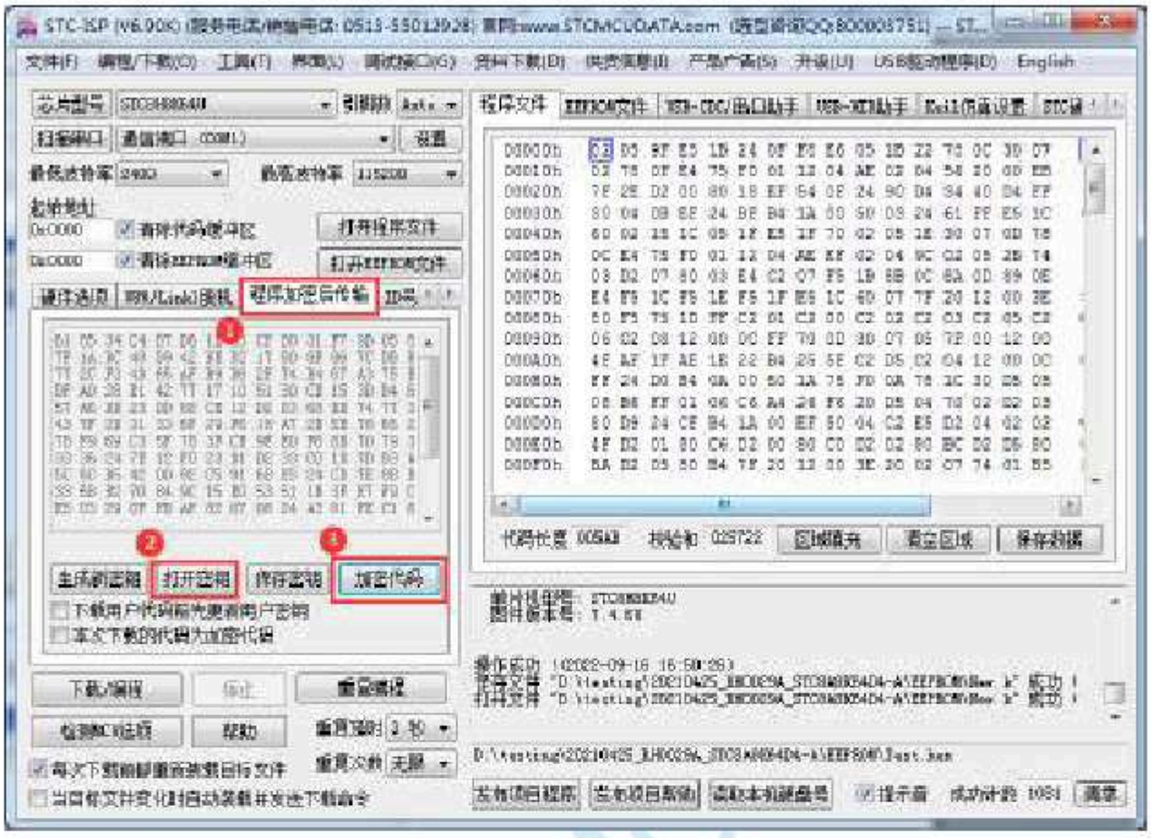
As shown in the figure below, go to the "Transmission after program encryption" page and click the "Generate New key" button to display the newly generated key in the buffer. By the key, then click the "Save key" button to save the generated new key as "K"Is the extension The key file (Note: This key file must be saved well. All code files released in the future need to be encrypted with this key, and the generation of this key is non-repetitive, that is, it is impossible to generate two identical keys at any time, once the key file is lost, it cannot be retrieved.) For example, we save the key as "



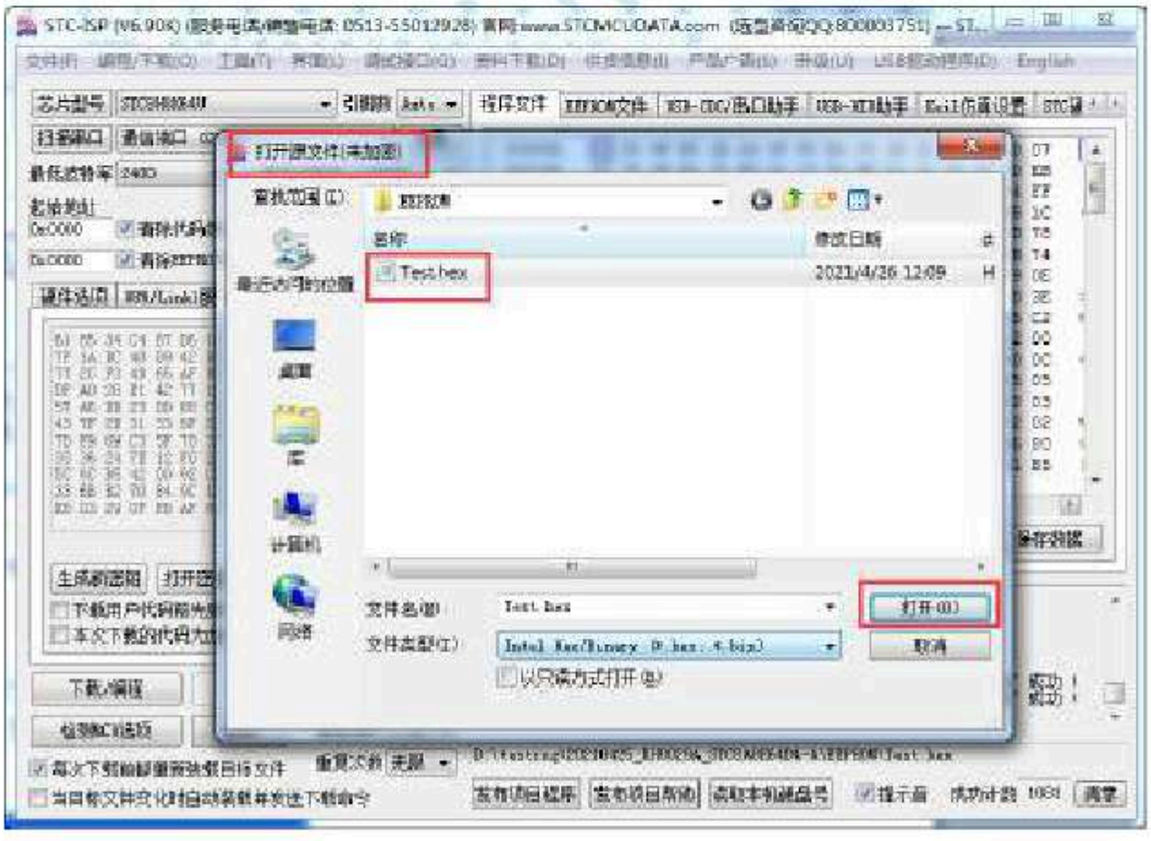
2, Encrypt the code file

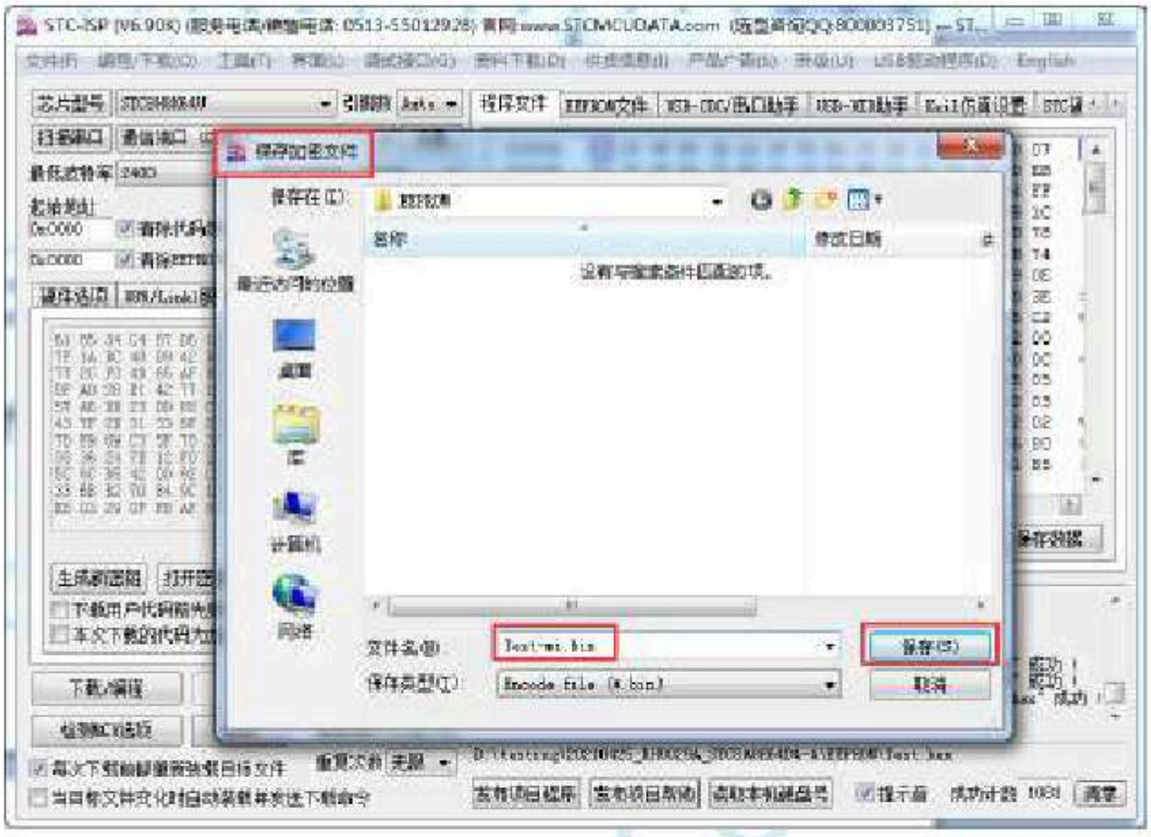
Before encrypting the file, we need to open our own key first. If our key is already stored in the buffer, don't open it again. As shown in the figure below, click the "Open Key" button on the "Transfer after Program Encryption" page to open the key file we saved "New. k", then return to the "Program encrypted transmission" page and click the "Encryption code" button, as shown in the

"Open source file (unencrypted)" will pop up a dialog box, at this time the original unencrypted code file is selected



After clicking the open button, a similar dialog box will pop up immediately, but at this time it is a conversation to save the encrypted file.box. As shown in the figure below, click the save button to save the encrypted file.





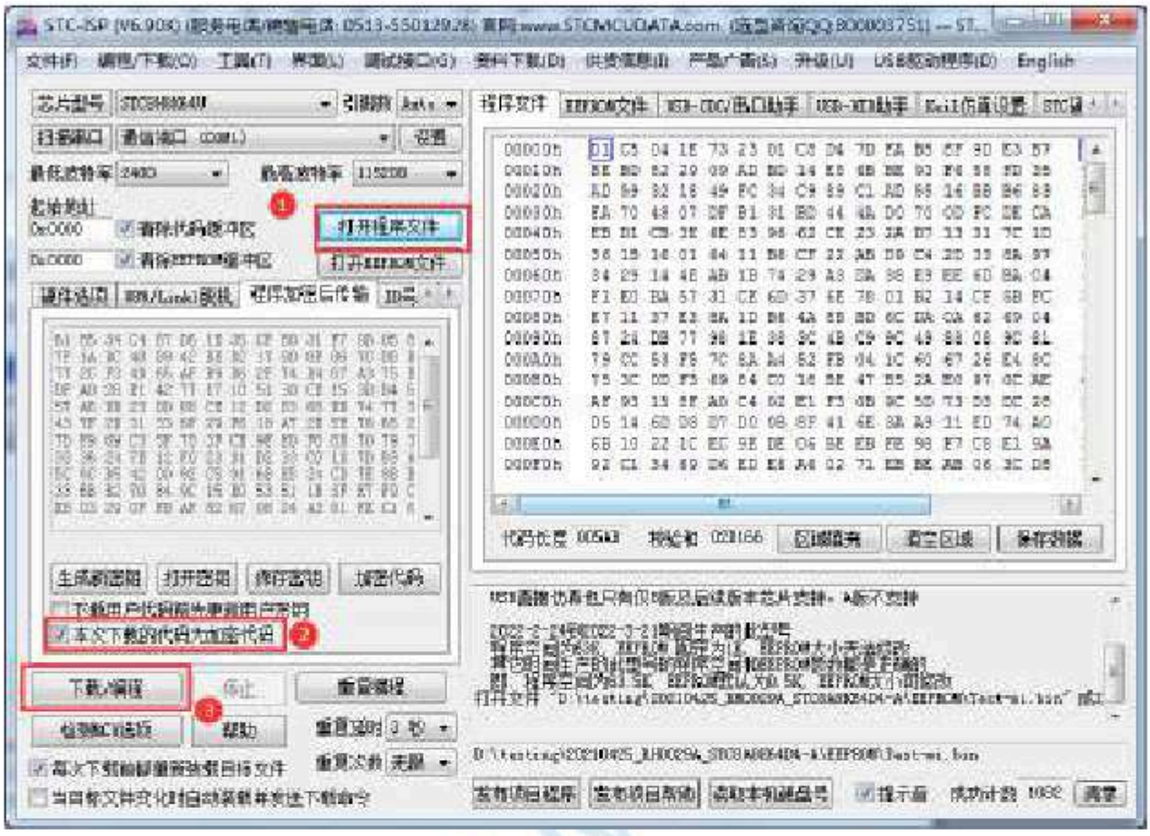
3 , Update the user key to the target chip

Before updating the key, we need to open our own key first. If our key is already stored in the buffer, don't open it again. As shown in the figure below, click the "Open Key" button on the "Custom Encryption Download" page to open the key file we saved earlier, for example, the key is opened as shown in the figure below, check the option "Update the user key before downloading the user code" and "This time is the encryption code", then open our previously encrypted file, and click "Download" in the lower left corner of the interface after opening. The "Program" button updates the user key after downloading the target chip in the normal way.

New. If The options of "This code downloaded

4, Encrypt and update user code

After the key is successfully updated, the target chip has the function of receiving the encryption code and restoring it. If you need to update the code, you only need to refer to the method of the second step to encrypt the object code, and then as shown in the figure



For a new piece STC Single chip microcomputer, the steps are: 1) The encryption key is completed, and the key is about to be updated to the target chip (That is, the key has been updated to the target chip

You can download the encrypted code to the MICROCONTROLLER together (That is, the key has been updated to the target chip

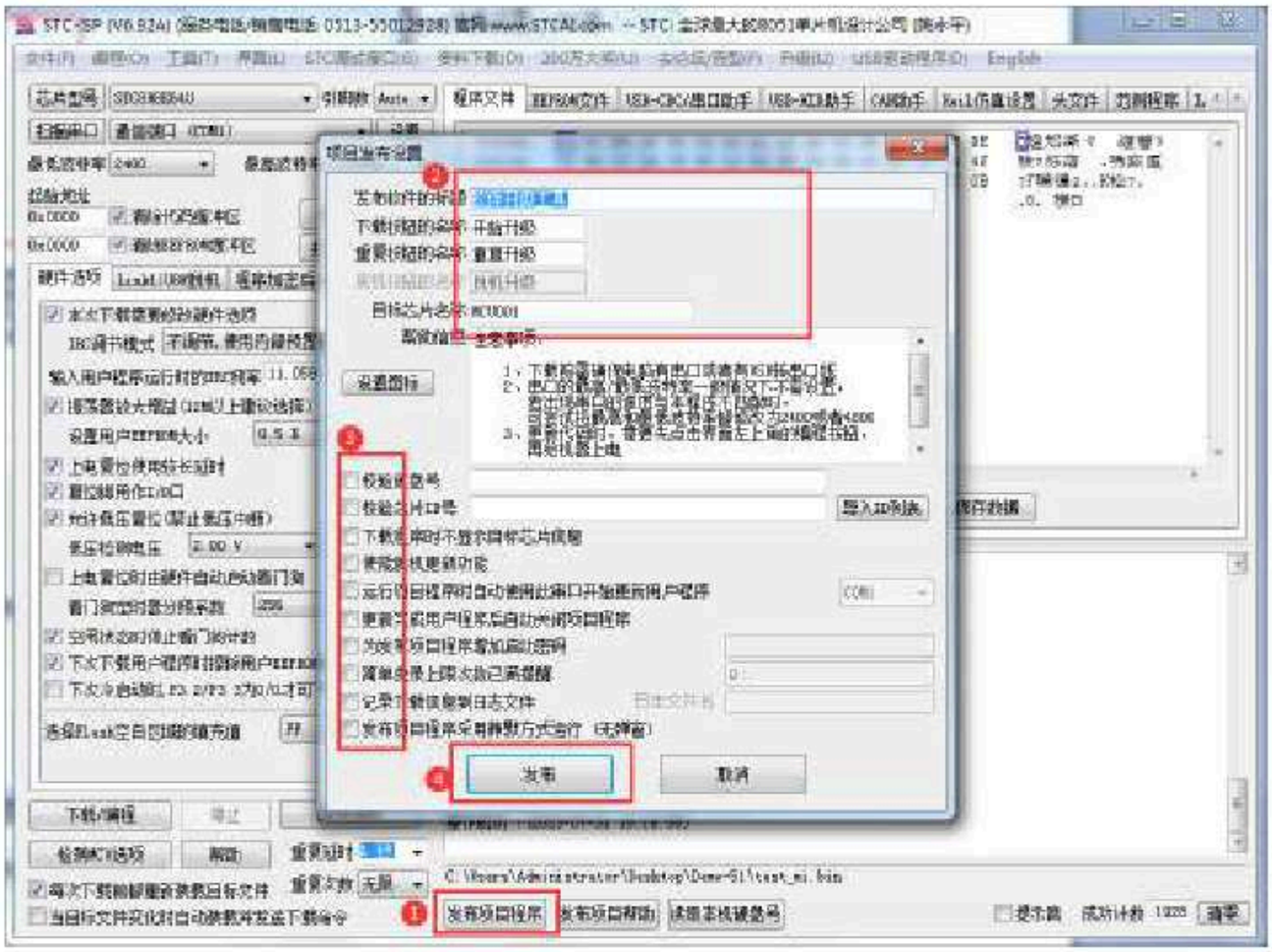
), then subsequent code updates only need to follow the steps, you only need to select the option "The code downloaded this time is in the "Program Encrypted transmission" page (The option "Update the user key before downloading the user code" does not click the "Download Programming" button in the lower left corner of the interface to program the target chip in the normal way.

The download can complete the purpose of updating the user code with the user's own encrypted file (The purpose of preventing the burning person from analyzing the code through the monitoring serial port when burning the program).

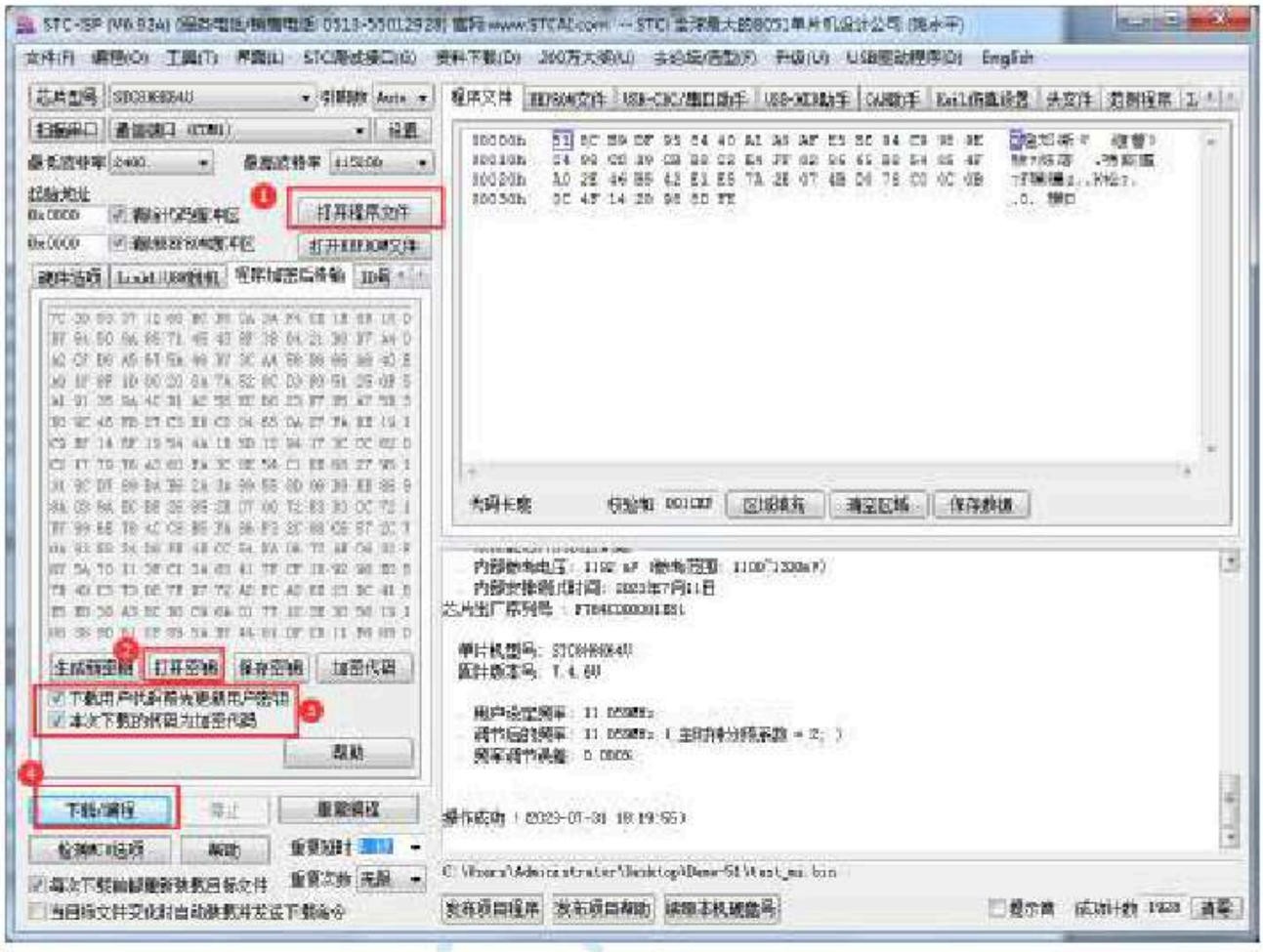
5.14.3 Publish project program+The program is encrypted and used in combination with the transmission

Two new special functions, the release of the project program and the transmission after the program is encrypted, can be used in com program is encrypted can ensure the confidentiality of the user code during the serial communication transmission process during programm the function of remote upgrade for end users (The personnel of the solution company do not needBe present in person). Therefore, the comb purpose of program company manufacturers to allow end users to update the software of the terminal product by themselves when the softw programmers cannot analyze useful programs through the serial port. It is strongly recommended that the program company use it.

The release project program can refer to the following steps, the schematic diagram is as follows：



After the program is encrypted, the chapter steps are shown in the schematic diagram as follows :

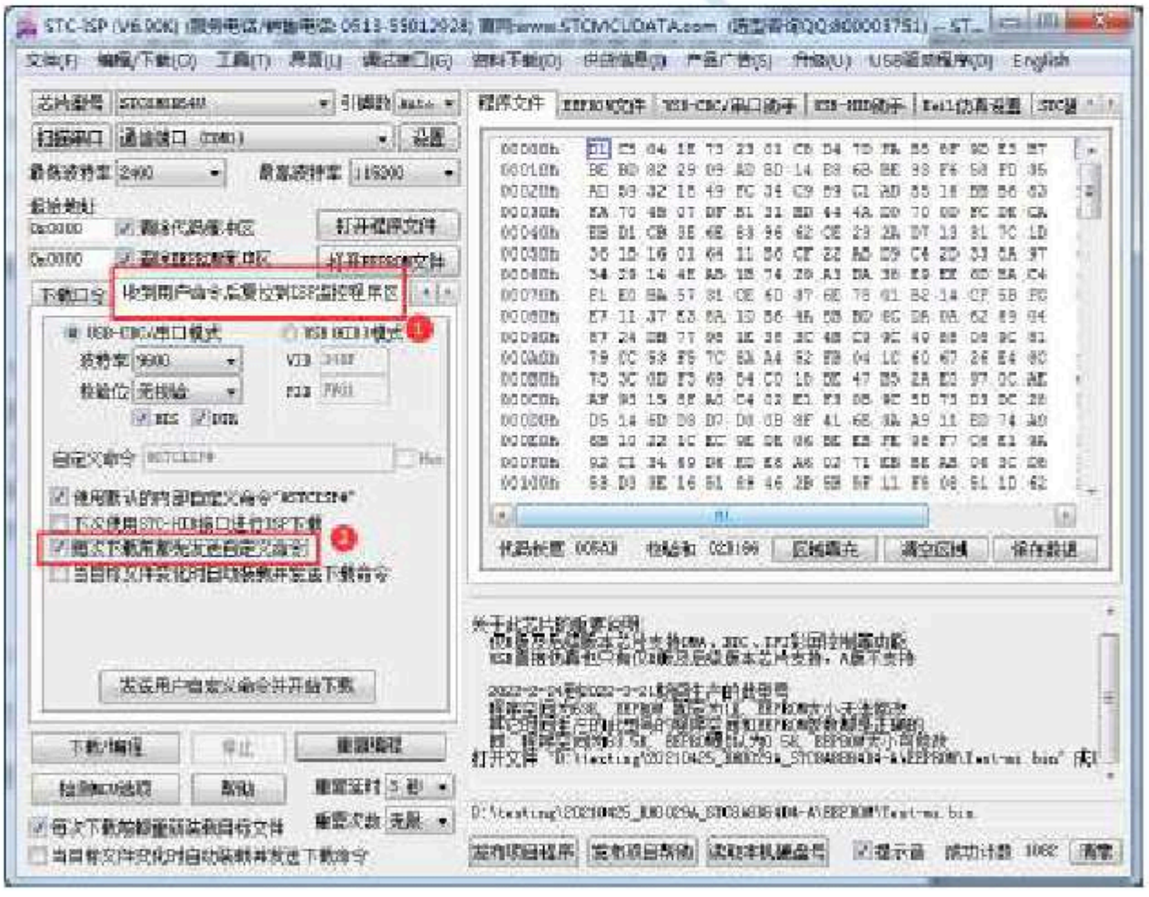


User-defined download (to achieve non-stop download) 5.14.4

Download the user's target program to STC The single-chip microcomputer is executed The system code is initialized with the test computer or USB Communication is achieved. But Inside the MCU ISP The system code will only be executed every time there is a power failure and then th Users are required to power up again every time they need to update the program of the target MCU. Programming tools to power target the chip, they also need to pull down the port to the target MCU during power-up. For projects in the development stage, frequent code changes are required、 Update the code, every time you download it, you need to power it up again, which will cause the operation to be very troublesome.

STC During the hardware design of the single-chip microcomputer, a soft reset register ( ) is that users can set this register by IAP_CONTR It's up to the device to decide STC Re-execute the user code after reset or reset to District execution System code. Current direction ISP register IAP_CONTR Write 0x20 the time， IAP_CONTR When the register is written， Reset after reset to ISP CPU Re-execute the user code after reset, when System code District execution ISP CPU 0x60

To achieve non-stop power ISP Download, the user can design a piece of code in the program, such as detecting a special button, or monito The control serial port waits for a special serial port command, and when it detects that the download conditions are met, the soft reset regist The download System code in order to achieve the trigger condition is an external key, it is monitored in real time in the user code ISP ISP Just press the status. To achieve full synchronization between the software and the user-triggered soft reset, you need to use the software pr The function of "reset to the monitoring program area after receiving the user command". ISP



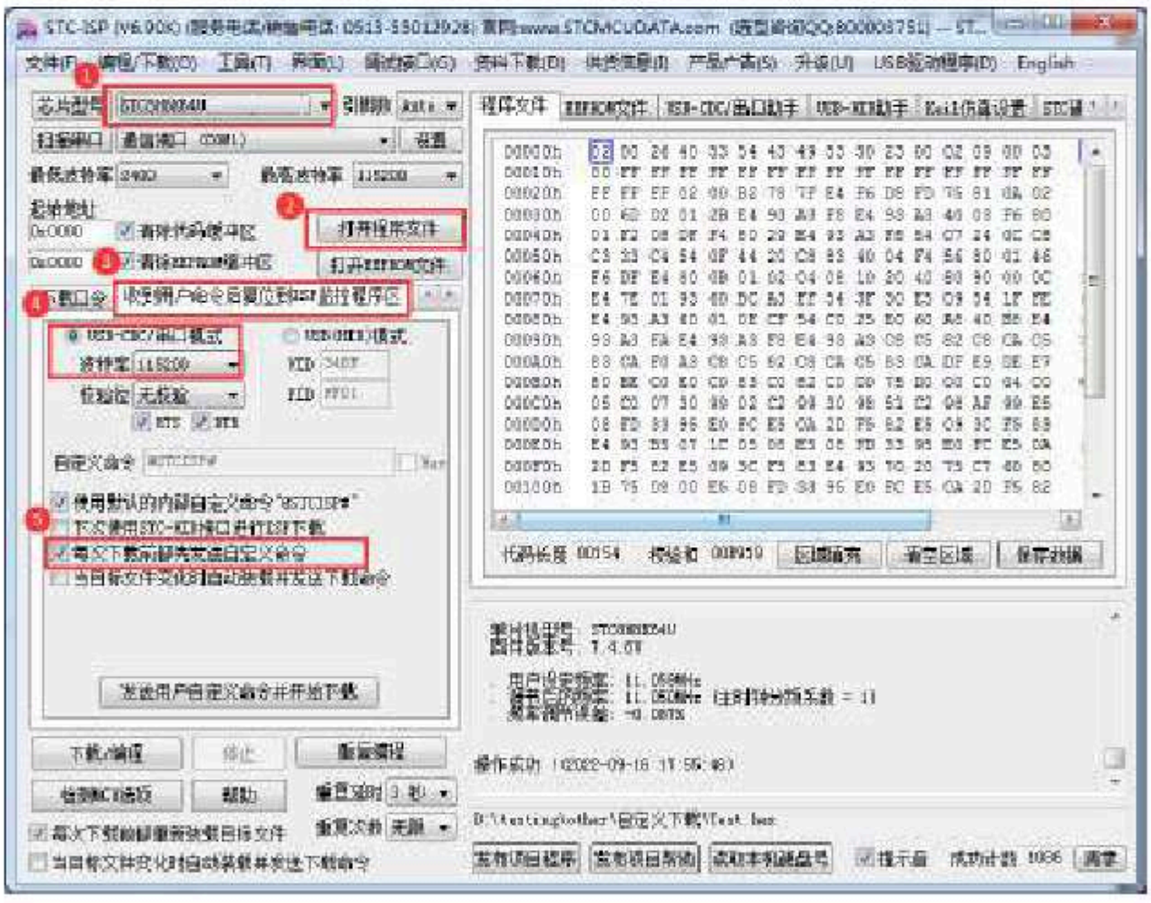
Achieve non-stop programming The steps to download are as follows :

1 , Write user code, and add a serial port command monitoring program to the user code (The reference code is as follows, the test MCU model is STC8H8K64U)

```
#include "stc8h. h"

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

char code *STCISPCMD = "@STCISP#";                  //Custom download command
char index;

void uart_isr() interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        dat = SBUF;                                 //Receive serial data

        if (dat == STCISPCMD[index])                // Determine whether the received data matches the current command chara
        {
            index++;                                //If it matches, index +1
            if (STCISPCMD[index] == '\0')           // Determine whether the command is matched,
                IAP_CONTR = 0x60;                   // and if the match is completed, the soft reset is to
        }
        else
        {
            index = 0;                              // If it doesn't match, You need to start from scratch
            if (dat == STCISPCMD[index])
                index++;
        }
    }
}

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

```
    P3M0 = 0x00; P3M1 = 0x00;

    SCON = 0x50;                                    //Serial port initialization
    AUXR = 0x40;
    TMOD = 0x00;
    TH1 = BAUD >> 8;
    TL1 = BAUD;
    TR1 = 1;
    ES = 1;
    EA = 1;

    index = 0;                                      //Initialization command

    while (1);
}
```

2 , Follow the steps shown in the figure to set up a custom download command (STC-ISP "Default power-up send")



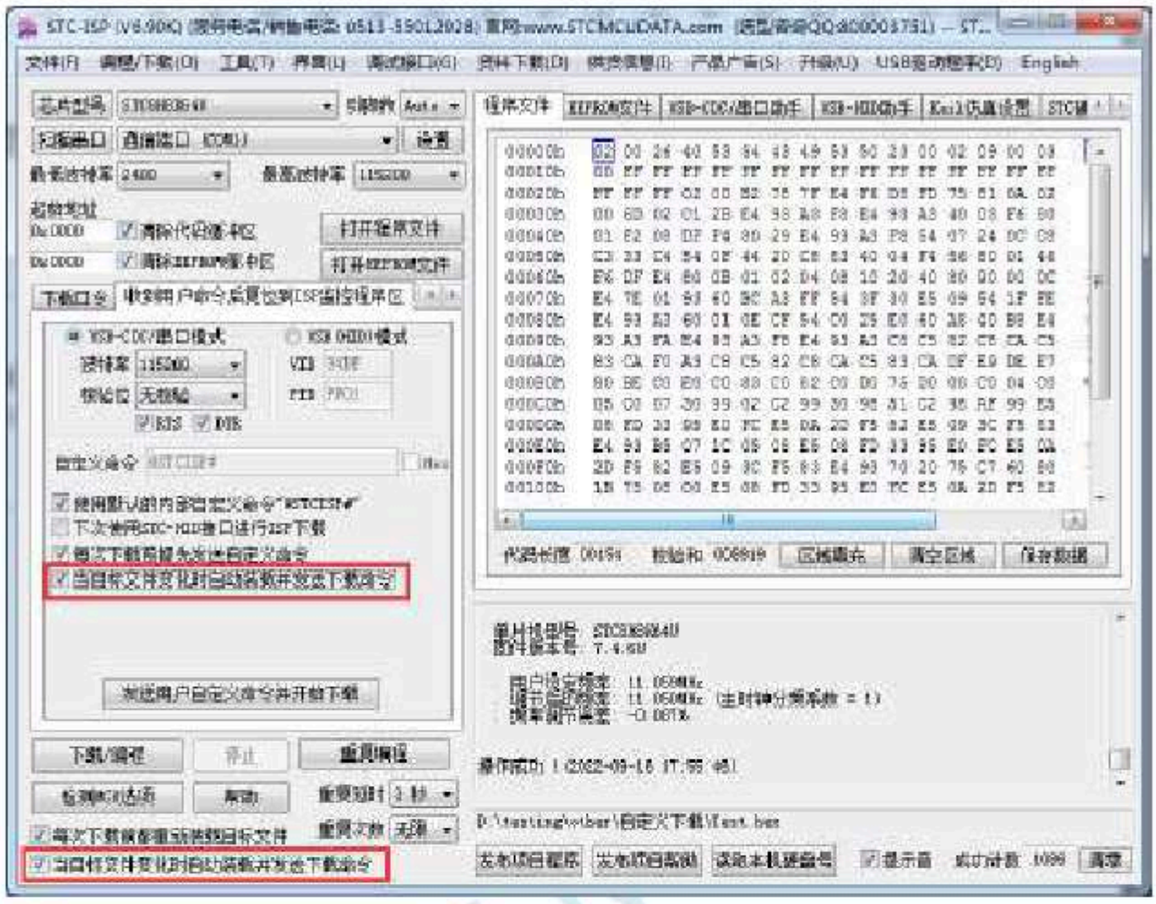
3 , You need to power up the target MCU again during the first download, and you only need to click "Download" in the download software

Button, the download software automatically sends the download command to the target MCU, and the target mcu automatically res

ISP Area, you can achieve non-stop power-up update of the user code.

It can also realize the fully automatic download function during the project development phase, that is, when the downloa

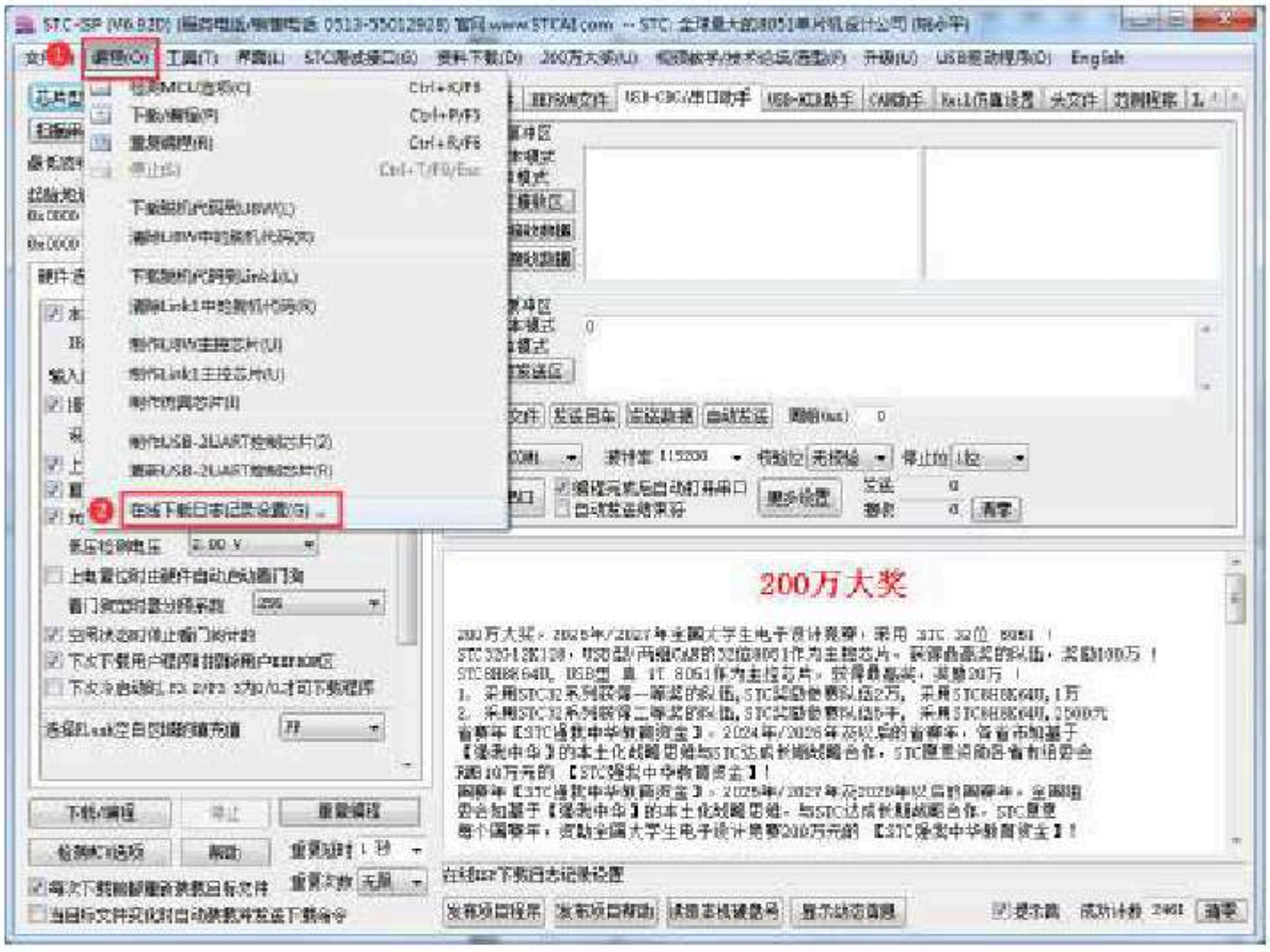
The download command will be automatically sent. To achieve this function, you only need to check any of the two options in the fi



5.14.5 How to simply control the number of downloads, through the actual download The number of MCU

------Download log + Publish advanced application of the project

The first step is to turn on the download log function



1 , Open the "Program Online Download Logging Settings" to open the following window

2 , Click "Online Download Logging"

1, Check "Enable online ISP Download the log recording function"

2 , click the "Browse" button to select the directory

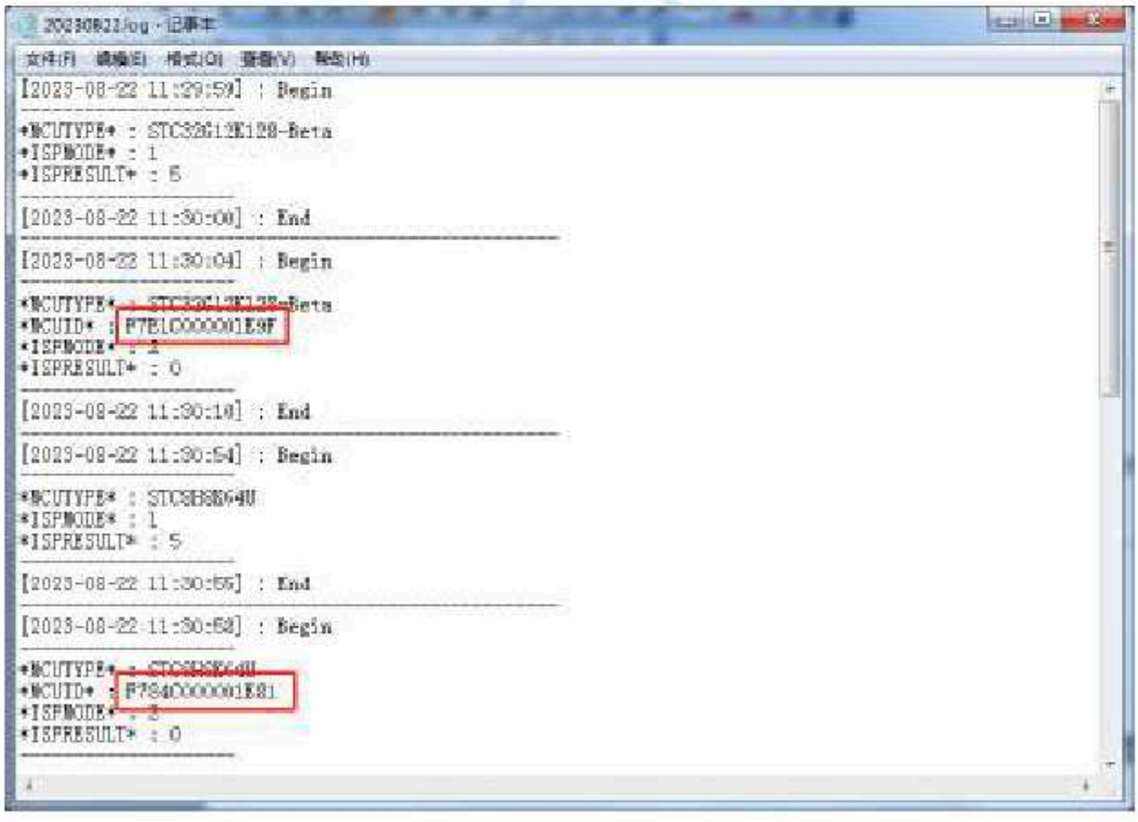
3 where the log file is stored , and click "OK" to confirm

After the setting is complete, all the subsequent ISP The download information downloaded online will be automatically recorded in the file, and the file nam dates have the extension log

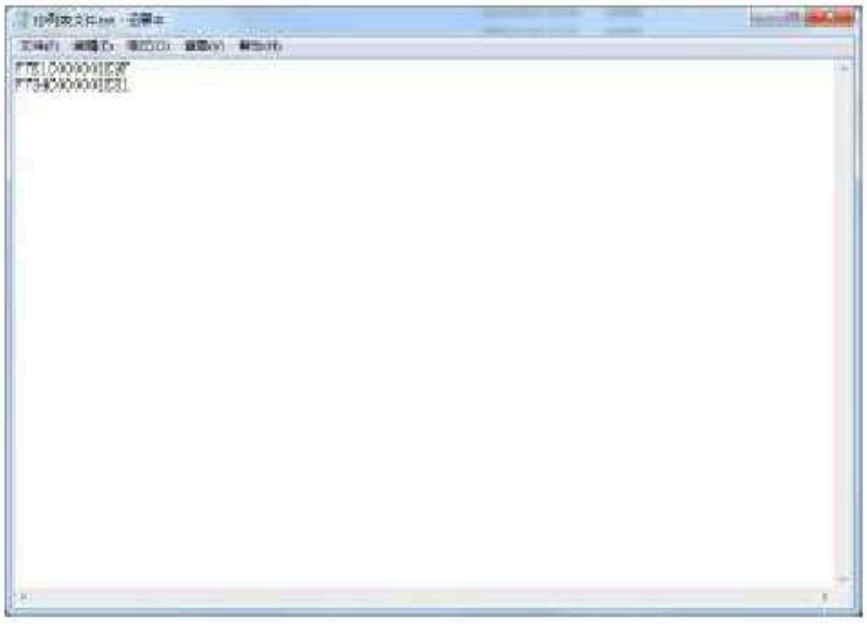
Step 2. Export from the downloaded Number list to list file ID

log file (Note : Ver6.92D Version and later ISP The software can be automatically imported from the list Number, if you need to import it automatically, you can skip this s

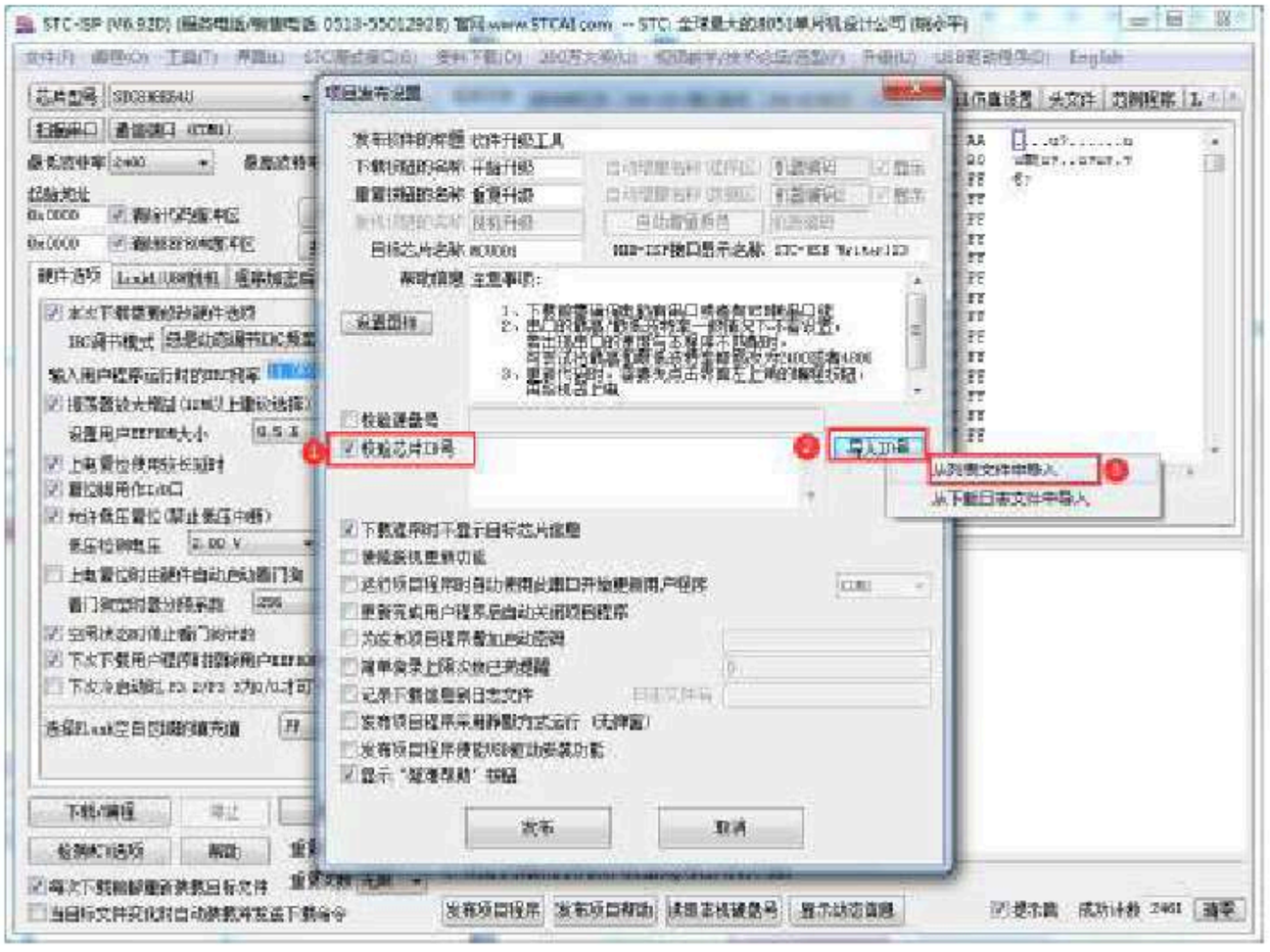
1 , Open the log file of the target date from the log file storage directory (for example, open 22 Log of the day, then open the log text "In the file storage directory" 20230822.log). The logging format is as shown in the figure below :



2 , Copy from the log file ID Enter the number into a list file, as shown in the figure below



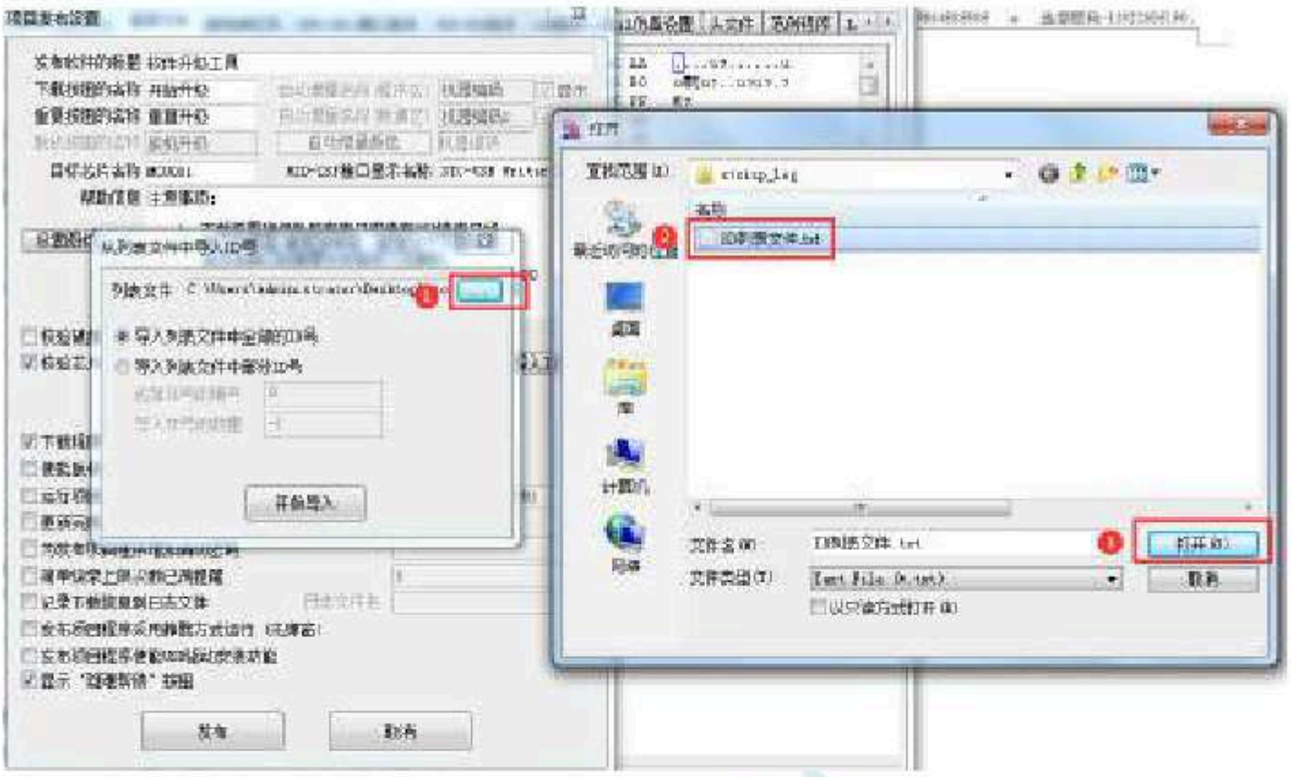
The third step is to import the list file when publishing the project program (if you need to automatically import from the log, you can skip to the fourth step)



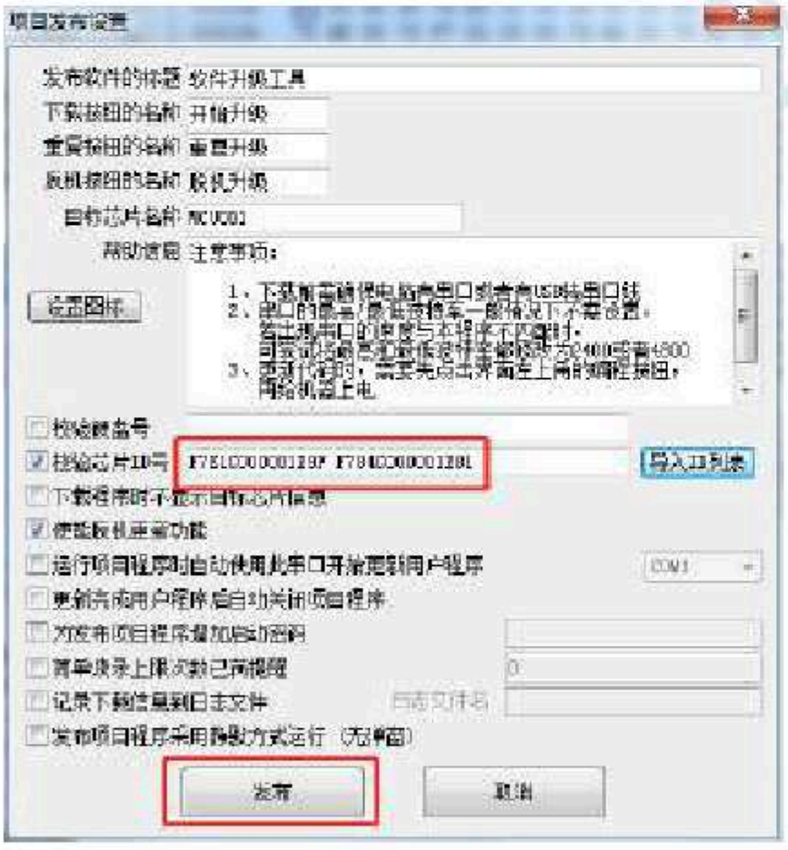
1, click STC-ISP The "Publish Project Program" button in the download interface

2, Check "Check chip number"

3, Click "Import ID Number" , select "Import from List file" , open the list file exported in the previous step

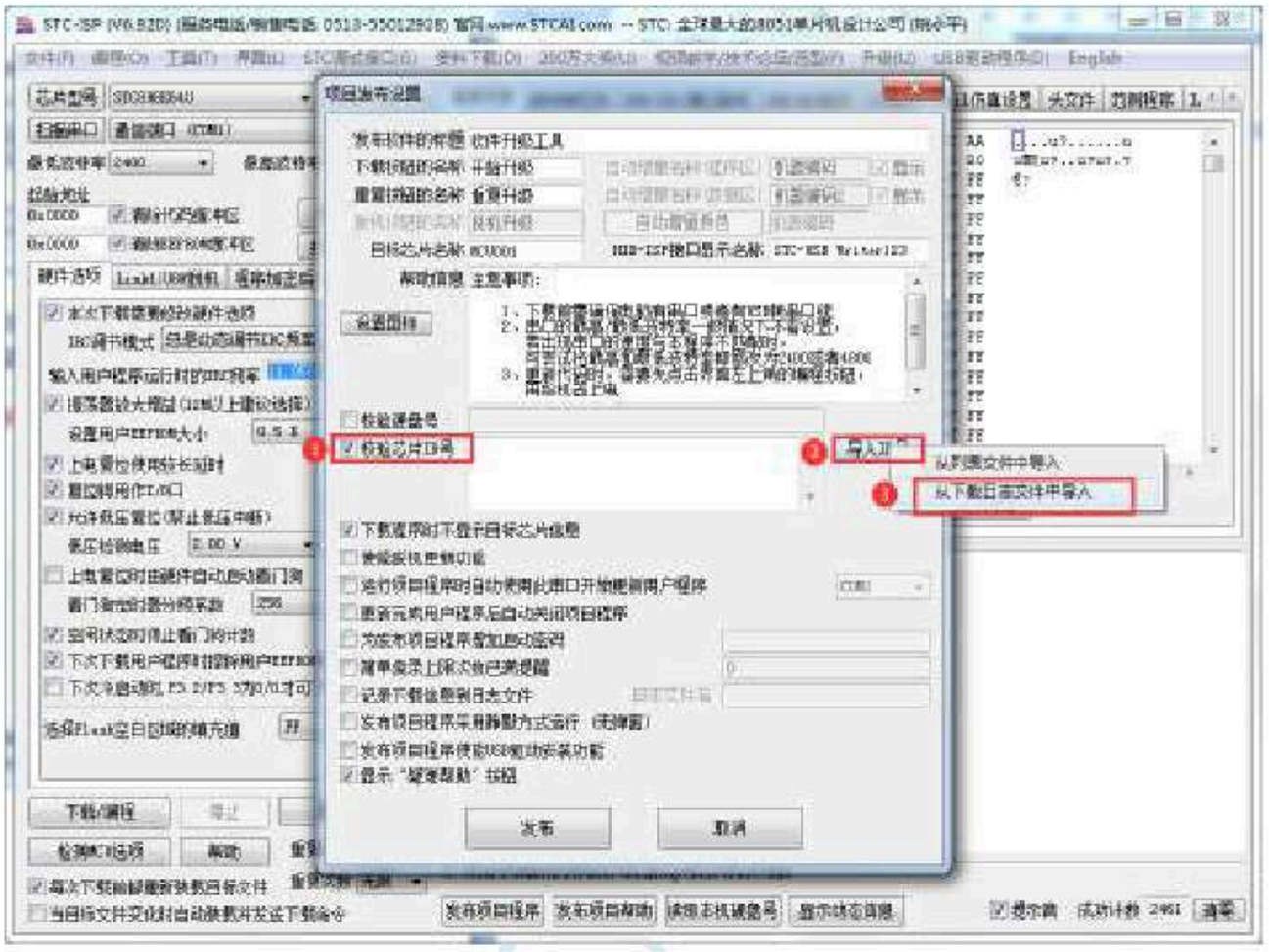


6 , After the list is successfully imported, the following imported will be displayed in the number text box



7 , And finally click the "Publish" button to publish the project.

Step 4. Automatically import from the log file when publishing the project program ID



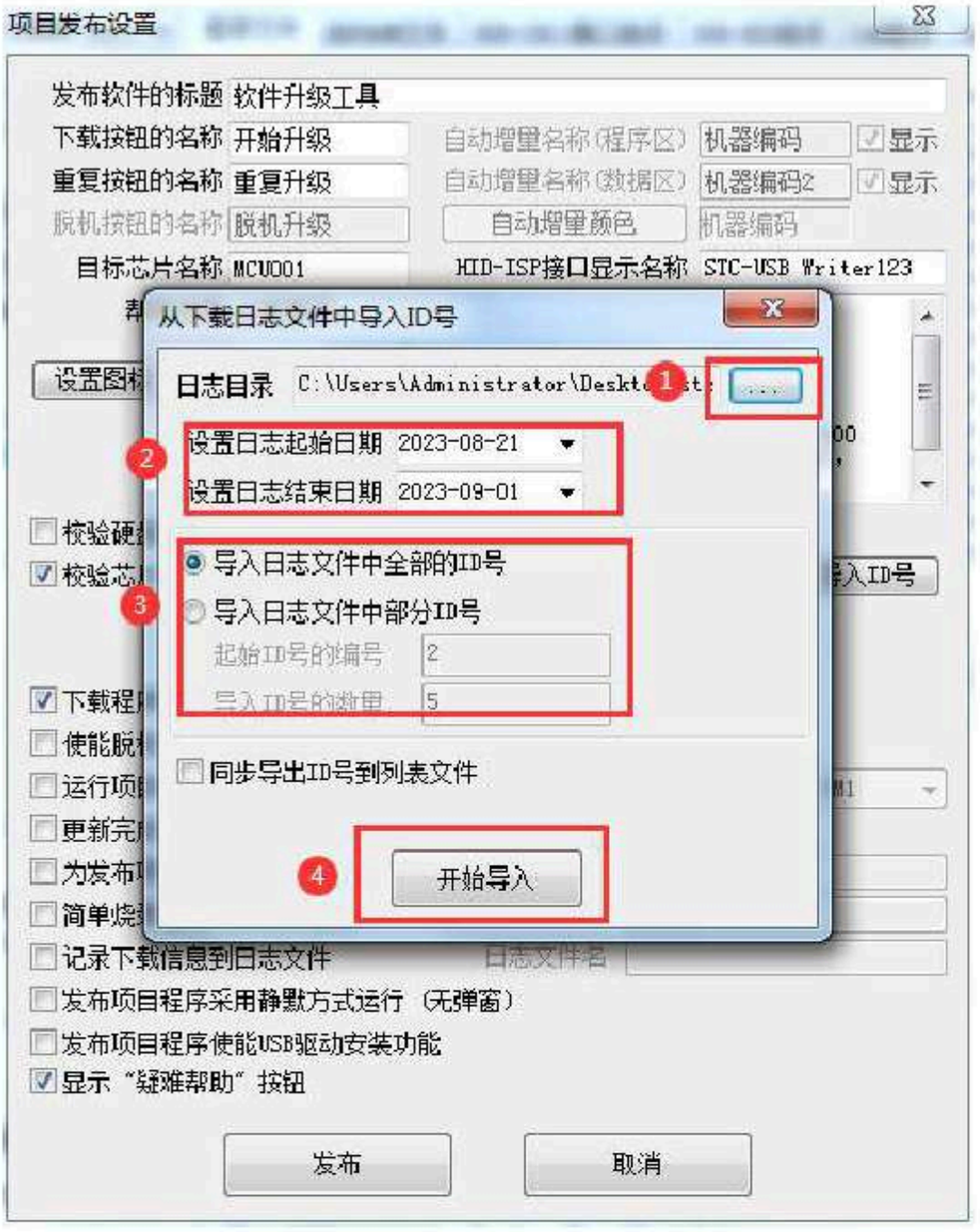
STC-ISP 1, click The "Publish Project Program" button in the download interface

2, Check "Check chip Number"

3, Click "Import ID

4 Number" , select

"Import from download log file"

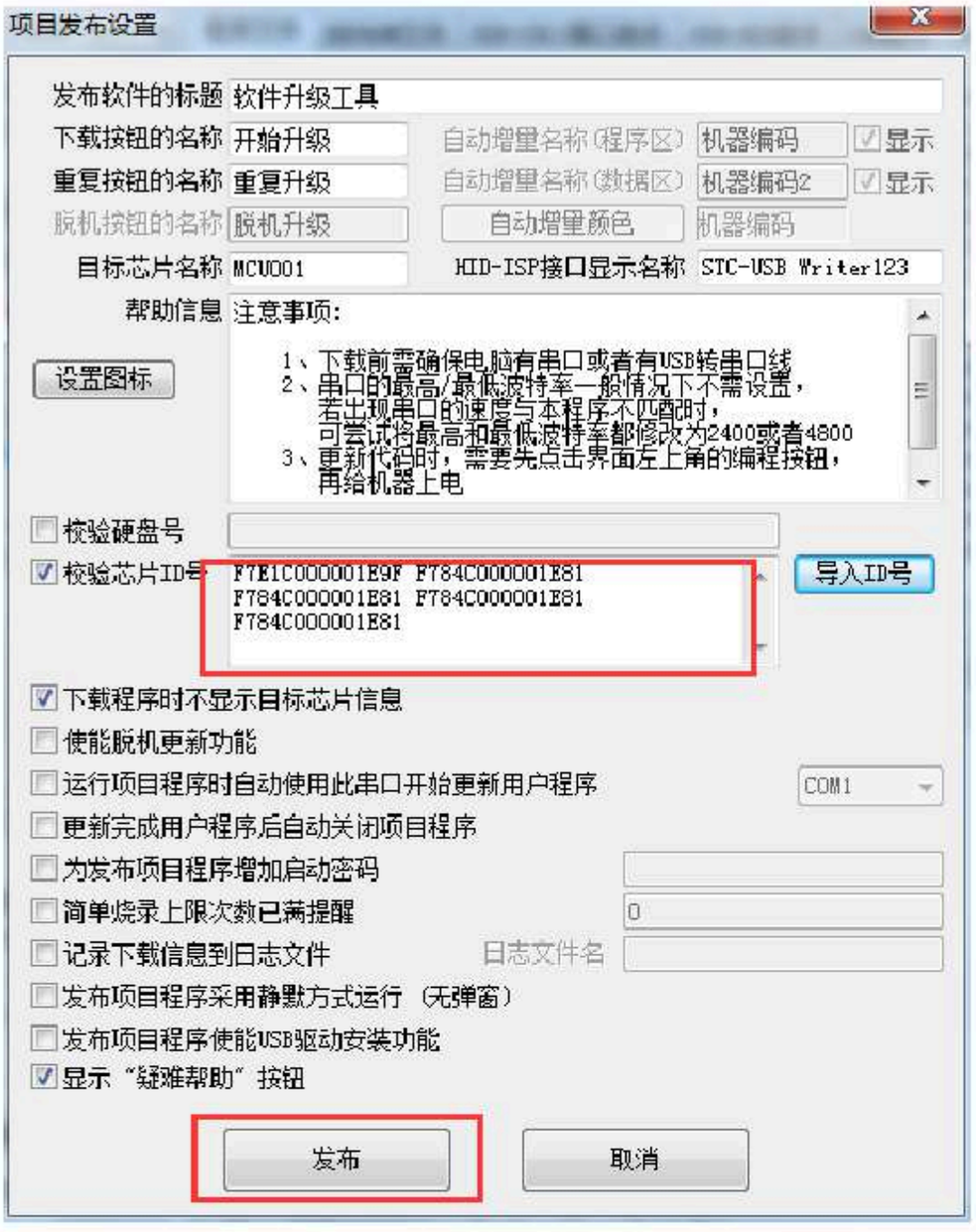


5、 **Open the log save directory**

6 **, Set the start time and end time of the logs that need to be imported**

7**, Select the one that needs to be imported** **After the serial number and list of the**

8 **number are successfully imported, the following list will be displayed in the number text box**



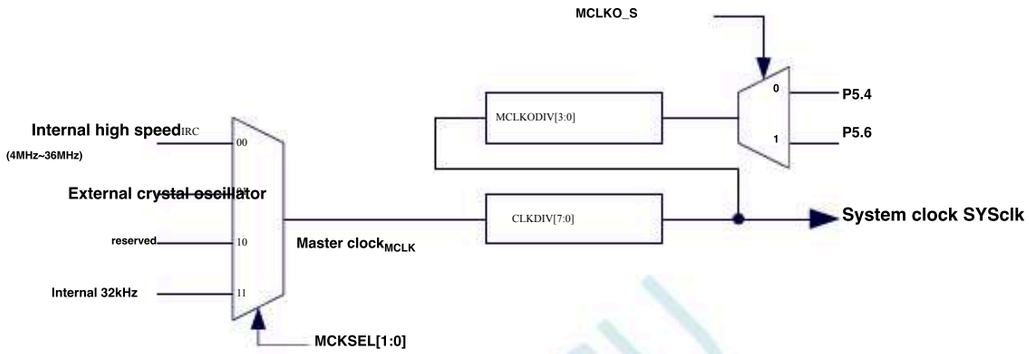
9 , **And finally click the "Publish" button to publish the project.**

6 Clock, reset and power management, chip-on-chip operation process

6.1 System clock control

The system clock controller is a high-precision internal peripheral systems provide a clock source, the system clock has a clock source to choo part of the single-chip microcomputer IRC, internal (Large error), external crystal oscillator. The user can enable and disable each clock separately thr Source, and the internal clock division is provided to achieve the purpose of reducing power consumption.

After the microcontroller enters the power-down mode, the clock controller will turn off all clock sources



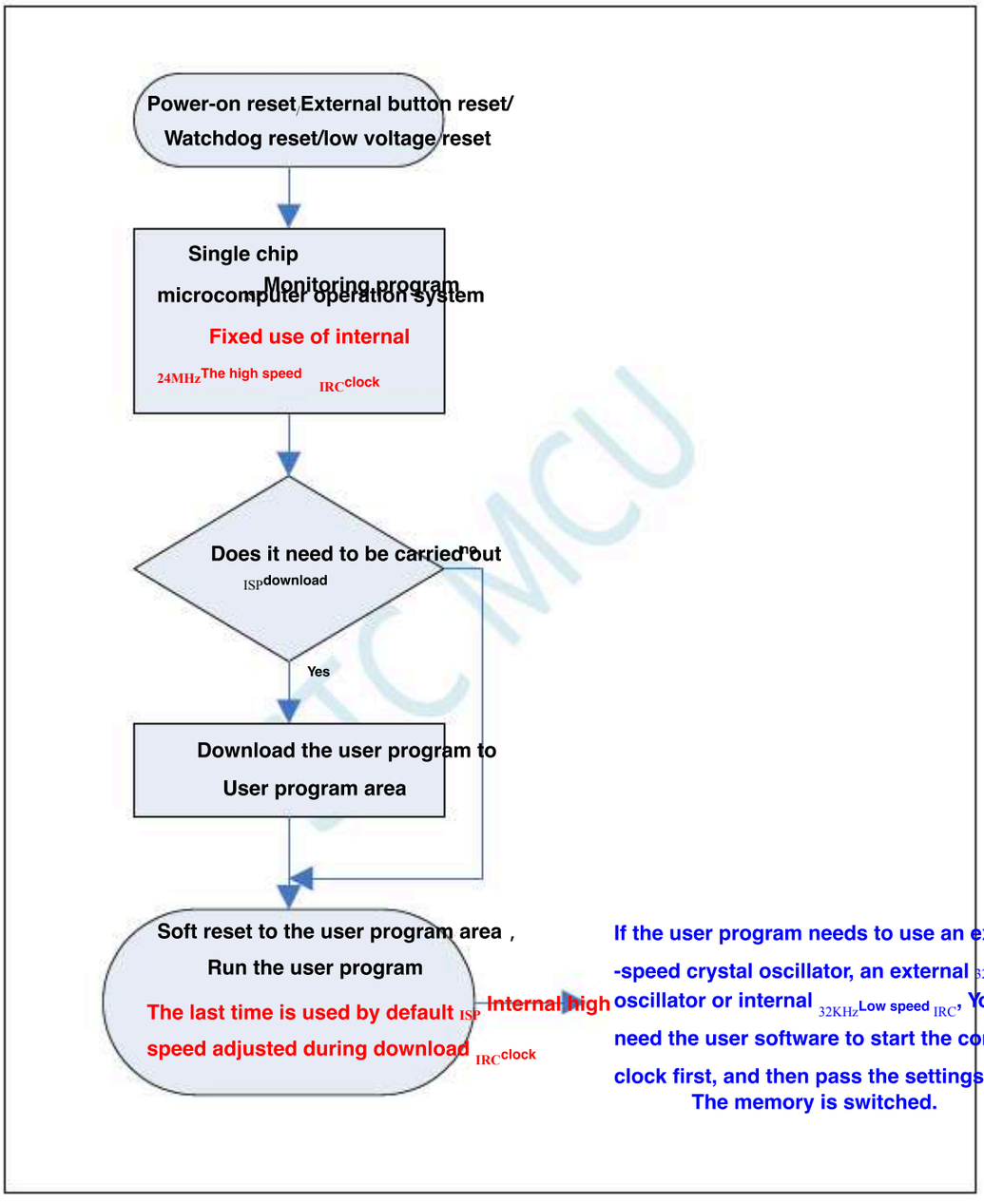
System clock structure diagram

6.2 Chip-on-chip operation process：

Power-on reset, Reset foot reset, Watchdog reset, When the low voltage detection is reset, the chip defaults to start to execute the code, at th
Fixed use of internal high speed 24MHz IRC Clock, when you need to download the user program and reset to the user program area after the do
When you want to download and reset directly to the user program area, the default is to use the high speed adjusted
when the user downloaded last time. Use an external high-speed crystal oscillator and an external low-speed oscillator. 32KHz
IRC 32.768KHz Clock, and then through the settings CLKSEL

Clock, if the user program needs IRC
You need the user software to start the corresponding tir
The crystal oscillator
or internal register is switched.

The startup process is as follows：



6.3 Related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	Clock selection register	FE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	Clock divider register	FE01H									nnnn,nnnn
HIRCCR	Internal high-speed oscillator control register	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	External crystal oscillator control register	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	inside 32K Oscillator control register	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST 0xxx,xxx0	
MCLKOCR	Master clock output control register	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000

6.3.1 System clock selection register (CLKSEL)

symbol	address	B7	B6	B5	B4	B3	B2		B0 B1
CLKSEL	FE00H	-						MCKSEL[1:0]	

MCKSEL[1:0]: Master clock

source selection MCKSEL[1:0]	High speed and high
00	precision inside the main clock source IRC
01	External crystal oscillator or external clock
10	External crystal oscillator or
11	external clock internal low speed 32KHz IRC

6.3.2 Clock divider register (CLKDIV)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

: Master clock division coefficient. System clock Yes to the master clock The clock signal after frequency division. SYSCLK MCLK

CLKDIV	System clock frequency
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
...	...
x	MCLK/x
...	...
255	MCLK/255

Note: After the user program is reset, the system will automatically load the frequency division coefficient required for the operating frequency set at the IRC The initial value of the device

6.3.3 Internal high speed and high precision IRC Control register (HIRCCR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
HIRCCR	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST

ENHIRC: Internal high speed and high precision IRC Enable bit

0: Close the internal high precision IRC

1: Enable internal high precision IRC

HIRCST: Internal high speed and high precision IRC frequency stability flag. (Read-only bit)

0: After it is enabled from the stopped vibration state, a period of time must pass before the frequency of the oscillator wi

After stabilization, the clock controller will automatically position the flag. So when the user program needs to switch the clock to use th

When, you must first set the enable oscillator, and then keep querying the oscillator stability flag. ENHIRC=1 HIRCST, until the flag bit becomes 1, before the clock source can be switched.

6.3.4 External oscillator control register (XOSCCR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST

: Enable bit of external crystal oscillator ENXOSC

0 : Turn off the external crystal oscillator

1 : Enable the external crystal oscillator

XITYPE : External clock source type

0 : The external clock source is an external clock signal (or active crystal oscillator). The signal source only needs to be connected to the microcontroller XTALI (P5.6) (At this time P5.6 The port is fixed as a high-impedance input mode, which can be used to read external digital signals)

: The external clock source is a crystal oscillator. The signal source is connected to the microcontroller 1 P5.7) and XTALO (XTALI (P5.6)

XOSCST : The frequency stability flag of the external crystal oscillator.

When the external crystal oscillator is enabled from the stopped state, it must pass a period of time before the frequency of the oscillator will stabilize. When the frequency is stable, the clock control Flag position. So when the user program needs to switch the clock to make automatically use the internal low speed, which must first be set 1 When using the crystal oscillator, enable the oscillator , ENXOSC=1 XOSCST must be set, then keep querying the oscillator stability flag until the flag bit becomes, the clock source can be switched. 1

6.3.5 inside 32KHz Low speed IRC Control register (IRC32KCR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
IRC32KCR	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

ENIRC32K : internal 32K Low speed IRC Enable bit

0 : Close the interior Low speed IRC

1 : Enable internal Low speed IRC

IRC32KST : internal 32K Low speed IRC Frequency stability flag (Read-only bit)

When inside 32K Low speed IRC After it is enabled from the stopped vibration state, a period of time must pass before the frequency of the osc

After the frequency of the device is stable, the clock control Flag position. So when the user program needs to switch the clock to make automatically use the internal low speed, which must first be set IRC32KST ENIRC32K 32 Enable the oscillator, and then keep querying the oscillator stability fl IRC bit IRC32KST , until the flag bit becomes 1 Only then can the clock source be switched.

6.3.6 Master clock output control register (MCLKOCR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	FE05H	MCLKO_S	MCLKODIV[6:0]						

MCLKODIV[6:0] : Master clock output frequency division coefficient

(Note: The clock source of the master clock divider output is through the system clock after divider)

MCLKODIV[6:0]	System clock divider output frequency
0000000	Does not output clock
0000001	SYSClk/1
0000010	SYSClk /2
0000011	SYSClk /3
. . .	. . .
1111110	SYSClk /126
1111111	SYSClk /127

MCLKO_S : System clock output pin selection

0 : System clock divider output to P5.4 Mouth

1 : System clock divider output to P5.4 to mouth

6.4 STC12H Inside the series IRC Frequency adjustment

STC12H The series of microcontrollers are integrated with a high-precision internal IRC oscillator. In use ISP Download the software to download when, ISP The downloaded software will be selected according to the user/The set frequency is automatically adjusted, and the general frequency value can be adjusted to ±0.3% or less. After the frequency is within the full temperature range (-40 °C ~ 85°C) The temperature drift can reach -1.35% ~ 1.30%.

STC12H Inside the series IRC There are two frequency bands, the center frequency of the frequency band Frequency band adjustment range The circumference is approximately 20MHz 13.5MHz ~ 27MHz 35M The adjustment range of the frequency band is approximately 23MHz ~ 47MHz (Note: different chips and different There may be a manufacturing error of about 5% in the generated batch)

After actual testing, the maximum operating frequency of some chips can only be 39.5MHz, the To For safety reasons, it is recommended that users set it when downloading the frequency should not be higher than 35MHz

Note: For general users, internal IRC You don't need to care about the adjustment of the frequency, because the frequency adjustment ISP At the time of download It has been completed automatically. Therefore, if the user does not need to adjust the frequency by himself, then the following related functions will lead to changes in the operating frequency. Registers cannot be modified at will, otherwise it can be

If the user needs to dynamically select the preset frequency of the chip in his own code, please refer to the IRC preset frequency list and the sample program of "User-defined internal frequency".

inside IRC Frequency adjustment mainly uses the following registers to adjust

Related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IRCBAND	IRC Frequency band selection	9DH	-	-	-	-	-	-	-	SEL	0000,00nn
LIRTRIM	IRC frequency fine-tuning register	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC Frequency adjustment register	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	Clock divider register	FE01H	CLKDIV[7:0]								nnnn,nnnn

6.4.1 IRC Frequency band selection register (IRCBAND)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	9DH	-	-	-	-	-	-	-	SEL

SEL: Frequency band selection

0: Choose 20MHz Frequency band

1: Choose 35MHz Frequency band

6.4.2 inside IRC Frequency adjustment register (IRTRIM)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
IRTRIM	9FH	IRTRIM[7:0]							

IRTRIM[7:0]: Internal high precision IRC Frequency adjustment register

IRTRIM Yes IRC Frequency 256 For the adjustment of each level, the frequency value adjusted by each level is linearly distributed as a whole. There will be fluctuations in the ministry. Macroscopically, the frequency adjusted at each level is approximately 0.24% (The frequency ratio at time IRTRIM n+1 IRTRIM n) The frequency of the time is about fast 0.24%, But because of every level of frequency adjusted IRC % (The frequency adjusted at each level The maximum value is approximately 0.55%, the minimum value is about 0.02%, the overall average is about 0.24%), so it will cause local fluctuations.

6.4.3 inside IRC Frequency trimmer register (IRTRIM)

symbol	address	B7	B6	B5	B4	B3	B2		B0 B1
LIRTRIM	9EH	-	-	-	-	-	-	IRTRIM[1:0]	

LIRTRIM[1:0]: Internal high precision IRC Frequency fine-tuning register

LIRTRIM Yes IRC frequency is carried out Adjustment of levels ,

LIRTRIM[1:0]	Adjusted frequency range
00	Do not fine -tune the
01	adjustment about 0.10 %
10	adjust approximately 0.04 %
11	adjust approximately 0.10 %

The frequency range adjusted by each level is shown in the following table :

6.4.4 Clock divider register (CLKDIV)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

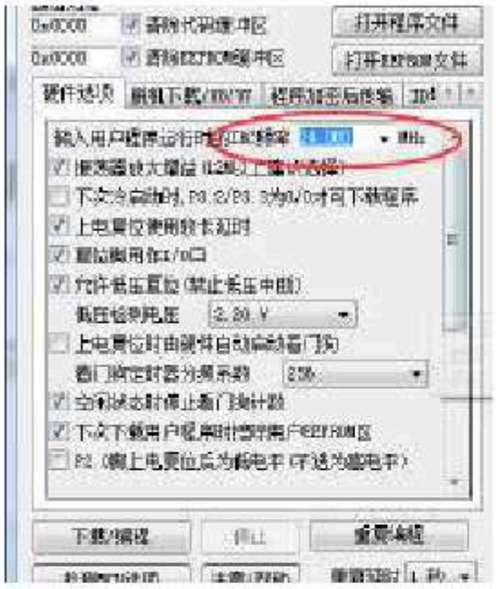
: Master clock division coefficient. System clock Yes to the master clock The clock signal after frequency division.

CLKDIV	System clock frequency
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
...	...
x	MCLK/x
...	...
255	MCLK/255

STC The adjustable range of the two frequency bands inside the series 25MHz 27MHz and 25.3MHz ~ 43.6MHz although 35MHz is the upper limit of the frequency band, which can be adjusted to memory inside the chip cannot be run Above speed , Therefore, the internal frequency set by the user when downloading cannot be higher than 35MHz , It is generally recommended that users set it to 40MHz IRC When users need a lower operating frequency, they can The register divides the adjusted frequency by frequency. For example, the us 11.0592MHz The frequency of using the internal cannot get this frequency if he needs to adjust it directly, but it can be adjusted internally t 22.1184MHz, in use CLKDIV Proceed 2 Divide by frequency to get 11.0592MHz.

6.4.5 Divide out 3MHz Example of the user's working frequency, and the user dynamically changes the fr

In order to get 3MHz The frequency can be used 24MHz Different methods. When downloading, the first in progress IRC select the internal operating frequency to be 24MHz, As shown in the figure below ,



Then select the clock source as internal IRC and use CLKDIV The register divides the frequency.

C Language code

```
//The test operating frequency is 24MHz

#include "reg51.h"
#include "intrins.h"
#define
CLKSEL          (*(unsigned char volatile xdata *)0xfe00)
#define CLKDIV          (*(unsigned char volatile xdata *)0xfe01)
#define HIRCCR          (*(unsigned char volatile xdata *)0xfe02)
#define XOSCCR          (*(unsigned char volatile xdata *)0xfe03)
#define IRC32KCR        (*(unsigned char volatile xdata *)0xfe04)
sfr P_SW2
sfr IRTRIM          =   0xba;
sfr P0M1            =   0x9f;
sfr P0M0
sfr P1M1            =   0x93;
sfr P1M0            =   0x94;
sfr P2M1            =   0x91;
sfr P2M0            =   0x92;
sfr P3M1            =   0x95;
sfr P3M0            =   0x96;
sfr P4M1            =   0xb1;
sfr P4M0            =   0xb2;
sfr P5M1            =   0xb3;
sfr P5M0            =   0xb4;
void main()         =   0xc9;
{                   =   0xca;


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;
    CLKSEL = 0x00;
    CLKDIV = 0x08;
    P_SW2 = 0x00;
    IRTRIM++;
//  IRTRIM--;
    while (1);
}
```

IRC (// Choose default internal

// Clock divider

//IRC Fine-tune the frequency up ‰ (pay attention to the judgment boundary)

//IRC Fine-tune the frequency down ‰ (pay attention to the judgment boundary)

Assembly code

; The test operating frequency is 24MHz

```
P_SW2       DATA    0BAH
IRTRIM      DATA    09FH

CLKSEL      EQU     0FE00H
CLKDIV      EQU     0FE01H
HIRCCR      EQU     0FE02H
XOSCCR      EQU     0FE03H
IRC32KCR    EQU     0FE04H

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
```

```
    MOV     P3M0, #00H
    MOV     P3M1, #00H
    MOV     P4M0, #00H
    MOV     P4M1, #00H
    MOV     P5M0, #00H
    MOV     P5M1, #00H

    MOV     P_SW2,#80H
    MOV     A,#00H          ;Choose internal
    MOV     DPTR,#CLKSEL
    MOVX    @DPTR,A
    MOV     A,#08H          ;Clock divider
    MOV     DPTR,#CLKDIV
    MOVX    @DPTR,A
    MOV     P_SW2,#00H

    INC     IRTRIM          ;IRC Fine-tune the frequency up ‰ (pay attention to the judgment boundary)
;   DEC     IRTRIM          ;IRC Fine-tune the frequency down ‰ (pay attention to the judgment boundary)

    JMP     $

    END
```

6.5 System reset

STC12H The reset of the series of microcontrollers is divided into two types: hardware reset and software reset.

When the hardware is reset, the values of all registers will be reset to their initial values, and the system will re-read all hardware options same time, power-up waiting is carried out according to the power-up waiting time set by the hardware options. Hardware reset mainly includ

Power-on reset , POR, nearby 1.7V

Low voltage reset, LVRESET (2.0V , 2.4V , 2.7V , 3.0V nearby)

Reset foot reset(Low-level reset) Watchdog reset

When the software is reset, except for the clock-related registers that remain unchanged, the values of all other registers will be reset to their initial values.Resetting the file will not re-read all hardware options. Software reset mainly includes :

write IAP_CONTR of SWRST Triggered reset

Related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
WDT_CONTR	Watchdog control register	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_CONTR	IAP Control register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	IAP_WT[2:0]			0000,x000
RSTCFG	Reset configuration register	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

6.5.1 Watchdog reset (WDT_CONTR)

In systems that require high reliability such as industrial control/automotive electronics/aerospace, in order to prevent "the system from Causing the system to work abnormally for a long time", usually the introduction of a watchdog, if Program running and flying Do not reset as required within the specified tir If you ask for access to the watchdog, if in an abnormal state, the watchdog will force MCU/CPU , so that the system can be executed from scratch again considered a user program.

The watchdog reset of the STC8 series is one of the hardware resets in the hot start reset The introduction of this function in the series of microcontrollers has the microcontroller system more convenient and concise. After the STC8 series watchdog reset state is over, the system is fixed to start from the ISP moni program area , which has nothing to do with the SWBS of the IAP_CONTR register before the watchdog reset (Note: This is different from the STC15 series MCU)

WDT_CONTR (Watchdog Control register)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
WDT_CONTR	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]		

WDT_FLAG: Watchdog Overflow sign

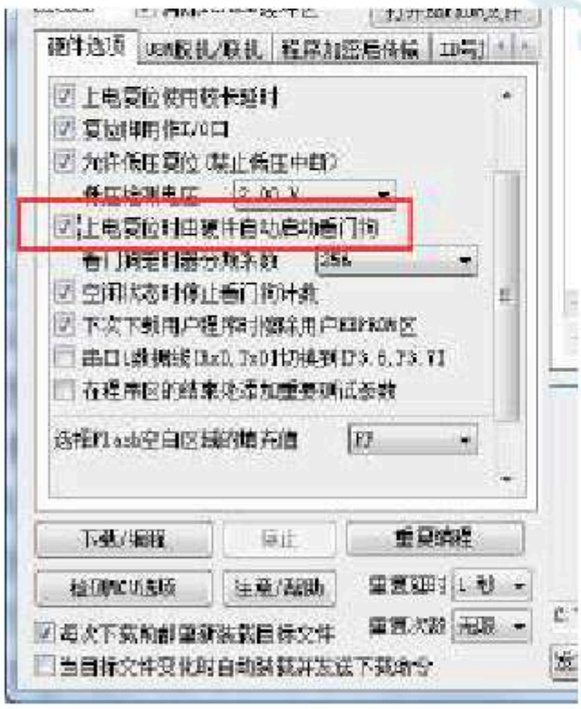
When the watchdog overflows, the hardware will automatically reset this location, and the software needs to be cleared to zero. EN_WDT: Watchdog enable bit

0: No effect on the MCU

1 : Start the watchdog timer.

Note: The watchdog timer can be started by software or automatically by hardware. Once the watchdog timer is started, the software will not be able to be turned off. The microcontroller must be powered on again before it can be turned off. The software startup wat Bit write

1 That's it. If you need hardware to start the watchdog, you need to set it up as shown in the figure below when downloading : ISP



: The watchdog timer is cleared to zero CLR_WDT

0 : No effect on the MCU

1 : Clear the watchdog timer, the hardware will automatically reset this bit

IDL_WDT : IDLE Watchdog control bit in mode

0 : IDLE The watchdog stops counting when in mode

In mode, the

IDLE

watchdog continues to count 1 :

WDT_PS[2:0] : Watchdog timer clock frequency division coefficient

WDT_PS[2:0]	Frequency division coefficient	12M Overflow time at the main	20M Overflow time at the main
000	2	frequency ≈ 65.5 milliseconds ≈131 milliseconds ≈ 262 milliseconds ≈524 milliseconds ≈ 1.05seconds ≈ 2.10 seconds ≈ 4.20 seconds ≈ 8.39 seconds	frequency ≈ 39.3 milliseconds ≈ 78.6 milliseconds 1157 milliseconds 3 315 milliseconds ≈ 629 milliseconds ≈ 1.26 seconds ≈ 2.52 seconds ≈ 5.03 seconds
001	4		
010	8		
011	16		
100	32		
101	64		
110	128		
111	256		

The watchdog overflow time calculation formula is as follows :

$$\text{Watchdog overflow time} = \frac{12 \times 32768 \times 2^{(WDT\_PS+1)}}{SYSclk}$$

6.5.2 Software reset (IAP_CONTR)

IAP_CONTR (IAP Control register)

symbol	address		B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H		IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

SWBS : Software reset, start selection

0: After the software is reset, the code execution starts from the user program area. The data in the user data area remains unchanged.

1: After the software is reset, the code execution starts from the system area. The data in the user data area will be initialized. ISP

SWRST: Software reset trigger bit

0: No effect on the MCU

1: Trigger software reset

6.5.3 Low voltage reset (RSTCFG)

(Reset configuration register) RSTCFG

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
RSTCFG	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]	

ENLVR: Low voltage reset control bit

0 :Low-voltage reset is prohibited. When the system detects a low-voltage event, a low-voltage interrupt is generated

1 : a low-voltage reset is enabled. When the system detects a low voltage event, it automatically resets

P54RST : Pin function selection RST

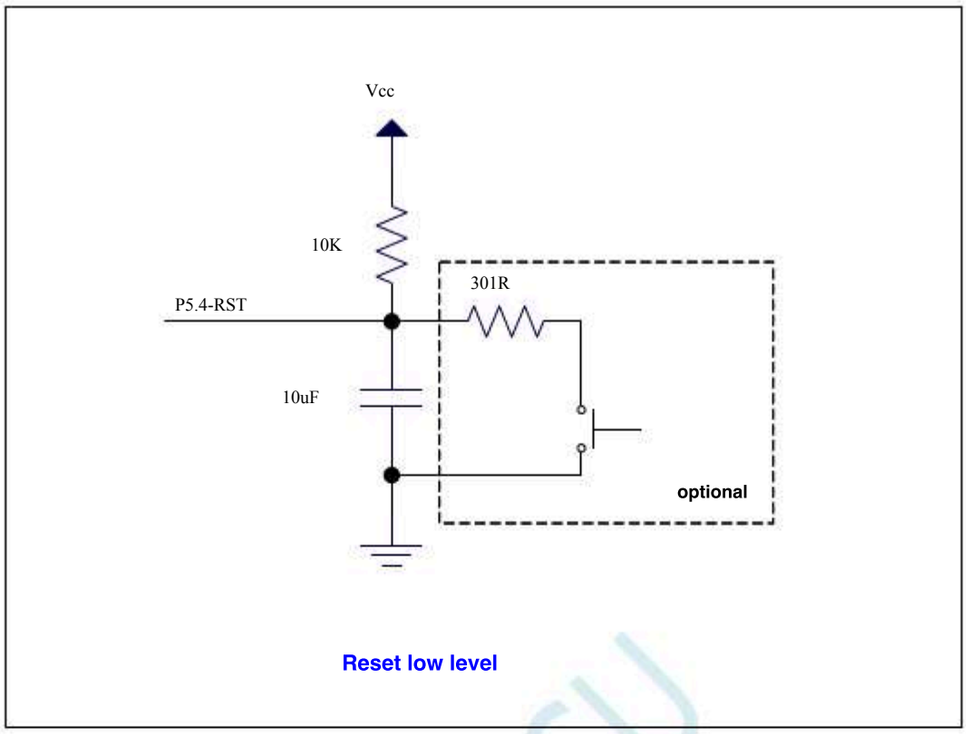
0 : RST The pins are used as ordinary I/O (P54)

1 : RST The pin is used as a reset pin (Low-level reset)

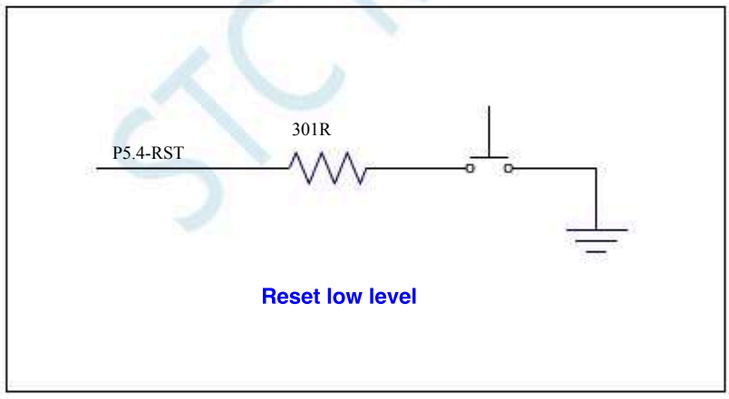
LVDS[1:0] : Low voltage detection threshold voltage setting

LVDS[1:0]	Low voltage detection threshold voltage
00	2.0V
01	2.4V
10	2.7V
11	3.0V

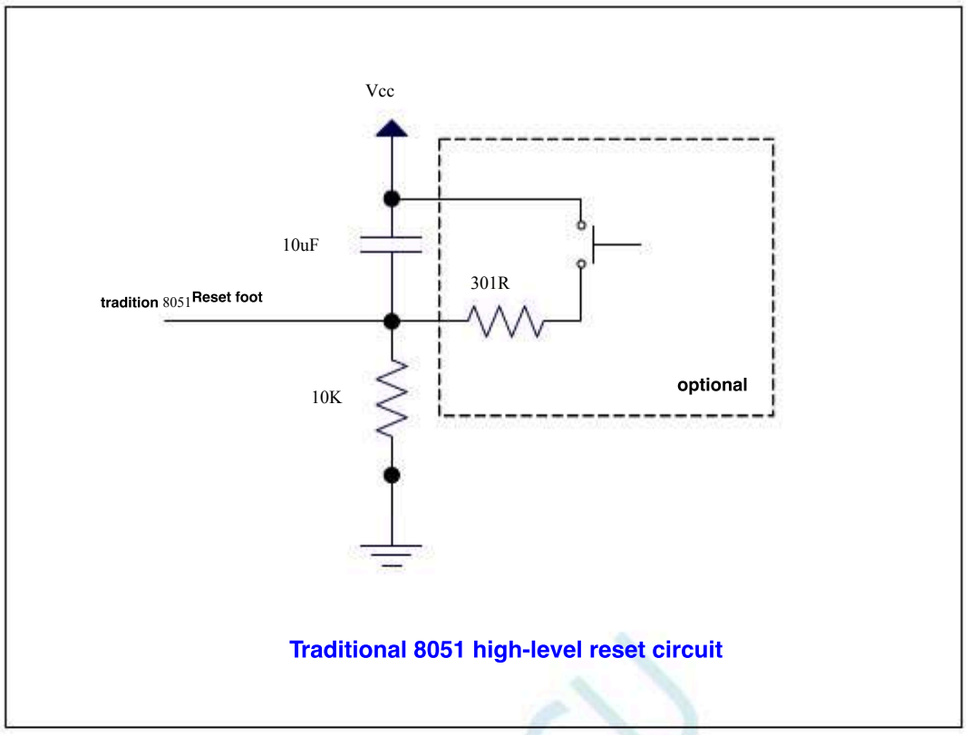
6.5.4 Low-level power-on reset reference circuit (generally not required)



6.5.5 Low-level button reset reference circuit



6.5.6 tradition 8051 High-level power-on reset reference circuit

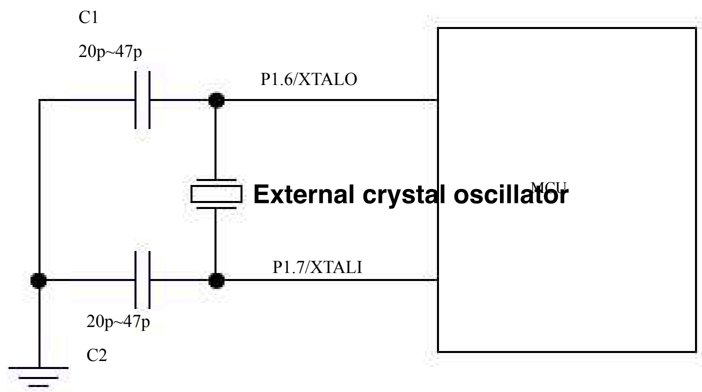


Traditional 8051 high-level reset circuit

The picture above shows the high-level reset circuit of the traditional 8051, STC12H The reset is a low-level reset, which is different from the traditional reset circuit.

6.6 External crystal oscillator and external clock circuit

6.6.1 External crystal oscillator input circuit



Note:

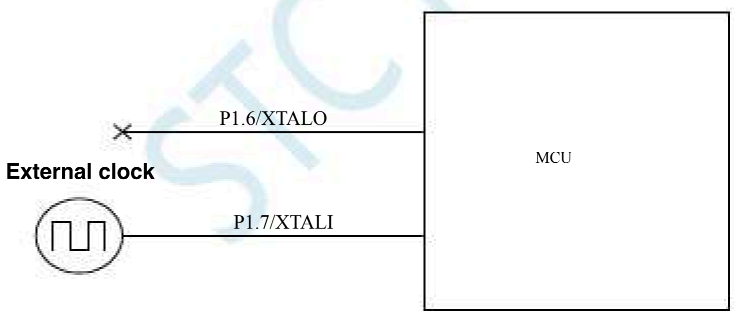
1、C1,C2 The two capacitors must not be saved. Without these two capacitors, the crystal oscillator may not vibrate.

2、RTC The clock must use an external 32768 Crystal oscillator, external capacitor required <=20pF, for example 20p/15p/10p

3, If the high-speed crystal oscillator is connected externally (for example: 24MHz), external capacitors 47pF As appropriate

4, the external crystal oscillator will add additional system power consumption, 32768 Crystal oscillator increase 1.5uA Power consumption, High-speed crystal oscillator (for example 24MHz) Will increase 2mA The above power consumption

6.6.2 External clock input circuit (P1.6 It is a high-impedance input mode and can be used as an input port)



Note: When using the internal clock, P1.6/P1.7 Can be regarded as ordinary I/O use; when P1.7 An external active clock or an external clock source is connected, when, P1.6 The port is a high-impedance input mode, which can be used as an input port. At this time, P1.6 The port mode cannot be changed. have RTC Work Can MCU, external 32768 The clock is available for P1.7 Port input.

6.7 Clock stops vibrating, Power saving mode and system power management

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000

6.7.1 Power control register (PCON)

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

LVDF : Low voltage detection flag. When the system detects a low-voltage event, the hardware automatically makes this position and to 1

This bit needs to be cleared by the user software.

POF : Power-on flag. When the hardware automatically puts this location. 1

PD : Clock stop mode, Power-down mode, Power failure mode control bit

0 : No effect

1 :The single-chip microcomputer enters the clock stop mode, Power-down mode, Power failure mode

CPU Clear to zero (Note: In clock stop mode , CPU up after waking up All peripherals of He stopped working, but the data in He is SRAM XRAM

Has remained the same)

IDL : IDL (Idle) mode control bit

0 : No effect

1 :Single chip microcomputer enters Mode, only CPU Stopped working, other peripherals are still running. The hardware is automatically cleare

Note: although LVD And the comparator can wake up the clock to stop the vibration mode, but in the clock to stop the vibration and powe LVD And the

Compare the device, otherwise the hardware system will automatically start the internal high-precision reference source, this high-precision reference source has correspon

Adjusting the line will increase the power consumption approximately MCU After entering the clock stop mode, only consumes about at operating voltage

The current, so it is not recommended to turn on the power-on 3.3V And comparator. If you really need to use it, it is recommended to turn on the powe

and wake-up timer when entering the clock stop mode, which will only increase by about Power consumption, this power consumption is generally acceptable to the system. Let the power-d

Wake up once MCU, Available after waking up LVD, Comparator Detecting the voltage of the external battery, the detection time takes

clock stop vibration, Power-saving mode, so that the increased average current is less than about time, and the overall power consumption is approximately 2.8uA

(ADC

6.8 Power-down wake-up timer

The internal power-down wake-up 15 Bit counter (by {WKTCH[6:0],WKTCL[7:0]} composition 15 Bit) 。 Used to wake up timer is a one that is in power-down mode MCU。

6.8.1 Power-down wake-up timer count register (WKTCL ， WKTCH)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
WKTCL	AAH								
WKTCH	ABH	WKTEN							

WKTEN : Enable control bit of the power-down

0 wake-up timer: Disable the power-down wake-up

1 timer : Enable the power-down wake-up timer

if The built-in dedicated timer for power-down and wake-up of the series of microcontrollers is allowed (through the software

1 Position), when MCU Enter power-down mode, After the shutdown mode, the dedicated timer for power-down and wake-up starts to count. Wh

When the values are equal, the dedicated timer for power-down and wake-up will, the program enters the power-down mode from the last setti

The next statement starts to execute down. Power-down wake-up After WKTCH and WKTCL The content in the acquisition is still being drop that, you can pass the sleep time in the power reading mode.

Please note here: if the user {WKTCH[6:0],WKTCL[7:0]} The value written in must be less than the actual count value. Such a needs to count the number of times in the register, it will be written to the register Similarly, if the user needs to count Next, then 32767

9 respond {WKTCH[6:0],WKTCL[7:0]} write 7FFEH (ie. (Count value and count value 032766 Reserved value for internal use, use Users cannot use). The internal power-down wake-up timer has its own internal clock, and the time when the power-down wake-up timer cour determined by this clock . The clock frequency of the internal, Power-down wake-up timer is approximately the sum of the regions 32KHz RAM F8H F Content (F8H Store the high byte of the frequency ，

F9H Store low bytes) to obtain the clock frequency recorded by the internal dedicated timer for power-down and wake-up at the factory .

The calculation formula of the counting time of the dedicated timer for power-down and (F8H and F9H Obtained inside wake-up is as follows: (Clock frequency of the dedicated timer for power-down and wake-up)

$$\text{Power-down wake-up timer timing time} = \frac{\text{Number of counts} \times 16 \times 10^6}{F_{wt}} \text{ (Microsecond)}$$

Hypothesis F_{wt}=32KHz，

there are: {WKTCH[6:0],WKTCL[7:0]}	Dedicated timer for power-down wake-up counting time
0 (Internal retention)	
1	$10^6 \div 32K \times 16 \times (1+1) \approx 1$ Milliseconds
9	$10^6 \div 32K \times 16 \times (1+9) \approx 5$ milliseconds
99	$10^6 \div 32K \times 16 \times (1+99) \approx 50$ milliseconds
999	$10^6 \div 32K \times 16 \times (1+999) \approx 0.5$ Seconds
4095	$10^6 \div 32K \times 16 \times (1+4095) \approx 2$ seconds
32766	$10^6 \div 32K \times 16 \times (1+32767) \approx 16$ seconds
32767 (Internal retention)	

6.9 Sample program

6.9.1 Select the system clock source

C Language code

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

#include "reg51. h"
#include "intrins. h"
#define CLKSEL (*(unsigned char volatile xdata *)0xfe00)
#define CLKDIV (*(unsigned char volatile xdata *)0xfe01)
#define HIRCCR (*(unsigned char volatile xdata *)0xfe02)
#define XOSCCR (*(unsigned char volatile xdata *)0xfe03)
#define IRC32KCR (*(unsigned char volatile xdata *)0xfe04)
sfr P_SW2 = 0xba;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;
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;
    CLKSEL = 0x00;
    P_SW2 = 0x00;                   //Choose internal RC (default)

/*
    P_SW2 = 0x80;
    XOSCCR = 0xc0;                  //Start the external crystal oscillator
    while (! (XOSCCR & 1));         //Wait for the clock to stabilize
    CLKDIV = 0x00;                  //The clock is not divided
    CLKSEL = 0x01;                  //Select an external crystal oscillator
```

```
    P_SW2 = 0x00;
*/

/*
    P_SW2 = 0x80;
    IRC32KCR = 0x80;
    while (! (IRC32KCR & 1));
    CLKDIV = 0x00;
    CLKSEL = 0x03;
    P_SW2 = 0x00;
*/
    while (1);
}
```

Start inside 32K IRC// Wait for the clock to stabilize // The clock is not divided // Choose internal 32K

Assembly code

```
; The test operating frequency is 11.0592MHz

P_SW2       DATA    0BAH

CLKSEL      EQU     0FE00H
CLKDIV      EQU     0FE01H
HIRCCR      EQU     0FE02H
XOSCCR      EQU     0FE03H
IRC32KCR    EQU     0FE04H

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,#00H                  ;Choose internal RC (default)
                MOV     DPTR,#CLKSEL
                MOVX    @DPTR,A
                MOV     P_SW2,#00H

;               MOV     P_SW2,#80H
;               MOV     A,#0C0H                 ;Start the external crystal oscillator
;               MOV     DPTR,#XOSCCR
;               MOV     @DPTR,A
;               MOVX    A,@DPTR
;               JNB     ACC.0,$-1               ;Wait for the clock to stabilize
;               CLR     A                       ;The clock is not divided
;               MOV     DPTR,#CLKDIV
                MOVX    @DPTR,A
                MOV     A,#01H                  ;Select an external crystal oscillator
;               MOV     DPTR,#CLKSEL
;               MOVX    @DPTR,A
;               MOV     P_SW2,#00H

;               MOV     P_SW2,#80H
;               MOV     A,#80H                  ;Start inside 32K IRC
;               MOV     DPTR,#IRC32KCR
;               MOVX    @DPTR,A
;               MOVX    A,@DPTR
;               JNB     ACC.0,$-1               ;Wait for the clock to stabilize
;               CLR     A                       ;The clock is not divided
;               MOV     DPTR,#CLKDIV
;               MOVX    @DPTR,A
;               MOV     A,#03H                  ;Choose internal IRC
;               MOV     DPTR,#CLKSEL
;               MOVX    @DPTR,A
;               MOV     P_SW2,#00H

                JMP     $

                END
```

6.9.2 Master clock divider output

C Language code

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

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

#define MCLKOCR (*(unsigned char volatile xdata *)0xfe05)
sfr P_SW2  = 0xba;
sfr P1M1   = 0x91;
sfr P1M0   = 0x92;
sfr P0M1   = 0x93;
sfr P0M0   = 0x94;
sfr P2M1   = 0x95;
sfr P2M0   = 0x96;
```

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

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;
//  MCLKOCR = 0x01;     // The master clock is output to P5.4
//  MCLKOCR = 0x02;     // Master clock divider output to Mouth
    MCLKOCR = 0x04;     // master clock divider output to P5.6
//  MCLKOCR = 0x84;     // master clock divider output to to mouth
    P_SW2 = 0x00;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

P_SW2       DATA    0BAH

MCLKOCR     EQU     0FE05H

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,#01H          ;The master clock is output to P5.4 Mouth
;   MOV     A,#02H          ;Master clock/2 P5.4 Mouth
    MOV     A,#04H          ;Master clock frequency division output to to mouth
;   MOV     A,#84H          ;Master clock/4 P5.4 frequency division output to
    MOV     DPTR,#MCLKOCR   frequency division output to
    MOVX    @DPTR,A
    MOV     P_SW2,#00H

    JMP     $

    END
```

6.9.3 Watchdog timer application

C Language code

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

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

sfr     WDT_CONTR =     0xc1;
sbit    P32 =           P3^2;
sfr
sfr     P1M1    =       0x91;
sfr     P1M0    =       0x92;
sfr P0M0        P0M1    =       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;
void main()     =       0xca;
{

P0M0 = 0x00;
P0M1 = 0x00;
P1M0 = 0x00;
```

```
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

//  WDT_CONTR = 0x23;          // Enable watchdog, The overflow time is approximately 1.1s
    WDT_CONTR = 0x24;          // Enable watchdog, The overflow time is approximately 2s
//  WDT_CONTR = 0x27;          // Enable watchdog, The overflow time is approximately 8s
    P32 = 0;                   // Test port

    while (1)
    {
//      WDT_CONTR = 0x33;      // Clear the watchdog, otherwise the system resets .
        WDT_CONTR = 0x34;      // Clear the watchdog, otherwise the system resets .
//      WDT_CONTR = 0x37;      // Clear the watchdog, otherwise the system resets .

        Display();             // Display module
        Scankey();             // Button scanning module

        MotorDriver();         // Motor drive module

    }
}
```

Assembly code

```
;  The test operating frequency is 11.0592MHz
;
WDT_CONTR DATA  0C1H

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     WDT_CONTR,#23H  ; Enable watchdog,The overflow time is approximately 1.1s
        MOV     WDT_CONTR,#24H  ; Enable watchdog,The overflow time is approximately 2s
;       MOV     WDT_CONTR,#27H  ; Enable watchdog,The overflow time is approximately 8s
        CLR     P3.2            ;Test port
LOOP:
;       MOV     WDT_CONTR,#33H  ; Clear the watchdog, otherwise the system resets ,
        MOV     WDT_CONTR,#34H  ; Clear the watchdog, otherwise the system resets ,
;       MOV     WDT_CONTR,#37H  ; Clear the watchdog, otherwise the system resets ,

        LCALL   DISPLAY         ;Display module
        LCALL   SCANKEY         ;Button scanning module
        LCALL   MOTORDRIVER     ;Motor drive module
        JMP     LOOP

        END
```

6.9.4 Soft reset to achieve custom download

C Language code

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

#include "reg51. h"
#include "intrins. h"
sfr IAP_CONTR = 0xc7;
sbit P32 = P3^2;
sbit P33 = P3^3;
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;
    P32 = 1;                                    //Test port
    P33 = 1;                                    //Test port
    while (1)
    {
        if (! P32 && ! P33) {                   //Checked to P3.2 and P3.3 Reset to at the same time
            IAP_CONTR |= 0x60;
        }
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

IAP_CONTR   DATA    0C7H

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    P3.2
            SETB    P3.3
```

```
LOOP:
        JB      P3.2,LOOP
        JB      P3.3,LOOP
        MOV     IAP_CONTR,#60H          ;Checked to P3.2 and P3.3 0 Reset to at the same time
        JMP     $

        END
```

6.9.5 Low voltage detection

C Language code

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

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

sfr RSTCFG = 0xff;
#define ENLVR 0x40 //RSTCFG.6
#define LVD2V0 0x00 //LVD@2.0V
#define LVD2V4 0x01 //LVD@2.4V
#define LVD2V7 0x02 //LVD@2.7V
#define LVD3V0 0x03 //LVD@3.0V
sbit ELVD = IE^6;
#define LVDF 0x20 //PCON.5
sbit P32 = P3^2;
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 Lvd_Isr() interrupt 6
{
    PCON &= ~LVDF;      //Clear interrupt sign
    P32 = ~P32;         //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;
    PCON &= ~LVDF;
//  RSTCFG = ENLVR | LVD3V0;
    RSTCFG = LVD3V0;
    ELVD = 1;
    EA = 1;

    while (1);
}
```

//Test port
//Enable 3.0V Low voltage reset, LVD interrupt
//Enable Low voltage interruption when not generated 3.0V
//Enable interrupt

Assembly code

;The test operating frequency is 11.0592MHz

```
RSTCFG      DATA    0FFH
ENLVR       EQU     40H         ;RSTCFG. 6
LVD2V0      EQU     00H         ;LVD@2.0V
LVD2V4      EQU     01H         ;LVD@2.4V
LVD2V7      EQU     02H         ;LVD@2.7V
LVD3V0      EQU     03H         ;LVD@3.0V

ELVD        BIT     IE. 6
LVDF        EQU     20H         ;PCON. 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     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF
            CPL     P3.2
            RETI
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
```

;Clear interrupt sign ;Test port

```
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        ANL     PCON,#NOT LVDF      ;Need to be cleared after power-up
;       MOV     RSTCFG,#ENLVR | LVD3V0  ;Enable low voltage reset function, 3.0V
        MOV     RSTCFG,#LVD3V0      ;Enable low voltage reset function, 3.0V
        SETB    ELVD                ;Enable low voltage interrupt
        SETB    EA
        JMP     $

        END
```

6.9.6 Power saving mode

C Language code

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

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

#define IDL     0x01    //PCON.0
#define PD      0x02    //PCON.1
sbit P34 = P3^4;
sbit P35 = P3^5;
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 INT0_Isr() interrupt 0
{
    P34 = ~P34;                     //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;
    EX0 = 1;                    //Enable INT0 interrupt is used to wake up MCU
    EA = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    PCON = IDL;                 //MCU enter IDLE pattern
//  PCON = PD;                  //MCU Enter power-down mode
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    P35 = 0;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

IDL     EQU     01H             ;PCON. 0
PD      EQU     02H             ;PCON. 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     0003H
        LJMP    INT0ISR

        ORG     0100H
INT0ISR:
        CPL     P3.4            ;Test port
        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    EX0             ;Enable INT0 Interrupt is used to wake up
        SETB    EA
        NOP
        NOP
;       MOV     PCON,#IDL       ;MCU enter IDLE pattern
        MOV     PCON,#PD        ;MCU Enter power-down mode
        NOP
        NOP
        NOP
        NOP
        CLR     P3.5            ;Test port
        JMP     $

        END
```

6.9.7 use INT0/INT1/INT2/INT3/INT4 Pin interrupt wake-up power saving mode

C Language code

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

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

sfr     INTCLKO =   0x8f;
#define EX2     0x10
#define EX3     0x20
#define EX4     0x40
sbit P10
sbit P11        =   P1^0;
sfr P1M1        =   P1^1;
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;


void INT0_Isr() interrupt 0
{
```

```
        P10 = ! P10;                                    //Test port
}

void INT1_Isr() interrupt 2
{
        P10 = ! P10;                                    //Test port
}

void INT2_Isr() interrupt 10
{
        P10 = ! P10;                                    //Test port
}

void INT3_Isr() interrupt 11
{
        P10 = ! P10;                                    //Test port
}

void INT4_Isr() interrupt 16
{
        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;
//      IT0 = 0;                                        //Enable INT0 Rising and falling edge interrupts
        IT0 = 1;                                        //Enable INT0 Falling edge interrupt
        EX0 = 1;                                        //Enable INT0 interrupt
//      IT1 = 0;                                        //Enable INT1 Rising and falling edge interrupts
        IT1 = 1;                                        //Enable INT1 Falling edge interrupt
        EX1 = 1;                                        //Enable INT1 interrupt
        INTCLKO = EX2;                                  //Enable INT2 Falling edge interrupt
        INTCLKO |= EX3;                                 //Enable INT3 Falling edge interrupt
        INTCLKO |= EX4;                                 //Enable INT4 Falling edge interrupt
        EA = 1;
        PCON = 0x02;                                    //Enter power-down mode
        _nop_();                                        //MCU This statement will be executed first
        _nop_();                                        //After the power-down mode is awakened
        _nop_();                                        //Then enter the interrupt service program
        _nop_();
```

```
while (1)
{
    P11 = ~P11;
}
}
```

Assembly code

The test operating frequency is 11.0592MHz

```
INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

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     0003H
            LJMP    INT0ISR
            ORG     0013H
            LJMP    INT1ISR
            ORG     0053H
            LJMP    INT2ISR
            ORG     005BH
            LJMP    INT3ISR
            ORG     0083H
            LJMP    INT4ISR

            ORG     0100H
INT0ISR:
            CPL     P1.0            ;Test port
            RETI
INT1ISR:
            CPL     P1.0            ;Test port
            RETI
INT2ISR:
            CPL     P1.0            ;Test port
            RETI
INT3ISR:
            CPL     P1.0            ;Test port
            RETI
INT4ISR:
            CPL     P1.0            ;Test port
            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     IT0             ;Enable INT0 Rising and
;       SETB    IT0             ;Enable Falling edge interrupt
        SETB    EX0             ;Enable INT0 interrupt

        CLR     IT1             ;Enable INT1 Rising and
;       SETB    IT1             ;Enable Falling edge interrupt
        SETB    EX1             ;Enable INT1 interrupt

        MOV     INTCLKO,#EX2    ;Enable Falling edge interrupt INT2
        ORL     INTCLKO,#EX3    ;Enable Falling edge interrupt
        ORL     INTCLKO,#EX4    ;Enable Falling edge interrupt

        SETB    EA

        MOV     PCON,#02H       Enter power-down mode
        NOP                     ;;MCU After the power-down mode is awakened, this statement will be executed first
                                ; Then enter the interrupt service program
        NOP
        NOP
LOOP:   NOP
        CPL     P1.1
        JMP     LOOP
        END
```

6.9.8 use T0/T1/T2/T3/T4 Pin interrupt wake-up power saving mode

C Language code

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

#include "reg51. h"
#include "intrins. h"
sfr T2L  = 0xd7;
sfr T2H  = 0xd6;
sfr T3L  = 0xd5;
sfr T3H  = 0xd4;
sfr T4L  = 0xd3;
sfr T4H  = 0xd2;
```

```
sfr     T4T3M   =   0xd1;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2     0x04
#define ET3     0x20
#define ET4     0x40
sfr     AUXINTIF    =   0xef;
#define T2IF    0x01
#define T3IF    0x02
#define T4IF    0x04

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

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 TM0_Isr() interrupt 1
{
    P10 = ! P10;                                    //Test port
}

void TM1_Isr() interrupt 3
{
    P10 = ! P10;                                    //Test port
}

void TM2_Isr() interrupt 12
{
    P10 = ! P10;                                    //Test port
}

void TM3_Isr() interrupt 19
{
    P10 = ! P10;                                    //Test port
}

void TM4_Isr() interrupt 20
{
    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;
    TMOD = 0x00;                    //65536-11.0592M/12/1000
    TL0 = 0x66;
    TH0 = 0xfc;                     //Start the timer
    TR0 = 1;                        //Enable timer interrupt
    ET0 = 1;
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;                     //Start the timer
    TR1 = 1;                        //Enable timer interrupt
    ET1 = 1;
    T2L = 0x66;                     //65536-11.0592M/12/1000
    T2H = 0xfc;                     //Start the timer
    AUXR = 0x10;                    //Enable timer interrupt
    IE2 = ET2;
    T3L = 0x66;                     //65536-11.0592M/12/1000
    T3H = 0xfc;                     //Start the timer
    T4T3M = 0x08;                   //Enable timer interrupt
    IE2 |= ET3;
    T4L = 0x66;                     //65536-11.0592M/12/1000
    T4H = 0xfc;                     //Start the timer
    T4T3M |= 0x80;                  //Enable timer interrupt
    IE2 |= ET4;
    EA = 1;
    PCON = 0x02;                    //MCU Enter power-down mode
    _nop_();                        // Will not enter the interrupt service program immediately after power-down and w
    _nop_();                        // Instead, it waits until the timer overflows before entering the interrupt service pr
    _nop_();
    _nop_();
    while (1)
    {

        P11 = ~P11;
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

```
T4T3M   DATA    0D1H
AUXR    DATA    8EH

IE2     DATA    0AFH
ET2     EQU     04H
ET3     EQU     20H
ET4     EQU     40H

AUXINT1F    DATA    0EFH
T2IF    EQU     01H
T3IF    EQU     02H
T4IF    EQU     04H

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

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    TM0ISR
        ORG     001BH
        LJMP    TM1ISR
        ORG     0063H
        LJMP    TM2ISR
        ORG     009BH
        LJMP    TM3ISR
        ORG     00A3H
        LJMP    TM4ISR

        ORG     0100H
TM0ISR:
        CPL     P1.0            ;Test port
        RETI
TM1ISR:
        CPL     P1.0            ;Test port
        RETI
TM2ISR:
        CPL     P1.0            ;Test port
        RETI
TM3ISR:
        CPL     P1.0            ;Test port
        RETI
TM4ISR:
        CPL     P1.0            ;Test port
        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     TMOD,#00H
        MOV     TL0,#66H                ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB    TR0                     ;Start the timer
        SETB    ET0                     ;Enable timer interrupt

        MOV     TL1,#66H                ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1                     ;Start the timer
        SETB    ET1                     ;Enable timer interrupt

        MOV     T2L,#66H                ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H               ;Start the timer
        MOV     IE2,#ET2                ;Enable timer interrupt

        MOV     T3L,#66H                ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#08H              ;Start the timer
        ORL     IE2,#ET3                ;Enable timer interrupt

        MOV     T4L,#66H                ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        ORL     T4T3M,#80H              ;Start the timer
        ORL     IE2,#ET4                ;Enable timer interrupt

        SETB    EA

        MOV     PCON,#02H               ;MCU Enter power-down mode
        NOP                             ; Will not enter the interrupt service program immediately after power-down and wa
                                        ; Instead, it waits until the timer overflows before entering the interrupt service pro
        NOP
        NOP
LOOP:   NOP
        CPL     P1.1
        JMP     LOOP

        END
```

6.9.9 use RxD/RxD2 Pin interrupt wake-up power saving mode

C Language code

//The test operating frequency is

```
#include "reg51. h"
#include "intrins. h"
sfr     IE2     =   0xaf;
#define ES2     0x01
#define ES3     0x08
#define ES4     0x10
sfr P_SW1       =   0xa2;
sfr P_SW2       =   0xba;
sbit P11        =   P1^1;
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 UART1_Isr() interrupt 4
{
}
void UART2_Isr() interrupt 8
{
}
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_SW1 = 0x00;
//  P_SW1 = 0x40;
//  P_SW1 = 0x80;
//  P_SW1 = 0xc0;
    P_SW2 = 0x00;
    P_SW2 = 0x01;
//
```

```
//RXD     Wake up on the falling edge
//RXD_2   Wake up on the falling edge Wake up on
//RXD_3
//RXD_4   the falling edge Wake up on the falling edge

//RXD2    Wake up on the falling edge
//RXD2_2  Wake up on the falling edge
```

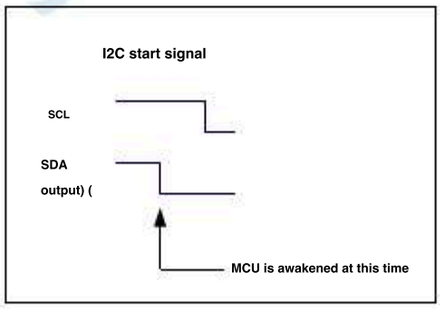
```
        ES = 1;                                 //Enable serial port interrupt
        IE2 = ES2;                              //Enable serial port interrupt
        EA = 1;

        PCON = 0x02;                            //MCU Enter power-down mode
        _nop_();                                // Will not enter the interrupt service program after power-down and wake-up
        _nop_();
        _nop_();
        _nop_();

while (1)
{               P11 = ~P11;
        }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

IE2     DATA    0AFH
ES2     EQU     01H

P_SW1   DATA    0A2H
P_SW2   DATA    0BAH

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     0023H
        LJMP    UART1ISR
        ORG     0043H
        LJMP    UART2ISR

        ORG     0100H
UART1ISR:
        RETI
UART2ISR:
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
```

```
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW1,#00H      ;RXD    Wake up on the falling edge
;       MOV     P_SW1,#40H      ;RXD_2  Wake up on the falling edge Wake up on
;       MOV     P_SW1,#80H      ;RXD_3
;       MOV     P_SW1,#0C0H     ;RXD_4  the falling edge Wake up on the falling edge

        MOV     P_SW2,#00H      ;RXD2   Wake up on the falling edge
;       MOV     P_SW2,#01H      ;RXD2_2 Wake up on the falling edge

        SETB    ES              ;Enable serial port interrupt
        MOV     IE2,#ES2        ;Enable serial port interrupt
        SETB    EA

        MOV     PCON,#02H       ;MCU    Enter power-down mode
        NOP                     ; Will not enter the interrupt service program after power-down and wake-up
        NOP
        NOP
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

6.9.10 use I2C of SDA Foot wake up MCU Power saving 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 I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
```

```
sbit        P11         =   P1^1;

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 i2c_isr() interrupt 24
{
    P_SW2 |= 0x80;
    I2CSLST &= ~0x40;
}

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 = 0x00;
//  P_SW2 = 0x10;
//  P_SW2 = 0x30;
    P_SW2 |= 0x80;
    I2CCFG = 0x80;
    I2CSLCR = 0x40;
    EA = 1;
    PCON = 0x02;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    while (1)
    {

        P11 = ~P11;
    }
}
```

//SDA Wake up on the falling edge
//SDA_2 Wake up on the falling edge
//SDA_4 Wake up on the falling edge

Slave mode of the module
// I2C Enable//
Enable start signal interrupt
//MCU Enter power-down mode
// Will not enter the interrupt service program after power-down and wake-up

Assembly code

; The test operating frequency is

```
P_SW2           DATA    0BAH

I2CCFG          XDATA   0FE80H
I2CSLCR         XDATA   0FE83H
I2CSLST         XDATA   0FE84H

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

                ORG     0000H
                LJMP    MAIN
                ORG     00C3H
                LJMP    I2CISR

                ORG     0100H
I2CISR:
                PUSH    ACC
                PUSH    DPH
                PUSH    DPL
                ORL     PSW2,#80H
                MOV     DPTR,#I2CSLST
                MOVX    A,@DPTR
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                POP     DPL
                POP     DPH
                POP     ACC
                RETI

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

                MOV     P_SW2,#00H      ;SDA    Wake up on the falling edge
//              MOV     P_SW2,#10H      ;SDA_2  Wake up on the falling edge
//              MOV     P_SW2,#30H      ;SDA_4  Wake up on the falling edge
```

```
                ORL     P_SW2,#80H
                MOV     DPTR,#I2CCFG
                MOV     A,#80H
                MOVX    @DPTR,A                 ; Enable I2C Slave mode of the module
                MOV     DPTR,#I2CSLCR
                MOV     A,#40H                  ; Enable start signal interrupt
                SETB    EA

                MOV     PCON,#02H               ;MCU Enter power-down mode
                NOP                             ; Will not enter the interrupt service program after power-down and wake-up
                NOP
                NOP
                NOP
                NOP
LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

6.9.11 Use the power-down wake-up timer to wake up the power-saving mode

C Language code

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

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

sfr WKTCL  = 0xaa;
sfr WKTCH  = 0xab;;
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 P11   = P1^1;

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

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
    WKTCL = 0xff;                   // Set the power-down wake-up clock to be about seconds 1
    WKTCH = 0x87;
    while (1)
    {
        _nop_();
        _nop_();
        PCON = 0x02;                //MCU Enter power-down mode
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        P11 = ~P11;
    }
}
```

Assembly code

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

WKTCL   DATA    0AAH
WKTCH   DATA    0ABH

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     WKTCL,#0FFH     ; Set the power-down wake-up clock to be about seconds 1
                MOV     WKTCH,#87H

LOOP:
                NOP
                NOP
                MOV     PCON,#02H       ;MCU Enter power-down mode
                NOP
                NOP
                NOP
                NOP
                NOP
                CPL     P1.1
                JMP     LOOP
                END
```

Interrupt the wake-up power saving mode, it is recommended to use the p

LVD In the power-saving mode when the clock is stopped, it is not recommended to start the comparator, otherwise the hardware syste LVD High precision 1.19V
Reference source, this high-precision reference source has corresponding anti-temperature drift and adjustment circuit, which will increase ap The power consumption will increase
After entering the clock stop mode , 3.3V Only consumes about The operating voltage is not recommended to turn on when entering the clock stop m 2mA
comparator. If you really need to use it, it is recommended to turn on the power-down wake-up timer. The power-down wake-up timer will only 1uA Power consumption, this
the power consumption by about, which is generally acceptable for the system. Let the power-down wake-up timer wake up every second Comparator LVD RTC MCU
Detect the voltage of the external battery, the detection work takes about timer ms Then enter the clock to stop vibrating Power-saving mode, so that the increased average cu 1uA
The overall power consumption is approximately 2.8uA (0.4uA + 1.4uA + 1uA).

Language codeC

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

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

sfr RSTCFG      =   0xff;
#define ENLVR       0x40        //RSTCFG. 6
#define LVD2V0      0x00        //LVD@2.0V
#define LVD2V4      0x01        //LVD@2.4V
#define LVD2V7      0x02        //LVD@2.7V
#define LVD3V0      0x03        //LVD@3.0V
sbit ELVD       =   IE^6;
#define LVDF        0x20        //PCON. 5
sbit P10        =   P1^0;
sbit P11        =   P1^1;
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 LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;          //Clear interrupt sign
    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;
    PCON &= ~LVDF;          //The interrupt flag needs to be cleared for power-up

    RSTCFG = LVD3V0;        //Set up LVD The voltage is 3.0V
    ELVD = 1;               //Enable LVD interrupt
    EA = 1;

    PCON = 0x02;            //MCU Enter power-down mode
    _nop_();                //Enter the interrupt service program immediately after power-down and wake-up
    _nop_();
    _nop_();
    _nop_();

while (1)
{   P11 = ~P11;
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H         ;RSTCFG.6
LVD2V0      EQU     00H         ;LVD@2.0V
LVD2V4      EQU     01H         ;LVD@2.4V
LVD2V7      EQU     02H         ;LVD@2.7V
LVD3V0      EQU     03H         ;LVD@3.0V

ELVD        BIT     IE.6
LVDF        EQU     20H         ;PCON.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     0033H
                LJMP    LVDISR

                ORG     0100H
LVDISR:
                ANL     PCON,#NOT LVDF  ;Clear interrupt sign
                CPL     P1.0            ;Test port
                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

                ANL     PCON,#NOT LVDF  ; The interrupt flag needs to be cleared for power-up
                MOV     RSTCFG,# LVD3V0 ;Set up LVD The voltage is 3.0V
                SETB    ELVD            ;Enable LVD interrupt
                SETB    EA

                MOV     PCON,#02H       ;MCU Enter power-down mode
                NOP                     ; Enter the interrupt service program immediately after power-down and wake-up
                NOP
                NOP
                NOP
                NOP
LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

6.9.13 The comparator interrupts the wake-up power-saving mode, it is recommended to use the power-dow

When the clock stops vibrating in power saving mode, it is recommended to use the internal 1.27V Analog comparator, otherwise the hardware system will automatically start the High precision internal Reference source, this high-precision reference source has corresponding anti-temperature drift and The power consumption, will increase ap After entering the clock stop mode , 3.3V Only consumes about The operating voltage is not recommended to turn on when entering the clock stop m comparator. If you really need to use it, it is recommended to turn on the power-down wake-up timer. The power-down wake-up timer will only the power consumption by about, which is generally acceptable for the system. Let the power-down wake-up timer wake up every second MCU Detect the voltage of the external battery, the detection work takes about timer, Then enter the clock to stop vibrating Power-saving mode, so that the increased average cu The overall power consumption is approximately 2.8uA (0.4uA + 1.4uA + 1uA).

Power consumption, this Comparator

C Language code

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

#include "reg51. h"
#include "intrins. h"
sfr
        CMPCR1          =       0xe6;
sfr CMPCR2
                        =       0xe7;
sbit P10
sbit P11
                        =       P1^0;
sfr P1M1                =       P1^1;
sfr P1M0
sfr P0M1                =       0x91;
sfr P0M0                =       0x92;
sfr P2M1                =       0x93;
                        =       0x94;
sfr P2M0                =       0x95;
sfr P3M1                =       0x96;
sfr P3M0                =       0xb1;
sfr P4M1                =       0xb2;
sfr P4M0                =       0xb3;
sfr P5M1                =       0xb4;
sfr P5M0                =       0xc9;
                        =       0xca;


void CMP_Isr() interrupt 21
{
        CMPCR1 &= ~0x40;                        //Clear interrupt sign
        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;
```

```
    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //Enable comparator module
    CMPCR1 |= 0x30;                 // Enable comparator edge interrupt
    CMPCR1 &= ~0x08;                //P3.6 CMP+ The input pin
    CMPCR1 |= 0x04;                 //P3.7 CMP- Input pin is
    CMPCR1 |= 0x02;                 //Enable comparator output
    EA = 1;


    PCON = 0x02;                    //MCU Enter power-down mode
    _nop_();                        // Enter the interrupt service program immediately after power-down and wake-up
    _nop_();
    _nop_();
    _nop_();


while (1)

{               P11 = ~P11;

    }

}
```

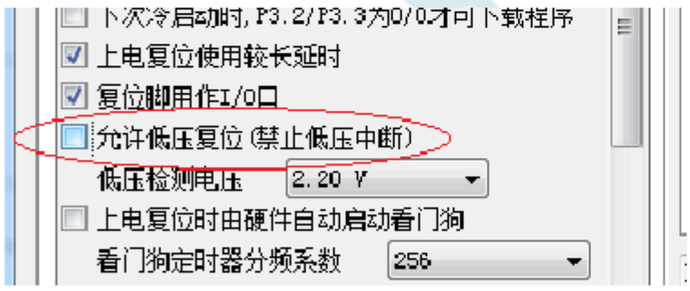
Assembly code

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

CMPCR1          DATA        0E6H
CMPCR2          DATA        0E7H

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

                ORG         0000H
                LJMP        MAIN
                ORG         00ABH
                LJMP        CMPISR

                ORG         0100H
CMPISR:
                ANL         CMPCR1,#NOT 40H ;Clear interrupt sign
                CPL         P1.0            ;Test port
                RETI

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
```

```
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CMPCR2,#00H
        MOV     CMPCR1,#80H             ;Enable the comparator module to enable
        ORL     CMPCR1,#30H             the comparator edge interrupt
        ANL     CMPCR1,#NOT 08H         ;P3.6 CMP+ The input pin
        ORL     CMPCR1,#04H             is P3.7 CMP- Input pin
        ORL     CMPCR1,#02H             ;Enable comparator output
        SETB    EA

        MOV     PCON,#02H               ;MCU Enter power-down mode
        NOP                             ; Enter the interrupt service program immediately after power-down and wake-up
        NOP
        NOP
        NOP

LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

use 6.9.14 Function detection of operating voltage (battery voltage) LVD

If you need to use LVD The function detects the battery voltage, then ISP When downloading, you need to remove the low-voltage reset function. As shown in Bit (low voltage interruption is prohibited)
"The check mark of the hardware option needs to be removed



C Language code

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

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

#define FOSC            11059200UL
#define T1MS            (65536 - FOSC/4/100)

sfr RSTCFG              = 0xff;

#define LVD2V0          0x00            //LVD@2.0V
#define LVD2V4          0x01            //LVD@2.4V
#define LVD2V7          0x02            //LVD@2.7V
```

```
#define		LVD3V0			0x03			//LVD@3.0V

#define		LVDF			0x20			//PCON. 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 delay()
{
	int i;

	for (i=0; i<100; i++)
	{
_nop_();
_nop_();
_nop_();
_nop_();
	}
}
void main()
{
	unsigned char power;

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

	PCON &= ~LVDF;
	RSTCFG = LVD3V0;
	while (1)
	{
		power = 0x0f;

RSTCFG = LVD3V0;
delay();
PCON &= ~LVDF;
delay();
if (PCON & LVDF)
```

```
    {
        power >>= 1;
        RSTCFG = LVD2V7;
        delay();
        PCON &= ~LVDF;
        delay();
        if (PCON & LVDF)
        {
            power >>= 1;
            RSTCFG = LVD2V4;
            delay();
            PCON &= ~LVDF;
            delay();
            if (PCON & LVDF)
            {
                power >>= 1;
                RSTCFG = LVD2V2;
                delay();
                PCON &= ~LVDF;
                delay();
                if (PCON & LVDF)
                {
                    power >>= 1;
                }
            }
        }
    }
    RSTCFG = LVD3V0;                    //P2.3~P2.0 Display battery level
    P2 = ~power;
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

RSTCFG      DATA    0FFH
LVD2V0      EQU     00H         ;LVD@2.0V
LVD2V4      EQU     01H         ;LVD@2.4V
LVD2V7      EQU     02H         ;LVD@2.7V
LVD3V0      EQU     03H         ;LVD@3.0V

LVDF        EQU     20H         ;PCON.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
```

```
        JMP     MAIN

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

        ANL     PCON,#NOT LVDF
        MOV     RSTCFG,#LVD3V0
LOOP:
        MOV     B,#0FH

        MOV     RSTCFG,#LVD3V0
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A

        MOV     RSTCFG,#LVD2V7
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A

        MOV     RSTCFG,#LVD2V4
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A
```

```
        MOV     RSTCFG,#LVD2V2
        CALL    DELAY
        ANL     PCON,#NOT LVDF
        CALL    DELAY
        MOV     A,PCON
        ANL     A,#LVDF
        JZ      SKIP
        MOV     A,B
        CLR     C
        RRC     A
        MOV     B,A

SKIP:
        MOV     A,B
        CPL     A
        MOV     P2,A            ;P2.3~P2.0      Display battery level
        JMP     LOOP

DELAY:
        MOV     R0,#100
NEXT:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R0,NEXT
        RET
        END
```

7 memory

STC12H The program memory and data memory of the series of microcontrollers are individually addressed. Because access to exter
The bus of the memory, all program memory of all microcontrollers is on-chip Memory, external program memory cannot be accessed.

STC12H A large-capacity data memory is integrated into the series of microcontrollers. The internal data memory of the series of microcontrollers STC12H
Physically and logically divided into two address spaces Internal Bytes and internal extensions inside of which RAM The height of byte
data memory and special function registers (SFRs). The addresses overlap, and different addressing methods are used to distinguish them in actu

7.1 Program memory

The program memory is used to store information such as user programs, data, and tables.

STC12H1K08 series monolithic internal integration of 28K Byte Flash Program memory (ROM).

After the microcontroller is reset, the content of the program counter (PC) is 0000H, and the program is executed from the 0000H unit. In addition, th
address of the interrupt service program (also known as the interrupt vector) is also located in the program memory cell. In the program memory, each inte
interrupt occurs and a response is received, the microcontroller will automatically jump to the corresponding interrupt entry address to execute the progra
program with external interrupt 0 (INT0) is 0003H, the entry address of the timer/counter 0 (TIMER0) interrupt service program is 000BH, the entry address
interrupt 1 (INT1) is 0013H, and the interrupt service of timer/counter 1 (TIMER1) is 000BH.The entry address of the program is 001BH, etc.
For more information about the entry address (interrupt vector) of the interrupt service program, please refer to the Interrupt Introduction section.

Since the interval between the adjacent interrupt entry addresses is only 8 bytes, it is generally impossible to save the complete interrupt service pro
unconditional transfer instruction in the address area of the interrupt response, pointing to the space where the interrupt service program is actually store

The STC12H series microcontrollers all contain Flash data memory (EEPROM). Read/write data in bytes, erase in 512 bytes per page,
and can be repeatedly programmed online to erase more than 100,000 times.It improves the flexibility and convenience of use.

7.2 Data memory

STC12H Internally integrated with a series of Can be used to store intermediate results and process data of program execution.

Single chip microcomputer series	Internal direct access RAM (DATA)	Internal indirect access RAM (IDATA)	Internal expansion RAM (XDATA)
STC12H1K08 series	128 byte	128 byte	1024 byte

7.2.1 inside RAM

Total internal RAM 256 Parts: low bytes, can be divided into 2 Byte and height 128 RAM Byte data storage, Low medium, Device and tradition Compatible, can be directly addressable or indirectly addressable 80C52 high extended, high 128 byte RAM)

It shares the same logical address as the special function 80H~FFH , But they are physically independent, and they are used through different register area, and both are distinguished by addressing. High byte 128 RAM It can only be addressed indirectly, and the special function register area can only be directly addressed.

inside RAM The structure is shown in the figure below :

byte Also known as universal Universal RAM The area can be divided into working register bank area, bit addressable ar low 128 Area and stack area. The address of the working register group area is from Groups, each group is called a register byte unit, RAM The working register of the bits, the numbers are all R0 ~ R7 But it belongs to a different physical space. By using the working register bank , Groups, each group contains RAM 8 8 Is a commonly used register that provides Groups are often not enough. The program status word PSW In the register Can improve the computing speed. RS1 and RS0 The combination determines the currently used working register bank, see below R0~R7 Introduction of the register.

7.2.2 Program status register (PSW)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	CY	AC	F0	**RS1**	**RS0**	OV	F1	P

CY Enter/Borrowing sign.

AC Auxiliary group advance/Borrowing sign.

F0: User flag.

0

RS1 , RS0 : Working register selection bit

RS1	RS0	Working register bank (R0~R7)
0	0	07H) 0 Group 1 (00H ~
0	1	Group 1 (1 08H ~
1	0	2 Group 1 (10H ~
1	1	Group 1 (3 18H ~ 0FH) 17H) 1FH)

OV: Overflow standard Ambition.

F1: User flag.

1

P : Parity check flag.

The address of the bit-addressable area is from The unit can be like an ordinary byte unit accessed by bytes the same way , The byte unit 20H~2FH

You can also access any one of the units separately, for a total of Bits, the corresponding logical bit address range is The bit address range is

00H~7FH, inside RAM low 128 The address of the byte is also , From the outside, the addresses of the two are the same, in fact, the two have this

Qualitative difference; the bit address points to a bit, while the byte address points to a byte unit, which is distinguished by using different instructions in

inside RAM in 30H~FFH And stack area. One unsigned Stack pointer of bits (SP), Used to point to the stack area. single

After the chip machine is reset, the SP 07H , Points to the working register bank in 0 R7; Therefore, the user initializer should deal with Set up the beginning

stack pointer value is generally reset 80H Fully reset units are appropriate.

Bit dedicated register. It indicates that the internal stack pointer is addressed in the Resident top of the stack. After the system is reset , initialize

bit 07H , So that the stack actually starts from the unit, consider 08H 08H~1FH SP RAM

When using these areas, it is best The value is changed to Or a larger value is appropriate. The stack of the series of microcontrollers is growing upwards, and the c

After pressing into the stack ,

SP The content increases.

7.2.3 Internal expansion RAM ， XRAM ， XDATA

STC12H In addition to the on-chip integration of a series of microcontrollers addition, the internal byte is also integrated RAM The extension of also Access to internal exhibition RAM Methods and traditional microcontrollers to access external The method is the same, but it does not use Data bus and high octet ground)Address bus P2 mouth (Low octet address bus), and RD WR and ALE Wait for the signal on the port.

In assembly language, internal extensions pass RAM MOVX Instruction access ，

```
MOVX    A,@DPTR
MOVX    @DPTR,A
MOVX    A,@Ri
MOVX    @Ri,A
```

in C Just declare the storage type. such as ：

xdata/pdata In

the language, it can be used unsigned char xdata i;

unsigned int pdata j;

That is to say, note ： Bytes, in the low The subscription variable in the Although type, the compiler will XDATA of pdata C Area, and use MOVX @Ri,A and MOVX A@Ri automatically assign the variable for access.

xdata 0000H~00FFH

MCU internal expansion RAM Whether it can be accessed, subject to the auxiliary register Bit control.

7.2.4 Auxiliary register (AUXR)

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

EXTRAM: Extended RAM Access control

0: Access to internal extensions XDATA

1: Internal expansion XDATA Disabled.

7.2.5 External extension

XDATA RAM ' XRAM '

series package STC12H 40 And above microcontrollers With extension

The number of pins is 64KB The ability of external data memory. Number of visits to the outsi

According to the memory period , WR/RD/ALE The signal must be effective. A new control external has been added to the series STC12H Bytes data bus speed

The special function register BUS_SPEED, The description is as follows :

7.2.6 Bus speed control register (BUS_SPEED)

symbol	address		B6 B7	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]					SPEED[2:0]		

RD/WR RW_S[1:0]: Control line selection bit

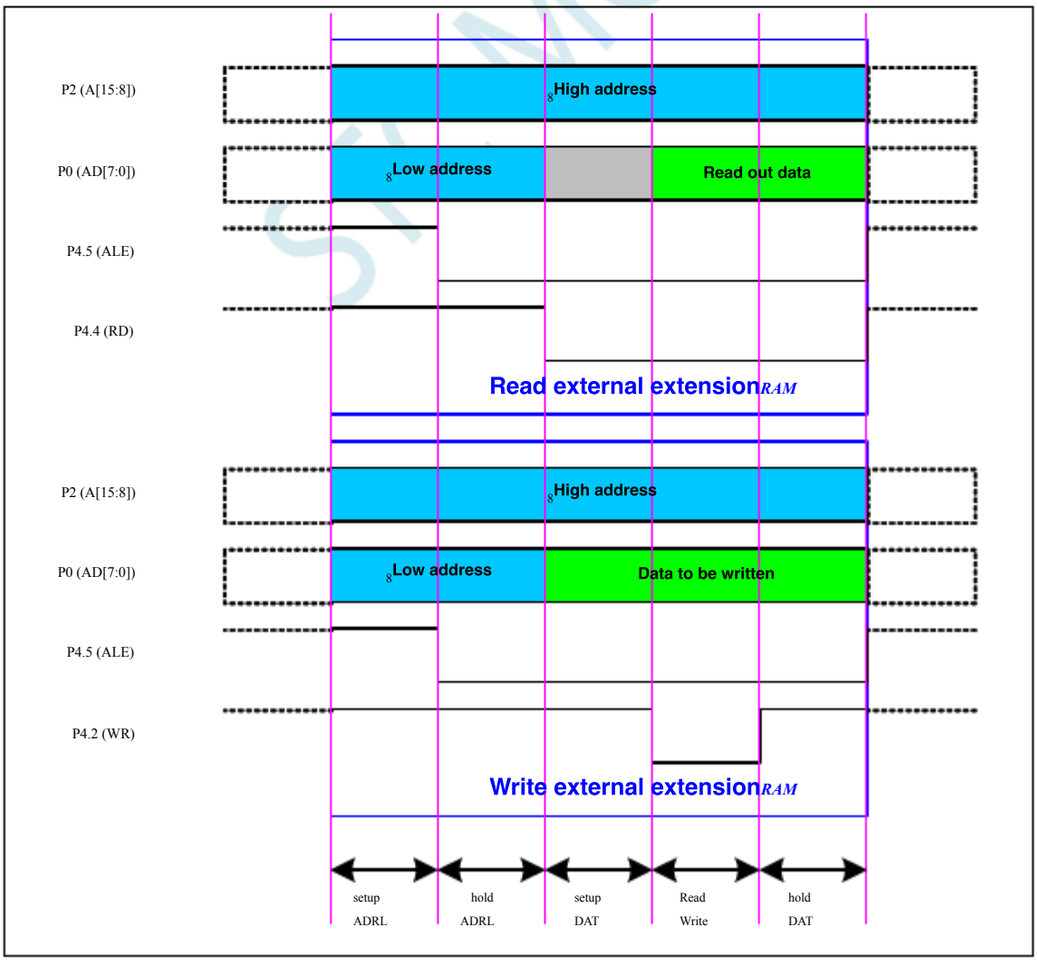
for RD P4.0 : P4.2 for WR

x1: Reserved

SPEED[2:0] : BUS read and write speed control (preparation time and retention time of control signal and data signal when reading and writing data)

instruction		Number of clocks	
		Access to internal extension RAM	Access to external RAM
MOVX	A,@Ri	3	extensions 3+5*
MOVX	@Ri,A	3	(SPEED+1) 3+5* (SPEED+1)
MOVX	A,@DPTR	2	2+5* (SPEED+1)
MOVX	@DPTR,A	2	2+5* (SPEED+1)

Read and write external extensions RAM The timing is shown in the figure below :



8051 7.2.7 Bit-addressable data memory

The bit-addressable data memory inside the single-chip microcomputer includes two parts: the address range of the first part is 8051 00H~7FH, The mapping of the address bytes of the second part; and the bit addressing area 80H~FFh The bit addressing area where the range is is the data area, which is 80H~FFH。 00H~7FH 20H~2FH 16 The domain is the address in all special function registers that can be 8 16 Divisible by A special function register (including 88H、90H、98H、A0H、A8H、 D8H、E0H、 E8H、F0H、F8H) of the mapping.

C0H、B0H、B8H、C8H、D0H、

Data storage address	Bit addressing address B4 B3							
	B7	B6	B5			B2	B1	B0
F8H（P7）F0H（B）	FFH F8H. 7	FEH F8H. 6	FDH F8H. 5	FCH F8H. 4	FBH F8H. 3	FAH F8H. 2	F9H F8H. 1	F8H F8H. 0
E8H（P6）	F7H F0H. 7	F6H F0H. 6	F5H F0H. 5	F4H F0H. 4	F3H F0H. 3	F2H F0H. 2	F1H F0H. 1	F0H F0H. 0
E0H（ACC）	EFH E8H. 7	EEH E8H. 6	EDH E8H. 5	ECH E8H. 4	EBH E8H. 3	EAH E8H. 2	E9H E8H. 1	E8H E8H. 0
D8H（CCON）	E7H E0H. 7	E6H E0H. 6	E5H E0H. 5	E4H E0H. 4	E3H E0H. 3	E2H E0H. 2	E1H E0H. 1	E0H E0H. 0
D0H（PSW）	DFH D8H. 7	DEH D8H. 6	DDH D8H. 5	DCH D8H. 4	DBH D8H. 3	DAH D8H. 2	D9H D8H. 1	D8H D8H. 0
	D7H D0H. 7	D6H D0H. 6	D5H D0H. 5	D4H D0H. 4	D3H D0H. 3	D2H D0H. 2	D1H D0H. 1	D0H D0H. 0
C8H（P5）	CFH C8H. 7	CEH C8H. 6	CDH C8H. 5	CCH C8H. 4	CBH C8H. 3	CAH C8H. 2	C9H C8H. 1	C8H C8H. 0
C0H（P4）	C7H C0H. 7	C6H C0H. 6	C5H C0H. 5	C4H C0H. 4	C3H C0H. 3	C2H C0H. 2	C1H C0H. 1	C0H C0H. 0
B8H（IP）	BFH B8H. 7	BEH B8H. 6	BDH B8H. 5	BCH B8H. 4	BBH B8H. 3	BAH B8H. 2	B9H B8H. 1	B8H B8H. 0
B0H（P3）	B7H B0H. 7	B6H B0H. 6	B5H B0H. 5	B4H B0H. 4	B3H B0H. 3	B2H B0H. 2	B1H B0H. 1	B0H B0H. 0
A8H（IE）	AFH A8H. 7	AEH A8H. 6	ADH A8H. 5	ACH A8H. 4	ABH A8H. 3	AAH A8H. 2	A9H A8H. 1	A8H A8H. 0
A0H（P2）	A7H A0H. 7	A6H A0H. 6	A5H A0H. 5	A4H A0H. 4	A3H A0H. 3	A2H A0H. 2	A1H A0H. 1	A0H A0H. 0
98H（SCON）	9FH 98H. 7	9EH 98H. 6	9DH 98H. 5	9CH 98H. 4	9BH 98H. 3	9AH 98H. 2	99H 98H. 1	98H 98H. 0
90H（P1）	97H 90H. 7	96H 90H. 6	95H 90H. 5	94H 90H. 4	93H 90H. 3	92H 90H. 2	91H 90H. 1	90H 90H. 0
88H（TCON）	8FH 88H. 7	8EH 88H. 6	8DH 88H. 5	8CH 88H. 4	8BH 88H. 3	8AH 88H. 2	89H 88H. 1	88H 88H. 0
80H（P0）	87H 80H. 7	86H 80H. 6	85H 80H. 5	84H 80H. 4	83H 80H. 3	82H 80H. 2	81H 80H. 1	80H 80H. 0
2FH	7FH 2FH. 7	7EH 2FH. 6	7DH 2FH. 5	7CH 2FH. 4	7BH 2FH.3	7AH 2FH. 2	79H 2FH. 1	78H 2FH. 0
2EH	77H 2EH. 7	76H 2EH. 6	75H 2EH. 5	74H 2EH. 4	73H 2EH. 3	72H 2EH. 2	71H 2EH. 1	70H 2EH. 0
2DH	6FH 2DH. 7	6EH 2DH. 6	6DH 2DH. 5	6CH 2DH. 4	6BH 2DH. 3	6AH 2DH. 2	69H 2DH. 1	68H 2DH. 0
2CH	67H 2CH. 7	66H 2CH. 6	65H 2CH. 5	64H 2CH. 4	63H 2CH. 3	62H 2CH. 2	61H 2CH. 1	60H 2CH. 0
2BH	5FH 2BH. 7	5EH 2BH. 6	5DH 2BH. 5	5CH 2BH. 4	5BH 2BH. 3	5AH 2BH. 2	59H 2BH. 1	58H 2BH. 0
2AH	57H 2AH. 7	56H 2AH. 6	55H 2AH. 5	54H 2AH. 4	53H 2AH. 3	52H 2AH. 2	51H 2AH. 1	50H 2AH. 0
29H	4FH 29H. 7	4EH 29H. 6	4DH 29H. 5	4CH 29H. 4	4BH 29H. 3	4AH 29H. 2	49H 29H. 1	48H 29H. 0

28H	47H 28H. 7	46H 28H. 6	45H 28H. 5	44H 28H. 4	43H 28H. 3	42H 28H. 2	41H 28H. 1	40H 28H. 0
27H	3FH 27H. 7	3EH 27H. 6	3DH 27H. 5	3CH 27H. 4	3BH 27H. 3	3AH 27H. 2	39H 27H. 1	38H 27H. 0
26H	37H 26H. 7	36H 26H. 6	35H 26H. 5	34H 26H. 4	33H 26H. 3	32H 26H. 2	31H 26H. 1	30H 26H. 0
25H	2FH 25H. 7	2EH 25H. 6	2DH 25H. 5	2CH 25H. 4	2BH 25H. 3	2AH 25H. 2	29H 25H. 1	28H 25H. 0
24H	27H 24H. 7	26H 24H. 6	25H 24H. 5	24H 24H. 4	23H 24H. 3	22H 24H. 2	21H 24H. 1	20H 24H. 0
23H	1FH 23H. 7	1EH 23H. 6	1DH 23H. 5	1CH 23H. 4	1BH 23H. 3	1AH 23H. 2	19H 23H. 1	18H 23H. 0
22H	17H 22H. 7	16H 22H. 6	15H 22H. 5	14H 22H. 4	13H 22H. 3	12H 22H. 2	11H 22H. 1	10H 22H. 0
21H	0FH 21H. 7	0EH 21H. 6	0DH 21H. 5	0CH 21H. 4	0BH 21H. 3	0AH 21H. 2	09H 21H. 1	08H 21H. 0
20H	07H 20H. 7	06H 20H. 6	05H 20H. 5	04H 20H. 4	03H 20H. 3	02H 20H. 2	01H 20H. 1	00H 20H. 0

Special parameters in the memory, in 7.8 ISP The program can be burned when downloading FLASH

STC12H The internal data memory and program memory of the series of microcontrollers store some special parameters related to the chip, unique ID including: Global number、32K The frequency of the power-down wake-up timer, the value of the internal reference signal source, and the parameters. 1.19V IRC These parameters are in the program memory (CODE) The storage addresses in are as follows：

Parameter name	Save address			Parameter description
	STC12H1K08	STC12H1K16	STC12H1K28	
Globally unique number ID	1FF9H~1FFFH	2FF9H~2FFFH	43F9H~43FFH	7 Byte
inside 1.19V Reference signal source	1FF7H~1FF8H	2FF7H~2FF8H	43F7H~43F8H	millivolt (high byte first)
The frequency of the power-down wake-up timer 32K	1FF5H~1FF6H	2FF5H~2FF6H	43F5H~43F6H	Hz (High byte comes first)
22.1184MHz of 20M Frequency band) IRC Parameters (	1FF4H	2FF4H	43F4H	-
24MHz of 20M Frequency band) IRC Parameters (	1FF3H	2FF3H	43F3H	-
20M Frequency band)	1FF2H	2FF2H	43F2H	The firmware version is 7.3.12U And subsequent versions are valid
IRC Parameters (20MHz of	1FF1H	2FF1H	43F1H	
IRC27MHz	1FF0H	2FF0H	43F0H	
of IRC30MHz Parameters (35M Frequency band) parameters (35M Frequency band) Parameters (35M of IRC33.1776MHz Frequency band)	1FEFH	2FEFH	43EFH	
35MHz of 35M Frequency band) IRC Parameters (	1FEEH	2FEEH	43EEH	
36.864MHz of IRC Parameters (35M Frequency band)	1FEDH	2FEDH	43EDH	
Reserved	1FECH	2FECH	43ECH	
reserved	1FEBH	2FEBH	43EBH	
Frequency band VRTRIM parameters 20M	1FEAH	2FEAH	43EAH	
Frequency band VRTRIM parameters 35M	1FE9H	2FE9H	43E9H	

These parameters are in the data memory (IRAM) The storage addresses in are as follows：

Parameter name	Save address	Parameter description
inside 1.19V Reference signal source	idata： 0EFH~0F0H	Millivolt (high byte front)
The only one in the world ID The frequency	：idata 0F1H~0F7H	7 Byte (high byte
32K of the power-down wake-up timer	：idata 0F8H~0F9H	Hz comes first)
of IRC parameters 22.1184MHz	：0FAHidata	-
24MHz of IRC parameters	idata：0FBH	-

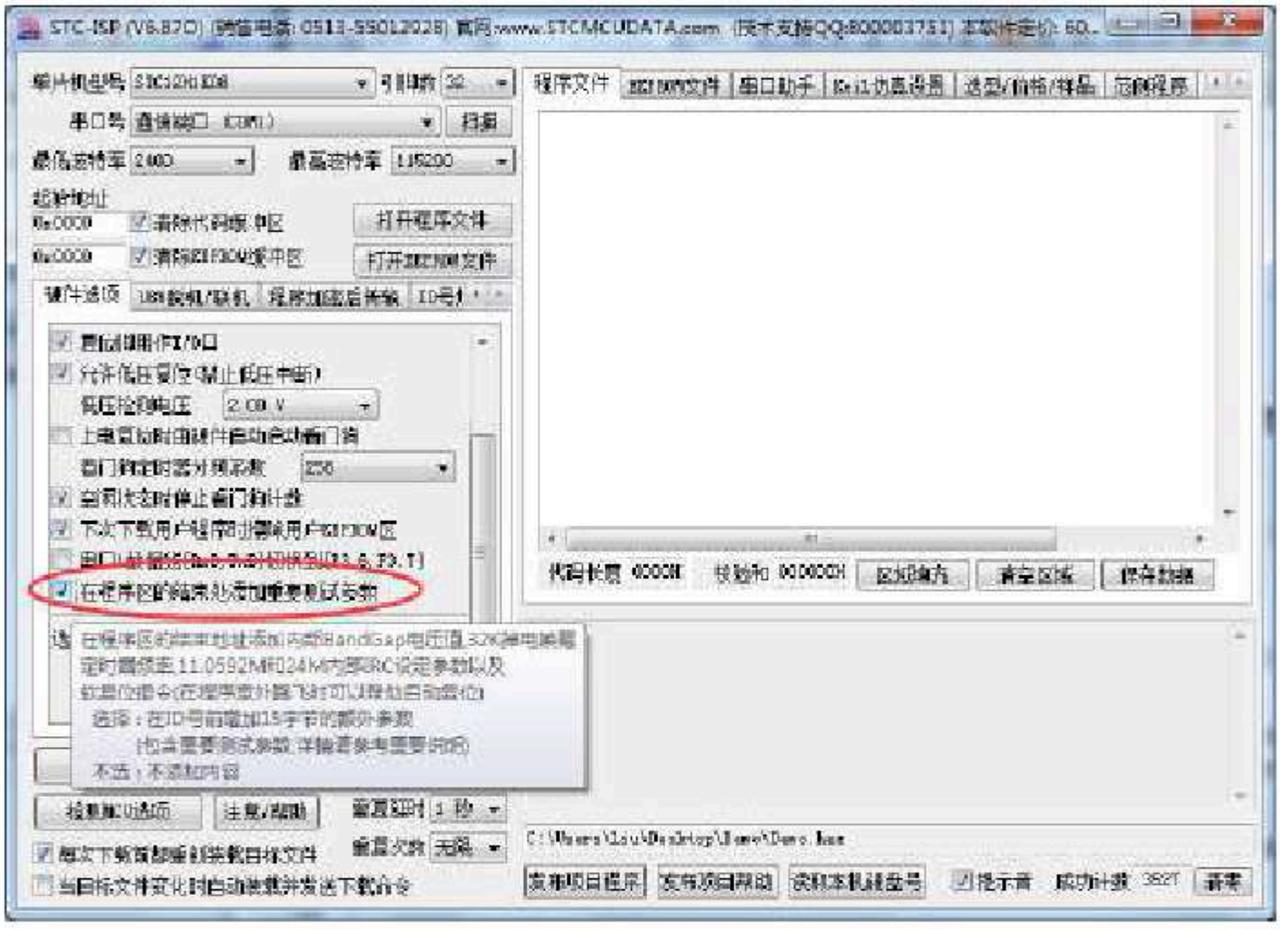
Special note

RAM 1, Because it is The parameters in it may be modified, so it is generally not recommended for users to use, especially when encrypted, it is strongly recommended to be used to read ROM in ID data.

STC12H1K28、2, Due STC12H1K33 These The size of the user can be set by themselves, it is possible to save EEPROM to important parameters ROM The space is set models of EEPROM And artificially erase or modify important parameters, so use this model for numbering. ID

You may need to consider this issue when encrypting the line.

3, By default, there is only in the program memory The only one in the world inside The data of the reference signal source Power-down wake-up timer Frequency and The parameters are not available, you need ISP When downloading, select the 32K option shown in the figure below to be available.



Read inside 1.19V (Reference signal source value from Flash Program memory (ROM) 1.23V Reading)

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 AUXR = 0x8e;
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;
bit busy;
int *BGV;
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 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;

    BGV = (int code *)0x3ff7;                   // STC12H1K16
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                        // Read inside 1.19V The high byte of the reference signal source
    UartSend(*BGV);                             // Read inside The low byte of the reference signal source

    while (1);
}
```

Assembly code

The test operating frequency is 11.0592MHz

```
;
AUXR    DATA    8EH
BGV     EQU     03FF7H                  ;STC12H1K16

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     0023H
        LJMP    UART_ISR

        ORG     0100H
```

```
UART_ISR:
        JNB     TI,CHKRI
        CLR     TI
        CLR     BUSY
CHKRI:
        JNB     RI,UARTISR_EXIT
        CLR     RI
UARTISR_EXIT:
        RETI

UART_INIT:
        MOV     SCON,#50H
        MOV     TMOD,#00H
        MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     TH1,#0FFH
        SETB    TR1
        MOV     AUXR,#40H
        CLR     BUSY
        RET

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

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

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#BGV
        CLR     A
        MOVC    A,@A+DPTR               ;Read inside 1.19V The high byte of the reference signal source
        LCALL   UART_SEND
        MOV     A,#1
        MOVC    A,@A+DPTR               ;Read inside 1.19V The low byte of the reference signal source
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

7.3.2 Read inside 1.19V (Reference signal source value from) Reading RAM

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 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;

bit busy;

int *BGV;

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 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;

        BGV = (int idata *)0xef;
        UartInit();
        ES = 1;
        EA = 1;
        UartSend(*BGV >> 8);                    //Read inside 1.19V The high byte of the reference signal source
        UartSend(*BGV);                         //Read inside The low byte of the reference signal source
        while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR    DATA    8EH
BGV     DATA    0EFH

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     0023H
        LJMP    UART_ISR

        ORG     0100H

UART_ISR:
```

```
                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     R0,#BGV
                MOV     A,@R0                   ;Read inside 5.19V The high byte of the reference signal source
                LCALL   UART_SEND
                INC     R0
                MOV     A,@R0                   ;Read inside 5.19V The low byte of the reference signal source
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

7.3.3 Read the only one in the world ID (Number from Flash Program memory (ROM) Read in)

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 AUXR = 0x8e;

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;

bit busy;

char *ID;

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 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;
        char i;

        ID = (char code *)0x3ff9;                       // STC12H1K16
        UartInit();
        ES = 1;
        EA = 1;

        for (i=0; i<7; i++)
        {
                UartSend(ID[i]);
        }

        while (1);
}
```

Assembly code

```
; The test operating frequency is 11.0592MHz

AUXR            DATA    8EH
ID              EQU     03FF9H          ; STC12H1K16

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     0023H
                LJMP    UART_ISR
```

```
                ORG     0100H

UART_ISR:
                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#ID
                MOV     R1,#7
NEXT:           CLR     A
                MOVC    A,@A+DPTR
                LCALL   UART_SEND
                INC     DPTR
                DJNZ    R1,NEXT

LOOP:
                JMP     LOOP
```

```
END
```

7.3.4 Read the only one in the world ID Number from RAM)Reading

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 AUXR = 0x8e;
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;
bit busy;
char *ID;
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 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;
    char i;


    ID = (char idata *)0xf1;
    UartInit();
    ES = 1;
    EA = 1;
    for (i=0; i<7; i++)
    {
        UartSend(ID[i]);
    }

    while (1);
}
```

Assembly code

The test operating frequency is 11.0592MHz

```
;

AUXR    DATA    8EH
ID      DATA    0F1H

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     0023H
                LJMP    UART_ISR

                ORG     0100H

UART_ISR:
                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H       ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     R0,#ID
                MOV     R1,#7
NEXT:           MOV     A,@R0
                LCALL   UART_SEND
                INC     R0
                DJNZ    R1,NEXT
```

```
LOOP:
        JMP     LOOP

        END
```

7.3.5 read 32K The frequency of the power-down wake-up timer (from Flash Program memory (ROM) Read in)

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 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;
bit busy;
int *F32K;
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 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;

        F32K = (int code *)0x3ff5;                              // STC12H1K16
        UartInit();
        ES = 1;
        EA = 1;

        UartSend(*F32K >> 8);                                   //read 32K High byte of frequency
        UartSend(*F32K);                                        //read Low byte of frequency

        while (1);
}
```

Assembly code

The test operating frequency is 11.0592MHz

```
;
AUXR            DATA    8EH
F32K            EQU     03FF5H                  ; STC12H1K16

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     0023H
                LJMP    UART_ISR

                ORG     0100H

UART_ISR:
                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#F32K
                CLR     A
                MOVC    A,@A+DPTR               ;read 32K High byte of frequency
                LCALL   UART_SEND
                INC     DPTR
```

```
                CLR     A
                MOVC    A,@A+DPTR
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

; read 32K Low byte of frequency

7.3.6 read 32K The frequency of the power-down wake-up timer is read (Reading

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 AUXR  = 0x8e;
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;

bit busy;

int *F32K;

void Uart1Isr() 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 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;

        F32K = (int idata *)0xf8;
        UartInit();
        ES = 1;
        EA = 1;
        UartSend(*F32K >> 8);
        UartSend(*F32K);
        while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR        DATA    8EH
F32K        DATA    0F8H

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     0023H
                LJMP    UART_ISR

                ORG     0100H

UART_ISR:
                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
UARTISR_EXIT:
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     R0,#F32K
                MOV     A,@R0                   ;read 32K High byte of frequency
                LCALL   UART_SEND
                INC     R0
```

```
        MOV     A,@R0                   ;read 32K Low byte of frequency
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

7.3.7 User-defined interior IRC (Frequency from Flash Program memory (ROM) in read)

Language code C

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

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

#define CLKSEL  (*(unsigned char volatile xdata *)0xfe00)
#define CLKDIV  (*(unsigned char volatile xdata *)0xfe01)

//The following table is STC8H1K08-20Pin List of parameters

#define ID_ROMADDR          ((unsigned char code *)0x1ff9)
#define VREF_ROMADDR        (*(unsigned int code *)0x1ff7)
#define F32K_ROMADDR        (*(unsigned int code *)0x1ff5)
#define T22M_ROMADDR        (*(unsigned char code *)0x1ff4)   //22.1184MHz
#define T24M_ROMADDR        (*(unsigned char code *)0x1ff3)   //24MHz
#define T20M_ROMADDR        (*(unsigned char code *)0x1ff2)   //20MHz
#define T27M_ROMADDR        (*(unsigned char code *)0x1ff1)   //27MHz
#define T30M_ROMADDR        (*(unsigned char code *)0x1ff0)   //30MHz
#define T33M_ROMADDR        (*(unsigned char code *)0x1fef)   //33.1776MHz
#define T35M_ROMADDR        (*(unsigned char code *)0x1fee)   //35MHz
#define T36M_ROMADDR        (*(unsigned char code *)0x1fed)   //36.864MHz
#define VRT20M_ROMADDR      (*(unsigned char code *)0x1fea)   //VRTRIM_20M
#define VRT35M_ROMADDR      (*(unsigned char code *)0x1fe9)   //VRTRIM_35M


sfr     P_SW2   =   0xba;
sfr     IRCBAND =   0x9d;
sfr     IRTRIM  =   0x9f;
sfr     VRTRIM  =   0xa6;


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;

//                  20MHz //choose
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T20M_ROMADDR;
//      VRTRIM = VRT20M_ROMADDR;
//      IRCBAND = 0x00;
//      CLKDIV = 0x00;

//                  22.1184MHz //choose
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T22M_ROMADDR;
//      VRTRIM = VRT20M_ROMADDR;
//      IRCBAND = 0x00;
//      CLKDIV = 0x00;

    //choose  24MHz
    P_SW2 = 0x80;
    CLKDIV = 0x04;
    IRTRIM = T24M_ROMADDR;
    VRTRIM = VRT20M_ROMADDR;
    IRCBAND = 0x00;
    CLKDIV = 0x00;

//                  27MHz //choose
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T27M_ROMADDR;
//      VRTRIM = VRT35M_ROMADDR;
//      IRCBAND = 0x01;
//      CLKDIV = 0x00;

//                  30MHz //choose
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T30M_ROMADDR;
//      VRTRIM = VRT35M_ROMADDR;
//      IRCBAND = 0x01;
//      CLKDIV = 0x00;

//                  33.1776MHz //choose
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T33M_ROMADDR;
```

```
//      VRTRIM = VRT35M_ROMADDR;
//      IRCBAND = 0x01;
//      CLKDIV = 0x00;

//                35MHz //choose
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T35M_ROMADDR;
//      VRTRIM = VRT35M_ROMADDR;
//      IRCBAND = 0x01;
//      CLKDIV = 0x00;

      while (1);
}
```

Assembly code

```
;The test operating frequency is //35.0000MHz
;The following table is STC12H1K08-20Pin List of parameters

ID_ROMADDR          EQU     01FF9H
VREF_ROMADDR        EQU     01FF7H
F32K_ROMADDR        EQU     01FF5H
T22M_ROMADDR        EQU     01FF4H      //22.1184MHz
T24M_ROMADDR        EQU     01FF3H      //24MHz
T20M_ROMADDR        EQU     01FF2H      //20MHz
T27M_ROMADDR        EQU     01FF1H      //27MHz
T30M_ROMADDR        EQU     01FF0H      //30MHz
T33M_ROMADDR        EQU     01FEFH      //33.1776MHz
T35M_ROMADDR        EQU     01FEEH      //35MHz
T36M_ROMADDR        EQU     01FEDH      //36.864MHz
VRT20M_ROMADDR      EQU     01FEAH      //VRTRIM_20M
VRT35M_ROMADDR      EQU     01FE9H      //VRTRIM_35M

P_SW2       DATA    0BAH
CLKSEL      EQU     0FE00H
CLKDIV      EQU     0FE01H

IRCBAND     DATA    09DH
IRCTRIM     DATA    09FH
VRTRIM      DATA    0A6H

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


;       ;choose 20MHz
;       MOV     P_SW2,#80H
;       MOV     A,#4
;       MOV     DPTR,#CLKDIV
;       MOV     DPTR,#T20M_ROMADDR
;       CLR     A
;       MOVC    A,@A+DPTR
;       MOV     IRTRIM,A
;       MOV     DPTR,#VRT20M_ROMADDR
;       CLR     A
;       MOVC    A,@A+DPTR
;       MOV     VRTRIM,A
;       MOV     IRCBAND,#00H
;       MOV     A,#0
;       MOV     DPTR,#CLKDIV
;       MOV     P_SW2,#00H


;       ;choose 22.1184MHz
;       MOV     P_SW2,#80H
;       MOV     A,#4
;       MOV     DPTR,#CLKDIV
;       MOV     DPTR,#T22M_ROMADDR
;       CLR     A
;       MOVC    A,@A+DPTR
;       MOV     IRTRIM,A
;       MOV     DPTR,#VRT20M_ROMADDR
;       CLR     A
;       MOVC    A,@A+DPTR
;       MOV     VRTRIM,A
;       MOV     IRCBAND,#00H
;       MOV     A,#0
;       MOV     DPTR,#CLKDIV
;       MOV     P_SW2,#00H


        ;choose 24MHz
        MOV     P_SW2,#80H
        MOV     A,#4
        MOV     DPTR,#CLKDIV
        MOV     DPTR,#T24M_ROMADDR
        CLR     A
        MOVC    A,@A+DPTR
        MOV     IRTRIM,A
        MOV     DPTR,#VRT20M_ROMADDR
        CLR     A
```

```
    MOVC    A,@A+DPTR
    MOV     VRTRIM,A
    MOV     IRCBAND,#00H
    MOV     A,#0
    MOV     DPTR,#CLKDIV
    MOV     P_SW2,#00H

;   ;choose 27MHz
;   MOV     P_SW2,#80H
;   MOV     A,#4
;   MOV     DPTR,#CLKDIV
;   MOV     DPTR,#T27M_ROMADDR
;   CLR     A
;   MOVC    A,@A+DPTR
;   MOV     IRTRIM,A
;   MOV     DPTR,#VRT35M_ROMADDR
;   CLR     A
;   MOVC    A,@A+DPTR
;   MOV     VRTRIM,A
;   MOV     IRCBAND,#01H
;   MOV     A,#0
;   MOV     DPTR,#CLKDIV
;   MOV     P_SW2,#00H

;   ;choose 30MHz
;   MOV     P_SW2,#80H
;   MOV     A,#4
;   MOV     DPTR,#CLKDIV
;   MOV     DPTR,#T30M_ROMADDR
;   CLR     A
;   MOVC    A,@A+DPTR
;   MOV     IRTRIM,A
;   MOV     DPTR,#VRT35M_ROMADDR
;   CLR     A
;   MOVC    A,@A+DPTR
;   MOV     VRTRIM,A
;   MOV     IRCBAND,#01H
;   MOV     A,#0
;   MOV     DPTR,#CLKDIV
;   MOV     P_SW2,#00H

;   ;choose 33.1776MHz
;   MOV     P_SW2,#80H
;   MOV     A,#4
;   MOV     DPTR,#CLKDIV
;   MOV     DPTR,#T33M_ROMADDR
;   CLR     A
;   MOVC    A,@A+DPTR
;   MOV     IRTRIM,A
;   MOV     DPTR,#VRT35M_ROMADDR
;   CLR     A
;   MOVC    A,@A+DPTR
;   MOV     VRTRIM,A
;   MOV     IRCBAND,#01H
;   MOV     A,#0
;   MOV     DPTR,#CLKDIV
;   MOV     P_SW2,#00H

;   ;choose 35MHz
```

```
;                   MOV     P_SW2,#80H
;                   MOV     A,#4
;                   MOV     DPTR,#CLKDIV
;                   MOV     DPTR,#T35M_ROMADDR
;                   CLR     A
;                   MOVC    A,@A+DPTR
;                   MOV     IRTRIM,A
;                   MOV     DPTR,#VRT35M_ROMADDR
;                   CLR     A
;                   MOVC    A,@A+DPTR
;                   MOV     VRTRIM,A
;                   MOV     IRCBAND,#01H
;                   MOV     A,#0
;                   MOV     DPTR,#CLKDIV
;                   MOV     P_SW2,#00H

;                   ;choose 36.864MHz
;                   MOV     P_SW2,#80H
;                   MOV     A,#4
;                   MOV     DPTR,#CLKDIV
;                   MOV     DPTR,#T36M_ROMADDR
;                   CLR     A
;                   MOVC    A,@A+DPTR
;                   MOV     IRTRIM,A
;                   MOV     DPTR,#VRT35M_ROMADDR
;                   CLR     A
;                   MOVC    A,@A+DPTR
;                   MOV     VRTRIM,A
;                   MOV     IRCBAND,#01H
;                   MOV     A,#0
;                   MOV     DPTR,#CLKDIV
;                   MOV     P_SW2,#00H

                    JMP     $

                    END
```

7.3.8 User-defined interior IRC (Frequency from RAM) Reading

C Language code

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

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

#define CLKDIV (*(unsigned char volatile xdata *)0xfe01)
sfr P_SW2  = 0xba;
sfr IRTRIM = 0x9f;
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;

char    *IRC22M;
char    *IRC24M;

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;

    IRC22M = (char idata *)0xfa;
    IRC24M = (char idata *) 0xfb;
//  IRTRIM = *IRC22M;
    IRTRIM = *IRC24M;
    P_SW2 = 0x80;
    CLKDIV = 0;
    P_SW2 = 0x00;
    while (1);
}
```

//load 22.1184MHz of parameters
//load 24MHz IRC The parameters of IRC

//The master clock does not prescribe the frequency

Assembly code

```
;The test operating frequency is 11.0592MHz

P_SW2   DATA    0BAH
CLKDIV  EQU     0FE01H

IRTRIM  DATA    09FH

IRC22M  DATA    0FAH
IRC24M  DATA    0FBH

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     R0,#IRC22M      ;load 22.1184MHz of IRC parameters
;               MOV     IRTRIM,@R0
                MOV     R0,#IRC24M      ;load 24MHz of IRC parameters
                MOV     IRTRIM,@R0

                MOV     P_SW2,#80H
                MOV     A,#0            ;The master clock does not prescribe the frequency
                MOV     DPTR,#CLKDIV
                MOVX    @DPTR,A
                MOV     P_SW2,#00H

                JMP     $

                END
```

Special function register

8.1 STC12H1K08 series

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		CH	CCAP0H	CCAP1H	CCAP2H			RSTCFG
F0H	B		PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS		
E8H		CL	CCAP0L	CCAP1L	CCAP2L			AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2		ADCCFG	
D0H	PSW	T4T3M	T4H	T4L	T3H	T3L	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H		WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2		ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H	P2		P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH				PCON

Bit addressable Non-bit addressable

Note: The register address can be 8 Only those that are divisible can be bit addressable, and those that cannot be divisible cannot be bit address

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEF8H	PWMB_CCR6L	PWMB_CCR7H	PWMB_CCR7L	PWMB_CCR8H	PWMB_CCR8L	PWMB_BKR	PWMB_DTR	PWMB_OISR
FEF0H	PWMB_PSCRH	PWMB_PSCRL	PWMB_ARRH	PWMB_ARRL	PWMB_RCR	PWMB_CCR5H	PWMB_CCR5L	PWMB_CCR6H
FEE8H	PWMB_CCMR1	PWMB_CCMR2	PWMB_CCMR3	PWMB_CCMR4	PWMB_CCER1	PWMB_CCER2	PWMB_CNTRH	PWMB_CNTRL
FEE0H	PWMB_CR1	PWMB_CR2	PWMB_SMCR	PWMB_ETR	PWMB_IER	PWMB_SR1	PWMB_SR2	PWMB_EGR
FED8H	PWMA_CCR2L	PWMA_CCR3H	PWMA_CCR3L	PWMA_CCR4H	PWMA_CCR4L	PWMA_BKR	PWMA_DTR	PWMA_OISR
FED0H	PWMA_PSCRH	PWMA_PSCRL	PWMA_ARRH	PWMA_ARRL	PWMA_RCR	PWMA_CCR1H	PWMA_CCR1L	PWMA_CCR2H
FEC8H	PWMA_CCMR1 PWMA_CCMR2		PWMA_CCMR3	PWMA_CCMR4	PWMA_CCER1	PWMA_CCER2 PWMA_CNTRH	PWMA_CNTRL	
FEC0H	PWMA_CR1	PWMA_CR2	PWMA_SMCR	PWMA_ETR	PWMA_IER	PWMA_SR1	PWMA_SR2	PWMA_EGR
FEB0H	PWMA_ETRPS	PWMA_ENO	PWMA_PS	PWMA_IOAUX	PWMB_ETRPS	PWMB_ENO	PWMB_PS	PWMB_IOAUX
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H		P1IE	P2IE					
FE28H	P0DR	P1DR	P2DR	P3DR		P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR		P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS		P5NCS		
FE10H	P0PU	P1PU	P2PU	P3PU		P5PU		

FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FD50H					CCAPM3	CCAP3L	CCAP3H	PCA_PWM3
FD30H	P0IM1	P1IM1	P2IM1	P3IM1		P5IM1		
FD20H	P0IM0	P1IM0	P2IM0	P3IM0		P5IM0		
FD10H	P0INTF	P1INTF	P2INTF	P3INTF		P5INTF		
FD00H	P0INTE	P1INTE	P2INTE	P3INTE		P5INTE		
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON

List of special function registers 8.2

Note: The register address can be 8 Only divisible ones can be bit-addressable, visible ones are not bit addressable.

STC12H **Bit-addressable register :** P0 (80H) 、 TCON (88H) 8、 P1 (90H) 、 SCON (98H) 、 A0H) 、 IE (A8H), (B0H) P (B8H), (IP P5 C8H) 、 PSW (D8H) 、 D0H) ACC (CON P2 ((E0H), (B F0H)

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 port	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
SP	Stack pointer , data	81H									0000,0111
DPL	pointer (low byte) , data	82H									0000,0000
DPH	pointer (high byte)	83H									0000,0000
PCON	, power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	Timer control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	Timer mode register	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	timer low 0 8 Bit register	8AH									0000,0000
TL1	Timer low 1 8 Bit register	8BH									0000,0000
TH0	Timer high 08 Bit register	8CH									0000,0000
TH1	Timer high 1 Bit register	8DH									0000,0000
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	Interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	P1 port	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P1M1	P1 Port configuration register	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	P1 Port configuration register 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0 0000,0000	
P0M1	P0 Port configuration register	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	P0 Port configuration register	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P2M1	P2 Port configuration register	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	P2 Port configuration register	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0 0000,0000	
SCON	1 serial port Control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	1 serial port Data register	99H									0000,0000
S2CON	2 serial port Control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	Serial port data register	9BH									0000,0000
IRCBAND	IRC Frequency band selection detection	9DH	-	-	-	-	-	-	-	SEL	xxxx,xxxn
LIRTRIM	IRC Frequency trimmer register	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		0000,00nn
IRTRIM	IRC Frequency adjustment register	9FH	IRTRIM[7:0]								nnnn,nnnn
P2	P2 port	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P_SW1	Peripheral port switch register	A2H	S1_S[1:0]				SPI_S[1:0]		0	-	nn00,000x
IE	Interrupt allow register	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
SADDR	Serial slave address register	A9H									0000,0000
WKTCL	Power-down wake-up timer low byte	AAH									1111,1111
WKTCH	Power-down wake-up timer high byte	ABH	WKTEN								0111,1111
TA	DPTR Timing control register	AEH									0000,0000
IE2	Interrupt allow register 2	AFH	-	ET4	ET3	-	-	ET2	ESPI	ES2	x000,0000
P3	P3 port	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P3M1	P3 Port configuration register	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	1111,1100

P3M0	P3 Port configuration register 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	0000,0000
IP2	Interrupt priority control register 2	B5H	-	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	High interrupt priority control register 2	B6H	-	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H	0000,0000
IPH	High interrupt priority control register	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP	Interrupt priority control register	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
SADEN	1 Serial Slave address mask register	B9H									0000,0000
P_SW2	port peripheral port switch register 2	BAH	EAXFR	-	I2C_S[1:0]		-	-	-	S2_S	0x00,0000
ADC_CONTR	ADC Control register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT		ADC_CHS[3:0]			000x,0000
ADC_RES	ADC The conversion result is sent in high position Memory	BDH									0000,0000
ADC_RESL	ADC Conversion result low register	BEH									0000,0000
WDT_CONTR	Watchdog control register	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
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
P5	P5 port	C8H	P57	P56	P55	P54	P53	P52	P51	P50	xx11,1111
P5M1	P5 Port configuration register	C9H	P57M1	P56M1	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	P5 Port configuration register	CAH	P57M0	P56M0	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx00,0000
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
PSW	Program status word	D0H	CY	AC	F0	RS1	RS0	OV	F1	P	0000,0000
T4T3M	register Timer control register 4/3	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	Timer high byte 4	D2H									0000,0000
T4L	timer low byte 4	D3H									0000,0000
T3H	3 Timer high byte	D4H									0000,0000
T3L	3 Timer low byte	D5H									0000,0000
T2H	timer high byte 2	D6H									0000,0000
T2L	Timer low byte 2	D7H									0000,0000
CCON	PCA Control register	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,0000
CMOD	PCA Mode register	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA Module 0 mode control register	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 2 mode control register	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
ADCCFG	ADC Configuration register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
ACC	Accumulator	E0H									0000,0000
DPS	Pointer selector DPTR	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
DPL1	The second set of data pointers (low	E4H									0000,0000
DPH1	byte) The second set of data pointers	E5H									0000,0000
CMPCR1	(high byte) 1 Comparator control	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	register 2 Comparator control register	E7H	INVCMPO		DISFLT	LCDTY[5:0]					0000,0000
CL	PCA Counter low byte	E9H									0000,0000

CCAP0L	PCA 0 Module low byte	EAH							0000,0000
CCAP1L	PCA 1 Module low byte	EBH							0000,0000
CCAP2L	PCA 2 Module low byte	ECH							0000,0000
AUXINTIF	Extended external interrupt flag register	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
B	B register	F0H									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
IAP_TPS	IAP Waiting time control register	F5H	-	-	IAPTPS[5:0]						xx00,0000
CH	PCA Counter high byte	F9H									0000,0000
CCAP0H	PCA 0 module High byte	FAH									0000,0000
CCAP1H	PCA 1 module High byte	FBH									0000,0000
CCAP2H	PCA Module 2 high byte	FCH									0000,0000
RSTCFG	Reset configuration register	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		0000,0000

The following special function registers are the highest STC12C5A Area, before visiting, you need to The logical address is located at) Register bits of the extender (EAXFR) Set, and then use 1 MOVX A,@DPTR and MOVX @DPTR,A instructions for access

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	Clock selection register	FE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	Clock divider register	FE01H									nnnn,nnnn
HIRCCR	Internal high-speed oscillator control register	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	External crystal oscillator control register	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	inside 32K Oscillator control register	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	Master clock output control register	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	Internal high-speed oscillator debounce control	FE06H	IRCDB_PAR[7:0]								1000,0000
P0PU	P0 Port pull-up resistor control register	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 Port pull-up resistor control register	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 Port pull-up resistor control register	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 Port pull-up resistor control register	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P5PU	P5 Port pull-up resistor control register	FE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xxx0,0000
P0NCS	P0 Port Schmidt trigger control register	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 Port Schmidt trigger control register	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 Port Schmidt trigger control register	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 Port Schmidt trigger control register	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P5NCS	P5 Port Schmidt trigger control register	FE1DH	P57NCS	P56NCS	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xxx0,0000
P0SR	P0 Port level conversion rate register	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	P1 Port level conversion rate register	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	P2 Port level conversion rate register	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	P3 Port level conversion rate register	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P5SR	P5 Port level conversion rate register	FE25H	P57SR	P56SR	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xxx1,1111
P0DR	P0 Port drive current control register	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 Port drive current control register	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 Port drive current control register	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 Port drive current control register	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111

P5DR	P5 Port drive current control register	FE2DH	P57DR	P56DR	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xxx1,1111
P1IE	P1 Port input enable control register	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P2IE	P2 Port input enable control register	FE32H	-	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE	x111,1111
I2CCFG	I²C Configuration register	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I²C Host control register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C Host status register	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO 00xx,xx00	
I2CSLCR	I²C Slave control register	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C Slave status register	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C Slave address register	FE85H	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I²C Data transmission register	FE86H									0000,0000
I2CRXD	I²C Data receiving register	FE87H									0000,0000
I2CMSAUX	I²C Host auxiliary control register	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
TM2PS	Timer clock prescaler register	FEA2H									0000,0000
TM3PS	Timer clock prescaler register	FEA3H									0000,0000
TM4PS	Timer clock prescaler register	FEA4H									0000,0000
ADCTIM	ADC Timing control register	FEA8H CSSETUP		CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010
PWMA_ETRPS PWMA	of ETR Select register	FEB0H						BRKAPS	ETRAPS[1:0]		xxxx,x000
PWMA_ENO	PWMA output enable control	FEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P	0000,0000
PWMA_PS	PWMA output pin select register	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMA_IOAUX PWMA	auxiliary register	FEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P	0000,0000
Select register PWMB_ETRPS PWMB of ETR		FEB4H						BRKBPS	ETRBPS[1:0]		xxxx,x000
PWMB Output enable control PWMB_ENO		FEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P	x0x0,x0x0
PWMB Output pin selection register PWMB_PS		FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMB_IOAUX PWMB Auxiliary register		FEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P	x0x0,x0x0
PWMA_CR1	PWMA 1 Control register	FEC0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMA_CR2	PWMA Control register 2	FEC1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMA_SMCR PWMA	Interrupt enable register from the	FEC2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMA_ETR	PWMA external trigger register of the mode	FEC3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMA_IER	PWMA control register 1 Status register	FEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA 2 Status register	FEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA Event occurrence register	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMA_EGR	PWMA Capture mode register 1	FEC7H	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG	0000,0000
PWMA_CCMR1	PWMA 1 Compare mode register	FEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		0000,0000
	PWMA 2 Capture mode register		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]		0000,0000
PWMA_CCMR2	PWMA 2 Compare mode register	FEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		0000,0000
	PWMA 3 Capture mode register		IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		0000,0000
PWMA_CCMR3	PWMA 3 Compare mode register	FECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]		0000,0000
	PWMA 4 Capture mode register		IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]		0000,0000
PWMA_CCMR4	PWMA Compare mode register 4	FECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]		0000,0000
	PWMA Capture comparison enable register 1		IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]		0000,0000
PWMA_CCER1	PWMA	FECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E	0000,0000
PWMA_CCER2 PWMA	Capture comparison enable register 2	FECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	0000,0000
PWMA_CNTRH PWMA	Counter high byte	FECEH									0000,0000
PWMA_CNTRL PWMA	Counter low byte	FECFH									0000,0000

PWMA_PSCRH	PWMA	Prescaler high byte	FED0H						0000,0000
PWMA_PSCRL	PWMA	Prescaler low byte	FED1H						0000,0000
PWMA_ARRH	PWMA	Automatic reloading of register high bytes	FED2H						0000,0000
PWMA_ARRL	PWMA	Automatic reloading of register low bytes	FED3H						0000,0000
PWMA_RCR	PWMA	Repeat counter register	FED4H						0000,0000
PWMA_CCR1H	PWMA	Compare the high position of the capture register	FED5H						0000,0000
PWMA_CCR1L	PWMA	Compare the low bits of the capture register	FED6H						0000,0000
PWMA_CCR2H	PWMA	Compare capture register 2 High position	FED7H						0000,0000
PWMA_CCR2L	PWMA	Compare capture register 2 Low position	FED8H						0000,0000
PWMA_CCR3H	PWMA	Compare the high position of the capture register	FED9H						0000,0000
PWMA_CCR3L	PWMA	Compare the low bits of the capture register	FEDAH						0000,0000
PWMA_CCR4H	PWMA	Compare the high position of the capture register	FEDBH						0000,0000
PWMA_CCR4L	PWMA	Compare the low bits of the capture register	FEDCH						0000,0000
PWMA_BKR	PWMA	Brake register dead	FEDDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		0000,000x
PWMA_DTR	PWMA	zone control register output	FEDEH	DTG[7:0]								0000,0000
PWMA_OISR	PWMA	idle status register	FEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	0000,0000
PWMB_CR1	PWMB	Control register 1	FEE0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMB_CR2	PWMB	Control register 2	FEE1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMB_SMCR	PWMB	Slave mode control register	FEE2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMB_ETR	PWMB	External trigger register	FEE3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMB_IER	PWMB	Interrupt enable register	FEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB	Status register 1	FEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB	Status register 2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x
PWMB_EGR	PWMB	Event occurrence register	FEE7H	BG	TG	COMG	CC8G	CC7G	CC6G	CC5G	UG	0000,0000
PWMB_CCMR1	PWMB	Capture mode register 1	FEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]		0000,0000
	PWMB	Compare mode register 1		IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]		0000,0000
PWMB_CCMR2	PWMB	Capture mode register 2	FEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]		0000,0000
	PWMB	Compare mode register 2		IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]		0000,0000
PWMB_CCMR3	PWMB	Capture mode register 3	FEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]		0000,0000
	PWMB	Compare mode register 3		IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]		0000,0000
PWMB_CCMR4	PWMB	Capture mode register 4	FEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]		0000,0000
	PWMB	Compare mode register 4		IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]		0000,0000
PWMB_CCER1	PWMB	Capture comparison enable register 1	FEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWMB_CCER2	PWMB	Capture comparison enable register 2	FEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E	xx00,xx00
PWMB_CNTRH	PWMB	Counter high byte	FEEEH									0000,0000
PWMB_CNTRL	PWMB	Counter low byte	FEEFH									0000,0000
PWMB_PSCRH	PWMB	Prescaler high byte	FEF0H									0000,0000
PWMB_PSCRL	PWMB	Prescaler low byte	FEF1H									0000,0000
PWMB_ARRH	PWMB	Automatic reloading of register high bytes	FEF2H									0000,0000
PWMB_ARRL	PWMB	Automatic reloading of register low bytes	FEF3H									0000,0000
PWMB_RCR	PWMB	Repeat counter register	FEF4H									0000,0000
PWMB_CCR5H	PWMB	Compare capture register 5 High position	FEF5H									0000,0000
PWMB_CCR5L	PWMB	Compare the low bits of the capture register	FEF6H									0000,0000
PWMB_CCR6H	PWMB	Compare the high position of the capture register	FEF7H									0000,0000

PWMB_CCR6L	PWMB	Compare capture register - Low position	FEF8H							0000,0000
PWMB_CCR7H PWMB		Compare capture register - High position	FEF9H							0000,0000
PWMB_CCR7L PWMB		Compare the low bits of the capture register	FEFAH							0000,0000
PWMB_CCR8H PWMB		Compare the high position of the capture register	FEFBH							0000,0000
PWMB_CCR8L PWMB		Compare the low bits of the capture register	FEFCH							0000,0000
PWMB_BKR	PWMB	Brake register	FEFDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]	-	0000,000x
PWMB_DTR	PWMB	Dead zone control register	FEFEH	DTG[7:0]								0000,0000
PWMB_OISR	PWMB	Output idle status register	FEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5	x0x0,x0x0
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 Control Register	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU		FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000
P0INTE	P0	Port interrupt enable register	FD00H	P07INTE	P06INTE P05INTE P04INTE P03INTE P02INTE					P01INTE	P00INTE 0000,0000	
P1INTE	P1	Port interrupt enable register	FD01H	P17INTE	P16INTE P15INTE P14INTE P13INTE P12INTE					P11INTE	P10INTE 0000,0000	
P2INTE	P2	Port interrupt enable register	FD02H	P27INTE	P26INTE P25INTE P24INTE P23INTE P22INTE					P21INTE	P20INTE 0000,0000	
P3INTE	P3	Port interrupt enable register	FD03H	P37INTE	P36INTE P35INTE P34INTE P33INTE P32INTE					P31INTE	P30INTE 0000,0000	
P5INTE	P5	Port interrupt enable register	FD05H	P57INTE	P56INTE P55INTE P54INTE P53INTE P52INTE					P51INTE	P50INTE 0000,0000	
P0INTF	P0	Port interrupt flag register	FD10H	P07INTF	P06INTF	P05INTF P04INTF P03INTF P02INTF				P01INTF	P00INTF 0000,0000	
P1INTF	P1	Port interrupt flag register	FD11H	P17INTF	P16INTF	P15INTF P14INTF P13INTF P12INTF				P11INTF	P10INTF 0000,0000	
P2INTF	P2	Port interrupt flag register	FD12H	P27INTF	P26INTF	P25INTF P24INTF P23INTF P22INTF				P21INTF	P20INTF 0000,0000	
P3INTF	P3	Port interrupt flag register	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF P32INTF		P31INTF	P30INTF 0000,0000	
P5INTF	P5	Port interrupt flag register	FD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF P52INTF		P51INTF	P50INTF 0000,0000	
P0IM0	P0	Port interrupt mode register	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1	Port interrupt mode register 0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2	Port interrupt mode register 0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3	Port interrupt mode register 0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P5IM0	P5	Port interrupt mode register 0	FD25H	P57IM0	P56IM0	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	0000,0000
P0IM1	P0	Port interrupt mode register 1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1	Port interrupt mode register 1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2	Port interrupt mode register 1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3	Port interrupt mode register 1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P5IM1	P5	Port interrupt mode register 1	FD35H	P57IM1	P56IM1	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	0000,0000
CCAPM3	PCA 3	Module mode control register	FD54H	-	ECOM3	CCAPP3 CCAPN3		MAT3	TOG3	PWM3	ECCF3	x000,0000
CCAP3L	PCA 3	Module low byte	FD55H									0000,0000
CCAP3H	PCA 3	Module high byte	FD56H									0000,0000
PCA_PWM3	PCA3	of PWM Mode register	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L	0000,0000

Note: The meaning of the initial value of the special function register

0: The initial value is 0;

1: The initial value is 1;

n: Initial value and ISP Hardware options when downloading are related ;

:

x This bit does not exist, the initial value is uncertain

9 I/O mouth

All of the series of microcontroller ports have a variety of working modes: quasi-two-way port/weak pull-up (standard 8051 output port), Push-pull output/Strong pull-up, high resistance input (current can neither flow in nor out), open-drain mode. The formula can be configured using software.

I/O The working mode of the mout

And not except P3.0 and P3.1, after the reset I/O The state of the port after power-on is a high-impedance input state, and the user is usi use I/O Must be set first when opening the mouth Mouth mode I/O

9.1 I/O Port-related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 port	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
P1	P1 port	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	P2 port	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	P3 port	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P5	P5 port	C8H	P57	P56	P55	P54	P53	P52	P51	P50	1111,1111
P0M1	P0 Port configuration register	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	P0 Port configuration register 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P1M1	P1 Port configuration register	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	P1 Port configuration register	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P2M1	P2 Port configuration register	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	P2 Port configuration register	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
P3M1	P3 Port configuration register	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	1111,1100
P3M0	P3 Port configuration register	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	0000,0000
P5M1	P5 Port configuration register 1	C9H	P57M1	P56M1	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	1111,1111
P5M0	P5 Port configuration register	CAH	P57M0	P56M0	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	0000,0000

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0PU	P0 Port pull-up resistor control register	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 Port pull-up resistor control register	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 Port pull-up resistor control register	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 Port pull-up resistor control register	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P5PU	P5 Port pull-up resistor control register	FE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	0000,0000
P0NCS	P0 Port Schmidt trigger control register	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 Port Schmidt trigger control register	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 Port Schmidt trigger control register	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 Port Schmidt trigger control register	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P5NCS	P5 Port Schmidt trigger control register	FE1DH	P57NCS	P56NCS	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	0000,0000
P0SR	P0 Port level conversion rate register	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	P1 Port level conversion rate register	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	P2 Port level conversion rate register	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	P3 Port level conversion rate register	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P5SR	P5 Port level conversion rate register	FE25H	P57SR	P56SR	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	1111,1111
P0DR	P0 Port drive current control register	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 Port drive current control register	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 Port drive current control register	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 Port drive current control register	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P5DR	P5 Port drive current control register	FE2DH	P57DR	P56DR	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	1111,1111
P1IE	P1 Port input enable control register	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P2IE	P2 Port input enable control register	FE32H	-	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE	x111,1111

9.1.1 Port data register (Px)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P5	C8H	P5.7	P5.6	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0

Read and write port status

Write: Output low level to the port buffer

Write: Output a high level to the port buffer

1

Read: Directly read the level on the port pin

9.1.2 Port mode configuration register (PxM0 , PxM1)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P0M1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P1M0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P1M1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P2M0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P2M1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P3M0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P3M1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P5M0	CAH	P57M0	P56M0	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0
P5M1	C9H	P57M1	P56M1	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1

Configure the mode of the port

PnM1. x	PnM0. x	Pn. x	Port working mode
0	0		Quasi-bidirectional
0	1		port push-pull output
1	0		, high resistance
1	1		input , open-drain mode

Note: When an I/O port is selected as the ADC input channel, the PxM0/PxM1 register must be set to set the I/O port mode to the input mode. In addition, if the MCU still needs to enable the ADC channel after entering the power-down mode/clock stop mode, you need to set the PXIe register to turn off the digital input to ensure that there will be no additional power consumption.

9.1.3 Port pull-up resistor control register (PxPU)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P5PU	FE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU

Inside the port 4.1K Pull-up resistor control bit (note : P3.0 and P3.1 The pull-up resistor on the port may be slightly smaller)

0 : Prohibit internal ports Pull-up resistor 4.1K

1 : Enable the internal port Pull-up resistor

9.1.4 Port Schmidt trigger control register (PxNCS)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS
P1NCS	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS
P2NCS	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS
P3NCS	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS
P5NCS	FE1DH	P57NCS	P56NCS	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS

Port Schmidt trigger control bit

0 : (Schmitt trigger is enabled by default after power-on reset)

1 Enable The Schmidt trigger function of the port.

: forbidden The Schmidt trigger function of the port.

9.1.5 Port level conversion speed control register (PxSR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0	Reset
P0SR	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	value 1111,1111
P1SR	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P5SR	FE25H	-	-	-	P54SR	P53SR	P52SR	P51SR	P50SR	1111,1111

Control the speed of port level conversion

0 : The level conversion speed is fast, and the corresponding up and down impulse will be relatively large

1 : the level conversion speed is slow, and the corresponding up and down impulse is relatively small.

9.1.6 Port drive current control register (PxDR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0	Reset
P0DR	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	value 1111,1111
P1DR	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P5DR	FE2DH	-	-	-	P54DR	P53DR	P52DR	P51DR	P50DR	1111,1111

Control the driving ability of the port

0: Enhance driving ability

1: General driving ability

9.1.7 Port digital signal input enables the control register (PxIE)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P1IE	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE
P2IE	FE32H	-	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE

Digital signal input enables control

: Digital signal input is prohibited. If 0 Is used as a comparator input port, When entering an analog port such as an input port or a touch bu

Before entering the clock stop mode, it must be set to, otherwise there will be additional

power consumption. : Enable digital signal input. If 1 When used as a digital port, it must be set, otherwise the external port cannot be read.

9.2 configuration I/O mouth

each I/O The configuration needs to be set using two registers.

To P0 Take the port as an example, the configuration port needs to use and P0M0 P0M1 The two registers are configured, as shown in the figure below :

That is, all others I/O

P0M0 0 The first bit of the configuration is combined P0.0 Mode of

P0M0 1 and the first bit of the configuration is combined and configured P0.1 mouth mode

The configuration is similar to this.

PnM0 with PnM1 The combination method is shown in the following table

PnM1	PnM0	Port working mode I/O
0	0	Quasi-two-way port (traditional 8051 Port mode, weak pull-up) Sink current up to 20mA, the pull current is 270 ~ 150μA (There is a manufacturing error)
0	1	Push-pull output (strong pull-up output, up to 20mA, To add a current limiting resistor)
1	0	High-resistance input (current can neither flow in nor out) open-drain mode (Open-Drain), the internal pull-up resistor is disconnected
1	1	Open-drain mode can read both the external state and the external output (high level or low level)。If you want to read the external state correctly or need to output a high level externally, you need to add a pull-up resistor, otherwise the external state cannot be read, and the external output cannot be high. === [Open-drain operating mode], the external output is set to, which is equivalent to [high impedance input] 【Open-drain working mode, the internal pull-up resistor ===, Simply equivalent to [quasi-two-way port] External plus pull-up resistor】

n note : = 0,1,2,3,4,5,6,7

attention:

although each I/O The mouth is weakly pulled up (quasi-two-way mouth) / Strong push-pull output / Can switch to open-drain mode and current limit Resistance, such as 560Ω, Etc.) / Can output during strong push-pull output ,

The working current is recommended not to exceed 70mA, However, the outflow current of the entire chip is recommended not to exceed 70mA , The pull current (also add a current limiting resistor)

Overall inflow It is recommended that the outflow current should not exceed 70mA , the current flowing in 70mA Vss

9.3 Structure diagram of I/O

9.3.1 Quasi-two-way port (weak pull-up)

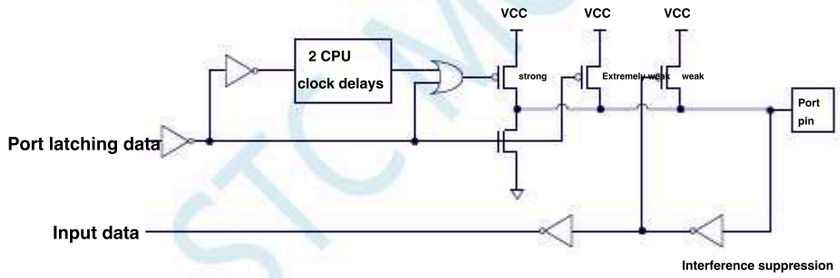
The quasi-bidirectional port (weak pull-up) output type can be used as an output and input function without the need to configure the po state. This is because the drive capability is weak when the port output is, allowing an external device to pull it down. When the pin output is strong driving ability and can absorb considerable current. The quasi-bidirectional port has a pull-up transistor to meet different needs.

Among the pull-up transistors, there is a pull-up transistor is called a "weak pull-up" And the pin itself is also played from time to open. This pull-up provides the basic drive current so that the quasi-bidirectional port outputs a logic 1. When the output is pulled down to low by an external device, The weak pull-up is turned off and the "extremely weak pull-up" remains on. In order to strongly pull this pin low, the external device must have sufficient sink current to drop the voltage on the pin below the threshold of the "weak pull-up" transistor. For a single-chip microcomputer, the current of the weak pull-up transistor is about Chip machine, the current of the "weak pull-up" transistor is about 150uA。

A pull-up transistor is called a "very weak pull-up". It is latched as When the pin is A weak pull-up current is generated to pull the pin high. For floating, this current of the "extremely weak pull-up" transistor is about For example, the current of the "very weak pull-up" transistor is about. The first pull-up transistor is called a "strong pull-up". It is used When the port latch changes to transition, this pull-up is used to speed up the quasi-b Edit to logical conversion. When this happens, a strong pull-up opens about pull up a clock so that the pin can be quickly pulled up to a high level.

The quasi-bidirectional port (weak pull-up) has a Schmidt trigger input and an interference suppression circuit. Quasi-two-way port (weak pull-up) before reading the external state, Must first be latched as '1', Only then can the correct external state be rea

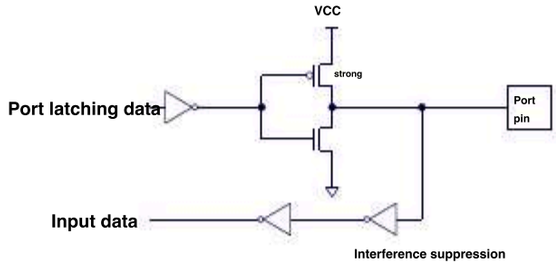
The output of the quasi-bidirectional port (weak pull-up) is shown in the figure below：



9.3.2 Push-pull output

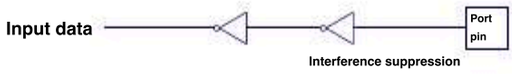
The pull-down structure of the strong push-pull output configuration is the same as that of the open-drain mode and the pull-down struc it provides a continuous strong pull-up when the latch is on. Push-pull mode is generally used in situations where greater drive current is req

The strong push-pull pin configuration is shown in the figure below：



9.3.3 High impedance input

Current can neither flow in nor outThe energy-flowing input port has a Schmidt trigger input and an interference suppression circuit . The high-resistance input pin configuration is shown in the figure below. :



9.3.4 Open-drain mode

[Open-drain operating mode], the external output is set to, which is equivalent to [high impedance input]

1 , Simply equivalent to [quasi-two-way port]

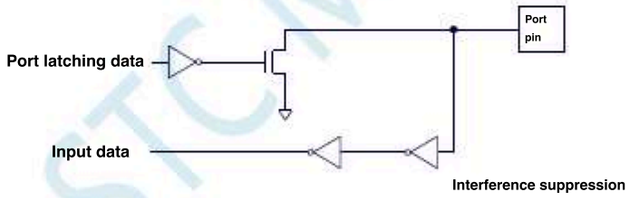
[Open-drain working mode] [turn on the internal pull-up resistor or the external pull-up resistor] ea l

The open-drain mode can read both the external state and the 。 If you want to read the external status correctly or need to outp external output (high or low) level , and a pull-up resistor is required.

When the port latch is When, the open-drain mode turns off all pull-up transistor. When the output is high as a logic, this configuration m The formula must have an external pull-up, and it is generally connected to the pull-up resistor, the open-drain also read the external status , that is, the port that is configured as an open-drain mode at this time The drop-down is the same as the quasi-bidirectional port.

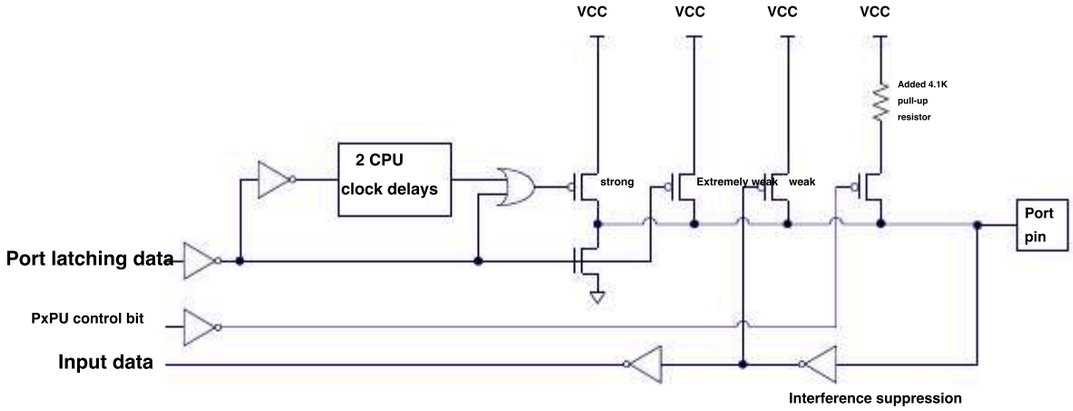
The open-drain port has a Schmidt trigger input and an interference suppression circuit.

The output port configuration is shown in the figure below :



9.3.5 new 4.1K Pull-up resistor

All of the series I/O One can be enabled inside the mouth 4.1K Approximately Pull-up resistor (due to manufacturing errors, the range of the pull-up resistor may be 3K ~ 5K)



Port pull-up resistor control register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P5PU	FE15H	P57PU	P56PU	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU

Inside the port 4.1K Pull-up resistor control bit (note : P3.0 and P3.1 The pull-up resistor on the port may be slightly smaller)

0 : Prohibit internal ports Pull-up resistor 4.1K

1 : Enable the internal port Pull-up resistor

9.3.6 How to set the external output speed of the port I/O

When the user needs I/O When the port outputs a faster frequency externally, it can be increased by I/O Port drive current and increase Port level conversion speed to achieve improvement The port outputs a speed register externally,

Set up PxSR which can be used to control I/O Port level conversion speed, When set to the corresponding I/O The output is flipped quickly, set to 1 The time is a slow flip.

Set up PxDR Register, can be used to control I/O The size of the port drive current is set to 1 when I/O The output is the general driving current, set to 0 When it is a strong driving current

9.3.7 How to set the port current drive capability I/O

If you need to change The current drive capability of the port can be set by I/O Register to achieve

the settings Pxn Register, can be used to control I/O The size of the port drive current is set to The output is the general driving current, set to

When it is a strong driving current

9.3.8 How to reduce external radiation from the mouth I/O

Due to the setting of the PnSR Register, can be used to control I/O Port level conversion speed, set PxDR Register, can be used to control I/O Oral drive

dynamic current size

When it is necessary When radiating from the outside of the mouth, it is necessary to reduce I/O PxSR Port level conversion speed, at the same time

to reduce the PxDR The register is set to reduce I/O The driving current is finally achieved to reduce the external radiation of the port I/O

9.4 Sample program

9.4.1 Port mode setting

C Language code

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

#include "reg51. h"
#include "intrins. h"
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 P4M0 = 0xb4;
sfr P4M1 = 0xb3;
sfr P5M0 = 0xca;
sfr P5M1 = 0xc9;
sfr P6M0 = 0xcc;
sfr P6M1 = 0xcb;
sfr P7M0 = 0xe2;
sfr P7M1 = 0xe1;
void main()
{
    P0M0 = 0x00;    //Set up P0.0~P0.7 Two-way port mode
    P0M1 = 0x00;
    P1M0 = 0xff;    //Set up P1.0~P1.7 Push-pull output mode
    P1M1 = 0x00;
    P2M0 = 0x00;    //Set up P2.0~P2.7 High impedance input mode
    P2M1 = 0xff;
    P3M0 = 0xff;    //Set up P3.0~P3.7 Open-drain mode
    P3M1 = 0xff;

    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

```
P5M1            DATA    0C9H
P6M0            DATA    0CCH
P6M1            DATA    0CBH
P7M0            DATA    0E2H
P7M1            DATA    0E1H

                ORG     0000H
                LJMP    MAIN

                ORG     0100H
MAIN:
                MOV     SP, #5FH

                MOV     P0M0,#00H       ;Set up P0.0~P0.7   Two-way port mode
                MOV     P0M1,#00H
                MOV     P1M0,#0FFH      ;Set up P1.0~P1.7   Push-pull output mode
                MOV     P1M1,#00H
                MOV     P2M0,#00H       ;Set up P2.0~P2.7   High impedance input mode
                MOV     P2M1,#0FFH
                MOV     P3M0,#0FFH      ;Set up P3.0~P3.7   Open-drain mode
                MOV     P3M1,#0FFH

                JMP     $

                END
```

9.4.2 Two-way port read and write operation

C Language code

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

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

sfr P0M0        =   0x94;
sfr P0M1        =   0x93;
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 P00        =   P0^0;

void main()
{

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
```

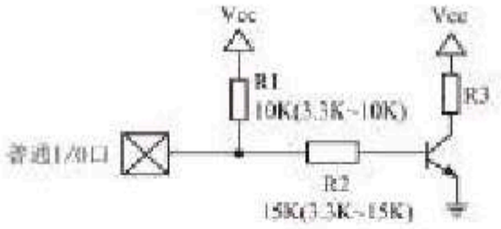
```
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;                    //Set up P0.0~P0.7 Two-way port mode
    P0M0 = 0x00;
    P0M1 = 0x00;
    P00 = 1;                        //P0.0 Port output high level
    P00 = 0;                        //P0.0 port output low level
    P00 = 1;
    _nop_();                        // Enable the internal weak pull-up resistor before reading the port
    _nop_();                        //Wait for two clocks
    CY = P00;                       //
    while (1);                      //Read port status
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

P0M0    DATA    094H
P0M1    DATA    093H
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
```

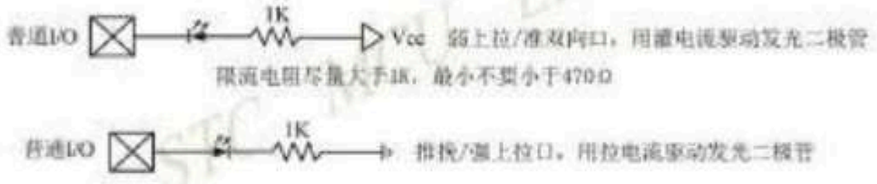
```
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     P0M0,#00H       ;Set up P0.0~P0.7 Two-way port mode
MOV     P0M1,#00H

SETB    P0.0            ;P0.0 Port output high level
CLR     P0.0            ;P0.0 port output low level

SETB    P0.0            ; Enable the internal weak pull-up resistor before reading the port
NOP                     ;Wait for two clocks
NOP
MOV     C,P0.0          ;Read port status

JMP     $

END
```

9.5 A typical triode control circuit



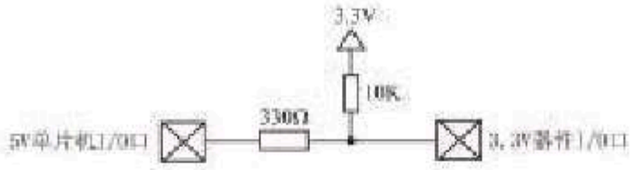
If the pull-up control is controlled, it is recommended to add a pull-up resistor or use a strong push-pull output. , If you do not add the pull-up resistor R1 It is recommended that the value of R2 is

9.6 Typical light-emitting diode control circuit



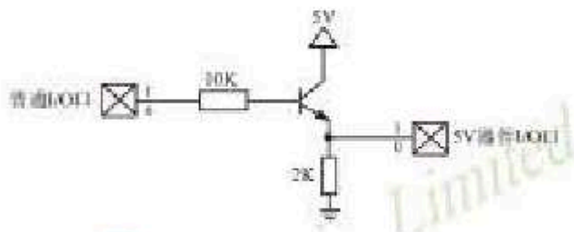
9.7 Hybrid voltage power supply system device I/O Port interconnection

When STC's 5V single-chip microcomputer is connected to a 3.3V device, in order to prevent the 3.3V device from not being able to withstand 5V, the corresponding I/O port can be connected with a 330Ω current limiting resistor to the 3.3V device I/O port. When the program is initialized, the I/O port of the 5V single-chip microcomputer configuration, disconnect the internal pull-up resistor, and add a 10K pull-up resistor to the Vcc of the 3.3V device outside the I/O port of the corresponding 3.3V device, s 0V, and everything is normal for the input and output.



STC When the 3V MCU is connected to a 5V device, in order to prevent the 3V MCU from not being able to withstand 5V, if the corresponding I/O po An isolation diode is connected in series on the port to isolate the high voltage part. The external signal voltage is cut off when it is higher than the operat port is pulled up to a high level due to the internal pull-up, so the state of the read I/O port is high; when the external signal voltage is low, it is turned on, a 0.8VThe state of the MCU's read I/O port is low.

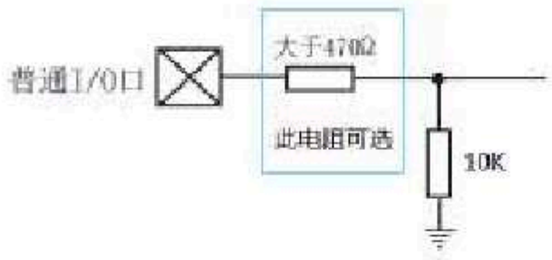
When STC's 3V MCU is connected to a 5V device, in order to prevent the 3V MCU from not being able to withstand 5V, if the corresponding I/O port is the output, an NPN transistor can be used to isolate it. The circuit is as follows：



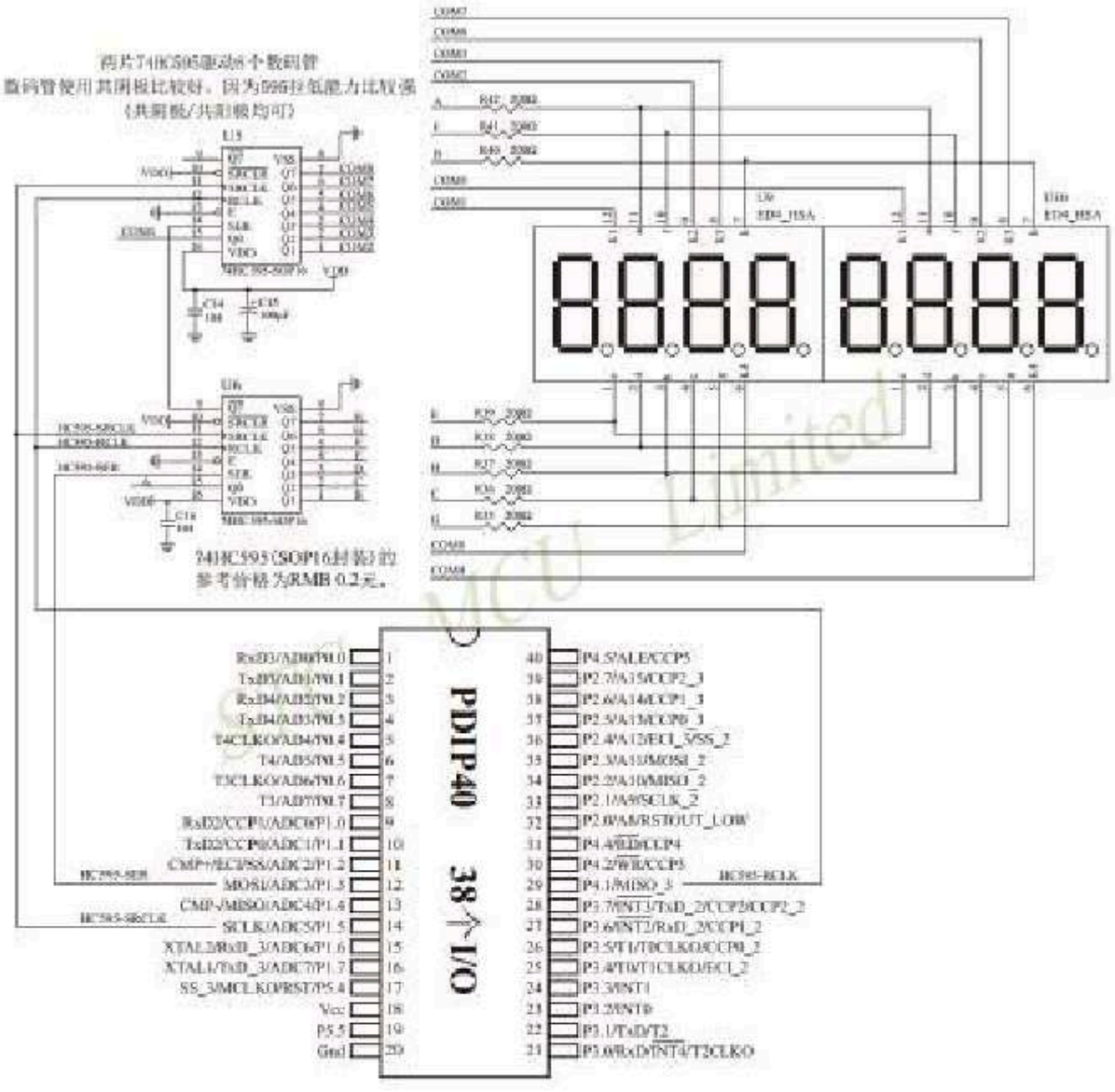
9.8 How to make I/O The port is low when it is powered on and reset

When the ordinary 8051 microcontroller is reset on power-up, the ordinary I/O port is a weak pull-up (quasi-bidirectional port) with a high-level outpu practical applications require some I/O ports to be low-level outputs at power-up, otherwise the controlled system (such as a motor) will malfunction. ,Now microcomputer has both a weak pull-up output and a strong push-pull output, this problem can be easily solved.

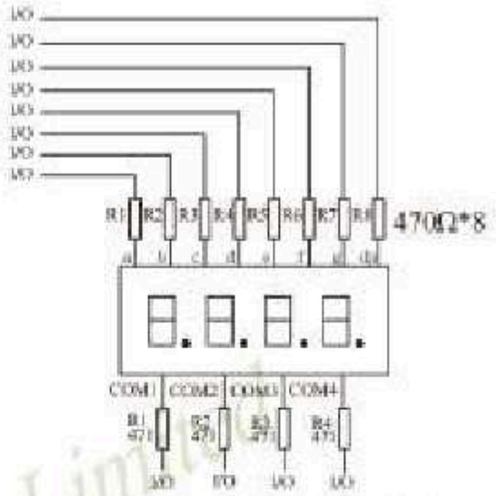
It is now possible to add a pull-down resistor (about 10K) to the I/O port of STC's single-chip microcomputer, so that when the power-on is reset, exc , which are weak pull-ups (quasi-bidirectional ports), the other I/O ports are all high-impedance input modes., And there is an external pull-down resistor, s I/O port is powered on and reset . If you want to drive this I/O port to a high level, you can set this I/O port to a strong push-pull output. When the push-pul the I/O port can reach 20mA, so it is definitely possible to drive this port to a high level output.



9.9 use 74HC595 drive 8 Serial expansion of a digital tube (3 wire) Circuit diagram of the root line



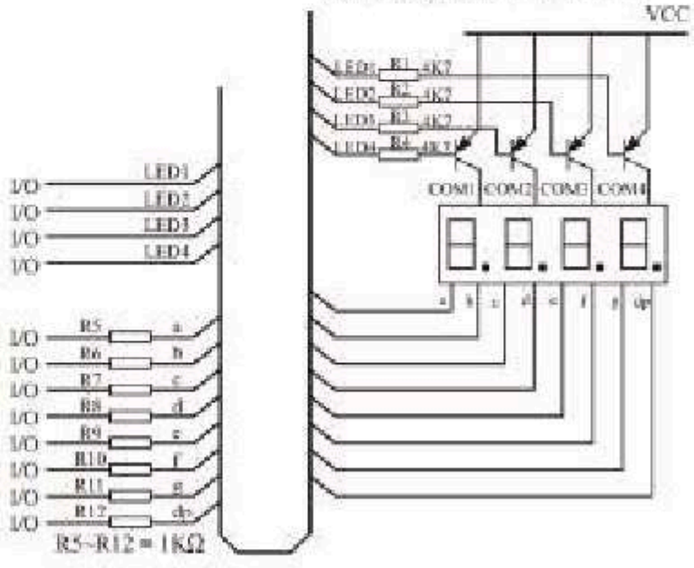
9.10 I/O Port direct drive LED Digital tube application circuit diagram



I/O口动态扫描驱动4个共阴极数码管参考电路图

I/O 口动态扫描驱动数码时，可以一次点亮一个数码管中的8段，但为降低功耗，建议可以一次只点亮其中的4段或者2段

I/O 口动态扫描驱动4个共阳极数码管参考电路图



9.11 use STC series MCU of I/O The port directly drives the LCD segment code

When the product needs a segment display, if you use without Drive drive IC, This will LCD LCD MCU, you n
Increase costs. In fact, for many small projects, such as a large number of small appliances, there are not many 8 segments that need to be dis
A colon with a decimal point or clock ":", so that if you use the port to scan and display directly, the cost will be reduced and the work will be

However, this scheme is not suitable for driving IO Too many), it is not suitable for very low-power occasions (there will be hundre
uA too many segments (occupying current)

Segment code LCD Simple driving principle: as shown in the figure

LCD It is a special kind of liquid crystal. Under the action of an electric field, the arrangement direction of the crystal will be reversed, t
And you can see the display content. There is a torsion voltage threshold, When the voltage across the terminal is higher than this threshold, th
When the threshold value is reached, it is not displayed normally. LCD 3 COM Number) (That is, bias, corresponding to
Threshold), such as 3.0V、 1/4 DUTY, 1/3 BIAS, means LCD DUTY (The corresponding and 3.0V, COM, The threshold value is approximately 1.5V,
When the voltage applied across a certain section is 3.0V When displayed, and added 1.0V display voltage is The response to the driving voltage is not very go
Sensitive, such as adding When it is, it may be faintly displayed, which is commonly referred to as not displayed when it is. But "ghosting." Therefore, it is necessary to ensu
The value voltage is relatively large, but when it is not displayed, a voltage that is much smaller than the threshold value should be used.

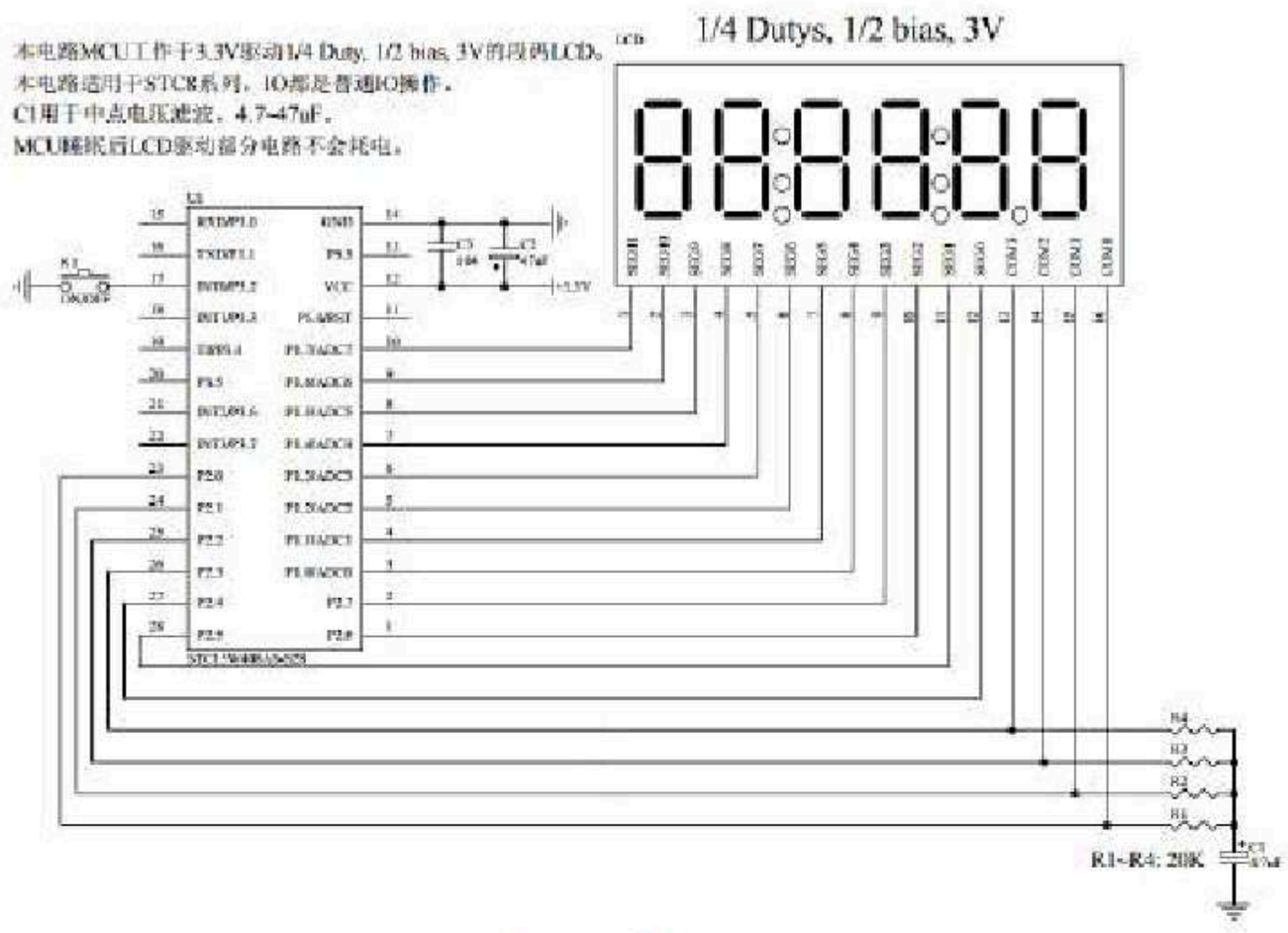
attention : LCD To be driven by AC, a DC voltage cannot be applied at both ends, otherwise it will be damaged after a long time, so make s
The average voltage of the driving voltage across the When using this division scanning method, at any time LCD0 COM, valid, in addition
In an invalid state.

Driven The circuit of the scheme is shown in the figure , LCD1 The scanning principle is shown in the figure . After work, and or 3.0V
and each COM When all in a series 20K 1/4Duty 1/2BIAS 3V The resistor is connected to a capacitor C1 4.2V 2.5V 3V filtering, a midpoint voltage is obtained and set to a high resistance. If it In the turn of a certain
COM of scans, the connected Set to push-pull output, the rest COM is the same as this one, the scanning principle is shown in the figure. connected COM not displayed ,
Then SEG In-phase, if displayed, then reversed. After scanning, this output is the same as COM of IO It is set to high resistance, each COM
pass 20K The resistor is connected to the voltage on the capacitor, and According to whether the output high and low levels are displayed, thi LCD
The voltage on the segment is displayed as +VDD When not displayed is +1/2VDD, guaranteed LCD The average DC voltage across both ends is 0

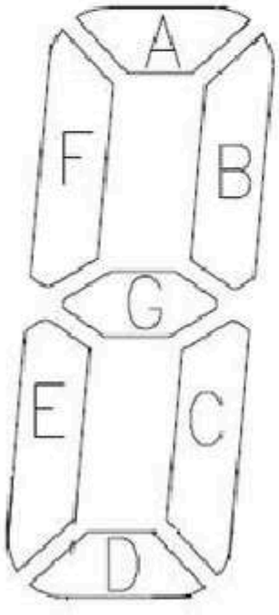
Drive the over-resistance The circuit of the scheme is shown in the figure , The scanning principle is shown in the figure , MCU work , SEG , Pass
voltage divider output LCD4 Connected to the voltage divider output through a resistor for 5V Picked up IO
The common point is connected to a COM1.5V、 3.5V , IO 1/2VDD° In the turn of a certain 2.0V 4.5V (At high resistance) When the voltage divider resistor COM is scanned, it is set to push-pull
capacitive output, if it is connected to this output displayed, the connected In-phase, if displayed, then reversed. After scanning , COM
This one COM IO one is set to high impedance, so this COM The resistor is connected to the voltage, depending on whether it is
Show the high and low output levels, so the The voltage on the display is +3.0V, When not displayed is +-1.0V, fully satisfied LCD sweep
it is added to the description requirements.

When you need to sleep to save power, put all COM and SEG drive IO All outputs are low , LCD No additional power will be added to the drive p
flow.

1 Picture: Driver 4Duty 1/2BIAS 3V LCD The circuit

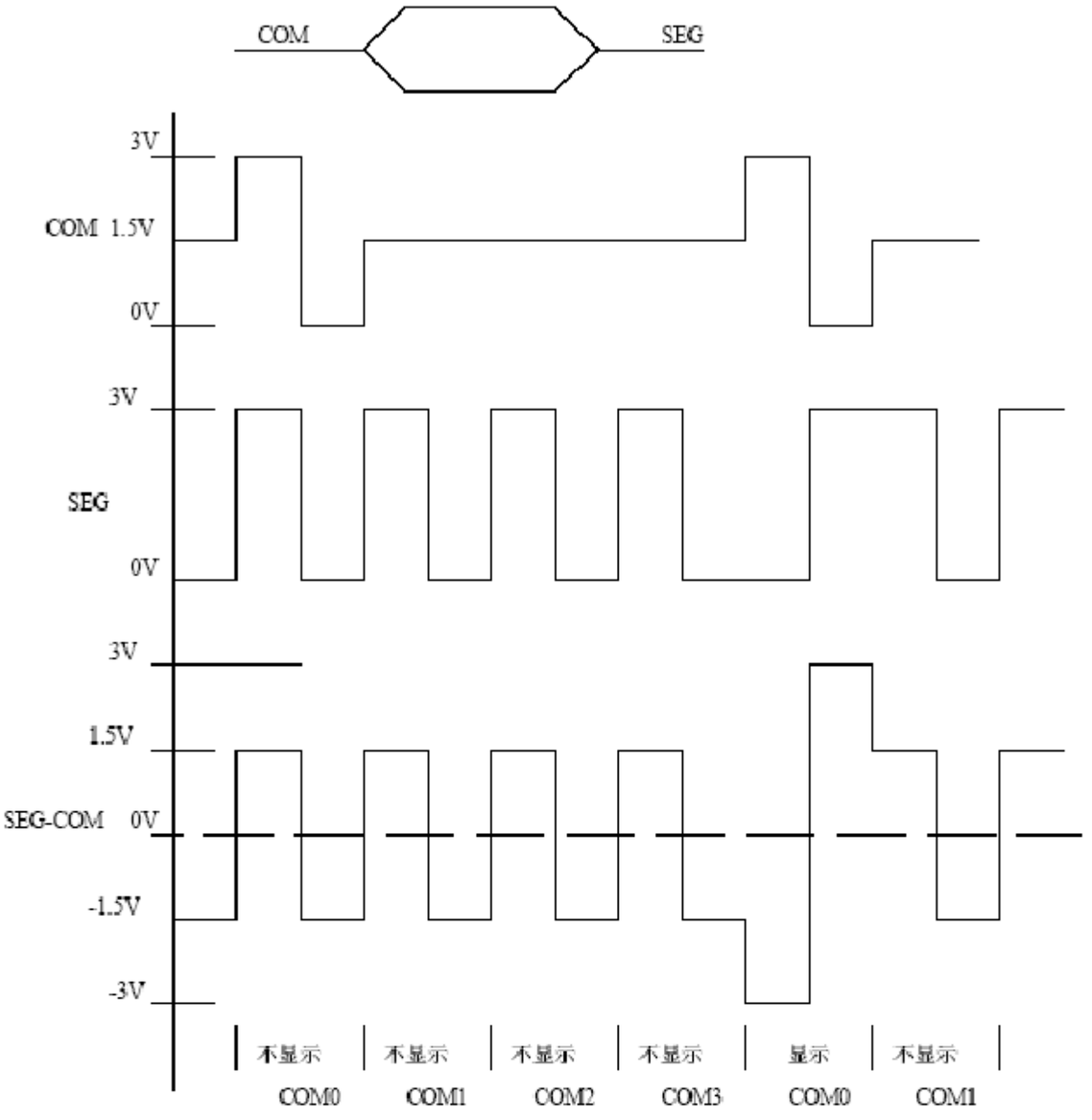


2 Picture: Segment code name picture



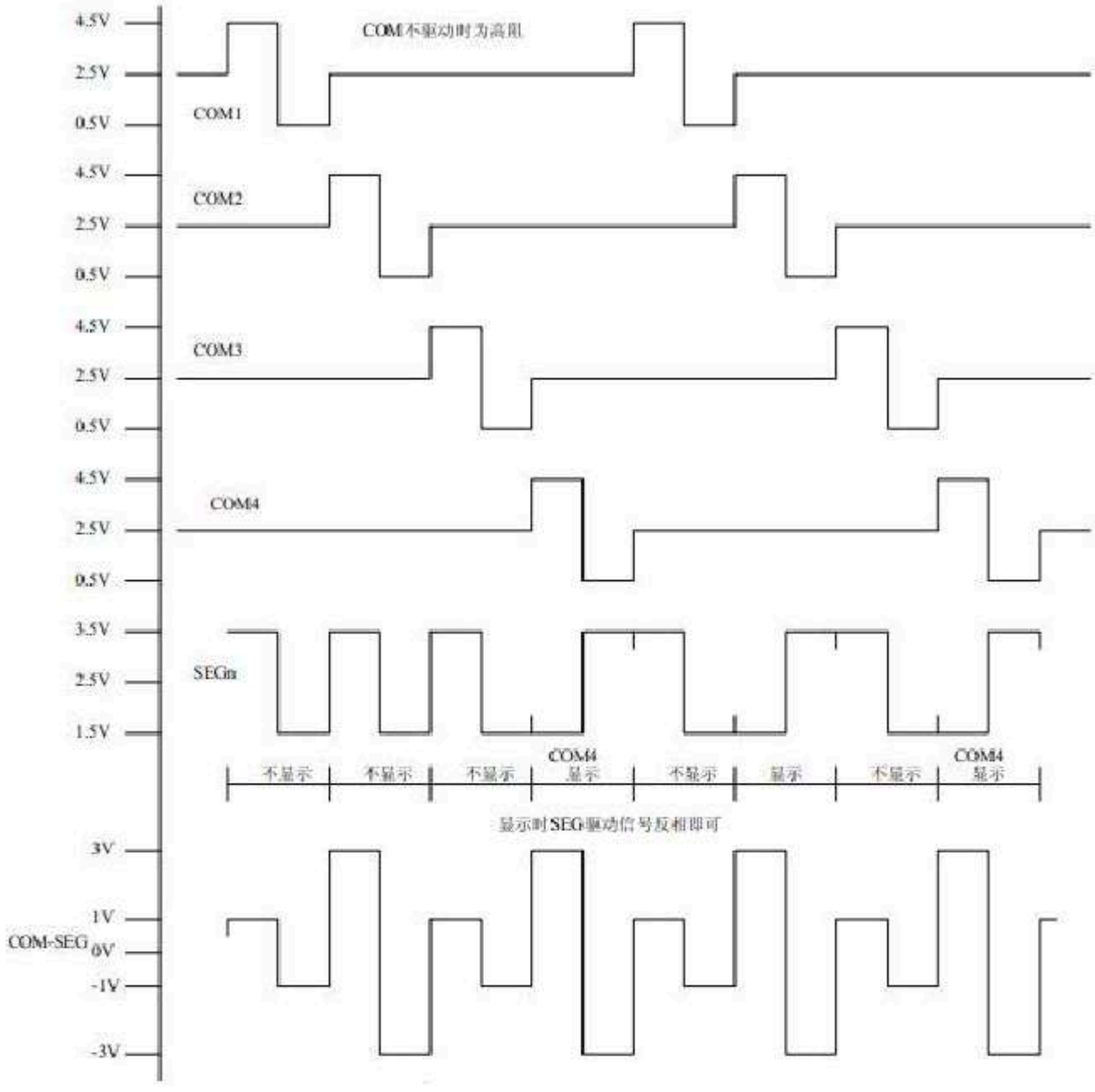
3 figure : 1/4Duty 1/2BIAS

Scanning schematic



4 Picture: Driver 4Duty 1/3BIAS 3V LCD The circuit

5figure ： 1/4Duty 1/3BIAS **Scanning schematic**



For ease of use, the display content is placed in a video memory, each bit of which is the same as the LCD The segment corresponds one by one, see the figure.6

6figure ： LCD Truth table and video memory innuendo table

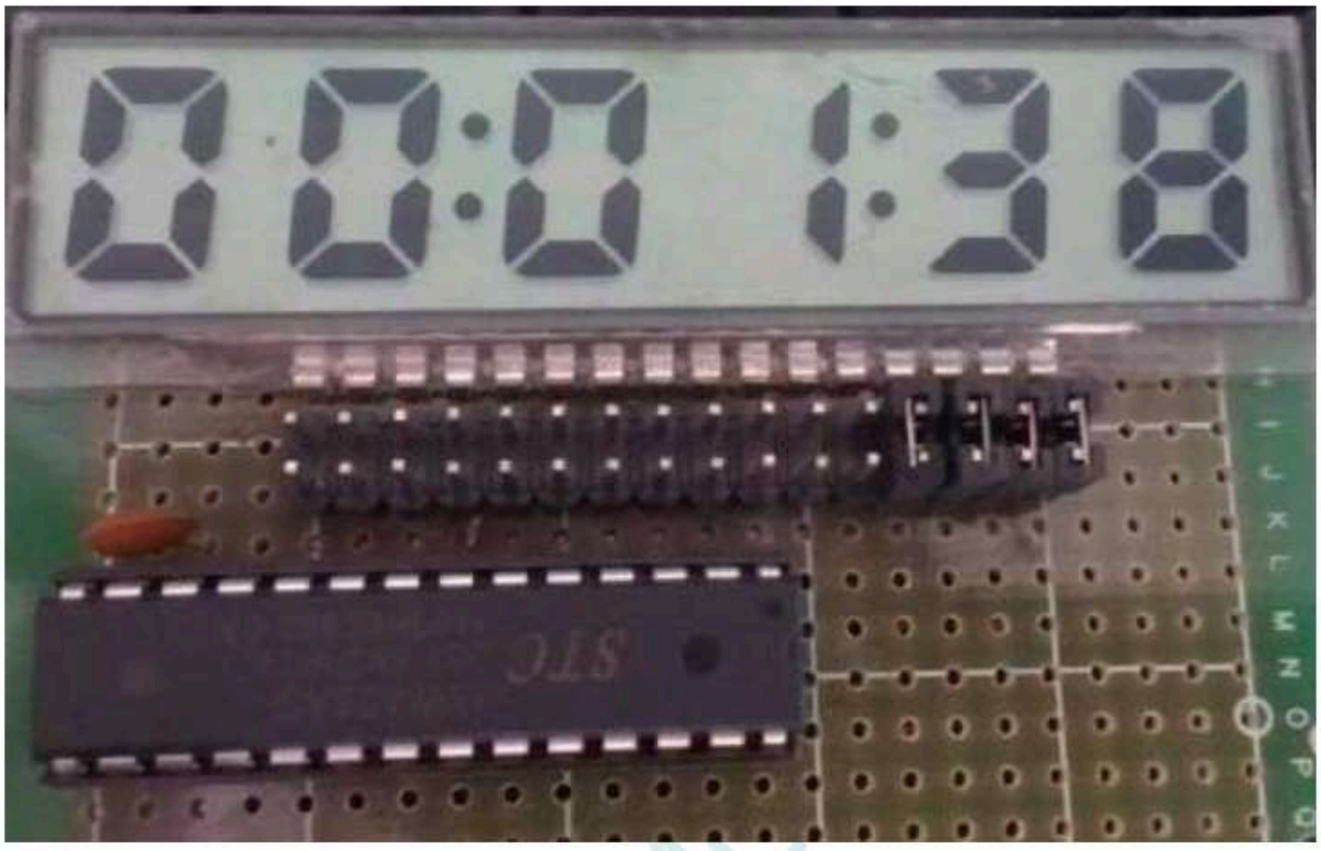
LCD真值表：

MCU PIN	P17	P16	P15	P14	P13	P12	P11	P10	P27	P26	P25	P24	P23	P22	P21	P20
LCD PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LCD PIN name	SEG11	SEG10	SEG9	SEG8	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	SEG0	COM3	COM2	COM1	COM0
	--	1D	2:	2D	2.	3D	4:	4D	4.	5D	5.	6D	COM3			
	1E	1C	2E	2C	3E	3C	4E	4C	5E	5C	6E	6C		COM2		
	1G	1B	2G	2B	3G	3B	4G	4B	5G	5B	6G	6B			COM1	
	1F	1A	2F	2A	3F	3A	4F	4A	5F	5A	6F	6A				COM0

显存影射表：

	B7	B6	B5	B4	B3	B2	B1	B0
buff[0]:	--	1D	2:	2D	2.	3D	4:	4D
buff[1]:	1E	1C	2E	2C	3E	3C	4E	4C
buff[2]:	1G	1B	2G	2B	3G	3B	4G	4B
buff[3]:	1F	1A	2F	2A	3F	3A	4F	4A
buff[4]:	5.	5D	5.	6D	--	--	--	--
buff[5]:	5E	5C	6E	6C	--	--	--	--
buff[6]:	5G	5B	6G	6B	--	--	--	--
buff[7]:	5F	5A	6F	6A	--	--	--	--

Picture: Driving effect photo



The scanner only needs two functions :

LCD LCD

Code scanning function of this paragraph

The contents of the display buffer, all displayed one at a time

1、 void LCD_scan(void)

Call cycle, the call interval is generally If you use , Then the scan cycle is , The refresh rate is LCD

The program calls this function every certain time, and it will

Segment code display buffer loading function

This function is used to put the displayed numbers or characters in , Just to be in the first number position

In the display buffer, such as

Display numbers, support display A~F , Other characters users can add by themselves.

In addition, use a macro to display, turn off, or flash a colon or decimal point.

C Language code

```
/*****************Series of tests for function*****************

description STC15 I/O Direct drive segment code 8 word LCD, 1/4 Duty, 1/3 bias)

Display a time after power-up (Minutes and seconds).

P3.2 Connect a switch to ground, Used to go to sleep or wake up.

*****************************************/


#include "reg51.h"

#include "intrins.h"

typedef unsigned char u8;

typedef unsigned int u16;

typedef unsigned long u32;

sfr AUXR = 0x8e;

sfr P1M1 = 0x91;
```

```
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;

/*****************Local constant declaration *****************************/
#define MAIN_Fosc                   11059200L           //Define the master clock

#define DIS_BLACK                   0x10
#define DIS_                        0x11
#define DIS_A                       0x0A
#define DIS_B                       0x0B
#define DIS_C                       0x0C
#define DIS_D                       0x0D
#define DIS_E                       0x0E
#define DIS_F                       0x0F

#define LCD_SET_DP2                 0x08
#define LCD_CLR_DP2                 LCD_buff[0] |= LCD_buff[0] &= ~0x08
#define LCD_FLASH_DP2               LCD_buff[0] ^=  0x08

#define LCD_SET_DP4                 0x80
#define LCD_CLR_DP4                 LCD_buff[4] |= LCD_buff[4] &= ~0x80
#define LCD_FLASH_DP4               LCD_buff[4] ^=  0x80

#define LCD_SET_2M                  0x20
#define LCD_CLR_2M                  LCD_buff[0] |= LCD_buff[0] &= ~0x20
#define LCD_FLASH_2M                LCD_buff[0] ^=  0x20

#define LCD_SET_4M                  0x02
#define LCD_CLR_4M                  LCD_buff[0] |= LCD_buff[0] &= ~0x02
#define LCD_FLASH_4M                LCD_buff[0] ^=  0x02

#define LCD_SET_DP5                 0x20
#define LCD_CLR_DP5                 LCD_buff[4] |= LCD_buff[4] &= ~0x20
#define LCD_FLASH_DP5               LCD_buff[4] ^=  0x20

#define P1n_standard(bitn)          P1M1 &= ~(bitn), P1M0 &= ~(bitn)
#define P1n_push_pull(bitn)         P1M1 &= ~(bitn), P1M0 |= (bitn)
#define P1n_pure_input(bitn)        P1M1 |= (bitn), P1M0 &= ~(bitn)
#define P1n_open_drain(bitn)        P1M1 |= (bitn), P1M0 |= (bitn)

#define P2n_standard(bitn)          P2M1 &= ~(bitn), P2M0 &= ~(bitn)
#define P2n_push_pull(bitn)         P2M1 &= ~(bitn), P2M0 |= (bitn)
#define P2n_pure_input(bitn)        P2M1 |= (bitn), P2M0 &= ~(bitn)
#define P2n_open_drain(bitn)        P2M1 |= (bitn), P2M0 |= (bitn)

/*****************Local variable declaration *****************************/
u8 cnt_500ms;
u8 second,minute,hour;
bit B_Second;
bit B_2ms;
u8 LCD_buff[8];
u8 scan_index;

/*****************Local function declaration *****************************/
void LCD_load(u8 n,u8 dat);
void LCD_scan(void);
void LoadRTC(void);
void delay_ms(u8 ms);
```

```
/**********Main function **************************/
void main(void)
{
    u8    i;

    AUXR = 0x80;
    TMOD = 0x00;
    TL0 = (65536 - (MAIN_Fosc / 500));
    TH0 = (65536 - (MAIN_Fosc / 500)) >> 8;
    TR0 = 1;
    ET0 = 1;
    EA = 1;

                                                    //initialize LCD Video memory
    for(i=0; i<8; i++) LCD_buff[i] = 0;
    P2n_push_pull(0xf0);
                                                    //segment Set to push-pull output
    P1n_push_pull(0xff);
    LCD_SET_2M;                                     //Display time interval :
    LCD_SET_4M;                                     //Display minute and second intervals :
    LoadRTC();                                      //Display time
    while (1)
    {
        PCON |= 0x01;                               //Enter idle mode, by Timer0 2ms Wake up and exit
        _nop_();
        _nop_();
        _nop_();

        if(B_2ms)                                   //2ms Beat to
        {
            B_2ms = 0;

            if(++cnt_500ms >= 250)                  //500ms to
            {
                cnt_500ms = 0;
//              LCD_FLASH_2M;                       //Flashing time interval :
//              LCD_FLASH_4M;                       //Flashing minutes and seconds interval :

                B_Second = ~B_Second;
                if(B_Second)
                {
                    if(++second >= 60)              //1 Minutes to arrive
                    {
                        second = 0;
                        if(++minute >= 60)          //1 The hour is up
                        {
                            minute = 0;
                            if(++hour >= 24) hour = 0; //24 The hour is up
                        }
                    }
                    LoadRTC();                      //Display time
                }
            } if(!P32)                              //Press the key to prepare for sleep //Display time interval :
            {
                LCD_CLR_2M;
```

```
                    LCD_CLR_4M;                          //Display minute and second intervals :
                    LCD_load(1,DIS_BLACK);
                    LCD_load(2,DIS_BLACK);
                    LCD_load(3,0);
                    LCD_load(4,0x0F);
                    LCD_load(5,0x0F);
                    LCD_load(6,DIS_BLACK);
                    while(! P32) delay_ms(10);           //Wait for the button to be released
                    delay_ms(50);                        //Wait for the release button again
                    while(! P32) delay_ms(10);           //Turn off the timer
                    TR0 = 0;                             //External interrupt flag 0
                    IE0 = 0;
                    EX0 = 1;                             //INT0 Enable
                    IT0 = 1;                             //INT0 Falling edge interrupt
                    P1n_push_pull(0xff);                 //com and seg All output 0
                    P2n_push_pull(0xff);
                    P1 = 0;
                    P2 = 0;
                    PCON |= 0x02;                        //Sleep
                    _nop_();
                    _nop_();
                    _nop_();
                    LCD_SET_2M;                          //Display time interval :
                    LCD_SET_4M;                          //Display minute and second intervals :
                    LoadRTC();                           //Display time
                    TR0 = 1;                             //Turn on the timer
                    while(! P32) delay_ms(10);           //Wait for the button to be released
                    delay_ms(50);                        //Wait for the release button again
                    while(! P32) delay_ms(10);
                }
            }
        }
}
/******************** Delay function *************************/
void delay_ms(u8 ms)
{
        unsigned int i;
        do{
        i = MAIN_Fosc / 13000;
        while(--i);                                      //14T per loop
        }while(--ms);
}

/******************** Timer0 Interrupt function ********************/
void timer0_int (void) interrupt 1
{
        LCD_scan();
        B_2ms = 1;
}

/******************** INT0 Interrupt function ********************/
void INT0_int (void) interrupt 0
{
```

```
        EX0 = 0;
        IE0 = 0;
}

/****************** LCD                    Segment code scanning function                    ***********/
void LCD_scan(void)                                                    //5us @22.1184MHZ
{
        u8 code T_COM[4]={0x08,0x04,0x02,0x01};
        u8 j;
        j = scan_index >> 1;
        P2n_pure_input(0x0f);                                                  COM//all , High output resistance is the midpoint voltage of COM
        if(scan_index & 1)                                             //Reverse scan
        {
                P1 = ~LCD_buff[j];
                P2 = ~(LCD_buff[j+4] & 0xf0);
        }
        else                                                           //Positive phase scan
        {
        P1 = LCD_buff[j];
        P2 = LCD_buff[j+4] & 0xf0;
        }
        P2n_push_pull(T_COM[j]);                                       //A certain COM Set to push-pull output
        if(++scan_index >= 8) scan_index = 0;
}

/******************          For the first n Digital loading display function                ***************/
void LCD_load(u8 n, u8 dat)                                            //n For the first few numbers to be displayed, dat the number to be displayed
{
        u8 code t_display[]={                                          //Standard font library
                // 0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,
        //black
                0x00,0x40
        };
        u8 code T_LCD_mask[4] = {~0xc0,~0x30,~0x0c,~0x03};
        u8 code T_LCD_mask4[4] = {~0x40,~0x10,~0x04,~0x01};
        u8 i,k;
        u8 *p;
        if((n == 0) || (n > 6)) return;
        i = t_display[dat];
        if(n <= 4) //1~4
        {
                n--;
                p = LCD_buff;
        }
        else
        {
                n = n - 5;
                p = &LCD_buff[4];
        }

        k = 0;
        if(i & 0x08) k |= 0x40;                                        //D
        *p = (*p & T_LCD_mask4[n]) | (k>>2*n);
        p++;
```

```
        k = 0;
        if(i & 0x04) k |= 0x40;                    //C
        if(i & 0x10) k |= 0x80;                    //E
        *p = (*p & T_LCD_mask[n]) | (k>>2*n);
        p++;
        k = 0;
        if(i & 0x02) k |= 0x40;                    //B
        if(i & 0x40) k |= 0x80;                    //G
        *p = (*p & T_LCD_mask[n]) | (k>>2*n);
        p++;
        k = 0;
        if(i & 0x01) k |= 0x40;                    //A
        if(i & 0x20) k |= 0x80;                    //F
        *p = (*p & T_LCD_mask[n]) | (k>>2*n);
}

/********************Display time **************************/
void LoadRTC(void)
{
        LCD_load(1,hour/10);
        LCD_load(2,hour%10);
        LCD_load(3,minute/10);
        LCD_load(4,minute%10);
        LCD_load(5,second/10);
        LCD_load(6,second%10);
}
```

Assembly code

```
; Direct drive segment code series test (4 words, use I/O, 1/4 Duty, 1/3 bias)
; I/O Display a time after power-up (Minutes and seconds).

;************************************************************
P0M1 0x93 DATA
P0M0 DATA 0x94
P1M1 DATA 0x91
P1M0 DATA 0x92
P2M1 DATA 0x95
P2M0 DATA 0x96
P3M1 DATA 0xB1
P3M0 DATA 0xB2
P4M1 DATA 0xB3
P4M0 DATA 0xB4
P5M1 DATA 0xC9
P5M0 DATA 0xC
P6M1 DATA 0xCB
P6M0 DATA 0xCC
P7M1 DATA 0xE1
P7M0 DATA 0xE2
AUXR DATA 0x8E
INT_CLKO DATA 0x8F
IE2 DATA 0xAF
P4 DATA 0xC0
T2H DATA 0xD6
T2L DATA 0xD7
```

```
;*************************************************************
DIS_BLACK 010H      EQU
DIS_ EQU 011H
DIS_A EQU 00AH
DIS_B EQU 00BH
DIS_C EQU 00CH
DIS_D EQU 00DH
DIS_E EQU 00EH
DIS_F EQU 00FH


B_2ms               BIT     20H.                signal
B_Second            BIT     0 20H.              ;;2ms Second signal
cnt_500ms           DATA    1 30H
second              DATA    31H
minute              DATA    32H
hour                DATA    33H
scan_index          DATA    34H

LCD_buff            DATA    40H                 ;40H~47H

;*************************************************************
                    ORG     0000H
                    LJMP    F_Main

                    ORG     000BH
                    LJMP    F_Timer0_Interrupt

;*************************************************************
                    ORG     0100H
F_Main:
                    CLR     A
                    MOV     P3M1, A             ;Set to prevail two-way port
                    MOV     P3M0, A
                    MOV     P5M1, A             ;Set to prevail two-way port
                    MOV     P5M0, A

                    MOV     P1M1, #0            ; segment   Set to push-pull output
                    MOV     P1M0, #0ffh
                    ANL     P2M1, #NOT 0f0h     ; segment   Set to push-pull output
                    ORL     P2M0, #0f0h
                    ORL     P2M1, #00fH         ;all COM ,High output resistance midpoint voltage is the COM
                    ANL     P2M0, #0f0H
                    MOV     SP, #0D0H
                    MOV     PSW, #0
                    USING   0                   ;Select the first group R0~R7

;*************************************************************
                    MOV     R2, #8
                    MOV     R0, #LCD_buff
L_ClearLcdRam:
                    MOV     @R0, #0
                    INC     R0
                    DJNZ    R2, L_ClearLcdRam

                    LCALL   F_Timer0_init
                    SETB    EA

;                   ORL     LCD_buff, #020H     ;Display time interval ;
```

```
;                   ORL     LCD_buff, #002H             ;Display minute and second intervals ;

                    MOV     hour, #12
                    MOV     minute, #00
                    MOV     second, #00
                    LCALL   F_LoadRTC                   ;Display time

;*************************************************************
L_Main_Loop:

                    JNB     B_2ms, L_Main_Loop          ;2ms Beat to
                    CLR     B_2ms

                    INC     cnt_500ms
                    MOV     A, cnt_500ms
                    CJNE    A, #250, L_Main_Loop
                                                        ;500ms to
                    MOV     cnt_500ms, #0;

                    XRL     LCD_buff, #020H             ;Flashing time interval ;
                    XRL     LCD_buff, #002H             ;Flashing minutes and seconds interval ;

                    CPL     B_Second
                    JNB     B_Second, L_Main_Loop

                    INC     second
                    MOV     A, second
                    CJNE    A, #60, L_Main_Load
                    MOV     second, #0                  ; 1 Minutes to arrive
                    INC     minute
                    MOV     A, minute
                    CJNE    A, #60, L_Main_Load
                    MOV     minute, #0;
                    INC     hour
                    MOV     A, hour
                    CJNE    A, #24, L_Main_Load
                    MOV     hour, #0
                                                        ;24 The hour is up
L_Main_Load:
                    LCALL   F_LoadRTC                   ;Display time
                    LJMP    L_Main_Loop

;*************************************************************

F_Timer0_init:
                    CLR     TR0                         ; Stop counting
                    ANL     TMOD, #0f0H
                    SETB    ET0                         ; Allow interruption
                    ORL     TMOD, #0                    ; Working mode 0: 16 Automatic reloading of bits
                    ANL     INT_CLKO, #NOT 0x01         ; Does not output clock
                    ORL     AUXR, #0x80                 ; 1T mode
                    MOV     TH0, #HIGH (-22118)         ; 2ms
                    MOV     TL0, #LOW (-22118)          ;
                    SETB    TR0                         ; Start running
                    RET

;*************************************************************
F_Timer0_Interrupt:                 ;Timer0 1ms         Interrupt function
                    PUSH    PSW                         ;PSW Into the stack
                    PUSH    ACC                         ;ACC into the stack
                    PUSH    AR0
```

```
PUSH    AR7
PUSH    DPH
PUSH    DPL

LCALL   F_LCD_scan
SETB    B_2ms

POP     DPL
POP     DPH
POP     AR7
POP     AR0
POP     ACC             ;ACC Out of the stack
POP     PSW             ;PSW out of the stack
RETI

;********************   Display time**************************
F_LoadRTC:

MOV     R6, #1          ;LCD_load(1,hour/10);
MOV     A, hour
MOV     B, #10
DIV     AB
MOV     R7, A
LCALL   F_LCD_load      ;R6 For the first few numbers, for R7 Is the number to be displayed

MOV     R6, #2          ;LCD_load(2,hour%10);
MOV     A, hour
MOV     B, #10
DIV     AB
MOV     R7, B
LCALL   F_LCD_load      ;R6 For the first few numbers, for R7 Is the number to be displayed

MOV     R6, #3          ;LCD_load(3,minute/10);
MOV     A, minute
MOV     B, #10
DIV     AB
MOV     R7, A
LCALL   F_LCD_load      ;R6 For the first few numbers, for R7 Is the number to be displayed

MOV     R6, #4          ;LCD_load(4,minute%10);
MOV     A, minute
MOV     B, #10
DIV     AB
MOV     R7, B
LCALL   F_LCD_load      ;R6 For the first few numbers, for R7 Is the number to be displayed

MOV     R6, #5          ;LCD_load(5,second/10);
MOV     A, second
MOV     B, #10
DIV     AB
MOV     R7, A
LCALL   F_LCD_load      ;R6 For the first few numbers, for R7 Is the number to be displayed

MOV     R6, #6          ;LCD_load(6,second%10);
MOV     A, second
MOV     B, #10
DIV     AB
MOV     R7, B
LCALL   F_LCD_load      ;R6 For the first few numbers, for R7 Is the number to be displayed
```

```
                RET

;************************************************************
T_COM:
                DB          008H, 004H, 002H, 001H

F_LCD_scan:
                MOV         A, scan_index           ;j = scan_index >> 1;
                CLR         C
                RRC         A
                MOV         R7, A                   ;R7 = j
                ADD         A, #LCD_buff
                MOV         R0, A                   ;R0 = LCD_buff[j]
                ORL         P2M1, #00fH             ;all COM High output resistance, COM is the midpoint voltage
                ANL         P2M0, #0f0H

                MOV         A, scan_index
                JNB         ACC. 0, L_LCD_Scan2     ;if(scan_index & 1) // Reverse scan
                MOV         A, @R0                  ;P1 = ~LCD_buff[j];
                CPL         A
                MOV         P1, A
                MOV         A, R0                   ;P2 = ~(LCD_buff[j|4] & 0xf0);
                ADD         A, #4
                MOV         R0, A
                MOV         A, @R0
                ANL         A, #0f0H
                CPL         A
                MOV         P2, A
                SJMP        L_LCD_Scan3

L_LCD_Scan2:                                        ; Positive phase scan
                MOV         A, @R0                  ;P1 = LCD_buff[j];
                MOV         P1, A
                MOV         A, R0                   ;P2 = (LCD_buff[j|4] & 0xf0);
                ADD         A, #4
                MOV         R0, A
                MOV         A, @R0
                ANL         A, #0f0H
                MOV         P2, A

L_LCD_Scan3:
                MOV         DPTR, #T_COM            ;A certain COM Set to push-pull output
                MOV         A, R7
                MOVC        A, @A+DPTR
                ORL         P2M0, A
                CPL         A
                ANL         P2M1, A

                INC         scan_index              ;if(++scan_index == 8) scan_index = 0;
                MOV         A, scan_index
                CJNE        A, #8, L_QuitLcdScan
                MOV         scan_index, #0

L_QuitLcdScan:
                RET

;******************* Standard font library ************************
T_Display:
; 0 1                   2     3     4     5     6     7     8     9     A     B     C     D     E     F
```

```
DB      03FH,006H,05BH,04FH,066H,06DH,07DH,007H,07FH,06FH,077H,07CH,039H,05EH,079H,071H
;       black -
DB      000H,040H

;*******************  For the first  Digital loading display function algorithm is simple  ********************
F_LCD_load:                                                    ;R6  For the first few numbers, for  R7  Is the number to be displayed
        MOV     DPTR, #T_Display                               ;i = t_display[dat];
        MOV     A, R7
        MOVC    A, @A+DPTR
        MOV     B, A                                           ;Number to display

        MOV     A, R6
        CJNE    A, #1, L_NotLoadChar1
        MOV     R0,                                            #LCD_buff
        MOV     A, @R0
        MOV     C, B. 3                                        ;D
        MOV     ACC. 6, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 2 ACC.                                      ;C
        MOV     6, C C,
        MOV     B. 4 ACC.                                      ;E
        MOV     7, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 1 ACC.                                      ;B
        MOV     6, C C,
        MOV     B. 6 ACC.                                      ;G
        MOV     7, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 0 ACC.                                      ;A
        MOV     6, C C,
        MOV     B. 5 ACC.                                      ;F
        MOV     7, C
        MOV     @R0, A
        RET

L_NotLoadChar1:
        CJNE    A, #2, L_NotLoadChar2
        MOV     R0,#LCD_buff
        MOV     A, @R0
        MOV     C, B. 3                                        ;D
        MOV     ACC. 4, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 2 ACC.                                      ;C
        MOV     4, C C,
        MOV     B. 4 ACC.                                      ;E
        MOV     5, C
        MOV     @R0, A
```

```
    INC     R0 A,
    MOV     @R0 C,
    MOV     B. 1 ACC.       ;B
    MOV     4, C C,
    MOV     B. 6 ACC.       ;G
    MOV     5, C
    MOV     @R0, A

    INC     R0 A,
    MOV     @R0 C,
    MOV     B. 0 ACC.       ;A
    MOV     4, C C,
    MOV     B. 5 ACC.       ;F
    MOV     5, C
    MOV     @R0, A
    RET

L_NotLoadChar2:
    CJNE    A, #3, L_NotLoadChar3
    MOV     R0,#LCD_buff
    MOV     A, @R0
    MOV     C, B. 3         ;D
    MOV     ACC. 2, C
    MOV     @R0, A

    INC     R0 A,
    MOV     @R0 C,
    MOV     B. 2 ACC.       ;C
    MOV     2, C C,
    MOV     B. 4 ACC.       ;E
    MOV     3, C
    MOV     @R0, A

    INC     R0 A,
    MOV     @R0 C,
    MOV     B. 1 ACC.       ;B
    MOV     2, C C,
    MOV     B. 6 ACC.       ;G
    MOV     3, C
    MOV     @R0, A

    INC     R0 A,
    MOV     @R0 C,
    MOV     B. 0 ACC.       ;A
    MOV     2, C C,
    MOV     B. 5 ACC.       ;F
    MOV     3, C
    MOV     @R0, A
    RET

L_NotLoadChar3:
    CJNE    A, #4, L_NotLoadChar4
    MOV     R0,#LCD_buff
    MOV     A, @R0
    MOV     C, B. 3         ;D
    MOV     ACC. 0, C
    MOV     @R0, A
```

```
        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 2 ACC.               ;C
        MOV     0, C C,
        MOV     B. 4 ACC.               ;E
        MOV     1, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 1 ACC.               ;B
        MOV     0, C C,
        MOV     B. 6 ACC.               ;G
        MOV     1, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 0 ACC.               ;A
        MOV     0, C C,
        MOV     B. 5 ACC.               ;F
        MOV     1, C
        MOV     @R0, A
        RET

L_NotLoadChar4:

        CJNE    A, #5, L_NotLoadChar5
        MOV     R0,#LCD_buff+4
        MOV     A, @R0
        MOV     C, B. 3                 ;D
        MOV     ACC. 6, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 2 ACC.               ;C
        MOV     6, C C,
        MOV     B. 4 ACC.               ;E
        MOV     7, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 1 ACC.               ;B
        MOV     6, C C,
        MOV     B. 6 ACC.               ;G
        MOV     7, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 0 ACC.               ;A
        MOV     6, C C,
        MOV     B. 5 ACC.               ;F
        MOV     7, C
        MOV     @R0, A
        RET

L_NotLoadChar5:
```

```
        CJNE    A, #6, L_NotLoadChar6
        MOV     R0,#LCD_buff+4
        MOV     A, @R0
        MOV     C, B. 3                 ;D
        MOV     ACC. 4, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 2 ACC.               ;C
        MOV     4, C C,
        MOV     B. 4 ACC.               ;E
        MOV     5, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 1 ACC.               ;B
        MOV     4, C C,
        MOV     B. 6 ACC.               ;G
        MOV     5, C
        MOV     @R0, A

        INC     R0 A,
        MOV     @R0 C,
        MOV     B. 0 ACC.               ;A
        MOV     4, C C,
        MOV     B. 5 ACC.               ;F
        MOV     5, C
        MOV     @R0, A
        RET
L_NotLoadChar6:
        RET

        END
```

10 Command system

10.1 Addressing method

The addressing method is an indispensable part of the instruction set of every computer. The addressing method specifies the source and destination of the data. For different procedural instructions, the source and destination regulations will be different. The addressing method in the STC M can be summarized as:

Address now
- Direct addressing
- Indirect addressing
- Register

addressing
- Relative addressing
- Re-addressing
- Bit addressing

10.1.1 Address now

Immediate addressing is also called immediate number. It is the number of operations to participate in the operation directly given in the instruction opera

such as：MOV A, #70H

The function of this instruction is to transfer the immediate number 70H to accumulator A.

10.1.2 Direct addressing

In the direct addressing method, the instruction operand field gives the address of the participating operation operand. Direct addressing can only b special function registers, internal data registers, and bit address space. Among them, the special function register and bit address space can only be acc

such as：ANL 70H, #48H

Represents a unit in the 70H The number in the unit and the immediate number Please and the results are stored in the 70H unit. Where 70H is the direct address, indicating the intern data memory RAM.

10.1.3 Indirect addressing

Indirect addressing is represented by adding the "@" symbol before R0 or R1. For example, assuming that the data in R1 is 40H and the data contained in the 40H unit of the internal data memory is 55H, then the following instructions：

MOV A, @R1

Transfer the data 55H to the accumulator.

10.1.4 Register addressing

Register addressing is done to the selected R7~R0 , Accumulator A, general purpose register B, address register and carry C in the number for oper working register . Among them, the register n is represented by the lower 3 digits of the instruction code, ACC, B, DPTR, and carry bit C are implied in the instruction code. Therefore address also contains an implied addressing method.

The selection of the register work area is determined by RS1 and RS0 in the program status word register PSW. The registers specified by the instruction operand refer to the registers in the current work area.

Such as: INC R0;(R0)+1→R0

10.1.5 Relative addressing

Relative addressing is to add the current value in the program counter PC to the number given in the second byte of the instruction, and the result is address of the transfer instruction. The transfer address is also called the purpose of the transferAddress, the current value in the PC is called the base ad instruction is called the offset. Since the destination address is relative to the base address in the PC, this addressing method is called relative addressing be represented is +127 ~ 128 - 。 This addressing method is mainly used to transfer instructions.

such as : JC 80H ;C=1 jump

It means that if the carry bit C is 0, the content in the program counter PC does not change, that is, it will not be transferred. If the carry bit C is 1, the value in the PC is used as the base address, and the result obtained after adding the offset of 80H is used as the destination address of the transfer instru

10.1.6 Re-addressing

In the index addressing mode, the instruction operand specifies a index register that stores the base value of the index. When re-addressing, the offset is added to the base value of the re-addressing, and the result is the address of the operand. The index register has a program counter PC and an address register DPTR.

For example: MOVC A, @ A + DPTR

means that accumulator A is an offset register, and its contents are added to the contents of the address register DPTR. The result is the address of the operand, and the number in the unit is taken out and fed to accumulator A.

10.1.7 Bit addressing

Bit addressing refers to the addressing of some internal data memory RAM and special function registers during bit operations. When performing a the carry bit C as the bit operation accumulator, the instruction operand directly gives the address of the bit, and then performs a bit operation on the bit a opcode. The bit address has exactly the same form as the byte address in byte direct addressing. It is mainly distinguished by the opcode. Attention shou

such as MOV C, 20H ; On-chip bit unit bit operation type instruction

10.2 Instruction

table Mnemonic		Instruction description	byte	clock
ADD	A,Rn	The contents of the register are added to the accumulator	1	1
ADD	A,direct	, the data of the direct address unit is added to the accumulator	2	1
ADD	A,@Ri	, the data of the indirect address unit is added to the accumulator	1	1
ADD	A,#data	, the immediate number is added to the accumulator	2	1
ADDC	A,Rn	register, and the carry bit is added to the accumulator.	1	1
ADDC	A,direct	The data of the direct address unit is carried and added to the	2	1
ADDC	A,@Ri	accumulator . The data of the indirect address unit is carried and	1	1
ADDC	A,#data	added to the accumulator . The immediate count. The carry is added to the	2	1
SUBB	A,Rn	accumulator . The accumulator with borrow minus the register content	1	1
SUBB	A,direct	accumulator with borrow minus the content accumulator of the direct	2	1
SUBB	A,@Ri	address unit with borrow minus the content	1	1
SUBB	A,#data	accumulator of the indirect address unit with borrow minus the	2	1
INC	A	immediate count . Accumulator plus 1	1	1
INC	Rn	register plus 1	1	1
INC	direct	direct address unit plus 1	2	1
INC	@Ri	indirect address unit plus 1	1	1
DEC	A	accumulator minus 1	1	1
DEC	Rn	registerMinus 1	1	1
DEC	direct	direct address unit minus 1	2	1
DEC	@Ri	indirect address unit minus 1	1	1
INC	DPTR	address register DPTR plus 1	1	1

MUL	AB	A multiplied by B, B stores the high byte, A stores the low byte	1	2
DIV	AB	A divided by B, B stores the remainder, A stores the quotient	1	6
DA	A	accumulator Decimal adjustment	1	3
ANL	A,Rn	accumulator and register phase and	1	1
ANL	A,direct	accumulator and direct address unit phase and	2	1
ANL	A,@Ri	accumulator and indirect address unit phase and	1	1
ANL	A,#data	accumulator and immediate number phase and	2	1
ANL	direct,A	direct address unit and accumulator phase and	2	1
ANL	direct,#data	Direct address unit phase with immediate number phase with	3	1
ORL	A,Rn	accumulator and register phase or	1	1
ORL	A,direct	accumulator phase with direct address unit or	2	1
ORL	A,@Ri	accumulator phase with indirect address unit or	1	1
ORL	A,#data	accumulator phase with immediate number or	2	1
ORL	direct,A	direct address unit phase with accumulator or	2	1
ORL	direct,#data	direct address unit phase with immediate number or	3	1
XRL	A,Rn	accumulator phase with register is different or	1	1
XRL	A,direct	accumulator	2	1
XRL	A,@Ri	is different from direct address unit or The accumulator is different from the indirect address unit or	1	1
XRL	A,#data	the accumulator is different from the immediate number or	2	1
XRL	direct,A	the direct address unit is different from the accumulator or	2	1
XRL	direct,#data	the direct address unit is different from the immediate number or	3	1
CLR	A	the accumulator clears	1	1
CPL	A	the accumulator, the accumulator is reversed	1	1
RL	A	, the accumulator is looped, the	1	1
RLC	A	accumulator is looped, the	1	1
RR	A	accumulator is looped, the	1	1
RRC	A	accumulator is looped, the accumulator is looped, the accumulator is looped, the accumulator is	1	1
SWAP	A	looped, the accumulator is looped, the accumulator is looped, the accumulator is looped, the accumulator	1	1
CLR	C	clear zero carry bit	1	1
CLR	bit	clear 0 Direct address	2	1
SETB	C	bit 1 carry bit	1	1
SETB	bit	1 Direct address bit	2	1
CPL	C	carry bit inverse	1	1
CPL	bit	Direct address bit Inverse	2	1
ANL	C,bit	carry bit and direct address bit phase and	2	1
ANL	C,/bit	carry bit and direct address bit inverse code phase and	2	1
ORL	C,bit	carry bit and direct address bit phase or	2	1

ORL	C,/bit	The inverse code phase of the carry bit and the direct address bit or the direct address bit is fed into the carry bit , the carry bit is fed into the direct address bit register, and the contents are fed into the accumulator . The number in the direct address unitData is fed into the accumulator , the data in the indirect address is fed into the accumulator , the immediate number is fed into the accumulator , the accumulator content is fed into the register , the data in the direct address unit is fed into the register , the immediate number is fed into the register , the accumulator content is fed into the direct address unit , the register content is fed into the direct address unit	2	1
MOV	C,bit		2	1
MOV	bit,C		2	1
MOV	A,Rn		1	1
MOV	A,direct		2	1
MOV	A,@Ri		1	1
MOV	A,#data		2	1
MOV	Rn,A		1	1
MOV	Rn,direct		2	1
MOV	Rn,#data		2	1
MOV	direct,A		2	1
MOV	direct,Rn		2	1
MOV	direct,direct	The data in the direct address unit is fed to another direct address unit, the data in the indirect address is fed to the direct address unit, the immediate number is fed to the direct address unit, the accumulator content is fed to the indirect address unit, the direct address unit data is fed to the indirect address unit, the immediate number is fed to the indirect address unit, the 16-bit immediate number is fed to the data pointer in DPTRThe data in the base address re-addressing unit is fed to the accumulator, the data in the PC-based address re-addressing unit is fed to the accumulator , the content of the extended address (8-bit address) is fed to the accumulator, and the content of the extended RAM (16-bit address) is fed to the accumulator A, and the content of the accumulator A is sent to the accumulator A. Into the extended RAM (8-bit address), the contents of the accumulator A are fed into the extended RAM (16-bit address), the data in the direct address unit is pressed into the stack, the data pops up and is fed into the direct address unit register and the accumulator exchange the direct address unit and the accumulator exchange the indirect address and the accumulator exchange the indirect address of the lowNibble and accumulator exchange short call subroutine long call subroutine subroutine return interrupt return short jump long jump	3	1
MOV	direct,@Ri		2	1
MOV	direct,#data		3	1
MOV	@Ri,A		1	1
MOV	@Ri,direct		2	1
MOV	@Ri,#data		2	1
MOV	DPTR,#data16		3	1
MOVC	A,@A+DPTR		1	4
MOVC	A,@A+PC		1	3
MOVX	A,@Ri		1	3[1]
MOVX	A,@DPTR		1	2[1]
MOVX	@Ri,A		1	3[1]
MOVX	@DPTR,A		1	2[1]
PUSH	direct		2	2
POP	direct		2	2
XCH	A,Rn		1	1
XCH	A,direct		2	1
XCH	A,@Ri		1	1
XCHD	A,@Ri		1	1
ACALL addr11			2	3
LCALL addr16			3	3
RET			1	3
RETI			1	3
AJMP addr11			2	3
LJMP addr16			3	3

SJMP	rel	Relative jump	2	3
JMP	@A+DPTR	Relative to DPTR, the indirect jump	1	4
JZ	rel	accumulator is zero	2	1/3[2]
JNZ	rel	, the jump accumulator is	2	1/3[2]
JC	rel	non-zero, the carry bit is 1, the jump carry	2	1/3[2]
JNC	rel	bit is 0, the jump direct address	2	1/3[2]
JB	bit,rel	bit is 1, the jump direct address	3	1/3[2]
JNB	bit,rel	bit is 0, the jump direct address bit	3	1/3[2]
JBC	bit,rel	is 1, the jump, this bit is cleared to 0	3	1/3[2]
CJNE	A,direct,rel	The accumulator is not equal to the direct address unit, the	3	2/3[3]
CJNE	A,#data,rel	jump accumulator is not equal to the immediate number	3	1/3[2]
CJNE	Rn,#data,rel	, the jump register is not equal to the immediate number	3	2/3[3]
CJNE	@Ri,#data,rel	, the indirect address unit is not equal to the immediate	3	2/3[3]
DJNZ	Rn,rel	number, the jump register is reduced1 After the non-zero jump	2	2/3[3]
DJNZ	direct,rel	, the direct address unit is reduced1 After non-zero jump	3	2/3[3]
NOP		to empty operation	1	1

: When accessing the external extended RAM, the execution cycle of the instruction is related to the SPEED [2:0] bit in the register BUS_SPEED[1]

:The execution time of the conditional jump statement will vary depending on whether the condition is met. When the conditions are not me
Execute the next instruction, then the execution time of the conditional jump statement is 1 clock; when the condition is met, a jump will occu
the execution time of the conditional jump statement is 3 clocks.

[3] :The execution time of the conditional jump statement will vary depending on whether the condition is met. When the condition is not met, a
and the next instruction will continue to be executed. At this time, the execution time of the conditional jump statement is 2 clocks; when the
. At this time, the execution time of the conditional jump statement is 3 clocks.

10.3 Detailed instructions (Chinese)

ACALL addr11

Function: Absolute call

Description : ACALL The instruction implements an increase in the subroutine of the address represented by the parameter. When executing the instruction, PC call location , even if it has to be pointed to 2 PC First, the next instruction will be, and then the low and high bits of the bits will be based on

Press into the stack twice, and add the stack pointer twice at the 5 The high position of the value The first byte of the instruction ACALL 1 7
same time. Then, combine the current bit and the first byte to get a 8
16 The destination address of the bit, which is the entry address of the subroutine to be called.
It is required that the starting address of the subroutine must be immediately followed by ACALL Subsequent instructions are in the same program storage page.

ACALL Each flag bit will not be changed when the instruction is
For example : executed. The initial value is located in the program memory 07H, label 034H SUBRTN At the address, if the execution is located at the address
Instructions from the office:

ACALL SUBRTN

Then the address and 09H, inside RAM 08HSP become 09H The contents of the unit are 08H and , The value becomes 01H PC

Instruction length (bytes) 2

Execution cycle: 2

binary encoding :

A10	A9	A8	1	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

Be careful : Bit destination address of A10~A8 bit , a7 a6 a5 a4 a3 a2 a1 a0 Yes addr11 of A7~A0 bit.

operation : a9 a8 ACALL

(PC)←(PC)+ 2

(SP)←(SP) + 1

((SP)) ←(PC_{7-0})

(SP)←(SP) + 1

((SP))←(PC_{15-8})

(PC_{10-0}))←Page number address

ADD A, <src-byte>

Function: addition

Description : ADD Instructions can be used to complete the Represented source operands and accumulators A The current values are added. And put the result in the box

Additive in. According to the result of the operation, if the first bit has a carry, the carry flag is set to, otherwise it is cleared to zero; if the first b

The auxiliary carry flag is, otherwise it is cleared to zero. If it is an unsigned integer, it is added to the position bit, indicating that the current op

If the first 6 Bits are carried to generate the position, or the first Bits are carried to generate the first no position, then set For otherwise

OV Is cleared to zero. When performing the addition operation of signed integers, Setting means that the sum of two positive integers is a negative number, Is the sum of two negative integers to a positive number.

The source operand of this type of instruction can accept three addressing methods: register addressing, direct addressing, register indirect addressing, and immediate addressing.

For example : Suppose the value of the data in the accumulator is A 0C3H(11000011B) , R0 0AAH(10101010B)。 Execute the following instructions: ADD A, R0

The result is Accumulator is cleared to zero, carry flag and overflow flag A 6DH(01101101B), Auxiliary carry flag AC C OV 1 Be set.

ADD A, Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

		1 0	0	1	r	r	r

operation : ADD

(A)← (A0) + (Rn)

ADD A, direct

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

			0	0	1	0	1		direct address

operation : 1

0 ADD

(A)← (A0) + (direct)

ADD A, @Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

		1 0	0	0	1	1	i

operation : ADD

(A)← (A0) + ((Ri))

ADD A, #data

Instruction length (bytes) : 2

Execution cycle : 1

Binary encoding :

		1 0	0	0	1	0	0		immediate data

operation : ADD

(A) ← (A) + #data

ADDC A, <src-byte>

Function: When the carry-in addition

Description: executes the instruction, put ADDC The represented source operand is added to the accumulator together with the carry flag src-byte Fruit juice accumulator. According to the result of the operation, if the first If a carry bit is generated, the carry flag is set, otherwise it is cleared; if the carry bit is If a carry is generated in the first bit, the auxiliary carry flag is set to, otherwise it is cleared to zero. If it is an unsigned integer addition, the carr to show that the current operation result has overflowed. If the first bit has a carry generation and the first bit does not have it, or the first bit has a carry generation and the first bit does not have it, it will 6 7 7 6 OV 1 it will be cleared to zero. When performing signed integer addition operations , OV Set, indicating that the sum of two positive integers is a negative number, or OV 4

For example : Is the sum of two negative integers to a positive number. The source operand of this type of instruction allows several addressing methods: register addressing, direct addressing, register indirect addressing, and immedi Assuming that the value of the data in the accumulator is, the carry flag is, execute A 0C3H(11000011B) ' R0 0AAH(10101010B) 1 Line the following instructions: The result is

ADDC A,R0

Accumulator is cleared to zero, carry flag and overflow flag A 6EH(01101110B), Auxiliary carry flag AC C OV Be set.

ADDC A, Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

			1 0	1	r	r	r	

operation : ADDC

(A) ← (A) + (C) + (Rn)

ADDC A, direct

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

			1 0		1	0	1		direct address

operation : ADDC

(A) ← (A) + (C) + (direct)

ADDC A, @Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

			1 0	0	1	1	i	

operation : ADDC

(A) ← (A) + (C) + ((Ri))

ADDC A, #data

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

			1 0	0	1	0	0		immediate data

operation : ADDC

(A) ← (A) + (C) + #data

AJMP addr11

Function: Absolute jump

Description : The instruction is used to transfer the program to the corresponding destination address for execution. The address is generated during PC (After two increments) the high bit, the bit of the opcode, and the byte of the instruction are connected to form. Request the destination of the

The address of the latter instruction of the instruction and The byte are in the same program storage page.

For example : 1 Hypothesis label 0123H, instruction :

AJMP JMPADR

Located

0123H.

in 2 2 located in the program memory

Instruction length (bytes) :

JMPADR The value becomes 0345H, After executing the instruction PC

Execution cycle:

binary encoding :

		A8		0	0		1		A7	A6	A5	A4	A3	A2	A1	A0
A9 A10			A8 = A10		a10~a8		0 A7		A0 = a7~a0							

operation : Note: The destination address

AJMP

(PC) ← (PC)+ 2

(PC$_{10\text{-}0}$) ← page address

ANL <dest-byte> , <src-byte>

Function: Logic and operations on byte variables

Description : The instruction will be given by <dest-byte>, <src-byte> ANL The specified two-byte variables perform logic and operations bit by bit, and the operation is co

The fruit is stored in <dest-byte> the specified destination operand. The execution of this instruction will not affect the flag bit.

The combination of the two operation arrays allows two addressing modes. When the destination operand is an accumulator, the source opera addressing, direct addressing, register indirect addressing, and immediate addressing. When the destination operand is a direct address, the s an accumulator or an immediate number.

Note: When this command is used to modify the output port, the original data read in comes from the latch of the output data rather than the in

For example : If the content of the accumulator is, the content of register 0 is 0C3H(11000011B) 55H(01010101B), then the instruction:

ANL A,R0

The result of execution is that the contents of the accumulator become 41H(01000001H).

When the destination operand is directly addressable data ,

ANL Instructions can be used to put any unit or hardware register in RAM

Some bits are cleared to zero. The masked byte will determine which bits will be cleared to zero. The masked byte may be a constant, or it may generated by the accumulator during the calculation process. The following instructions:

ANL Pl, #01110011B

Clear the port and bits to zero. 1 7 Bit, bit 3 2

ANL A, Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

	1	0	1	1	r	r	r

operation : 0

ANL ∧ (Rn)

(A) ← (A)

ANL A, direct

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

0	1	0	1	0	1	0	1		direct address

operation :

ANL

(A) ← (A) ∧ (direct)

ANL A, @Ri

Instruction length (bytes): 1

Execution cycle: 1

binary encoding:

	1	0	1	0	1	1	i

operation: 0

ANL (A) ← (A) ∧ ((Ri))

ANL A, #data

Instruction length (bytes): 2

Execution cycle: 1

binary encoding:

	1	0	1	0	1	0	0		immediate data

operation: 0

ANL (A) ← (A) ∧ #data

ANL direct, A

Instruction length (bytes): 2

Execution cycle: 1

binary encoding:

0	1	0	1	0	0	1	0		direct address

operation: ANL (direct) ← (direct) ∧ (A)

ANL direct, #data

Instruction length (bytes): 3

Execution cycle: 2

binary encoding:

0	1	0	1	0	0	1	1		direct address		immediate data

operation: ANL (direct) ← (direct) ∧ #data

ANL C, <src-bit>

Function: Perform logic and

Description operations on bit variables if $_{\text{src-bit}}$ The represented Booleans are logic, and the carry flag is cleared; otherwise, the current state of the carry flag remains unchanged. In an assembly language program, the "/" symbol means that the addressed bit needs to be reversed before being used as the source operand in the calculation, but the source operand itself will not change. This instruction will not affect each other flag when executed. The source operand can only be directly addressed.

For example: The following sequence of instructions if and only if $_{\text{P1.0}}$ = 、 $_{\text{ACC. 7}}$ = and $_{\text{C 1:}}$ When the carry flag is set $_{0\text{OV}}$ =

```
MOV C, P1.0 ; LOAD CARRY WITH INPUT PIN STATE
ANL C, ACC. 7 ; AND CARRY WITH ACCUM. BIT. 7
ANL C, /OV ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C, bit

Instruction length (bytes): 2

Execution cycle: 2

binary encoding:

	0	0	0	0	0	1	0		bit address

operation: 1

ANL (C) ← (C) ∧ (bit)

ANL C, /bit

Instruction length (bytes): 2

Execution cycle: 2

binary encoding :

1	0	1	1	0	0	0	0		bit address

operation :

ANL

(C)← (C) (∧ $\overline{bit}$)

CJNE <dest-byte>, <src-byte>, rel

Function: If the two operands are not equal, transfer

Description : CJNE First compare the size of the two operands, if the two are not equal, the programSequence transfer. The destination address is lo

Make the signed offset of the last byte and PC The current value of (adjacent The address of the next instruction) add up

And made. If the target operand is an unsigned integer and its value is less than the unsigned integer corresponding to the source operand, then the carry flag is set, otherwise the carry flag is cleared to zero. But the operand itself will not be affected.

<src-byte> Combined, it allows <dest-byte> Kind of addressing mode. Accumulator directly addressable with any data or immediate numbers, any indirectly addressable RAM The unit or the current working register can be connected to the immediate c

Make a comparison.

For example : Set accumulator A The data contained is the median value 34H 00H :

```
        CJNE R7,#60H, NOT_EQ
;       ...             ......          ; R7 = 60H.
NOT_EQ: JC              REQ_LOW         ; IF R7 < 60H.
;...                    ......          ; R7 > 60H.
```

The first 1 An instruction sets the carry flag, and the program jumps to the label Next, pass the test carry flag ,

Can be determined R7 Is greater than 60H Still less than 60H .

Suppose the data of the port is also 34H Then the following instructions :

```
WAIT: CJNE A,P1,WAIT
```

Clear the carry flag and continue to execute, because at this time the value of the Value of the The data of the port is equal. P1

accumulator is also (if the data of the port is another value, then the program keeps looping to wait for the port becomes 34H So far.)

CJNE A, direct, rel

Instruction length (bytes) : 3

Execution cycle: 2

binary encoding :

		1	1	0	1	0	1		direct address		rel. address

operation :

0

1 (PC) ←(PC) + 3

IF (A) < > *(direct)*
THEN

(PC) ←(PC) +*relative offset*

IF (A) < *(direct)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE A, #data, rel

Instruction length (bytes) : 3

Execution cycle : 2

Binary encoding :

		1	1	0	1	0	0		immediate data		rel. address

operation :

0

1 (PC) ← (PC) + 3

IF (A) <> *(data)*
THEN

(PC) ← (PC) +*relative offset*

IF (A) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE Rn, #data, rel

Instruction length (bytes) 3

Execution cycle: 2

binary encoding :

		1	1	1	r	r	r		immediate data		rel. address

operation :

0

1 (PC) ← (PC) + 3

IF (Rn) <> *(data)*
THEN

(PC) ← (PC) +*relative offset*

IF (Rn) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE @Ri, #data, rel

Instruction length (bytes) 3

Execution cycle: 2

binary encoding :

		1	1	0	1	1	i		immediate data		rel. address

operation :

0

1 (PC) ← (PC) + 3

IF (Ri) <> *(data)*
THEN

(PC) ← (PC) +*relative offset*

IF (Ri) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CLR A

Function: Clear accumulator This command

Description: is used to assume that the accumulator is an accumulator. All bits of are cleared to zero, which does not affect the flag bits. The content is $_A$ 5CH(01011100B)^A, then the instruction :

Example :

CLR A

After execution, the value of the accumulator becomes 1

00H(00000000B)^A

Instruction length (bytes) :

Execution cycle: 1

binary encoding :

1	1	1	0	0	1	0	0

operation : CLR

(A) ← 0

CLR bit

Function: Clearing the specified bit

Description : will bit The bit represented is cleared to zero, and no flag bit will be affected used for carry signal CLR Can be directly

The addressing bit.

For example : Suppose the data of the port is 1 5DH(01011101B), then the instruction :

CLR P1.2

After execution , Pl The port is set to 59H(01011001B)°

CLR C

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

1	1	0	0	0	0	1	1

operation : CLR

(C) ← 0

CLR bit

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

	1	0	0	0	0	1	0		bit address

operation : 1

CLR

(bit) ← 0

CPL A

Function: Inverse accumulator

Description : will accumulator Each bit of is reversed, that is, the bit that was originally becomes, and the bit that was originally becomes. The instructio

Ring the flag.

Set accumulator

For example : The content is 5AH(01011100B), then the instruction :

CPL A

After execution, the contents of the accumulator become 0A3H (10100011B).

Instruction length (bytes) 1

Execution cycle: 1

binary encoding :

1	1	1	1	0	1	0	0

operation : CPL

(A) ← ( ) $\bar{A}$

CPL bit

Function: Will bit Inverse of the represented bit

Description : be bit The bit represented by the variable is reversed, that is, the original bit is The bit becomes 0, and no flag bit is affected 1

To the impact. CPL Can be used for carry signal Small directly addressable bits.

Note: If this command is used to modify the state of the bit The data represented is the endIn the port latch output port, then the data is not the current state entered from the pin. Set the data of the port to P1 5BH(01011011B), Then the instruction: CPL P1.1

For example :

CPL P1.2

P1 After execution , The port is set to 5DH(01011101B)。

CPL C

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

1	0	1	1	0	0	1	1

operation : CPL

(C) ← ( ) $\bar{C}$

CPL bit

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

1	0	1	1	0	0	1	0		bit address

operation : CPL

(bit) ← ($\overline{bit}$)

DA A

Function: After the addition operation, the accumulator Decimal adjustment

Description : Instruction pair accumulator The bit data stored in the previous addition operation is adjusted from the bit data generated by the previous a Instructions can be used to achieve two Addition of codes) to generate two-digit numbers. compressions if the accumulator is low 4 Bit Bit) greater than 9 (xxxx1010~xxxx 1111), or after the addition operation, assist (carry sign AC 1 For, then DA The instruction will be added to the accumulator to generate the correct one in the low bit BCD If added, low 6 4 There is a carry in the upper position, and the high position is, the carry will always be passed forward, so that the la

The flag is set; however, in other cases, the carry flag will not be cleared to zero, and the carry flag will remain at its original value. If the carry flag is, or the value of the high bit exceeds 4 9 (1010xxxx~1111xxxx), then DA 6 The instruction will add to the high bit and generate the correct number in the high bit, but do not clear the flag bit. If the high bit has a carry output, then 4 4 BCD 4 Set the carry flag to, otherwise, the carry flag will not be changed. The status of the carry flag indicates the original two BCD Whether the sum is greater than the instruction makes it possible to perform decimal addition operations accurately. attention , OV 99, and thus DA CPU Zhi will not be affected.

The above operations of the instruction are completed within one instruction cycle. In fact, according to the accumulator DA Different content of , DA Add to the accumulator 00H、 06H、 60H、 66H A In order to achieve decimal conversion.

Note: If the addition operation is not performed earlier, it cannot be used directly. DA The hexadecimal data in the instruction is transfer Change to a number, in addition, if the previous subtraction operation was performed, The instruction will not have the expected effect.

For example : DA BCD

BCD Code, the contents of the register

If the content in the accumulator is 56H for 67H ( ), represents a decimal number of BCD code. The carry flag is, then the command : (01010110B), which represents a decimal number 56

ADDC A,R3

DA A

First perform standard complement binary addition, accumulator The value becomes, The carry flag and the auxiliary carry flag are cleared

zero.

Then , DA Perform decimal adjustment and add The content becomes), represents a decimal number

24 of BCD the accumulator code, which is And the last two digits of the sum of the carry instruction will set the

, Which means that an overflow occurred during decimal addition. carry flag to and the sum is 1 。

Addition or subtraction can be achieved. Format variables plus Suppose the initial value of the accumulator is

Show decimal number), instruction sequence:

```
ADD A, #99H
DA
   A
```

Will carry in position Set to, accumulator The data becomes because 。 The low data of the addition sum can be

To be regarded as the result of the subtraction operation, that is,

Instruction length (bytes)

Execution cycle:

binary encoding :

1	1	0	1	0	1	0	0

operation :

```
DA
-contents of Accumulator are BCD
IF [[(A3-0) > 9] V [(AC) = 1]]
THEN(A3-0) ← (A3-0) + 6
AND
IF [[(A7-4) > 9] V [(C) = 1]]
THEN (7-4A)←(A7-4) + 6
```

DEC byte

Function: put BYTE The number of operations represented is reduced

Description : BYTE The represented variable is subtracted. If the original value is Taken after subtracting , become 。 No standard

1 bit, it will be affected. This instruction supports Supports operand addressing methods: accumulator addressing, register addressing, direct addr

And register indirect addressing. Note: When

the command is used to modify the status of the output port , BYTE DEC The data represented is the output data lock from the port

The input state obtained from the memory, not the input state read from

For example : the pin. Suppose the contents of the register are ( ), inside of RAM 7EH and 7FH The contents of the unit are

and 。 Then the instruction :

```
DEC @R0
DEC R0

DEC @R0
```

After execution, the contents 7EH, inside RAM of 7EH and 7FH The content of the unit becomes

of the register become and 。

DEC A

Instruction length (bytes) ;

Execution cycle:

binary encoding :

	0 0	0	1	0	1	0	0

operation :

```
DEC
(A)← (A) -1
```

DEC Rn

Instruction length (bytes) ;

Execution cycle :

Binary encoding :

		0	1	1	r	r	r

operation :

0

0 DEC

(Rn) ← (Rn) - 1

DEC direct

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

			1	0	1	0	1		Direct address

operation :

0

0 DEC

(direct) ← 0 (direct) - 1

DEC @Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

		0 0	1	0	1	1	i

operation :

DEC

((Ri)) ← 0 ((Ri)) - 1

DIV AB

Function: division

Description : DIV The bits in the unsigned integer are divided by the register, and the unsigned integer, and the quotient is placed in Accumulator A is placed in the register. Carry signs and overflow signs B C Instruction accumulator OV Is cleared to zero.

Exception: if the register The initial value is 00 B (That is, the divisor is) 0, then execute DIV After the instruction, the accumulator register B The value in is uncertain, and the overflow flag OV Will be set. But in any case, the carry flag C Du

Will be cleared to zero.

For example : Assuming that the value of 251 0FBH or 11111011B), register B The value is 18 (12H or 00010010B).

the accumulator is the

instruction: DIV AB

After execution, The value of the accumulator becomes 13 0DH or 00001101B), The value of the register becomes 17 (11H or 00010001B) ,

it is exactly in line with 251 = 13 × 18 + 17 。 Both carry and overflow flags are cleared to zero.

Instruction length (bytes) : 1

Execution cycle: 4

binary encoding :

1	0	0	0	0	1	0	0

operation :

DIV

(A) 15-8 (B 7-0) ← (A)/(B)

DJNZ <byte>, <rel-addr>

Function: Minus, 1 0 Then jump

Description : if not DJNZ The instruction first subtracts the variable represented by the first operand, and if the result is not, it is transferred to the first op

Execute at the address specified by the number. If the value of the first operand is, it becomes 1 00H, then reduce 1 0FFH° This instruction does not affect the flag bit. Calculation of the jump destination address: first add the value (that is, the first byte pointing to the next inst

2 Then add the signed relative offset represented by the first operand to it. 2

PC

The represented operand can be register-addressable or directly addressable. byte

Note: If this instruction is used to modify the state on the output pin, then the data represented is the number output from the port. byte

According to what is obtained in the latch, instead of reading the pin directly.

For example : Inside the hypothesis and 40H The units are stored separately RAM 40H~ 50H and 15H, then the instruction :

```
DJNZ 40H, LABEL_1
DJNZ 50H, LABEL_2
DJNZ 60H, LABEL_3
```

After execution, the program will jump to the label Executed at the place, and the corresponding RAM The content of the unit becomes a 3 and 6FH 15H。 The reason for the first A jump was not executed because of The result is that the jump condition is not met.

use DJNZ Instructions can easily implement a specified number of cycles in the program, and in addition, one instruction can be use

Achieve a medium-length time delay (2~512 Machine cycle). Instruction sequence :

```
        MOV R2,#8
TOOOLE: CPL P1.7
        DJNZ R2, TOOGLE
```

Will make P1.7 The level is flipped several times, thus Generate a pulse, each pulse will last for a machine cycle , 3 one of them 2 DJNZ Execution time of instruction , CPL The execution time of the instruction.

DJNZ Rn, rel

Instruction length (bytes) : 2

Execution cycle: 2

binary encoding :

		0 1	1	1	r	r	r		rel. address

operation : DJNZ

(PC) ← (PC) + 2

(Rn) ← (Rn) - 1

IF (Rn) > 0 or (Rn) < 0

THEN

(PC) ← (PC)+ rel

DJNZ direct, rel

Instruction length (bytes) : 3

Execution cycle: 2

binary encoding :

			1	0	1	0	1		direct address		rel. address

operation : DJNZ

(PC) ← (PC) + 2

(direct) ← (direct) - 1

IF (direct) > 0 or (direct) < 0

THEN

(PC) ← (PC)+ rel

INC <byte>

Function: plus 1

Description : INC The instruction will The represented data is added. If the original value is , then after addition becomes The instruction Does not affect the flag position. Three addressing modes are supported: register addressing, direct addressing, and register indirect addressing. Note: If this instruction is used to modify the state on the output pin, then the data represented is the number output from th According to the pins obtained in the latch, not directly read.

For example : Suppose the contents of the register are stored in the unit and the unit respectively , inside RAM 7E 7F and 40H, then the instruction sequence:

```
INC @R0
INC R0
```

INC @R0

After execution, the contents of the register becomes 0 7FH, and inside RAM of 7EH and 7FH The content of the unit becomes

00H and 41H.

INC A

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

	0 0	0	0	0	1	0	0

operation : INC

(A) ← (A)+1

INC Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

		0	0	1	r	r	r

operation : 0

0 INC

(Rn) ← (Rn)+1

INC direct

Instruction length (bytes) : 2

Execution cycle: 1

binaryencoding :

			0	0	1	0	1		direct address

operation : 0

0 0 INC

(direct) ← (direct)+1

INC @Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

		0 0	0	0	1	1	i

operation : INC

((Ri)) ← 0((Ri)) + 1

INC DPTR

Function: Data pointer plus 1

Description The implementation of this instruction will Add function. It should be noted that this is Bit increment instruction, low byte 16 from

increase FFH 1 00H After that, it becomes, and at the same time it is carried to the high byte. This operation does not affect the flag bit.

1 This instruction is the only 16 Bit register increment instruction.

For example : hypothetical register and DPH DPL The contents are 12H and 0FEH, then the instruction sequence :

IINC DPTR

INC DPTR

INC DPTR

After execution is complete , and DPH DPL become 13H and 01H.

Instruction length (bytes) 1

Execution cycle: 2

binary encoding :

			0 1	0	0	1	1

operation : INC

(DPTR) 01 ← (DPTR) + 1

JB bit, rel

Function: If the bit 1 Then jump

Description data is if The bit data represented is, then jump to Execute at the specified address; otherwise, continue to execute the next one.

instruction. The destination address of the jump is calculated as follows: First add to that it points to the first word of the next instruction

Section address, and then put the signed relative offset represented (the first part of the instruction Bytes) added to Go up, new

of PC The value is the destination address. This instruction only tests the corresponding bit data, but does not change its value, and th

Will not affect the flag position.

For example : Suppose the input data of the port is , The value of the accumulator is). Then the instruction :

JB P1.2, LABEL1

JB ACC. 2, LABEL2

Will cause the program to go to the label Go to execution.

Instruction length (bytes) : 3

Execution cycle: 2

binary encoding :

0	0	1	0	0	0	0	0		bit address		rel. address

operation :

```
JB
(PC)
        (PC) + 3
←
IF (bit) = 1
    THEN
        (PC) ← (PC)+ rel
```

JBC bit, rel

Function: If the bit 1 Then jump and clear it to zero

Description data is if The bit data represented is, then it is cleared to zero and jumped to Execute at the specified address. if

The bit data represented is, then continue to execute the next instruction. The destination address of the jump is calculated as follows:

The value of, so that it points to the first byte address of the next instruction, and then put PC The signed relative offset represented (the reference value is

Add the first byte of the order) to it, the new one the destination address, and this operation will not affect the flag.

Note: If this command is used to modify the state on the output pin, then The data represented is the number of outputs from the port According to what is obtained in

the latch, instead of reading the index directlyfeet. Suppose the content of the accumulator is , then the

For example : instruction sequence:

JBC ACC. 3, LABEL1

JBC ACC. 2, LABEL2

Will cause the program to go to the label to execute, and the contents of the accumulator become .

Instruction length (bytes) 3

Execution cycle: 2

binary encoding :

		0	1	0	0	0	0		bit address		rel. address

operation :

```
JB0
(PC) ←0 (PC) + 3
IF (bit) = 1
    THEN
        (bit) ← 0
        (PC) ← (PC)+ rel
```

JC rel

Function: If the carry flag is, then jump . If the carry

Description: flag is, then the program jumps to rel Execute at the address represented; otherwise, continue to execute the following instructio

The destination address of the jump is calculated as follows: First add PC so that it points to the next instruction of the instruction JC

The first address, and then put rel offset immediately represented (the first of the instruction Bytes) added to it , PC

The new value is the destination address. This operation will not affect the flag bit. PC

For example : Assuming that the carry flag is at this time, the instruction sequence : 0

```
JC LABEL1
CPL C
JC LABEL2
```

After execution is completed, the carry flag becomes and causes the program to jump to LABEL2 Go to execution.

the reference numeral 2

Instruction length (bytes) :

2

Execution cycle:

binary encoding :

		0	0	0	0	0	0		rel. address

operation : JC

(PC) ← (PC) + 2

IF (C) = 1

THEN

(PC) ← (PC)+ rel

JMP @A+DPTR

Function: Indirect jump

Description: handle accumulator Bits of unsigned data and The values of the data pointers of the bits are added, and the sum is used as the next one

The address of the instruction, transmitted to the execute When adding bits, the low byte The carry will be passed to the high

program counter byte and data pointer DPH° Accumulator The content will not change. Does not affect any flags.

For example : Suppose the value in the accumulator is even) (from The following sequence of instructions will cause the program to jump to the jump tabl

JMP_TBL The strip AJMP Execute one of the instructions :

```
MOV DPTR, #JMP_TBL
JMP @A+DPTR

JMP-TBL: AJMP LABEL0
AJMP LABEL1
AJMP LABEL2
AJMP LABEL3
```

If you start to execute the above instruction sequence, the value in Then the program will eventually jump to the label

Go to execution. LABEL2

attention : AJMP Is a Byte instruction, so in the jump table, the entry address of each jump instruction is different in turn byte.

Instruction length (bytes)

Execution cycle:

binary compilationcode :

			1 0	0	0	1	1		

operation : JMP

(PC) ← (A) + (DPTR)

JNB bit, rel

Function: If bit If the bit represented is not, then the bit

Description if bit represented by the jump is, then it is transferred to rel Execute the address represented; otherwise, continue to execute the next instruction.

The destination address of the jump is calculated as follows: first add PC The value of, so that it points to the first byte address of the next instruction, and then pu

rel The signed relative offset represented by (the first part of the instruction) Bytes, added to PC Go up, new one PC The value is the destination

Mark the address. This instruction only tests the corresponding bit data, but does not change its value, and this

For example : operation will not affect the flag bit. Suppose the input data of the port is 11001010B, The value of the accumulator is 56H

(01010110B). Then

the instruction sequence: JNB P1.3, LABEL1

JNB ACC. 3, LABEL2

Instruction length (bytes) : After execution, the program will be transferred to the label for execution. LABEL2

Execution cycle: 3

binary encoding : 2

		1	1	0	0	0	0		bit address		rel. address

operation : 0

0 JNB

(PC) ←(PC) + 3

IF (bit) = 0

THEN (PC) ← (PC) + rel

JNC rel

Function: If the carry flag is not, then jump . If the carry

Description flag is, then the program jumps to rel Execute at the address represented; otherwise, continue to execute the following instructi

The destination address of the jump is calculated as follows: first add PC The value is added so that it points to The immediate of the instruction

The address of the instruction, and then put rel The signed relative offset represented (the first byte of the instruction) is added to PC on 2

Go, new one PC The value is the destination address. This operation will not affect the flag bit.

For example : Assuming that the carry flag is at this time, the instruction sequence : 1

JNC LABEL1

CPL C

JNC LABEL2

After execution is completed, the carry flag becomes and causes the 0 LABEL2 Go to execution.

program to jump to the reference numeral 2

Instruction length (bytes) :

2

Execution cycle:

binary encoding :

		0	1	0	0	0	0		rel. address

operation : 1

0 JNC

(PC) ←(PC) + 2

IF (C) = 0

THEN (PC) ← (PC) + rel

JNZ rel

Function: If the content of the accumulator is not,

Description then jump if the accumulator 1A For any one of them, then the program jumps to Execute at the address represented, if each bit

All for, 0 continue to execute the next instruction. The destination address of the jump is calculated as follows: first put PC The value increases

Then add the signed relative offset (the first byte of the instruction) represented by it, and the new one rel 2 PC value

That is the destination address. The value of the accumulator will not change during operation and will not

For example : affect the flag. Set the initial value of the accumulator to 00H, then the instruction sequence:

```
JNZ LABEL1
INC A
JNZ LAEEL2
```

After execution, the contents of the accumulator become, and the program will jump to the label to execute. 01H

LABEL2 2

Instruction length (bytes) :

2

Execution cycle:

binary encoding :

		1	1	0	0	0	0		rel. address

operation : 1

0 JNZ

(PC) ← (PC) + 2
(A) ≠ 0IF

THEN (PC) ← (PC) + rel

JZ rel

Function: If the content of the accumulator

Description is if the accumulator For any one of them, then the program jumps to execute at the address represented, if each bit

All for, 0 continue to execute the next instruction. The destination address of the jump is calculated as follows: first put PC

Then add the signed relative offset (the first byte of the instruction) represented by it, and the new one

That is the destination address. The value of the accumulator will not change during operation and will not affect the

For example : flag. Set the initial value of the accumulator to 01H, then the instruction sequence:

```
JZ LABEL1
DEC A
JZ LAEEL2
```

After execution, the contents of the accumulator become, and the program will jump to the label to execute. 00H LABEL2

2

Instruction length (bytes) :

2

Execution cycle:

binary encoding :

		1	0	0	0	0	0		rel. address

operation : 1

JZ 0

(PC) ← (PC) + 2

IF (A) = 0
THEN (PC) ← (PC) + rel

LCALL addr16

Function: Long

Description : call LCALL

LCALL The address of the next instruction, and then put the bit

First), and at the same time add the stack pointer.

Then put the high-bit bytes and low-bit bytes in

DPH The subroutine can be located at

The initial value of the stack pointer

For example : is the following address, the label 's instruction :

```
LCALL SUBRTN
```

The subroutine at the address referred to. Will first be used to call

The low and high bits are pressed into the stack in turn (the first byte and the first byte of data of the low-byte instruction are loaded separately).

, The program is from the new

Execution starts at the corresponding address. thus

Any address of the program storage space. This operation does not affect the flag bit.

, After The allocated program memory address is Then execute as follows

The stack pointer first, inside RAM of 08H and 09H The contents of the unit are PC(PC) and 01H, and PC The current

becomes the value of 1234H.

Instruction length (bytes): 3

Execution cycle: 2

binary encoding:

		0	1	0	0	1	0		addr15-addr8		addr7-addr0

operation:

0

0 LCALL

(PC) ← (PC) + 3

(SP) ← (SP) + 1

((SP)) ← (PC_{7-0})

(SP) ← (SP) + 1

((SP)) ← (PC $_{15-8}$)

(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long jump

Description: Unconditionally jump to make Execute the program at the address referred to. Put the first and second instruction

Bytes are loaded into the upper byte of the program counter separately. PC DPH And low-bit bytes。 Program renewal PC DPL to the byte and the first value

Execution begins at the address. The destination address of this bit can be located at 16 ORG Any address of the program storage space. This operation does not affect

Flag bit.

For example: Hypothesis label ADDR The allocated program memory address is It is located at the address 1234H The instruction:

LJMP JMPADR

After execution is complete The current value becomes PC

Instruction length (bytes): 3

Execution cycle: 2

binary encoding:

		0	0	0	0	1	0		addr15-addr8		addr7-addr0

operation:

0

0 LJMP

(PC) ← $addr_{15-0}$

MOV <dest-byte> , <src-byte>

Function: Transfer the byte variable

Description to the first The operand represents the contents of the byte variable stored in the specified address represented by the operand. The instruction will not

1 Changing the source operand will not affect other registers and flags.

MOV The instruction is by far the most flexible instruction used. The source operand

15 and the destination operand are combined, and there are various addressing methods.

Inside Delta

For example: The content of the unit is 1 RAM 40H The content of the unit is 1 The data is

11001010B (0CAH 30H). Then the instruction sequence:

```
MOV R0, #30H        ; R0<= 30H
MOV A, @R0          ; A <= 40H
MOV R1, A           ; R1 <= 40H
MOV B, @R1          ; B <= 10H
MOV @R1, P1         ; RAM (40H) <= 0CAH
MOV P2, P1          ; P2 #0CAH
```

After the execution is complete, the register 0 The content is, Accumulator and register The content is 40H, register B inside

Capacity is 128H ' RAM In 40H Unit and P2 The contents of the mouth are all

MOV A,Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

	1	1	0	1	r	r	r

operation : 1 MOV

(A) ← (Rn)

***MOV A,direct**

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

		1	0	0	1	0	1		direct address

operation : 1

1 MOV

(A) ← (direct)

MOV A, ACC **Is an invalid instruction.**

attention :

MOV A,@Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

	1 1	1	0	0	1	1	i

operation : MOV

(A) ←((Ri))

MOV A,#data

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

	1 0	1	1	0	1	0	0		immediate data

operation : MOV

(A) ← #data

MOV Rn, A

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

	1	1	1	1	r	r	r

operation : 1 MOV

(Rn) ← **(A)**

MOV Rn,direct

Instruction length (bytes) : 2

Execution cycle: 2

binary encoding :

		1	0	1	r	r	r		direct address

operation : 0

1 MOV

(Rn) ← (direct)

MOV Rn,#data

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

	1 0	1	1	1	r	r	r		immediate data

operation : MOV

(Rn) ← #data

MOV direct, A

Instruction length (bytes) 2

Execution cycle: 1

binary encoding :

		1	1	0	1	0	1		direct address

operation :

1

1 MOV

(direct) ← (A)

MOV direct, Rn

Instruction length (bytes) 2

Execution cycle: 2

binary encoding :

		0	0	1	r	r	r		direct address

operation :

0

1 MOV

(direct) ← (Rn)

MOV direct, direct

Instruction length (bytes) 3

Execution cycle: 2

binary encoding :

1	0	0	0	0	1	0	1		dir. addr. (src)		dir. addr. (dest)

operation :

MOV

(direct) ← (direct)

MOV direct, @Ri

Instruction length (bytes) 2

Execution cycle: 2

binary encoding :

1	0	0	0	0	1	1	i		direct address

operation :

MOV

(direct) ← ((Ri))

MOV direct, #data

Instruction length (bytes) 3

Execution cycle: 2

binary encoding :

		1	1	0	1	0	1		direct address		immediate data

operation :

1

0 MOV

(direct) ← #data

MOV @Ri, A

Instruction length (bytes) 1

Execution cycle: 1

binary encoding :

	1 1	1	1	0	1	1	i

operation :

MOV

((Ri)) ← (A)

MOV @Ri, direct

Instruction length (bytes) 2

Execution cycle: 2

binary encoding :

		1	0	0	1	1	i		direct address

operation :

0

1 MOV

((Ri)) ← (direct)

MOV @Ri, #data

Instruction length (bytes) 2

Execution cycle: 1

binary encoding :

0	1	1	1	0	1	1	i		immediate data

operation : MOV

((Ri)) ← #data

MOV <dest-bit> , <src-bit>

Function: The transfer bit variable

Description will <src-bit> Copy the represented booleans to <dest-bit> In the specified data unit, there must be one of the two operands One is a carry flag, while the other is a directly addressable bit. This instruction does not affect other registers and flags. Assuming that the data in the carry flag is, the data of port 1 is set to C 1 The initial value is, port P3 11000101B 35H(00110101B)° Then the instruction sequence:

For example :

```
MOV P1.3, C
MOV C, P3.3
MOV P1.2, C
```

After execution, the carry flag is cleared to zero, and the data of the port becomes 39H (00111001B)·

MOV C, bit

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

	0	1	0	0	0	1	0		bit address

operation : MOV

(C) ← (bit)

MOV bit, C

Instruction length (bytes) : 2

Execution cycle: 2

binary encoding :

	0	0	1	0	0	1	0		bit address

operation : MOV

(bit) ← (C)

MOV DPTR, #data 16

Function: The constant of the bit is stored in the data pointer to 16

Description The instruction will The bit constant is contained in the byte and word of the instruction 3 DPTR° 16 be in the section. Its bit constant is passed to the data pointer ·, What is stored in is The high byte, and what is stored in it is #data16 The low byte. not DPL DPH Affect the flag.

This instruction is the only instruction that

For example : can move 16 bits of data at once.

Instruction: MOV DPTR, #1234H

The value of the immediate number is 1234H Load into the data pointer register. DPH 12H ' DPL The value is 34H°

Instruction length (bytes) : 3

Execution cycle: 2

binary encoding :

			1	0	0	0	0		immediate data15-8		immediate data7-0	

operation : 0

1 MOV

(DPTR) 0 ← $\#data_{15-0}$ $\#data_{7-0}$

DPH DPL ← $\#data_{15-8}$

MOVC A , @A+ <base-reg>

Function: Transfer the code byte data (constant data) in the program memory to the accumulator A

Description :

MOVC Instructions transfer code bytes or constant bytes in the program memory to the accumulator. Of the transmitted data bytes

The address is composed of unsigned bit data and bit base address register in the accumulator. The value of the addition is generated and needs

of. If before , PC PC PC is the base address register, then the accumulator content is added to the address of the subsequent statement;

MOVC DPTR if it is the base address register, then the accumulator content is added to the address of the subsequent statement.

When adding, the carry generated by the low bit will be passed to the high bit. This instruction does not affect the flag bit.

For example : Assuming that the value of the accumulator is in between, the following subroutine will accumulate the accumulator A 0~4 A

Definition byte) One of the values defined:

```
REL-PC: INC A
        MOVC A, @A+PC
        RET
        DB 66H
        DB 77H
        DB 88H
        DB 99H
```

If the value of the accumulator before calling the subroutine is , after executing the subroutine, the value of the accumulator becomes 77H.

MOVC Before the instruction PC The command is set to cross during the look-up table. If and form of RET MOVC

The space is separated by multiple code bytes, so in order to read the table

correctly, the corresponding number of bytes must be pre-added to the accumulator. A

MOVC A,@A+DPTR

Instruction length (bytes) 1

Execution cycle: 2

binary encoding :

			1	0	0	1	1

operation :

1 0 MOVC

(A) ←((A)+(DPTR))

MOVC A,@A+PC

Instruction length (bytes) ; 1

Execution cycle: 2

binary encoding :

		0 1	0	0	0	1	1

operation : MOVC

(PC) ←(PC)+1

(A) ←((A)+(PC))

MOVX <dest-byte> , <src-byte>

Function: External

Description : transmission MOVX instructions are used to transfer data between the accumulator and the external data memory. Therefore, the command is being already. The It is divided into two types, the difference between them is to access external data MOVX The indirect address is

position is still Bit of.

For the first type, the current working register bank and R1 Provide a bit address to the multiplexing port For the outside

I/O Extended decoding or smaller array , RAM array ,

A high-bit address signal can be output on the port pin. The address of the bit is enough. To access the larger MOVX Add output instructions before the instructions, for these

At this time, control can be applied to the port pin. For

the first type, it is generated by a data pointer 2 DPTR16 The address of the bit The output buffer of the port is sent

DPH When the content The special function register maintains the original data. When accessing a larger data array ,
P2
This method is more effective and fast, because no additional instructions are required to configure the
output port. In some cases, two types of instructions can be mixed. Accessing large-capacity MOVX In space, both
You can use the data pointer in DPTR The high byte of the output address on the port, you can also use a certain instruction first to put the hi
Bits and bytes are output from the port, and then used to address through or inter-address P2 R0 Rl MOVX
instruction.
Suppose there is an external memory with a time-sharing multiplexed address/data line , RAM 256B (Such as :
For example : capacityfor of Intel 8155 RAM
), the memory is connected to the port of The CPU is 8051 P0 On, port P3 The data that are used to RAM So
The required control signal. Ports and Pl used as a general purpose input/output port R0 provide externally are 12H and
RAM34H, of 34H P2 The data stored in the unit is 56H , Then the following sequence of instructions :
external MOVX
A, @R1 MOVX @R0,
A Transfer data 56H Copy to accumulator And external RAM of 12H In the unit.

MOVX A,@Ri

Instruction length (bytes) : 1

Execution cycle: 2

binary encoding :

	1 1	1	0	0	0	1	i

operation : MOVX

(A) ← ((Ri))

MOVX A,@DPTR

Instruction length (bytes) : 1

Execution cycle: 2

binary encoding :

		1	0	0	0	0	0

operation : 1

1 MOVX

(A) ←((DPTR))

MOVX @Ri, A

Instruction length (bytes) : 1

Execution cycle: 2

binary encoding :

	1 1	1	1	0	0	1	i

operation : MOVX

((Ri))←(A)

MOVX @DPTR, A

Instruction length (bytes) : 1

Execution cycle: 2

binary encoding :

		1	1	0	0	0	0

operation : 1

1 MOVX

(DPTR)←(A)

MUL AB

Function: multiplication

Description : This instruction can be used to implement accumulator A Multiplication of registers B Bit integers in. The resulting Bit product of
The low bits are stored in the accumulator, while the high bits are stored in the register B). then set overflow
Out of the flag; otherwise, the flag bit is cleared. When this instruction is executed, the carry flag is
always cleared to zero. Assuming that the initial value of the accumulator is 160 (0A0H), register B then the instruction :
For example : MUL AB

Get the product 12000 (3200H), So the register B The value becomes), the accumulator is cleared and overflows 32H (00110010B

flag is set and the carry flag is cleared.

Instruction length (bytes) 1

Execution cycle: 4

binary encoding :

1		1	0	0	1	0	0

operation : (A) ← 0 7-0 (A)×(B)

(B) 15-8

NOP

Function: Empty operation

Description: Example : After this instruction is executed, subsequent instructions will continue PC In addition, other registers and flags will not change.

to be executed. In addition to assuming that the expectation is in the port A long-term low-level pulse is output on the No. 1 pin of the machine, which lasts for a period of time. 5

Cycle (precise). If you only use and SETB CLR Instruction sequence, the generated pulse is a machine cycle. A machine cycle

Therefore, it is necessary to try to add an additional machine cycle. The required functions can be implemented in the following ways (a

Break is not enabled) :

```
CLR P2.7
NOP
NOP
NOP
NOP
SETB P2.7
```

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

		0	0	0	0	0	0

operation : 0

0 NOP

(PC) ←(PC)+1

ORL <dest-byte> , <src-byte>

Function: Logic or operation of a two-byte variable

Description : The instruction will be given by <dest-byte> Stored in <src-byte> ORL The specified two-byte variable performs bit-by-bit logic or operation, and the result is

In the data unit represented. This operation does not affect the flag bit.

<dest-byte> 6 A When the source operand can be taken

The two operation arrays are combined to support multiple addressing methods. When the destination operand is an accumulator

Use register addressing, direct addressing, register indirect addressing, or immediate addressing. When the destination operand adopts

addressing , the source operand can be an accumulator or an immediate number.

Note: If this command is used to modify the state on the output pin, then the data represented is from the port <dest-byte>

Outputs the data obtained in the data latch, not the data read from the pin.

For example : Suppose the data in the accumulator is 0C3H (11000011B), register R0 The data in is 55H(01010101) ,then the instruction:

```
ORL A, R0
```

After execution, the contents of the accumulator become. When the destination operand is directly addressing data bytes ,

```
ORL 0D7H(11010111B)
```

Instructions can be used to set each bit in any unit or hardware register to. Which bits will be set RAM 1 1

It is determined by the masked byte, which can be either a constant contained in the instruction or a constant during the operation of th

The value calculated in real time. Execute instructions:

```
ORL P1, #00110010B
```

After that, put Mouth of the first 、 45 、 1 location.1

ORL A, Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

	1 0	0	0	1	r	r	r

operation : ORL

(A) ←(A)∨ (Rn)

ORL A, direct

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

	1 0	0	0	0	1	0	1		direct address

operation : ORL

(A) ←(A)∨ (direct)

ORL A, @Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

	1 0	0	0	0	1	1	i

operation : ORL

(A) ←(A)∨ ((Ri))

ORL A, #data

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

	1 0	0	0	0	1	0	0		immediate data

operation : ORL

(A) ←(A)∨ #data

ORL direct, A

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

			0	0	0	1	0		direct address

operation : 0

0 ORL

(direct) ← 1 (direct)∨ (A)

ORL direct, #data

Instruction length (bytes) : 3

Execution cycle: 2

binary encoding :

			0	0	0	1	1		direct address		immediate data

operation : 0

0 ORL

(direct) ← 1 (direct)∨ #data

ORL C, <src-bit>

Function: The logic or operation

Description of a bit variable if $_{src\text{-}bit}$ If the bit variable represented is, the carry flag is set; otherwise, the current state of the carry flag is kept unchanged. 1

In assembly language, the "before the source operand" / "Means that the source operand is reversed and used, but the source operand itself does not change. When this instruction is executed, other flags are not affected.

For example : When the following sequence of instructions is executed, if and only if $=_{0OV}$ When setting the carry flag :

C P1.0 = or ACC. 7 = or

```
MOV C, P1.0                ;LOAD CARRY WITH INPUT PIN P10
ORL C, ACC. 7              ;OR CARRY WITH THE ACC. BIT 7
ORL C, /OV                 ;OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

Instruction length (bytes) : 2

Execution cycle: 2

binary encoding :

	1 0	1	1	0	0	1	0		bit address

operation : ORL

(C) ←(C)∨ (bit)

ORL C, /bit

Instruction length (bytes) : 2

Execution cycle: 2

binary encoding :

1	0	1	0	0	0	0	0		bit address

operation : ORL

(C) ←(C)∨($\overline{bit}$)

POP direct

Function: Out of the stack

Description : Read the internal specified by the stack The content of the unit, the stack pointer is subtracted. Then, the read content is transmitted to the pointer Go to the indicated storage unit (direct addressing method). This operation does not affect the flag bit.

For example : Set the initial value of the stack pointer to 32H, 30H~32H The data of the unit are 20H, 23H and 01H, then Internal execution instructions:

```
POP DPH
POP DPL
```

After that, the value of the stack pointer 30H, The data pointer becomes 0123H. At this time the command : becomes POP SP

Will change the stack pointer to 20H.

Note: In this special case, when writing out-of-stack data (20H) Before, the stack pointer is reduced to 21H and then again With the writing, it becomes 20H 20H.

Instruction length (bytes) : 2

Execution cycle: 2

binary encoding :

		0 1	1	0	0	0	0		direct address

operation : POP

(direct) ←((SP))

(SP) ← (SP) - 1

PUSH direct

Function: Press stack

Description : The stack pointer is added first, and then the contents of the represented variable are copied to the internal specified by the stack pointer Go to Yuanzhong. This operation does not affect the flag bit.

For example : When the program enters the interrupt service program, data pointer DPTR The value is 0123H. Then execute as follows the value of the stack pointer is the instruction sequence :

PUSH DPL

PUSH DPH

After that, the 0BH, and put the data 23H and 01H Stored separately inside of 0AH and 0BH Storage order stack pointer becomes a meta pointer. 2 2

Instruction length (bytes) :

Execution cycle:

binary encoding :

		0 1	0	0	0	0	0		direct address

operation : 1 PUSH

(SP) ←(SP) + 1

((SP)) ← (direct)

RET

Function: Return to execution from subroutine

Description : RET The upper and lower bytes of the value are ejected from the stack, and the stack pointer is subtract The program is formed from PC Execution begins at the address corresponding to the value. Under normal circumstances, the instruction and or A

Used in conjunction. The execution of the change instruction does not affect the flag bit. Set the initial value of the stack pointer to and 0BH inside The data in the storage unit are 23H and 01H°

For example : Then the instruction:

RET

After execution, the stack pointer becomes 09H° The program will start from 0123H Execution continues at the address.

Instruction length (bytes) : 1

Execution cycle: 2

binary encoding :

0	0	1	0	0	0	1	0

operation : RET

(PC15-8) ←((SP))

(SP) ←(SP) -1

) ←(PC((SP))

(SP) ←(SP) -1

RETI

Function: Interrupt return

Description : When this instruction is executed, it first pop The high and low bytes of the value, and then the recovery interrupt is enabled, ready to ac

For other interrupts of the same priority, the stack pointer is reduced. Other registers are not affected. However, the program status word reverts to the state before the interrupt. The program will continue to generate Execution starts at the address corresponding to the value. Under normal cir the next instruction from the interrupt entry. Other interrupts at the execution RETI if there is a lower priority or the same priority, the execution will start at the addr level are waiting for processing, so they need to be executed before processing these waiting interrupts. PSW1 Set the initial value of the s

For example : The contents of the sum unit are and 0AH 0BH 23H 01H° Then the instruction: An interrupt is generated during the end of the instruction execution, and the inte

RETI

After execution is completed, the stack pointer becomes, and the program continues to execute from the address after the interrupt retu

Instruction length (bytes) : 1

Execution cycle: 2

binary encoding :

0	0	1	1	0	0	1	0

operation : RETI

(PC15-8) ←((SP))

(SP) ←(SP) -1

(P7.0C) ←((SP))

(SP) ←(SP) -1

RL A

Function: The data bits in the loop are shifted to the left to move the accumulator A

Description: Example : Shift the content of the accumulator in Move in place. The execution of this instruction does not affect the flag bit. 7 the accumulator to the left by setting the accumulator to bit data, the bit of which is shifted to the left. 8 1 0C5H ( ), then the instruction :

(

RL A

, And the flag position is not affected. 10001011B)

After execution, the contents of the accumulator become 8BH (

Instruction length (bytes) 1

Execution cycle: 1

binary encoding :

		1	0	0	0	1	1

operation : 0

RL 0 n = 0-6

(An+1) ← (An)

(A0) ←(A7)

RLC A

Function: With carry loop

Description: left shift accumulator The bit data and carry flag cycle to the left bit. The middle of 7 Move in the carry flag, the beginning of the carry flag together

The initial state value is moved into place. This command does not affect other flags1.

For example : Suppose the value of the accumulator is A 0C5H(11000101B), then the instruction:

RLC A

After execution, the data of the accumulator will be changed to A 8BH(10001011B) , The carry flag is set.

Instruction length (bytes) 1

Execution cycle: 1

binary encoding :

0	0	1	1	0	0	1	1

operation : RLC

(An +1) ←(An) n = 0-6

(A0) ←(C)

(C) ←(A7)

RR A

Function: Cycle

Description : the data bits of the accumulator to the right and move the data bits of the accumulator to the right to the right, this instruction does no will be moved into place, that is, the loop moves

bit.

For example : Set the content of the accumulator to 0C5H(11000101B), then the instruction :

(

RR A

0E2H (11100010B), the flag position is not affected.

After execution, the content of the accumulator becomes

Instruction length (bytes) :

1

Execution cycle: 1

binary encoding :

		0	0	0	0	1	1

operation : 0

RR 0

n = 0 - 6

($_n$A) ←(A_n+1)

(A_7) ←(A_0)

RRC A

Function: Right shift with carry cycle

Description : The bit data of the accumulator and the carry flag cycle to the right together 1。 The center of it is moved into the carry flag, and the begi

The initial state value is moved into place. This instruction does not affect other flags.

For example : Assuming that the value of the accumulator is and the carry flag is, the instruction : 0C5H(11000101B) 0

RRC A

After execution, the data of the accumulator will be changed to, and the carry flag will be set. 62H(01100010B)

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

		0	1	0	0	1	1

operation : 0

0 RRC

n = 0 - 6

(A_n+1) ←(A_n)

(A_7) ←(C)

(C) ←(A_0)

SETB <bit>

Function: Set

Description : SETB The instruction can be placed in the corresponding position, and the object of its operation can be a carry flag or other straightConnect to the addressing bit. This instruction does not affect other flags.

For example : The carry flag is cleared to zero, and the output status of the port is 1 34H(00110100B), then the instruction:

SETB C

SETB P1.0

After execution, the carry flag becomes, and the output status of the port becomes 1 35H(00110101B)。

SETB C

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

	1	0	1	0	0	1	1

operation : 1

SETB

(C) ← 1

SETB bit

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

	1	0	1	0	0	1	0		bit address

operation : 1

SETB

(bit) ← 1

SJMP rel

Function: Short jump

Description : The program unconditionally jumps to the address shown to execute. The destination address is calculated as follows: First, the value is added and then PC

Then the first byte of the instruction (ie The value is a short jump PC

) Before the address On, the signed offset represented by the obtained new) is add

The destination address. Therefore, the range of the jump is the current instruction (that SJMP 128 Byte and back byte. 127

For example : is, the instruction address corresponding to the label is located in the program memory) Address, then command :

SJMP RELADR

Located after compilation. When the instruction is executed , 0123H

PC The value becomes 0100H 0102H , Therefore, the offset of the jump is

Note: In the above example, the address of the next instruction immediately after is SJMP 21H In addition, if 0123H SJMP The offset is 0FEH, Then the composition is only Unlimited instructions

0102H =

Cycle.

Instruction length (bytes) : 2

Execution cycle: 2

binary encoding :

1	0	0	0	0	0	0	0		rel. address

operation : SJMP

PC( ) ←(PC)+2

(PC) ← (PC)+rel

SUBB A, <src-byte>

Function: Subtraction

Description with borrowing SUBB The instruction is subtracted from the The value and carry flag of the byte variable represented, the result of the subtraction ope

The fruit is placed in the accumulator. If the first place needs to be borrowed The carry flag (indicating subtraction) will be set; before the

Otherwise, the carry flag is cleared. (If it is being executed SUBB instruction, the carry flag has been set, which means that in C

When the multi-precision subtraction operation was performed earlier, a debit was generated. Therefore, when executing this instruction

be subtracted from the accumulator together with the source operand.) If during the subtraction operation, the first If there is a place to borrow, then 3

The auxiliary carry flag will be set; if the first bit is borrowed; but the first bit is not, or the first bit is borrowed, and AC 6 7 7

If the first bit is not available, the overflow flag is set. 6 OV

When performing a signed integer subtraction operation, if it is set, it means that a negative number is generated in the process of subtr

Or, in negative numbers Positive numbers are produced in the process of subtracting positive numbers.

The addressing methods supported by the source operand are: register addressing, direct addressing, register indirect addressing, and

For example : Set the value of the data in the accumulator to be 0C9H(11001001B) register R2 54H(01010100B), Carry sign C

Is set. Then the following instructions:

SUBB A, R2

After execution, the data of the accumulator becomes and the auxiliary carry flag is cleared to zero, overflow 74H(01110100B), Carry sign C AC

The output flag is set.

C

attention : 0C9H Minus should be, but in the above calculation, since before the instruction is executed, carry 54H 75H SUBB

The flag has been set, so the final result still needs to be subtracted from the carry flag to obtain it. Therefore, if the order is in progress

Before the precision or multi-precision subtraction operation, the status of the carry flag is unknown, then the instruction should be use

The flag is cleared to zero. C

SUBB A, Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

1	0	0	1	1	r	r	r

operation : SUBB

(A) ← (A) - (C) - (Rn)

SUBB A, direct

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

			1 1 0	0	1	0	1		direct address

operation : 0 SUBB

(A) ← (A) - (C) - (direct)

SUBB A, @Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

			1 1	0	1	1	i

operation : 0 0 SUBB

(A) ← (A) - (C) - ((Ri))

SUBB A, #data

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

			1 1	0	1	0	0		immediate data

operation : 0 0 SUBB

(A) ← (A) - (C) - #data

SWAP A

Function: Exchange the high and low nibs of the accumulator

Description : SWAP The command puts the accumulator low 4 bit (bit bit_{3-0}) and high 4 bit (bit bit_{7-4}) Data is exchanged. Actually SWAP Instructions can also be regarded as cyclic instructions for bits. This

For example : instruction does not affect the flag bit. Set the content of the accumulator to 0C5H (11000101B),

then the instruction: SWAP A

After execution, the contents of the accumulator become 5CH (01011100B).

Instruction length (bytes) 1

Execution cycle: 1

binary encoding :

1	1	0	0	0	1	0	0

operation : SWAP

(A_{3-0}) ↔ $(A)_{7-4}$

XCH A, <byte>

Function: Exchange the contents of the accumulator and byte variables

Description : The instruction will <byte> The contents of the specified byte variable are loaded into the accumulator, and the old contents of the accumula

write <byte> loaded into the specified byte variable. The addressing method allowed for the source and destination operands in the ins

Addressing, direct addressing, and register indirect addressing.

For example : Set the content as the address R0 20H, The value of 3FH (00111111B)° inside RAM of 20H Content of the unit the accumulator is 75H (01110101B)° Then the instruction:

XCH A, @R0

After execution, the data of the internal unit becomes RAM 20H , The content of the accumulator becomes 3FH (00111111B)

75H(01110101B)。

XCH A, Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

1	1	0	0	1	r	r	r

operation : XCH

(A) ↔ (Rn)

XCH A, direct

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

1	1	0	0	0	1	0	1	direct address

operation : XCH

(A) ↔ (direct)

XCH A, @Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

1	1	0	0	0	1	1	i

operation : XCH

(A) ↔ ((Ri))

XCHD A, @Ri

Function: Exchange accumulator and @Ri The low bit of the data in the corresponding unit 4

Description : XCHD The instruction will be the lower nibble (bit) of the accumulator content 0-3 Code) and indirect addressing, usually hexad

The interior of RAM The data of the unit is exchanged, and the respective high half-characters (bits) are exchanged.), The section is not affected. In addition, the instruction

Does not affect the flag

For example : position. Set and save the address as 20H, The content of the 36H (00110110B)。 inside RAM of 20H Unit storage

The data is accumulator is 75H (011110101B)。 Then the instruction :

XCHD A, @R0

After execution, the internal RAM 20H The content of the unit becomes , The content of the accumulator becomes 76H (01110110B)

35H(00110101B)。

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

1	1	0	1	0	1	1	i

operation : XCHD

(A_{3-0}) ↔ (Ri)$_{3-0}$

XRL <dest-byte>, <src-byte>

Function: Logical xor of byte variables

Description : XRL The instruction will <dest-byte> and <src-byte> The represented byte variables perform logical XOR operations bit by bit, and the results are

<dest-byte> In the byte variable represented. This instruction does not affect the flag bit.

The two operands are combined to 6 support a total of three addressing methods: when the destination operand is an accumulator, the so
can be register-addressed or directly found.Address, register indirect addressing, and immediate number addressing; when the destina
directly addressable data, the source operand can be an accumulator or an immediate number.

Note: If this command is used to modify the state on the output pin, then the data represented is from the port

dest-byte

For example : Outputs the data obtained in the data latch, not the data read from the pin. 0

If the contents of the accumulator and register are respectively 0C3H (11000011B) and 0AAH(10101010B), then the instruction:

XRL A, R0

After execution, the contents of the accumulator become 69H (01101001B). When
the destination operand is directly addressable byte data, this instruction can put each of any unit or register

RAM

The bit is reversed. Which specific bits will be reversed will be determined during operation. Instruction:

XRL P1, #00110001B The position of the mouth 、 45 、
0 Is reversed.
After execution , P1

XRL A, Rn

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

0	1	1	0	1	r	r	r

operation : XRL

(A) ← (A) ∀ (Rn)

XRL A, direct

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

0	1	1	0	0	1	0	1		direct address

operation : XRL

(A) ← (A) ∀ (direct)

XRL A, @Ri

Instruction length (bytes) : 1

Execution cycle: 1

binary encoding :

0	1	1	0	0	1	1	i

operation : XRL

(A) ← (A) ∀ ((Ri))

XRL A, #data

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

0	1	1	0	0	1	0	0		immediate data

operation : XRL

(A) ← (A) ∀ #data

XRL direct, A

Instruction length (bytes) : 2

Execution cycle: 1

binary encoding :

0	1	1	0	0	0	1	0		direct address

operation : XRL

(direct) ← (direct) (A)

XRL direct, #data

Instruction length (bytes) : 3

Execution cycle: 2

binary encoding :

0	1	1	0	0	0	1	1		direct address		immediate data

operation : XRL

(direct) ← (direct) #data

10.4 Detailed instructions (English)

ACALL addr11

Function: Absolute Call

Description :

ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice.

The destination address is obtained by successively concatenating the five high-order bits of the incremented PC opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding:

		A8	1	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

Operation:

A9

A10 ACALL

(PC) ← (PC)+ 2

(SP) ← (SP) + 1

((SP)) ← (PC_{7-0})

(SP) ← (SP) + 1

((SP)) ← (PC_{15-8})

(PC_{10-0}) ← page address

ADD A, <src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The

carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A, R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A, Rn

Bytes: 1

Cycles: 1

Encoding:

		1 0	0	1	r	r	r

Operation:

ADD

(A) ← (A^0) + (Rn)

ADD A, direct

Bytes: 2

Cycles: 1

Encoding:

			0	0	1	0	1		direct address

1

Operation:

0 ADD

(A) ← (A^0) + (direct)

ADD A, @Ri

Bytes: 1

Cycles: 1

Encoding:

		1 0	0	0	1	1	i

Operation:

ADD

(A) ← (A^0) + ((Ri))

ADD A, #data

Bytes: 2

Cycles: 1

Encoding:

		1 0	0	0	1	0	0		immediate data

Operation:

ADD

(A) ← (A^0) + #data

ADDC A, <src-byte>

Function: Add with Carry

Description:

ADDC simultaneously adds the byte variable indicated, the Carry flag and the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B) with the Carry.

The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the Accumulator with the AC flag cleared and both the carry flag and

OV set to 1.

ADDC A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	1	r	r	r

Operation: ADDC

(A) ← (A) + (C) + (Rn)

ADDC A, direct

Bytes: 2

Cycles: 1

Encoding:

			1 0		1	0	1		direct address

Operation: ADDC

(A) ← (A^{010}) + (C) + (direct)

ADDC A, @Ri

Bytes: 1

Cycles: 1

Encoding:

			1 0	0	1	1	i

Operation: ADDC

(A) ← (A^{01}) + (C) + ((Ri))

ADDC A, #data

Bytes: 2

Cycles: 1

Encoding:

			1 0	0	1	0	0		immediate data

Operation: ADDC

(A) ← (A^{01}) + (C) + #data

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: **The label "JMPADR" is at program memory location 0123H. The instruction,**

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 2

Encoding:

A10	A9	A8	0	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

Operation: AJMP

(PC) ← (PC)+ 2

(PC_{10-0}) ← page address

ANL <dest-byte> , <src-byte>

Function: Logical-AND for byte variables

Description:

ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch not the input pins.

Example: If the Accumulator holds 0C3H(11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time.

The instruction,

ANL Pl, #01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A, Rn

Bytes: 1

Cycles: 1

Encoding:

	1	0	1	1	r	r	r

Operation:

0

ANL ∧ (Rn)

(A) ← (A)

ANL A, direct

Bytes: 2

Cycles: 1

Encoding:

	1	0	1	0	1	0	1		direct address

Operation:

0

ANL ∧ (direct)

(A) ← (A)

ANL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

	1	0	1	0	1	1	i

Operation:

0

ANL ∧ ((Ri))

(A) ← (A)

ANL A, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	0	1	0	1	0	0		immediate data

Operation: ANL

(A) ← (A) ∧ #data

ANL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	0	1	0	0	1	0		direct address

Operation: ANL

(direct) ← (direct) ∧ (A)

ANL direct, #data

Bytes: 3

Cycles: 2

Encoding:

0	1	0	1	0	0	1	1		direct address		immediate data

Operation: ANL

(direct) ← (direct) ∧ #data

ANL C, <src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV C, P1.0 ; LOAD CARRY WITH INPUT PIN STATE
ANL C, ACC. 7 ; AND CARRY WITH ACCUM. BIT. 7
ANL C, /OV ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C, bit

Bytes: 2

Cycles: 2

Encoding:

	0	0	0	0	0	1	0		bit address

Operation: 1

ANL ∧ (bit)

(C) ← (C)

ANL C, /bit

Bytes: 2

Cycles: 2

Encoding:

1	0	1	1	0	0	0	0		bit address

Operation: ANL

(C) ← (C) ∧ ($\overline{bit}$)

CJNE <dest-byte>, <src-byte>, rel

Function: Compare and Jump if Not Equal

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not

equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
                CJNE R7,#60H, NOT_EQ
;               . . .           . . . . . .         ; R7 = 60H.
NOT_EQ: JC                      REQ_LOW             ; IF R7 < 60H.
; . . .                         . . . . . .         ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT-EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

WAIT: CJNE A,P1,WAIT

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on Pl, the program will loop at this point until the P1 data changes to 34H.)

CJNE A, direct, rel

Bytes: 3

Cycles: 2

Encoding:

		1	1	0	1	0	1		direct address		rel. address

Operation:

0

1 **(PC)** ← (PC) + 3

IF (A) < > *(direct)*
THEN

(PC) ← (PC) +*relative offset*

IF (A) < *(direct)*

THEN

(C) ← 1

ELSE

(C) ← **0**

CJNE A, #data, rel

Bytes: 3

Cycles: 2

Encoding:

		1	1	0	1	0	0		immediate data		rel. address

Operation:

0

1 **(PC)** ← (PC) + 3

IF (A) < > *(data)*
THEN

(PC) ← (PC) +*relative offset*

IF (A) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE Rn, #data, rel

Bytes: 3

Cycles: 2

Encoding:

		1	1	1	r	r	r		immediate data		rel. address

Operation:

0

1 **(PC)** ←(PC) + 3

IF (Rn) < > *(data)*
THEN

(PC) ←(PC) +*relative offset*

IF (Rn) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← **0**

CJNE @Ri, #data, rel

Bytes: 3

Cycles: 2

Encoding:

		1	1	0	1	1	i		immediate data		rel. address

Operation:

0

1 **(PC)** ←(PC) + 3

IF (Ri) < > *(data)*
THEN

(PC) ←(PC) +*relative offset*

IF (Ri) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← **0**

CLR A

Function: Clear Accumulator

Description:

The Accumulator is cleared (all bits set on zero). No flags are affected.

Example:

The Accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the Accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	0	1	0	0

Operation:

CLR

(A) ← 0

CLR bit

Function: Clear bit

Description:

The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the

carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	0	1	1

Operation: CLR

(C)←0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

	1	0	0	0	0	1	0		bit address

Operation: 1

CLR

(bit)←0

CPL A

Function: Complement Accumulator

Description: **Each bit of the Accumulator is logically complemented (one's complement). Bits which** previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH(01011100B). The instruction,

CPL A

will leave the Accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1	1	1	1	0	1	0	0

Operation: CPL.

(A)← ( ) $\overline{A}$

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: Port 1 has previously been written with 5BH(01011011B). The instruction,

CPL P1.1

CPL P1.2

will leave the port set to 5DH(01011101B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1	0	1	1	0	0	1	1

Operation: CPL

(C)← ($\overline{C}$)

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	1	0	0	1	0		bit address

Operation: CPL

(bit)← ($\overline{bit}$)

DA A

Function: Decimal-adjust Accumulator for Addition

Description:

DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set or if the four high-order bits now exceed nine(1010xxxx- 1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H(01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence.

ADDC A,R3

DA A

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56,67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

ADD A, #99H

DA A

will leave the carry set and 29H in the Accumulator, since 30+99=129. The low-order byte of the sum can be interpreted to mean 30 - 1 = 29.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	0	0

Operation:

DA

-contents of Accumulator are BCD

IF [[(A_{3-0}) > 9] V [(AC) = 1]]

THEN$(_{3-0}A)$ ← (A_{3-0}) + 6

AND

IF [[(A_{7-4}) > 9] V [(C) = 1]]

THEN (A_{7-4}) ← (A_{7-4}) + 6

DEC byte

Function: Decrement

Description:

The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example:

Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

DEC @R0

DEC R0

DEC @R0

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

	0 0	0	1	0	1	0	0

Operation:

DEC

(A) ← $(A)^{-1}$

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

		0	1	1	r	r	r

Operation:

0

0 DEC

DEC direct

(Rn) ← (Rn) - 1

Bytes: 2

Cycles: 1

Encoding:

			1	0	1	0	1		Direct address

Operation:

0

0 DEC

DEC @Ri

(direct) ← 0(direct) - 1

Bytes: 1

Cycles: 1

Encoding:

		0 0	1	0	1	1	i

Operation:

DEC

((Ri)) ← 0((Ri)) - 1

DIV AB

Function: Divide

Description:

DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251(OFBH or 11111011B) and B contains 18(12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13×18) + 17. Carry a nd OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

1	0	0	0	0	1	0	0

Operation:

DIV

$(A)_{15-8}$ $\leftarrow (B)_{7-0}(A)/(B)$

DJNZ <byte>, <rel-addr>

Function: Decrement and Jump if Not Zero

Description:

DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed

relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

DJNZ 40H, LABEL_1

DJNZ 50H, LABEL_2

DJNZ 60H, LABEL_3

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

MOV R2,#8

TOOOLE: CPL P1.7

DJNZ R2, TOOGLE

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn, rel

Bytes: 2

Cycles: 2

Encoding:

		0	1	1	r	r	r		rel. address

Operation:

1

1 DJNZ

(PC) ← (PC) + 2
(Rn) 1

(Rn) ← -

IF (Rn) > 0 or (Rn) < 0

THEN

(PC) ← (PC)+ rel

DJNZ direct, rel

Bytes: 3

Cycles: 2

Encoding:

			1	0	1	0	1		direct address		rel. address

Operation:

1 DJNZ

(PC) ← 1(PC) +0 2

(direct) ← (direct) - 1

IF (direct) > 0 or (direct) < 0

THEN

(PC) ← (PC)+ rel

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H.No

flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

INC @R0

INC R0

INC @R0

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

	0 0	0	0	0	1	0	0

Operation: INC

(A) ← (A)+1

INC Rn

Bytes: 1

Cycles: 1

Encoding:

		0	0	1	r	r	r

Operation: 0

0 INC

(Rn) ← (Rn)+1

INC direct

Bytes: 2

Cycles: 1

Encoding:

			0	0	1	0	1		direct address

Operation: 0

0 0 INC

(direct) ← (direct)+1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

		0 0	0	0	1	1	i

Operation: INC

((Ri)) ← 0((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2_{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order-byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Register DPH and DPL contains 12H and 0FEH, respectively. The instruction sequence,

IINC DPTR

INC DPTR

INC DPTR

will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

Encoding:

	0	1	0	0	0	1	1

Operation:

1

INC ← **(DPTR) + 1**

(DPTR)

JB bit, rel

Function: Jump if Bit set

Description:

If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction.

The bit tested is not modified. No flags are affected

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

JB P1.2, LABEL1

JB ACC. 2, LABEL2

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

		1	0	0	0	0	0		bit address		rel. address

Operation:

JB 0

(PC) ← 0 (PC) + 3

IF (bit) = 1

THEN

(PC) ← (PC)+ rel

JBC bit, rel

Function: Jump if Bit is set and Clear bit

Description:

If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

JBC ACC. 3, LABEL1

JBC ACC. 2, LABEL2

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

		0	1	0	0	0	0		bit address		rel. address

Operation: JB

(PC) ← 0 (PC) + 3

IF (bit) = 1

THEN

(bit) ← **0**

(PC) ← (PC)+ rel

JC rel

Function: Jump if Carry is set

Description:

If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

JC LABEL1

CPL C

JC LABEL2

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: **2**

Cycles: **2**

Encoding:

		0	0	0	0	0	0		rel. address

Operation: JC

(PC) ← 1 (PC) + 2

IF (C) = 1

THEN

(PC) ← (PC)+ rel

JMP @A+DPTR

Function: Jump indirect

Description:

Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
MOV DPTR, #JMP_TBL
JMP @A+DPTR
```

JMP-TBL: AJMP LABEL0

AJMP LABEL1

AJMP LABEL2

AJMP LABEL3

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0	1	1	1	0	0	1	1

Operation: JMP

(PC) ← (A) + (DPTR)

JNB bit, rel

Function: Jump if Bit is not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

JNB P1.3, LABEL1

JNB ACC. 3, LABEL2

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

		1	1	0	0	0	0		bit address		rel. address

Operation:

0

0 JNB

(PC) ←(PC) + 3

IF (bit) = 0

THEN (PC) ← (PC) + rel

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

JNC LABEL1

CPL C

JNC LABEL2

will clear the carry and cause program execution to continue at the instruction identified by the

label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

		0	1	0	0	0	0		rel. address

Operation:

1

0 JNC

(PC) ←(PC) + 2

IF (C) = 0

THEN (PC) ← (PC) + rel

JNZ rel

Function: Jump if Accumulator Not Zero

Description:

If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

JNZ LABEL1

INC A

JNZ LAEEL2

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

		1	1	0	0	0	0		rel. address

Operation:

1

0 JNZ

(PC) ←(PC) + 2

(A) ≠ 0IF

THEN (PC) ← (PC) + rel

JZ rel

Function: Jump if Accumulator Zero

Description:

If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally contains 01H. The instruction sequence,

JZ LABEL1

DEC A

JZ LAEEL2

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

		1	0	0	0	0	0		rel. address

Operation:

1

JZ 0

(PC) ← (PC) + 2

IF (A) = 0

THEN (PC) ← (PC) + rel

LCALL addr16

Function: Long call

Description:

LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: **Initially the Stack Pointer equals 07H. The label "SUBRTN" is assigned to progra**m memory location 1234H. After executing the instruction,

LCALL SUBRTN

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

Bytes: 3

Cycles: 2

Encoding:

		0	1	0	0	1	0		addr15-addr8		addr7-addr0

Operation:

0

0 LCALL

(PC) ← (PC) + 3

(SP) ← (SP) + 1

((SP)) ← (PC_{7-0})

(SP) ← (SP) + 1

((SP)) ← (PC $_{15-8}$)

(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long Jump

Description:

LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: **The label "JMPADR" is assigned to the instruction at program memory** location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 2

Encoding:

		0	0	0	0	1	0		addr15-addr8		addr7-addr0

Operation:

0

0 LJMP

(PC) ← $addr_{15-0}$

MOV <dest-byte> , <src-byte>

Function: Move byte variable

Description:

The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

MOV R0, #30H ; R0< = 30H

MOV A, @R0 ; A < = 40H

MOV R1, A ; R1 < = 40H

MOV B, @Rl ; B < = 10H

MOV @Rl, Pl ; RAM (40H) < = 0CAH

MOV P2, P1 ; P2 #0CAH

leaves the value 30H in register 0,40H in both the Accumulator and register 1,10H in register B, and 0CAH(11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

	1	1	0	1	r	r	r

Operation:

1 MOV

(A) ←(Rn)

***MOV A,direct**

Bytes: 2

Cycles: 1

Encoding:

1	1	1	0	0	1	0	1		direct address

Operation:

MOV

(A) ← (direct)

***MOV A, ACC is not a valid instruction.**

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding:

	1 1	1	0	0	1	1	i

Operation:

MOV

(A) ←((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	1	0	1	0	0		immediate data

Operation:

MOV

(A) ← #data

MOV Rn, A

Bytes: 1

Cycles: 1

Encoding:

	1	1	1	1	r	r	r

Operation:

1 MOV

(Rn) ← (A)

MOV Rn,direct

Bytes: 2

Cycles: 2

Encoding:

		1	0	1	r	r	r		direct address

Operation:

0

1 MOV

(Rn) ← (direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:

	1 0	1	1	1	r	r	r		immediate data

Operation:

MOV

(Rn) ← #data

MOV direct, A

Bytes: 2

Cycles: 1

Encoding:

		1	1	0	1	0	1		direct address

Operation:

1

1 MOV

(direct) ← (A)

MOV direct, Rn

Bytes: 2

Cycles: 2

Encoding:

		0	0	1	r	r	r		direct address

Operation:

0

1 MOV

(direct) ← (Rn)

MOV direct, direct

Bytes: 3

Cycles: 2

Encoding:

		0 1	0	0	1	0	1		dir. addr. (src)		dir. addr. (dest)

Operation:

MOV

(direct) ← 0 (direct)

MOV direct, @Ri

Bytes: 2

Cycles: 2

Encoding:

		0	0	0	1	1	i		direct address

Operation:

0

1 MOV

(direct) ← ((Ri))

MOV direct, #data

Bytes: 3

Cycles: 2

Encoding:

		1	1	0	1	0	1		direct address		immediate data

Operation:

1

0 MOV

(direct) ← #data

MOV @Ri, A

Bytes: 1

Cycles: 1

Encoding:

	1 1	1	1	0	1	1	i

Operation:

MOV

((Ri)) ← (A)

MOV @Ri, direct

Bytes: 2

Cycles: 2

Encoding:

		1	0	0	1	1	i		direct address

Operation:

0

1 MOV

((Ri)) ← (direct)

MOV @Ri, #data

Bytes: 2

Cycles: 1

Encoding:

		1	1	0	1	1	i		immediate data

Operation:

1

0 MOV

((Ri)) ← #data

MOV <dest-bit> , <src-bit>

Function: Move bit data

Description:

The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example:

The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

MOV P1.3, C

MOV C, P3.3

MOV P1.2, C

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C, bit

Bytes: 2

Cycles: 1

Encoding:

	0	1	0	0	0	1	0		bit address

Operation:

1

MOV

(C) ← (bit)

MOV bit, C

Bytes: 2

Cycles: 2

Encoding:

1	0	0	1	0	0	1	0		bit address

Operation:

MOV

(bit) ← (C)

MOV DPTR , #data 16

Function: Load Data Pointer with a 16-bit constant

Description:

The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example:

The instruction,

MOV DPTR, #1234H

will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 2

Encoding:

			1	0	0	0	0		immediate data15-8		immediate data7-0

Operation:

0

1 MOV

(DPTR)^0 ← #data$_{15-0}$ #data$_{7-0}$

DPH DPL ← **#data**$_{15-8}$

MOVC A , @A+ <base-reg>

Function: Move Code byte

Description:

The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit. Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example:

A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

REL-PC: INC A

```
MOVC A, @A+PC
RET
DB 66H
DB 77H
DB 88H
DB 99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the **Accumulator. The INC A before the MOVC instruction is needed to "get around" the RET** instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding:

			1	0	0	1	1

Operation:

0

1 0 MOVC

(A) ←((A)+(DPTR))

MOVC A,@A+PC

Bytes: 1

Cycles: 2

Encoding:

		0 1	0	0	0	1	1

Operation:

MOVC

(PC) ←0**(PC)+1**

(A) ←((A)+(PC))

MOVX <dest-byte> , <src-byte>

Function: Move External

Description:

The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example:

An external 256 byte RAM using multiplexed address/data lines (e. g. , an Intel 8155 RAM/ I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

MOVX A, @R1

MOVX @R0, A

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1

Cycles: 2

Encoding:

	1 1	1	0	0	0	1	i

Operation:

MOVX

(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

		1	0	0	0	0	0

Operation:

1 1 MOVX

(A) ←((DPTR))

MOVX @Ri, A

Bytes: 1

Cycles: 2

Encoding:

	1 1	1	1	0	0	1	i

Operation:

MOVX

((Ri))←(A)

MOVX @DPTR, A

Bytes: 1

Cycles: 2

Encoding:

		1	1	0	0	0	0

Operation:

1

1 MOVX

(DPTR)←(A)

MUL AB

Function: Multiply

Description:

MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1	0	1	0	0	1	0	0

Operation:

(A) $\leftarrow_{7\text{-}0}$ **(A)×(B)**

$(B)_{15\text{-}8}$

NOP

Function: No Operation

Description:

Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence.

CLR P2.7

NOP

NOP

NOP

NOP

SETB P2.7

Bytes: 1

Cycles: 1

Encoding:

		0	0	0	0	0	0

Operation:

0

0 NOP

(PC) ←(PC)+1

ORL <dest-byte> , <src-byte>

Function: Logical-OR for byte variables

Description:

ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

ORL A, R0

will leave the Accumulator holding the value 0D7H (1101011B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

ORL P1, #00110010B

will set bits 5,4, and 1of output Port 1.

ORL A, Rn

Bytes: 1

Cycles: 1

Encoding:

	1 0	0	0	1	r	r	r

Operation: ORL

(A) ←(A) ∨ (Rn)

ORL A, direct

Bytes: 2

Cycles: 1

Encoding:

	1 0	0	0	0	1	0	1		direct address

Operation: ORL

(A) ←(A) ∨ (direct)

ORL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1 0	0	0	0	1	1	i	

Operation: ORL

(A) ← (A) ∨ ((Ri))

ORL A, #data

Bytes: 2

Cycles: 1

Encoding:

1 0	0	0	0	1	0	0		immediate data	

Operation: ORL

(A) ← (A) ∨ #data

ORL direct, A

Bytes: 2

Cycles: 1

Encoding:

			0	0	0	1	0		direct address

Operation: 0

0 ORL

(direct) ← 1 (direct) ∨ (A)

ORL direct, #data

Bytes: 3

Cycles: 2

Encoding:

			0	0	0	1	1		direct address		immediate data

Operation: 0

0 ORL

(direct) ← 1 (direct) ∨ #data

ORL C, <src-bit>

Function: Logical-OR for bit variables

Description:

Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise.

A slash (" / ") preceding the operand in the assembly language indicates that the logical

complement of the addressed bit is used as the source value, but the source bit itself is not

affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV C, P1.0 ;LOAD CARRY WITH INPUT PIN P10
ORL C, ACC. 7 ;OR CARRY WITH THE ACC. BIT 7
ORL C, /OV ;OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

Bytes: 2

Cycles: 2

Encoding:

1 0	1	1	0	0	1	0		bit address	

Operation: ORL

(C) ← (C) ∨ (bit)

ORL C, /bit

Bytes: 2

Cycles: 2

Encoding:

1	0	1	0	0	0	0	0		bit address

Operation: ORL

(C) ←(C)$^{\vee}$ ($\overline{bit}$)

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

POP DPH

POP DPL

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

POP SP

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 2

Encoding:

		0 1	1	0	0	0	0		direct address

Operation: POP

(direct) ←^{1}((SP))

(SP) ← (SP) - 1

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

PUSH DPL

PUSH DPH

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2

Cycles: 2

Encoding:

		0 1	0	0	0	0	0		direct address

Operation: 1 PUSH

(SP) ←(SP) + 1

((SP)) ← (direct)

RET

Function: Return from subroutine

Description:

RET pops the high-and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RET

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0	0	1	0	0	0	1	0

Operation:

RET

$(PC_{15-8}) \leftarrow ((SP))$

(SP) $\leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

(SP) $\leftarrow (SP) - 1$

RETI

Function: Return from interrupt

Description:

RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0	0	1	1	0	0	1	0

Operation:

RETI

$(PC_{15-8}) \leftarrow ((SP))$

(SP) $\leftarrow (SP) - 1$

$(P_{7-0}C) \leftarrow ((SP))$

(SP) $\leftarrow (SP) - 1$

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RL A

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	0	1	1

Operation:

RL

(An+1) ← (An) n = 0-6

(A0) ← (A7)

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RLC A

leaves the Accumulator holding the value 8BH (10001011B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	0	0	1	1

Operation:

RLC

$(A_{n+1}) \leftarrow (A_n)$ n = 0-6

$(A_0) \leftarrow (C)$

$(C) \leftarrow (A_7)$

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RR A

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

		0	0	0	0	1	1

Operation:

0

RR 0 n = 0 - 6

$(A_n) \leftarrow (A_n+1)$

$(A_7) \leftarrow (A_0)$

RRC A

Function: Rotate Accumulator Right through the Carry flag

Description:

The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example:

The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RRC A

leaves the Accumulator holding the value 62H (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

		0	1	0	0	1	1

Operation:

0

0 RRC n = 0 - 6

$(A_n+1) \leftarrow (A_n)$

$(A_7) \leftarrow (C)$

(C) $\leftarrow (A_0)$

SETB <bit>

Function: Set bit

Description:

SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Example:

The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B).

The instructions,

SETB C

SETB P1.0

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

	1	0	1	0	0	1	1

Operation:

1

SETB

(C) ← 1

SETB bit

Bytes: 2

Cycles: 1

Encoding:

	1	0	1	0	0	1	0		bit address

Operation:

1

SETB

(bit) ← 1

SJMP rel

Function: Short Jump

Description:

Program control branches unconditionally to the address indicated. The branch destination is

computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128bytes preceding this instruction to 127 bytes following it.

Example: **The label "RELADR" is** assigned to an instruction at program memory location 0123H. The instruction,

SJMP RELADR

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(*Note:* Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H - 0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be an one-instruction infinite loop).

Bytes: **2**

Cycles: **2**

Encoding:

1	0	0	0	0	0	0	0		rel. address

Operation:

SJMP

PC( ) ← (PC)+2

(PC) ← (PC)+rel

SUBB A, <src-byte>

Function: Subtract with borrow

Description:

SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow)flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand).AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A, R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A, Rn

Bytes: **1**

Cycles: **1**

Encoding:

			1 1	1	r	r	r

Operation: 0 0 SUBB

(A) ← (A) - (C) - (Rn)

SUBB A, direct

Bytes: 2

Cycles: 1

Encoding:

			1 1 0	0	1	0	1		direct address

Operation: 0 SUBB

(A) ← (A) - (C) - (direct)

SUBB A, @Ri

Bytes: 1

Cycles: 1

Encoding:

			1 1	0	1	1	i

Operation: 0 0 SUBB

(A) ← (A) - (C) - ((Ri))

SUBB A, #data

Bytes: 2

Cycles: 1

Encoding:

			1 1	0	1	0	0		immediate data

Operation: 0 0 SUBB

(A) ← (A) - (C) - #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction.

No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	0	0

Operation: SWAP

$(A_{3-0}) \leftrightarrow (A)_{7-4}$

XCH A, <byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal

RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A, @R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A, Rn

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	1	r	r	r

Operation: XCH

(A) ↔ (Rn)

XCH A, direct

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	1	0	1		direct address

Operation: XCH

(A) ↔ (direct)

XCH A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	1	i

Operation: XCH

(A) ↔ ((Ri))

XCHD A, @Ri

Function: Exchange Digit

Description:

XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A, @R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	1	i

Operation: XCHD

(A_{3-0}) ↔ ($Ri)_{3-0}$

XRL <dest-byte>, <src-byte>

Function: Logical Exclusive-OR for byte variables

Description:

XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

(*Note*: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A, R0

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combination of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1, #00110001B

will complement bits 5,4 and 0 of output Port 1.

XRL A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	1	0	1	r	r	r

Operation: XRL

(A) ← (A) ∀ (Rn)

XRL A, direct

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	1	0	1		direct address

Operation: XRL

(A) ← (A) ∀ (direct)

XRL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	1	0	0	1	1	i

Operation: XRL

(A) ← (A) ∀ ((Ri))

XRL A, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	1	0	0		immediate data

Operation: XRL

(A) ← (A) ∀ #data

XRL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	0	1	0		direct address

Operation: XRL

(direct) ← (direct) ∀ (A)

XRL direct, #data

Bytes: 3

Cycles: 2

Encoding:

0	1	1	0	0	0	1	1		direct address		immediate data

Operation: XRL

(direct) ← (direct) ∀ #data

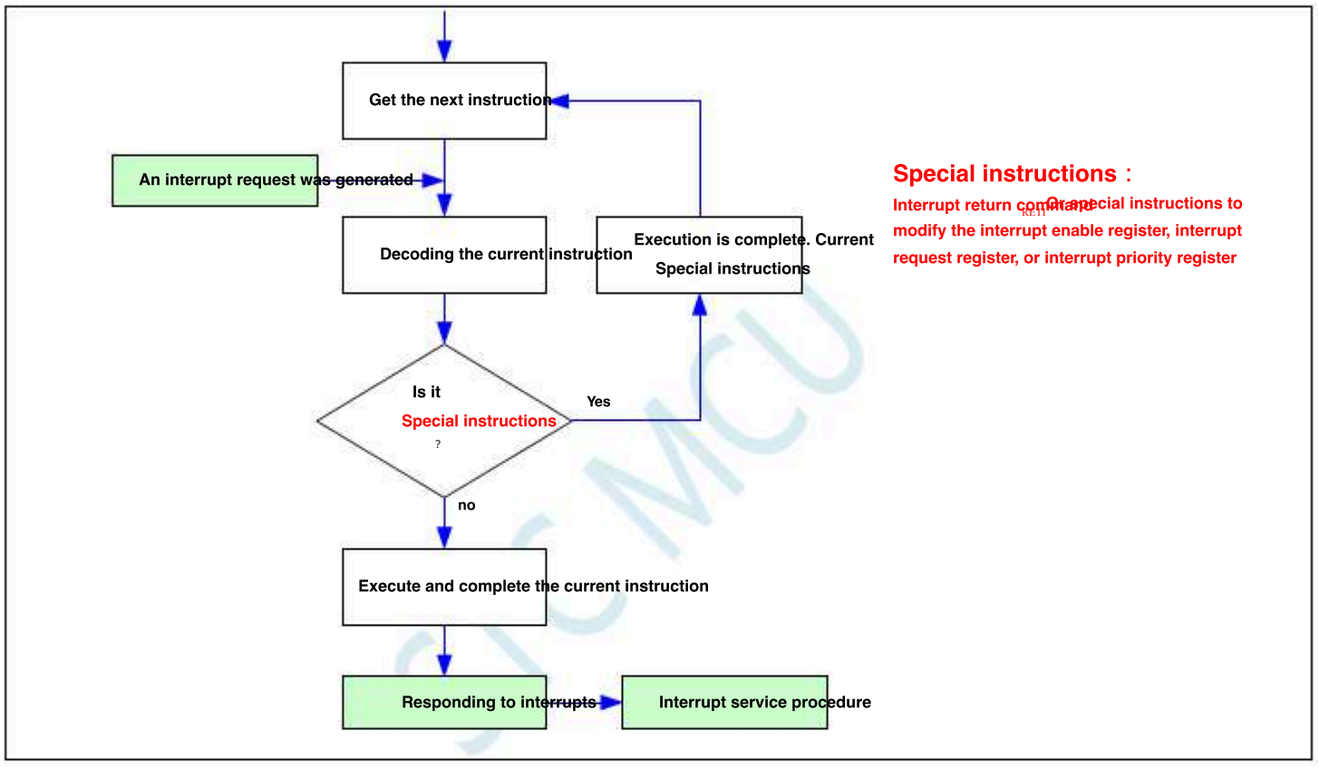
10.5 Interrupt response of multi-stage pipeline core

STC Enhanced 8051 (For example: STC8G/STC8H Series) and 32 bit 8051 (For example: STC32G Series) of MCU The core is designed as a multi-stage pipeline, and the 8051 (For example: STC89C52 design of interrupt response is slightly different from the traditional series.

For traditional 8051 (For example STC89C52 Series):

If the instruction currently being executed is a special instruction that accesses the interrupt enable register, interrupt request register, or interrupt priority register, or RETI that interrupts the return instruction to break the priority register, Between when the current special instruction is executed, another instruction can be executed to resp CPU request;

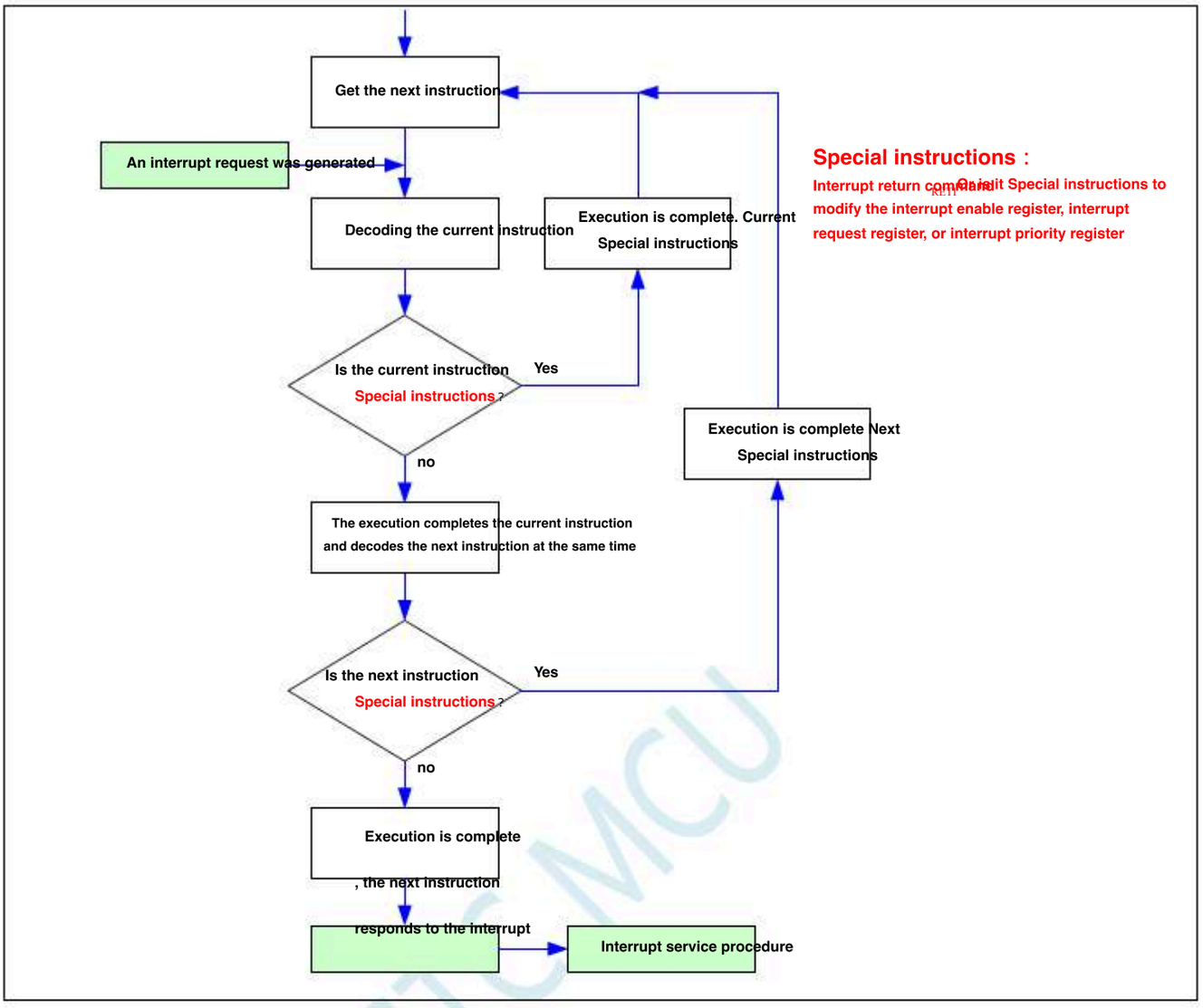
If the instruction currently being executed is not the special instruction referred to above, the interrupt request will be responded to imm



The enhanced version of single-chip microcomputer STC8G/STC8H , Because it is a multi-stage pipeline design, the response interr Will be better than traditional 8051 (for example: STC89C52 Series) Execute one more statement:

If the instruction currently being executed is a special instruction that accesses the interrupt enable register, interrupt request register, or interrupt priority register, RETI interrupts the return instruction to break the current special instruction is executed, the next instruction will be decrypted at the CPU's If an instruction is not a special instruction, the interrupt request can only be responded to when the next instruction is executed.;

If the instruction currently being executed is not the special instruction referred to above, the next instruction will be decrypted at the sa is executed . If the next instruction is not a special instruction, it will wait for the next instruction to be executed before responding to the inte



Special instructions：

Interrupt return completed Can't Special instructions to modify the interrupt enable register, interrupt request register, or interrupt priority register

11 Interrupt system

(The interrupt number used in the language program is greater than An error will be reported in compilation, please refer to the appendix f

The interrupt system is to enable the CPU to deal with real-time processing power for external emergencies.

the central processor CPU When something is being processed, there is an emergency request from the outside world, requesting CPU Pause current work ,

Instead of dealing with this emergency, after dealing with it, go back to the place where it was interrupted and continue the original work. This interruption. The component that implements this function is called the interrupt system, please consult CPU The source of the interrupt request is called the interrupt source. The interrup Multiple interrupt sources are allowed. When several interrupt sources interrupt the request at the same time and request to serve it, there is a CPU Which one interrupts the problem of the source request. Usually queued according to the priority of the interrupt source, the most urgent i source is prioritized, that is, each interrupt source is specified to have a priority level, CPU Always respond to the highest priority interrupt request first.

When an interrupt source request is being processed (the corresponding interrupt service program is executed), another prior Also a high interrupt source request. If you can suspend the service program to the original interrupt source, you can instead process the hig After processing, return to the original low-level interrupt service program. This process is called interrupt nesting. Such an interrupt system system, and an interrupt system without interrupt nesting function is called a single-level interrupt system.

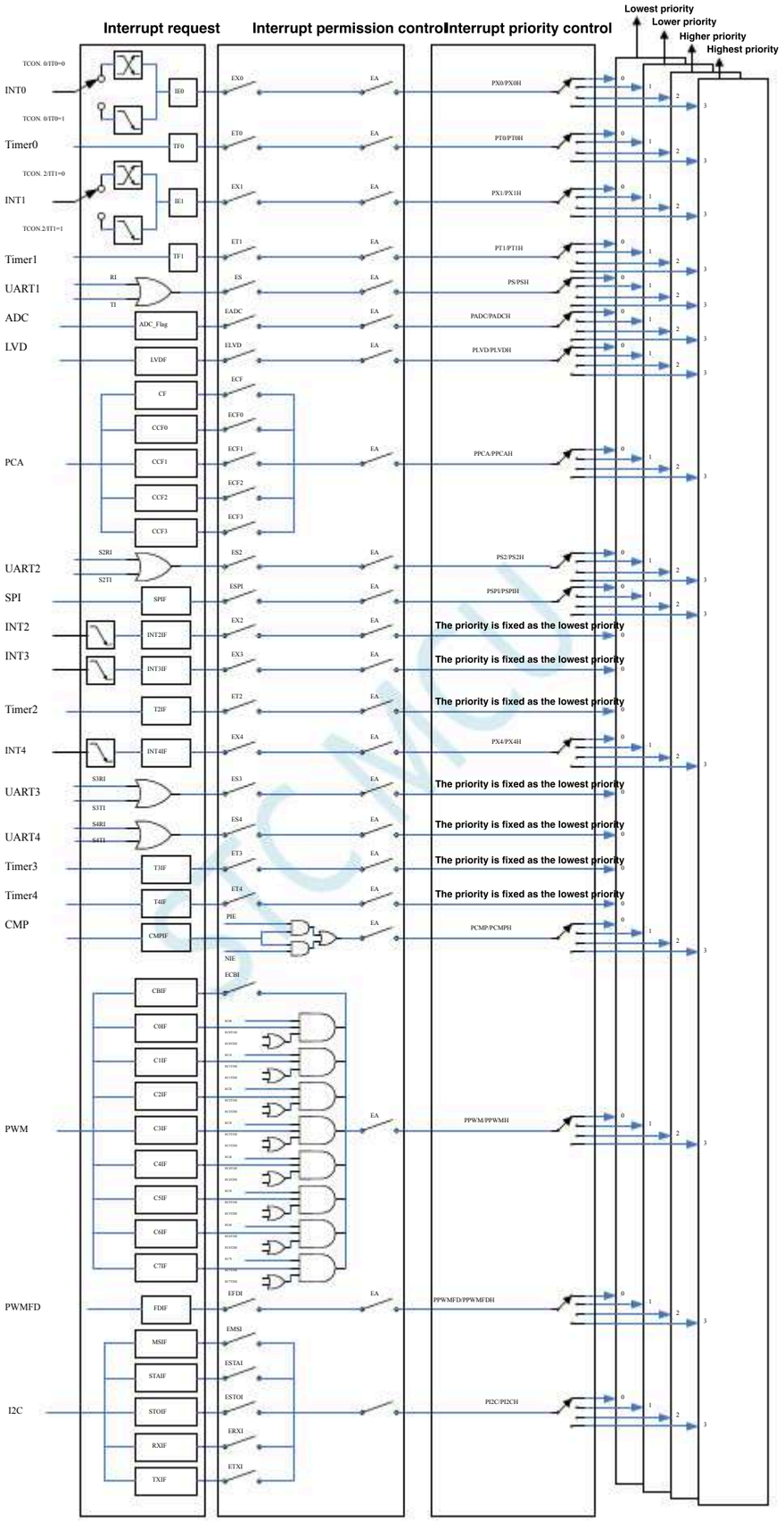
The user can use the off total interrupt permission bit (EA/IE.7) or the permission bit of the corresponding interrupt to block the corresponding interrupt The corresponding interrupt permission bit is used to make CPU In the corresponding interrupt application, each interrupt source can be independently controlled by software Status, the priority level of some interrupts can be set by software. High-priority interrupt requests can interrupt low-priority interrupts, on the contrary, low -priority interrupt requests cannot interrupt high-priority interrupts. When two interrupts of the same priority are generated at the same time, the query order will determine which interrupt the system responds to first.

11.1 STC12H Series interrupt source

√ in the table below indicates that the corresponding series has a corresponding interrupt source

Interrupt source	STC12H1K08 series
Interrupt (external interrupt	√
timer 0	√
External interrupt interrupt (timer 1 INT0) Interrupt (Timer0) INT1) Interrupt (Timer1)	√
Interrupt (UART1) Serial port 1	√
Analog-to-digital conversion ADC)	√
interrupt (low voltage detection interrupt (	√
Capture interrupt (CCP/PCA/PWM	√
Interrupt (UART2) Serial port 2	√
Serial peripheral interface interrupt (SPI)	√
External interrupt interrupt (INT2)	√
Interrupt (external interrupt	√
timer 2	√
External interrupt interrupt (timer 3	√
Timer 4 INT3) Interrupt (Timer2) INT4) Interrupt (Timer3) Interrupt (Timer4)	√
Comparator interrupt (CMP)	√
I2C Bus interrupt	√
PWMA	√
PWMB	√
P0 Port Interrupt port	√
P1 Interrupt port Interrupt	√
P2 Port Interrupt Port	√
P3 Interrupt Port Interrupt	√
P4 Port Interrupt	√
P5 Port Interrupt	√
P6 Port Interrupt	
P7 Port Interrupt	

11.2 STC12H Interrupt structure diagram



11.3 STC12H Series interrupt list

Interrupt source	Interrupt vector	order	Priority setting	Priority	Interrupt request bit	Interrupt allow bit
INT0	0003H	0	PX0PX0H	0/1/2/3	IE0	EX0
Timer0	000BH	1	PT0,PT0H	0/1/2/3	TF0	ET0
INT1	0013H	2	PX1,PX1H	0/1/2/3	IE1	EX1
Timer1	001BH	3	PT1,PT1H	0/1/2/3	TF1	ET1
UART1	0023H	4	PS,PSH	0/1/2/3	RI || TI	ES
ADC	002BH	5	PADC,PADCH	0/1/2/3	ADC_FLAG	EADC
LVD	0033H	6	PLVD,PLVDH	0/1/2/3	LVDF	ELVD
PCA	003BH	7	PPCA,PPCAH	0/1/2/3	CF	ECF
					CCF0	ECCF0
					CCF1	ECCF1
					CCF2	ECCF2
					CCF3	ECCF3
UART2	0043H	8	PS2,PS2H	0/1/2/3	S2RI || S2TI	ES2
SPI	004BH	9	PSPI,PSPIH	0/1/2/3	SPIF	ESPI
INT2	0053H	10		0	INT2IF	EX2
INT3	005BH	11		0	INT3IF	EX3
Timer2	0063H	12		0	T2IF	ET2
INT4	0083H	16	PX4,PX4H	0/1/2/3	INT4IF	EX4
Timer3	009BH	19		0	T3IF	ET3
Timer4	00A3H	20		0	T4IF	ET4
CMP	00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE|NIE
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI

Interrupt source	Interrupt vector	order	Priority setting	Priority	Interrupt request bit	Interrupt allow bit
PWMA	00D3H	26	PPWMA,PPWMAH	0/1/2/3	PWMA_SR	PWMA_IER
PWMB	00DBH	27	PPWMB,PPWMBH	0/1/2/3	PWMB_SR	PWMB_IER
P0 Interrupt	0123H	37		0	P0INTF	P0INTE
P1 interrupt	0133H	38		0	P1INTF	P1INTE
P2 interrupt	013BH	39		0	P2INTF	P2INTE
P3 interrupt	0143H	40		0	P3INTF	P3INTE
P5 interrupt	0153H	42		0	P5INTF	P5INTE

in C Declare the interrupt service program in the language

```
void INT0_Routine(void) interrupt 0;
void TM0_Rountine(void) interrupt 1;
void INT1_Routine(void) interrupt 2;
void TM1_Rountine(void) interrupt 3;
void UART1_Routine(void) interrupt 4;
void ADC_Routine(void) interrupt 5;
void LVD_Routine(void) interrupt 6;
void PCA_Routine(void) interrupt 7;
void UART2_Routine(void) interrupt 8;
void SPI_Routine(void) interrupt 9;
void INT2_Routine(void) interrupt 10;
void INT3_Routine(void) interrupt 11;
void TM2_Routine(void) interrupt 12;
void INT4_Routine(void) interrupt 16;
void TM3_Routine(void) interrupt 19;
void TM4_Routine(void) interrupt 20;
void CMP_Routine(void) interrupt 21;
void I2C_Routine(void) interrupt 24;
void USB_Routine(void) interrupt 25;
void PWMA_Routine(void) interrupt 26;
void PWMB_Routine(void) interrupt 27;

//void P0Int_Routine(void) interrupt 37;
//void P1Int_Routine(void) interrupt 38;
//void P2Int_Routine(void) interrupt 39;
//void P3Int_Routine(void) interrupt 40;
//void P5Int_Routine(void) interrupt 42;
```

of C31 The language interrupt service program cannot d For the statement, please refer to the processo

interrupt number to exceed the method, and the assembly language is not affected.

11.4 Interrupt related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	Interrupt allow register	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	Interrupt allow register 2	AFH	-	ET4	ET3	-	-	ET2	ESPI	ES2	0000,0000
INTCLKO	Interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
IP	Interrupt priority control register	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
IPH	High interrupt priority control register	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP2	Interrupt priority control register 2	B5H	-	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	High interrupt priority control register	B6H	-	PI2CH	PCMPH	PX4H	PPWMBH PPWMAH		PSPIH	PS2H	0000,0000
TCON	Timer control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
AUXINTIF	Extended external interrupt flag register	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	serial port 1 Control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	serial port 2 Control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
	Control register ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT			ADC_CHS[3:0]		000x,0000
SPSTAT	ADC SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
CMPCR1	comparator control register 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CMSCR	I²C Host control register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C Host status register	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C Slave control register	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C Slave status register	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
PWMA_IER	PWMA Interrupt enable register	FEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA Status register 1	FEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA Status register 2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMB_IER	PWMB Interrupt enable register	FEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB Status register 1	FEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	Status register 2 PWMB	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x
P0INTE	P0 Port interrupt Enable Register	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 port interrupt Enable Register	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 port interrupt Enable register	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 port interrupt Enable register	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P5INTE	P5 port interrupt Enable register	FD05H	-	-	-	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	0000,0000
P0INTF	P0 port interrupt flag Register	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 port interrupt flag Register	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 port interrupt flag Register	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 Port Interrupt flag Register	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P5INTF	P5	FD15H	-	-	-	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	0000,0000

11.4.1 Interrupt enable register (interrupt allow bit)

IE (Interrupt enable register)

Address symbol		B7	B6	B5	B4	B3	B2	B1	B0
A8H IE		EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA : The total interrupt allows the control bit. The role is to make the interrupt allow the formation of multi-level control. That is, each interrupt source Secondly, the control is also affec

Each interrupt source's own interrupt allows the control bit to control.

CPU Block all interrupt requests 0 :

Open interruption

CPU

1 :

: The allowable bit of low-voltage detection interrupt. ELVD

0 : Prohibit interruption of low-voltage detection

1 : allow interruption of low-voltage detection

EADC : A/D Convert the interrupt permission bit.

0: Prohibited Conversion interruptionA/D

1: Allow Conversion interruption

ES : Serial port interrupt permission bit.

: Serial port interruption

is prohibited 0

1 : Allow serial port interruption

ET1: Timing counter The overflow

interrupt

0: Prohibited interrupt permission bit. T1

1: Allow interrupt

EX1 :External interrupt interrupt

permission bit. interrupt 0 Prohibited INT1

interrupt 1 : Allow INT1

ET0: Timing counter The overflow interrupt permission bit.

T0 T0

interrupt 0: Prohibited interrupt

1: Allow T0

EX0 :External interrupt interrupt

permission bit. interrupt 0 Prohibited INT0

interrupt 1 : Allow INT0

IE2 (Interrupt enable register) 2

Symbol address		B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	-	ET4	ET3	-	-	ET2	ESPI	ES2

ET4 : Timing counter T4 The overflow interrupt permission bit.

0 : Prohibited T4 interrupt

1 : Allow T4 interrupt

ET3 : Timing counter The overflow interrupt permission bit.

T3 T3

interrupt 0 : Prohibited

1 : Allow T3 interrupt

ET2 : Timing counter T2 The overflow interrupt permission bit.

0 : Prohibited T2 interrupt

1 : Allow T2 interrupt

ESPI : SPI Interrupt permission bit.

0 : Prohibited interruptSPI

1 : Allow SPI interrupt

ES2 : Serial port interrupt permission bit.

0 : Serial port interruption is prohibited

1 : Serial port interruption is allowed

(External interrupt and clock output control register)INTCLKO

symbol address			B6	B5	B4	B3	B2	B1	B0
INTCLKO 8FH			EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4 : External interrupt interrupt allowable bit B7 。

0 : Prohibited EX4 interrupt

1 : Allow INT4 interrupt

EX3 :External interrupt interrupt

0 : Prohibited permission bit. INT3 interrupt

1 : Allow INT3 interrupt

EX2 :External interrupt interrupt permission bit. 2

interrupt 0 : Prohibited IN12

1 : Allow INT2 interrupt

(Comparator control register)1CMPCR1

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

PIE : The rising edge of the comparator interrupts the allowable bit.

0 : Disable the rising edge interrupt of the

1 comparator: Allow the rising edge interrupt of the comparator

NIE : The falling edge of the comparator interrupts the allowable bit.

0 : Disable the falling edge interrupt of the

1 comparator: Allow the falling edge interrupt of the comparator

Control register I2C

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	-	MSCMD[2:0]		
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI : I²C The host mode interrupt allows the bit.

0: Prohibited I²C Host mode interrupt

1: Allow I²C Host mode interrupt

ESTAI : I²C Receive from the machine The event interrupt permission

START bit. Slave receives event interrupt START

0: Prohibited I²C

1: Allow I²C Slave receives event interrupt

I²C Receiving data from the machine ERXI :

0: Prohibited I²C The slave receives the data to complete the event interrupt

1: Allow I²C The slave receives the data to complete the event interrupt

ETXI : done The event interrupt permission bit. I²C Send data from the machine

0: Prohibited I²C The slave sends data to complete the event interrupt

1: Allow I²C The slave sends data to complete the event interrupt

ESTOI : I²C Receive from the machine The event interrupt permission

STOP bit. Slave receives event interrupt STOP

0: Prohibited I²C

1: Allow I²C Slave receives event interrupt STOP

Interrupt enable register PWMA

Symbol address		B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER FEC4H		BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE

BIE :

PWMA The brake interrupt is allowed.

0 : Prohibited Brake interruption PWMA

1 : Allow Brake interruption PWMA

TIE : PWMA The interrupt permission bit is triggered.

0 : Prohibited PWMA Trigger interrupt

1 : Allow PWMA trigger interrupt

COMIE : PWMA Compare the interrupt permission bit.

0 : Prohibited PWMA Compare interrupts

1 : Allow PWMA compare interrupts

: PWMACC4IE Capture comparison channel 4 Interrupt permission bit.

0 : Prohibited PWMA 4 Interrupt capture comparison

1 : Allow PWMA channel Capture comparison channel Interrupt capture

: PWMACC3IE comparison channel 3 Interrupt permission bit.

0 : Prohibited PWMA 3 Interrupt capture comparison

1 : Allow PWMA channel Capture comparison channel Interrupt capture

CC2IE : PWMA comparison channel 2 Interrupt permission bit.

0 : Prohibited PWMA Capture comparison channel Interrupt capture

1 : Allow PWMA comparison channel Interrupt capture

CC1IE : PWMA comparison channel 1 Interrupt permission bit.

0 : Prohibited PWMA Capture compare channel interrupt

1 : Allow PWMA Capture compare channel interrupt 1

UIE :

PWMA Update the interrupt permission bit.

0 : Prohibited PWMA Update interrupt

1 : Allow PWMA update interrupt

Interrupt enable register PWMB

Symbol address		B7	B6	B5	B4	B3	B2	B1	B0
PWMB_IER FEE4H		BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE

BIE : PWMB The brake interrupt is allowed.

0: Prohibited PWMB Brake interrupt

1: Allow PWMB Brake interrupt

TIE :

PWMB The interrupt permission bit is triggered.

0: Prohibited PWMB Trigger interrupt

1: Allow PWMB trigger interrupt

COMIE : PWMB Compare the interrupt permission bit.

0: Prohibited PWMB Compare interrupts

1: Allow PWMB compare interrupts

: PWMBCC8IE Capture comparison channel 8 Interrupt permission bit.

0: Prohibited PWMB 8 Interrupt capture comparison

1: Allow PWMB 8 Channel Capture comparison channel Interrupt capture

: PWMBCC7IE comparison channel 7 Interrupt permission bit.

0: Prohibited PWMB 7 Interrupt capture comparison

1: Allow PWMB 7 Channel Capture comparison channel Interrupt capture

CC6IE : PWMB comparison channel 6 Interrupt permission bit.

0: Prohibited PWMB 6 Capture comparison channel Interrupt capture

1: Allow PWMB 6 comparison channel Interrupt capture

CC5IE : PWMB comparison channel 5 Interrupt permission bit.

0: Prohibited PWMB Capture compare 5 channel interrupt

1: Allow PWMB Capture compare channel interrupt 5

UIE :

PWMB Update the interrupt permission bit.

0: Prohibited PWMB Update interrupt

1: Allow PWMB update interrupt

Port interrupt enable register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P5INTE	FD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE

: Port interrupt enables the control bit (PnINTE.x (n=0~7) x=0~7

0: Closed Port interrupt function Pn.

1: Enable Pn. Port interrupt function

11.4.2 Interrupt request register (interrupt flag)

Timer control register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 : Timer$_1$ Overflow interrupt flag. During the interrupt service program, the hardware is automatically cleared to zero.

TF0 : Timer$_0$ Overflow interrupt flag. During the interrupt service program, the hardware is automatically cleared to zero. :External

IE1 interrupt$_1$ Interrupt request flag. During the interrupt service program, the hardware is automatically cleared to zero. :External

IE0 interrupt$_0$ Interrupt request flag. During the interrupt service program, the hardware is automatically cleared to zero.

Interrupt flag auxiliary register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
AUXINTIF	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF

INT4IF :External interrupt$_4$ Interrupt request flag. The hardware is automatically cleared to zero in the interrupt service program.

INT3IF :External interrupt$_3$ Interrupt request flag. The hardware is automatically cleared to zero in the interrupt service program.

INT2IF :External interrupt$_2$ Interrupt request flag. The hardware is automatically cleared to zero in the interrupt service program.

T4IF : Timer$_4$ Overflow interrupt flag. The hardware is automatically cleared to zero in the interrupt service program (Note: This bit is a write-c

T3IF : Timer$_3$ Overflow interrupt flag. The hardware is automatically cleared to zero in the interrupt service program (Note: This bit is a write-c

T2IF : Timer$_2$ Overflow interrupt flag. The hardware is automatically cleared to zero in the interrupt service program (Note: This bit is a write-c

Serial port control register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

TI : Serial port 1 Send a complete interrupt request flag. Software clearance is required. : Serial

RI port 1 The interrupt request flag is received to complete. Software clearance is required.

S2TI : Serial port 2 Send a complete interrupt request flag. Software clearance is required. : Serial

S2RI port 2 The interrupt request flag is received to complete. Software clearance is required.

Power management register

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

LVDF : Low voltage detection interrupt request flag. Software clearance is required.

ADC Control register

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

ADC_FLAG : ADC The conversion completes the interrupt request flag. Software clearance is required.

Status register SPI

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

: SPISPIF Interrupt request flag for data transmission completion. Software clearance is required.

Comparator control register 1

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

CMPIF : Comparator interrupt request flag. Software clearance is required.

Status register I2C

Symbol address		B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST FE82H		MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO
I2CSLST FE84H		SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO

Host mode interrupt request flag. Software clearance is required. : MSIF I²C

ESTAI : Event interrupt request flag. Software clearance is required.

ERXI : done Event interrupt request flag. Software clearance is required.

ETXI : done Event interrupt request flag. Software clearance is required.

ESTOI : I²C Receive from the machine START I²C Receiving data from the machine I²C Send data from the machine I²C Receive from the machine STOP Event interrupt request flag. Software clearance is required.

Status register PWMA

Symbol address		B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1 FEC5H		BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
PWMA_SR2 FEC6H		-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-

BIF : PWMA Brake interrupt request flag. Software clearance is required. The interrupt

TIF : PWMA request flag is triggered. Software clearance is required.

COMIF : PWMA Compare interrupt request flags. Software clearance is required.

CC4IF : PWMA channel$_4$ A capture comparison interrupt request flag has occurred. Software clearance is required.

CC3IF : PWMA channel$_3$ A capture comparison interrupt request flag has occurred. Software clearance is required.

CC2IF : PWMA channel$_2$ A capture comparison interrupt request flag has occurred. Software clearance is required.

CC1IF : PWMA channel$_1$ A capture comparison interrupt request flag has occurred. Software clearance is required.

TIF : PWMA Update the interrupt request flag. Software clearance is required.

: PWMACC4OF channel$_4$ A duplicate capture interrupt request flag occurred. Software clearance is required.

: PWMACC3OF channel$_3$ A duplicate capture interrupt request flag occurred. Software clearance is required.

CC2OF : PWMA channel$_2$ A duplicate capture interrupt request flag occurred. Software clearance is required.

CC1OF : PWMA channel$_1$ A duplicate capture interrupt request flag occurred. Software clearance is required.

Status register PWMB

Symbol address		B7	B6	B5	B4	B3	B2	B1	B0
PWMB_SR1 FEE5H		BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF
PWMB_SR2 FEE6H		-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

BIF : PWMB Brake interrupt request flag. Software clearance is required. The interrupt

TIF : PWMB request flag is triggered. Software clearance is required.

COMIF : PWMB Compare interrupt request flags. Software clearance is required.

CC8IF : PWMB channel$_8$ A capture comparison interrupt request flag has occurred. Software clearance is required.

CC7IF : PWMB channel$_7$ A capture comparison interrupt request flag has occurred. Software clearance is required.

CC6IF : PWMB channel$_6$ A capture comparison interrupt request flag has occurred. Software clearance is required.

CC5IF : PWMB channel$_5$ A capture comparison interrupt request flag has occurred. Software clearance is required.

TIF : PWMB Update the interrupt request flag. Software clearance is required.

: PWMBCC8OF channel$_8$ A duplicate capture interrupt request flag occurred. Software clearance is required.

: PWMBCC7OF channel$_7$ A duplicate capture interrupt request flag occurred. Software clearance is required.

: PWMBCC6OF channel$_6$ A duplicate capture interrupt request flag occurred. Software clearance is required.

CC5OF : PWMB channel$_5$ A duplicate capture interrupt request flag occurred. Software clearance is required.

Port interrupt flag register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P5INTF	FD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF

: Port interrupt request flag (PnINTF. x n=0~7, x=0~7)

Pn. x 0: The port did not interrupt the request

There is an interrupt request at the port. If the interrupt is enabled, the interrupt service program will be entered.

Pn. x

0

1 :

The logo needs to be cl

11.4.3 Interrupt priority register

Except that the , Timer, timer, timer In addition to all port interrupts, other interrupts have priority interrupts. priority can be set

Interrupt priority control register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	-	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2
IP2H	B6H	-	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H

PX0H,PX0 :External interrupt$_0$ Interrupt priority control bit

00 : INT0 The interrupt priority is level (lowest level)

01 : INT0 priority, the interrupt priority is the Level (Lower level)

10 : INT0 Level (Higher level)

11 : INT0 interrupt priority, the interrupt priority is Level (highest level)

: Timer$_0$ Interrupt priority control bit PT0H,PT0

00 :The timer interrupt priority is Level (lowest level)

The interrupt priority is (lower level)

10 : The timer interrupt priority is Level (higher level)

11 : Timer Interrupt priority is priority$_3$(Most advanced)

PX1H,PX1 :External interrupt$_1$ Interrupt priority control bit

00 : INT1 The interrupt priority is level (lowest level)

01 : INT1 interrupt priority, the interrupt Level (Lower level)

10 : INT1 priority is the interrupt priority Level (higher level)

11 : INT1 Interrupt priority is priority$_3$(Most advanced)

PT1H,PT1 : timer$_1$ Interrupt priority control bit

00: The interrupt priority is level (lowest level)

Timer : Timer interrupt priority is : Timer level (lower level)

10 interrupt priority The priority is Level (higher level)

11: Timer Interrupt priority is priority$_3$(Most advanced)

: serial port$_1$ Interrupt priority control bit PSH,PS

00 :Serial port interrupt priority is priority (lowest level)

01: The interrupt priority is level (lower level)

Serial port serial port interrupt priority is level (higher level)

11 :Serial port interrupt priority is priority$_3$(Most advanced) PADCH,PADC : ADC Interrupt priority control bit Interrupt priority is

00 : ADC Level (lowest level)

01 : ADC Level (lower level)

10 : ADC level (higher level) Interrupt priority is

11 : ADC level$_3$(Highest level) : low voltage detection

PLVDH,PLVD interrupt priority control bit

00 : LVD The level (lowest level) interrupt

01 : LVD priority is the interrupt priority is the level 01 (Lower level)

10 : LVD Interrupt priority is Level (higher level)

11 : LVD Interrupt priority is Level (highest level)

: Serial port 2 Interrupt priority control bit PS2H,PS2

00 : The serial port interrupt priority is 0 Level 2 (Lowest level)

01 : The serial port interrupt priority is 1 Level 2 (Lower level)

10 : The serial port interrupt priority is 2 Level 2 (Higher level)

11 : Serial port 2 The interrupt priority is 3 Level (highest level)

PSPIH,PSPI : SPI Interrupt priority control bit Interrupt

00 : SPI priority is priority 0 (Lowest level)

01 : SPI Interrupt priority is level (lower level) 1

10 : SPI Interrupt priority is priority 2 (Higher

11 : SPI level) Interrupt priority is level (highest level) 3

PPWMAH,PPWMA : Advanced PWMA Interrupt priority control bit

00 : Advanced PWMA Interrupt priority is priority 0 (Lowest level)

01 : Advanced PWMA Interrupt priority is level 1 (Lower level)

10 : Advanced PWMA Interrupt priority is level (higher level) 2

11 : Advanced PWMA Interrupt priority is priority 3 (Most advanced)

PPWMBH,PPWMB : Advanced PWMB Interrupt priority control bit

00 : Advanced PWMB Interrupt priority is priority 0 (Lowest level)

01 : Advanced PWMB Interrupt priority is level 1 (Lower level)

10 : Advanced PWMB Interrupt priority is priority 2 (Higher level)

11 : Advanced PWMB Interrupt priority is level (highest level) 3

PX4H,PX4 :External interrupt 4 Interrupt priority control bit

00 : INT4 The interrupt priority level (lowest level)

01 : INT4 interrupt priority, the level (lower level)

10 : INT4 priority is the interrupt priority 2 Level (higher level)

11 : INT4 Interrupt priority is priority 3 (Highest level)

PCMPH,PCMP : Comparator interrupt priority control bit

00 : CMP interrupt priority is level (lowest level) 0

01 : CMP Interrupt priority is first-level (lower-level) 1

10 : CMP Interrupt priority is level (higher level) 2

11 : CMP Interrupt priority is priority 3 (Most advanced)

: Interrupt priority control bit

I2CPI2CH,PI2C

00 : I2C Interrupt priority is level (lowest level) 0

01 : I2C Interrupt priority is priority 1 (Lower level)

10 : I2C Interrupt priority is priority 2 (Higher level)

11 : I2C Interrupt priority is level (highest level) 3

11.5 Sample program

11.5.1 INT0 Interrupt (rising and falling edges), can support both rising and falling edges

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;
sbit P10 = P1^0;
sbit P11 = P1^1;

void INT0_Isr() interrupt 0
{
    if (P32)                    //Judge the rising and falling edges
    {
        P10 = ! P10;            //Test port
    }
    else
    {
        P11 = ! P11;            //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;
    IT0 = 0;                    //Enable INT0 Rising and falling edge interrupts
    EX0 = 1;                    //Enable INT0 interrupt
    EA = 1;
```

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

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     0003H
        LJMP    INT0ISR

        ORG     0100H
INT0ISR:
        JB      P3.2,RISING     ; Judge the rising and falling edges
        CPL     P1.0            ;Test port
        RETI
RISING:
        CPL     P1.1            ;Test port
        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     IT0             ;Enable INT0 Rising and falling edge interrupts
        SETB    EX0             ;Enable INT0 interrupt
        SETB    EA
        JMP     $

        END
```

11.5.2 INT0 Interrupt (falling edge)

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;
sbit P10 = P1^0;

void INT0_Isr() interrupt 0
{
    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;
    IT0 = 1;                        //Falling edge interrupt INT0
    EX0 = 1;                        //Enable interrupt
    EA = 1;                         //Enable interrupt
    while (1);

}
```

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     0003H
                LJMP    INT0ISR

                ORG     0100H
INT0ISR:
                CPL     P1.0                ;Test port
                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    IT0                 ;Enable Falling edge interrupt INT0
                SETB    EX0                 ;Enable INT0 interrupt
                SETB    EA
                JMP     $

                END
```

11.5.3 INT1 Interrupt (rising and falling edges), can support both rising and falling edges

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;

sbit        P10         =   P1^0;
sbit        P11         =   P1^1;

void INT1_Isr() interrupt 2
{
    if (INT1)                       // Judge the rising and falling edges
    {
        P10 = ! P10;                //Test port
}
else
{
        P11 = ! P11;                //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;
    IT1 = 0;                        //Enable INT1 Rising and falling edge interrupts
    EX1 = 1;                        //Enable INT1 interrupt
    EA = 1;
    while (1);
}
```

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     0013H
                LJMP    INT1ISR

                ORG     0100H
INT1ISR:
                JB      INT1,RISING             ; Judge the rising and falling edges
                CPL     P1.0                    ; Test port
                RETI
RISING:
                CPL     P1.1                    ; Test port
                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     IT1                     ; Enable INT1 Rising and falling edge interrupts
                SETB    EX1                     ; Enable INT1 interrupt
                SETB    EA
                JMP     $

                END
```

11.5.4 INT1 Interrupt (falling edge)

C Language code

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

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

sfr P1M1   = 0x91;
sfr P0M1   = 0x92;
sfr P0M0   = 0x93;
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 INT1_Isr() interrupt 2
{
    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;
    IT1 = 1;                                        //Enable Falling edge interrupt INT1
    EX1 = 1;                                        //Enable interrupt
    EA = 1;
    while (1);
}
```

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     0013H
        LJMP    INT1ISR

        ORG     0100H
INT1ISR:
        CPL     P1.0                                ;Test port
```

```
        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    IT1             ;Enable Falling edge interrupt INT1
        SETB    EX1             ;Enable INT1 interrupt
        SETB    EA
        JMP     $

        END
```

11.5.5 INT2 Interrupt (falling edge) Only supports falling edge interrupts

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 INTCLKO = 0x8f;
#define EX2  0x10
#define EX3  0x20
#define EX4  0x40
sbit P10 = P1^0;

void INT2_Isr() interrupt 10
{
    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;
    INTCLKO = EX2;                  //Enable INT2 interrupt
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

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     0053H
            LJMP    INT2ISR

            ORG     0100H
INT2ISR:
            CPL     P1.0            ;Test port
            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     INTCLKO,#EX2        ;Enable INT2 interrupt
        SETB    EA
        JMP     $

        END
```

11.5.6 INT3 Interrupt (falling edge) Only supports falling edge interrupts

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 INTCLKO
#define EX2 = 0x8f;
#define EX3 0x10
#define EX4 0x20
sbit P10 = P1^0;
                0x40

void INT3_Isr() interrupt 11
{
    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;
    INTCLKO = EX3;                  //Enable INT3 interrupt
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

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     005BH
            LJMP    INT3ISR

            ORG     0100H
INT3ISR:
            CPL     P1.0            ;Test port
            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     INTCLKO,#EX3        ;Enable INT3 interrupt
        SETB    EA
        JMP     $

        END
```

11.5.7 INT4 Interrupt (falling edge) Only supports falling edge interrupts

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 INTCLKO = 0x8f;
#define EX2  0x10
#define EX3  0x20
#define EX4  0x40
sbit P10 = P1^0;

void INT4_Isr() interrupt 16
{
    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;
```

```
    INTCLKO = EX4;                                  //Enable INT4 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

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     0083H
            LJMP    INT4ISR

            ORG     0100H
INT4ISR:
            CPL     P1.0                            ;Test port
            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     INTCLKO,#EX4                    ;Enable INT4 interrupt
            SETB    EA
            JMP     $

            END
```

0 Timer interrupt 11.5.8

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;
sbit P10 = P1^0;

void TM0_Isr() interrupt 1
{
    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;
    TMOD = 0x00;
    TL0 = 0x66;                                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                        //Start the timer
    ET0 = 1;                                        //Enable timer interrupt
    EA = 1;
    while (1);
}
```

Assembly code

;The test operating frequency is

```
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     000BH
        LJMP    TM0ISR

        ORG     0100H
TM0ISR:
        CPL     P1.0                    ;Test port
        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     TMOD,#00H
        MOV     TL0,#66H                ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB    TR0                     ;Start the timer
        SETB    ET0                     ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

1 Timer interrupt 11.5.9

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;
sbit P10
                                        =       P1^0;


void TM1_Isr() interrupt 3
{
        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;
        TMOD = 0x00;
        TL1 = 0x66;                                             //65536-11.0592M/12/1000
        TH1 = 0xfc;
        TR1 = 1;                                                //Start the timer
        ET1 = 1;                                                //Enable timer interrupt
        EA = 1;
        while (1);

}
```

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     001BH
                LJMP    TM1ISR

                ORG     0100H
TM1ISR:
                CPL     P1.0                ;Test port
                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     TMOD,#00H
                MOV     TL1,#66H            ;65536-11.0592M/12/1000
                MOV     TH1,#0FCH
                SETB    TR1                 ;Start the timer
                SETB    ET1                 ;Enable timer interrupt
                SETB    EA

                JMP     $

                END
```

2 Timer interrupt 11.5.10

C Language code

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

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

sfr T2L  = 0xd7;
sfr T2H  = 0xd6;
sfr AUXR = 0x8e;
sfr IE2  = 0xaf;
```

```
#define   ET2             0x04
sfr       AUXINTIF    =   0xef;
#define   T2IF            0x01

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

sbit      P10         =   P1^0;

void TM2_Isr() interrupt 12
{
    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;
    T2L = 0x66;                                     //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                                    //Start the timer
    IE2 = ET2;                                      //Enable timer interrupt
    EA = 1;
    while (1);

}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

```
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     0063H
        LJMP    TM2ISR

        ORG     0100H
TM2ISR:
        CPL     P1.0                ;Test port
        RETI

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

        MOV     T2L,#66H            ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H           ;Start the timer
        MOV     IE2,#ET2            ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

3 Timer interrupt 11.5.11

C Language code

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

#include "reg51.h"
```

```
#include "intrins. h"

sfr         T3L             =   0xd5;
sfr         T3H             =   0xd4;
sfr         T4T3M           =   0xd1;
sfr         IE2             =   0xaf;
#define     ET3                 0x20
sfr         AUXINTIF        =   0xef;
#define     T3IF                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        P10             =   P1^0;

void TM3_Isr() interrupt 19
{
    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;
    T3L = 0x66;                             //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;                           //Start the timer
    IE2 = ET3;                              //Enable timer interrupt
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

T3L     DATA    0D5H
```

```
T3H             DATA    0D4H
T4T3M           DATA    0D1H
IE2             DATA    0AFH
ET3             EQU     20H
AUXINTIF        DATA    0EFH
T3IF            EQU     02H

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

                ORG     0000H
                LJMP    MAIN
                ORG     009BH
                LJMP    TM3ISR

                ORG     0100H
TM3ISR:
                CPL     P1.0            ;Test port
                RETI

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

                MOV     T3L,#66H        ;65536-11.0592M/12/1000
                MOV     T3H,#0FCH
                MOV     T4T3M,#08H      ;Start the timer
                MOV     IE2,#ET3        ;Enable timer interrupt
                SETB    EA

                JMP     $

                END
```

4 Timer interrupt 11.5.12

C Language code

The test operating frequency is 11.0592MHz

```
#include "reg51. h"
#include "intrins. h"
sfr T3L = 0xd5;
sfr T3H = 0xd4;
sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T4T3M = 0xd1;
sfr IE2 = 0xaf;
#define ET3 0x20
#define ET4 0x40
sfr AUXINTIF = 0xef;
#define T3IF 0x02
#define T4IF 0x04
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;
sbit P10 = P1^0;

void TM4_Isr() interrupt 20
{
    P10 = ! P10;                                //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;
    T4L = 0x66;
    T4H = 0xfc;                                 //65536-11.0592M/12/1000
```

```
    T4T3M = 0x80;                   //Start the timer
    IE2 = ET4;                      //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

T3L     DATA    0D5H
T3H     DATA    0D4H
T4L     DATA    0D3H
T4H     DATA    0D2H
T4T3M   DATA    0D1H
IE2     DATA    0AFH
ET3     EQU     20H
ET4     EQU     40H
AUXINTIF DATA   0EFH
T3IF    EQU     02H
T4IF    EQU     04H

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

        ORG     0000H
        LJMP    MAIN
        ORG     00A3H
        LJMP    TM4ISR

        ORG     0100H
TM4ISR:
        CPL     P1.0                ;Test port
        RETI

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

```
        MOV     P5M1, #00H

        MOV     T4L,#66H        ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#80H      ;Start the timer
        MOV     IE2,#ET4        ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

11.5.13 UART1 interrupt

C Language code

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

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

sfr T2L     = 0xd7;
sfr T2H     = 0xd6;
sfr AUXR    = 0x8e;
sfr P1M1    = 0x91;
sfr P1M0    = 0x92;
sfr P0M1    = 0x93;
sfr P0M0    = 0x94;
sfr P2M1    = 0x95;
sfr P2M0    = 0x96;
sfr P3M1    = 0xb1;
sfr P3M0    = 0xb2;
sfr P4M1    = 0xb3;
sfr P4M0    = 0xb4;
sfr P5M1    = 0xc9;
sfr P5M0    = 0xca;
sbit P10    = P1^0;
sbit P11    = P1^1;

void UART1_Isr() interrupt 4
{
    if (TI)
    {
        TI = 0;             //Clear interrupt sign
        P10 = ! P10;        //Test port
    }
    if (RI)
    {
        RI = 0;             //Clear interrupt sign
        P11 = ! P11;        //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;
    SCON = 0x50;
    T2L = 0xe8;                     //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x15;                    //Start the timer
    ES = 1;                         //Enable serial port interrupt
    EA = 1;
    SBUF = 0x5a;                    //Send test data
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

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     0023H
        LJMP    UART1ISR

        ORG     0100H
UART1ISR:
        JNB     TI,CHECKRI
        CLR     TI              ;Clear interrupt sign
        CPL     P1.0            ;Test port
CHECKRI:
        JNB     RI,ISREXIT
        CLR     RI              ;Clear interrupt sign
```

```
                CPL     P1.1                    ;Test port
ISREXIT:
                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     SCON,#50H
                MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#15H               ;Start the timer
                SETB    ES                      ;Enable serial port interrupt
                SETB    EA
                MOV     SBUF,#5AH               ;Send test data

                JMP     $

                END
```

11.5.14 UART2 interrupt

C Language code

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

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

sfr T2L  = 0xd7;
sfr T2H  = 0xd6;
sfr AUXR = 0x8e;
sfr S2CON = 0x9a;
sfr S2BUF = 0x9b;
sfr IE2  = 0xaf;
#define ES2 0x01
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    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART2_Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
    S2CON &= ~0x02;                 //Clear interrupt sign
    P12 = ! P12;                    //Test port
    }
    if (S2CON & 0x01)
    {
    S2CON &= ~0x01;                 //Clear interrupt sign
    P13 = ! P13;                    //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;
    S2CON = 0x10;
    T2L = 0xe8;                     //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                    //Start the timer
    IE2 = ES2;                      //Enable serial port interrupt
    EA = 1;
    S2BUF = 0x5a;                   //Send test data
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
S2CON   DATA    9AH
S2BUF   DATA    9BH
IE2     DATA    0AFH
ES2     EQU     01H
```

```
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     0043H
        LJMP    UART2ISR

        ORG     0100H
UART2ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     A,S2CON
        JNB     ACC. 1,CHECKRI
        ANL     S2CON,#NOT 02H          ;Clear interrupt sign ;Test port
        CPL     P1.2
CHECKRI:
        MOV     A,S2CON
        JNB     ACC. 0,ISREXIT
        ANL     S2CON,#NOT 01H          ;Clear interrupt sign ;Test port
        CPL     P1.3
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

        MOV     S2CON,#10H
        MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H               ;Start the timer
        MOV     IE2,#ES2                ;Enable serial port interrupt
        SETB    EA
        MOV     S2BUF,#5AH              ;Send test data
```

```
        JMP     $

        END
```

11.5.15 ADC interrupt

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;
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
    P0 = ADC_RES;           //Test port
    P2 = ADC_RESL;          //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;
    ADCCFG = 0x00;
```

```
    ADC_CONTR = 0xc0;           //Enable and start ADC module
    EADC = 1;                   //Enable ADC Inbroken
    EA = 1;

    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
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  ;Clear interrupt sign
            MOV     P0,ADC_RES          ;Test port
            MOV     P2,ADC_RESL         ;Test port
            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     ADCCFG,#00H
            MOV     ADC_CONTR,#0C0H     ;Enable and start ADC module
```

```
        SETB    EADC            ;Enable ADC interrupt
        SETB    EA

        JMP     $

        END
```

11.5.16 LVD interrupt

C Language code

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

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

sfr RSTCFG = 0xff;
#define ENLVR 0x40          //RSTCFG.6
#define LVD2V2 0x00         //LVD@2.2V
#define LVD2V4 0x01         //LVD@2.4V
#define LVD2V7 0x02         //LVD@2.7V
#define LVD3V0 0x03         //LVD@3.0V
sbit ELVD = IE^6;
#define LVDF 0x20           //PCON.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;
sbit P10 = P1^0;


void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;          //Clear interrupt sign
    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;
    PCON &= ~LVDF;                  // The interrupt flag needs to be cleared for power-up
    RSTCFG = LVD3V0;                //Set up LVD The voltage is 3.0V
    ELVD = 1; //Enable LVD interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

RSTCFG  DATA    0FFH
ENLVR   EQU     40H             ;RSTCFG.6
LVD2V2  EQU     00H             ;LVD@2.2V
LVD2V4  EQU     01H             ;LVD@2.4V
LVD2V7  EQU     02H             ;LVD@2.7V
LVD3V0  EQU     03H             ;LVD@3.0V
ELVD    BIT     IE.6
LVDF    EQU     20H             ;PCON.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     0033H
        LJMP    LVDISR

        ORG     0100H
LVDISR:
        ANL     PCON,#NOT LVDF  ;Clear interrupt sign
        CPL     P1.0            ;Test port
        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

ANL     PCON,#NOT LVDF      ; The interrupt flag needs to be cleared for power-up
MOV     RSTCFG,# LVD3V0     ;Set up LVD The voltage is 3.0V
SETB    ELVD                ;Enable LVD interrupt
SETB    EA
JMP     $

END
```

11.5.17 Comparator interrupt

C Language code

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

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

sfr     CMPCR1  =   0xe6;
sfr CMPCR2      =   0xe7;
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;
sbit P10        =   0xca;
                =   P1^0;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;        //Clear interrupt sign
    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;
    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //Enable comparator module
    CMPCR1 |= 0x30;                 // Enable comparator edge interrupt
    CMPCR1 &= ~0x08;                //P3.6 CMP+ The input pin
    CMPCR1 |= 0x04;                 //P3.7 CMP- Input pin is
    CMPCR1 |= 0x02;                 //Enable comparator output
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

CMPCR1  DATA    0E6H
CMPCR2  DATA    0E7H

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

        ORG     0000H
        LJMP    MAIN
        ORG     00ABH
        LJMP    CMPISR

        ORG     0100H
CMPISR:
        ANL     CMPCR1,#NOT 40H ;Clear interrupt sign
        CPL     P1.0            ;Test port
        RETI

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

```
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     CMPCR2,#00H
MOV     CMPCR1,#80H          ;Enable the comparator module to enable
ORL     CMPCR1,#30H          the comparator edge interrupt
ANL     CMPCR1,#NOT 08H      ;P3.6 CMP+ The input pin
ORL     CMPCR1,#04H          is P3.7 CMP- Input pin
ORL     CMPCR1,#02H          ;Enable comparator output
SETB    EA

JMP     $

END
```

11.5.18 SPI interrupt

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 P10 = P1^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;          //Clear interrupt sign
    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;
    SPCTL = 0x50;                   //Host mode ,Enable SPI
    SPSTAT = 0xc0;                  //Clear interrupt sign
    IE2 = ESPI;                     //Enable SPI interrupt
    EA = 1;
    SPDAT = 0x5a;                   //Send test data
    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

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
        CPL     P1.0            ;Test port
        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,#50H          ;Host mode ,Enable SPI
MOV     SPSTAT,#0C0H        ;Clear interrupt sign
MOV     IE2,#ESPI           ;Enable SPI interrupt
SETB    EA
MOV     SPDAT,#5AH          ;Send test data

JMP     $

END
```

11.5.19 I2C interrupt

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 P10                = P1^0;


void I2C_Isr() interrupt 24
{
```

```
        _push_(P_SW2);
        P_SW2 |= 0x80;
        if (I2CMSST & 0x40)
        {
        I2CMSST &= ~0x40;                       //Clear interrupt sign
        P10 = ! P10;                            //Test port
        }
        _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 = 0xc0;                          //Enable I2C Host mode
        I2CMSCR = 0x80;                         //Enable I2C interrupt
        P_SW2 = 0x00;
        EA = 1;
        P_SW2 = 0x80;
        I2CMSCR = 0x81;                         //Send start 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

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
        PUSH    P_SW2
        MOV     P_SW2,#80H
        MOV     DPTR,#I2CMSST
        MOVX    A,@DPTR
        ANL     A,#NOT 40H          ;Clear interrupt sign
        MOVX    @DPTR,A             ;Test port
        CPL     P1.0
        POP     P_SW2
        POP     DPH
        POP     DPL
        POP     ACC
        RETI

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

        MOV     P_SW2,#80H
        MOV     A,#0C0H             ;Enable I2C Host mode
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#80H              ;Enable I2C interrupt
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H
        SETB    EA

        MOV     P_SW2,#80H
        MOV     A,#081H             ;Send start command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
```

```
        MOV     P_SW2,#00H

        JMP     $

        END
```

12 I/O Port interrupt

STC12H Interrupt, and supports several interrupt modes: falling edge interrupt, rising edge interrupt, low-leve
High-level interrupt. Each group I/O ports have independent interrupt entry addresses, and each can set the interrupt mode independently. I/O

After testing, it was found that STC12H Series of I/O There is a problem with the mouth interruption, please do not use it for the time being

12.1 I/O Port interrupt related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	P0 Port interrupt enable register	FD00H	P07INTE	P06INTE	P05INTE P04INTE P03INTE P02INTE				P01INTE	P00INTE 0000,0000	
P1INTE	P1 Port interrupt enable register	FD01H	P17INTE	P16INTE	P15INTE P14INTE P13INTE P12INTE				P11INTE	P10INTE 0000,0000	
P2INTE	P2 Port interrupt enable register	FD02H	P27INTE	P26INTE	P25INTE P24INTE P23INTE P22INTE				P21INTE	P20INTE 0000,0000	
P3INTE	P3 Port interrupt enable register	FD03H	P37INTE	P36INTE	P35INTE	P34INTE P33INTE P32INTE			P31INTE	P30INTE 0000,0000	
P5INTE	P5 Port interrupt enable register	FD05H	P57INTE	P56INTE	P55INTE	P54INTE P53INTE P52INTE			P51INTE	P50INTE 0000,0000	
P0INTF	P0 Port interrupt flag register	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF 0000,0000	
P1INTF	P1 Port interrupt flag register	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF 0000,0000	
P2INTF	P2 Port interrupt flag register	FD12H	P27INTF	P26INTF	P25INTF P24INTF P23INTF P22INTF				P21INTF	P20INTF 0000,0000	
P3INTF	P3 Port interrupt flag register	FD13H	P37INTF	P36INTF	P35INTF P34INTF P33INTF P32INTF				P31INTF	P30INTF 0000,0000	
P5INTF	P5 Port interrupt flag register	FD15H	P57INTF	P56INTF	P55INTF P54INTF P53INTF P52INTF				P51INTF	P50INTF 0000,0000	
P0IM0	P0 Port interrupt mode register	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 Port interrupt mode register	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 Port interrupt mode register	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 Port interrupt mode register 0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P5IM0	P5 Port interrupt mode register	FD25H	P57IM0	P56IM0	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	0000,0000
P0IM1	P0 Port interrupt mode register	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1 Port interrupt mode register	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 Port interrupt mode register	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 Port interrupt mode register	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P5IM1	P5 Port interrupt mode register	FD35H	P57IM1	P56IM1	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	0000,0000

Port interrupt enable register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P5INTE	FD05H	P57INTE	P56INTE	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE

: Port interrupt enables the control bit (PnINTE. x n=0~7 , x=0~7)

0 : Closed Port interrupt function Pn.

1 : Enable Pn. Port interrupt function

Port interrupt flag register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P5INTF	FD15H	P57INTF	P56INTF	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF

: Port interrupt request flag (PnINTF. x n=0~7 , x=0~7)

Pn. x 0 : The port did not interrupt the request

Pn.x There is an interrupt request at the port. If the interrupt is enabled, the interrupt service program will be entered. The logo needs to be cl

0

1 :

Port interrupt mode configuration register

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0IM0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0
P0IM1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1
P1IM0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0
P1IM1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1
P2IM0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0
P2IM1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1
P3IM0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0
P3IM1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1
P5IM0	FD25H	P57IM0	P56IM0	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0
P5IM1	FD35H	P57IM1	P56IM1	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1

Configure the mode of the port

PnIM1. x	PnIM0. x	Port interrupt mode Pn. x
0	0	Falling edge interrupt
0	1	rising edge interrupt
1	0	low-level interrupt
1	1	high-level interrupt

12.2 Sample program

12.2.1 P0 The falling edge of the mouth is interrupted

C Language code

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

#include "reg51. h"
#include "intrins. h"
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 P4M0 = 0xb4;
sfr P4M1 = 0xb3;
sfr P5M0 = 0xca;
sfr P5M1 = 0xc9;
sfr P6M0 = 0xcc;
sfr P6M1 = 0xcb;
sfr P7M0 = 0xe2;
sfr P7M1 = 0xe1;
sfr P_SW2 = 0xba;
#define P0INTE (*(unsigned char volatile xdata *)0xfd00)
#define P0INTF (*(unsigned char volatile xdata *)0xfd10)
#define P0IM0 (*(unsigned char volatile xdata *)0xfd20)
#define P0IM1 (*(unsigned char volatile xdata *)0xfd30)
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;
    P0IM0 = 0x00;                   //Falling edge interrupt
    P0IM1 = 0x00;
    P0INTE = 0xff;                  //Enable P0 Port interrupt
    P_SW2 &= ~0x80;
    EA = 1;
```

```
        while (1);
}

//Since the interrupt vector is greater than 31, it cannot be compiled directly in //Must borrow the first interrupt entry address
void common_isr() interrupt 13
{
        unsigned char psw2_st;
        unsigned char intf;
        psw2_st = P_SW2;
        P_SW2 |= 0x80;
        intf = P0INTF;
        if (intf)
        {

        P0INTF = 0x00;
        if (intf & 0x01)
        {                                                   //P0.0 Port interrupt
        }
        if (intf & 0x02)
        {                                                   //P0.1 Port interrupt
        }
        if (intf & 0x04)
        {                                                   //P0.2 Port interrupt
        }
        if (intf & 0x08)
        {                                                   //P0.3 Port interrupt
        }
        if (intf & 0x10)
        {                                                   //P0.4 Port interrupt
        }
        if (intf & 0x20)
        {                                                   //P0.5 Port interrupt
        }
        if (intf & 0x40)
        {                                                   //P0.6 Port interrupt
        }
        if (intf & 0x80)
        {
                }                                           //P0.7 Port interrupt
        }
        P_SW2 = psw2_st;
}

// ISR. ASM
// Save the following code as ISR.ASM , Then add the file to the project

                CSEG            AT 012BH                    ;P0 Port interrupt entry address
                JMP             P0INT_ISR
P0INT_ISR:
```

```
                JMP     006BH                   ;borrow 13 The entry address of the number interruption
                END
```

Assembly code

The test operating frequency is 11.0592MHz

```
P0M0            DATA    094H
P0M1            DATA    093H
P1M0            DATA    092H
P1M1            DATA    091H
P2M0            DATA    096H
P2M1            DATA    095H
P3M0            DATA    0B2H
P3M1            DATA    0B1H
P4M0            DATA    0B4H
P4M1            DATA    0B3H
P5M0            DATA    0CAH
P5M1            DATA    0C9H
P6M0            DATA    0CCH
P6M1            DATA    0CBH
P7M0            DATA    0E2H
P7M1            DATA    0E1H

P_SW2           DATA    0BAH

P0INTE          XDATA   0FD00H
P0INTF          XDATA   0FD10H
P0IM0           XDATA   0FD20H
P0IM1           XDATA   0FD30H

                ORG     0000H
                LJMP    MAIN

                ORG     012BH                   ;P0 Port interrupt entry address
P0INT_ISR:
                PUSH    ACC
                PUSH    B
                PUSH    DPL
                PUSH    DPH
                PUSH    P_SW2

                MOV     DPTR,#P0INTF
                MOVX    A,@DPTR
                MOV     B,A
                CLR     A
                MOVX    @DPTR,A
                MOV     A,B
CHECKP00:
                JNB     ACC. 0,CHECKP01
                NOP                             ;P0.0 Port interrupt
CHECKP01:
                JNB     ACC. 1,CHECKP02
                NOP                             ;P0.1 Port interrupt
CHECKP02:
                JNB     ACC. 2,CHECKP03
                NOP                             ;P0.2 Port interrupt
```

```
CHECKP03
        JNB     ACC. 3,CHECKP04
        NOP                             ;P0.3   Port interrupt
CHECKP04:
        JNB     ACC. 4,CHECKP05
        NOP                             ;P0.4   Port interrupt
CHECKP05:
        JNB     ACC.5,CHECKP06
        NOP                             ;P0.5   Port interrupt
CHECKP06:
        JNB     ACC. 6,CHECKP07
        NOP                             ;P0.6   Port interrupt
CHECKP07:
        JNB     ACC. 7,P0ISREXIT
        NOP                             ;P0.7   Port interrupt

P0ISREXIT:
        POP     P_SW2
        POP     DPH
        POP     DPL
        POP     B
        POP     ACC
        RETI

        ORG     0200H
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

        ORL     P_SW2,#80H
        CLR     A
        MOV     DPTR,# P0IM0            ;Falling edge interrupt
        MOVX    @DPTR,A
        MOV     DPTR,# P0IM1
        MOVX    @DPTR,A
        MOV     DPTR,# P0INTE
        MOV     A,#0FFH
        MOVX    @DPTR,A                 ;Enable P0 Port interrupt
        ANL     P_SW2,#7FH

        SETB    EA

        JMP     $

        END
```

12.2.2 P1 The rising edge of the mouth is interrupted

C Language code

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

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

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 P4M0 = 0xb4;
sfr P4M1 = 0xb3;
sfr P5M0 = 0xca;
sfr P5M1 = 0xc9;
sfr P6M0 = 0xcc;
sfr P6M1 = 0xcb;
sfr P7M0 = 0xe2;
sfr P7M1 = 0xe1;

sfr P_SW2 = 0xba;

#define P1INTE (*(unsigned char volatile xdata *)0xfd01)
#define P1INTF (*(unsigned char volatile xdata *)0xfd11)
#define P1IM0 (*(unsigned char volatile xdata *)0xfd21)
#define P1IM1 (*(unsigned char volatile xdata *)0xfd31)

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;
    P1IM0 = 0xff;                   //Rising edge interrupt
    P1IM1 = 0x00;
    P1INTE = 0xff;                  //Enable P1 Port interrupt
    P_SW2 &= ~0x80;
    EA = 1;
    while (1);

}

//Since the interrupt vector is greater than 127, Cannot be compiled directly in //Must borrow the first 13 Interrupt entry address
void common_isr() interrupt 13
```

```
{
    unsigned char psw2_st;
    unsigned char intf;
    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P1INTF;
    if (intf)
    {

    P1INTF = 0x00;
    if (intf & 0x01)
    {                                                    //P1.0  Port interrupt
    }
    if (intf & 0x02)
    {                                                    //P1.1  Port interrupt
    }
    if (intf & 0x04)
    {                                                    //P1.2  Port interrupt
    }
    if (intf & 0x08)
    {                                                    //P1.3  Port interrupt
    }
    if (intf & 0x10)
    {                                                    //P1.4  Port interrupt
    }
    if (intf & 0x20)
    {                                                    //P1.5  Port interrupt
    }
    if (intf & 0x40)
    {                                                    //P1.6  Port interrupt
    }
    if (intf & 0x80)
    {
        }                                                //P1.7  Port interrupt
    }
    P_SW2 = psw2_st;
}

// ISR. ASM
// Save the following code as ISR.ASM , Then add the file to the project

        CSEG    AT 0133H                                 ;P1  Port interrupt entry address
        JMP     P1INT_ISR
P1INT_ISR:
        JMP     006BH                                    ;borrow 13  The entry address of the number interruption
        END
```

Assembly code

; The test operating frequency is

```
P0M0    DATA    094H
P0M1    DATA    093H
P1M0    DATA    092H
P1M1    DATA    091H
P2M0    DATA    096H
P2M1    DATA    095H
P3M0    DATA    0B2H
P3M1    DATA    0B1H
P4M0    DATA    0B4H
P4M1    DATA    0B3H
P5M0    DATA    0CAH
P5M1    DATA    0C9H
P6M0    DATA    0CCH
P6M1    DATA    0CBH
P7M0    DATA    0E2H
P7M1    DATA    0E1H

P_SW2   DATA    0BAH

P1INTE  XDATA   0FD01H
P1INTF  XDATA   0FD11H
P1IM0   XDATA   0FD21H
P1IM1   XDATA   0FD31H

        ORG     0000H
        LJMP    MAIN

        ORG     0133H                   ;P1 Port interrupt entry address
P1INT_ISR:
        PUSH    ACC
        PUSH    B
        PUSH    DPL
        PUSH    DPH
        PUSH    P_SW2

        MOV     DPTR,#P1INTF
        MOVX    A,@DPTR
        MOV     B,A
        CLR     A
        MOVX    @DPTR,A
        MOV     A,B
CHECKP10:
        JNB     ACC. 0,CHECKP11
        NOP                             ;P1.0 Port interrupt
CHECKP11:
        JNB     ACC. 1,CHECKP12
        NOP                             ;P1.1 Port interrupt
CHECKP12:
        JNB     ACC. 2,CHECKP13
        NOP                             ;P1.2 Port interrupt
CHECKP13
        JNB     ACC. 3,CHECKP14
        NOP                             ;P1.3 Port interrupt
CHECKP14:
        JNB     ACC. 4,CHECKP15
        NOP                             ;P1.4 Port interrupt
CHECKP15:
```

```
                JNB     ACC.5,CHECKP16
                NOP                             ;P1.5 Port interrupt
CHECKP16:
                JNB     ACC.6,CHECKP17
                NOP                             ;P1.6 Port interrupt
CHECKP17:
                JNB     ACC.7,P1ISREXIT
                NOP                             ;P1.7 Port interrupt

P1ISREXIT:
                POP     P_SW2
                POP     DPH
                POP     DPL
                POP     B
                POP     ACC
                RETI

                ORG     0200H
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

                ORL     P_SW2,#80H
                CLR     A
                MOV     DPTR,#P1IM0             ;Falling edge interrupt
                MOVX    @DPTR,A
                MOV     DPTR,#P1IM1
                MOVX    @DPTR,A
                MOV     DPTR,#P1INTE
                MOV     A,#0FFH
                MOVX    @DPTR,A                 ;Enable P1 Port interrupt
                ANL     P_SW2,#7FH

                SETB    EA

                JMP     $

                END
```

12.2.3 P2 Port low-level interrupt

C Language code

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

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

```
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         P4M0        =   0xb4;
sfr         P4M1        =   0xb3;
sfr         P5M0        =   0xca;
sfr         P5M1        =   0xc9;
sfr         P6M0        =   0xcc;
sfr         P6M1        =   0xcb;
sfr         P7M0        =   0xe2;
sfr         P7M1        =   0xe1;

sfr         P_SW2       =   0xba;

#define     P2INTE      (*(unsigned char volatile xdata *)0xfd02)
#define     P2INTF      (*(unsigned char volatile xdata *)0xfd12)
#define     P2IM0       (*(unsigned char volatile xdata *)0xfd22)
#define     P2IM1       (*(unsigned char volatile xdata *)0xfd32)

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;
    P2IM0 = 0x00;
    P2IM1 = 0xff;
    P2INTE = 0xff;
    P_SW2 &= ~0x80;
    EA = 1;
    while (1);

}

//Since the interrupt vector is greater than 128, it cannot be compiled directly in Keil. Must borrow the first 13 Interrupt entry address
void common_isr() interrupt 13
{
    unsigned char psw2_st;
    unsigned char intf;
    psw2_st = P_SW2;
    P_SW2 |= 0x80;
```

//Low-level interrupt

//Enable P2 Port interrupt

```
        intf = P2INTF;
        if (intf)
        {
        P2INTF = 0x00;
        if (intf & 0x01)
        {                                                   //P2.0  Port interrupt
        }
        if (intf & 0x02)
        {                                                   //P2.1  Port interrupt
        }
        if (intf & 0x04)
        {                                                   //P2.2  Port interrupt
        }
        if (intf & 0x08)
        {                                                   //P0.3  Port interrupt
        }
        if (intf & 0x10)
        {                                                   //P2.4  Port interrupt
        }
        if (intf & 0x20)
        {                                                   //P2.5  Port interrupt
        }
        if (intf & 0x40)
        {                                                   //P2.6  Port interrupt
        }
        if (intf & 0x80)
        {
                }                                           //P2.7  Port interrupt
        }
        P_SW2 = psw2_st;
}

// ISR. ASM
// Save the following code as ISR.ASM , Then add the file to the project

        CSEG    AT 013BH                                    ;P2  Port interrupt entry address
        JMP     P2INT_ISR
P2INT_ISR:
        JMP     006BH                                       ;borrow 13  The entry address of the number interruption
        END
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

```
P2M0            DATA    096H
P2M1            DATA    095H
P3M0            DATA    0B2H
P3M1            DATA    0B1H
P4M0            DATA    0B4H
P4M1            DATA    0B3H
P5M0            DATA    0CAH
P5M1            DATA    0C9H
P6M0            DATA    0CCH
P6M1            DATA    0CBH
P7M0            DATA    0E2H
P7M1            DATA    0E1H

P_SW2           DATA    0BAH

P2INTE          XDATA   0FD02H
P2INTF          XDATA   0FD12H
P2IM0           XDATA   0FD22H
P2IM1           XDATA   0FD32H

                ORG     0000H
                LJMP    MAIN

                ORG     013BH                   ;P2  Port interrupt entry address
P2INT_ISR:
                PUSH    ACC
                PUSH    B
                PUSH    DPL
                PUSH    DPH
                PUSH    P_SW2

                MOV     DPTR,#P2INTF
                MOVX    A,@DPTR
                MOV     B,A
                CLR     A
                MOVX    @DPTR,A
                MOV     A,B
CHECKP20:
                JNB     ACC. 0,CHECKP21
                NOP                             ;P2.0  Port interrupt
CHECKP21:
                JNB     ACC. 1,CHECKP22
                NOP                             ;P2.1  Port interrupt
CHECKP22:
                JNB     ACC. 2,CHECKP23
                NOP                             ;P2.2  Port interrupt
CHECKP23
                JNB     ACC. 3,CHECKP24
                NOP                             ;P2.3  Port interrupt
CHECKP24:
                JNB     ACC. 4,CHECKP25
                NOP                             ;P2.4  Port interrupt
CHECKP25:
                JNB     ACC. 5,CHECKP26
                NOP                             ;P2.5  Port interrupt
CHECKP26:
                JNB     ACC. 6,CHECKP27
                NOP                             ;P2.6  Port interrupt
CHECKP27:
```

```
                JNB     ACC. 7,P2ISREXIT
                NOP                                     ;P2.7 Port interrupt

P2ISREXIT:
                POP     P_SW2
                POP     DPH
                POP     DPL
                POP     B
                POP     ACC
                RETI

                ORG     0200H
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

                ORL     P_SW2,#80H
                CLR     A
                MOV     DPTR,# P2IM0                    ;Low-level interrupt
                MOVX    @DPTR,A
                MOV     DPTR,# P2IM1
                MOVX    @DPTR,A
                MOV     DPTR,# P2INTE
                MOV     A,#0FFH
                MOVX    @DPTR,A                         ;Enable P2 Port interrupt
                ANL     P_SW2,#7FH

                SETB    EA

                JMP     $

                END
```

12.2.4 P3 Port high-level interrupt

C Language code

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

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

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         P4M0        =   0xb4;
sfr         P4M1        =   0xb3;
sfr         P5M0        =   0xca;
sfr         P5M1        =   0xc9;
sfr         P6M0        =   0xcc;
sfr         P6M1        =   0xcb;
sfr         P7M0        =   0xe2;
sfr         P7M1        =   0xe1;

sfr         P_SW2       =   0xba;

#define     P3INTE      (*(unsigned char volatile xdata *)0xfd03)
#define     P3INTF      (*(unsigned char volatile xdata *)0xfd13)
#define     P3IM0       (*(unsigned char volatile xdata *)0xfd23)
#define     P3IM1       (*(unsigned char volatile xdata *)0xfd33)

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;
    P3IM0 = 0xff;
    P3IM1 = 0xff;
    P3INTE = 0xff;
    P_SW2 &= ~0x80;
    EA = 1;
    while (1);
}

//Since the interrupt vector is greater than 31, it cannot be compiled directly in Keil. Must borrow the first 13 Interrupt entry address
void common_isr() interrupt 13
{
    unsigned char psw2_st;
    unsigned char intf;
    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P3INTF;
    if (intf)
    {
        P3INTF = 0x00;
        if (intf & 0x01)
        {
```

//High-level interrupt

//Enable P3 Port interrupt

```
                                                                    //P3.0    Port interrupt
        }
        if (intf & 0x02)
        {                                                           //P3.1    Port interrupt
        }
        if (intf & 0x04)
        {                                                           //P3.2    Port interrupt
        }
        if (intf & 0x08)
        {                                                           //P3.3    Port interrupt
        }
        if (intf & 0x10)
        {                                                           //P3.4    Port interrupt
        }
        if (intf & 0x20)
        {                                                           //P3.5    Port interrupt
        }
        if (intf & 0x40)
        {                                                           //P3.6    Port interrupt
        }
        if (intf & 0x80)
        {
                }                                                   //P3.7    Port interrupt
        }
        P_SW2 = psw2_st;
}

// ISR. ASM
// Save the following code as ISR.ASM , Then add the file to the project

        CSEG    AT 0143H                                    ;P3     Port interrupt entry address
        JMP     P3INT_ISR
P3INT_ISR:
        JMP     006BH                                       ;borrow 13 The entry address of the number interruption
        END
```

Assembly code

```
; The test operating frequency is 11.0592MHz

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

```
P5M0            DATA    0CAH
P5M1            DATA    0C9H
P6M0            DATA    0CCH
P6M1            DATA    0CBH
P7M0            DATA    0E2H
P7M1            DATA    0E1H

P_SW2           DATA    0BAH

P3INTE          XDATA   0FD03H
P3INTF          XDATA   0FD13H
P3IM0           XDATA   0FD23H
P3IM1           XDATA   0FD33H

                ORG     0000H
                LJMP    MAIN

                ORG     0143H           ;P3 Port interrupt entry address
P3INT_ISR:
                PUSH    ACC
                PUSH    B
                PUSH    DPL
                PUSH    DPH
                PUSH    P_SW2

                MOV     DPTR,#P3INTF
                MOVX    A,@DPTR
                MOV     B,A
                CLR     A
                MOVX    @DPTR,A
                MOV     A,B
CHECKP30:
                JNB     ACC.0,CHECKP31
                NOP                     ;P3.0 Port interrupt
CHECKP31:
                JNB     ACC.1,CHECKP32
                NOP                     ;P3.1 Port interrupt
CHECKP32:
                JNB     ACC.2,CHECKP33
                NOP                     ;P3.2 Port interrupt
CHECKP33
                JNB     ACC.3,CHECKP34
                NOP                     ;P3.3 Port interrupt
CHECKP34:
                JNB     ACC.4,CHECKP35
                NOP                     ;P3.4 Port interrupt
CHECKP35:
                JNB     ACC.5,CHECKP36
                NOP                     ;P3.5 Port interrupt
CHECKP36:
                JNB     ACC.6,CHECKP37
                NOP                     ;P3.6 Port interrupt
CHECKP37:
                JNB     ACC.7,P3ISREXIT
                NOP                     ;P3.7 Port interrupt

P3ISREXIT:
                POP     P_SW2
                POP     DPH
```

```
        POP     DPL
        POP     B
        POP     ACC
        RETI

        ORG     0200H
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

        ORL     P_SW2,#80H
        CLR     A
        MOV     DPTR,# P3IM0        ;High-level interrupt
        MOVX    @DPTR,A
        MOV     DPTR,# P3IM1
        MOVX    @DPTR,A
        MOV     DPTR,# P3INTE
        MOV     A,#0FFH
        MOVX    @DPTR,A             ;Enable P3 Port interrupt
        ANL     P_SW2,#7FH

        SETB    EA

        JMP     $

        END
```

13 / Timer counter

STC12H The series of microcontrollers are set up with Bit timer/counter. 5 a Bit timer and T0、T4、T2、T3 16

Both have two working methods, counting mode and timing mode. Pair timer and T1 , Use them in the special function register

The corresponding control bit in the counter is selected or a counter? Pair timer Is the T2 , Use special function registers

AUXR In the control bit in Come, choose counter a timer or a counter? Pair timer Is the T3 , use special function registers

T4T3M the controlControl , choose, T3 counter a timer or a counter? Pair timer counter T4 , use special function registers

T4T3M bit in the control bit choose T4 Is it a timer or a counter? Timer The core component of the counter is an addition count

The essence of the device is to count the pulses. It's just that the source of the counting pulse is different: if the counting pulse comes from t

method. At this time, the timer/counter gets a counting pulse for each clock, and the counting value is added; If the counting pulse comes fro

12 1 1 The external pins of the machine are counted, and each pulse is added. 1

T0、T1 When the timer counters are determined When operating in timing mode, special function registers and in T1x12 and

T2x12 to be the system clock separately Or the system clock /1 (Non-frequency division) post-concatenation When the timer count

device T3 When operating in timing mode, special function registers and T4 and T4x12 Decide whether it is the system clock separat

Or the system clock /1 (Non-frequency division) post-concatenation When the timer When the counter is operating in counting mode, the external pulse n

The number is not divided.

Timer The counter has 0 Working modes: mode (0 16 Bit automatic reloading mode), mode (1 Bit non-reloadable mode) ,

Mode (82 Bit automatic reloading mode), mode (outside the mode that can be Bit automatic reload mode). Timer Counter de multiplexing in

shielded from interruption) 3, Other operating modes and timer The counter When is the same, stop counting. Timer operating mode 3 T2

Fixed to position, automatic reloading mode It can be used as a timer, or as a baud rate generator and programmable clock output for the serial po

3 Timer, timer 4 With timer T2 In the same way, their working mode is fixed to bit automatic reloading mode。

It can also be used as a baud rate generator and a programmable clock output for the serial port.

13.1 Timer related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	Timer control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	Timer mode register	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	Timer 0 low 8 Bit register	8AH									0000,0000
TL1	Timer 1 low 8 Bit register	8BH									0000,0000
TH0	Timer 0 high 8 Bit register	8CH									0000,0000
TH1	Timer high 1 Bit register	8DH									0000,0000
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	Interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	Power-down wake-up timer low byte	AAH									1111,1111
WKTCH	Power-down wake-up timer high byte	ABH	WKTEN								0111,1111
T4T3M	Timer 4/3 Control register	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO 0000,0000	
T4H	Timer 4 High byte	D2H									0000,0000
T4L	Timer low byte	D3H									0000,0000
T3H	3 Timer high byte	D4H									0000,0000
T3L	3 Timer Low byte	D5H									0000,0000
T2H	timer 2 High byte	D6H									0000,0000
T2L	timer 2 Low byte	D7H									0000,0000

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
TM2PS	Timer clock prescaler register	FEA2H									0000,0000
TM3PS	Timer clock prescaler register	FEA3H									0000,0000
TM4PS	Timer clock prescaler register	FEA4H									0000,0000

13.2 Timer 0/1

13.2.1 Timer 0/1 Control register (TCON)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1: Timer 1 overflow interrupt flag. After being allowed to count, add from the initial value of 1 count. When an overflow occurs, TF1 is set to "1". Location "", parallel to CPU occurs, the hardware interrupts the request and keeps it until When responding to an interrupt, it is cleared by the hardware." 0 "(It can also be cle

TR1: Timer 1 The operation control bit. This bit is set and cleared by the software. When (GATE) TMOD. 7 ', When it is allowed to open T1 Start counting, TR1=1=0 and Only when the input is high, it is allowed T1 count. When prohibited T1 count. when =1, INT1GATE (TMOD. 7)

TF0: T0 Overflow interrupt flag. to CPU1 After being allowed to count, add from the initial value of 1 Count, when an overflow occurs, it is set by the hard The request is interrupted and when the interrupt is responded to, it is cleared by the hardware 0 (It can also be cleared by the

TR0: Timer T0). The operation control bit. This bit is set and cleared by the software. when GATE (TMOD.3) =1, TR0=1 and INT0 When it is allowed T0 on and the power is high, it is allowed T0 Start counting, T0 count. when count, TR0=0 When prohibited TR0=0 When prohibited GATE (T0 count.

IE1: External interrupt 1 Request source (INT1/P3.3) logo. IE1=1, External interrupt to CPU The request is interrupted and when CPU Hardware response to the interrupt, by 0 Hardware clearance" IE1.

IT1: External interrupt source 1 Trigger the control bit. Either the rising or falling edge can trigger an external, if set to IT1=0 External interrupt 1 Program control is Falling edge trigger method. IT1=1

IE0: External interrupt 0 Request source (INT0/P3.2) logo. IE0=1 External interrupt 0 to CPU The request is interrupted and when responding to an external interru 0 Cleared by the hardware IE0 (Edge trigger method).

IT0: External interrupt source 0 Trigger the control bit. Either the rising or falling edge can trigger an external, if set to IT0=0 External interrupt 0 Program control is Falling edge trigger method. IT0=1

13.2.2 Timer 0/1 Mode register (TMOD)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T T1_M1		T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

: Control the timer$_1$, set$_1$ Only when INT1 Feet are high and TR1 Control position$_1$ Only then can the timer be turned on/counter$_1$.

: Control the timer$_0$, set$_1$ Only when INT0 Feet are high and TR0 Control position$_1$ Only then can the timer be turned on/counter$_0$.

T1_C/T : Control the timer$_1$ Used as a timer or counter, clear$_0$ It is used as a timer (counting the internal system clock), set$_1$ Used as a counter (pair pin T1/P3.5 The external pulse is counted).

T0_C/T : Control the timer$_0$ Used as a timer or counter, clear$_0$ It is used as a timer (counting the internal system clock), set$_1$ Used as a counter (pair pin T0/P3.4 The external pulse is counted).

T1_M1/T1_M0 :Timer timer/counter$_1$

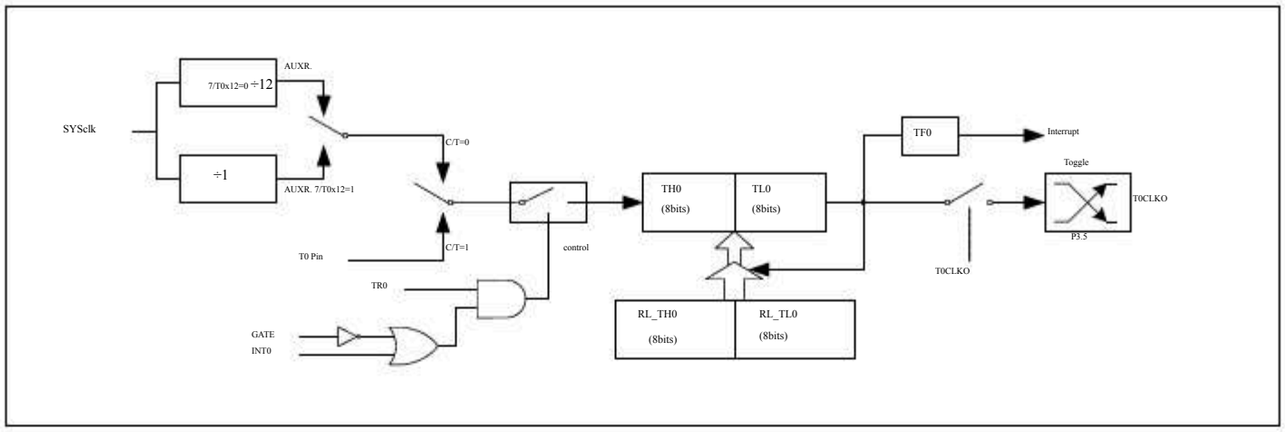
T1_M1	Mode selection T1_M0	Timer/counter$_1$ Working mode
0	0	16 Bit automatic overloading mode when the overloading value in [TH1,TL1] in 16 When the bit count value overflows, the system will automatically send the internal 16 bit the overloading register is loaded [TH1,TL1] in.
0	1	16 Bits are not automatically overloaded mode when [TH1,TL1] in 16 When the bit count value overflows, the timer$_1$ Will be from$_0$ Start counting
1	0	8 Bit automatic overload mode when in 8 When the bit count value TL1 TH1 Overloaded value in overflows, the system will automatically TL1 in. Load
	1 1	T1 Stop working

T0_M1/T0_M0 : timer timer/counter$_0$ Mode selection

T0_M1	T0_M0	Timer/counter$_0$ Working mode
0	0	16 Bit automatic overload mode when [TH0,TL0] The overloaded value in 16 When the bit count value overflows, the system will automatically send the internal 16 bit in the overloaded register is loaded [TH0,TL0] in.
0	1	16 Bits are not automatically overloaded mode when [TH0,TL0] in 16 When the bit count value overflows, the timer$_0$ Will be from$_0$ Start counting
1	0	8 Bit automatic overload mode when TL0 in 8 When the bit count value overflows, the system TH0 Overloaded value in will automatically TL0 in. Load
1	1	Unshielded interruption 16 Bit automatic overload mode With mode$_0$ The same, non-maskable interrupts, the interrupt has the highest priority, higher than the priority of all other interrupts, and cannot be turned off. It can be used as a system beat timer for the operating system, or a system monitoring timer.

13.2.3 0 Timer mode (16 0 Bit automatic reloading mode)

In this mode, the timer/counter 0 is used as a 16-bit counter that can be automatically reloaded, as shown in the figure below. :



/ Timer counter 0 0 The pattern of : Bit automatic reload mode

when GATE=0 (TMOD. 3) When, such as When external input is allowed INT0 Control timer ,
This enables pulse width measurement. TR0 Control bits in the register , TCON See the previous section for a description of the specific functions of the
Introduction to memory. then for TCON

when C/T =0 clock When the multi-channel switch is connected to the system, Work in timing mode. when
=1C/T When the multi-channel switch is connected to an external pulse input
P3.4/T0, that is T0 Work in counting mode.
There are two types of timer for single chip microcomputer Mode, each is A clock plus, with tradition 12 8051 Same as the single chip microcom
The other is the mode, each clock is added, and the speed is traditional Single chip microcomputer The rate is determined by the special function register
in T0x12 Decide if T0x12=0 , T0 Then work in 12T Mode; if T0x12=1 , T0 Then work in T pattern

Timer 0 There are two hidden ones register and RL_TH0 RL_TH0 with Share the same address , RL_TL0 with TH0 TL0
Share the same address. when That is, the timer counter When it is forbidden to work, yes The written content will be written at the same time RL_TL0, When
TH0 The written content will also be written at the same time That is, the timer counter 0 the pair is allowed to work, the pair writes the content, actually TL
In fact, it is not written to the current register is written to a hidden register In, right TH0 Writing content is not actually writing
Current register TH0 , but is written to a hidden register , This can be cleverly achieved Bit reload timer. When reading TH0
When the content of the sum is, the content read is TH0 and TL0 The content instead of RL_TH0 and RL_TL0 The content.

When the timer is working at pattern TMOD[1:0]/[M1,M0]=00B) time ,
will [RL_TH0,RL_TL0] Reload the content [TH0,TL0] The overflow is not only set TF0, and it will automatically

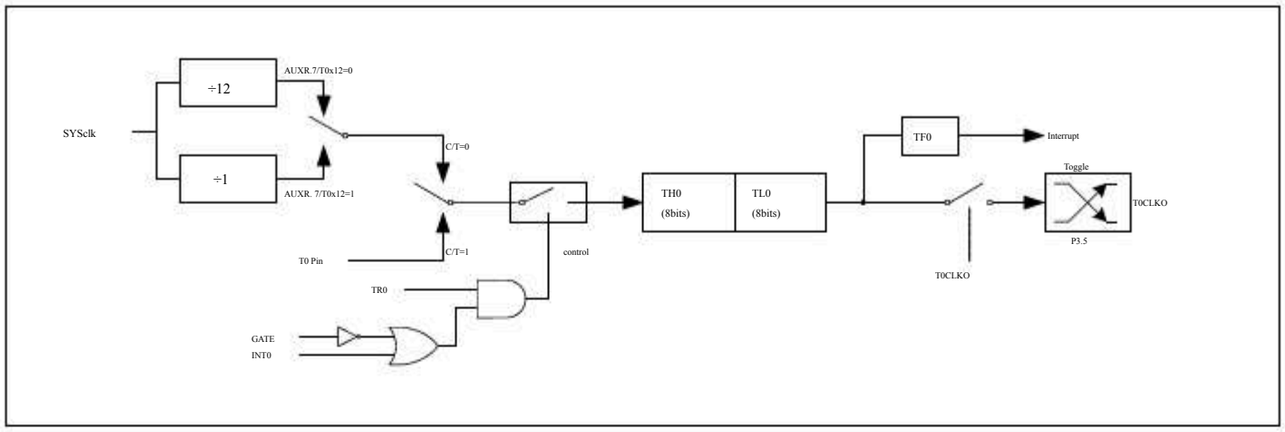
when T0CLKO/INT_CLKO. 0=1 when , P3.5/T1 The pin is configured as a Clock output T0CLKO The output clock frequency is
overflow rate.

if C/T=0, Timer counter T0 Count the internal system clock, then :
T0 Working at Mode (= (SYSclk)/(65536-[RL_TH0, RL_TL0])/2
T0 work at 12T Mode ( ) The output clock frequency at the time AUXR. 7/T0x12=1
= (SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2
) The output clock frequency at the time AUXR. 7/T0x12=0

if / Is for external pulse input C/T=1 , then : Timer counter T0
= (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2
Clock frequency output

13.2.4 0 Timer mode (16 1 Bit non-reloadable mode)

In this mode, the timer/counter 0 operates in a 16-bit non-reloadable mode, as shown in the figure below.



/ Timer counter 0 1 The pattern of : Bit non-reloadable mode

In this mode, the timer/Counter Configured as 0 of 8 Bit sum TH0 of 8 Composed of bits. TL0 of 8 bit overflow to carry bit , TH0 TH0 Count overflow set TF0 8 Bit non-reloadable mode, by TL0 Overflow flag in TCON

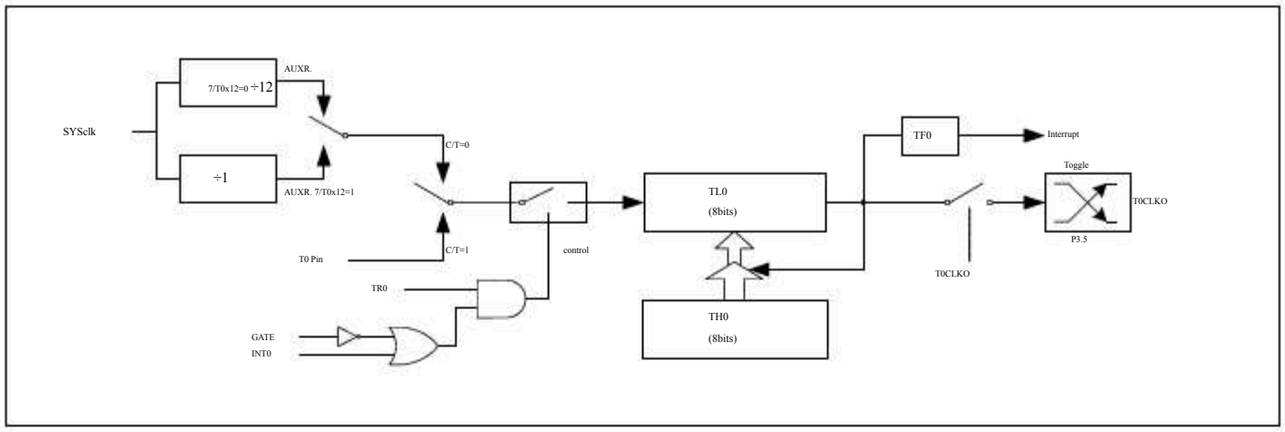
when GATE=0(TMOD. 3) When, such as TR0=1, the timer counts. GATE=1 When external input is allowed 0 Control timer , This enables pulse width measurement. TCON TR0 Control bits in the register , TCON See the previous section for a description of the specific functions of the Introduction to memory.

when C/T =0 When the multiplexer is connected to the divider output of the system clock, Count the internal system clock , Work in timing mode. when =1C/T When the multi-channel switch is connected to an external pulse input P3.4/T0 T0 Work in counting mode.

There are two types of timer for single-chip microcomputer STC Mode, each 12T A clock plus, with tradition 12 8051 Same as the single chip microcom The other is 1T 1 count 0 Single chip microcomputer 12 The rate is determined by the special function register in T0x12 Decide if T0x12=0 , T0 Then work in 12T Mode; if T0x12=1 , T0 Then work in T pattern Mode, each clock is added, the speed is traditional 8051

Timer mode (82 Bit automatic reloading mode) 13.2.5

In this mode, the timer/counter 0 is used as an 8-bit counter that can be automatically reloaded, as shown in the figure below. :



Timer/Counter mode 0 : 82 Bit automatic reload mode

TL0 The overflow is not only set TF0 will TH0 Reload the content TL0, TH0 The content is preset by the software, when reinstalling Content is not change.

when T0CLKO/INT_CLKO. 0=1 when, P3.5/T1 The pin is configured as a Clock output T0CLKO° The output clock frequency is overflow rate/2。

if C/T=0, Timer counter/ T0 Count the internal system clock, then :

T0 Working at 1T Mode (= (SYSclk)/(256-TH0)/2

T0 work at 12T Mode ( ) The output clock frequency at the time AUXR. 7/T0x12=1 = (SYSclk)/12/(256-TH0)/2

) The output clock frequency at the time AUXR.7/T0x12=0

if / Is for external pulse input C/T= Timer counter T0 T(P3.4/T0)

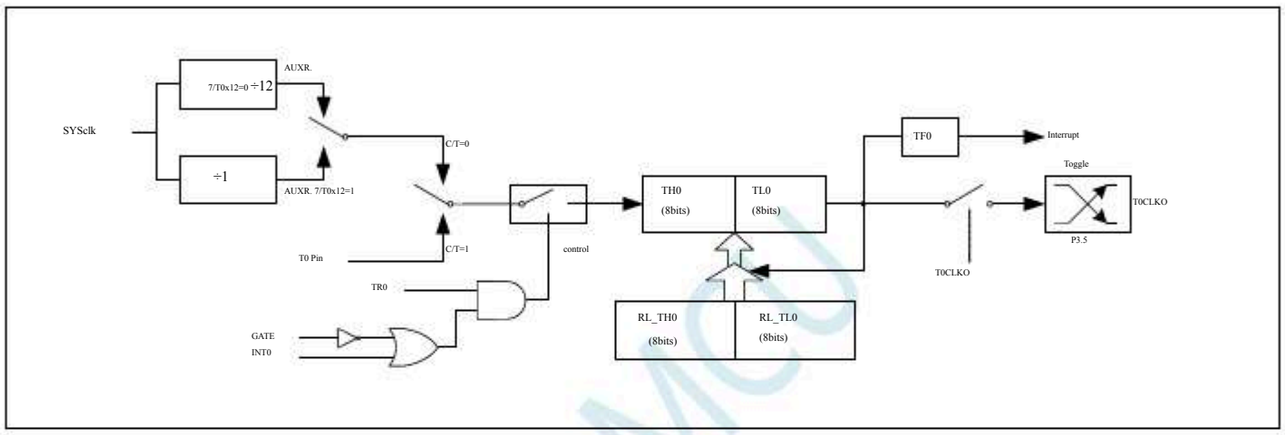
Clock frequency output = (T0_Pin_CLK) / (256-TH0)/2

Timer mode (non-maskable interrupt 13.2.6 System metronome)

Automatic bit reloading, real-time operation

Pair timer/Counter, its working mode mode 3 And working mode 0 It's the same (timer mode in the picture below) Schematic diagram, and the working mo It's the same). The only difference is: When the timer/When the counter is in mode 3, Just allow formula ET0/IE.1(Timer/Counter in progress Break allowable bit), No need to allow the total interrupt enable bit) You can turn on the timer/counter 0 EA/IE.7(Interrupt, the timer in this mode/counter

Interrupt and total interrupt enable bits It has nothing to do with the timer once it is operating in mode/Counter interrupt, then the interrupt is (ET0=1)4 is turned on 0 EA Unshielded, the priority of the interrupt is the highest, that is, the interrupt cannot be interrupted by any interrupt, and the interrupt is neither affected by it 7 The control is no longer controlled or this interrupt cannot be shielded from time to time. Therefore, this The mode is called the bit of non-maskable interr Control, when Dynamic reload mode.

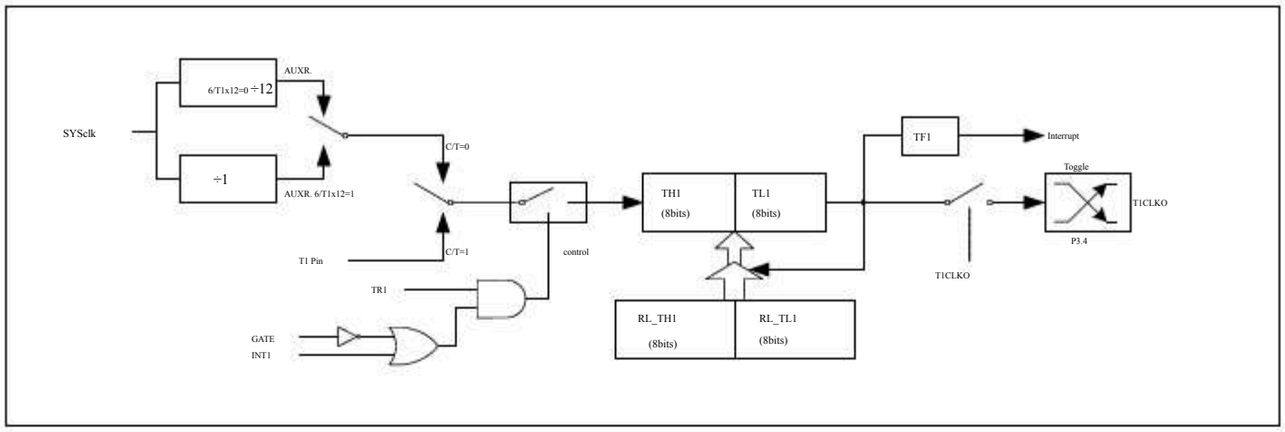


Timer/Counter mode 0: non-maskable interrupt 3 16 Bit automatic reload mode

attention : When the timer/counter Working in mode 3(Unshielded interruption Bit automatic reload mode/When you don't need to allow Total interrupt enable bit), EA/IE.7(ET0/IE.1(Timer counter / Only the interrupt and total Interrupt allow bit/You can turn on the timer/counter 0 Interrupt, the timer in this mode/Counter in progress 0 interrupt enable bits are allowed EA Invalid. Once the timer in this mode/counter 0 After the interrupt is turned on, the timer/double Interrupt priority is The highest, it cannot be interrupted by any other interruption/Whether it is better than the timer/Neither the counter interrupt interrupt with a low priority nor the higher priority than it can interrupt the timer at this time./Counter interrupt), And the interrupt is neither controlled nor controlled anymore after it is turned on, it i EA/IE. 7 ET0 EA Or this interrupt cannot be turned off. ET0

13.2.7 1 Timer mode (16 0 Bit automatic reloading mode)

In this mode, the timer/counter 1 is used as a 16-bit counter that can be automatically reloaded, as shown in the figure below. :



/ Timer counter 1 0 The pattern of : Bit automatic reload mode

when GATE=0 (TMOD. 7) When, such as When external input is allowed INT1 Control timer ,
This enables pulse width measurement. TR1 Control bits in the register , TCON See the previous section for a description of the specific functions of the
Introduction to memory. then for TCON

when C/T =0 clock When frequency division counting, it is connected to the system Work in timing mode. when
=1C/T When the multi-channel switch is connected to an external pulse input
P3.5/T1, that is T1 Work in counting mode.
There are two types of timer for single chip microcomputer Mode, each 12 Clock plus, with tradition 12 8051 Same as the single chip microcom
The other is the mode, each clock is added, and the speed is traditional Single chip microcomputer The rate is determined by the special function register
in T1x12 Decide if T1x12=0 , T1 Then work in 2T Mode; if T1x12=1 , T1 Then work in T pattern

Timer 1 There are two hidden ones register and RL_TL1 RL_TH1 with Share the same address , RL_TL1 with TH1 TL1
Share the same address. when TR1=0 / That is, the timer counter When it is forbidden to work, yes The written content will be written at the same time RL_TL1, When
TH1 The written content will also be written at the same time RL_TH1 That is, the timer counter 1 the pair is allowed to work, the pair writes the content, actually TL
In fact, it is not written to the current register It is written to a hidden register RL_TL1 In, right TH1 Writing content is not actually writing
Current register TH1 , but is written to a hidden register , This can be cleverly achieved RL_ Bit reload timer. When reading TH1
When the content of the sum is, the content read is TH1 and TL1 The content instead of RL_TH1 and RL_TL1 The content.

When the timer is working at pattern TMOD[5:4]/[M1,M0]=00B) time , [TH1,TL1] The overflow is not only set TF1, and it will automatically
will [RL_TH1,RL_TL1] Reload the content [TH1,TL1].

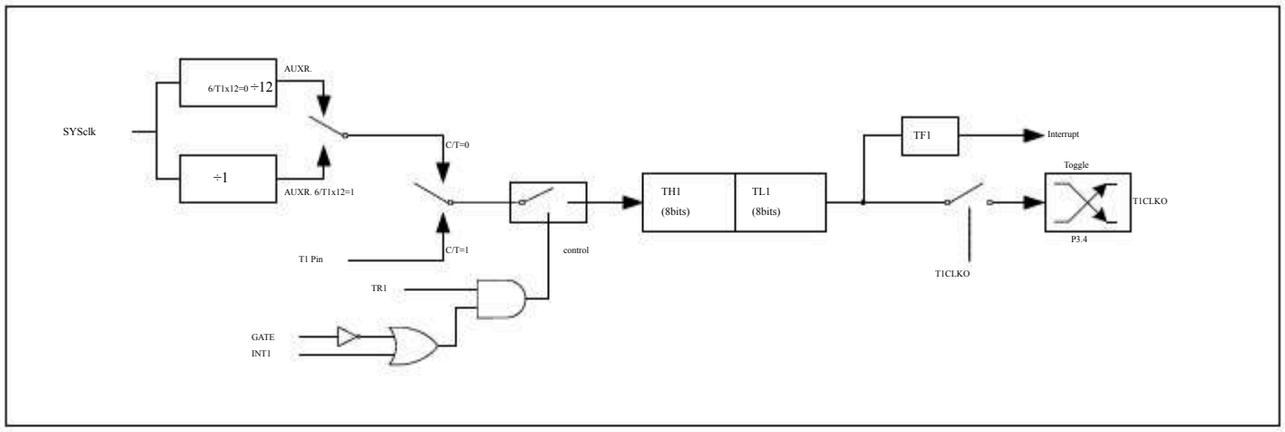
when T1CLKO/INT_CLKO. 1=1 when , P3.4/T0 The pin is configured as a Clock output T1CLKO. Output clock frequency The rate is
overflow/2 rate.

if C/T=0, Timer counter / T1 Count the internal system clock, then :
T1 Working at Mode (= (SYSclk)/(65536-[RL_TH1, RL_TL1])/2
) The output clock frequency at the time AUXR. 6/T1x12=1
T1 work at 12T Mode (= (SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2
) The output clock frequency at the time AUXR. 6/T1x12=0

if / Is for external pulse input C/T=1, Timer counter T1
(P3.5/T1)
= (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2
Clock frequency output

13.2.8 1 Timer mode (16 1 Bit non-reloadable mode)

In this mode, the timer/counter 1 operates in a 16-bit non-reloadable mode, as shown in the figure below.



/ Timer counter 1 1 The pattern of : Bit non-reloadable mode 16

In this mode, the timer/Counter Configured as 1 of 8 Bit sum TH1 of 8 Composed of bits. TL1 of 8 bit overflow to carry bit , TH1 TH1 Count overflow set TFL1 Bit non-reloadable mode, by TL1 TCON Overflow flag in

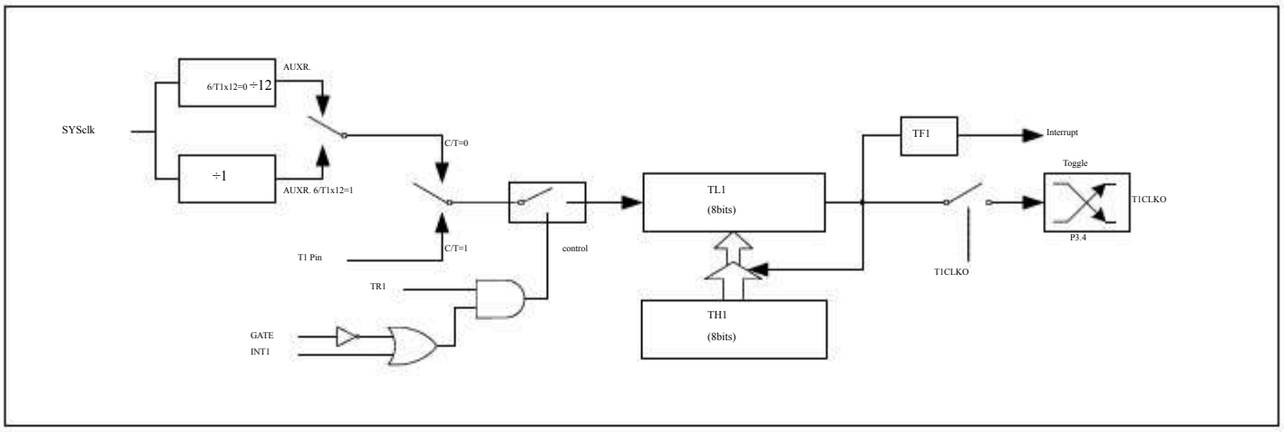
when GATE=0(TMOD. 7) When, such as TR1=1, the timer counts. GATE=1 When external input is allowed INT1 1 Control timer , This enables pulse width measurement. TCON TR1 Control bits in the register , TCON See the previous section for a description of the specific functions of the Introduction to memory.

when C/T =0 When the multiplexer is connected to the divider output of the system clock, Counter internal system clock , Work in timing mode. when =1C/T When the multi-channel switch is connected to an external pulse input T1 P3.5/T1 1 Work in counting mode.

There are two types of timer for single-chip microcomputer STC Mode, each 12T A clock plus, with tradition 12 8051 Same as the single chip microcom The other is 1T 1 count 1 Single chip microcomputer The rate is determined by the special function register AUXR in T1x12 Decide if T1x12=0 , T1 Then work in 12T Mode; if T1x12=1 , T1 Then work in T pattern Mode, each clock is added, the speed is traditional 8051

Timer mode (82 Bit automatic reloading mode) 13.2.9

In this mode, the timer/counter 1 is used as an 8-bit counter that can be automatically reloaded, as shown in the figure below. :



Timer/Counter mode 1 : 82 Bit automatic reload mode

TL1 The overflow is not only set TF1 and will TH1 Reload the content TL1 , TH1 The content is preset by the software, when reinstalling Content is not change.

when T1CLKO/INT_CLKO. 1=1 when , P3.4/T0 The pin is configured as a Clock output T1CLKO° The output clock frequency is overflow rate 2。

if C/T=0, Timer counter T1 Count the internal system clock, then :

T1 Working at 1T Mode (

= (SYSclk)/(256-TH1)/2

T1 work at 12T Mode (

) The output clock frequency at the time AUXR. 6/T1x12=1
= (SYSclk)/12/(256-TH1)/2

) The output clock frequency at the time AUXR. 6/T1x12=0

if / Is for external pulse input C/T=1 (P3.5/T1) Count, then : Timer counter T1

Clock frequency output = (T1_Pin_CLK) / (256-TH1)/2

13.2.10 Timer count register (TL0 , TH0)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

When the timer/counter 0 Work in 16 Bit mode (mode 0 , mode 1 , mode 3) time , TL0 and TH0 Combine into one 16 Bit register, TL0 Is the low byte , TH0 Is the high byte. If it is 8 Bit mode (mode 2) time , TL0 and TH0 for two independent 8 Bit register.

13.2.11 Timer count register (TL1 , TH1)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

When the timer/counter 1 Work in 16 Bit mode (mode 0 , mode 1) time , TL1 And is the 16 Bit register , TL1 Is low TH1 Combine byte , TH1 high byte. If it is 8 Bit mode (mode 2) When, and TH1TL1 into one for two independent 8 Bit register.

13.2.12 1 Auxiliary register (AUXR)

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

T0x12 : Timer 0 Speed control bit

0 : 12T Mode, that is CPU Frequency division (FOSC/12) Clock 12

1 : 1T Mode, that is CPU The clock is not divided by frequency (FOSC/1)

T1x12 : Timer 1 Speed control bit

0 : 12T Mode, that is, mode, Frequency division (FOSC/12) Clock CPU

1 : 1T that is CPU 12 The clock is not divided by frequency (FOSC/1)

13.2.13 Interrupt and clock output control register (INTCLKO)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

: Timer 0 Clock output control T0CLKO

0 : Turn off the clock output

1 : Enable The port is the timer 0 clock output function

When the timer When the count overflows , P3.5 The level of the port is automatically flipped.

T1CLKO : Timer 01 Clock output control

0 : Turn off the clock output

: Enable

The port is a timer 1 Clock output function

1 When the timer count overflows , P3.4 The level of the port is automatically flipped.

13.2.14 Timer calculation formula

Timer mode	Timer speed	Cycle calculation formula
pattern0/3 (16 Bits are automatically overloaded)	1T	Timer period = $\frac{65536 - [TH0, TL0]}{SYSclk}$ (Automatic reloading)
	12T	Timer period = $\frac{65536 [TH0, TL0]-}{SYSclk} \times 12$ (Automatic reloading)
pattern1 (16 Bits are not automatically overloaded)	1T	Timer period = $\frac{65536 [TH0, TL0]-}{SYSclk}$ (Software loading required)
	12T	Timer period = $\frac{65536 [TH0, TL0]-}{SYSclk} \times 12$ (Software loading required)
pattern2 (8 Bits are automatically overloaded)	1T	Timer period = $\frac{256 - TH0}{SYSclk}$ (Automatic reloading)
	12T	Timer period = $\frac{256 - TH0}{SYSclk} \times 12$ (Automatic reloading)

13.2.15 1 Timer calculation formula

Timer mode	Timer speed	Cycle calculation formula
pattern0 (16 Bits are automatically overloaded)	1T	Timer period = $\frac{65536 - [TH1, TL1]}{SYSclk}$ (Automatic reloading)
	12T	TimerCycle = $\frac{65536 [TH1, TL1]-}{SYSclk} \times 12$ (Automatic reloading)
pattern1 (16 Bits are not automatically overloaded)	1T	Timer period = $\frac{65536 [TH1, TL1]-}{SYSclk}$ (Software loading required)
	12T	Timer period = $\frac{65536 [TH1, TL1]-}{SYSclk} \times 12$ (Software loading required)
pattern2 (8 Bits are automatically overloaded)	1T	Timer period = $\frac{256 - TH1}{SYSclk}$ (Automatic reloading)
	12T	Timer period = $\frac{256 - TH1}{SYSclk} \times 12$ (Automatic reloading)

13.3 2 Timer (24 Bit timer , 8 Bit prescaler +16 Bit timing)

13.3.1 1 Auxiliary register (AUXR)

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

T2R : Timer$_2$ The operation control bit

0 : The timer stops counting

1 : The timer starts counting

T2_C/T : Control the timer$_2$ Used as a timer or counter, clear$_0$ It is used as a timer (counting the internal system clock), set$_1$ Used as a counter (pair pin$_{T2/P1.2}$ The external pulse is counted).

T2x12 : Timer$_2$ Speed control bit

0 : 12T Mode, that is, mode, Frequency division (FOSC/12) Clock CPU

1 : 1T that is CPU 12 The clock is not divided by frequency (FOSC/1)

13.3.2 Interrupt and clock output control register (INTCLKO)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

: Timer$_2$ Clock output control T2CLKO

0 : Turn off the clock output

1 : Enable The port is the timer clock output function P1.3 When the timer count overflows , P1.3 The level of the port is automatically flipped.

13.3.3 Timer count register (T2L , T2H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

Timer/counter$_2$ The working mode is fixed as $_{16}$Bit overload mode , T2L and T2H Combine into one $_{16}$ Bit register , T2L Is the low byte , Is the high byte. When [T2H,T2L] T2H in$_{16}$ When the bit count value overflows, the system will automatically send the internal$_{16}$ Overloading The value is loading. [T2H,T2L]

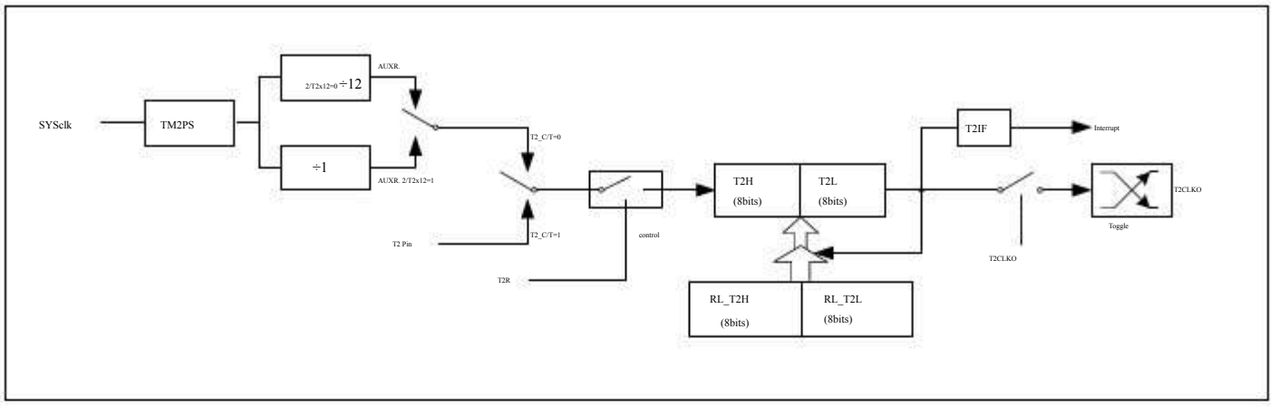
13.3.4 Timer's$_2$ Bit prescaler register (TM2PS)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TM2PS	FEA2H								

Timer$_2$ Clock = system clock$_{SYSclk}$ ÷ (TM2PS + 1)

13.3.5 Timer operating mode 2

The schematic diagram of timer/counter 2 is as follows：



Timer counter 2 Working mode：16 Bit automatic reload mode

T2R/AUXR. 4 for AUXR Control bits in the register，AUXR See the previous section for a description of the specific functions of the register

Shao.

when T2_C/T =0 When the multiplexer is connected to the system clock output，T2 Count the internal system clock， T2_C/T =1 When the multi-channel switch is connected to an external pin T2 Work in timing mode, when Work in counting mode.

There are two types of timer for single chip microcomputer Rate: One is STC Mode, each 12T A clock plus, with tradition 12 8051 Same as the single chip microcomputer The other is 1T Mode, each clock is added, the speed is traditional Single chip microcomputer Timer rate is determined by the special function register AUXR in T2x12 Decide if T2x12=0，T2 Then work in 12T Mode; if T2x12=1，T2 Then work in 1T pattern

Timer 2 and RL_T2L。 RL_T2H with Share the same address，RL_T2L with T2H T2L Share the same address. when T2R=0 register There are two hidden ones, The written content will be written at the same time RL_T2L, When T2H The written content will also be written at the same time That is, the timer counter RL_T2H When it is forbidden to work, yes That is, the timer counter 2 the pair is allowed to work, the pair writes the content, actually T2H In fact, it is not written to the current register but is written to a hidden register RL_T2L In, right T2H Writing content is not actually writing Current register T2H , but is written to a hidden register , This can be cleverly achieved RL_T2L Bit reload timer. When reading T2H When the content of the sum is, the content read is T2H and T2L The content instead of RL_T2H and RL_T2L The content.

[T2H,T2L] The overflow is not only set to the interrupt request flag (T2IF), Go to the execution time Interrupt the program, and it will automatically will make [RL_T2H,RL_T2L] Reload the content [T2H,T2L]。

13.3.6 Timer calculation formula 2

Timer speed	Cycle calculation formula
1T	Timer period = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)}$ (Automatic reloading)
12T	Timer period = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)} \times 12$ (Automatic reloading)

13.4 Timer 3/4 (24 Bit timer, bit prescaler +16 Bit timing)

13.4.1 Timer 4/3 Control register (T4T3M)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

T4R : Timer4 The operation control bit

0 : The timer stops counting

1 : The timer starts counting

T4_C/T : Control the timer4 Used as a timer or counter, clear0 It is used as a timer (counting the internal system clock), set1 Used as a counter (pair pinT4/P0.6 The external pulse is counted).

T4x12 : Timer4 Speed control bit

0 Mode, that is, clock CPU Frequency division (FOSC/12) 12

1 Mode, that is CPU The clock is not divided by frequency (FOSC)

T4CLKO : Timer4 Clock output control 0: Turn off the clock output 1: Enable P0.7 The port uses the timer clock output function when the timer count overflows. , P0.7 The level of the port is automatically flipped.

T3R : Timer3 The operation control bit

0: Timer Stop counting 3

1 : The timer starts counting

T3_C/T : Control the timer3 Used as a timer or counter, clear0 It is used as a timer (counting the internal system clock), set1 Used as a counter (pair pinT3/P0.4 The external pulse is counted).

T3x12 : Timer3 Speed control bit

0 Mode, that is, clock CPU Frequency division (FOSC/12) 12

1 Mode, that is CPU The clock is not divided by frequency (FOSC)

T3CLKO : Timer3 Clock output control 0: Turn off the clock output 1: Enable P0.5 The port uses the timer clock output function when the timer count overflows. , P0.5 The level of the port is automatically flipped.

13.4.2 Timer count register (T3L , T3H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

Timer/counter 3 The working mode is fixed as 16 Bit overload mode , T3L and T3L Is the low byte , T3H Combine into T3H Is the high byte. When [T3H,T3L] in 16 When the bit count value overflows, the system will automatically send the internal 16 Overloadin the value is loaded [T3H,T3L] in.

13.4.3 Timer count register (T4L , T4H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

/Timer counter The working mode is fixed as 4 Bit overload mode , and T4H Combine into one 16 Is a low register , bytes , T4H 16 Is the high byte. when [T4H,T4L] in T4L 16 When the bit count value overflows, the system will automatically send the internal Overload value loading [T4H,T4L] in. Bits in the overload register

13.4.4 Timer's 3 Bit prescaler register (TM3PS)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TM3PS	FEA3H								

Timer 3 Clock = system clock SYSclk ÷ (TM3PS + 1)

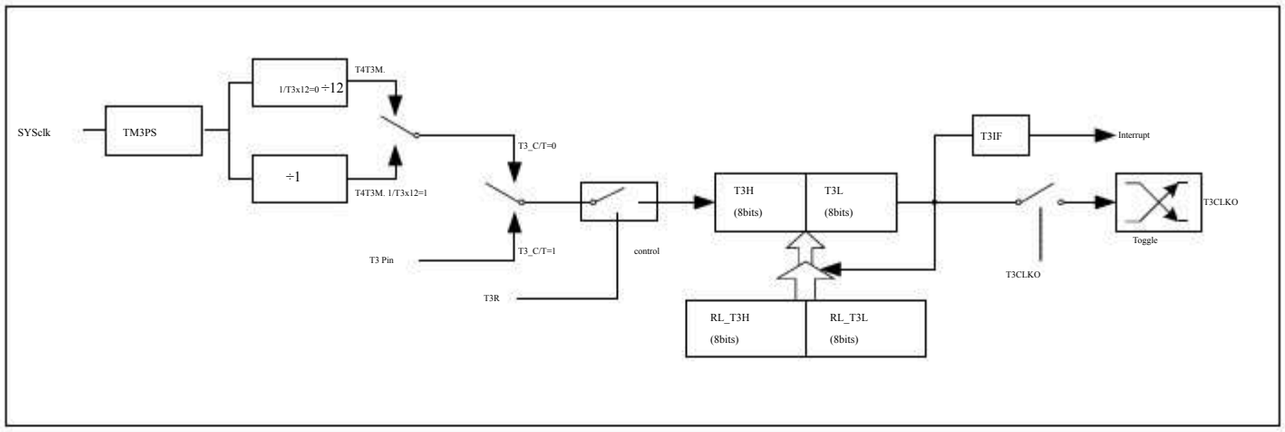
13.4.5 Timer's 4 Bit prescaler register (TM4PS)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TM4PS	FEA4H								

Timer 4 Clock = system clock SYSclk ÷ (TM4PS + 1)

3 Timer operating mode 13.4.6

The schematic diagram of timer/counter 3 is as follows：



/ Timer counter 3 Working mode：16 Bit automatic reload mode

T3R/T4T3M.3 for T4T3M Control bits in the register，T4T3M See the previous section for a description of the specific functions of the register. Registration introduction.

when T3_C/T =0 When the multiplexer is connected to the system clock output，T3 Count the internal system clock， T3_C/T =1 When the multi-channel switch is connected to an external pulse input, that is, the counting mode. T3 Work in timing mode, when T3 Work pin counting

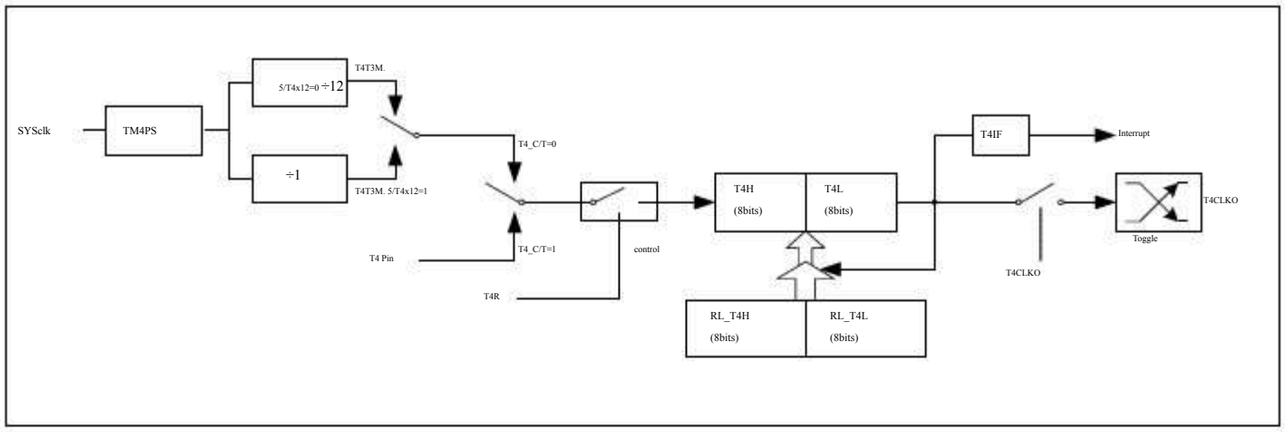
There are two types of timer for single chip microcomputer Standard Rate: One is Mode, each 12T A clock plus, with tradition 12 8051 Same as the single chip microcom The other is 1T Mode, each clock is added, the speed is traditional 8051 Single chip microcomputer 12 The rate is determined by the special function register in T3x12 Decide if T3x12=0，T3 Then work in 12T Mode; if T3x12=1，T3 Then work in 1T pattern

Timer 3 and RL_T3L。 RL_T3H with Share the same address，RL_T3L with T3H T3L Share the same address. when T3R=0 register There are two hidden ones RL_T3L The written content will be written at the same time RL_T3L, When T3H The written content will also be written at the same time That is, the timer counter RL_T3H At the same time That is, the timer counter 3 When it is forbidden to work, yes the pair is allowed to work, the pair writes the content, actually T3H In fact, it is not written to the current register but is written to a hidden register RL_T3L In, right T3H Writing content is not actually writing Current register T3H , but is written to a hidden register , This can be cleverly achieved 16 Bit reload timer. When reading T3H When the content of the sum is, the content read is T3H and T3L The content instead of RL_T3H and RL_T3L The content.

[T3H,T3L] The overflow is not only set to the interrupt request flag (T3IF), Go to the execution time Interrupt the program, and it will automatically will make [RL_T3H,RL_T3L] Reload the content [T3H,T3L]。

13.4.7 Timer operating mode 4

The schematic diagram of timer/counter 4 is as follows：



/ Timer counter 4 Working mode：16 Bit automatic reload mode

T4R/T4T3M.7 for T4T3M Control bits in the register，T4T3M See the previous section for a description of the specific functions of the register. Register

introduction.

when T4_C/T=0 When the multiplexer is connected to the system clock output，T4 Count the internal system clock， T4_C/T=1

When the multi-channel switch is connected to an external pulse input, that is, when T4 Work in timing mode, when T4 Work in counting mode.

There are two types of timer for single chip microcomputer Rate: One is STC Mode, each 12T A clock plus, with tradition 12 8051 Same as the single chip microcom

The other is 1T Mode, each clock is added, the speed is traditional 8051 Single chip microcomputer 12 T4 The rate is determined by the special function register T4T3M

in T4x12 Decide if T4x12=0，T4 Then work in 12T Mode; if T4x12=1，T4 Then work in 1T pattern

Timer 4 and RL_T4L。 RL_T4H with Share the same address，RL_T4L with T4H T4L

Share the same address. when T4R=0 register There are two hidden ones, The written content will be written at the same time RL_T4L, When

T4H The written content will also be written at the same time That is, the timer counter RL_T4H while That is, the timer counter 4 the pair is allowed to work, the pair writes the content, actually T4H

In fact, it is not written to the current register but is written to a hidden register RL_T4L In, right T4H Writing content is not actually writing

Current register T4H , but is written to a hidden register , This can be cleverly achieved RL_T4H Bit reload timer. When reading T4H

When the content of the sum is, the content read is T4H and T4L The content instead of RL_T4H and RL_T4L The content.

[T4H,T4L] The overflow is not only set to the interrupt request flag (T4IF), Go to the execution time Interrupt the program, and it will automatically

will make [RL_T4H,RL_T4L] Reload the content [T4H,T4L]。

13.4.8 3 Timer calculation formula

Timer speed	Cycle calculation formula
1T	Timer period = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)}$ (Automatic reloading)
12T	Timer period = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)} \times 12$ (Automatic reloading)

13.4.9 4 Timer calculation formula

Timer speed	Cycle calculation formula
1T	Timer period = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)}$ (Automatic reloading)
12T	Timer period = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)} \times 12$ (Automatic reloading)

13.5 Sample program

13.5.1 Timer (mode 0) - 160 Bits are automatically used for testing

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;
sbit P10 = P1^0;

void TM0_Isr() interrupt 1
{
    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;
    TMOD = 0x00;                    //pattern 0
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //Start the timer
    ET0 = 1;                        //Enable timer interrupt
    EA = 1;
    while (1);
}
```

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     000BH
        LJMP    TM0ISR

        ORG     0100H
TM0ISR:
        CPL     P1.0                ;Test port
        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     TMOD,#00H           ;pattern 0
        MOV     TL0,#66H            ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB    TR0                 ;Start the timer
        SETB    ET0                 ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.2 Timer (mode - 161 Bits are not automatically reloaded)

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;

sbit P10        =   P1^0;


void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                 //Reset timing parameters
    TH0 = 0xfc;
    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;
    TMOD = 0x01;                //pattern 1
    TL0 = 0x66;                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                    //Start the timer
    ET0 = 1;                    //Enable timer interrupt
    EA = 1;
    while (1);

}
```

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     000BH
                LJMP    TM0ISR

                ORG     0100H
TM0ISR:
                MOV     TL0,#66H        ;Reset timing parameters
                MOV     TH0,#0FCH
                CPL     P1.0            ;Test port
                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     TMOD,#01H       ;pattern 1
                MOV     TL0,#66H        ;65536-11.0592M/12/1000
                MOV     TH0,#0FCH
                SETB    TR0             ;Start the timer
                SETB    ET0             ;Enable timer interrupt
                SETB    EA

                JMP     $

                END
```

13.5.3 Timer (mode 2 - 8 Bits are automatically overloaded), Used as timing

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;

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    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;
    TMOD = 0x02;                                    //pattern 2
    TL0 = 0xf4;                                     //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;                                        //Start the timer
    ET0 = 1;                                        //Enable timer interrupt
    EA = 1;
    while (1);

}
```

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     000BH
                LJMP    TM0ISR

                ORG     0100H
TM0ISR:
                CPL     P1.0            ;Test port
                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     TMOD,#02H       ;pattern 2
                MOV     TL0,#0F4H       ;256-11.0592M/12/76K
                MOV     TH0,#0F4H
                SETB    TR0             ;Start the timer
                SETB    ET0             ;Enable timer interrupt
                SETB    EA

                JMP     $

                END
```

13.5.4 Timer (mode 0) - 16 bits are automatically overloaded, non-maskable interrupts), used as

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;

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    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;
    TMOD = 0x03;                                    //pattern 3
    TL0 = 0x66;                                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                        //Start the timer
    ET0 = 1;                                        //Enable timer interrupt
//  EA = 1;                                         //Not affected by control
    while (1);
}
```

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     000BH
```

```
                LJMP    TM0ISR

                ORG     0100H
TM0ISR:
                CPL     P1.0            ;Test port
                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     TMOD,#03H       ;pattern 3
                MOV     TL0,#66H        ;65536-11.0592M/12/1000
                MOV     TH0,#0FCH
                SETB    TR0             ;Start the timer
                SETB    ET0             ;Enable timer interrupt
;               SETB    EA              ;Not affected by control

                JMP     $

                END
```

13.5.5 Timer (external counting-extended T0 Interrupt for external falling edge)

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;

sbit P10 = P1^0;
```

```
void TM0_Isr() interrupt 1
{
    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;
    TMOD = 0x04;                                    //External counting mode
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                                        //Start the timer
    ET0 = 1;                                        //Enable timer interrupt
    EA = 1;
    while (1);
}
```

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     000BH
        LJMP    TM0ISR

        ORG     0100H
TM0ISR:
        CPL     P1.0                            ;Test port
        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     TMOD,#04H       ;External counting mode
        MOV     TL0,#0FFH
        MOV     TH0,#0FFH
        SETB    TR0             ;Start the timer
        SETB    ET0             ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.6 Timer (measuring pulse width-$_{INT0}$ High level width)

C Language code

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

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

sfr AUXR  = 0x8e;
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 INT0_Isr() interrupt 0
{
    P0 = TL0;   //TL0 The low byte of the measured value
    P1 = TH0;   //TH0 is the high byte of the measured value
}

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;
    AUXR = 0x80;                    pattern //1T
    TMOD = 0x08;                    //Enable GATE,INT0          1Enable timing for time
    TL0 = 0x00;
    TH0 = 0x00;
    while (P32);                                              Is low //wait INT0
    TR0 = 1;                                                  //Start the timer
    IT0 = 1;                                                  //Enable INT0 Falling edge interrupt
    EX0 = 1;
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR    DATA    8EH
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     0003H
        LJMP    INT0ISR

        ORG     0100H
INT0ISR:
        MOV     P0,TL0          ;TL0 The low byte of the measured value
        MOV     P1,TH0          ;TH0 is the high byte of the measured value
        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     AUXR,#80H
MOV     TMOD,#08H
MOV     TL0,#00H
MOV     TH0,#00H
JB      P3.2,$
SETB    TR0
SETB    IT0
SETB    EX0
SETB    EA

JMP     $

END
```

pattern
;;IT Enable GATE,INT0 ,Enable timing for time

Is low ;wait INT0
;Start the timer
;Enable INT0 Falling edge interrupt

13.5.7 Timer (mode)0, Clock divider output

C Language code

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

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

sfr     INTCLKO     =   0x8f;
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;
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;
    TMOD = 0x00;                    //pattern 0
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //Start the timer
    INTCLKO = 0x01;                 //Enable clock output
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

INTCLKO     DATA    8FH
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     TMOD,#00H       ;pattern 0
            MOV     TL0,#66H        ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0             ;Start the timer
```

```
        MOV     INTCLKO,#01H    ;Enable clock output
        JMP     $
        END
```

13.5.8 Timer (mode -160 Bits are automatically used and cleared)

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;
sbit P10        =   P1^0;

void TM1_Isr() interrupt 3
{
    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;
    TMOD = 0x00;
    TL1 = 0x66;                 //pattern 0
    TH1 = 0xfc;                 //65536-11.0592M/12/1000
    TR1 = 1;                    //Start the timer
    ET1 = 1;                    //Enable timer interrupt
    EA = 1;
```

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

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     001BH
        LJMP    TM1ISR

        ORG     0100H
TM1ISR:
        CPL     P1.0            ;Test port
        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     TMOD,#00H       ;pattern 0
        MOV     TL1,#66H        ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1             ;Start the timer
        SETB    ET1             ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.9 Timer (mode - 161 Bits are not automatically reloaded)

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;
sbit P10 = P1^0;

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;                 //Reset timing parameters
    TH1 = 0xfc;
    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;
    TMOD = 0x10;                //pattern 1
    TL1 = 0x66;                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                    //Start the timer
    ET1 = 1;                    //Enable timer interrupt
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is
;

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     001BH
        LJMP    TM1ISR

        ORG     0100H
TM1ISR:
        MOV     TL1,#66H        ;Reset timing parameters
        MOV     TH1,#0FCH
        CPL     P1.0            ;Test port
        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     TMOD,#10H       ;pattern 1
        MOV     TL1,#66H        ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1             ;Start the timer
        SETB    ET1             ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.10 Timer (mode 2 - 8 Bits are automatically overloaded), Used as timing

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;
sbit P10                =       P1^0;

void TM1_Isr() interrupt 3
{
        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;
        TMOD = 0x20;                                    //pattern 2
        TL1 = 0xf4;                                     //256-11.0592M/12/76K
        TH1 = 0xf4;
        TR1 = 1;                                        //Start the timer
        ET1 = 1;                                        //Enable timer interrupt
        EA = 1;
        while (1);
}
```

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     001BH
                LJMP    TM1ISR

                ORG     0100H
TM1ISR:
                CPL     P1.0                ;Test port
                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     TMOD,#20H           ;pattern 2
                MOV     TL1,#0F4H           ;256-11.0592M/12/76K
                MOV     TH1,#0F4H
                SETB    TR1                 ;Start the timer
                SETB    ET1                 ;Enable timer interrupt
                SETB    EA

                JMP     $

                END
```

13.5.11 Timer (external counting-extended T1 Interrupt for external falling edge)

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;

sbit        P10         =   P1^0;

void TM1_Isr() interrupt 3
{
    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;
    TMOD = 0x40;                                    //External counting mode
    TL1 = 0xff;
    TH1 = 0xff;
    TR1 = 1;                                        //Start the timer
    ET1 = 1;                                        //Enable timer interrupt
    EA = 1;
    while (1);
}
```

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     001BH
        LJMP    TM1ISR

        ORG     0100H
TM1ISR:
        CPL     P1.0                ;Test port
        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     TMOD,#40H           ;External counting mode
        MOV     TL1,#0FFH
        MOV     TH1,#0FFH
        SETB    TR1                 ;Start the timer
        SETB    ET1                 ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.12 Timer (measuring pulse width-$_{INT1}$ High level width)

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     AUXR    =   0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TL1;                                   //TL1 The low byte of the measured value
    P1 = TH1;                                   //TH1 is the high byte of the measured value
}

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;
    AUXR = 0x40;                                //pattern //1T
    TMOD = 0x80;                                //Enable GATE,INT1 1Enable timing for time
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);                               //Is low //wait INT1
    TR1 = 1;                                    //Start the timer
    IT1 = 1;                                    //Enable INT1 Falling edge interrupt
    EX1 = 1;
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR    DATA    8EH
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             0013H
                LJMP            INT1ISR

                ORG             0100H
INT1ISR:
                MOV             P0,TL1          ;TL1 The low byte of the measured value
                MOV             P1,TH1          ;TH1 is the high byte of the measured value
                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             AUXR,#40H
                MOV             TMOD,#80H       ;;1T pattern Enable GATE,INT1 ,Enable timing for time
                MOV             TL1,#00H
                MOV             TH1,#00H
                JB              INT1,$          Is low ;wait INT1
                SETB            TR1             ;Start the timer
                SETB            IT1             ;Enable INT1 Falling edge interrupt
                SETB            EX1
                SETB            EA

                JMP             $

                END
```

13.5.13 Timer (mode)$_0$, Clock divider output

C Language code

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

#include "reg51. h"
#include "intrins. h"
sfr INTCLKO     =   0x8f;
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;
    TMOD = 0x00;                //pattern 0
    TL1 = 0x66;                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                    //Start the timer
    INTCLKO = 0x02;             //Enable clock output
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

INTCLKO     DATA    8FH
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     TMOD,#00H       ;pattern 0
MOV     TL1,#66H        ;65536-11.0592M/12/1000
MOV     TH1,#0FCH
SETB    TR1             ;Start the timer
MOV     INTCLKO,#02H    ;Enable clock output

JMP     $

END
```

13.5.14 Timer (mode) for serial port 0 1 Baud rate generator

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 AUXR = 0x8e;
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;
bit busy;
char wptr;
char rptr;
char buffer[16];
void Uart1Isr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
```

```
        }
        if (RI)
        {
RI = 0;
buffer[wptr++] = SBUF;
wptr &= 0x0f;
        }
}
void UartInit()
{
        SCON = 0x50;
        TMOD = 0x00;
        TL1 = BRT;
        TH1 = BRT >> 8;
        TR1 = 1;
        AUXR = 0x40;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

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

void UartSendStr(char *p)
{
while (*p)
{
                UartSend(*p++);
        }
}
void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

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

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

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR    DATA    8EH

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

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     0023H
        LJMP    UART_ISR

        ORG     0100H

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

        JNB     TI,CHKRI
        CLR     TI
        CLR     BUSY
CHKRI:
        JNB     RI,UARTISR_EXIT
        CLR     RI
        MOV     A,WPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     @R0,SBUF
        INC     WPTR
UARTISR_EXIT:
        POP     PSW
        POP     ACC
```

```
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

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

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

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

```
                ADD             A,#BUFFER
                MOV             R0,A
                MOV             A,@R0
                LCALL           UART_SEND
                INC             RPTR
                JMP             LOOP

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

                END
```

13.5.15 Timer (mode) for serial port 2 1 Baud rate generator

C Language code

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

#include "reg51. h"
#include "intrins. h"
#define FOSC 11059200UL
#define BRT (256 - FOSC / 115200 / 32)
sfr AUXR = 0x8e;
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;
bit busy;
char wptr;
char rptr;
char buffer[16];
void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
```

```
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void UartSendStr(char *p)
{
while (*p)
{
        UartSend(*p++);
    }
}
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");
    while (1)
    {
        if (rptr ! = wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

The test operating frequency is 11.0592MHz

```
AUXR    DATA    8EH

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

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     0023H
        LJMP    UART_ISR

        ORG     0100H

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

        JNB     TI,CHKRI
        CLR     TI
        CLR     BUSY
CHKRI:
        JNB     RI,UARTISR_EXIT
        CLR     RI
        MOV     A,WPTR
        ANL     A,#0FH
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     @R0,SBUF
        INC     WPTR
UARTISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

UART_INIT:
        MOV     SCON,#50H
        MOV     TMOD,#20H
        MOV     TL1,#0FDH               ;256-11059200/115200/32=0FDH
        MOV     TH1,#0FDH
```

```
        SETB    TR1
        MOV     AUXR,#40H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

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

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

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

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART_SENDSTR

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

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

            END
```

2 Timer (16 13.5.16 Bits are automatically overloaded), used as timing

C Language code

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


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

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

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

sbit P10    = P1^0;


void TM2_Isr() interrupt 12
{
    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;
```

```
    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                                //Start the timer
    IE2 = ET2;                                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

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     0063H
        LJMP    TM2ISR

        ORG     0100H
TM2ISR:
        CPL     P1.0                            ;Test port
        RETI

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

```
MOV     T2L,#66H        ;65536-11.0592M/12/1000
MOV     T2H,#0FCH
MOV     AUXR,#10H       ;Start the timer
MOV     IE2,#ET2        ;Enable timer interrupt
SETB    EA

JMP     $

END
```

13.5.17 Timer (external counting-extended T2 Interrupt for external falling edge)

C Language code

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

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

sfr T2L  = 0xd7;
sfr T2H  = 0xd6;
sfr AUXR = 0x8e;
sfr IE2  = 0xaf;
#define ET2 0x04
sfr AUXINT1F = 0xef;
#define T2IF 0x01
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;
sbit P10 = P1^0;

void TM2_Isr() interrupt 12
{
    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;
    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;                //Set the external counting mode and start the timer
    IE2 = ET2;                  //Enable timer interrupt
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
IE2     DATA    0AFH
ET2     EQU     04H
AUXINT1F DATA   0EFH
T2IF    EQU     01H

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     0063H
        LJMP    TM2ISR

        ORG     0100H
TM2ISR:
        CPL     P1.0            ;Test port
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
```

```
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T2L,#0FFH
        MOV     T2H,#0FFH
        MOV     AUXR,#18H       ; Set the external counting mode and start the timer
        MOV     IE2,#ET2        ; Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.18 Timer, clock divider output

C Language code

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

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

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

    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                                //Start the timer
    INTCLKO = 0x04;                             //Enable clock output

    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

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     T2L,#66H                        ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H                       ;Start the timer
        MOV     INTCLKO,#04H                    ;Enable clock output

        JMP     $

        END
```

2 Timer for serial port 1 Baud rate generator 13.5.19

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 AUXR
sfr T2H                                 =       0x8e;
sfr T2L                                 =       0xd6;
sfr P1M1                                =       0xd7;
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;
bit busy;                               =       0xca;
char wptr;
char rptr;
char buffer[16];
void UartIsr() interrupt 4
{
        if (TI)
        {
        TI = 0;
        busy = 0;
        }
        if (RI)
        {
RI = 0;
buffer[wptr++] = SBUF;
wptr &= 0x0f;
        }
}
void UartInit()
{
SCON = 0x50;
T2L = BRT;
T2H = BRT >> 8;
AUXR = 0x15;
```

```
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

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

void UartSendStr(char *p)
{
while (*p)
{
                UartSend(*p++);
        }
}
void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSendStr("Uart Test !\r\n");
        while (1)
        {
                if (rptr ! = wptr)
                {
                        UartSend(buffer[rptr++]);
                        rptr &= 0x0f;
                }
        }
}
```

Assembly code

```
;The test operating frequency is
                        11.0592MHz

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

BUSY            BIT     20H. 0
```

```
WPTR            DATA    21H
RPTR            DATA    22H
BUFFER          DATA    23H             ;16 bytes

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     0023H
                LJMP    UART_ISR

                ORG     0100H

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

                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,SBUF
                INC     WPTR
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#15H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
```

```
                MOV     SBUF,A
                RET

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

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

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

                END
```

13.5.20 2 Timer for serial port 2 Baud rate generator

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 AUXR

sfr T2H                                 =       0x8e;

sfr T2L                                 =       0xd6;

sfr S2CON                               =       0xd7;

sfr S2BUF                               =       0x9a;

sfr IE2                                 =       0xaf;

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;

bit busy;                               =       0xca;

char wptr;

char rptr;

char buffer[16];



void Uart2Isr() interrupt 8
{
        if (S2CON & 0x02)
        {
        S2CON &= ~0x02;
        busy = 0;
        }
        if (S2CON & 0x01)
        {
S2CON &= ~0x01;
buffer[wptr++] = S2BUF;
wptr &= 0x0f;
        }
}
void Uart2Init()
{

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

}
```

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

void Uart2SendStr(char *p)
{
while (*p)
{
        Uart2Send(*p++);
    }
}
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");
    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S2CON   DATA    9AH
S2BUF   DATA    9BH
IE2     DATA    0AFH

BUSY    BIT     20H.
WPTR    DATA    0 21H
```

```
RPTR            DATA    22H
BUFFER          DATA    23H             ;16 bytes

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     0043H
                LJMP    UART2_ISR

                ORG     0100H

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

                MOV     A,S2CON
                JNB     ACC. 1,CHKRI
                ANL     S2CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC. 0,UART2ISR_EXIT
                ANL     S2CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S2BUF
                INC     WPTR
UART2ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

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

UART2_SEND:
                JB      BUSY,$
                SETB    BUSY
```

```
                MOV     S2BUF,A
                RET

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

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

                LCALL   UART2_INIT
                MOV     IE2,#01H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART2_SENDSTR

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

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

                END
```

13.5.21 2 Timer for serial port 3 Baud rate generator

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 AUXR
sfr T2H
                                        =       0x8e;
sfr T2L
                                        =       0xd6;
sfr S3CON                               =       0xd7;
sfr S3BUF                               =       0xac;
sfr IE2                                 =       0xad;
sfr P0M1                                =       0xaf;
sfr P0M0
                                        =       0x93;
sfr P1M1                                =       0x94;
sfr P1M0                                =       0x91;
sfr P2M1                                =       0x92;
sfr P2M0                                =       0x95;
sfr P3M1                                =       0x96;
sfr P3M0                                =       0xb1;
sfr P4M1                                =       0xb2;
sfr P4M0                                =       0xb3;
sfr P5M1                                =       0xb4;
sfr P5M0                                =       0xc9;
bit busy;                               =       0xca;
char wptr;
char rptr;
char buffer[16];


void Uart3Isr() interrupt 17
{
        if (S3CON & 0x02)
        {
        S3CON &= ~0x02;
        busy = 0;
        }
        if (S3CON & 0x01)
        {
S3CON &= ~0x01;
buffer[wptr++] = S3BUF;
wptr &= 0x0f;
        }
}
void Uart3Init()
{
        S3CON = 0x10;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR = 0x14;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}
```

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

void Uart3SendStr(char *p)
{
while (*p)
{
        Uart3Send(*p++);
    }
}
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");
    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
The test operating frequency is
11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S3CON   DATA    0ACH
S3BUF   DATA    0ADH
IE2     DATA    0AFH

BUSY    BIT     20H.
WPTR    DATA    0 21H
RPTR    DATA    22H
```

```
BUFFER          DATA    23H             ;16 bytes

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

                ORG     0000H
                LJMP    MAIN
                ORG     008BH
                LJMP    UART3_ISR

                ORG     0100H

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

                MOV     A,S3CON
                JNB     ACC. 1,CHKRI
                ANL     S3CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC. 0,UART3ISR_EXIT
                ANL     S3CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S3BUF
                INC     WPTR
UART3ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

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

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

```
                RET

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

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

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

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

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

                END
```

13.5.22 2 Timer for serial port 4 Baud rate generator

C Language code

// The test operating frequency is

```
#include "reg51. h"
#include "intrins. h"
#define
                FOSC                    11059200UL
#define BRT
                                        (65536 - FOSC / 115200 / 4)
sfr AUXR
sfr T2H                                 =       0x8e;
sfr T2L                                 =       0xd6;
sfr S4CON                               =       0xd7;
sfr S4BUF                               =       0x84;
sfr IE2                                 =       0x85;
sfr P0M1                                =       0xaf;
sfr P0M0
sfr P1M1                                =       0x93;
sfr P1M0                                =       0x94;
sfr P2M1                                =       0x91;
sfr P2M0                                =       0x92;
sfr P3M1                                =       0x95;
sfr P3M0                                =       0x96;
sfr P4M1                                =       0xb1;
sfr P4M0                                =       0xb2;
sfr P5M1                                =       0xb3;
sfr P5M0                                =       0xb4;
bit busy;                               =       0xc9;
char wptr;                              =       0xca;
char rptr;
char buffer[16];


void Uart4Isr() interrupt 18
{
        if (S4CON & 0x02)
        {
        S4CON &= ~0x02;
        busy = 0;
        }
        if (S4CON & 0x01)
        {
S4CON &= ~0x01;
buffer[wptr++] = S4BUF;
wptr &= 0x0f;
        }
}
void Uart4Init()
{
        S4CON = 0x10;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR = 0x14;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}
```

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

void Uart4SendStr(char *p)
{
while (*p)
{
                Uart4Send(*p++);
        }
}
void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        Uart4Init();
        IE2 = 0x10;
        EA = 1;
        Uart4SendStr("Uart Test !\r\n");
        while (1)
        {
                if (rptr ! = wptr)
                {
                        Uart4Send(buffer[rptr++]);
                        rptr &= 0x0f;
                }
        }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH

BUSY    BIT     20H.
WPTR    DATA    0 21H
RPTR    DATA    22H
```

```
BUFFER      DATA    23H             ;16 bytes

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

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

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

            MOV     A,S4CON
            JNB     ACC. 1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC. 0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

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

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

```
                RET

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

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

                LCALL   UART4_INIT
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR
LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART4_SEND
                INC     RPTR
                JMP     LOOP

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

                END
```

13.5.23 3 Timer (16 Bits are automatically overloaded), used as timing

C Language code

//The test operating frequency is

```
#include "reg51. h"
#include "intrins. h"
sfr
                T4T3M
sfr T4L
sfr T4H
sfr T3L
sfr T3H
sfr T2L
sfr T2H
sfr AUXR
sfr IE2
#define ET2
#define ET3
#define ET4
sfr AUXINTIF
#define T2IF
#define T3IF
#define T4IF
sfr P1M1
sfr P1M0
sfr P0M1
sfr P0M0
sfr P2M1
sfr P2M0
sfr P3M1
sfr P3M0
sfr P4M1
sfr P4M0
sfr P5M1
sfr P5M0
sbit P10
                                =       P1^0;

void TM3_Isr() interrupt 19
{
        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;
        T3L = 0x66;
        T3H = 0xfc;                                     //65536-11.0592M/12/1000
```

```
    T4T3M = 0x08;                       //Start the timer
    IE2 = ET3;                          //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

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

        ORG     0000H
        LJMP    MAIN
        ORG     009BH
        LJMP    TM3ISR

        ORG     0100H
TM3ISR:
        CPL     P1.0                    ;Test port
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
```

```
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     T3L,#66H        ;65536-11.0592M/12/1000
MOV     T3H,#0FCH
MOV     T4T3M,#08H      ;Start the timer
MOV     IE2,#ET3        ;Enable timer interrupt
SETB    EA

JMP     $

END
```

13.5.24 Timer (external counting-extended T3 Interrupt for external falling edge)

C Language code

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

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

sfr T4T3M   = 0xd1;
sfr T4L     = 0xd3;
sfr T4H     = 0xd2;
sfr T3L     = 0xd5;
sfr T3H     = 0xd4;
sfr T2L     = 0xd7;
sfr T2H     = 0xd6;
sfr AUXR    = 0x8e;
sfr IE2     = 0xaf;
#define ET2     0x04
#define ET3     0x20
#define ET4     0x40
sfr AUXINTIF= 0xef;
#define T2IF    0x01
#define T3IF    0x02
#define T4IF    0x04
sfr P1M1    = 0x91;
sfr P1M0    = 0x92;
sfr P0M1    = 0x93;
sfr P0M0    = 0x94;
sfr P2M1    = 0x95;
sfr P2M0    = 0x96;
sfr P3M1    = 0xb1;
sfr P3M0    = 0xb2;
sfr P4M1    = 0xb3;
sfr P4M0    = 0xb4;
sfr P5M1    = 0xc9;
sfr P5M0    = 0xca;
```

```
sbit            P10             =       P1^0;


void TM3_Isr() interrupt 19
{
        P10 = ! P10;                                    //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;
        T3L = 0x66;                                     //65536-11.0592M/12/1000
        T3H = 0xfc;
        T4T3M = 0x0c;                                   // Set the external counting mode and start the timer
        IE2 = ET3;                                      //Enable timer interrupt
        EA = 1;
        while (1);

}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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


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

```
P4M1            DATA    0B3H
P4M0            DATA    0B4H
P5M1            DATA    0C9H
P5M0            DATA    0CAH

                ORG     0000H
                LJMP    MAIN
                ORG     009BH
                LJMP    TM3ISR

                ORG     0100H
TM3ISR:
                CPL     P1.0                ;Test port
                RETI

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

                MOV     T3L,#66H            ;65536-11.0592M/12/1000
                MOV     T3H,#0FCH
                MOV     T4T3M,#0CH          ; Set the external counting mode and start the timer
                MOV     IE2,#ET3            ;Enable timer interrupt
                SETB    EA

                JMP     $

                END
```

13.5.25 Timer, clock divider output

C Language code

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

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

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

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;               // Enable the clock output and start the timer
    while (1);

}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

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

```
P5M0            DATA    0CAH

                ORG     0000H
                LJMP    MAIN

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

                MOV     T3L,#66H        ;65536-11.0592M/12/1000
                MOV     T3H,#0FCH
                MOV     T4T3M,#09H      ; Enable the clock output and start the timer

                JMP     $

                END
```

3 Timer for serial port 3 Baud rate generator 13.5.26

C Language code

The test operating frequency is

```
#include "reg51. h"
#include "intrins. h"
#define
                FOSC            11059200UL
#define BRT
                (65536 - FOSC / 115200 / 4)
sfr T4T3M
sfr T4L         =       0xd1;
sfr T4H         =       0xd3;
sfr T3L         =       0xd2;
sfr T3H         =       0xd5;
sfr T2L         =       0xd4;
sfr T2H         =       0xd7;
sfr S3CON       =       0xd6;
sfr S3BUF       =       0xac;
sfr IE2         =       0xad;
sfr P0M1        =       0xaf;
sfr P0M0        =       0x93;
sfr P1M1        =       0x94;
sfr P1M0        =       0x91;
                =       0x92;
```

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

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

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
    S3CON &= ~0x02;
    busy = 0;
    }
    if (S3CON & 0x01)
    {
S3CON &= ~0x01;
buffer[wptr++] = S3BUF;
wptr &= 0x0f;
    }
}
void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void Uart3SendStr(char *p)
{
while (*p)
{
        Uart3Send(*p++);
    }
}
void main()
{
P0M0 = 0x00;
P0M1 = 0x00;
```

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

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");
    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

T4T3M   DATA    0D1H
T4L     DATA    0D3H
T4H     DATA    0D2H
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H
S3CON   DATA    0ACH
S3BUF   DATA    0ADH
IE2     DATA    0AFH

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

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

```
                ORG     0000H
                LJMP    MAIN
                ORG     008BH
                LJMP    UART3_ISR

                ORG     0100H

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

                MOV     A,S3CON
                JNB     ACC. 1,CHKRI
                ANL     S3CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC. 0,UART3ISR_EXIT
                ANL     S3CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S3BUF
                INC     WPTR
UART3ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

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

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

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

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
```

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

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

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

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

                END
```

4 Timer (16 13.5.27 Bits are automatically overloaded), used as timing

C Language code

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

#include "reg51. h"
#include "intrins. h"
sfr T4T3M   = 0xd1;
sfr T4L     = 0xd3;
sfr T4H     = 0xd2;
sfr T3L     = 0xd5;
sfr T3H     = 0xd4;
sfr T2L     = 0xd7;
sfr T2H     = 0xd6;
sfr AUXR    = 0x8e;
sfr IE2     = 0xaf;
#define ET2 0x04
```

```
#define     ET3                 0x20
#define     ET4                 0x40
sfr         AUXINTIF    =       0xef;
#define     T2IF                0x01
#define     T3IF                0x02
#define     T4IF                0x04

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

sbit        P10         =       P1^0;

void TM4_Isr() interrupt 20
{
    P10 = ! P10;                            //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;
    T4L = 0x66;                             //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80;                           //Start the timer
    IE2 = ET4;                              //Enable timer interrupt
    EA = 1;
    while (1);

}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
```

```
T3H             DATA    0D4H
T2L             DATA    0D7H
T2H             DATA    0D6H
AUXR            DATA    8EH
IE2             DATA    0AFH
ET2             EQU     04H
ET3             EQU     20H
ET4             EQU     40H
AUXINT1F        DATA    0EFH
T21F            EQU     01H
T31F            EQU     02H
T41F            EQU     04H

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

                ORG     0000H
                LJMP    MAIN
                ORG     00A3H
                LJMP    TM4ISR

                ORG     0100H
TM4ISR:
                CPL     P1.0            ;Test port
                RETI

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

                MOV     T4L,#66H        ;65536-11.0592M/12/1000
                MOV     T4H,#0FCH
                MOV     T4T3M,#80H      ;Start the timer
                MOV     IE2,#ET4        ;Enable timer interrupt
                SETB    EA

                JMP     $
```

```
END
```

13.5.28 Timer (external counting-extended T4 Interrupt for external falling edge)

C Language code

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

#include "reg51. h"
#include "intrins. h"
sfr T4T3M = 0xd1;
sfr T4L = 0xd3;
sfr T4H = 0xd2;
sfr T3L = 0xd5;
sfr T3H = 0xd4;
sfr T2L = 0xd7;
sfr T2H = 0xd6;
sfr AUXR = 0x8e;
sfr IE2 = 0xaf;
#define ET2 0x04
#define ET3 0x20
#define ET4 0x40
sfr AUXINTIF = 0xef;
#define T2IF 0x01
#define T3IF 0x02
#define T4IF 0x04
sfr P1M1
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;
sbit P10 = 0xca;

void TM4_Isr() interrupt 20
{
    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;
    T4L = 0x66;                     //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0xc0;                   // Set the external counting mode and start the timer
    IE2 = ET4;                      // Enable timer interrupt
    EA = 1;
    while (1);
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

T4T3M   DATA    0D1H
T4L     DATA    0D3H
T4H     DATA    0D2H
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
IE2     DATA    0AFH
ET2     EQU     04H
ET3     EQU     20H
ET4     EQU     40H
AUXINT1F DATA   0EFH
T2IF    EQU     01H
T3IF    EQU     02H
T4IF    EQU     04H

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

        ORG     0000H
        LJMP    MAIN
        ORG     00A3H
        LJMP    TM4ISR

        ORG     0100H
TM4ISR:
        CPL     P1.0                ;Test port
        RETI
```

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

        MOV     T4L,#66H        ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#0C0H     ; Set the external counting mode and start the timer
        MOV     IE2,#ET4        ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.29 Timer, clock divider output

C Language code

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

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

sfr T4T3M   = 0xd1;
sfr T4L     = 0xd3;
sfr T4H     = 0xd2;
sfr T3L     = 0xd5;
sfr T3H     = 0xd4;
sfr T2L     = 0xd7;
sfr T2H     = 0xd6;
sfr P1M1    = 0x91;
sfr P1M0    = 0x92;
sfr P0M1    = 0x93;
sfr P0M0    = 0x94;
sfr P2M1    = 0x95;
sfr P2M0    = 0x96;
sfr P3M1    = 0xb1;
sfr P3M0    = 0xb2;
sfr P4M1    = 0xb3;
sfr P4M0    = 0xb4;
sfr P5M1    = 0xc9;
sfr P5M0    = 0xca;
```

```
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P3M1 = 0x00;
    T4L = 0x66;                     //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x90;                   // Enable the clock output and start the timer
    while (1);

}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

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

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

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
```

```
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     T4L,#66H        ;65536-11.0592M/12/1000
MOV     T4H,#0FCH
MOV     T4T3M,#90H      ; Enable the clock output and start the timer

JMP     $

END
```

4 Timer for serial port 4 Baud rate generator 13.5.30

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 T4T3M
sfr T4L                 =       0xd1;
sfr T4H                 =       0xd3;
sfr T3L                 =       0xd2;
sfr T3H                 =       0xd5;
sfr T2L                 =       0xd4;
sfr T2H                 =       0xd7;
sfr S4CON               =       0xd6;
sfr S4BUF               =       0x84;
sfr IE2                 =       0x85;
sfr P0M1                =       0xaf;
sfr P0M0
sfr P1M1                =       0x93;
sfr P1M0                =       0x94;
sfr P2M1                =       0x91;
sfr P2M0                =       0x92;
sfr P3M1                =       0x95;
sfr P3M0                =       0x96;
sfr P4M1                =       0xb1;
sfr P4M0                =       0xb2;
sfr P5M1                =       0xb3;
sfr P5M0                =       0xb4;
bit busy;               =       0xc9;
char wptr;              =       0xca;
char rptr;
char buffer[16];
```

```
void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
    S4CON &= ~0x02;
    busy = 0;
    }
    if (S4CON & 0x01)
    {
S4CON &= ~0x01;
buffer[wptr++] = S4BUF;
wptr &= 0x0f;
    }
}
void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

void Uart4SendStr(char *p)
{
while (*p)
{
        Uart4Send(*p++);
    }
}
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
    Uart4Init();
    IE2 = 0x10;
    EA = 1;
```

```
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

The test operating frequency is 11.0592MHz

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

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

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

        ORG     0000H
        LJMP    MAIN
        ORG     0093H
        LJMP    UART4_ISR

        ORG     0100H

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

        MOV     A,S4CON
        JNB     ACC. 1,CHKRI
```

```
                ANL     S4CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC. 0,UART4ISR_EXIT
                ANL     S4CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S4BUF
                INC     WPTR
UART4ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

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

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

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

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

                LCALL   UART4_INIT
                MOV     IE2,#10H
```

```
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

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

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

                END
```

14 Serial communication

STC12H The series of microcontrollers provides a full-duplex asynchronous serial communication interface. A serial port consists of It consists of a register, a serial control register and a baud rate generator. The data buffer of each serial port is composed of independent receiving and transmitting buffers, which can transmit and receive data at the same time.

STC12H The serial ports of the series of microcontrollers are Two working methods, of which the baud rate of two methods is variable For different applications to choose from. Serial port 1 There are only two working methods, and the baud rate of both methods is variable. Users can use the software to set different baud rates and choose different working methods. The host can receive it by query or interrupt It is sent for program processing, and it is very flexible to use.

Serial port, serial port All communication ports can be switched to multiple sets of ports through the switching function of the function pin Time-sharing multiplexing is multiple communication ports.

14.1 Serial port related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	Serial port control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	Serial port data register	99H									0000,0000
S2CON	Serial port control register 2	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	Serial port data register 2	9BH									0000,0000
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	Auxiliary register	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	Serial slave address register	A9H									0000,0000
SADEN	Serial slave address masking register	B9H									0000,0000

14.2 serial port 1

14.2.1 Serial port control register (SCON)

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

PCON In the register SM0 When The bit is When, this bit is the frame error detection flag. When Detected during reception an invalid stop position is reached, pass The receiver will be in this position , Must be cleared to zero by the software. when PCON The bit is When, the bit and Specify the serial port together The communication working mode is shown in the following table :

SM0	SM1	Serial port working mode	Function description
0	0	pattern 0	Synchronous shift serial mode variable
0	1	pattern 1	baud rate 8 Bit data method
1	0	pattern 2	fixed baud rate 9 Bit data mode ,
1	1	pattern 3	variable baud rate 9 Bit data method

SM2 : Allowed mode 2 Or mode 3 multi-machine communication control bit When bit When 1 , if REN Bit is, the receiver is in the address frame filtering state. At this time, you can use the received bit 9 (That The position is and SM2 is, if) To filter the address frame , It means that the frame is an address frame, the address information can be entered SBUF and make For, and then in the interrupt service program Then compare the address number; if, it means that the frame is not an address frame, it should be RI=0 Or mode 2 Or mode 3 thrown away and kept , if SM2 0 The position is and Bit is, the receiver is in the state where address frame filtering is prohibited, regardless of the rec Or, the received information can be entered 0 RI=1 , at this time RB8 SBUF , Usually it is a parity bit. pattern And mode 0 It is a non-multi-machine communication method, in these two ways 0 , and make Should be set to.

REN SM2

: Allow Prohibit serial port from receiving control bits

0 : Serial port is prohibited from receiving data

1 : Serial port is allowed to receive data

Usage mode TB8 : When the serial port When , TB8 For the first to be sent Bit data, set or cleared by the software as needed. In the mold formula 0 And mode Or mode 2

RB8 : When the serial port 1 In, this bit is not used. 3 Is the received bit of data, generally used as a parity bit or address frame data Frame flag. In 1 modes and 0 modes 1 , This bit is not needed.

When using mode or mode : Serial port Send an interrupt request flag. In modeln, when the serial port sends data, the first bit, the hardware will automatically Request an interrupt to the host, after responding to the interrupt Must be cleared with software. In other modes, the hardware requests an interrupt when the Automatically TI 1 Set, to CPU stop bit starts to be sent , and the software must be cleared to zero after responding to the interrupt. TI

RI :The serial port receives the interrupt request flag. In mode 0 , When the serial port receives the end of the first bit of data, the hardware will Request an interrupt to the host, and after Must be cleared with software. In other modes, the intermediate time when the stop bit is rece responding to the interrupt , the hardware will Interrupt request is issued, and the software must be cleared to zero after responding to the Set, to

14.2.2 Serial port data register (SBUF)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF: Serial port 1 Data reception / Send buffer. SBUF Actually is 2 A buffer, a read buffer and a write buffer, the two operations are respectivel

Corresponds to two different registers , 1 One is to write only the register (write buffer) , SBUF

1 One is a read-only register (read buffer). For a

read operation, the serial port reception buffer is actually read, and for a write operation, the serial port is triggered to start sending d

14.2.3 Power management register (PCON)

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

: Serial port baud rate control bit

1 SMOD

T: Serial port

0 The baud rate of each mode is not

doubled The baud rate of mode, mode, and mode is doubled

1 Serial port 1 1 2 3

SMOD0 : Frame error detection control bit

0 : No frame error detection function

1 : Enable the frame error detection function. At this time SCON of S1M0FE for FE Function, that is, the frame error detection flag.

14.2.4 1 Auxiliary register (AUXR)

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

0 : Communication speed control in serial mode UART_M0x6 1

0 : Serial port 1 Fosc/2

The baud rate is not doubled, fixed as the

1 : Serial port 1 mode 00 The baud rate multiplier of the mode is fixed as Fosc/12 6 Fosc/2

S1ST2: Serial port 1 Baud rate generator selection bit

0 : Select the timer as the baud rate generator 1

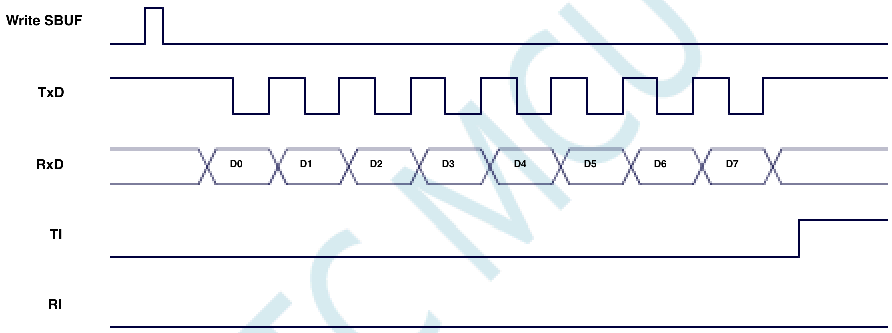
1 : Select the timer as the baud rate generator 2

14.2.5 Serial port mode, mode 0 0 Baud rate calculation formula

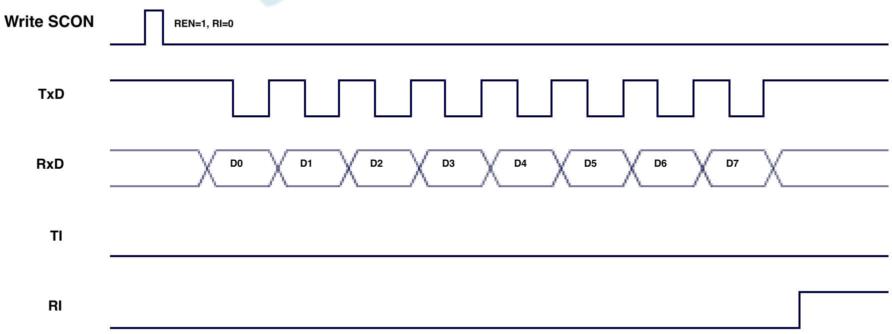
When serial port 1 Select the working mode as the mode When the serial communication interface operates in synchronous shift register mode, when the s Communication speed UART_M0x6 for 0 port is in mode , its baud rate is fixed as the frequency division of the system clock clock (12 SYSclk/12 setting bit UART_M0x6 When the baud rate is fixed as the system Frequency division of the clock frequency ; when set (for 1 serial communication , Is a data port TxD It is a synchronous shift pulse output pin, which transmits and receives bit data, and the low bit comes first. 8

Mode 0 transmission process: when the host performs writing data to the SBUF Start sending when the instruction is executed, the number o transmission buffer , or determines whether it is divided by frequency or SYSclk/12 SYSclk/2 Frequency division) the baud rate from RxD Pin output from 12 Low to high), After sending the interrupt flag TI 1 set , TxD The pin outputs a synchronous shift pulse signal. When the writing signal is valid, one Clock, the transmission control terminal is valid (High level), allow SEND RxD Outputs a synchronous shift pulse. One frame (8 S bit) When the data is sent, each control terminal is restored to its original state, only TI Keep the level high and show an interrupt request state. Sending data agai Before, you must use software to TI 0 Clear.

Mode 0 receiving process: first, the interrupt request flag will be cleared RI 0 Clear and set to allow receiving control bits REN When the start-up mode is connecte Collection process. After starting the receiving process , It is the output terminal of the synchronization pulse. The baud rate RxD Is the serial data input terminal TxD UART_M0x6 SYSclk/2 (by Sure it is 12 SYSclk/12 reception is 2 divider or divider). When a frame of data is received (8 Bit) after , Control signal reset, interrupt flag RI It was placed in the state of interruption application. When receiving again, you must pass the software to RI Clear 0



Send data (serial port 1 mode 0)



Receive data (serial port 1 mode 0)

Working in mode 0, When it must be used for machine communication control bit SM2 . So that it does not affect Bit sum RB8 bit. Due to the fixed baud rate SYSclk/2 , Without the need for a timer to provide, the clock of the microcontroller is directly used as a synchronous shif

The baud rate calculation formula for serial port mode is shown in the following table (SYSclk For the operating frequency of the system) :

UART_M0x6	Baud rate calculation formula
0	Baud rate = SYSclk / 12
1	Baud rate = SYSclk / 2

14.2.6 Serial port mode, mode 1 1 Baud rate calculation formula

When the software sets the format, a frame of SM0, SM1 for" "Time, serial port D1 Carry out work. If this mode is, the mode bit is used 1 8 U bit : 1 Bit start bit , 8 Bit data bit (low bit first) and 1 Bit stop bit. The baud rate is variable, which can be based information SCON for 10 The baud rate needs to be set. TxD For the data transmission port, For the data receiving port, the serial port accepts full duplex send. RxD

Mode transmission process: When the serial communication mode is transmitted, the data is written to the SBUF of the write serial sender 's instruction to start the serial communication signal. "The tag, write still there." 1 "Load the first part of the transmission shift register The control unit starts sending. The shift register continuously transfers data to the right, and "" is constantly moved in on the left side of When the highest displacement of the data reaches the output position of the shift register, it is followed by the first this A state condition, so that 1X The control unit makes the last shift output, and then allows the signal to be "Failed," complete a frame of information , and set the interrupt request bit to request interrupt processing to the host. TI, that is TI=1

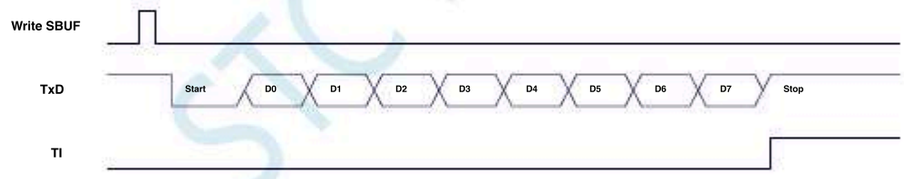
The receiving process: when the software sets the receiving permission flag that is REN=1 When the receiver is RxD pattern 1 The port is sent from "1 " → " 0 "When the falling edge transitions, the receiver is activated to prepare to rece For testing, when it is detected RxD Load the shift register. The received data is moved in from the right side of the received sh Loaded 1FFH Move out to the left, when the starting position 0 to the far left of the shift register, make the controller shift for the last time , Complete the reception of a frame. If the following two conditions are met at the same time :

. RI=0 ;

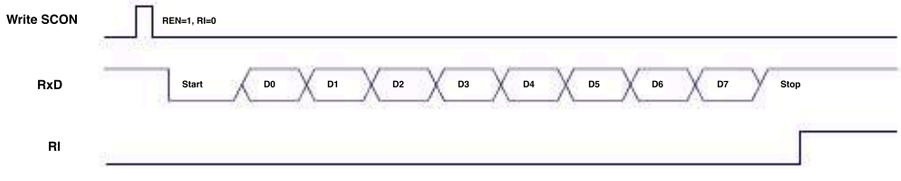
. SM2=0 Or the received stop bit is. 1

The received data is valid and loaded into SBUF, Stop bit entry RB8 , RI The flag is set and an interrupt is requested from the host If the above two conditions cannot be met at the same time, the received data is invalid and lost. Regardless of whether the conditions are me the transition on the port again and continues to receive the next frame. The reception is valid, after the response is interrupted , RI The flag must be cleared by the software RxD 。 Pass "1"→ "0"

Under normal circumstances, when serial communication is operating in mode , SM2 Set to 1 "0"。



Send data (serial port 1 mode) 1



Receive data (serial port 1 mode 1)

The baud rate of the serial port is variable, and its baud rate can be generated by a timer or timer. When the timer is used 1 2 Times the speed), the speed of the corresponding baud rate will also be doubled accordingly. 12

serial port pattern 1 The baud rate calculation formula is shown in the following table (SYSclk is the operating frequency of the system)

Select timer	Timer speed degree	Baud rate calculation formula
Timer2	1T	Timer2 Overload value = 65536 - SYSclk / (Baud rate × 4)
	12T	Timer2 Overload value = 65536 - SYSclk / (12 × 4 × Baud rate)
1 Timer mode 0	1T	Timer1 Overload value = 65536 - SYSclk / (Baud rate × 4)
	12T	Timer1 Overload value = 65536 - SYSclk / (12 × 4 × Baud rate)
1 Timer mode 2	1T	Timer1 Overload value = 256 - SYSclk × 2^SMOD / (Baud rate × 32)
	12T	Timer1 Overload value = 256 - SYSclk × 2^SMOD / (Baud rate × 32 × 12)

The following is the overload value of the timer corresponding to the common frequency and common baud rate

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

14.2.7 Serial port mode, mode 2 2 Baud rate calculation formula

when SM0、SM1 The two digits are when , Serial port 1 2 Work in mode. Serial port 1 The working mode of asynchronous communicati

Mode, the information of one frame is determined by Bit composition : 1 bit start bit , of bit data is , 9

And bit stop bit. The programmable bit (bit data) at the time of transmission is determined by Bit data bit (low bit comes first) In TB8 Software setting Bit programmable bit (bit data)

Load the value PSW / Odd even parity bit in P TB8 (TB8 It can be used as the address data flag in multi-machine communication , and can al

Parity bit) 9 SCON。 The first bit of data is loaded when receiving sending the sending port of the data. ,

Receive/send. RxD For the receiving port, in full duplex mode RB8。 TxD

pattern 2 The baud rate is fixed to the system clock Divider or 32 Frequency division (depends on PCON The value of)

The baud rate calculation formula for serial port mode is shown in the following table (SYSclk For the operating frequency of the system) :

SMOD	Baud rate calculation formula
0	Baud rate = SYSclk / 64
1	Baud rate = SYSclk / 32

2 Patterns and patterns In contrast, except for the slightly different source of baud rate generation, the sending and receiving shift registers are different

In addition, the rest of the functional structure is basically the same, and its reception / The sending operation process and timing are basically

When the receiver receives a frame of information, the following conditions must

be met at the same time: · RI=0

· Data bits or SM2=0 SM2=1 9 RB8=1 · And the received first

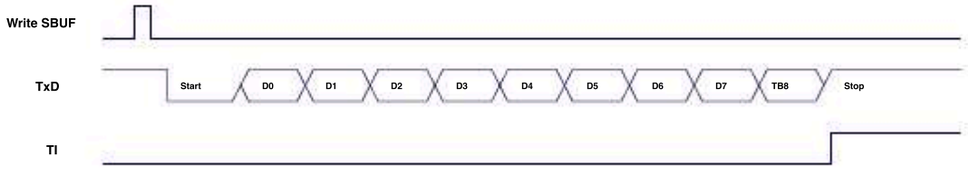
conditions are met at the same time, the data of the received shift register is loaded. and RB8 In , RI The flag is set , 1

And request interrupt processing from the host. If one of the above conditions is not met, the data in the shift register

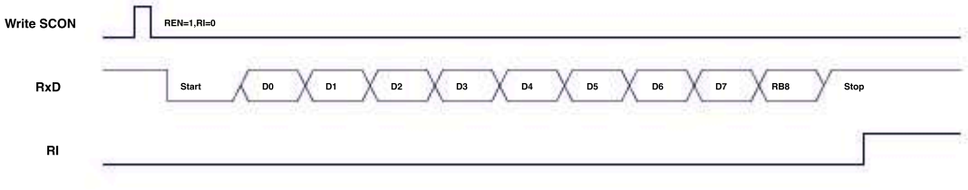
just received is Regardless of whether the above conditions are met or not, the receiver restarts the detection

Enter the information. In the mode, the received stop bits have nothing to do with and. 2 SBUF、 RB8 RI

Through the software pair SM2、 TB8 Settings and communication Tacit agreed agreement provides convenience for multi-machine commu



Send data (serial port 1 mode 2)



Receive data (serial port 1 mode 2)

14.2.8 Serial port mode, mode 3 3 Baud rate calculation formula

when SM0、SM1 The two digits are Work in mode. 3 In serial communication, Asynchronous serial communication of bit data

Mode, the information of one frame is composed of: Bit start bit , , 9

And bit stop bit. The programmable bit (bit data) at the time of transmission is determined by Bit programmable bit (bit data) is provided,

Load the value PSW / Odd even parity bit in P TB8 (TB8 It can be used as the address data flag in multi-machine communication , and it can al

Parity bit) 9 SCON。 The first bit of data is loaded when receiving receiving the sending port of the data. ,

Receive/send. RxD For the receiving port, in full duplex mode RB8。 TxD

pattern 3 And mode 1 Compared to, except when sampling by Supplied by Except for the different data bits, the rest of the functional st

The same, the process and timing of its receiving and sending operations are basically the same.

When the receiver receives a frame of information, the following conditions must be met at the same time: ·RI=0

· Data bits or SM2=0 SM2=1 9 RB8=1 ·And the received first

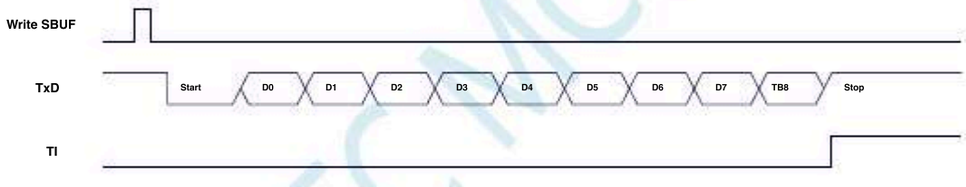
conditions are met at the same time, the data of the received shift register is loaded. and RB8 In , RI The flag is set , 1

And request interrupt processing from the host. If one of the above conditions is not met, the data in the shift register

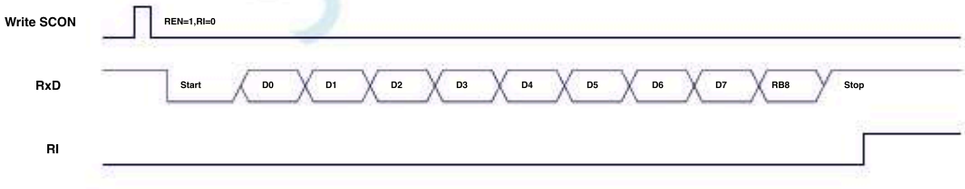
just received is Regardless of whether the above conditions are met, the receiver starts the detection RxD is received.

Enter the information. In the mode, the received stop bits have nothing to do with and. 3 SBUF、RB8 RI

Through the software pair SM2、TB8 The setting and the agreement of the communication protocol provide convenience for multi-



Send data (serial port 1 mode 3)



Receive data (serial port 1 mode 3)

The baud rate calculation formula of the serial port mode is exactly the same as the mode. Please refer to the formula for calculating the

14.2.9 Automatic address recognition

14.2.10 Serial slave address control register (SADDR , SADEN)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR :Slave address register

:Slave address mask bit register SADEN

The automatic address recognition function is typically used in the field of multi-machine communication. Its main principle is that the s
the address information in the data stream from the host serial port. and The slave address of the machine set by the register, the hardware autom
SADEN The slave address is filtered. When the slave address information from the host matches the slave address set by the machine, the hardw
port interrupt; otherwise, the hardware automatically discards the serial port data without interruption. When many slaves in idle mode are lin
the slaves whose slave addresses match will wake up from idle mode, which can greatly reduce the slave cost. The power consumption, even if the slave is in normal c MCU
The state can also avoid constantly entering the serial port to interrupt and reduce the system execution efficiency.

To use the automatic address recognition function of the MCU The serial communication mode is set to mode 2 or
Mode (usually you use mode 1 variable baud rate communication), mode 3 the baud rate of the mode is fixed and it is not easy to adjust), and turn on th
of SM2 SCON bit. For serial port mode 2 or mode 3 9 Bit data (stored in the RB8 / Medium) is the address
The flag of the data, when The bit data is 9 When, it means the previous bit data, the bit data (stored in 9 RB8 / Medium) is the address information. when
the first is set to 1, slave MCU Will automatically filter out non-address data (data with the first digit) The address data in (the first
9 The bit is 1 Data) automatically and SADDR and SADEN The set local address is compared, and if the address matches, it will be set
1" " , And an interrupt is generated, otherwise the serial port data received this time will not be processed.

The setting of the slave address is SADDR and SADEN Two registers are set. SADDR For the slave address register, inside
to store the slave address of the machine, this is the slave address masking bit register, which is used to set the ignore bit in the address informatio SADEN
As follows :

For example

SADDR = 11001010

SADEN = 10000001

The matching address is xxxxxxx0

That is, as long as the address data bit0 for 0 and bit7 for 1 It can match the local address
sent by the host , for example

SADDR = 11001010 Then the
SADEN 00001111 matching
address is xxxx1010

That is, as long as the low in the 4 The bit is 1010 It can match the address of the machine, but the high value is ignored, you c
address data sent by the host is any value.

The host can use the broadcast address (FFH) Select all slaves at the same time to communicate.

14.3 serial port 2

14.3.1 Serial port control register (S2CON)

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

S2SM0 : Specify the serial port 2 The communication working mode is shown in the following table :

S2SM0	serial port 2 Working mode	Function description
0	0 pattern	Variable baud rate 8 Bit data mode , variable
1	1 pattern	baud rate 9 Bit data method

S2SM2 : Allows the serial port 2 to allow multi-machine communication control bits in mode 2 In mode 1, if 1 Bit for bit and 1 S2REN ; The receiver is in the address frame filtering state. At this time, you can use the received first Bit (ie S2RB8) To filter the address frame : If, it means that the frame is an address frame, the address information can be entered S2BUF For, and then interrupt the service make the address number comparison in the program again : if S2RB8 , indicating that the frame is not an address frame, it should be thrown away and the receiver In , if S2SM2 0 The position is and S2REN should be kept in the state where address frame filtering is prohibited. In The bit is Regardless of the received , For or, the received information can be entered 0 1 S2RI=1 , at this time S2BUF , And make S2RB8 Usually it is a parity bit. pattern it a non-multi-machine communication method. In this way, it is necessary to set 0 Should be. S2SM2

S2REN : Allow / Prohibit serial port reception control bit

0 : Prohibit serial port from receiving data

1 : Allow serial port to receive data

S2TB8 : When the serial port is in use mode 1 , S2TB8 For the first to be sent 29 Bit data, generally used as a parity bit or address frame / Data frame mark The log position can be set or cleared by the software as needed. In mode 0 , This bit is not needed. S2RB8 : When the serial port 4 When using the mode , S2RB8 is the received bit of data, generally used as a parity bit or address frame / Data frame mark Ambition. In mode 0 , This bit is not needed. S2TI : Serial port Send an interrupt request flag. When the stop bit starts to be sent, the hardware will automatically Set to 1 , Ask for an interrupt , After responding to the interrupt, the software must be cleared to zero.

S2RI : Serial port S2TI S2RI 1 Set, to CPU hair Receive interrupt request flag. The intermediate time when the stop bit is received serially is automatically sent by the hardware Interrupt request, after responding to the interrupt by the software.

14.3.2 Serial port data register (S2BUF)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

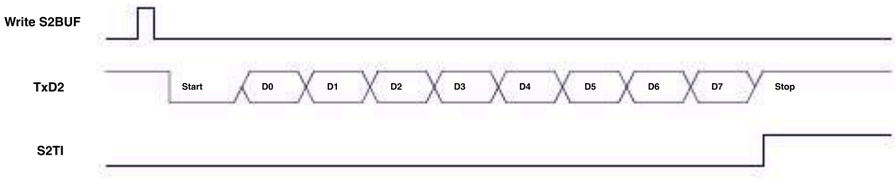
S2BUF : Serial port Data reception / Send buffer. S2BUF 1 Actually is 2 A buffer, a read buffer and a write buffer, two operations are divided into Do not correspond to two different registers , That is to , write only the register (write buffer) , correct 。 S2BUF 1 One is a read-only register (read buffer) to perform a read data. operation, which is actually to read the serial port to receive the buffer, and the other is to trigger the serial port to start send

14.3.3 Serial port mode, mode 0 Baud rate calculation formula 0

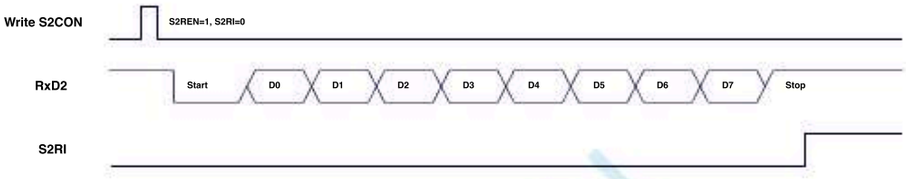
for 8 Bit data bit variable baud rate UART Working mode. A frame of information in this mode is 10 Bit. The starting bit of the bit is

8 Mode serial port 2 Bit stop bit. The baud rate is variable, and the baud rate can be set as needed. TxD2 the data transmission port ,

RxD2 Bit data bit (low bit first) 0 and For the data receiving port, the serial port accepts full duplex send.



Send data (serial port 2 mode 0)



Receive data (serial port 2 mode 0)

The baud rate of the serial port is variable, and its baud rate is generated by the timer. When the timer is used 2 corresponding to 1T (12 Double speed) The speed of the baud rate will also be doubled accordingly. 12

The formula for calculating the baud rate of serial port mode is shown in the following table: 2 For the operating frequency (of the system) SYSclk

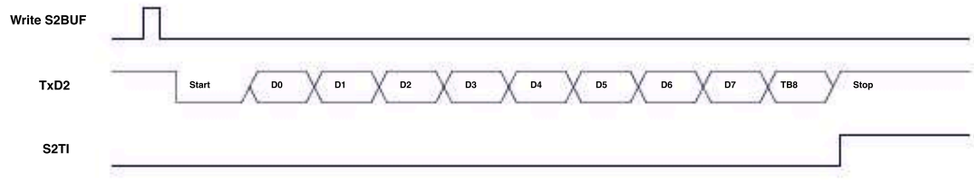
0 Select the timer	timer speed	baud rate calculation formula
Timer2	1T	Timer 2 Overload value = 65536 - SYSclk / (Baud rate × 4)
	12T	Timer 2 Overload value = 65536 - SYSclk / (Baud rate × 12 × 4)

14.3.4 Serial port mode, mode 1 Baud rate calculation formula 1

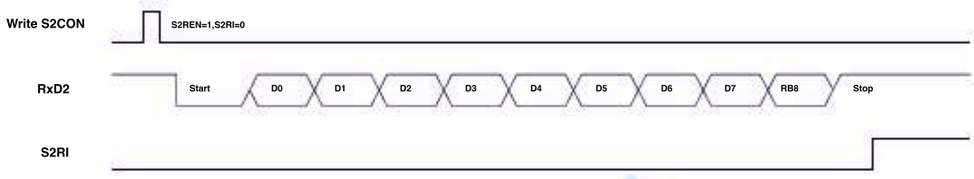
for 9 Bit data bit variable baud rate UART Working mode. A frame of information in this mode is 11 Bit: The starting bit of the bit is

9 Mode serial port 2 Bit stop bit. The baud rate is variable, and the baud rate can be set as needed. TxD2 the data transmission port ,

RxD2 Bit data bit (low bit first) and For the data receiving port, the serial port accepts full duplex send.



Send data (serial port 2 mode 1)



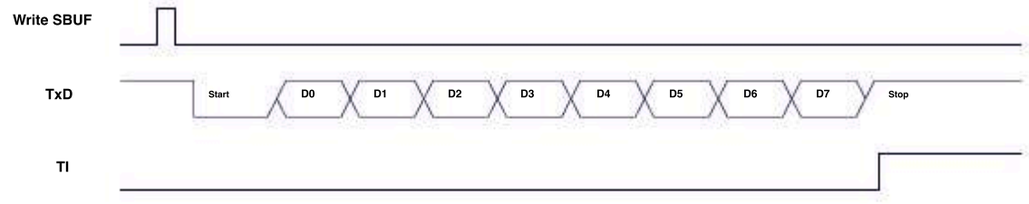
Receive data (serial port 2 mode 1)

The baud rate calculation formula of the serial port mode is exactly the same as the mode. Please refer to the formula for calculating the

14.4 Serial port precautions

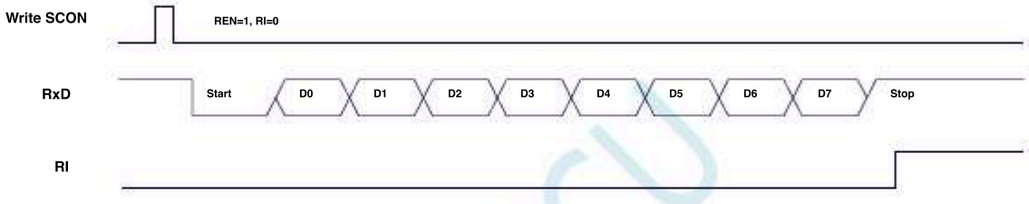
Regarding the serial port interrupt request, there is only one serial port, serial port, serial port, and all are similar, the following serial port issues that need attention: Take an example to illustrate)

8 When the bit data mode is in place, the transmission is completed or the interrupt request is generated after the stop bit is received, as shown in the figure below :



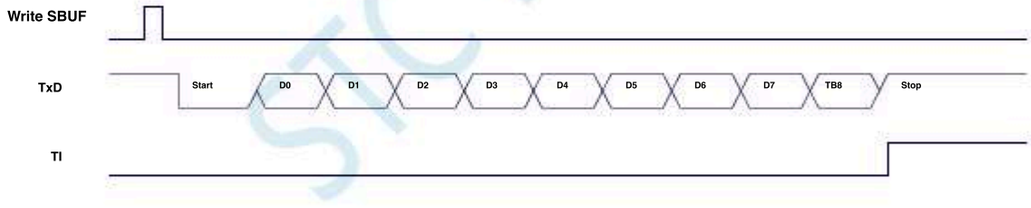
Send data (8-bit data)

8 In bit data mode, it is generated after receiving half of the stop bit. The interrupt request, as shown in the figure below :



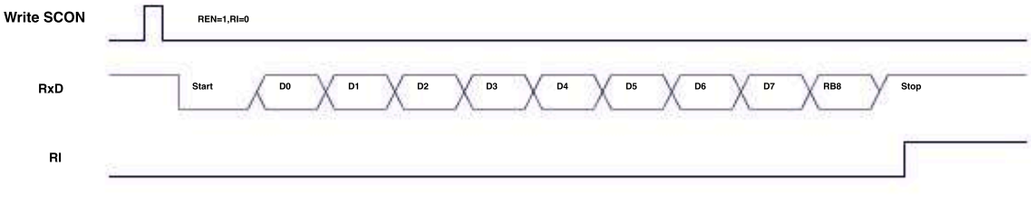
Receive data (8-bit data)

9 When the bit data mode is in place, the transmission is completed or the interrupt request is generated after the stop bit is received :



Send data (9 digits of data)

9 In bit data mode, it is generated after half of the stop bit is received. The interrupt request, as shown in the figure below :



Received data (9 bits of data)

14.5 Sample program

1 Serial port uses timer 2 to make a baud rate generator 14.5.1

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 AUXR = 0x8e;
sfr T2H = 0xd6;
sfr T2L = 0xd7;
sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;
bit busy;
char wptr;
char rptr;
char buffer[16];
void UartIsr() interrupt 4
{
    if (TI)
    {
    TI = 0;
    busy = 0;
    }
    if (RI)
    {
RI = 0;
buffer[wptr++] = SBUF;
wptr &= 0x0f;
    }
}
void UartInit()
{
SCON = 0x50;
T2L = BRT;
T2H = BRT >> 8;
AUXR = 0x15;
```

```
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

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

void UartSendStr(char *p)
{
while (*p)
{
                UartSend(*p++);
        }
}
void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSendStr("Uart Test !\r\n");
        while (1)
        {
                if (rptr ! = wptr)
                {
                        UartSend(buffer[rptr++]);
                        rptr &= 0x0f;
                }
        }
}
```

Assembly code

```
;The test operating frequency is
11.0592MHz

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

BUSY            BIT         20H. 0
```

```
WPTR            DATA    21H
RPTR            DATA    22H
BUFFER          DATA    23H             ;16 bytes

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     0023H
                LJMP    UART_ISR

                ORG     0100H

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

                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,SBUF
                INC     WPTR
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#15H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
```

```
                MOV     SBUF,A
                RET

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

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

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

                END
```

14.5.2 The serial port uses a timer (mode) as a baud rate generator 1 0

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 AUXR = 0x8e;

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

bit busy;

char wptr;

char rptr;

char buffer[16];

void UartIsr() interrupt 4

{

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

void UartInit()

{

    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;

}

void UartSend(char dat)

{
```

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

void UartSendStr(char *p)
{
while (*p)
{
                UartSend(*p++);
        }
}
void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSendStr("Uart Test !\r\n");
        while (1)
        {
                if (rptr ! = wptr)
                {
                        UartSend(buffer[rptr++]);
                        rptr &= 0x0f;
                }
        }
}
```

Assembly code

```
;The test operating frequency is
                11.0592MHz

AUXR            DATA    8EH

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

P0M1            DATA    093H
P0M0            DATA    094H
P1M1            DATA    091H
P1M0            DATA    092H
P2M1            DATA    095H
```

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

                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    UART_ISR

                ORG     0100H

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

                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,SBUF
                INC     WPTR
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

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

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
```

```
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

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

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

                END
```

14.5.3 The serial port uses a timer (mode) as a baud rate generator 1 2

C Language code

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

#include "reg51.h"
#include "intrins.h"
#define FOSC 11059200UL
```

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

sfr		AUXR		=	0x8e;

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

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

void UartIsr() interrupt 4
{
	if (TI)
	{
	TI = 0;
	busy = 0;
	}
	if (RI)
	{
RI = 0;
buffer[wptr++] = SBUF;
wptr &= 0x0f;
	}
}
void UartInit()
{
	SCON = 0x50;
	TMOD = 0x20;
	TL1 = BRT;
	TH1 = BRT;
	TR1 = 1;
	AUXR = 0x40;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

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

void UartSendStr(char *p)
{
```

```
while (*p)
{
        UartSend(*p++);
    }
}
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");
    while (1)
    {
        if (rptr ! = wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR    DATA    8EH

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

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

```
                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    UART_ISR

                ORG     0100H

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

                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,SBUF
                INC     WPTR
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#20H
                MOV     TL1,#0FDH               ;256-11059200/115200/32=0FDH
                MOV     TH1,#0FDH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

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

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

MAIN:
```

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

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

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

                END
```

2 Serial port uses timer 2 1 Make a baud rate generator 14.5.4

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

```
sfr         IE2         =   0xaf;


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


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

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
    S2CON &= ~0x02;
    busy = 0;
    }
    if (S2CON & 0x01)
    {
S2CON &= ~0x01;
buffer[wptr++] = S2BUF;
wptr &= 0x0f;
    }
}
void Uart2Init()
{

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


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


void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2Send(*p++);
    }
```

```
}

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

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SendStr("Uart Test !\r\n");
    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart2Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S2CON   DATA    9AH
S2BUF   DATA    9BH
IE2     DATA    0AFH

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

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

```
P5M0            DATA    0CAH

                ORG     0000H
                LJMP    MAIN
                ORG     0043H
                LJMP    UART2_ISR

                ORG     0100H

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

                MOV     A,S2CON
                JNB     ACC. 1,CHKRI
                ANL     S2CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC. 0,UART2ISR_EXIT
                ANL     S2CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S2BUF
                INC     WPTR
UART2ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

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

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

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

MAIN:
```

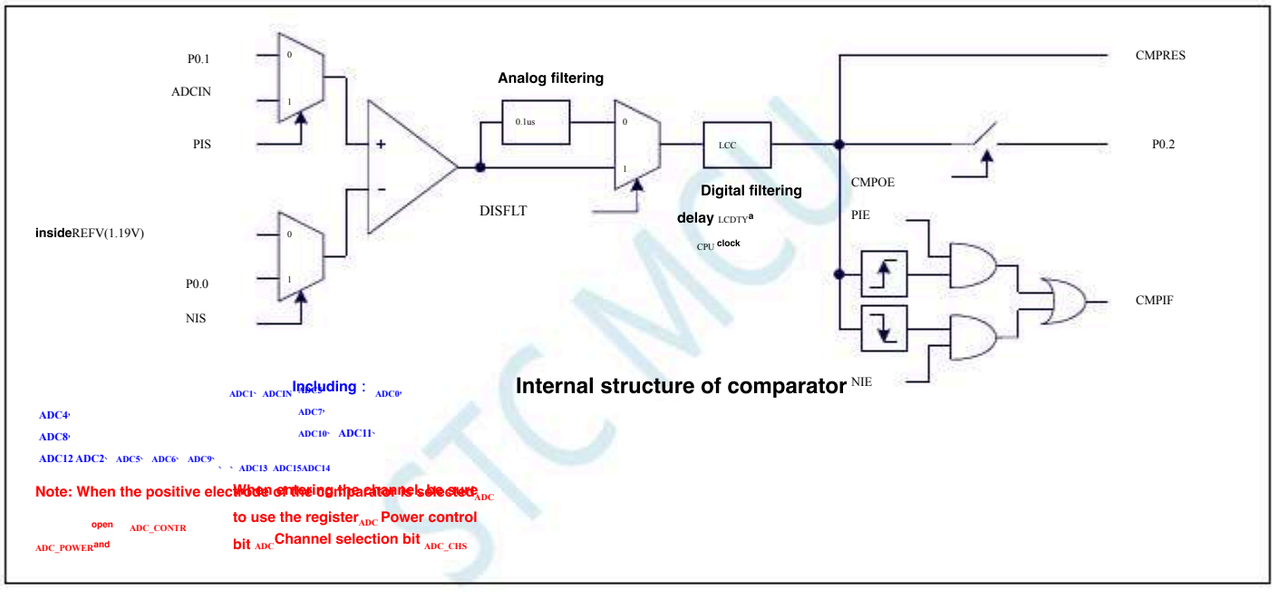
```
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV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

                LCALL   UART2_INIT
                MOV     IE2,#01H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART2_SENDSTR

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

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

                END
```

15 Comparator, power-down detection, internal 1.19V Reference signal source

STC12H A comparator is integrated inside the series of microcontrollers. The positive electrode of the comparator can be Analog input Channel, while the negative electrode can Port or internal BandGap After passing by After REFV Voltage (internal fixed comparison voltage) Multiplexer and time-sharing multiplexing can realize the application of multiple comparators。

There are two stages of filtering that can be programmable inside the comparator: analog filtering and digital filtering. Analog filtering ca the comparison input signal , and digital filtering can wait for the input signal to be more stable before comparing. The comparison result can reading the internal register bits, or the comparator result can be output forward or reverse to an external port. The output of the comparison be used as a trigger signal and feedback signal for external events, which can expand the scope of application of the comparison.

15.1 Internal structure diagram of comparator



15.2 Comparator-related registers

symbol	description	address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	Comparator control register 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	Comparator control register 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000

15.2.1 Comparator control register (CMPCR1)

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

: Enable bit of comparator module CMPEN

0: Turn off the comparison function

1: Enable the comparison function

: Comparator interrupt flag. when NIE or CMPIF NIE After being enabled, if a corresponding interrupt signal is generated, CMPIF set the hardware will au

Parallel to CPU Make an interrupt request. This flag must be cleared by the user software.

(Note: When the comparator interrupt is not enabled, the hardware will not set this interrupt flag, that is, when the comparator is accessed by query , this interrupt flag cannot be queried)

PIE : The rising edge of the comparator interrupts the enable bit.

0 : Prohibit the rising edge of the comparator from interrupting.

1 : Enable the rising edge interrupt of the comparator. The comparison result of the comparator is An interrupt request is generated when it becomes.

NIE comparator is determined by : the falling edge of the comparator interrupts the enable bit.

0 : Disable the falling edge interrupt of the comparator.

1 : Enable the falling edge interrupt of the comparator. The comparison result of the comparator is from 1 become 0 When an interrupt request is generated.

PIS comparator is determined by : the positive electrode of the comparator selects the bit

0: Select an external port P5.5 Is the positive input source of the comparator.

1: Pass ADC_CONTR in The analog input terminal of the comparator is used as the positive input

(Note: When the positive electrode of the comparator is selected ADC, Please be sure to open the register in ADC_CONTR ADC electricity

Source control bit ADC_POWER and ADC Channel selection bit ADC_CHS)

(Note: When you need to use the comparator to interrupt the wake-up and power-down mode, When the clock stops vibration mode, the can't

Input channel) use ADC

NIS : The negative electrode of the comparator is selected

BandGap 0: Select internal After the voltage passes through REFV As the negative input source of the comparator. (When the chip is out of the fac

The test signal source is 1.19V OP)

adjusted to 1: Select an external port P0.0

Is the negative input source of the comparator.

: Comparator result output control bit CMPOE

0 : Disable the output of comparator results

1 : Enable the output of the comparator result. The comparator result is output to P0.2

:The comparison result of the comparator. This bit is read-only. CMPRES

CMP+ The level is lower than CMP- The level of

Indicates 1: Indicates CMP- The level of

CMPRES CMP+ The level is higher than It is the output signal after digital filtering, not the direct output of the comparator.

15.2.2 Comparator control register (2 CMPCR2)

symbol	address	B7	B6	B5	B4		B2 B3	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

: Output control of comparator result INVCMPO

0 : The comparator result is output forward. If 0 For, then P0.2 output Low level, and vice versa, the output is high level.

1 : The comparator result is output in reverse. if 0 For, then P0.2 The output is high, and vice versa, the output is low.

: Analog filter function control DISFLT

0 : Enable 0.1us Analog filtering function

1 : Closed 0.1us The analog filtering function can slightly increase the comparison speed of the comparator.

LCDTY[5:0] : Digital filtering function control

The digital filtering function is the digital signal de-jitter function. When the comparison result changes on the rising or falling edge, the

If the transformed signal must be maintained LCDTY Set by CPU If the number of clocks does not change, the data change is considered valid;

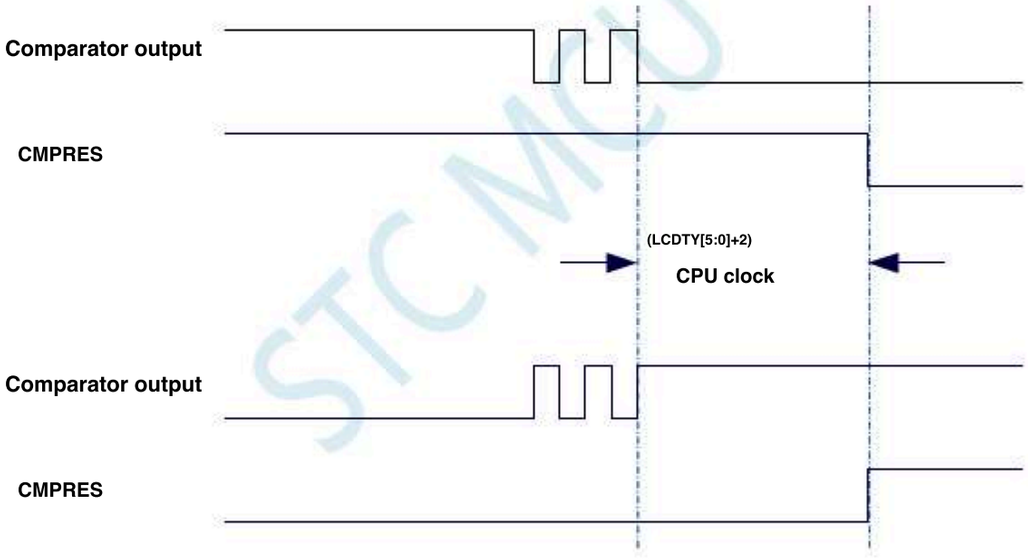
, it will be regarded as no change in the signal.

Note: When the digital filtering function is enabled, the actual waiting clock inside the chip needs to increase the switching time of two a

When the digital filtering function is turned off, Set to non- 0 n Value (n=1~63) When the actual

Set to LCDTY

0 The digital filtering time is (n+2) A system clock



15.3 Sample program

15.3.1 Use of comparator (interrupt mode)

C Language code

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

#include "reg51. h"
#include "intrins. h"
sfr
        CMPCR1          =       0xe6;
sfr CMPCR2
                        =       0xe7;
sfr P1M1
sfr P1M0
                        =       0x91;
sfr P0M1                =       0x92;
sfr P0M0                =       0x93;
sfr P2M1                =       0x94;
sfr P2M0                =       0x95;
                        =       0x96;
sfr P3M1
                        =       0xb1;
sfr P3M0                =       0xb2;
sfr P4M1                =       0xb3;
sfr P4M0                =       0xb4;
sfr P5M1                =       0xc9;
sfr P5M0                =       0xca;
sbit P10
sbit P11                =       P1^0;
                        =       P1^1;


void CMP_Isr() interrupt 21
{
        CMPCR1 &= ~0x40;                        //Clear interrupt sign
        if (CMPCR1 & 0x01)
        {
                P10 = ! P10;                    // Falling edge interrupt test port
}
else
{
                P11 = ! P11;                    // Rising edge interrupt test port
        }
}
void main()
{
P0M0 = 0x00;
P0M1 = 0x00;
P1M0 = 0x00;
P1M1 = 0x00;
P2M0 = 0x00;
P2M1 = 0x00;
P3M0 = 0x00;
P3M1 = 0x00;
P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;
```

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


        EA = 1;


        while (1);
}
```

//Comparator forward output
//Comparator reverse output
//Enable filter 0.1us
//forbidden filter 0.1us
// Direct output of comparator results
//The comparator result has passed Output after a debounce clock

// Enable comparator edge interrupt
// disable comparator rising edge interrupt
// Enable comparator rising edge interrupt
// Disable the falling edge of the comparator from interrupting
// Enable comparator falling edge interrupt
//P0.1 for CMP+ Input pin
//ADC input pin is CMP+ The reference
//inside 1.19V signal source of the input pin is CMP+
for //P0.0 Input pin CMP-
//Disable comparator output
//Enable comparator output
//Enable comparator module

Assembly code

```
;The test operating frequency is 11.0592MHz

CMPCR1          DATA        0E6H
CMPCR2          DATA        0E7H

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

                ORG         0000H
                LJMP        MAIN
                ORG         00ABH
                LJMP        CMPISR

                ORG         0100H
CMPISR:
                PUSH        ACC
                ANL         CMPCR1,#NOT 40H
                MOV         A,CMPCR1
                JB          ACC. 0,RSING
FALLING:
```

;Clear interrupt sign

```
                CPL     P1.0            ; Falling edge interrupt test port
                POP     ACC
                RETI
RSING:
                CPL     P1.1            ; Rising edge interrupt test port
                POP     ACC
                RETI

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

                MOV     CMPCR2,#00H
                ANL     CMPCR2,#NOT 80H ;Comparator forward output
;               ORL     CMPCR2,#80H     ;Comparator reverse output
                ANL     CMPCR2,#NOT 40H ;Enable filter 0.1us
;               ORL     CMPCR2,#40H     ;forbidden filter 0.1us
;               ANL     CMPCR2,#NOT 3FH ; Direct output of comparator results
                ORL     CMPCR2,#10H     ;The comparator result is output after a debounce clock
                MOV     CMPCR1,#00H     ;Disable comparator edge interrupt
                ORL     CMPCR1,#30H     ;
                ANL     CMPCR1,#NOT 20H ; Disable the rising edge of the comparator from interrupting
;               ORL     CMPCR1,#20H     ; Enable comparator rising edge interrupt
;               ANL     CMPCR1,#NOT 10H ; Disable the falling edge of the comparator from interrupting
;               ORL     CMPCR1,#10H     ; Enable comparator falling edge interrupt
                ANL     CMPCR1,#NOT 08H ;P0.1 for CMP+ Input pin
                ORL     CMPCR1,#08H     ;ADC input pin is CMP+ The reference
;               ANL     CMPCR1,#NOT 04H ;inside 1.19V signal source of the input pin is
;               ORL     CMPCR1,#04H     ;CMP- The input pin
                ANL     CMPCR1,#NOT 02H ;       for CMP- Input pin
;               ORL     CMPCR1,#02H     ;;P0.0 Disable comparator output
                ORL     CMPCR1,#80H     ;Enable comparator output
                SETB    EA              ;Enable comparator module


                JMP     $

                END
```

15.3.2 Use of comparator (query method)

C Language code

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

#include "reg51.h"
```

```
#include "intrins. h"

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

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

sbit        P10             =       P1^0;
sbit        P11             =       P1^1;

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;
        CMPCR2 = 0x00;

        CMPCR2 &= ~0x80;                //Comparator forward output
//      CMPCR2 |= 0x80;
        CMPCR2 &= ~0x40;                //Comparator reverse output
//      CMPCR2 |= 0x40;
//      CMPCR2 &= ~0x3f;                //Enable filter 0.1us
        CMPCR2 |= 0x10;                 //forbidden filter 0.1us
        CMPCR1 = 0x00;
        CMPCR1 |= 0x30;                 // Direct output of comparator results
//      CMPCR1 &= ~0x20;                //The comparator result has passed through a debounce clock
//      CMPCR1 |= 0x20;
//      CMPCR1 &= ~0x10;
//      CMPCR1 |= 0x10;
        CMPCR1 &= ~0x08;                // Enable comparator edge interrupt
        CMPCR1 |= 0x08;                 // disable comparator rising edge interrupt
//      CMPCR1 &= ~0x04;                // Enable comparator rising edge interrupt
//      CMPCR1 |= 0x04;                 // Disable the falling edge of the comparator from interrupting
        CMPCR1 &= ~0x02;                // Enable comparator falling edge interrupt
//      CMPCR1 |= 0x02;                 //P0.1 for CMP+ Input pin
        CMPCR1 |= 0x80;                 //ADC input pin is CMP+ The reference
                                        //inside 1.19V signal source of the input pin is CMP+ for //P0.0 Input pin
                                        //Disable comparator output
                                        //Enable comparator output
                                        //Enable comparator module

        while (1)
{
```

```
        P10 = CMPCR1 & 0x01;
    }
}
```

Assembly code

```
;The test operating frequency is 11.0592MHz

CMPCR1  DATA    0E6H
CMPCR2  DATA    0E7H

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

        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     CMPCR2,#00H
        ANL     CMPCR2,#NOT 80H
;       ORL     CMPCR2,#80H
        ANL     CMPCR2,#NOT 40H
;       ORL     CMPCR2,#40H
;       ANL     CMPCR2,#NOT 3FH
        ORL     CMPCR2,#10H
        MOV     CMPCR1,#00H
        ORL     CMPCR1,#30H
        ANL     CMPCR1,#NOT 20H
;       ORL     CMPCR1,#20H
;       ANL     CMPCR1,#NOT 10H
;       ORL     CMPCR1,#10H
;       ANL     CMPCR1,#NOT 08H
        ORL     CMPCR1,#08H
;       ANL     CMPCR1,#NOT 04H
;
```

// Read the comparison result of the comparator

;Comparator forward output

;Comparator reverse output

;Enable filter 0.1us

;forbidden filter 0.1us

; Direct output of comparator results

;The comparator result is passed through a debounce clock f_0

Enable comparator edge interrupt

;

; Disable the rising edge of the comparator from interrupting

; Enable comparator rising edge interrupt

; Disable the falling edge of the comparator from interrupting

; Enable comparator falling edge interrupt

;P0.1 for $CMP+$ Input pin

;ADC input pin is $CMP+$ The reference

;inside 1.19V signal source of the input pin is $CMP-$

```
                ORL     CMPCR1,#04H             ; for CMP- Input pin
;               ANL     CMPCR1,#NOT 02H         ;;P0.0 Disable comparator output
                ORL     CMPCR1,#02H             ; Enable comparator output
                ORL     CMPCR1,#80H             ; Enable comparator module

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

                END
```

15.3.3 Multiplexing application of comparator (comparator+ADC Input channel)

Due to the positive electrode of the comparator, you can choose ADC Analog input channel, so multiple ratios can be achieved through multiple selectors and time-sharin

The application of the comparator.

Note: When the positive electrode of the comparator is selected When operating the channel, please be sure to turn it on in the register ADC_CONTR ADC Power control bit ADC_POWER and ADC channelSelect bit ADC_CHS

Language codeC

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

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

sfr CMPCR1  = 0xe6;
sfr CMPCR2  = 0xe7;
sfr ADC_CONTR = 0xbc;
sfr P1M1  = 0x91;
sfr P1M0  = 0x92;
sfr P0M1  = 0x93;
sfr P0M0  = 0x94;
sfr P2M1  = 0x95;
sfr P2M0  = 0x96;
sfr P3M1  = 0xb1;
sfr P3M0  = 0xb2;
sfr P4M1  = 0xb3;
sfr P4M0  = 0xb4;
sfr P5M1  = 0xc9;
sfr P5M0  = 0xca;
sbit P10  = P1^0;
sbit P11  = P1^1;

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

```
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
    P1M0 &= 0xfe;                   //Set up P1.0 For the input port
    P1M1 |= 0x01;
    ADC_CONTR = 0x80;               //Enable ADC Module and select ADC0 for ADC Input pin

    CMPCR2 = 0x00;
    CMPCR1 = 0x00;
    CMPCR1 |= 0x08;                 //The input pin CMP+ is for Input pin
    CMPCR1 |= 0x04;                 //P0.6 is for CMP- Input pin
    CMPCR1 |= 0x02;                 //Enable comparator output
    CMPCR1 |= 0x80;                 //Enable comparator module
    while (1);
}
```

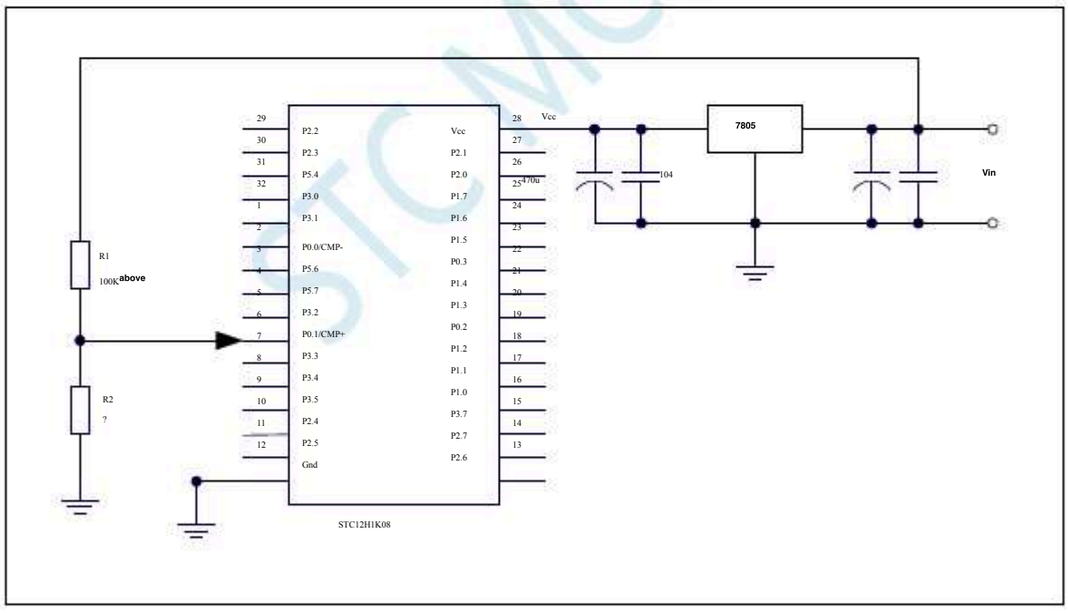
Assembly code

```
;The test operating frequency is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN

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

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        ANL     P1M0,#0FEH      ;Set up P1.0 For the input port
        ORL     P1M1,#01H
        MOV     ADC_CONTR,#80H  ;Enable ADC Module and select for ADC Input pin

        MOV     CMPCR2,#00H
        MOV     CMPCR1,#00H

        ORL     CMPCR1,#08H     ;The input pin CMP+ Input pin
        ORL     CMPCR1,#04H     ;P0.1 is for CMP- Input pin
        ORL     CMPCR1,#02H     ;Enable comparator output
        ORL     CMPCR1,#80H     ;Enable comparator module

LOOP:
        JMP     LOOP

        END
```

15.3.4 The comparator is used for external power-down detection (During the power-down process, user da

EEPROM In)



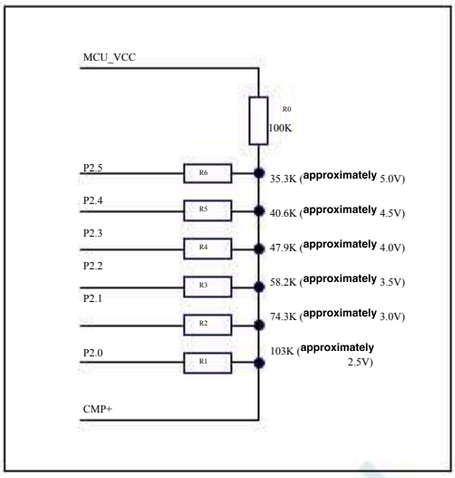
The resistor input and Pair the regulator Block and voltage of the voltage divider is divided, and the voltage after the voltage divi
internal in the figure above Refer to the signal source for comparison.

Generally when the at When the voltage is regulator block The DC voltage at the front end is When the voltage is stabilize
block 7805 The DC voltage at the When the regulator block The DC voltage at the front end is lower than or equal to The DC voltage at the front end input is straight
The current voltage is resistive and R2 Voltage divider to the positive input of the comparator Terminal input voltage is lower than internal Reference signal source ,
At this time, a comparator interrupt can be generated, so that during the power-down detection, When the regulator block time to save
the data until the DC voltage at the front end is higher than that, the DC voltage input at the front end is resistive and Voltage divider to the positive input of the comparator
The terminal input voltage is high Refer to the signal source, at this time Continue to work normally. CPU

inside 1.19V The reference signal source is internal After passing by After the voltage (When the chip is out of the factory, the internal referenc

The source is 1.19V) 。 The specific value is obtained by reading the reference signal source is inside District of 1.19V RAM Program memory adjusted to value of the address occupied by the internal buffer ROM Series, the internal reference signal source value is stored in RAM Flash programs Flash Memory (ROM) For the storage address, please refer to "Special parameters in memory" Chapter

15.3.5 The comparator detects the operating voltage (battery voltage)



In the figure above, the low level can be roughly measured using the operating voltage (The comparator detects the voltage (strobe channel , of IO Export voltage divider , and the port voltage value is close to Gnd , Channels that are not strobeated IO The high open-drain mode of the port output does not affect other

The negative end of the comparator Reference signal source, the positive terminal is selected through the resistor voltage divider and in CMP+ The voltage value of the pin

The ports are set to open-drain mode and the output is high When initializing, this is Low voltage in 2.5V The comparison value of the comparator is, on the contrary, the voltage is higher than the comparator is ; 1 2.5V

If determined Vcc Higher than High port output , The output of the port is low, at this time, if the voltage is lower than Vcc 3.0V Then compare then the The comparison value of the device is, and vice versa The comparison value of the comparator is ;

If determined Vcc Higher than 3.0V, then the The voltage is higher than Vcc Voltage is lower than 3.5V Then compare The comparison value of the device is, and vice versa The comparison value of the comparator is ;

If determined Vcc Higher than 3.5V, then the The output of the port is low, at this time if Vcc Voltage is lower than 4.0V Then compare The comparison value of the device is, and vice versa The comparison value of the comparator is ;

If determined Vcc Higher than 4.0V, then the The output of the port is low, at this time if Vcc Voltage is lower than 4.5V Then compare The comparison value of the device is, and vice versa The comparison value of the comparator is ;

If determined Vcc Higher than 4.5V, then the The output of the port is low, at this time if Vcc Voltage is lower than 5.0V Then compare The comparison value of the device is, and vice versa The comparison value of the comparator is. 1

The output of the port is low, at this time if

Language code C

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

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

sfr CMPCR1 = 0xe6;
sfr CMPCR2 = 0xe7;
sfr P1M1 = 0x91;
```