STC micro™

STC8A8K64D4 series of Microcontrollers Reference Manual

Date of update: **2022/3/1**

Contents

1 Overview

STC8A8K64D4 series of microcontrollers do not require an external crystal oscillator and external reset circuit. They are 8051 microcontrollers with the properties of strong anti-interference/ultra low price/high speed/low power consumption. Under the same operating frequency, STC8A8K64D4 series of microcontrollers are about 12 times faster (11.2 ~ 13.2 times) than traditional 8051. To execute all 111 instructions in sequence, the STC8A8K64D4 series microcontrollers only need 147 clocks, while the traditional 8051 requires 1944 clocks. STC8A8K64D4 series of microcontrollers are single clock/machine cycle (1T) microcontrollers produced by STC. They are new generation 8051 microcontrollers with wide voltage/high speed / high reliability / low power consumption / strong antistatic / strong anti-interference, and is super encrypted. The instruction codes are fully compatible with traditional 8051.

High precision of ±0.3% @+25°C RC clock is integrated in MCU with −1.38% to +1.42% temperature drift under the temperature range of -40°C to +85°C, and -0.88% to +1.05% temperature drift under temperature range from -20°C to +65°C. The frequency of RC clock can be set from 4MHz to 35MHz when programming a MCU using ISP. Note: The maximum frequency must be controlled below 45MHz when the temperature range is -40°C to + 85°C. Moreover, high reliable reset circuit is integrated in MCU with 4 levels optional reset threshold voltages, which can be selected when user programming using ISP. So, external expensive crystal and the external reset circuit can be eliminated completely.

There are three optional clock sources inside the MCU, internal high precision IRC which can be adjusted appropriately, internal 32KHz low speed IRC, external 4MHz~33MHz oscillator or external clock signal. The clock source can be freely chosen in user codes. After the clock source is selected, it may be 8-bit divided and then be supplied to the CPU and the peripherals, such as timers, UARTs, SPI, and so on.

Two low power modes are provided in MCU, the IDLE mode and the STOP mode. In IDLE mode, MCU stops clocking CPU, CPU stops executing instructions without clock, while all peripherals are still working. At this moment, the power consumption is about 1.0mA at 6MHz working frequency. The STOP mode is the power off or power-down mode. At this 1oment, the main clock stops, CPU and all peripherals stop working, and the power consumption can be reduced to about 0.6uA when VCC is 5.0V, 0.4uA when VCC is 3.3V.

The Power-down mode can be woke-up by one of the following interrupts: INT0(P3.2), INT1(P3.3), INT2(P3.6), INT3(P3.7), INT4(P3.0), T0(P3.4), T1(P3.5), T2(P1.2), T3(P0.4), T4(P0.6), RXD(P3.0/P3.6/P1.6/P4.3), RXD2(P1.0/P4.6), RXD3(P0.0/P5.0), RXD4(P0.2/P5.2), CCP0(P1.7/P2.3/P7.0/P3.3), CCP1(P1.6/P2.4/P7.1/P3.2), CCP2(P1.5/P2.5/P7.2/P3.1), CCP3(P1.4/P2.6/P7.3/P3.0), I2C_SDA(P1.4/P2.4/P3.3), SPI_SS(P1.2/P2.2/P3.5) , all I/O interrupts, Comparator, LVD, Power-down wake-up timer.

Rich digital peripherals and analog peripherals are provided in MCU, including UARTs, timers, PCA, enhanced PWMs and I2C, SPI, 12-bit*15 ultra-high-speed ADC with a speed of up to 800K, that is, 800,000 samples per second, and comparator, which can meet the requirements of users when designing a product.

The enhanced dual data pointers are integrated in the STC8A8K64D4 series of microcontrollers. Using user codes, the function of automatic increasing or decreasing of data pointer and automatic switching of two sets of data pointers can be realized.

Products Line	I/O	UART	Timers	ADC	Enhanced PWM	PCA	CMP	SPI	I2C	MDU16	I/O Int.	LCM	DMA
STC8A8K64D4 series-64Pin/48Pin	59	4	5	15_{CH}*12_{B}	●	●	●	●	●	●	●	●	●

2 Features, Price and Pins

2.1 STC8A8K64D4-64Pin/48Pin family

2.1.1 Features and Price(Quasi 16-bit MCU with 16-bit hardware multiplier and divider MDU16)

➢ **Selection and price (No external crystal and external reset required with 9 channels 10-bit ADC)**

MCU	Operating voltage（V）	Flash Code Memory (100 thousand times) (Byte)	xdata Internal extended SRAM (Byte)	Enhanced Dual DPTR increasing or decreasing	EEPROM 100 thousand times) (Byte)	Maximum I/O Lines	UARTs which can wake-up CPU	MDU16 (Hardware 16-bit Multiplier and Divider)	8080/6800 interface/ LCM driver(8-bit and 16-bit)	DMA (supports ADC, UART, SPI, LCD etc.)	SPI	I²C	All I/O ports support interrupts and can wake up MCU	Timers/Counters (T0-T4 Pin can wake-up CPU)	15-bit enhanced PWM meets stage lighting requirements	PCA/CCP/PWM (can be used as external interrupt and wake-up after power-	Power-down Wake-up timer	15 channels high speed ADC (8 PWMs can be used as 8 DACs)	Comparator (May be used as a ADC to detect external power-down)	Internal LVD interrupt (can wake-up CPU)	Watch-dog Timer	Internal high reliable reset circuit with 4 levels optional reset threshold voltage	Internal high precision Clock (adjustbal under 24MHz)	Clock output and Reset	Program encrypted transmission (Anti-blocking)	Password can be set for next update	Support RS485 download	Support software USB download directly	Online debug itself	Price & Package						products supply information
																														LQFP64	QFN64 <8x8mm>	LQFP48	QFN48 <6x6mm>	LQFP44	PDIP40	
STC8A8K16D4	1.9-5.5	16K	8K	2	48K	59	4	Y	Y	Y	Y	Y	Y	5	8	4	Y	12 bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	√	√	√		Available
STC8A8K32D4	1.9-5.5	32K	8K	2	32K	59	4	Y	Y	Y	Y	Y	Y	5	8	4	Y	12 bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	¥4.5	√	¥4.2	√	¥4.2		
STC8A8K48D4	1.9-5.5	48K	8K	2	16K	59	4	Y	Y	Y	Y	Y	Y	5	8	4	Y	12 bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	¥4.5	√	¥4.2	√	¥4.2		
STC8A8K60D4	1.9-5.5	60K	8K	2	4K	59	4	Y	Y	Y	Y	Y	Y	5	8	4	Y	12 bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	√	√	√	√	√		
STC8A8K64D4	1.9-5.5	64K	8K	2	IAP	59	4	Y	Y	Y	Y	Y	Y	5	8	4	Y	12 bit	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	¥4.5	√	¥4.2	√	¥4.2		

➢ **Core**
- ✓ Ultra-high speed 8051 Core with single clock per machine cycle, which is called 1T, and the speed is about 12 times faster than traditional 8051
- ✓ Fully compatible instruction set with traditional 8051
- ✓ 43 interrupt sources and 4 interrupt priority levels
- ✓ Online debugging is supported

➢ **Operating voltage**
- ✓ 1.9V～5.5V
- ✓ Built-in LDO

➢ **Operating temperature**
- ✓ -40℃~85℃(Please refer to the description of the electrical characteristics chapter for applications beyond the temperature range)

➢ **Flash memory**
- ✓ Up to 64Kbytes of Flash memory to be used for storing user code
- ✓ Configurable size EEPROM, 512bytes single page for being erased, which can be repeatedly erased more than 100 thousand times.
- ✓ In-System-Programming, ISP in short, can be used to update the application code. No dedicated programmer is needed.

- ✓ Online debugging with single chip is supported, and no dedicated emulator is needed. The number of breakpoints is unlimited theoratically.

➢ **SRAM**
- ✓ 128 bytes internal direct access RAM (DATA, use keyword *data* to declare in C language program)
- ✓ 128 bytes internal indirect access RAM (IDATA, use keyword *idata* to declare in C language program)
- ✓ 8192 bytes internal extended RAM (internal XDATA, use keyword *xdata* to declare in C language program)

➢ **Clock**
- ✓ Internal high precise RC clock(IRC for short, ranges from 4MHz to 45MHz), adjustable while ISP and can be divided to lower frequency by user software, 100KHz for instance.
  - ✓ Error:±0.3% (at the temperature 25℃)
  - ✓ -1.38%～+1.42% temperature drift (at the temperature range of -40°C to +85°C)
  - ✓ -0.88%～+1.05% temperature drift (at the temperature range of -20℃ to 65℃)
- ✓ Internal 32KHz low speed IRC with large error
- ✓ External 4MHz~45MHz oscillator or external clock

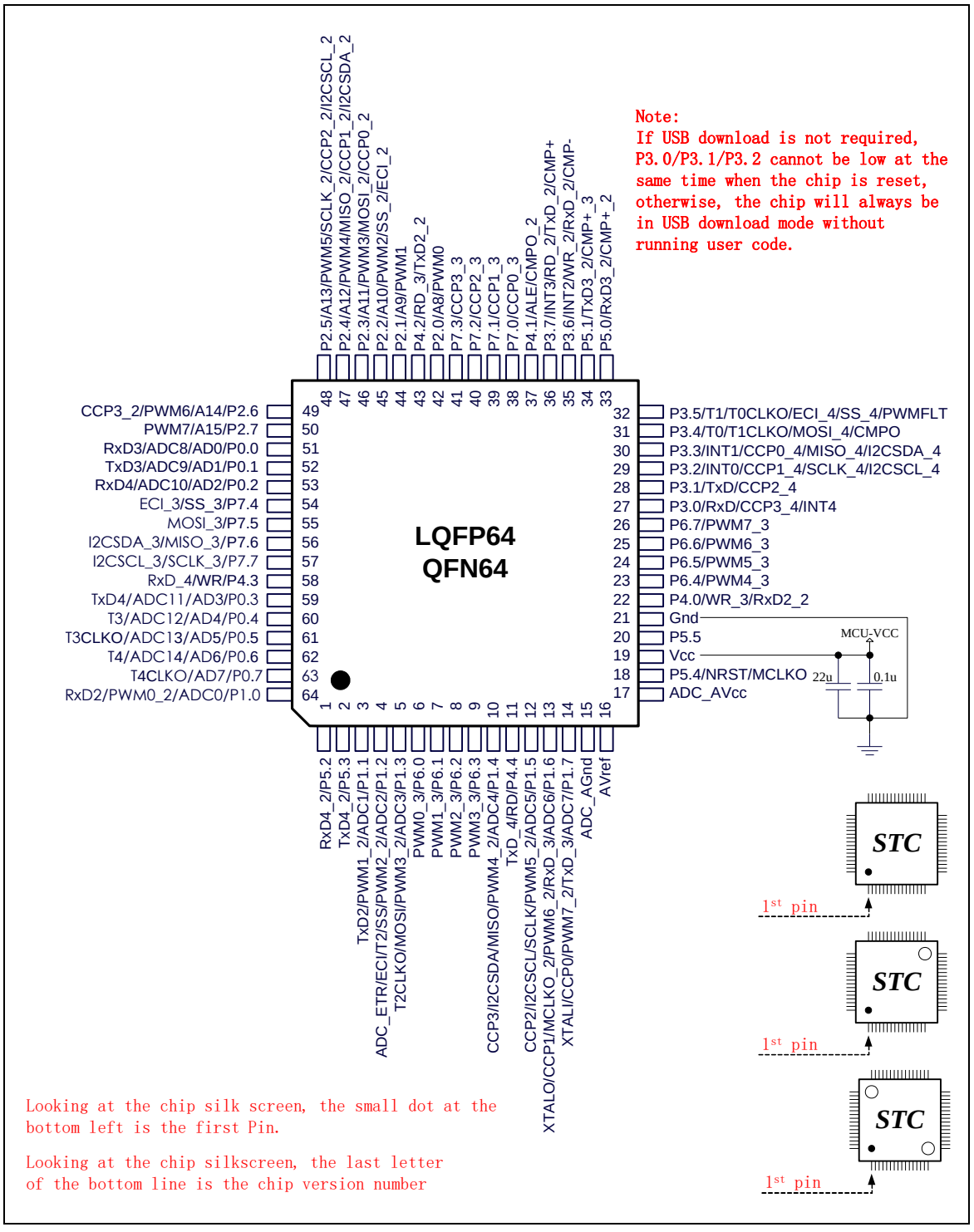
➢ **Reset**
- ✓ Hardware reset
  - ✓ Power-on reset. Measured voltage value is 1.69V~1.82V.
    (**Effective when the chip does not enable the low voltage reset function**)
    The power-on reset voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of the power-on reset, the chip is in reset state. When the voltage rises from 0V to the upper threshold voltage of power-on reset, the chip is released from the reset state.
  - ✓ Reset by reset pin. The default function of P5.4 is the I/O port. P5.4 pin can be set as the reset pin while ISP download. (**Note: When the P5.4 pin is set as the reset pin, the reset level is low**.)
  - ✓ Watch dog timer reset
  - ✓ Low voltage detection reset. 4 low voltage detection levels are provided, 2.0V (Measured as 1.90V~2.04V), 2.4V (Measured as 2.30V~2.50V), 2.7V(Measured as 2.61V~2.82V), 3.0V(Measured as 2.90V~3.13V). Each level of low-voltage detection voltage is a voltage range consisting of an upper limit voltage and a lower limit voltage. When the operating voltage drops from 5V / 3.3V to the lower limit threshold voltage of low-voltage detection, the low-voltage detection takes effect. When the voltage rises from 0V to the upper threshold voltage, the low voltage detection becomes effective.
- ✓ Software reset
  - ✓ Writing the reset trigger register using software

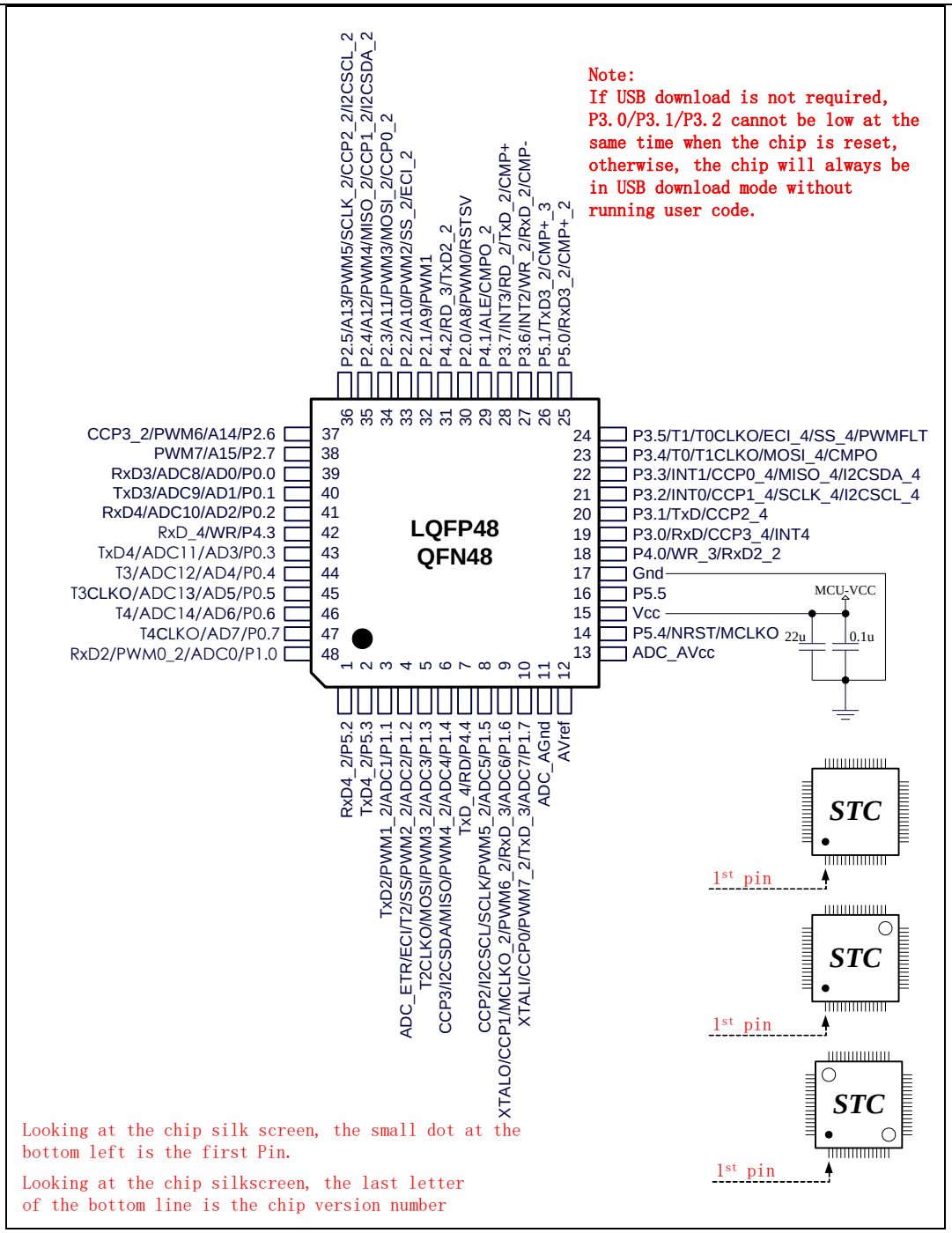
➢ **Interrupts**
- ✓ 43 interrupt sources: INT0(Supports rising edge and falling edge interrupt), INT1(Supports rising edge and falling edge interrupt), INT2(Supports falling edge interrupt only), INT3(Supports falling edge interrupt only), INT4(Supports falling edge interrupt only), timer 0, timer 1, timer 2, timer 3, timer 4, UART 1, UART 2, UART 3, UART 4, ADC, LVD, SPI, I^2C, comparator, PCA/CCP/PWM, Enhanced PWM, Enhanced PWM Anomaly Detection, all I/O interrupts (8 groups), LCD driver, DMA receive and transmit interrupts of USART 1, DMA receive and transmit interrupts of USART 2, DMA receive and transmit interrupts of UART 3, DMA receive and transmit interrupts of UART 4, DMA interrupt of SPI, DMA interrupt of ADC, DMA interrupt of LCD driver and DMA interrupt of memory-to-memory.
- ✓ 4 interrupt priority levels
- ✓ Interrupts that can wake up the CPU in clock stop mode: INT0(P3.2), INT1(P3.3), INT2(P3.6), INT3(P3.7), INT4(P3.0), T0(P3.4), T1(P3.5), T2(P1.2), T3(P0.4), T4(P0.6), RXD(P3.0/P3.6/P1.6/P4.3), RXD2(P1.0/P4.0), RXD3(P0.0/P5.0), RXD4(P0.2/P5.0), CCP0(P1.7/P2.3/P7.0/P3.3), CCP1(P1.6/P2.4/P7.1/P3.2), CCP2(P1.5/P2.5/P7.2/P3.1), CCP3(P1.4/P2.6/P7.3/P3.0), I2C_SDA(P1.4/P2.4/P3.3), SPI_SS(P1.2/P2.2/P3.5), all I/O interrupts, Comparator interrupt, LVD interrupt, Power-down wake-up timer.

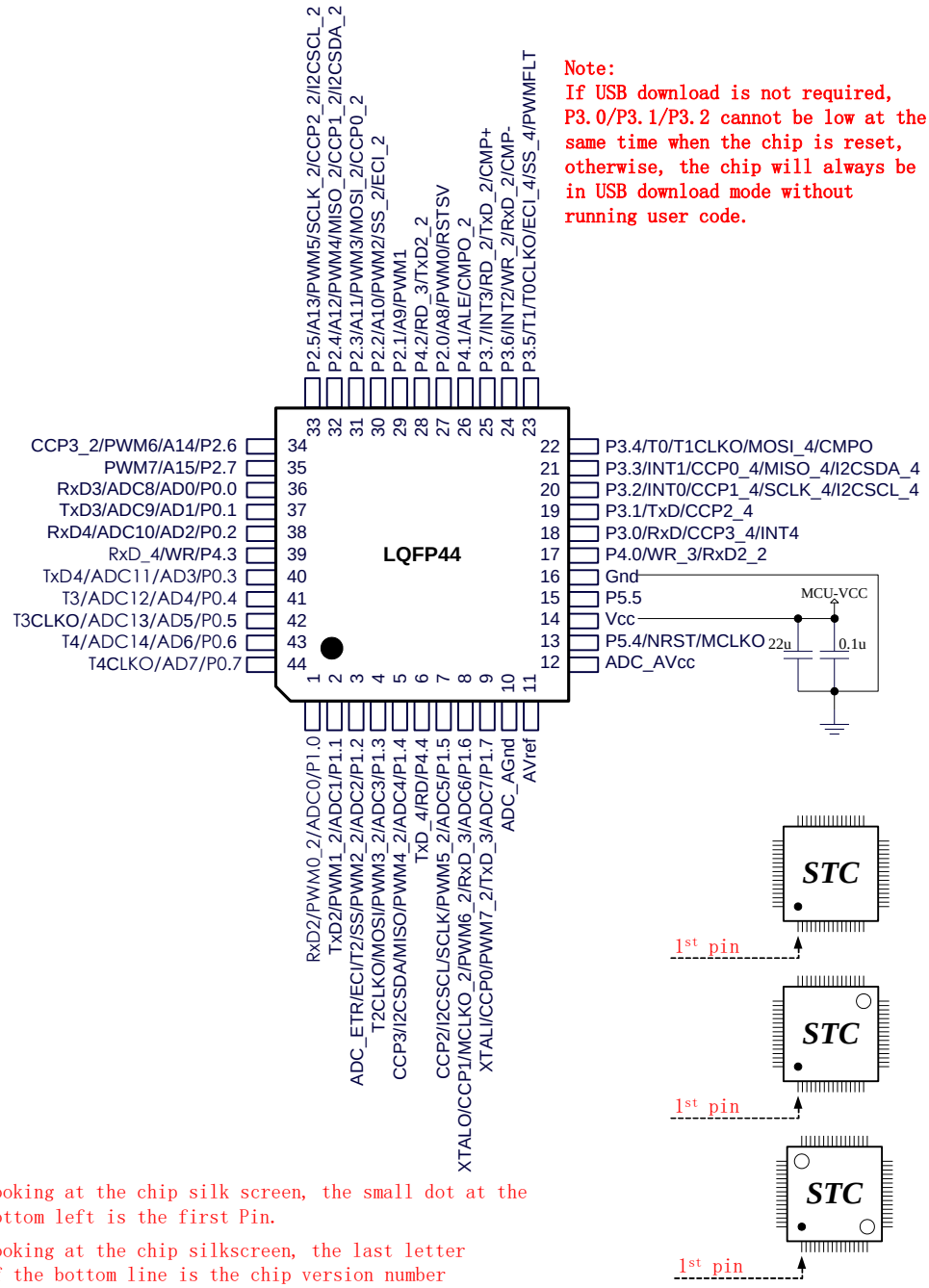
➢ **Digital peripherals**
- ✓ 5 16-bit timers: timer0, timer1, timer2, timer 3, timer 4, where the mode 3 of timer 0 has the Non-Maskable Interrupt (NMI in short) function. Mode 0 of timer 0 and timer 1 is 16-bit Auto-reload mode.
- ✓ 4 high speed UARTs: UART1, UART2, UART3, UART4, whose maximum baudrate clock may be FOSC/4
- ✓ 4 groups of 16-bit PCA modules: CCP0, CCP1, CCP2, CCP3, which can be used for capture, high-speed pulse output, and 6/7/8/10-bit PWM output
- ✓ 8 groups of 15-bit enhanced PWMs, which can realize control signals with dead time, and support external fault detection function. In addition, there are 4 groups of traditional PCA/CCP/PWM can be used as PWM.
- ✓ SPI: Master mode, slave mode or master/slave automatic switch mode are supported.
- ✓ I^2C: Master mode or slave mode are supported.
- ✓ MDU16: Hardware 16-bit Multiplier and Divider which supports operations such as 32-bit divided by 16-bit, 16-bit divided by 16-bit, 16-bit multiplied by 16-bit, data shift, and data normalization.
- ✓ I/O port interrupt: All I/Os support interrupts, each group of I/O interrupts has an independent interrupt entry address, all I/O interrupts can support 4 types interrupt mode: high level interrupt, low level interrupt, rising edge interrupt, falling edge interrupt. Provides 4 levels of interrupt priority and supports power-down wake-up function. (The I/O port interrupts of this series can wake up CPU from power-down, and have 4 levels of interrupt priority)
- ✓ LCD dirver: support 8080 and 6800 interface, and support 8-bit and 16-bit data width.
- ✓ DMA: Support SPI shift to receive data to memory, SPI shift to send data from memory, I2C receive data to memory, USART 1/2/3/4 receive data to memory, USART 1/2/3/4 send data from memory, ADC automatically sample data to memory (calculate average value at the same time), LCD driver send data from memory, and copy data from memory to memory
- ✓ Hardware digital ID: support 32 bytes

- **Analog peripherals**
  - ✓ 15 channels (channel 0 to channel 14) ultra high speed ADC which supports 12-bit precision. The maximum speed can be 800K(800,000 ADC conversions per second)
  - ✓ Channel 15 of ADC is used to test the internal reference voltage. (The default internal reference voltage is 1.19V when the chip is shipped)
  - ✓ A set of comparator (the CMP+, CMP+_2, CMP+_3 and all ADC input ports can be selected as the positive terminal of the comparator, the CMP- and the internal 1.19V reference source can be selected as the negative terminal of the comparator, so the comparator can be used as a multi-channel comparator for time division multiplexing)
  - ✓ DAC: 8 channels advanced PWMs timers can be used as 8 channels DAC, 4-channel PCA can be used as 4-channel DAC
- **GPIO**
  - ✓ Up to 59 GPIOs: P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.0~P4.4, P5.0~P5.5, P6.0~P6.7, P7.0~P7.7
  - ✓ 4 modes for all GPIOs: quasi_bidirectional mode, push-pull outputmode, open drain mode, high-impedance input mode
  - ✓ Except for P3.0 and P3.1, all other I/O ports are in a high-impedance state after power-on. User must configure the I/O ports mode before using them. In addition, the internal 4K pull-up resistor of every I/O can be enabled independently.
- **Package**
  - ✓ LQFP64, LQFP48, LQFP44

2.1.2 Pinouts


Note:
If USB download is not required, P3.0/P3.1/P3.2 cannot be low at the same time when the chip is reset, otherwise, the chip will always be in USB download mode without running user code.
LQFP64
QFN64
1 P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
2 P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
3 P2.3/A11/PWM3/MOSI_2/CCP0_2
4 P2.2/A10/PWM2/SS_2/ECI_2
5 P2.1/A9/PWM1
6 P4.2/RD_3/TxD2_2
7 P2.0/A8/PWM0
8 P7.3/CCP3_3
9 P7.2/CCP2_3
10 P7.1/CCP1_3
11 P7.0/CCP0_3
12 P4.1/ALE/CMPO_2
13 P3.7/INT3/RD_2/TxD_2/CMP+
14 P3.6/INT2/WR_2/RxD_2/CMP-
15 P5.1/TxD3_2/CMP+_3
16 P5.0/RxD3_2/CMP+_2
17 ADC_AVcc
18 P5.4/NRST/MCLKO
19 Vcc
20 P5.5
21 Gnd
22 P4.0/WR_3/RxD2_2
23 P6.4/PWM4_3
24 P6.5/PWM5_3
25 P6.6/PWM6_3
26 P6.7/PWM7_3
27 P3.0/RxD/CCP3_4/INT4
28 P3.1/TxD/CCP2_4
29 P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
30 P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
31 P3.4/T0/T1CLKO/MOSI_4/CMPO
32 P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
33 AVref
34 ADC_AGnd
35 P7.7/CCP0/PWM7_2/TxD_3/ADC7/P1.7
36 XTALI/CCP0/PWM6_2/RxD_3/ADC6/P1.6
37 CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
38 CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
39 TxD_4/RD/P4.4
40 PWM3_3/P6.3
41 PWM2_3/P6.2
42 PWM1_3/P6.1
43 PWM0_3/P6.0
44 T2CLKO/MOSI/PWM3_2/ADC3/P1.3
45 ADC_ETR/ECI/T2/SS/PWM2_2/ADC2/P1.2
46 TxD2/PWM1_2/ADC1/P1.1
47 TxD4_2/P5.3
48 RxD4_2/P5.2
49 CCP3_2/PWM6/A14/P2.6
50 PWM7/A15/P2.7
51 RxD3/ADC8/AD0/P0.0
52 TxD3/ADC9/AD1/P0.1
53 RxD4/ADC10/AD2/P0.2
54 ECI_3/SS_3/P7.4
55 MOSI_3/P7.5
56 I2CSDA_3/MISO_3/P7.6
57 I2CSCL_3/SCLK_3/P7.7
58 RxD_4/WR/P4.3
59 TxD4/ADC11/AD3/P0.3
60 T3/ADC12/AD4/P0.4
61 T3CLKO/ADC13/AD5/P0.5
62 T4/ADC14/AD6/P0.6
63 T4CLKO/AD7/P0.7
64 RxD2/PWM0_2/ADC0/P1.0
MCU-VCC
22u
0.1u
STC
1st pin
STC
1st pin
STC
1st pin
Looking at the chip silk screen, the small dot at the bottom left is the first Pin.
Looking at the chip silkscreen, the last letter of the bottom line is the chip version number

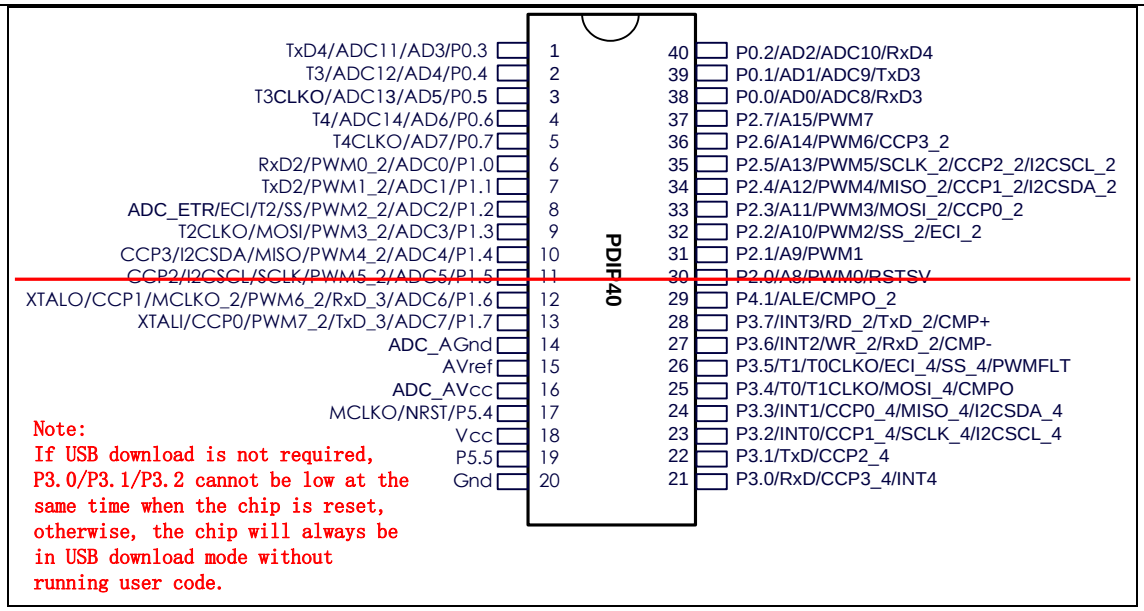

Note:
If USB download is not required, P3.0/P3.1/P3.2 cannot be low at the same time when the chip is reset, otherwise, the chip will always be in USB download mode without running user code.
LQFP48
QFN48
P2.5/A13/PWM5/SCLK_2/CCP2_2/I2CSCL_2
P2.4/A12/PWM4/MISO_2/CCP1_2/I2CSDA_2
P2.3/A11/PWM3/MOSI_2/CCP0_2
P2.2/A10/PWM2/SS_2/ECI_2
P2.1/A9/PWM1
P4.2/RD_3/TxD2_2
P2.0/A8/PWM0/RSTSV
P4.1/ALE/CMPO_2
P3.7/INT3/RD_2/TxD_2/CMP+
P3.6/INT2/WR_2/RxD_2/CMP-
P5.1/TxD3_2/CMP+_3
P5.0/RxD3_2/CMP+_2
CCP3_2/PWM6/A14/P2.6
PWM7/A15/P2.7
RxD3/ADC8/AD0/P0.0
TxD3/ADC9/AD1/P0.1
RxD4/ADC10/AD2/P0.2
RxD_4/WR/P4.3
TxD4/ADC11/AD3/P0.3
T3/ADC12/AD4/P0.4
T3CLKO/ADC13/AD5/P0.5
T4/ADC14/AD6/P0.6
T4CLKO/AD7/P0.7
RxD2/PWM0_2/ADC0/P1.0
P3.5/T1/T0CLKO/ECI_4/SS_4/PWMFLT
P3.4/T0/T1CLKO/MOSI_4/CMPO
P3.3/INT1/CCP0_4/MISO_4/I2CSDA_4
P3.2/INT0/CCP1_4/SCLK_4/I2CSCL_4
P3.1/TxD/CCP2_4
P3.0/RxD/CCP3_4/INT4
P4.0/WR_3/RxD2_2
Gnd
P5.5
Vcc
P5.4/NRST/MCLKO
ADC_AVcc
MCU-VCC
22u
0.1u
RxD4_2/P5.2
TxD4_2/P5.3
TxD2/PWM1_2/ADC1/P1.1
ADC_ETR/ECI/T2/SS/PWM2_2/ADC2/P1.2
T2CLKO/MOSI/PWM3_2/ADC3/P1.3
CCP3/I2CSDA/MISO/PWM4_2/ADC4/P1.4
TxD_4/RD/P4.4
CCP2/I2CSCL/SCLK/PWM5_2/ADC5/P1.5
XTALO/CCP1/MCLKO_2/PWM6_2/RxD_3/ADC6/P1.6
XTALI/CCP0/PWM7_2/TxD_3/ADC7/P1.7
ADC_AGnd
AVref
STC
1st pin
STC
1st pin
STC
1st pin
Looking at the chip silk screen, the small dot at the bottom left is the first Pin.
Looking at the chip silkscreen, the last letter of the bottom line is the chip version number



Note:
If USB download is not required, P3.0/P3.1/P3.2 cannot be low at the same time when the chip is reset, otherwise, the chip will always be in USB download mode without running user code.

Looking at the chip silk screen, the small dot at the bottom left is the first Pin.

Looking at the chip silkscreen, the last letter of the bottom line is the chip version number



Typical download circuit

universal USB to UART tool

ISP download steps:

1. **Connect the universal USB to UART tool to the target chip according to the connection method shown in the figure above.**
2. **Press the power button to confirm that the target chip is in a power-off state (the power-on LED is off).**
   Note: When the tool is powered on for the first time, there is no external power supply, so if it is the first time to use this tool, you can skip this step.
3. **Click the "Download/Program" button in the STC-ISP download software.**
4. **Press the power button again to power on the target chip (the power-on LED is on).**
5. **Start ISP download.**
   Note: It has been found that when using the USB cable for ISP download, if the USB cable is too thin and the voltage drop on the USB cable is too large, this will result in insufficient power supply during the ISP download. Therefore, please be sure to use the booster USB cable for ISP download.

2.1.3 Pin descriptions

Pin number				name	type	description
LQFP64	LQFP48	LQFP44	PDIP40			
1	1			P5.2	I/O	Standard IO port
				RxD4_2	I	Serial input of UART4
2	2			P5.3	I/O	Standard IO port
				TxD4_2	O	Serial Transmit pin of UART4
3	3	2	7	P1.1	I/O	Standard IO port
				ADC1	I	ADC analog input 1
				PWM1_2	O	Output pin of Enhance PWM 1
				TxD2	O	Serial Transmit pin of UART2
4	4	3	8	P1.2	I/O	Standard IO port
				ADC2	I	ADC analog input 2
				PWM2_2	O	Output pin of Enhance PWM2
				SS	I/O	Slave selection of SPI
				T2	I	T2 input
				ECI	I	External pulse input of PCA
				ADC_ETR	I	ADC external trigger input
5	5	4	9	P1.3	I/O	Standard IO port
				ADC3	I	ADC analog input 3
				PWM3_2	O	Enhanced PWM channel 3 output pin
				MOSI	I/O	Master Output/Slave Input of SPI
				T2CLKO	O	Clock out of timer 2
6				P6.0	I/O	Standard IO port
				PWM0_3	O	Enhanced PWM channel 0 output pin
7				P6.1	I/O	Standard IO port
				PWM1_3	O	Enhanced PWM channel 1 output pin
8				P6.2	I/O	Standard IO port
				PWM2_3	O	Enhanced PWM channel 2 output pin
9				P6.3	I/O	Standard IO port
				PWM3_3	O	Enhanced PWM channel 3 output pin

Pin number				name	type	description
LQFP64	LQFP48	LQFP44	PDIP40			
10	6	5	10	P1.4	I/O	Standard IO port
				ADC4	I	ADC analog input 4
				PWM4_2	O	Enhanced PWM channel 4 output pin
				MISO	I/O	Master Iutput/Slave Onput of SPI
				SDA	I/O	Serial data line of I2C
				CCP3	I/O	Capture input and pulse output of PCA
11	7	6		P4.4	I/O	Standard IO port
				RD	O	READ signal of external bus
				TxD_4	O	Transmit pin of USART1
12	8	7	11	P1.5	I/O	Standard IO port
				ADC5	I	ADC analog input 5
				PWM5_2	O	Enhanced PWM channel 5 output pin
				SCLK	I/O	Serial Clock of SPI
				SCL	I/O	Clock line of I2C
				CCP2	I/O	Capture input and pulse output of PCA2
13	9	8	12	P1.6	I/O	Standard IO port
				ADC6	I	ADC analog input 6
				RxD_3	I	Input of USART1
				PWM6_2	O	Enhanced PWM channel 6 output pin
				MCLKO_2	O	Main clock output
				CCP1	I/O	Capture input and pulse output of PCA1
				XTALO	O	Connect to external oscillator
14	10	9	13	P1.7	I/O	Standard IO port
				ADC7	I	ADC analog input 7
				TxD_3	O	Transmit pin of USART 1
				PWM7_2	O	Enhanced PWM channel 7 output pin
				CCP0	I/O	Capture input and pulse output of PCA0
				XTALI	I	Connect to external oscillator
15	11	10	14	ADC_AGnd	GND	ADC Ground
16	12	11	15	AVref	I	Reference voltage pin of ADC
17	13	12	16	ADC_AVcc	VCC	ADC Power Supply
18	14	13	17	P5.4	I/O	Standard IO port
				NRST	I	Reset pin (low level reset)
				MCLKO	O	Main clock output
19	15	14	18	Vcc	VCC	Power Supply
20	16	15	19	P5.5	I/O	Standard IO port
21	17	16	20	Gnd	GND	Ground

Pin number				name	type	description
LQFP64	LQFP48	LQFP44	PDIP40			
22	18	17		P4.0	I/O	Standard IO port
				WR_3	O	WRITE signal of external bus
				RxD2_2	I	Input of USART2
23				P6.4	I/O	Standard IO port
				PWM4_3	O	Enhanced PWM channel 4 output pin
24				P6.5	I/O	Standard IO port
				PWM5_3	O	Enhanced PWM channel 5 output pin
25				P6.6	I/O	Standard IO port
				PWM6_3	O	Enhanced PWM channel 6 output pin
26				P6.7	I/O	Standard IO port
				PWM7_3	O	Enhanced PWM channel 7 output pin
27	19	18	21	P3.0	I/O	Standard IO port
				RxD	I	Input of USART1
				CCP3_4	I/O	Capture input and pulse output of PCA3
				INT4	I	External interrupt 4
28	20	19	22	P3.1	I/O	Standard IO port
				TxD	O	Transmit pin of USART1
				CCP2_4	I/O	Capture input and pulse output of PCA2
29	21	20	23	P3.2	I/O	Standard IO port
				INT0	I	External interrupt 0
				CCP1_4	I/O	Capture input and pulse output of PCA1
				SCLK_4	I/O	Serial Clock of SPI
				SCL_4	I/O	Serial Clock line of I2C
30	22	21	24	P3.3	I/O	Standard IO port
				INT1	I	External interrupt 1
				CCP0_4	I/O	Capture input and pulse output of PCA0
				MISO_4	I/O	Master Iutput/Slave Onput of SPI
				SDA_4	I/O	Serial data line of I2C
31	23	22	25	P3.4	I/O	Standard IO port
				T0	I	Timer0 external input
				T1CLKO	O	Clock out of timer 1
				MOSI_4	I/O	Master Output/Slave Input of SPI
				CMPO	O	Output of comparator

Pin number				name	type	description
LQFP64	LQFP48	LQFP44	PDIP40			
32	24	23	26	P3.5	I/O	Standard IO port
				T1	I	Timer1 external input
				T0CLKO	O	Clock out of timer 0
				ECI_4	I	External pulse input of PCA
				SS_4	I	Slave selection of SPI (it is output with regard to master)
				PWMFLT	I	Enhance PWM external anomaly detection pin
33	25			P5.0	I/O	Standard IO port
				RxD3_2	I	Input of USART3
				CMP+_2	I	Positive input of comparator
34	26			P5.1	I/O	Standard IO port
				TxD3_2	O	Transmit pin of USART3
				CMP+_3	I	Positive input of comparator
35	27	24	27	P3.6	I/O	Standard IO port
				INT2	I	External interrupt 2
				WR_2	O	WRITE signal of external bus
				RxD_2	I	Input of USART1
				CMP-	I	Negative input of comparator
36	28	25	28	P3.7	I/O	Standard IO port
				INT3	I	External interrupt 3
				RD_2	O	READ signal of external bus
				TxD_2	O	Transmit pin of USART1
				CMP+	I	Positive input of comparator
37	29	26	29	P4.1	I/O	Standard IO port
				ALE	O	Address latch signal
				CMPO_2	O	Output of comparator
38				P7.0	I/O	Standard IO port
				CCP0_3	I/O	Capture input and pulse output of PCA0
39				P7.1	I/O	Standard IO port
				CCP1_3	I/O	Capture input and pulse output of PCA1
40				P7.2	I/O	Standard IO port
				CCP2_3	I/O	Capture input and pulse output of PCA2
41				P7.3	I/O	Standard IO port
				CCP3_3	I/O	Capture input and pulse output of PCA3
42	30	27	30	P2.0	I/O	Standard IO port
				A8	I	Address bus
				PWM0	O	Enhanced PWM channel 0 output pin
				RSTSV	-	The initial level of the port can be configured when the ISP downloads
43	31	28		P4.2	I/O	Standard IO port
				RD_3	O	READ signal of external bus
				TxD2_2	O	Transmit pin of USART2

Pin number				name	type	description
LQFP64	LQFP48	LQFP44	PDIP40			
44	32	29	31	P2.1	I/O	Standard IO port
				A9	I	Address bus
				PWM1	O	Enhanced PWM channel 1 output pin
45	33	30	32	P2.2	I/O	Standard IO port
				A10	I	Address bus
				PWM2	O	Enhanced PWM channel 2 output pin
				SS_2	I	Slave selection of SPI (it is output with regard to master)
				ECI_2	I	External pulse input of PCA
46	34	31	33	P2.3	I/O	Standard IO port
				A11	I	Address bus
				PWM3	O	Enhanced PWM channel 3 output pin
				MOSI_2	I/O	Master Output/Slave Input of SPI
				CCP0_2	I/O	Capture input and pulse output of PCA0
47	35	32	34	P2.4	I/O	Standard IO port
				A12	I	Address bus
				PWM4	O	Enhanced PWM channel 4 output pin
				MISO_2	I/O	Master Iutput/Slave Onput of SPI
				SDA_2	I/O	Serial data line of I2C
				CCP1_2	I/O	Capture input and pulse output of PCA1
48	36	33	35	P2.5	I/O	Standard IO port
				A13	I	Address bus
				PWM5	O	Enhanced PWM channel 5 output pin
				SCLK_2	I/O	Serial Clock of SPI
				SCL_2	I/O	Serial Clock line of I2C
				CCP2_2	I/O	Capture input and pulse output of PCA2
49	37	34	36	P2.6	I/O	Standard IO port
				A14	I	Address bus
				PWM6	O	Enhanced PWM channel 6 output pin
				CCP3_2	I/O	Capture input and pulse output of PCA3
50	38	35	37	P2.7	I/O	Standard IO port
				A15	I	Address bus
				PWM7	O	Enhanced PWM channel 7 output pin
51	39	36	38	P0.0	I/O	Standard IO port
				AD0	I	Address bus
				ADC8	I	ADC analog input 8
				RxD3	I	Input of USART3

Pin number				name	type	description
LQFP64	LQFP48	LQFP44	PDIP40			
52	40	37	39	P0.1	I/O	Standard IO port
				AD1	I	Address bus
				ADC9	I	ADC analog input 9
				TxD3	O	Transmit pin of USART3
53	41	38	40	P0.2	I/O	Standard IO port
				AD2	I	Address bus
				ADC10	I	ADC analog input 10
				RxD4	I	Serial input of UART4
54				P7.4	I/O	Standard IO port
				SS_3	I	Slave selection of SPI (it is output with regard to master)
				ECI_3	I	External pulse input of PCA
55				P7.5	I/O	Standard IO port
				MOSI_3	I/O	Master Output/Slave Input of SPI
56				P7.6	I/O	Standard IO port
				MISO_3	I/O	Master Iutput/Slave Onput of SPI
				SDA_3	I/O	Serial data line of I2C
57				P7.7	I/O	Standard IO port
				SCLK_3	I/O	Serial Clock of SPI
				SCL_3	I/O	Serial Clock line of I2C
58	42	39		P4.3	I/O	Standard IO port
				WR	O	WRITE signal of external bus
				RxD_4	I	Input of USART1
59	43	40	1	P0.3	I/O	Standard IO port
				AD3	I	Address bus
				ADC11	I	ADC analog input 11
				TxD4	O	Transmit pin of USART4
60	44	41	2	P0.4	I/O	Standard IO port
				AD4	I	Address bus
				ADC12	I	ADC analog input 12
				T3	I	Timer3 external input
61	45	42	3	P0.5	I/O	Standard IO port
				AD5	I	Address bus
				ADC13	I	ADC analog input 13
				T3CLKO	O	Clock out of timer3
62	46	43	4	P0.6	I/O	Standard IO port
				AD6	I	Address bus
				ADC14	I	ADC analog input 14
				T4	I	Timer4 external input
63	47	44	5	P0.7	I/O	Standard IO port
				AD7	I	Address bus
				T4CLKO	O	Clock out of timer4
64	48	1	6	P1.0	I/O	Standard IO port
				ADC0	I	ADC analog input 0
				PWM0_2	O	Enhanced PWM channel 0 output pin
				RxD2	I	Input of USART12

3 Function pins switch

Some special peripherals of STC8A8K64D4 series of microcontrollers can be switched among several I/O pins to realize one peripheral used as multiple device through time-sharing, such as UART, SPI, PCA, I2C and bus control pins.

3.1 Register related to function pin switch

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
BUS_SPEED	Bus speed control register	A1H	RW_S[1:0]						SPEED[1:0]		00xx,xx00
P_SW1	Peripheral port switch register 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,000x
P_SW2	Peripheral port switch register 2	BAH	EAXFR		I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
PWM0CR	PWM0 Control register	FF14H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI	00x0,0000
PWM1CR	PWM1 Control register	FF1CH	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI	00x0,0000
PWM2CR	PWM2 Control register	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI	00x0,0000
PWM3CR	PWM3 Control register	FF2CH	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI	00x0,0000
PWM4CR	PWM4 Control register	FF34H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI	00x0,0000
PWM5CR	PWM5 Control register	FF3CH	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI	00x0,0000
PWM6CR	PWM6 Control register	FF44H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI	00x0,0000
PWM7CR	PWM7 Control register	FF4CH	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI	00x0,0000
MCLKOCR	Master clock output control register	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
LCMIFCFG	LCM Interface Configuration Register	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFPS[1:0]		D16_D8	M68_80	0x00,0000
LCMIFCFG2	LCM Interface Configuration Register 2	FE51H		LCMIFEXPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000

3.1.1 Bus Speed Control Register (BUS_SPEED)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]						SPEED[1:0]	

RW_S[1:0]: External bus RD/WR control line select bits

RW_S[1:0]	RD	WR
00	P4.4	P4.3
01	P3.7	P3.6
10	P4.2	P4.0
11	Reserved	

3.1.2 Peripheral port switch register 1(P_SW1) (for UART1, CCP, SPI switching)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-

S1_S[1:0]：USART1 pin selection bits

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

CCP_S[1:0]: PCA pin selection bits

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2	CCP3
00	P1.2	P1.7	P1.6	P1.5	P1.4
01	P2.2	P2.3	P2.4	P2.5	P2.6

10	P7.4	P7.0	P7.1	P7.2	P7.3
11	P3.5	P3.3	P3.2	P3.1	P3.0

SPI_S[1:0]：SPI pin selection bits

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P7.4	P7.5	P7.6	P7.7
11	P3.5	P3.4	P3.3	P3.2

3.1.3 Peripheral port switch register 2(P_SW2) (for UART2/3/4, I2C, Comparator output switching)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

EAXFR：Extended RAM area Special Function Register (XFR) access control register

0: Access to XFR is prohibited

1: Enable access to XFR.

When XFR needs to be accessed, EAXFR must be set to 1 before XFR can be read or written properly

I2C_S[1:0]：I^2C pin selection bits

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P2.5	P2.4
10	P7.7	P7.6
11	P3.2	P3.3

CMPO_S：Comparator output pin selection bit

CMPO_S	CMPO
0	P3.4
1	P4.1

S4_S：UART4 pin selection bit

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

S3_S：UART3 pin selection bit

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

S2_S：USART2 pin selection bit

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

3.1.4 Clock selection register(MCLKOCR)

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

MCLKO_S：Main clock out pin selection bit

MCLKO_S	MCLKO
0	P5.4
1	P1.6

3.1.5 Enhanced PWM Control Register (PWMnCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF14H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF1CH	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF2CH	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF34H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI
PWM5CR	FF3CH	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF44H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF4CH	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI

C0_S[1:0]: Enhanced PWM Channel 0 Output Pin Select Bits

C0_S[1:0]	PWM0
00	P2.0
01	P1.0
10	P6.0
11	Reserved

C1_S[1:0]: Enhanced PWM Channel 1 Output Pin Select Bits

C1_S[1:0]	PWM1
00	P2.1
01	P1.1
10	P6.1
11	Reserved

C2_S[1:0]: Enhanced PWM Channel 2 Output Pin Select Bits

C2_S[1:0]	PWM2
00	P2.2
01	P1.2
10	P6.2
11	Reserved

C3_S[1:0]: Enhanced PWM Channel 3 Output Pin Select Bits

C3_S[1:0]	PWM3
00	P2.3
01	P1.3
10	P6.3
11	Reserved

C4_S[1:0]: Enhanced PWM Channel 4 Output Pin Select Bit

C4_S[1:0]	PWM4
00	P2.4
01	P1.4
10	P6.4
11	Reserved

C5_S[1:0]: Enhanced PWM Channel 0 Output Pin Select Bit

C5_S[1:0]	PWM5
00	P2.5
01	P1.5
10	P6.5
11	Reserved

C6_S[1:0]: Enhanced PWM Channel 0 Output Pin Select Bit

C6_S[1:0]	PWM6
00	P2.6
01	P1.6
10	P6.6
11	Reserved

C7_S[1:0] : Enhanced PWM Channel 7 Output Pin Select Bit

C7_S[1:0]	PWM7
00	P2.7
01	P1.7
10	P6.7
11	Reserved

3.1.6 LCM Interface Configuration Register (LCMIFCFG)

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

LCMIFCPS[1:0]: LCM interface control pin selection bits

LCMIFCPS [1:0]	RS	READ signal (RD)of I8080 Enable signal (E) of M6800	Wirte signal (WR) of I8080 Write/Read signal (RW) of M6800
00	P4.1	P4.4	P4.3
01	P4.1	P3.7	P3.6
10	P4.1	P4.2	P4.0
11	P4.0	P3.7	P3.6

LCMIFDPS[1:0]: LCM interface data pin selection bits

LCMIFDPS [1:0]	D16_D8	data high byte DB[15:8]	data low byte DB[7:0]
00	0	N/A	P2[7:0]
01	0	N/A	P6[7:0]
10	0	N/A	P2[7:0]
11	0	N/A	P6[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	P7[7:0]
11	1	P6[7:0]	P7[7:0]

3.2 Example Routines

3.2.1 USART1 switch

C language code

```
// Operating frequency for test is 11.0592MHz

#include “reg51.h”

sfr         P_SW1         =    0xa2;

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

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

        P_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
//      P_SW1 = 0xc0;                   //RXD_4/P4.3, TXD_4/P4.4

        while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW1       DATA        0A2H

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_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
;           MOV         P_SW1,#0C0H         ;RXD_4/P4.3, TXD_4/P4.4

            SJMP        $

            END
```

3.2.2 USART2 switch

C language code

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"

sfr         P_SW2       =   0xba;

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

void main()
{
    P0M0 = 0x00;
```

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

    P_SW2 = 0x00;                   //RXD2/P1.0, TXD2/P1.1
//  P_SW2 = 0x01;                   //RXD2_2/P4.0, TXD2_2/P4.2

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

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     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/P4.0, TXD2_2/P4.2

            SJMP    $
```

```
END
```

3.2.3 UART3 switch

C language code

```
// Operating frequency for test is 11.0592MHz

#include “reg51.h”

sfr     P_SW2       =   0xba;

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

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

    P_SW2 = 0x00;                   //RXD3/P0.0, TXD3/P0.1
//  P_SW2 = 0x02;                   //RXD3_2/P5.0, TXD3_2/P5.1

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

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         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          ;RXD3/P0.0, TXD3/P0.1
;           MOV         P_SW2,#02H          ;RXD3_2/P5.0, TXD3_2/P5.1

            SJMP        $

            END
```

3.2.4 UART4 switch

C language code

```
// Operating frequency for test is 11.0592MHz

#include “reg51.h”

sfr         P_SW2       =   0xba;

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

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

```
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x00;                   //RXD4/P0.2, TXD4/P0.3
//  P_SW2 = 0x04;                   //RXD4_2/P5.2, TXD4_2/P5.3

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

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     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      ;RXD4/P0.2, TXD4/P0.3
;           MOV     P_SW2,#04H      ;RXD4_2/P5.2, TXD4_2/P5.3

            SJMP    $

            END
```

3.2.5 SPI switch

C language code

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"

sfr         P_SW1         =    0xa2;

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

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

    P_SW1 = 0x00;                   //SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
//  P_SW1 = 0x04;                   //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
//  P_SW1 = 0x08;                   //SS_3/P7.4, MOSI_3/P7.5, MISO_3/P7.6, SCLK_3/P7.7
//  P_SW1 = 0x0c;                   //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW1       DATA        0A2H

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_SW1,#00H          ;SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
;           MOV     P_SW1,#04H          ;SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
;           MOV     P_SW1,#08H          ;SS_3/P7.4, MOSI_3/P7.5, MISO_3/P7.6, SCLK_3/P7.7
;           MOV     P_SW1,#0CH          ;SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

            SJMP    $

            END
```

3.2.6 PWM switch

Assembly code

```
P_SW2       DATA    0BAH
PWM0CR      EQU     0FF14H
PWM1CR      EQU     0FF1CH
PWM2CR      EQU     0FF24H
PWM3CR      EQU     0FF2CH
PWM4CR      EQU     0FF34H
PWM5CR      EQU     0FF3CH
PWM6CR      EQU     0FF44H
PWM7CR      EQU     0FF4CH

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, #3FH
        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          ;PWM0/P2.0
;       MOV     A,#08H          ;PWM0_2/P1.0
;       MOV     A,#10H          ;PWM0_3/P6.0
        MOV     DPTR,#PWM0CR
        MOVX    @DPTR,A
        MOV     A,#00H          ;PWM1/P2.1
;       MOV     A,#08H          ;PWM1_2/P1.1
;       MOV     A,#10H          ;PWM1_3/P6.1
        MOV     DPTR,#PWM1CR
        MOVX    @DPTR,A
        MOV     A,#00H          ;PWM2/P2.2
;       MOV     A,#08H          ;PWM2_2/P1.2
;       MOV     A,#10H          ;PWM2_3/P6.2
        MOV     DPTR,#PWM2CR
        MOVX    @DPTR,A
        MOV     A,#00H          ;PWM3/P2.3
;       MOV     A,#08H          ;PWM3_2/P1.3
;       MOV     A,#10H          ;PWM3_3/P6.3
        MOV     DPTR,#PWM3CR
        MOVX    @DPTR,A
        MOV     A,#00H          ;PWM4/P2.4
;       MOV     A,#08H          ;PWM4_2/P1.4
;       MOV     A,#10H          ;PWM4_3/P6.4
        MOV     DPTR,#PWM4CR
        MOVX    @DPTR,A
        MOV     A,#00H          ;PWM5/P2.5
;       MOV     A,#08H          ;PWM5_2/P1.5
;       MOV     A,#10H          ;PWM5_3/P6.5
        MOV     DPTR,#PWM5CR
        MOVX    @DPTR,A
        MOV     A,#00H          ;PWM6/P2.6
;       MOV     A,#08H          ;PWM6_2/P1.6
;       MOV     A,#10H          ;PWM6_3/P6.6
        MOV     DPTR,#PWM6CR
        MOVX    @DPTR,A
        MOV     A,#00H          ;PWM7/P2.7
;       MOV     A,#08H          ;PWM7_2/P1.7
;       MOV     A,#10H          ;PWM7_3/P6.7
        MOV     DPTR,#PWM7CR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H
```

```
        SJMP    $

        END
```

C language code

```
#include "reg51.h"

#define PWM0CR  (*(unsigned char volatile xdata *)0xff14)
#define PWM1CR  (*(unsigned char volatile xdata *)0xff1c)
#define PWM2CR  (*(unsigned char volatile xdata *)0xff24)
#define PWM3CR  (*(unsigned char volatile xdata *)0xff2c)
#define PWM4CR  (*(unsigned char volatile xdata *)0xff34)
#define PWM5CR  (*(unsigned char volatile xdata *)0xff3c)
#define PWM6CR  (*(unsigned char volatile xdata *)0xff44)
#define PWM7CR  (*(unsigned char volatile xdata *)0xff4c)

sfr     P_SW2       =   0xba;

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

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

        P_SW2 = 0x80;
        PWM0CR = 0x00;          //PWM0/P2.0
//      PWM0CR = 0x08;          //PWM0_2/P1.0
//      PWM0CR = 0x10;          //PWM0_3/P6.0
        PWM1CR = 0x00;          //PWM1/P2.1
//      PWM1CR = 0x08;          //PWM1_2/P1.1
//      PWM1CR = 0x10;          //PWM1_3/P6.1
        PWM2CR = 0x00;          //PWM2/P2.2
//      PWM2CR = 0x08;          //PWM2_2/P1.2
//      PWM2CR = 0x10;          //PWM2_3/P6.2
        PWM3CR = 0x00;          //PWM3/P2.3
//      PWM3CR = 0x08;          //PWM3_2/P1.3
//      PWM3CR = 0x10;          //PWM3_3/P6.3
```

```
        PWM4CR = 0x00;          //PWM4/P2.4
//      PWM4CR = 0x08;          //PWM4_2/P1.4
//      PWM4CR = 0x10;          //PWM4_3/P6.4
        PWM5CR = 0x00;          //PWM5/P2.5
//      PWM5CR = 0x08;          //PWM5_2/P1.5
//      PWM5CR = 0x10;          //PWM5_3/P6.5
        PWM6CR = 0x00;          //PWM6/P2.6
//      PWM6CR = 0x08;          //PWM6_2/P1.6
//      PWM6CR = 0x10;          //PWM6_3/P6.6
        PWM7CR = 0x00;          //PWM7/P2.7
//      PWM7CR = 0x08;          //PWM7_2/P1.7
//      PWM7CR = 0x10;          //PWM7_3/P6.7
        P_SW2 = 0x00;

        while (1);
}
```

3.2.7 PCA/CCP/PWM switch

Assembly code

```
P_SW1   DATA    0A2H

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, #3FH
        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          ;ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5,CCP3/P1.4
;       MOV     P_SW1,#10H          ;ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5,CCP3_2/P2.6
;       MOV     P_SW1,#20H          ;ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2,CCP3_3/P7.3
;       MOV     P_SW1,#30H          ;ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1,CCP3_4/P3.0

        SJMP    $
```

```
        END
```

C language code

```
#include "reg51.h"

sfr     P_SW1       =   0xa2;

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

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

        P_SW1 = 0x00;               //ECI/P1.2, CCP0/P1.7, CCP1/P1.6, CCP2/P1.5,CCP3/P1.4
//      P_SW1 = 0x10;               //ECI_2/P2.2, CCP0_2/P2.3, CCP1_2/P2.4, CCP2_2/P2.5,CCP3_2/P2.6
//      P_SW1 = 0x20;               //ECI_3/P7.4, CCP0_3/P7.0, CCP1_3/P7.1, CCP2_3/P7.2,CCP3_3/P7.3
//      P_SW1 = 0x30;               //ECI_4/P3.5, CCP0_4/P3.3, CCP1_4/P3.2, CCP2_4/P3.1,CCP3_4/P3.0

        while (1);
}
```

3.2.8 I2C switch

C language code

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"

sfr     P_SW2       =   0xba;

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

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

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

    P_SW2 = 0x00;                   //SCL/P1.5, SDA/P1.4
//  P_SW2 = 0x10;                   //SCL_2/P2.5, SDA_2/P2.4
//  P_SW2 = 0x20;                   //SCL_3/P7.7, SDA_3/P7.6
//  P_SW2 = 0x30;                   //SCL_4/P3.2, SDA_4/P3.3

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

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     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/P1.5, SDA/P1.4
;       MOV     P_SW2,#10H          ;SCL_2/P2.5, SDA_2/P2.4
;       MOV     P_SW2,#20H          ;SCL_3/P7.7, SDA_3/P7.6
;       MOV     P_SW2,#30H          ;SCL_4/P3.2, SDA_4/P3.3

        SJMP    $
        END
```

3.2.9 Comparator output switch

C language code

```
// Operating frequency for test is 11.0592MHz

#include “reg51.h”

sfr     P_SW2       =   0xba;

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

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

    P_SW2 = 0x00;                   //CMPO/P3.4
//  P_SW2 = 0x08;                   //CMPO_2/P4.1

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

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     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          ;CMPO/P3.4
;           MOV     P_SW2,#08H          ;CMPO_2/P4.1

            SJMP    $

            END
```

3.2.10 Main clock output switch

C language code

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"

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

sfr       P_SW2       =   0xba;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
```

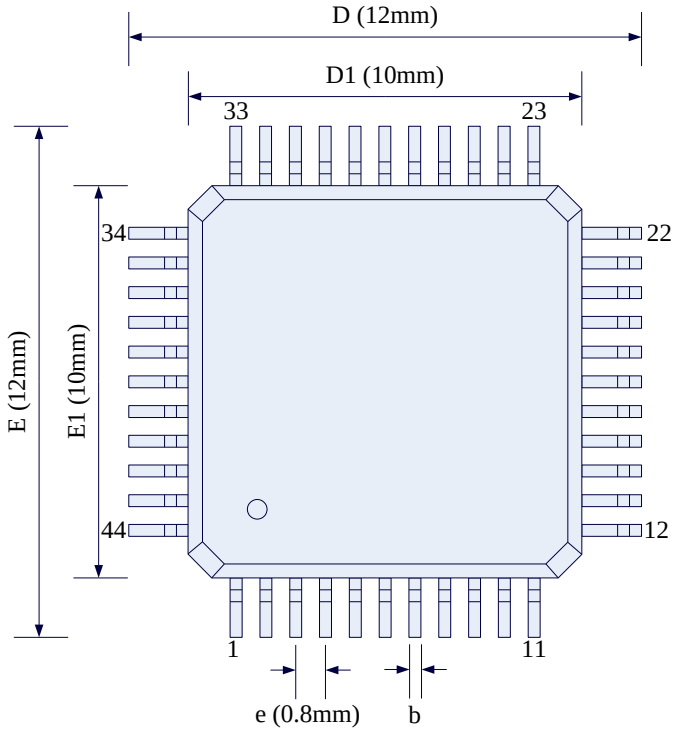
```
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

    P_SW2 = 0x80;
    CLKOCR = 0x04;                  //IRC/4 output via MCLKO/P5.4
//  CLKOCR = 0x84;                  //IRC/4 output via MCLKO_2/P1.6
    P_SW2 = 0x00;

    while (1);
}
```

Assembly code

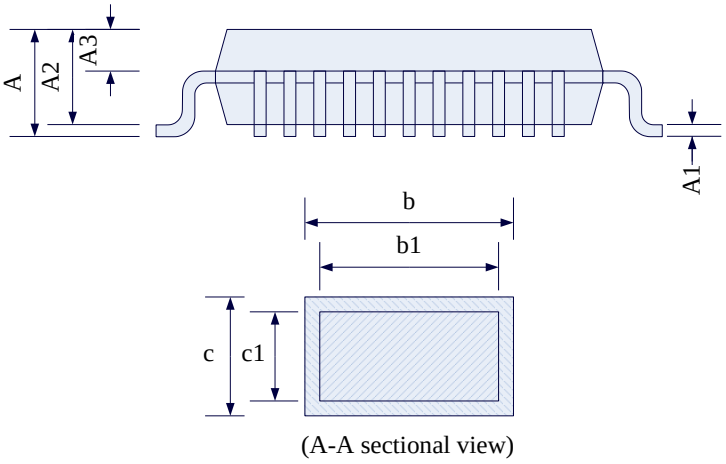
```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

CLKOCR      EQU     0FE05H

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,#04H          ;IRC/4 output via MCLKO/P5.4
;           MOV     A,#84H          ;IRC/4 output via MCLKO_2/P1.6
            MOV     DPTR,#CLKOCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            SJMP    $

            END
```

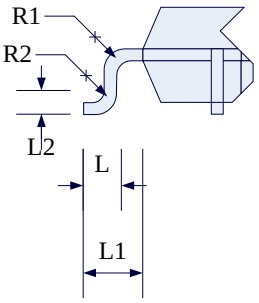
4 Package Dimensions

4.1 LQFP44 Package mechanical data (12mm*12mm)

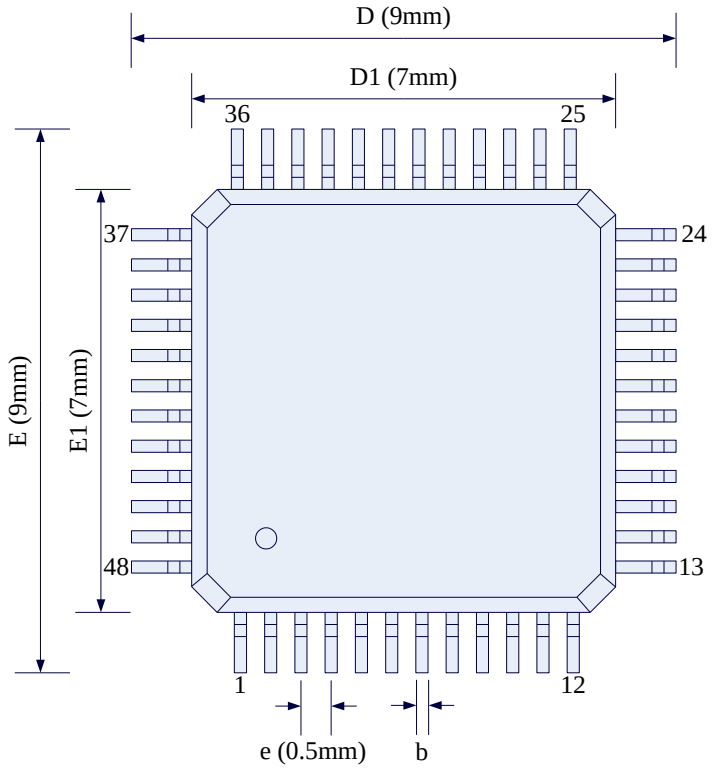


General size			
			mm
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.33	-	0.42
b1	0.32	0.35	0.38
c	0.13	-	0.18
c1	0.12	0.13	0.14
D	11.95	12.00	12.05
D1	9.90	10.00	10.10
E	11.95	12.00	12.05
E1	9.90	10.00	10.10
e	0.70	0.80	0.90
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

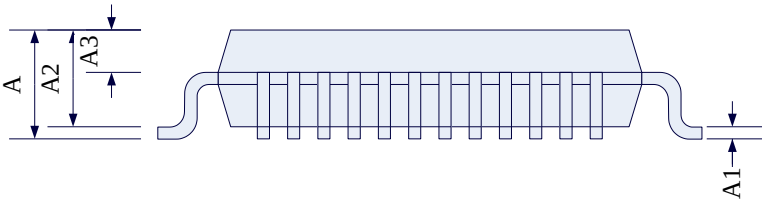


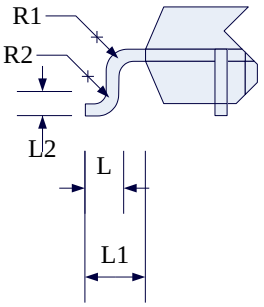


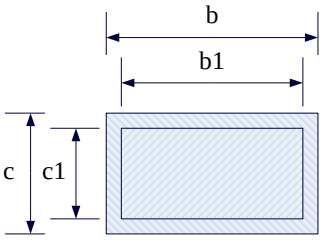
4.2 LQFP48 Package mechanical data（9mm*9mm）



General size			
			mm
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
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	0.45	0.50	0.55
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

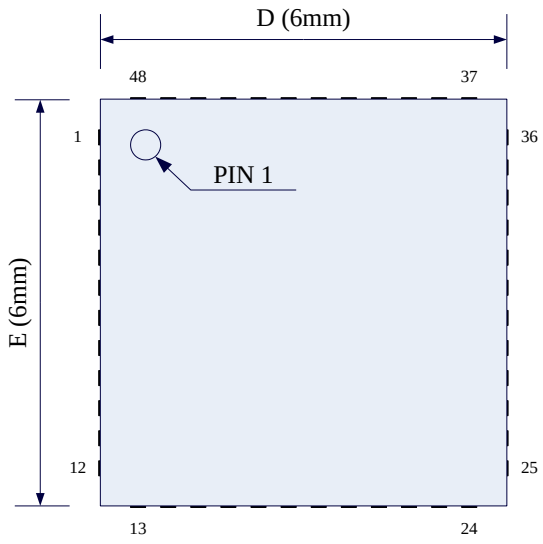


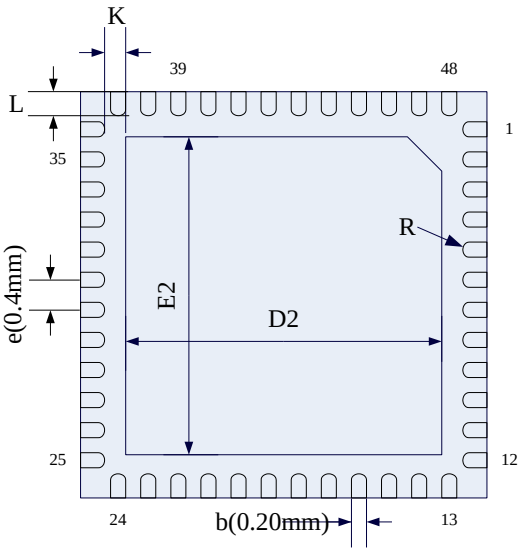


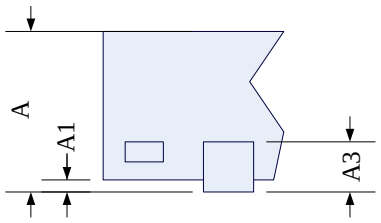


(A-A sectional view)

4.3 QFN48 Package mechanical data (6mm*6mm)



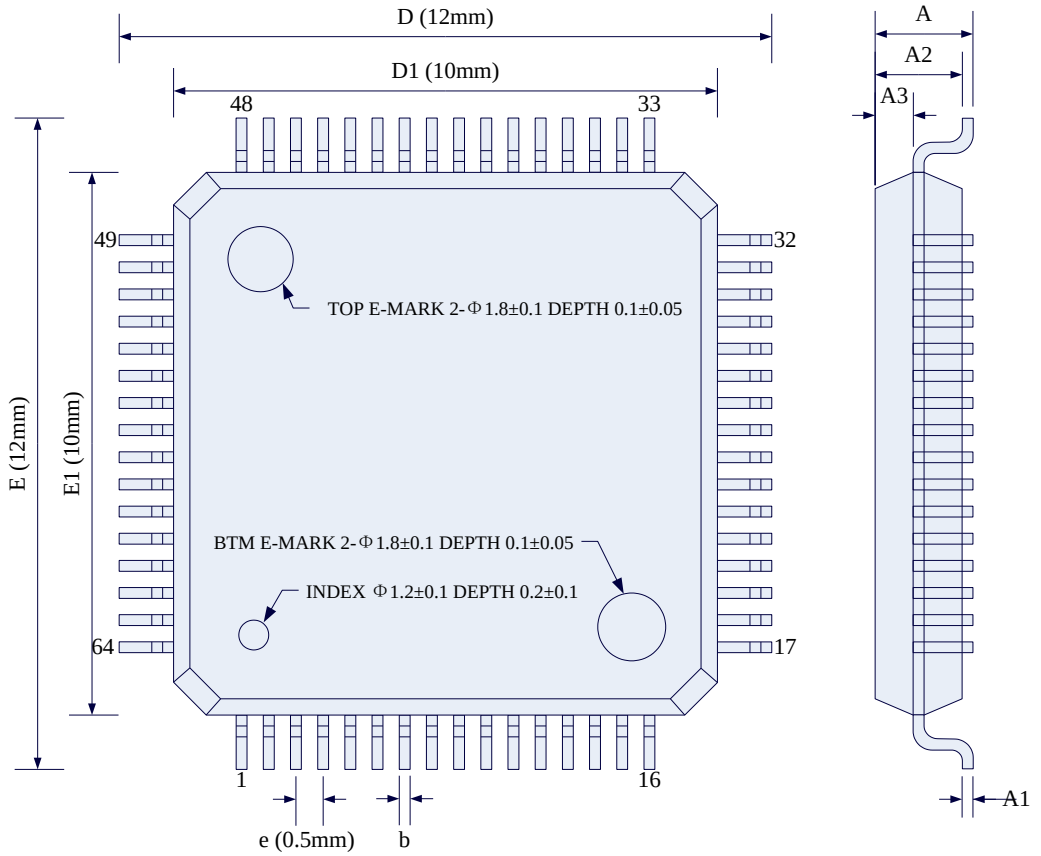




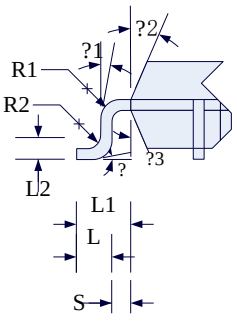
General size			
			mm
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3		0.20REF	
b	0.15	0.20	0.25
D	5.90	6.00	6.10
E	5.90	6.00	6.10
D2	3.95	4.05	4.15
E2	3.95	4.05	4.15
e	0.35	0.40	0.45
L	0.35	0.40	0.45
K		0.20	
R	0.09	-	-

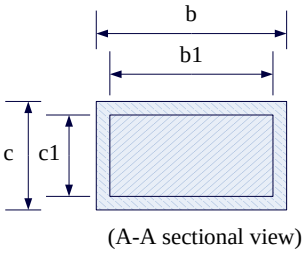
The back metal sheet (substrate) of STC's existing **QFN48**packaged chip is not grounded inside the chip. It can be grounded or ungrounded on the user's PCB board, which will not affect the performance of the chip.

4.4 LQFP64 Package mechanical data（12mm*12mm）



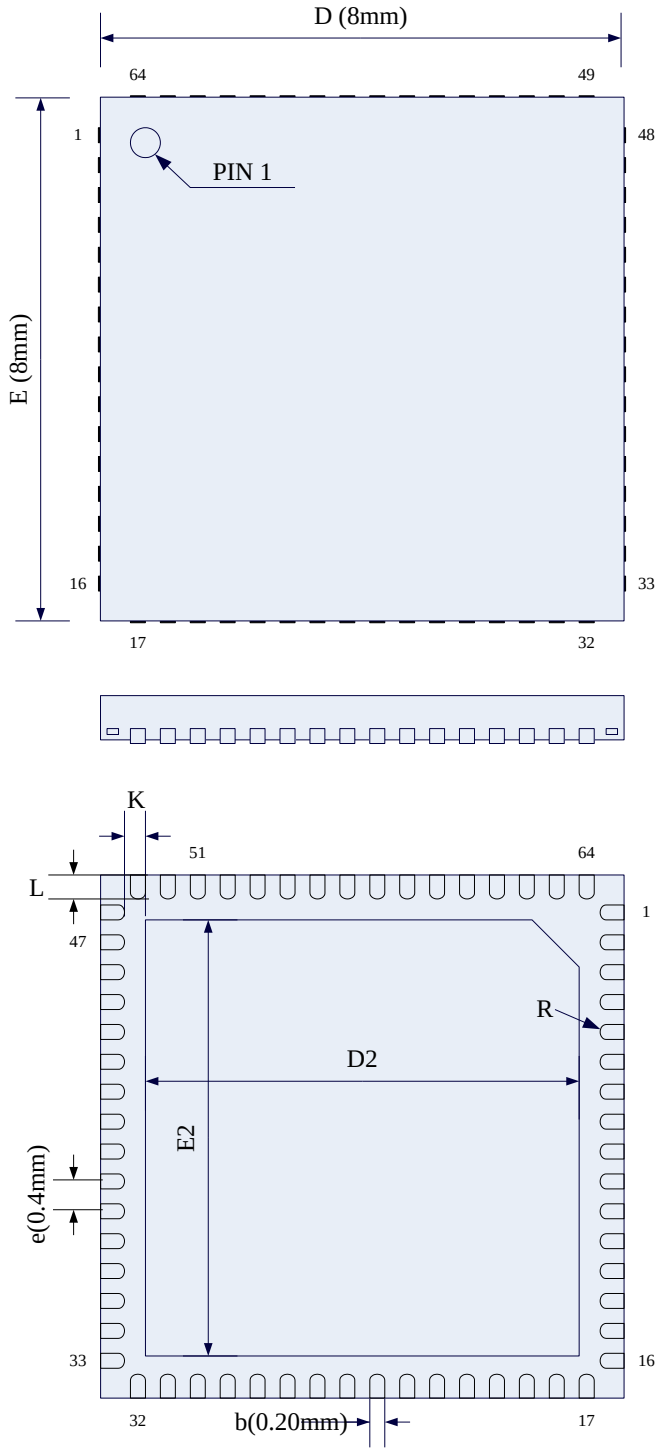
General size			
			mm
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-
?	0°	3.5°	7°
?1	0°	-	-
?2	11°	12°	13°
?3	11°	12°	13°



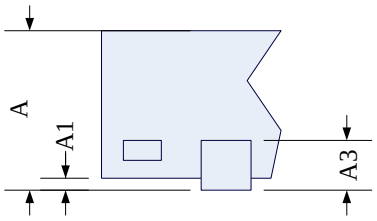


(A-A sectional view)

4.5 QFN64 Package mechanical data（8mm*8mm）

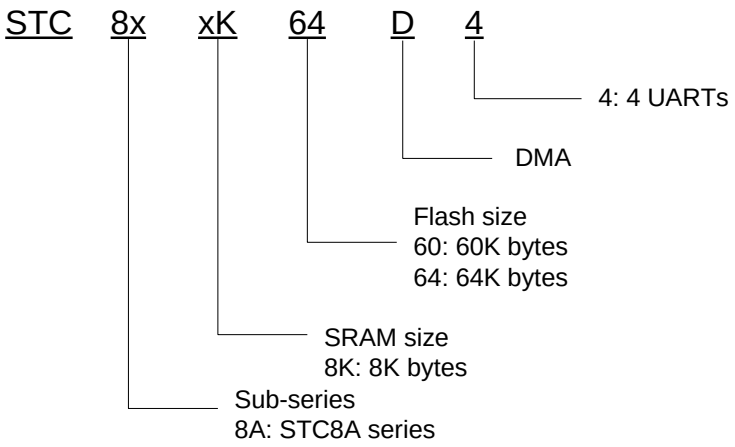


General size			
			mm
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A2	0.50	0.55	0.60
A3		0.20REF	
b	0.15	0.20	0.25
D	7.90	8.00	8.10
E	7.90	8.00	8.10
D2	5.90	6.00	6.10
E2	5.90	6.00	6.10
e	0.30	0.40	0.50
L	0.30	0.40	0.50
K		0.40	
R	0.09	-	-



The back metal sheet (substrate) of STC's existing DFN8 packaged chip is not grounded inside the chip. It can be grounded or ungrounded on the user's PCB board, which will not affect the performance of the chip.

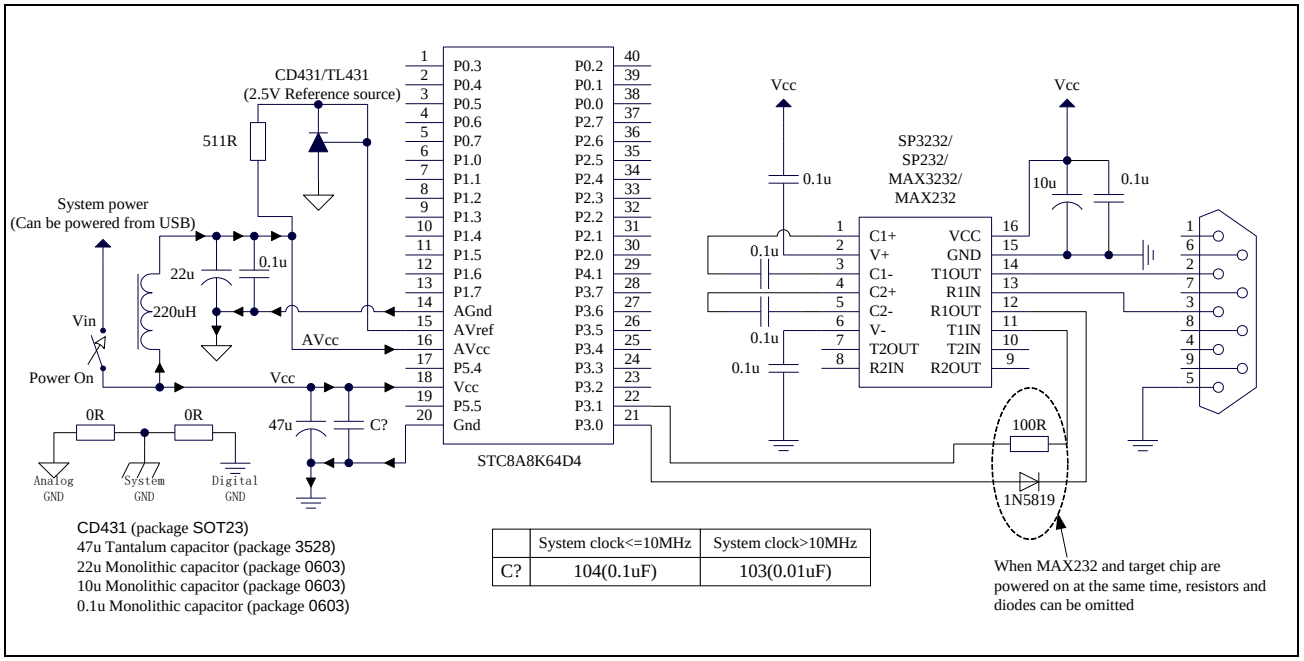
4.6 Naming rules of STC8A8K64D4 family


STC 8x xK 64 D 4
4: 4 UARTs
DMA
Flash size
60: 60K bytes
64: 64K bytes
SRAM size
8K: 8K bytes
Sub-series
8A: STC8A series

5 ISP Download and typical application circuit

5.1 STC8A8K64D4 series ISP download application circuit

5.1.1 Download using RS-232 converter (High precision ADC), Emulation supported

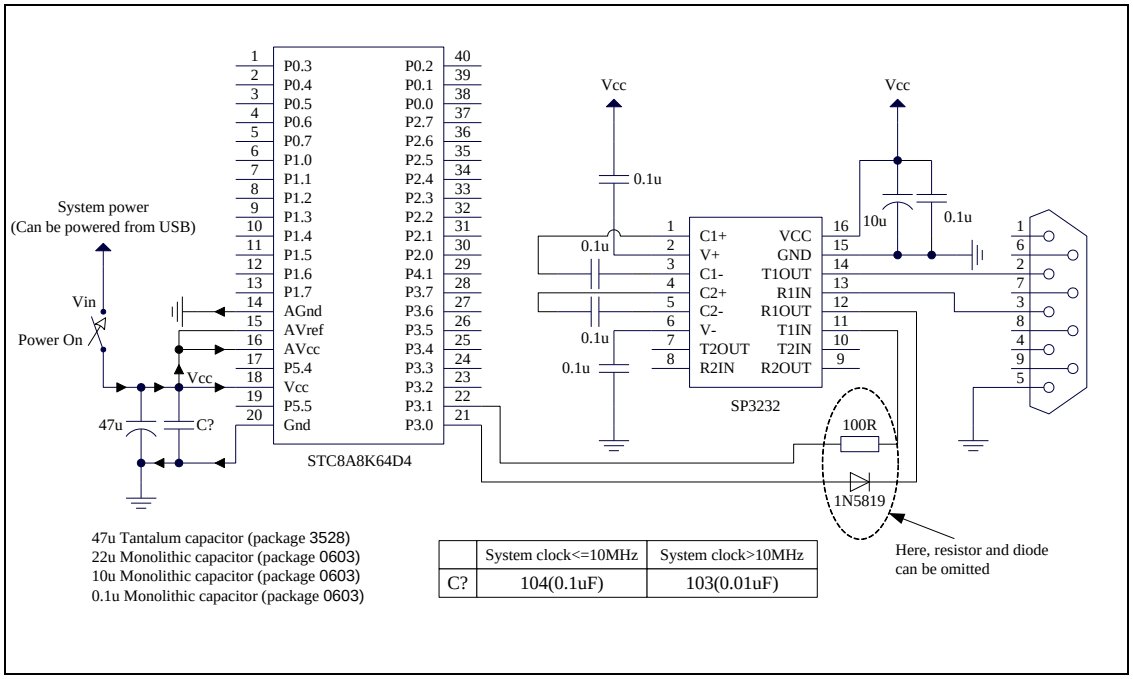


	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP download steps:

1. Power off the target chip.
2. Click the "Download/Program" button in the STC-ISP download software.
3. Power on the target chip.
4. Start ISP download.

5.1.2 Download using RS-232 converter (General precision ADC), Emulation supported

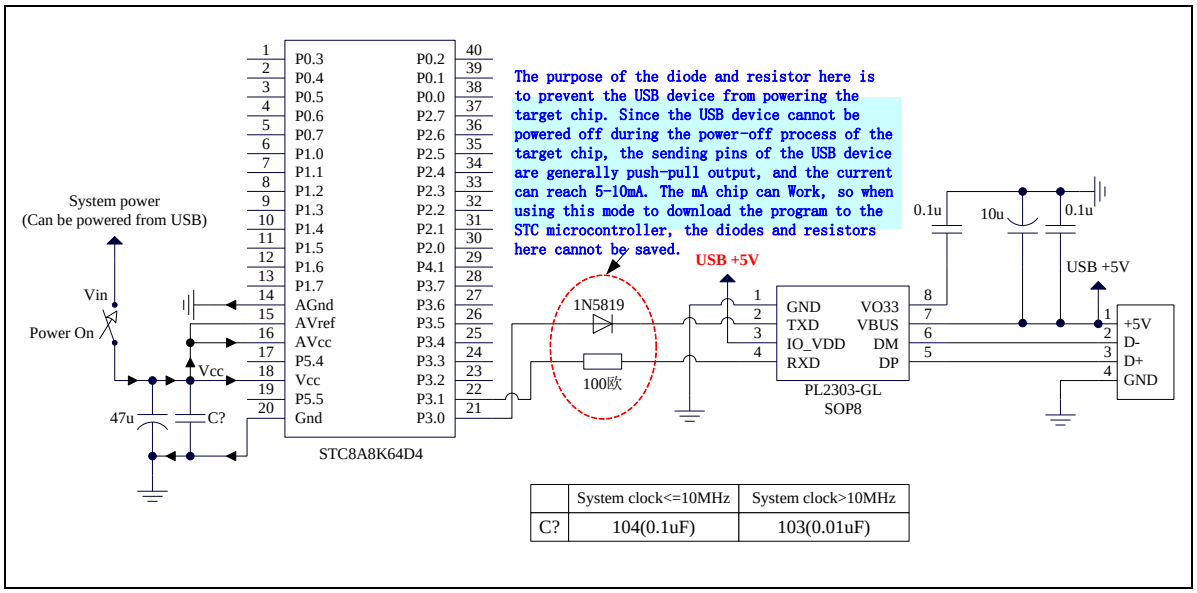


	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP download steps:

1. Power off the target chip.
2. Click the “Download/Program” button in the STC-ISP download software.
3. Power on the target chip.
4. Start ISP download.

5.1.3 Download using PL2303-GL, Emulation supported



	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP download steps:

1. Power off the target chip, be careful not to power off the USB-to-serial chip (such as: CH340, PL2303-GL, etc.)

Note: Some baud rate errors of PL2303-SA are very large, it is recommended to use PL2303-GL

2. Since the sending pins of the USB-to-serial chip are generally strong push-pull outputs, a diode must be connected in series between the P3.0 port of the target chip and the sending pin of the USB-to-serial chip, otherwise the target chip cannot be completely powered off. The goal of powering down the target chip is not reached.

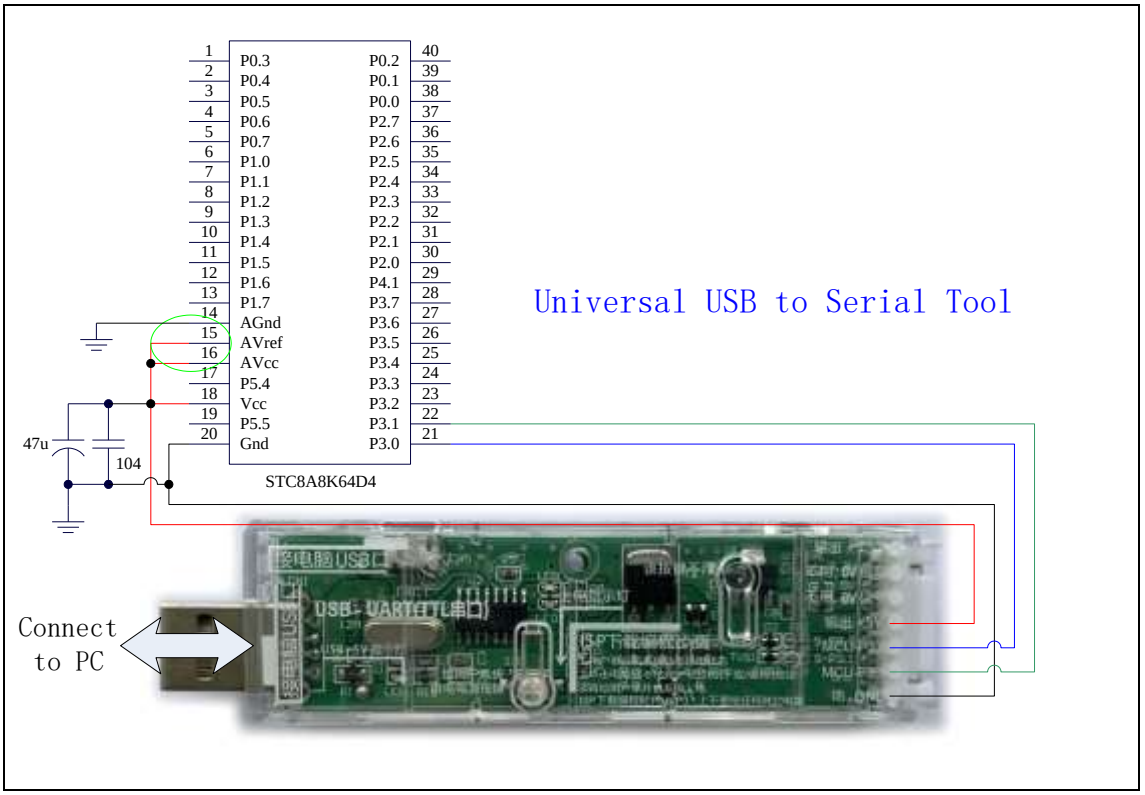
3. Click the "Download/Program" button in the STC-ISP download software

4. Power on the target chip

5. Start ISP download

Note: At present, it has been found that when using the USB cable for ISP download, the USB cable is too thin and the voltage drop on the USB cable is too large, resulting in insufficient power supply during the ISP download. Therefore, please make sure to use the booster USB cable for ISP download.

5.1.4 Download using universal USB to Serial Tool, Emulation supported

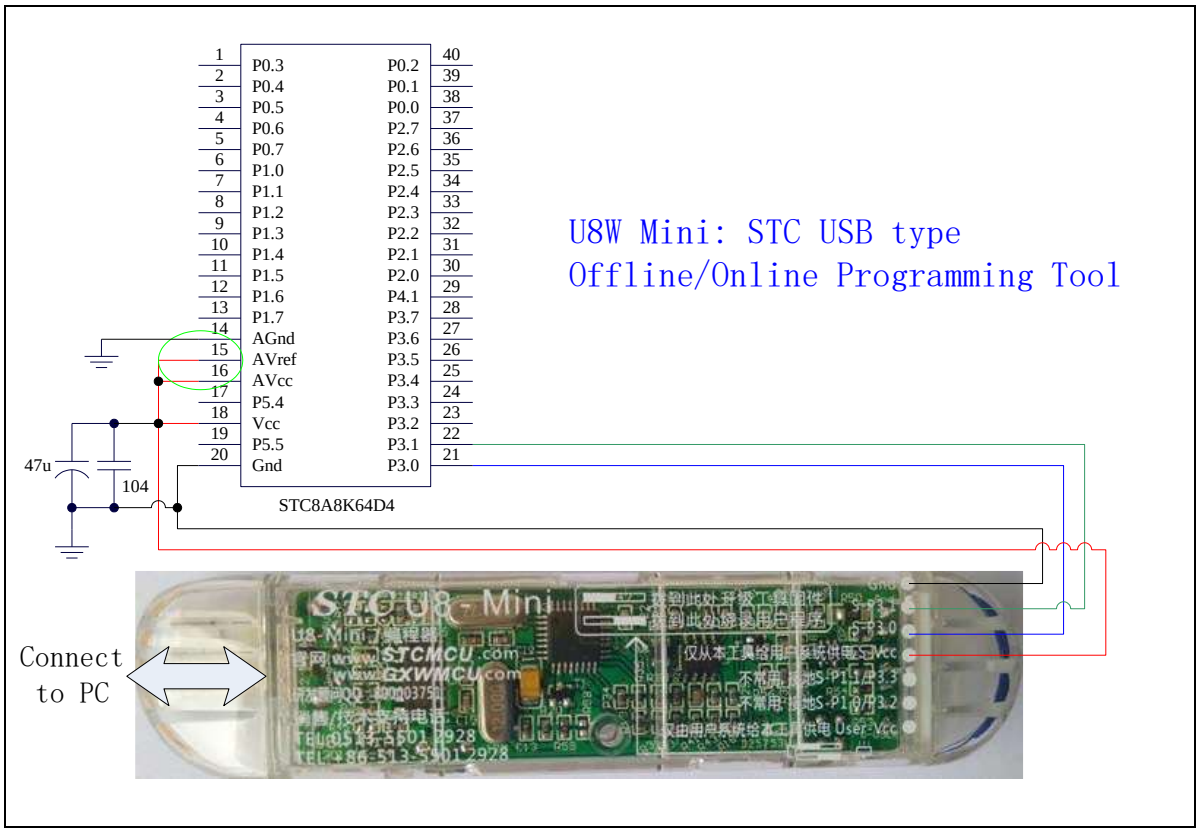


ISP download steps:

1. Connect the universal USB-to-serial tool to the target chip according to the connection method shown in the figure.
2. Press the power button to confirm that the target chip is in a power-off state (the power-on indicator is off). Note: When the tool is powered on for the first time, it will not supply power to the outside world, so if you use the tool for the first time, you can skip this step.
3. Click the "Download/Program" button in the STC-ISP download software
4. Press the power button again to power on the target chip (the power-on indicator is on)
5. Start ISP download.

Note: At present, it has been found that when using the USB cable for ISP download, the USB cable is too thin and the voltage drop on the USB cable is too large, resulting in insufficient power supply during the ISP download. Therefore, please make sure to use the booster USB cable for ISP download.

5.1.5 Download using U8-Mini tool, Support ISP online and offline download, also support emulation

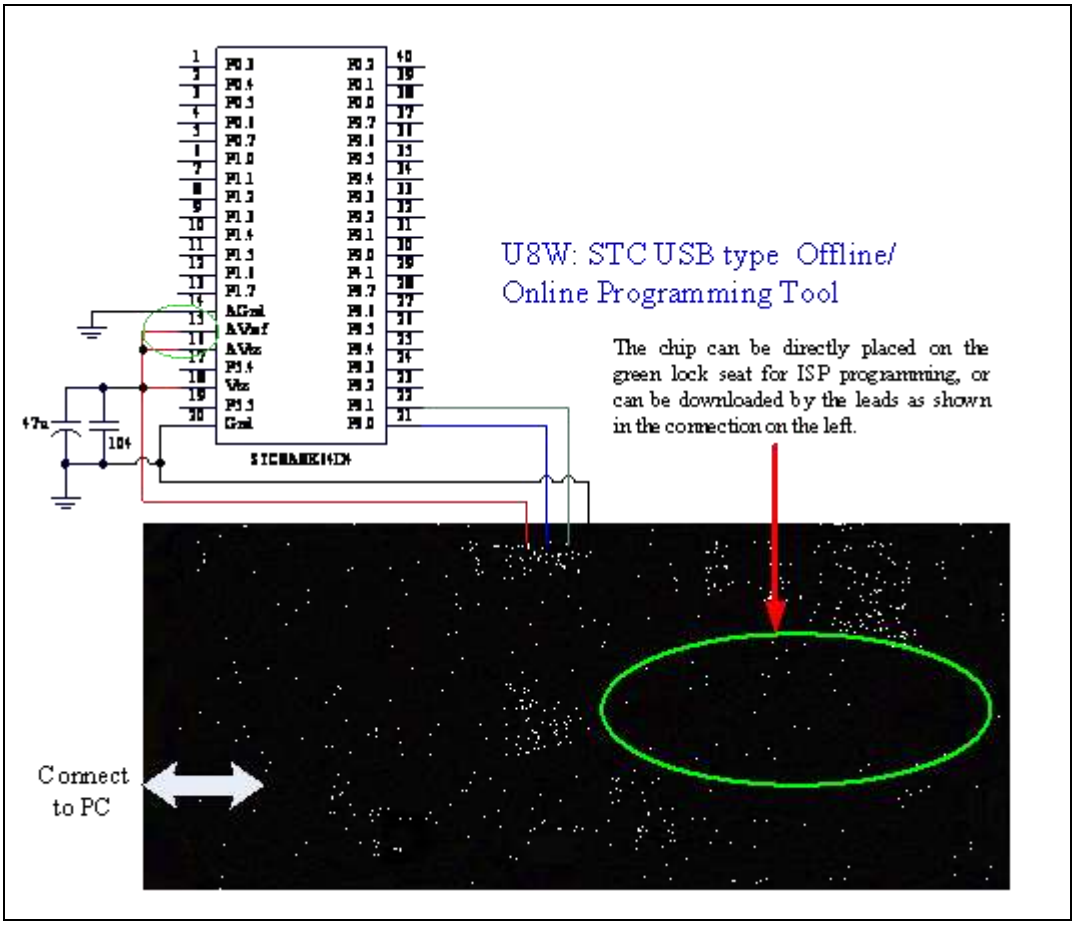


ISP download steps:
1. Connect the U8-Mini and the target chip according to the connection method shown in the figure.
2. Click the "Download/Program" button in the STC-ISP download software
3. Start ISP download
Note: If the U8-Mini is used to power the target system, the total current of the target system cannot be greater than 200mA, otherwise the download will fail.

Note: At present, it has been found that when using the USB cable for ISP download, the USB cable is too thin and the voltage drop on the USB cable is too large, resulting in insufficient power supply during the ISP download. Therefore, please make sure to use the booster USB cable for ISP download.

To emulate using the U8-Mini, you must set the U8-Mini to pass-through mode firstly. The method for U8W/U8W-Mini to realize the USB-to-serial pass-through mode is as follows:
1. Firstly, the U8W/U8W-Mini firmware must be upgraded to v1.37 and above.
2. After the U8W/U8W-Mini is powered on, it is in the normal download mode. At this time, press and hold the Key1 (download) button on the tool and do not release it, then press the Key2 (power) button again, and then release the Key2 (power) button. Release the Key1 (download) button again, and the U8W/U8W-Mini will enter the USB-to-serial pass-through mode. (Press Key1→Press Key2→Release Key2→Release Key1).
3. The U8W/U8W-Mini tool that enters the pass-through mode is just a simple USB to serial port and does not have the offline download function. If you need to restore the original function of the U8W/U8W-Mini, you only need to press the Key2 (power) button again.

5.1.6 Download using U8W tool, Support ISP online and offline download, also support emulation



ISP download steps(Connection method):

1. Connect the U8W and the target chip according to the connection method shown in the figure.
2. Click the "Download/Program" button in the STC-ISP download software
3. Start ISP download

Note: If the U8W is used to power the target system, the total current of the target system cannot be greater than 200mA, otherwise the download will fail.

ISP download steps (on-board mode):

1. Place the chip in the direction that pin 1 is close to the locking wrench and the pins are aligned downwards.
2. Click the "Download/Program" button in the STC-ISP download software
3. Start ISP download.

Note: At present, it has been found that when using the USB cable for ISP download, the USB cable is too thin and the voltage drop on the USB cable is too large, resulting in insufficient power supply during the ISP download. Therefore, please make sure to use the booster USB cable for ISP download.

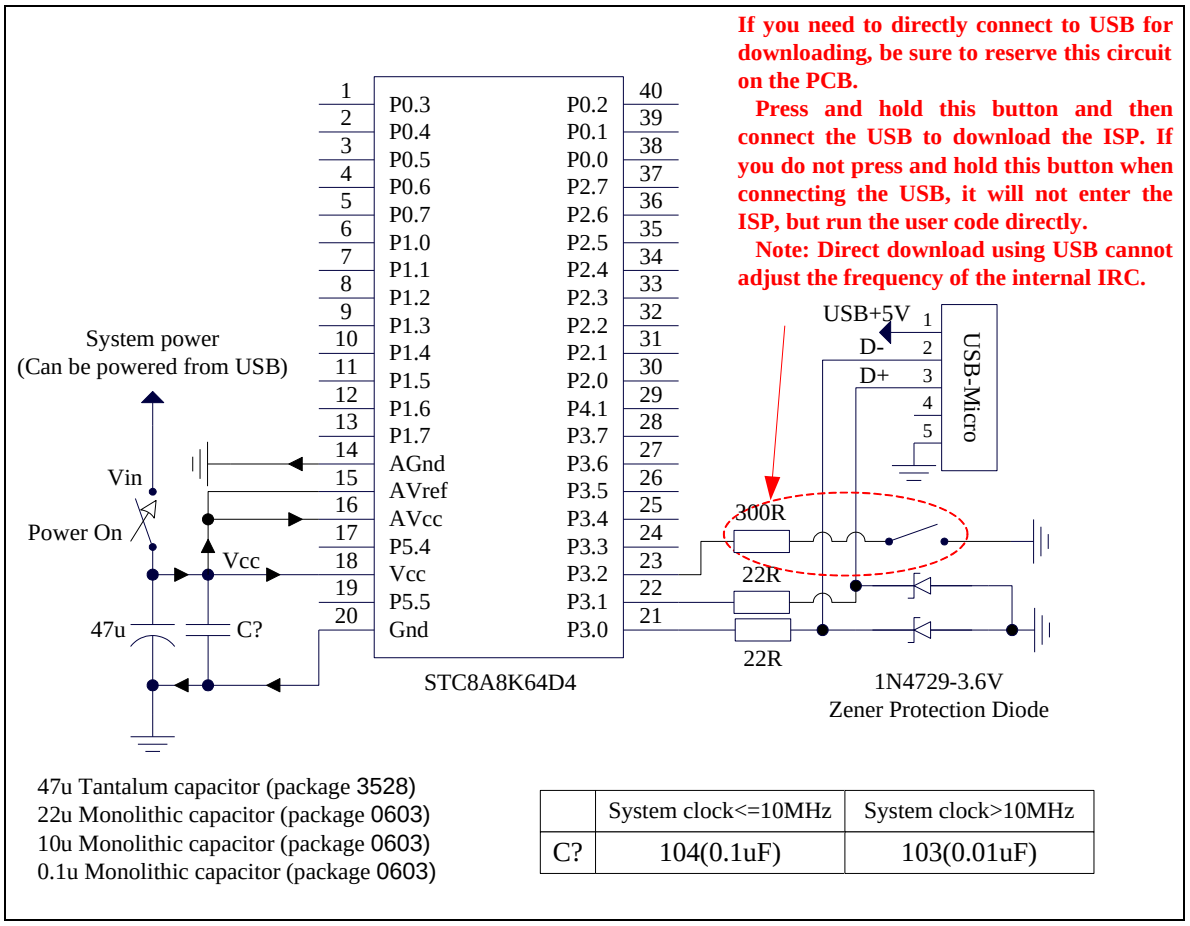
To emulate using the U8W, you must set the U8W to pass-through mode firstly. The method for U8W/U8W-Mini to realize the USB-to-serial pass-through mode is as follows:

1. Firstly, the U8W/U8W-Mini firmware must be upgraded to v1.37 and above.
2. After the U8W/U8W-Mini is powered on, it is in the normal download mode. At this time, press and hold the Key1 (download) button on the tool and do not release it, then press the Key2 (power) button again, and then release the Key2 (power) button. Release the Key1 (download) button again, and the U8W/U8W-Mini will enter the USB-to-serial pass-through mode. (Press Key1→Press Key2→Release Key2→Release Key1).
3. The U8W/U8W-Mini tool that enters the pass-through mode is just a simple USB to serial port and does not have the offline download function. If you need to restore the original function of the U8W/U8W-Mini, you only need to press the Key2

(power) button again.

5.1.7 Software simulation USB ISP download directly, and does not support simulation

Note 1: When using USB to download, you need to connect P3.2 to Gnd for normal download.
Note 2: If USB download is not required, P3.0/P3.1/P3.2 cannot be low at the same time when the chip is reset, otherwise the chip will always be in USB download mode without running user code.

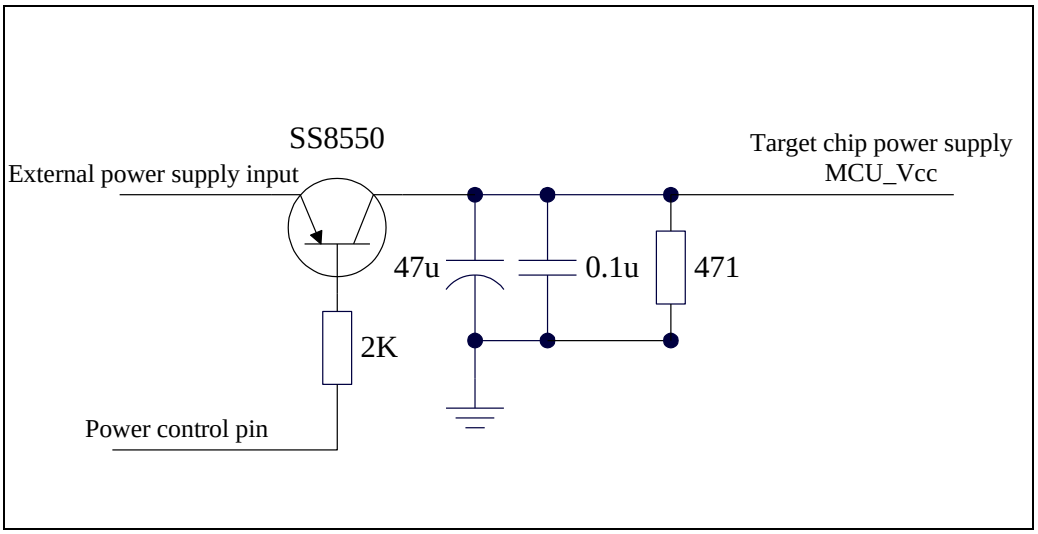


	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

ISP download steps:
1. Power off the target chip.
2. Connect P3.0/P3.1 to the USB port according to the connection method shown in the figure.
3. Short-circuit P3.2 with GND.
4. Power on the target chip and wait for "STC USB Writer (USB1)" to be automatically recognized in the STC-ISP download software.
5. Click the "Download/Program" button in the download software (note: the sequence of operations is different from serial download).
6. Start the ISP download.

Note: At present, it has been found that when using the USB cable for ISP download, the USB cable is too thin and the voltage drop on the USB cable is too large, resulting in insufficient power supply during the ISP download. Therefore, please make sure to use the booster USB cable for ISP download.

5.1.8 Microcontroller Power supply Control Reference Circuit

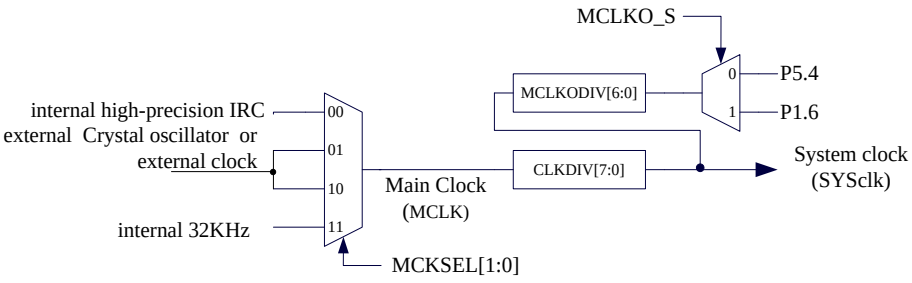


6. Clock, Reset, Power saving mode and Power Management

6.1 System Clock Control

The system clock controller provides the clock sources for the microcontroller's CPU and all peripherals. One of the following three clock sources can be selected as the system clock: internal high-precision IRC, internal 32KHz IRC with large error, external crystal oscillator. Every clock source can be enabled or disabled respectively using programs, as well as internally provide clock divider for the purpose of reducing power consumption.

When the microcontroller enters Power-down mode, the clock controller will shut down all clock sources.



System clock structure diagram

Related registers

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	Clock selection register	FE00H	-						MCKSEL[1:0]		xxxx,xx00
CLKDIV	Clock Division Register	FE01H									nnnn,nnnn
HIRCCR	Internal High speed Oscillator control register	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	External Oscillator control register	FE03H	ENXOSC	XITYPE	XCFILTER[1:0]		GAIN	-	-	XOSCST	00xx,xxx0
IRC32KCR	Internal 32KHz Oscillator control register	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	Main clock output control register	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	Internal IRC start-up and debounce control	FE06H	IRCDB[7:0]								1000,0000

6.1.1 System clock selection register (CKSEL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CKSEL	FE00H	-						MCKSEL[1:0]	

MCKSEL[1:0]: Main clock source selection

MCKSEL[1:0]	Main clock source
00	internal high speed high precision IRC
01	external high speed crystal oscillator
10	-
11	internal 32KHz low speed IRC

6.1.2 Clock Division register (CLKDIV)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

CLKDIV: Main clock dividing factor. The system clock (SYSCLK) is the clock signal of main clock (MCLK) after being divided.

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 set the initial value of this register automatically according to the frequency division factor required for the operating frequency set during the last ISP download.

6.1.3 Internal high speed high precision IRC control register (HIRCCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
HIRCCR	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST

ENHIRC: internal high speed high precision IRC enable bit
0: disable internal high-precision IRC
1: enable internal high-precision IRC
HIRCST: internal high speed high precision IRC frequency stability flag (read-only)
After the internal IRC is enabled from the stopped state, it must take some time for the frequency of the oscillator to become stable. The clock controller will set the HIRCST flag automatically after the internal oscillator frequency stabilizes. When the user program needs to switch the clock to internal IRC, ENHIRC must be set at first to enable the oscillator and then keep polling the oscillator stable flag HIRCST until the flag changes to 1.

6.1.4 External Oscillator control register (XOSCCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	FE03H	ENXOSC	XITYPE	XCFILTER[1:0]		NMXCG	-	-	XOSCST

ENXOSC: external oscillator enable bit
0: disable external oscillator
1: enable external oscillator
XITYPE: external clock source type
0: The external clock source is the external clock signal (or active crystal). The signal source only needs to be connected to the XTALI(P1.7) of microcontroller. (At this time, P1.6 is fixed in high-impedance input mode, which can be used to read external digital signals or as ADC input, but it is generally not recommended to be used, because the high-frequency oscillation signal of the adjacent P1.7 will affect P1.6 signal.)
1: The external clock source is a crystal oscillator which is connected to XTALI (P1.7) and XTALO (P1.6) of microcontroller.
XOSCST: external crystal oscillator frequency stability flag (read-only)
After the external crystal oscillator is enabled from the stopped state, it must take some time for the frequency of the oscillator to become stable. The clock controller will set the XOSCST flag automatically after the oscillator frequency stabilizes. When the user program needs to switch the clock to external crystal oscillator, ENXOSC must be set at first to enable the oscillator and then keep polling the oscillator stable flag XOSCST until the flag changes to 1.
XCFILTER[1:0]: External crystal oscillator anti-interferencecontrol register.
00: When the external crystal oscillator frequency is 48M and below.
01: When the external crystal oscillator frequency is 24M and below
1x: When the external crystal oscillator frequency is 12M and below
NMXCG: External crystal oscillator oscillator gain control bits
0: disable oscillator gain (low gain)
1: enable oscillator gain (high gain)

6.1.5 Internal 32KHz low speed IRC control register (IRC32KCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IRC32KCR	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

ENIRC32K: internal 32KHz low speed IRC enable bit
0: disable internal 32KHz low speed IRC
1: enable internal 32KHz low speed IRC
IRC32KST: internal 32KHz low speed IRC frequency stability flag (read-only)
After the internal 32KHz low speed IRC is enabled from the stopped state, it must take some time for the frequency

of the oscillator to become stable. The clock controller will set the IRC32KST flag automatically after the internal oscillator frequency stabilizes. When the user program needs to switch the clock to the internal 32KHz low speed IRC, ENIRC32K must be set at first to enable the oscillator and then keep polling the oscillator stable flag IRC32KST until the flag changes to 1.

6.1.6 Main 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]: Main clock output division factor

(**Note: The clock source of main clock output is system clock divided by CLKDIV**)

MCLKODIV[6:0]	Divided system clock output frequency
0000000	No clock out
0000001	SYSClk/1
0000010	SYSClk /2
0000011	SYSClk /3
…	…
1111110	SYSClk /126
1111111	SYSClk /127

MCLKO_S: Main clock output pin selection

0: Main clock output to P5.4

1: Main clock output to P1.6

6.2 STC8A8K64D4 series internal IRC frequency adjustment

All STC8A8K64D4 series of microcontrollers integrate a high-precision internal IRC oscillator. The ISP download software will automatically adjust the frequency according to the frequency selected / set by the user when users download user program using ISP. The general frequency value can be adjusted below ±0.3%. The temperature drift of the adjusted frequency can be -1.35% ~ 1.30% within the full temperature range (-40℃~85℃).

The internal IRC of the STC8A8K64D4 series has four frequency bands whose center frequencies are 6MHz, 10MHz, 27MHz and 44MHz respectively. The adjustment range of the 6M band is about 4.5MHz to 8MHz. The adjustment range of the 10M band is about 7.5MHz to 13.5MHz. The adjustment range of the 27M band is about 20.5MHz to 36MHz. The adjustment range of the 44M band is about 29MHz to 55MHz. Note: Different chips or different manufacturing batches may have manufacturing errors of about 5%. It is recommended that users set the IRC frequency not higher than 45MHz during ISP download.

Note: For general users, the adjustment of the internal IRC frequency can be ignored because the frequency adjustment is automatically completed when the ISP is downloaded. Therefore, if the user does not need to adjust the frequency by itself, the following four registers cannot be modified at will, otherwise the operating frequency may change.

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

Internal IRC frequency adjustment is mainly adjusted using the following 4 registers.

Related registers

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IRCBAND	IRC band selection register	9DH	-	-	-	-	-	-	SEL[1:0]		xxxx,xxxn
LIRTRIM	IRC frequency trim register	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		xxxx,xxnn
IRTRIM	IRC frequency adjustment register	9FH	IRTRIM[7:0]								nnnn,nnnn
CLKDIV	Clock Divide Register	FE01H	CLKDIV[7:0]								nnnn,nnnn

6.2.1 IRC band selection register (IRCBAND)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	9DH	-	-	-	-	-	-	-	SEL

SEL[1:0]: band selection

00: Select 6MHz band
01: Select 10MHz band
10: Select 27MHz band
11: Select 44MHz band

6.2.2 Internal 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 can adjust 256 levels of IRC frequency. The frequency value adjusted by each level is linearly distributed as a whole, and there may be local fluctuations. Macroscopically, the frequency adjusted by each stage is about 0.24%, that is, the frequency when the IRTRIM is (n + 1) is about 0.24% faster than the frequency when the IRTRIM is (n). However, not every level of IRC frequency adjustment is 0.24% (the maximum value of the adjusted frequency of each level is about 0.55%, the minimum value is about 0.02%, and the overall average value is about 0.24%), so it will cause local fluctuations.

6.2.3 Internal IRC frequency trim register (LIRTRIM)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LIRTRIM	9EH	-	-	-	-	-	-	-	IRTRIM

LIRTRIM: Internal high precision IRC frequency trim register

6.2.4 Clock Divide Register (CLKDIV)

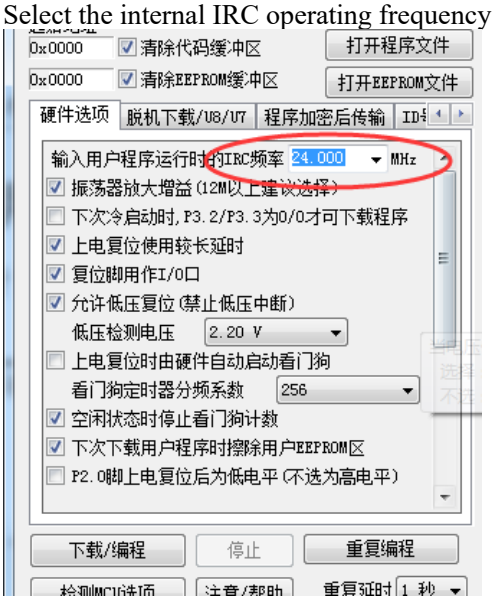
Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	FE01H								

CLKDIV: Frequency division factor of the main clock. The system clock SYSCLK is a clock signal obtained by dividing the main clock MCLK.

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

6.2.5 Example of fine-tuning to get a user frequency of 3MHz

To get a frequency of 3MHz, you can use the method of 24MHz divided by 8.
Select the internal IRC operating frequency as 24MHz firstly when downloading the ISP, as shown in the figure below.



Then select the internal IRC as the clock source in the code and use the CLKDIV register to divide by 8.

C language code

```
// Operating frequency for test is 24MHz

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

#define    CKSEL      (*(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        IRTRIM     =    0x9f;

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

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

```
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    CKSEL = 0x00;                                   // Select internal IRC (default)
    CLKDIV = 0x08;                                  // Clock divided by 8
    P_SW2 = 0x00;

    IRTRIM++;   // Fine-tune the IRC frequency up 3 ‰ (pay attention to judging the boundary)
//  IRTRIM--;   // Fine-tune the IRC frequency down 3 ‰ (pay attention to judging the boundary)

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 24MHz

P_SW2       DATA    0BAH
IRTRIM      DATA    09FH

CKSEL       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                  ; Select internal IRC
        MOV     DPTR,#CKSEL
        MOVX    @DPTR,A
        MOV     A,#08H                  ; Clock divided by 8
        MOV     DPTR,#CLKDIV
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        INC     IRTRIM  ;IRC Fine-tune frequency up 3 ‰ (pay attention to judging boundaries)
;       DEC     IRTRIM  ;IRC Fine-tune frequency down 3 ‰ (pay attention to judging boundaries)

        JMP     $

        END
```

6.3 System reset

There are two types of resets in STC8A8K64D4 series of microcontrollers, hardware reset and software reset.

When hardware reset occurs, all registers are reset to their original values and the system rereads all hardware options. At the same time, after being powered on, the system will wait for some time according to the hardware power-on wait time option set. Hardware reset includes,

- Power-on reset, POR, about 1.7V
- Low-voltage detection reset, LVD-RESET (2.0V, 2.4V, 2.7V, 3.0V)
- RST pin reset (Low-level reset)
- Watch-Dog-Timer reset

When software reset occurs, all the registers values are reset to the initial value except that the clock-related registers remain unchanged. Software reset does not re-read all hardware options. Software reset mainly includes,

- Write SWRST bit in IAP_CONTR register to trigger 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 Registe	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		x0x0,xx00

6.3.1 Watch dog timer reset（WDT_CONTR）

In industrial/automotive electronic control/aerospace need high reliability in the system, in order to prevent "system in exceptional cases, the disturbance of MCU/CPU program run fly, resulting in abnormal system for a long time work", is usually introduced watchdog, if the MCU/CPU is not within the stipulated time visit watchdog, according to the requirement as MCU/CPU in abnormal state, the watchdog will force MCU/CPU reset, enables the system to perform user program from the very beginning.

The STC8 series Guard Dog reset is one of the hardware reset in thermal boot reset. STC8 series SCM introduces this function, which makes the reliability design of SCM system more convenient and simple. After the reset state of STC8 series watchdog, the system is fixed to start from THE ISP monitor area, independent of the SWBS of IAP_CONTR register before the reset of watchdog (Note: This is different from 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 : WDT reset flag.

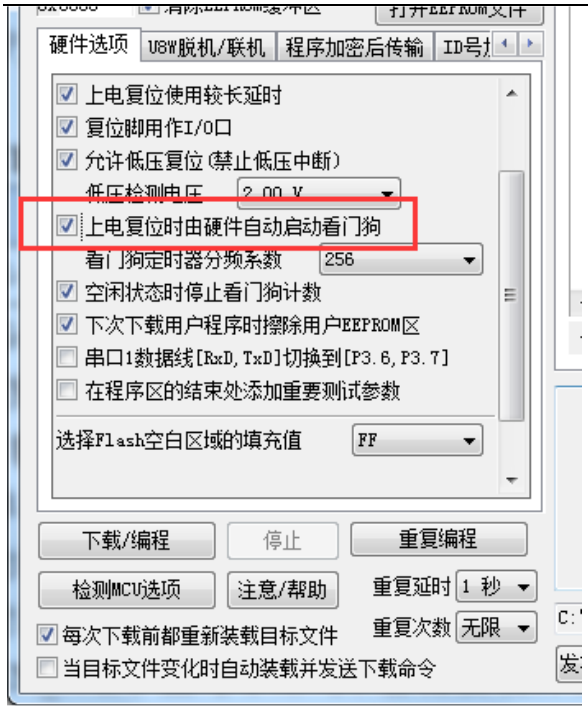
When WDT overflows, this bit is set by hardware automatically. This bit should be cleared by software.

EN_WDT: WDT enable bit.

0: No operation

1: WDT is started.

Note: The watchdog timer can be started by software or hardware automatically. Once the watchdog timer is started, the software cannot be shut down. The SCM must be recharged before it can be shut down. Software to start the watchdog only needs to write 1 to the EN_WDT bit. If you need the hardware to boot the watchdog, you need to set it up at your ISP when you download it, as shown in the figure below:



CLR_WDT: WDT clear bit.

0: No operation

1: WDT is cleared. This bit will be cleared by hardware automatically.

IDL_WDT: WDT control bit in IDLE mode.

0: WDT is disabled in IDLE mode.

1: WDT is enabled in IDLE mode, and the WDT will continue counting.

WDT_PS[2:0]: Watchdog timer clock division factor

WDT_PS[2:0]	division factor	Overflow time @12MHz	Overflow time @20MHz
000	2	≈ 65.5 ms	≈ 39.3 ms
001	4	≈ 131 ms	≈ 78.6 ms
010	8	≈ 262 ms	≈ 157 ms
011	16	≈ 524 ms	≈ 315 ms
100	32	≈ 1.05 s	≈ 629 ms
101	64	≈ 2.10 s	≈ 1.26 s
110	128	≈ 4.20 s	≈ 2.52 s
111	256	≈ 8.39 s	≈ 5.03 s

The WDT overflow time is determined by the following equation:

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

6.3.2 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 boot selection bit

0: The microcontroller executes the code from user program space (main flash memory) after the software reset. The data in the user data space remains unchanged.

1: The microcontroller executes the code from ISP space after the software reset. The data in the user data space is initialized.

SWRST: Software reset trigger bit.

0: No operation

1: Trigger software reset.

Writing 60H to the IAP control register can achieve the effect of cold start of the microcontroller

6.3.3 Low voltage reset (RSTCFG)

RSTCFG (Reset Configuration Register)

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

ENLVR: Low voltage detection reset enable bit

0: Disable low voltage detection reset. When the system detects a low-voltage event, a low-voltage interrupt will occur.

1: Enable low voltage detection reset. When the system detects a low-voltage event, it will reset automatically.

P54RST: RST pin function selection bit

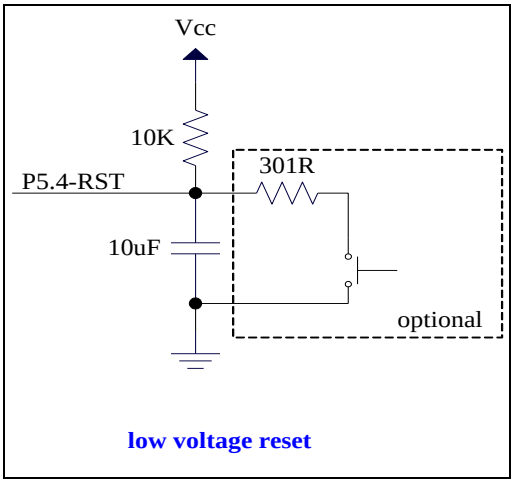
0: RST pin is used as common I/O (P5.4).

1: RST pin is used as reset pin. **(Low-level reset)**

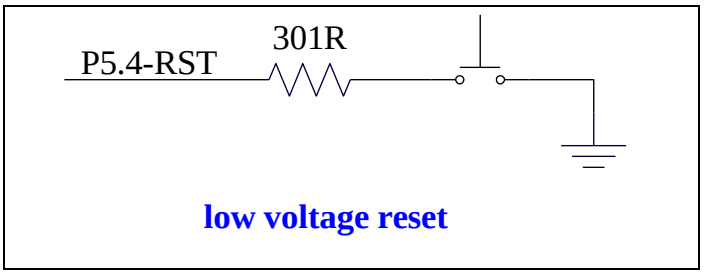
LVDS[1:0] : Low voltage detection threshold voltage setting bits

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

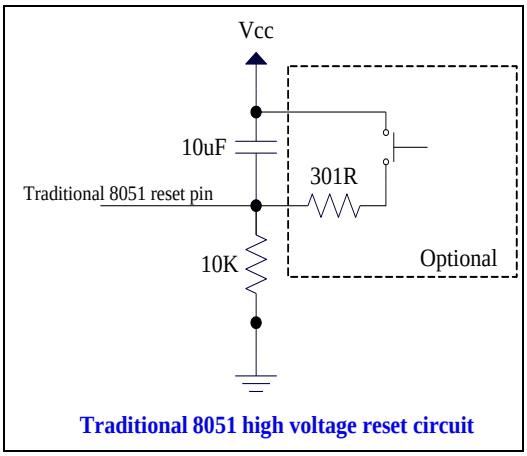
6.3.4 Low voltage power-on reset reference circuit (generally not required)



6.3.5 Low voltage button reset reference circuit



6.3.6 Traditional 8051 high voltage power-on reset reference circuit

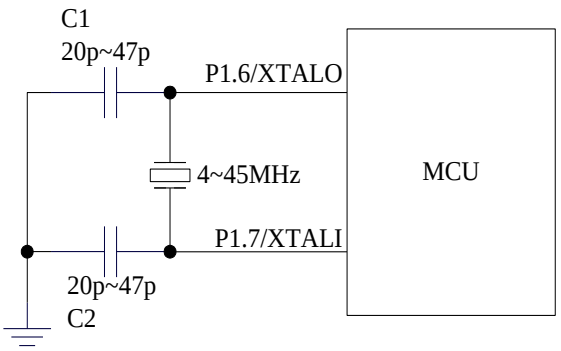


Traditional 8051 high voltage reset circuit

The picture above shows the high-level reset circuit of the traditional 8051. The reset of the **STC8A8K64D4** is a low-level reset, which is different from the traditional reset circuit.

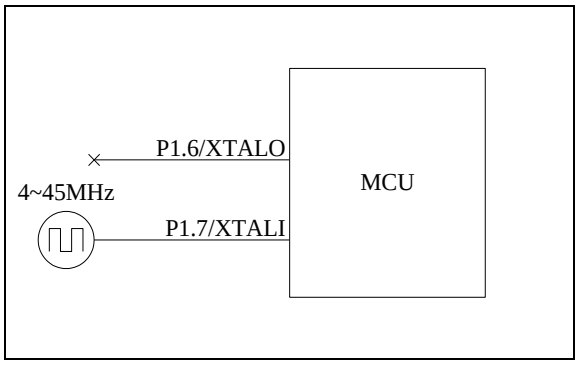
6.4 External crystal oscillator and external clock circuit

6.4.1 External crystal input circuit



Note: The two capacitors C1 and C2 must not be omitted. Without these two capacitors, the crystal oscillator may not oscillate. In addition, even if the crystal oscillator can start to oscillate, the MCU will increase the power consumption of 5~8mA.

6.4.2 External clock input circuit (P1.6 cannot be used as general I/O)



6.5 Clock stop / 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

6.5.1 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, it is set by the hardware automatically and an interrupt request to the CPU ocuurs. It should be cleared by user software.

POF: Power-On reset flag. It is set by the hardware automatically when power-off-on action occurs.

PD: Clock stop mode / Power-Down mode / Power stop mode control bit

0: No operation.

1: Make the microcontroller entering Clock stop mode / Power-Down mode / Power stop mode. CPU and all peripherals stop working. It is cleared by hardware automatically after the microcontroller wakes up. (Note: In the clock stop mode, the CPU and all peripherals stop working, but the data in the SRAM and XRAM remain unchanged.)

IDL: IDLE mode control bit

0: No operation.

1: Make the microcontroller entering IDLE mode. CPU stops working and all peripherals keep working. It is cleared by hardware automatically after the microcontroller wakes up.

Note: In the power saving mode when the clock is stopped, it is not recommended to start the LVD and comparator, or hardware system will automatically start internal high precision 1.19 V reference source which has a corresponding temperature drift and calibration circuit, and leads to about 300uA extra power consumption. After MCU enters into the clock stopped vibration mode, the working voltage of 3.3V only takes about 0.4uA current, so it is not recommended to open the LVD and comparator when the MCU enters the clock stop mode. If it is really needed, it is recommended to turn on the power-down wake-up timer. The power-down wake-up timer will only increase the power consumption of about 1.4uA, which is acceptable for general systems. The MCU is awakened every 5 seconds by the power-down wake-up timer. After wake-up LVD, comparator and ADC can be used to detect the external battery voltage. The detection took about 1ms. Then it enters the clock shutdown/power-saving mode. In this way, the increased average current was less than 1uA, and the overall power consumption was about 2.8uA (0.4uA + 1.4uA +1uA).

6.6 Power-down wake-up timer

The internal power-down wake-up timer is a 15-bit counter (composed of {WKTCH[6:0],WKTCL[7:0]}), which is used to wake up an MCU in power off mode.

6.6.1 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: wake-up timer enable bit

0: No operation

1: WKT is started.

If the built-in power-down wake-up dedicated timer of STC8 series microcontrollers is enabled (set the WKTEN bit in the WKTCH register to 1 through software), when the MCU enters the power-down mode/stop mode, the power-down wake-up dedicated timer starts counting, when the count value is equal to When the values set by the user are equal, the power-down wake-up dedicated timer will wake up the MCU. After the MCU wakes up, the program executes from the statement next to the statement that set the MCU to enter the power-down mode last time. After waking up from power down, the sleep time of the MCU in power down mode can be obtained by reading the contents of WKTCH and WKTCL.

Please note here: The value written by the user in the register {WKTCH[6:0], WKTCL[7:0]} must be one less than the actual count value. If the user needs to count 10 times, write 9 into the register {WKTCH[6:0], WKTCL[7:0]}. Similarly, if the user needs to count 32767 times, he should write 7FFEH (ie 32766) in {WKTCH[6:0], WKTCL[7:0]}. (Count value 0 and count value 32767 are internal reserved values and cannot be used by users). The internal power-down wake-up timer has its own internal clock, and the time for the power-down wake-up timer to count once is determined by this clock. The clock frequency of the internal power-down wake-up timer is about 32KHz, and the error is relatively large. Users can read the contents of RAM area F8H and F9H (F8H stores the high byte of frequency, F9H stores the low byte) to obtain the clock frequency recorded by the internal power-down wake-up dedicated timer when it leaves the factory.

The formula for calculating the counting time of the dedicated timer for power-down wake-up is as follows: (F*wt* is the clock frequency of the dedicated timer for internal power-down wake-up we obtained from RAM area F8H and F9H):

$$\text{Power down wakeup timer timing time} = \frac{10_6 \times 16 \times \text{count times}}{F_{wt}} \text{ (us)}$$

Assuming Fwt=32KHz, there are:

{WKTCH[6:0],WKTCL[7:0]}	Counting time of dedicated timer for wake-up after power failure
1	$10^6 \div 32K \times 16 \times (1+1) \approx 1ms$
9	$10^6 \div 32K \times 16 \times (1+9) \approx 5ms$
99	$10^6 \div 32K \times 16 \times (1+99) \approx 50ms$
999	$10^6 \div 32K \times 16 \times (1+999) \approx 0.5s$
4095	$10^6 \div 32K \times 16 \times (1+4095) \approx 2s$
32766	$10^6 \div 32K \times 16 \times (1+32767) \approx 16s$

6.7 Example Routines

6.7.1 System Clock Soure Selection

C language code

```
// Operating frequency for test is 11.0592MHz

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

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

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

    P_SW2 = 0x80;
    CKSEL = 0x00;                   //Select internal IRC (default)
    P_SW2 = 0x00;

/*
    P_SW2 = 0x80;
    XOSCCR = 0xc0;                  //Start external crystal
    while (!(XOSCCR & 1));          //Waiting for the clock to stabilize
    CLKDIV = 0x00;                  //Clock is not divided
    CKSEL = 0x01;                   //Select external crystal
    P_SW2 = 0x00;
*/
```

```
/*
    P_SW2 = 0x80;
    IRC32KCR = 0x80;                    //Start internal 32KHz IRC
    while (!(IRC32KCR & 1));            //Waiting for the clock to stabilize
    CLKDIV = 0x00;                      //Clock is not divided
    CKSEL = 0x03;                       //Select internal 32KHz
    P_SW2 = 0x00;
*/
    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

CKSEL       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              ;Select internal IRC (default)
            MOV     DPTR,#CKSEL
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
```

```
;               MOV         P_SW2,#80H
;               MOV         A,#0C0H             ;Start external crystal
;               MOV         DPTR,#XOSCCR
;               MOVX        @DPTR,A
;               MOVX        A,@DPTR
;               JNB         ACC.0,$-1           ;Waiting for the clock to stabilize
;               CLR         A                   ;Clock is not divided
;               MOV         DPTR,#CLKDIV
                MOVX        @DPTR,A
;               MOV         A,#01H              ;Select external crystal
;               MOV         DPTR,#CKSEL
;               MOVX        @DPTR,A
;               MOV         P_SW2,#00H

;               MOV         P_SW2,#80H
;               MOV         A,#80H              ;Start internal 32KHz IRC
;               MOV         DPTR,#IRC32KCR
;               MOVX        @DPTR,A
;               MOVX        A,@DPTR
;               JNB         ACC.0,$-1           ;Waiting for the clock to stabilize
;               CLR         A                   ;Clock is not divided
;               MOV         DPTR,#CLKDIV
;               MOVX        @DPTR,A
;               MOV         A,#03H              ;Select internal 32KHz
;               MOV         DPTR,#CKSEL
;               MOVX        @DPTR,A
;               MOV         P_SW2,#00H

                JMP         $

                END
```

6.7.2 Main Clock Output

C language code

```
// Operating frequency for test is 11.0592MHz

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

#define     MCLKOCR     (*(unsigned char volatile xdata *)0xfe05)

sfr         P_SW2       =   0xba;

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

void main()
```

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

    P_SW2 = 0x80;
//  MCLKOCR = 0x01;                 //Main clock output to P5.4
//  MCLKOCR = 0x02;                 //Divide the main clock by 2 and output to P5.4
    MCLKOCR = 0x04;                 //Divide the main clock by 4 and output to P5.4
//  MCLKOCR = 0x84;                 //Divide the main clock by 4 and output to P1.6
    P_SW2 = 0x00;

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

MCLKOCR     EQU     0FE05H

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,#01H          ;Main clock output to P5.4
;       MOV     A,#02H          ;Divide the main clock by 2 and output to P5.4
        MOV     A,#04H          ;Divide the main clock by 4 and output to P5.4
;       MOV     A,#84H          ;Divide the main clock by 4 and output to P1.6
        MOV     DPTR,#MCLKOCR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        JMP     $

        END
```

6.7.3 Application of Watch-dog Timer

C language code

```
// Operating frequency for test is 11.0592MHz

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

sfr     WDT_CONTR   =   0xc1;
sbit    P32         =   P3^2;

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

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

//  WDT_CONTR = 0x23;           //Watchdog enabled, overflow time is about 0.5s
    WDT_CONTR = 0x24;           //Watchdog enabled, overflow time is about 1s
//  WDT_CONTR = 0x27;           //Watchdog enabled, overflow time is about 8s
    P32 = 0;                    //Test port
```

```
    while (1)
    {
//      WDT_CONTR = 0x33;                   // Clear watchdog timer, otherwise system reset
        WDT_CONTR = 0x34;                   // Clear watchdog timer, otherwise system reset
//      WDT_CONTR = 0x37;                   // Clear watchdog timer, otherwise system reset

        Display();                          //Call Display module
        Scankey();                          //Call Key scan module
        MotorDriver();                      //Call Motor drive module
    }
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

WDT_CONTR   DATA    0C1H

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     WDT_CONTR,#23H      ;Watchdog enabled, overflow time is about 0.5s
            MOV     WDT_CONTR,#24H      ;Watchdog enabled, overflow time is about 1s
;           MOV     WDT_CONTR,#27H      ;Watchdog enabled, overflow time is about 8s
            CLR     P3.2                ;Test port
LOOP:
;           MOV     WDT_CONTR,#33H      ; Clear watchdog timer, otherwise system reset
            MOV     WDT_CONTR,#34H      ; Clear watchdog timer, otherwise system reset
;           MOV     WDT_CONTR,#37H      ; Clear watchdog timer, otherwise system reset

            LCALL   DISPLAY             ;Call Display module
```

```
        LCALL   SCANKEY         ;Call Key scan module
        LCALL   MOTORDRIVER     ;Call Motor drive module
        JMP     LOOP

        END
```

6.7.4 User Defined Downloading by Using Software Reset

C language code

```
// Operating frequency for test is 11.0592MHz

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

sfr     IAP_CONTR   =   0xc7;
sbit    P32         =   P3^2;
sbit    P33         =   P3^3;

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

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

    P32 = 1;                        //Test port
    P33 = 1;                        //Test port

    while (1)
    {
        if (!P32 && !P33)
        {
            IAP_CONTR |= 0x60;      //Reset to ISP when P3.2 and P3.3 are both 0
        }
    }
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

IAP_CONTR   DATA    0C7H

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

            ORG     0000H
            LJMP    MAIN

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

            SETB    P3.2
            SETB    P3.3
LOOP:
            JB      P3.2,LOOP
            JB      P3.3,LOOP
            MOV     IAP_CONTR,#60H      ;Reset to ISP when P3.2 and P3.3 are both 0
            JMP     $

            END
```

6.7.5 Low Voltage Detection

C language code

```
// Operating frequency for test 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       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 Lvd_Isr() interrupt 6
{
    PCON &= ~LVDF;                  //Clear interrupt flag
    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;                  //Test port
//  RSTCFG = ENLVR | LVD3V0;        //Low voltage reset when 3.0V is enabled, no LVD interrupt is generated
    RSTCFG = LVD3V0;                //Low voltage interrupt when 3.0V is enabled
    ELVD = 1;                       //Enable LVD interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
; Operating frequency for test 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

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         0033H
            LJMP        LVDISR

            ORG         0100H
LVDISR:
            ANL         PCON,#NOT LVDF      ;Clear interrupt flag
            CPL         P3.2                ;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      ;LVDF flag needs to be cleared after power on
;           MOV         RSTCFG,#ENLVR | LVD3V0  ;Low voltage reset when 3.0V is enabled, no LVD interrupt is generated
            MOV         RSTCFG,#LVD3V0      ;Low voltage interrupt when 3.0V is enabled
            SETB        ELVD                ;Enable LVD interrupt
            SETB        EA
            JMP         $

            END
```

6.7.6 Power Saving Mode

C language code

```
// Operating frequency for test 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       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 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 to wake up MCU
    EA = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    PCON = IDL;                   //MCU enters IDLE mode
//  PCON = PD;                    //MCU enters power-down mode
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    P35 = 0;

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

IDL         EQU         01H             ;PCON.0
PD          EQU         02H             ;PCON.1
```

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

            ORG     0000H
            LJMP    MAIN
            ORG     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 to wake up MCU
            SETB    EA
            NOP
            NOP
;           MOV     PCON,#IDL           ;MCU enters IDLE mode
            MOV     PCON,#PD            ;MCU enters power down mode
            NOP
            NOP
            NOP
            NOP
            CLR     P3.5                ;Test port
            JMP     $

            END
```

6.7.7 Wake up MCU from Power Saving Mode using INT0/INT1/INT2/INT3/INT4 interrupts

C language code

```
// Operating frequency for test is 11.0592MHz
```

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

sfr     INTCLKO     =   0x8f;
#define EX2             0x10
#define EX3             0x20
#define EX4             0x40

sbit    P10         =   P1^0;
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 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 edge and falling edge interrupts
//  IT0 = 1;                                    //Enable INT0 falling edge interrupt
    EX0 = 1;                                    //Enable INT0 interrupt

    IT1 = 0;                                    //Enable INT1 rising edge 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;                                //MCU enters power down mode
    _nop_();                                    //Enter interrupt service routine immediately after wake-up from power
mode
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

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     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 edge and falling edge interrupts
;           SETB    IT0             ;Enable INT0 falling edge interrupt
            SETB    EX0             ;Enable INT0 interrupt

            CLR     IT1             ;Enable INT1 rising edge and falling edge interrupts
;           SETB    IT1             ;Enable INT1 falling edge interrupt
            SETB    EX1             ;Enable INT1 interrupt

            MOV     INTCLKO,#EX2    ;Enable INT2 falling edge interrupt
            ORL     INTCLKO,#EX3    ;Enable INT3 falling edge interrupt
            ORL     INTCLKO,#EX4    ;Enable INT4 falling edge interrupt

            SETB    EA

            MOV     PCON,#02H       ;MCU enters power-down mode
            NOP                     ;Enter interrupt service routine immediately after wake-up from power
mode
            NOP
            NOP
            NOP
LOOP:
            CPL     P1.1
            JMP     LOOP
```

END

6.7.8 Wake up MCU from Power Saving Mode using T0/T1/T2/T3/T4 pin interrupts

C language code

```
// Operating frequency for test 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         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 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;
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //Start timer
    ET0 = 1;                                    //Enable timer interrupt

    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //Start timer
    ET1 = 1;                                    //Enable timer interrupt

    T2L = 0x66;                                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                                //Start timer
    IE2 = ET2;                                  //Enable timer interrupt

    T3L = 0x66;                                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x08;                               //Start timer
    IE2 |= ET3;                                 //Enable timer interrupt

    T4L = 0x66;                                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M |= 0x80;                              //Start timer
    IE2 |= ET4;                                 //Enable timer interrupt

    EA = 1;

    PCON = 0x02;                                //MCU enters power down mode
    _nop_();    //Does not enter the interrupt service routine immediately after wake-up from power down mode
                //Instead, wait for the timer to overflow before entering the interrupt service routine.
    _nop_();
    _nop_();
```

```
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

Assembly code

```
; Operating frequency for test 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

AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

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     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 timer
        SETB    ET0             ;Enable timer interrupt

        MOV     TL1,#66H        ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1             ;Start timer
        SETB    ET1             ;Enable timer interrupt

        MOV     T2L,#66H        ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H       ;Start timer
        MOV     IE2,#ET2        ;Enable timer interrupt

        MOV     T3L,#66H        ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#08H      ;Start timer
        ORL     IE2,#ET3        ;Enable timer interrupt

        MOV     T4L,#66H        ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        ORL     T4T3M,#80H      ;Start timer
        ORL     IE2,#ET4        ;Enable timer interrupt

        SETB    EA

        MOV     PCON,#02H       ;MCU enters power down mode
        NOP     ;Does not enter the interrupt service routine immediately after wake-up from power down mode
                ;Instead, wait for the timer to overflow before entering the interrupt service routine.
        NOP
        NOP
```

```
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

6.7.9 Wake up MCU from Power Saving Mode using RxD/RxD2/RxD3/RxD4 pin interrupts

C language code

```
// Operating frequency for test is 11.0592MHz

#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 UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}

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;                   //Wake up MCU on the falling edge of RXD / P3.0
//  P_SW1 = 0x40;                   //Wake up MCU on the falling edge of RXD_2/P3.6
//  P_SW1 = 0x80;                   //Wake up MCU on the falling edge of RXD_3/P1.6
//  P_SW1 = 0xc0;                   //Wake up MCU on the falling edge of RXD_4/P4.3

    P_SW2 = 0x00;                   //Wake up MCU on the falling edge of RXD2/P1.0
//  P_SW2 = 0x01;                   //Wake up MCU on the falling edge of RXD2_2/P4.6

    P_SW2 = 0x00;                   //Wake up MCU on the falling edge of RXD3/P0.0
//  P_SW2 = 0x02;                   //Wake up MCU on the falling edge of RXD3_2/P5.0

    P_SW2 = 0x00;                   //Wake up MCU on the falling edge of RXD4/P0.2
//  P_SW2 = 0x04;                   //Wake up MCU on the falling edge of RXD4_2/P5.2

    ES = 1;                         //Enable UART interrupt
    IE2 = ES2;                      //Enable UART2 interrupt
    IE2 |= ES3;                     //Enable UART3 interrupt
    IE2| = ES4;                     //Enable UART4 interrupt
    EA = 1;

    PCON = 0x02;                    //MCU enters power-down mode
    _nop_();    //It will not enter the interrupt service routine after wake-up from power-down mode.
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

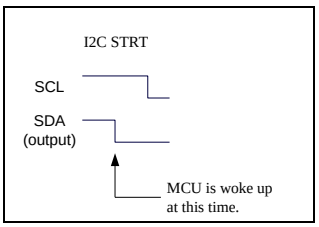
Assembly code

```
; Operating frequency for test is 11.0592MHz

IE2         DATA    0AFH
ES2         EQU     01H
ES3         EQU     08H
ES4         EQU     10H

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     008BH
            LJMP    UART3ISR
            ORG     0093H
            LJMP    UART4ISR

            ORG     0100H
UART1ISR:
            RETI
UART2ISR:
            RETI
UART3ISR:
            RETI
UART4ISR:
            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          ;Wake up MCU on the falling edge of RXD / P3.0
;           MOV     P_SW1,#40H          ;Wake up MCU on the falling edge of RXD_2/P3.6
;           MOV     P_SW1,#80H          ;Wake up MCU on the falling edge of RXD_3/P1.6
;           MOV     P_SW1,#0C0H         ;Wake up MCU on the falling edge of RXD_4/P4.3

            MOV     P_SW2,#00H          ;Wake up MCU on the falling edge of RXD2/P1.0
;           MOV     P_SW2,#01H          ;Wake up MCU on the falling edge of RXD2_2/P4.6

            MOV     P_SW2,#00H          ;Wake up MCU on the falling edge of RXD3/P0.0
;           MOV     P_SW2,#02H          ;Wake up MCU on the falling edge of RXD3_2/P5.0

            MOV     P_SW2,#00H          ;Wake up MCU on the falling edge of RXD4/P0.2
;           MOV     P_SW2,#04H          ;Wake up MCU on the falling edge of RXD4_2/P5.2

            SETB    ES                  ;Enable UART interrupt
            MOV     IE2,#ES2            ;Enable UART2 interrupt
```

```
            ORL     IE2,#ES3            ;Enable UART3 interrupt
            ORL     IE2,#ES4            ;Enable UART4 interrupt
            SETB    EA

            MOV     PCON,#02H           ;MCU enters power down mode
            NOP     ;It will not enter the interrupt service routine after wake-up from power down mode.
            NOP
            NOP
            NOP
LOOP:
            CPL     P1.1
            JMP     LOOP

            END
```

6.7.10 Wake up MCU from Power Saving Mode using I2C SDA pin



C language code

```
// Operating frequency for test is 11.0592MHz

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

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     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;                   //Wake up MCU on the falling edge of SDA/P1.4
//  P_SW2 = 0x10;                   //Wake up MCU on the falling edge of SDA_2/P2.4
//  P_SW2 = 0x30;                   //Wake up MCU on the falling edge of SDA_4/P3.3
    P_SW2 |= 0x80;
    I2CCFG = 0x80;                  //Enable slave mode of I2C module
    I2CSLCR = 0x40;                 //Enable start signal interrupt
    EA = 1;

    PCON = 0x02;                    //MCU enters power-down mode, it will not enter the interrupt service routine after power-down wake-up
    _nop_();
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

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          ;Wake up MCU on the falling edge of SDA/P1.4
//          MOV     P_SW2,#10H          ;Wake up MCU on the falling edge of SDA_2/P2.4
//          MOV     P_SW2,#30H          ;Wake up MCU on the falling edge of SDA_4/P3.3
            ORL     P_SW2,#80H
            MOV     DPTR,#I2CCFG
            MOV     A,#80H
            MOVX    @DPTR,A             ;Enable slave mode of I2C module
            MOV     DPTR,# I2CSLCR
            MOV     A,#40H              ;Enable start signal interrupt
            SETB    EA

            MOV     PCON,#02H           ;MCU enters power down mode
            NOP     ;It will not enter the interrupt service routine after wake-up from power down mode.
            NOP
            NOP
            NOP
LOOP:
            CPL     P1.1
            JMP     LOOP

            END
```

6.7.11 Wake up MCU from Power Saving Mode using Power-down wake-up timer

C language code

```
// Operating frequency for test 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 1 second
        WKTCH = 0x87;

        while (1)
        {
            _nop_();
            _nop_();
            PCON = 0x02;        //MCU enters power-down mode
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            P11 = ~P11;
        }
}
```

Assembly code

```
; Operating frequency for test 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 1 second
        MOV     WKTCH,#87H

LOOP:
        NOP
        NOP
        MOV     PCON,#02H           ;MCU enters power-down mode
        NOP
        NOP
        NOP
        NOP
        CPL     P1.1
        JMP     LOOP

      END
```

6.7.12 Wake up MCU from Power Saving Mode using LVD interrupt (Recommended for use with power-down wake-up timer)

In the power saving mode when the clock is stopped, it is not recommended to start the LVD and comparator, or hardware system will automatically start internal high precision 1.19 V reference source which has a corresponding temperature drift and calibration circuit, and leads to about 300uA extra power consumption. After MCU enters into the clock stopped vibration mode, the working voltage of 3.3V only takes about 0.4uA current, so it is not recommended to open the LVD and comparator when the MCU enters the clock stop mode. If it is really needed, it is recommended to turn on the power-down wake-up timer. The power-down wake-up timer will only increase the power consumption of about 1.4uA, which is acceptable for general systems. The MCU is awakened every 5 seconds by the power-down wake-up timer. After wake-up LVD, comparator and ADC can be used to detect the external battery voltage. The detection took about 1ms. Then it enters the clock shutdown/power-saving mode. In this way, the increased average current was less than 1uA, and the overall power consumption was about 2.8uA (0.4uA + 1.4uA +1uA).

C language code

```
// Operating frequency for test 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       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 LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;                            //Clear interrupt flag
    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;                            //Interrupt flag needs to be cleared after power-on
    RSTCFG = LVD3V0;                          //Set the LVD voltage to 3.0V
    ELVD = 1;                                 //Enable LVD interrupt
    EA = 1;

    PCON = 0x02;                              //MCU enters power-down mode
    _nop_();    //Enter interrupt service routine immediately after wake-up from power mode
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

Assembly code

```
; Operating frequency for test 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

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     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;Clear interrupt flag
            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  ;Interrupt flag needs to be cleared after power-on
            MOV     RSTCFG,# LVD3V0 ;Set the LVD voltage to 3.0V
            SETB    ELVD            ;Enable LVD interrupt
            SETB    EA

            MOV     PCON,#02H       ;MCU enters power-down mode
            NOP     ;Enter interrupt service routine immediately after wake-up from power mode
            NOP
            NOP
```

```
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

6.7.13 Wake up MCU from Power Saving Mode using comparator interrupt (Recommended for use with power-down wake-up timer)

In the power saving mode when the clock is stopped, it is not recommended to start the LVD and comparator, or hardware system will automatically start internal high precision 1.19 V reference source which has a corresponding temperature drift and calibration circuit, and leads to about 300uA extra power consumption. After MCU enters into the clock stopped vibration mode, the working voltage of 3.3V only takes about 0.4uA current, so it is not recommended to open the LVD and comparator when the MCU enters the clock stop mode. If it is really needed, it is recommended to turn on the power-down wake-up timer. The power-down wake-up timer will only increase the power consumption of about 1.4uA, which is acceptable for general systems. The MCU is awakened every 5 seconds by the power-down wake-up timer. After wake-up LVD, comparator and ADC can be used to detect the external battery voltage. The detection took about 1ms. Then it enters the clock shutdown/power-saving mode. In this way, the increased average current was less than 1uA, and the overall power consumption was about 2.8uA (0.4uA + 1.4uA +1uA).

C language code

```
// Operating frequency for test is 11.0592MHz

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

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

sbit    P10     =   P1^0;
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 CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                    //Clear interrupt flag
    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 edge interrupt of comparator
    CMPCR1 &= ~0x08;                        //P3.6 is CMP + input pin
    CMPCR1 |= 0x04;                         //P3.7 is CMP- input pin
    CMPCR1 |= 0x02;                         //Enable Comparator output
    EA = 1;

    PCON = 0x02;                            //MCU enters power-down mode
    _nop_();                        //Enter interrupt service routine immediately after wake-up from power mode
    _nop_();
    _nop_();
    _nop_();

    while (1)
    {
        P11 = ~P11;
    }
}
```

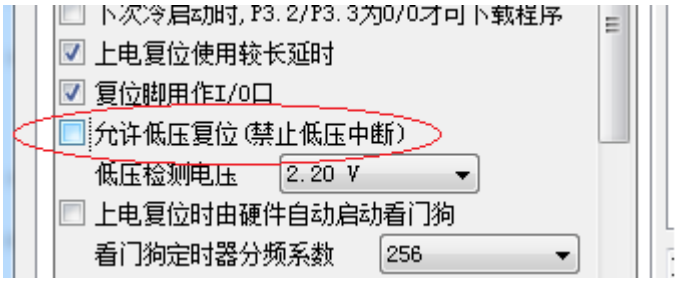
Assembly code

```
; Operating frequency for test is 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            ANL     CMPCR1,#NOT 40H     ;Clear interrupt flag
            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 comparator module
        ORL     CMPCR1,#30H             ;Enable edge interrupt of comparator
        ANL     CMPCR1,#NOT 08H         ;P3.6 is CMP + input pin
        ORL     CMPCR1,#04H             ;P3.7 is CMP- input pin
        ORL     CMPCR1,#02H             ;Enable Comparator output
        SETB    EA

        MOV     PCON,#02H               ;MCU enters power-down mode
        NOP             ;Enter interrupt service routine immediately after wake-up from power mode
        NOP
        NOP
        NOP

LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

6.7.14 Detect the Operating Voltage (Battery Voltage) using LVD

If you need to use LVD to detect the battery voltage, you need to remove the low-voltage reset function when downloading from the ISP, as shown in the following figure.
(It is recommended to use the 15th channel of ADC to detect battery voltage, see ADC chapter)



C language code

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

void delay()
{
    int i;

    for (i=0; i<100; i++)
    {
        _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 = ~power;      // P2.3 ~ P2.0 are used to display battery level
        }
}
```

Assembly code

```
; Operating frequency for test 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

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
        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 are used to display battery level
        JMP     LOOP

DELAY:
        MOV     R0,#100
NEXT:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R0,NEXT
        RET

        END
```

7 Memory

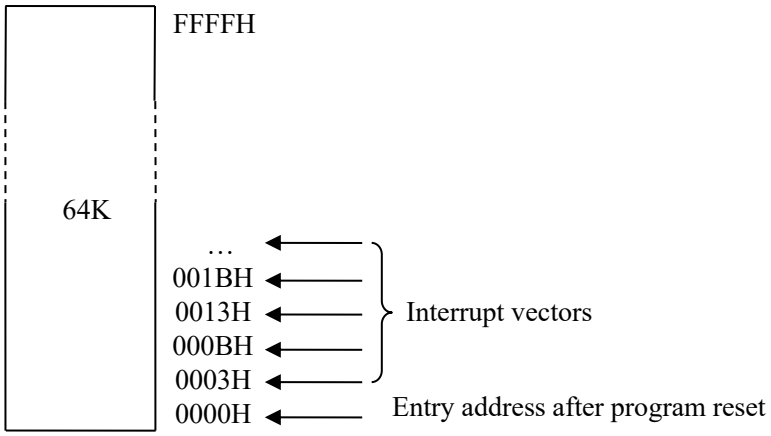
The STC8A8K64D4 series of microcontrollers have separate address spaces for Program Memory and Data Memory. Since no bus is provided for accessing external program memory, all program memory for all microcontrollers is on-chip Flash memory. The microcontrollers can not access external program memory.

Large-capacity data memory is integrated in STC8A8K64D4 series of microcontrollers. The data memory inside the STC8A8K64D4 series of microcontrollers is physically and logically separated into two address spaces: 256 bytes of internal RAM and internal extended RAM. The addresses of the high 128 bytes of internal RAM and special function registers (SFRs) overlap. They can be accessed through different addressing modes in actual use.

7.1 Program Memory

Program memory is used to store user programs, data, tables and other information.

64K bytes of Flash program memory is integrated in STC8A8K64D4-64Pin/48Pin family of microcontrollers.



After the microcontroller resets, the content of the Program Counter (PC) is 0000H, and the CPU begins to execute program from 0000H of Program Memory. The entry addresses of interrupt service routines, which are also called interrupt vectors, are also located in the program memory. Each interrupt has a fixed entry address in Program Memory. When an interrupt occurs and gets response, the microcontroller will automatically jump to its corresponding interrupt entry address to execute the service routine. The entry address of the interrupt service routine for the external interrupt 0 (INT0) is 0003H, the entry address for the timer / counter 0 (TIMER0) interrupt service routine is 000BH, and the entry address for the interrupt service routine for the external interrupt 1 (INT1) is 0013H. The counter/counter 1 (TIMER1) interrupt service routine's entry address is 001BH. More interrupt service routine entry address (interrupt vector), please refer to interrupt chapter.

The interval of adjacent interrupt entry addresses is only 8 bytes, which is not enough to save the complete interrupt service routine in general, so an unconditional jump instruction is stored in the the interrupt vector to jump to the space where the real interrupt service routine is stored, then execute interrupt service routine.

All STC8A8K64D4 series of microcontrollers integrate Flash data memory (EEPROM). The EEPROM is read or written in byte, and is erased in page of 512bytes. It can be repeatedly programmed and erased over 100,000 times, which improves the flexibility and convenience of use.

7.2 Data Memory

The RAM integrated in the STC8A8K64D4 series of microcontrollers can be used to store intermediate results and process data during program execution.

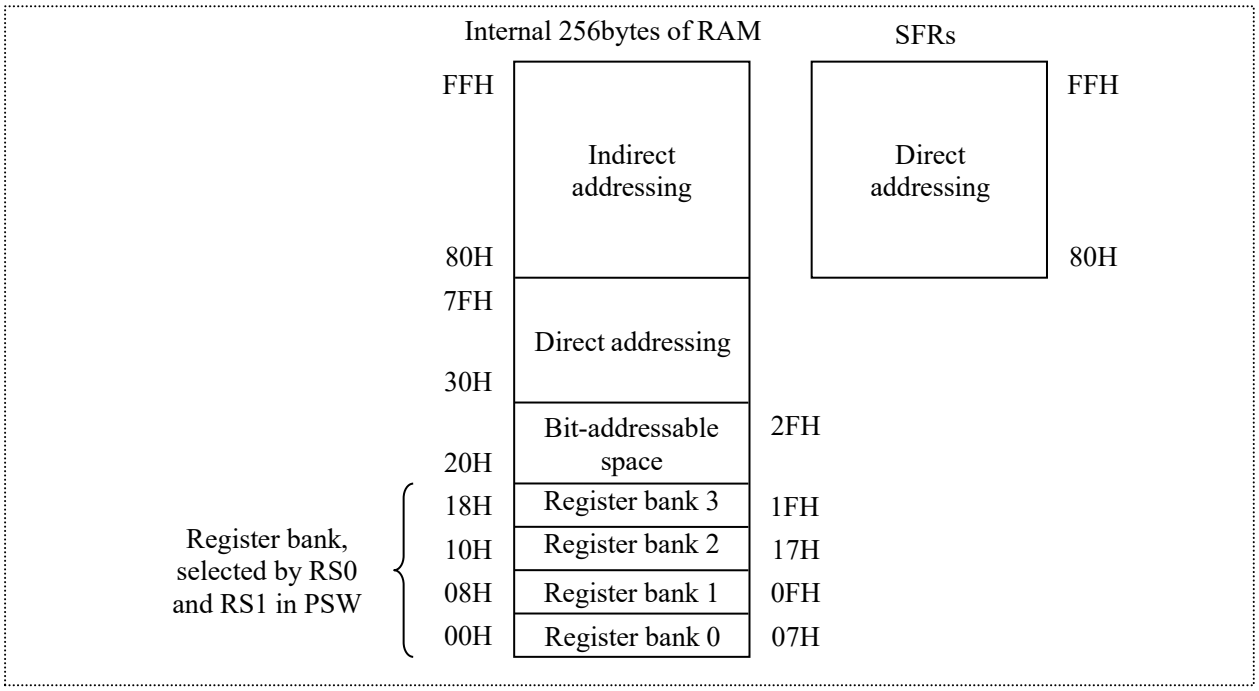
Family of microcontrollers	Internal direct access RAM (DATA)	Internal indirect access RAM (IDATA)	On-chip extended RAM (XDATA)
STC8A8K64D4-64Pin/48Pin	128 bytes	128 bytes	8192 bytes

family			

7.2.1 Internal RAM

A total of 256 bytes of internal RAM can be divided into two parts: Lower 128 bytes of RAM and Upper 128 bytes of RAM. The Lower 128 bytes of data memory are compatible with the traditional 8051 microcontroller, which can be access by either Direct addressing or Indirect addressing. The Upper 128 bytes of RAM (upper 128 bytes of RAM is extended in 8052) and special function registers, SFRs in short, occupy the same block of addresses, 80H to FFH, but they are physically separate entities and are accessed using different addressing modes. Upper 128 bytes of RAM can only be accessed by Indirect addressing, SFRs area can only be accessed by Direct addressing.

Internal RAM is mapped in the following figure.



The Lower 128 bytes of RAM are also called as general purpose RAM space. The general purpose RAM space can be divided into working register banks space, bit addressable space, user RAM space and stack space. Total of 32 bytes of working register bank space, 00H t0 1FH, are divided into 4 groups. Each group is called a register bank, which contains 8 8-bit working registers. All the numbers in different register bank are R0 through R7, but they belong to different Physical space. By using the working register registers, the operation speed can be increased. R0 ~ R7 are commonly used registers. Four bank sare provided because one bank is often not enough. The combination of RS1 and RS0 in the PSW register determines the working register bank currently used, see the introduction of PSW register below.

7.2.2 PSW (program status word register)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	CY	AC	F0	**RS1**	**RS0**	OV	F1	P

CY: Carry/borrow flag bit.

AC: Auxiliary carry/borrow flag bit.

F0: User flag bit 0。

RS1, RS0: Working register select bit

RS1	RS0	working register bank (R0~R7)
0	0	Bank 0 (00H~07H)
0	1	Bank 1 (08H~0FH)
1	0	Bank 2 (10H~17H)

1	1	Bank 3 (18H~1FH)

OV: Overflow flag bit。

F1: User flag bit 1。

P: Parity flag bit.

There are 16 bytes in the bit addressable space, 20H to 2FH. They can either be accessed by byte like ordinary RAM or be individually accessed by any one bit in the byte unit. There are totally 128 bits in this space, whose logic bit addresses are 00H to 7FH. From the appearance, bit addresses and the internal Lower 128 bytes RAM addresses are the same as 00H to 7FH, but in fact, they are essentially different: bit address points to a bit, and the byte address points to a byte unit. They are distinguished by using different instructions in programs.

The addresses 30H to FFH in the internal RAM are the user RAM and stack space. An 8-bit stack pointer, SP in short is used to point to the stack space. On reset, SP is 07H, which is R7 of register bank 0. Therefore, the initial value of SP should be set in the user 104nitialization codes. You would better to set the initial value of SP at 80H or higher.

SP is an 8-bit dedicated register. It indicates the top of the stack in the internal RAM. On reset, SP is initialized to 07H, which makes the stack space begin from 08H. The addresses 08H to 1FH are also the addresses of working register bank 1 through 3. It is better to change the SP value to a value of 80H or more if these spaces are used in user application. The stack of STC8 series of microcontrollers grows upward, which means that when a datum is pushed into the stack, the content of SP will increase.

7.2.3 On-chip extended RAM

In addition to 256 bytes of internal RAM, on-chip extended RAM is integrated in STC8A8K64D4 series of microcontrollers. The method of accessing the on-chip extended RAM is the same as that of the traditional 8051 MCU accessing the external extended RAM. However, the P0 port (data bus and low-order address bus), P2 port (high-order address bus), RD, WR and ALE are not affected.

In assembly language, the on-chip extended RAM is accessed through the MOVX instruction,

```
MOVX    A,@DPTR
MOVX    @DPTR,A
MOVX    A,@Ri
MOVX    @Ri,A
```

In C language, xdata / pdata can be used to declare the storage type, such as,

```
unsigned char xdata i;
```

The control bit EXTRAM located in AUXR register is used to control the access of on-chip extended RAM can be used or not.

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: on-chip extended RAM access control bit

0: On-chip extended RAM is enabled or can be accessed.

1: On-chip extended RAM is disabled.

7.2.5 External extended RAM，XRAM，XDATA

The STC8A8K64D4 series of packages with a pin count of 40 and above have the ability to expand 64KB of external data memory. During access to the external data memory, the WR/RD/ALE signal must be valid. A new special function register BUS_SPEED for controlling the speed of the external 64K byte data bus has been added to the STC8A8K64D4 series of single-chip microcomputers. The description is as follows:

7.2.6 Bus speed control register（BUS_SPEED）

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
BUS_SPEED	A1H	RW_S[1:0]					SPEED[2:0]		

RW_S[1:0]： RD/WR control line selection bit

00: P4.4 is RD, P4.3 is WR

01: P3.7 is RD，P3.6 is WR

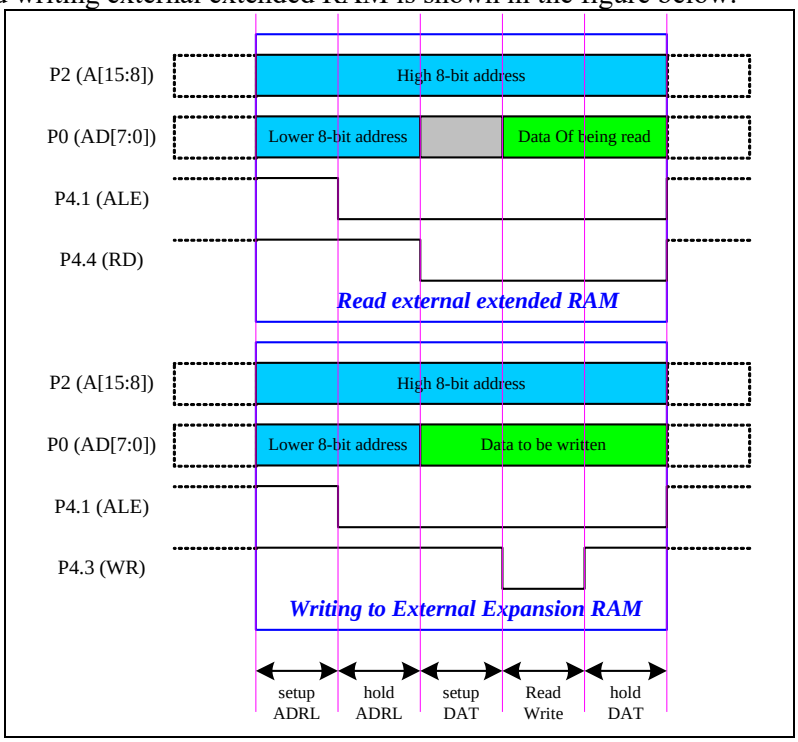
10: P4.2 is RD，P4.0 is WR

11: Reserved.

SPEED[2:0]: Bus read and write speed control (preparation time and hold time of control signal and data signal when reading and writing data)

instruction	Number of clocks	
	Access internal expansion RAM	Access external expansion RAM
MOVX A,@Ri	3	3+5* (SPEED+1)
MOVX @Ri,A	3	3+5* (SPEED+1)
MOVX A,@DPTR	2	3+5* (SPEED+1)
MOVX @DPTR,A	2	3+5* (SPEED+1)

The timing of reading and writing external extended RAM is shown in the figure below:



7.2.7 Bit Addressable Data Memory in 8051

Bit addressable data memory integrated in 8051 single-chip includes two parts: the address range of the first part is 00H ~ 7FH, and the address range of the second part is 80H ~ FFH. The 00H ~ 7FH bit addressing area is a mapping of the 16 bytes of the data area 20H ~ 2FH, and the 80H ~ FFh bit addressing area is the 16 special function registers whose addresses are divisible by 8. (Including 80H, 88H, 90H, 98H, A0H, A8H, B0H, B8H, C0H, C8H, D0H, D8H, E0H, E8H, F0H, F8H).

Address of Data Memory	Address of Bit-addressable							
	B7	B6	B5	B4	B3	B2	B1	B0
F8H (P7)	FFH F8H. 7	FEH F8H. 6	FDH F8H. 5	FCH F8H. 4	FBH F8H. 3	FAH F8H. 2	F9H F8H. 1	F8H F8H. 0
F0H (B)	F7H F0H. 7	F6H F0H. 6	F5H F0H. 5	F4H F0H. 4	F3H F0H. 3	F2H F0H. 2	F1H F0H. 1	F0H F0H. 0

E8H (P6)	EFH E8H.7	EEH E8H.6	EDH E8H.5	ECH E8H.4	EBH E8H.3	EAH E8H.2	E9H E8H.1	E8H E8H.0
E0H (ACC)	E7H E0H.7	E6H E0H.6	E5H E0H.5	E4H E0H.4	E3H E0H.3	E2H E0H.2	E1H E0H.1	E0H E0H.0
D8H (CCON)	DFH D8H.7	DEH D8H.6	DDH D8H.5	DCH D8H.4	DBH D8H.3	DAH D8H.2	D9H D8H.1	D8H D8H.0
D0H (PSW)	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

7.3 Special parameters of memory

The data memory and program memory of the STC8A8K64D4 series of microcontrollers store some special parameters related to the chip, including the global unique ID, the frequency of the 32K power-down wake-up timer, the internal reference voltage value, and the IRC parameters.

The addresses of these parameters in the Flash program memory (ROM) are as follows:

Parameters	Addresses				**Parameter Description**
	STC8A8K16D4	**STC8A8K32D4**	**STC8A8K60D4**	**STC8A8K64D4**	
global unique ID	3FF9H~3FFFH	7FF9H~7FFFH	0EFF9H~0EFFFH	0FDF9H~0FDFFH	7 bytes
internal 1.19V reference voltage	3FF7H~3FF8H	7FF7H~7FF8H	0EFF7H~0EFF8H	0FDF7H~0FDF8H	mV (high byte first)
frequency of the 32K power-down wake-up timer	3FF5H~3FF6H	7FF5H~7FF6H	0EFF5H~0EFF6H	0FDF5H~0FDF6H	Hz (high byte first)
parameters of 22.1184MHz IRC	3FF4H	7FF4H	0EFF4H	0FDF4H	-
parameters of 24MHz IRC	3FF3H	7FF3H	0EFF3H	0FDF3H	-
parameters of 20MHz IRC	3FF2H	7FF2H	EFF2H	FDF2H	
parameters of 27MHz IRC	3FF1H	7FF1H	EFF1H	FDF1H	
parameters of 30MHz IRC	3FF0H	7FF0H	EFF0H	FDF0H	
parameters of 33.1776MHz IRC	3FEFH	7FEFH	EFEFH	FDEFH	
parameters of 35MHz IRC	3FEEH	7FEEH	EFEEH	FDEEH	
parameters of 36.684MHz IRC	3FEDH	7FEDH	EFEDH	FDEDH	
parameters of 40MHz IRC	3FECH	7FECH	EFECH	FDECH	
parameters of 45MHz IRC	3FEBH	7FEBH	EFEBH	FDEBH	
parameters of VRTRIM in 6M band	3FEAH	7FEAH	EFEAH	FDEAH	
parameters of VRTRIM in 10M band	3FE9H	7FE9H	EFE9H	FDE9H	
parameters of VRTRIM in 27M band	3FE8H	7FE8H	EFE8H	FDE8H	
parameters of VRTRIM in 44M band	3FE7H	7FE7H	EFE7H	FDE7H	

The addresses of these parameters in the data memory (RAM) are as follows:

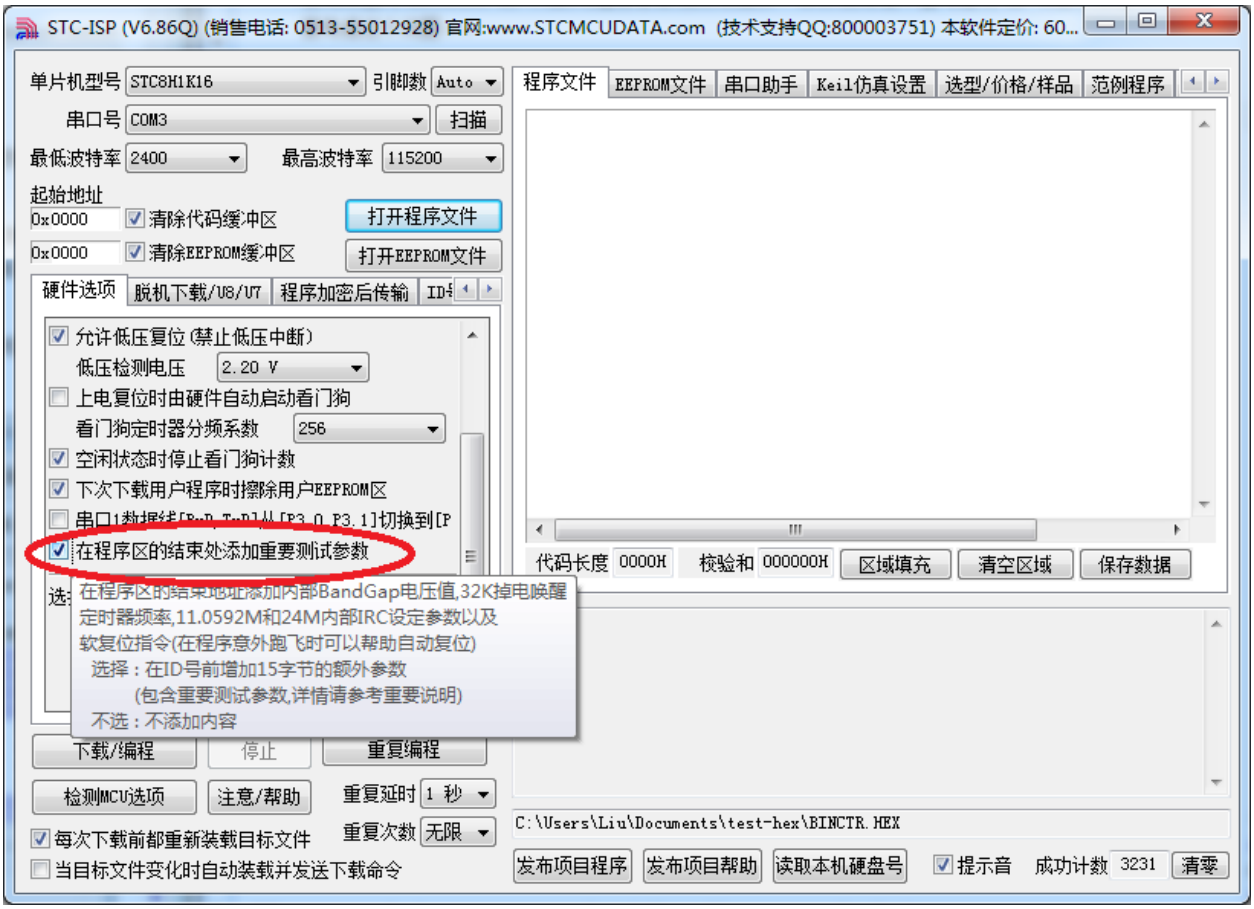
Parameters	Addresses	**Parameter Description**
internal reference voltage	idata：0EFH~0F0H	mV (high byte first)
global unique ID	idata：0F1H~0F7H	7 bytes
frequency of the 32K power-down wake-up timer	idata：0F8H~0F9H	Hz (high byte first)
parameters of 22.1184MHz IRC	idata：0FAH	-
parameters of 24MHz IRC	idata：0FBH	-

Special Note

1. Since the parameters in RAM may be modified, it is generally not recommended to be used. Especially, it is strongly recommended to read ID data in FLASH program memory (ROM) when you use ID for encryption.

2. Due to the size of STC8A8K64S4 and STC8A8K64S2 can be set by user, important parameters stored in the FLASH program memory (ROM) space may be erased or modified when the FLASH is used as EEPROM. So this issue needs to be considered when using these microcontrollers for ID number encryption.

3. By default, only the global unique ID is in the Flash program memory (ROM), and the internal reference 1.19 voltage value, the frequency of the 32K power-down wake-up timer, and the IRC parameters are not available. You need to select the option in the following figure when downloading by ISP, and then the options shown are available.



7.4 Unique ID number and important parameter (CHIPID) stored in read-only special function register

Some STC8A8K64D4 series microcontrollers have built-in 32-byte read-only special function register CHIPID. The content in CHIPID can only be read by the user program and cannot be modified. Using the data in CHIPID to encrypt user programs is the optimal solution officially recommended by STC.

Related registers

Symbol	Description	Address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID00	hardware digital ID00	FDE0H	Globally Unique ID Number (0th byte)								nnnn,nnnn
CHIPID01	hardware digital ID01	FDE1H	Globally Unique ID Number (1st byte)								nnnn,nnnn
CHIPID02	hardware digital ID02	FDE2H	Globally Unique ID Number (2nd byte)								nnnn,nnnn
CHIPID03	hardware digital ID03	FDE3H	Globally Unique ID Number (3rd byte)								nnnn,nnnn
CHIPID04	hardware digital ID04	FDE4H	Globally Unique ID Number (4th byte)								nnnn,nnnn
CHIPID05	hardware digital ID05	FDE5H	Globally Unique ID Number (5th byte)								nnnn,nnnn
CHIPID06	hardware digital ID06	FDE6H	Globally Unique ID Number (6th byte)								nnnn,nnnn
CHIPID07	hardware digital ID07	FDE7H	Internal 1.19V reference signal source (high byte)								nnnn,nnnn
CHIPID08	hardware digital ID08	FDE8H	Internal 1.19V reference signal source (low byte)								nnnn,nnnn
CHIPID09	hardware digital ID09	FDE9H	32K Power-down wake-up timer frequency(high byte)								nnnn,nnnn
CHIPID10	hardware digital ID10	FDEAH	32K Power-down wake-up timer frequency (low byte)								nnnn,nnnn
CHIPID11	hardware digital ID11	FDEBH	IRC parameter of 22.1184MHz (27M band)								nnnn,nnnn
CHIPID12	hardware digital ID12	FDECH	IRC parameter of 24MHz (27M band)								nnnn,nnnn
CHIPID13	hardware digital ID13	FDEDH	IRC parameter of 20MHz (27M band)								nnnn,nnnn
CHIPID14	hardware digital ID14	FDEEH	IRC parameter of 27MHz (27M band)								nnnn,nnnn
CHIPID15	hardware digital ID15	FDEFH	IRC parameter of 30MHz (27M band)								nnnn,nnnn
CHIPID16	hardware digital ID16	FDF0H	IRC parameter of 33.1776MHz (27M band)								nnnn,nnnn
CHIPID17	hardware digital ID17	FDF1H	IRC parameter of 35MHz (44M band)								nnnn,nnnn
CHIPID18	hardware digital ID18	FDF2H	IRC parameter of 36.864MHz(44M band)								nnnn,nnnn
CHIPID19	hardware digital ID19	FDF3H	IRC parameter of 40MHz(44M band)								nnnn,nnnn
CHIPID20	hardware digital ID20	FDF4H	IRC parameter of 45MHz(44M band)								nnnn,nnnn
CHIPID21	hardware digital ID21	FDF5H	VRTRIM parameter of 6M band								nnnn,nnnn
CHIPID22	hardware digital ID22	FDF6H	VRTRIM parameter of 10M band								nnnn,nnnn
CHIPID23	hardware digital ID23	FDF7H	VRTRIM parameter of 27M band								nnnn,nnnn
CHIPID24	hardware digital ID24	FDF8H	VRTRIM parameter of 44M band								nnnn,nnnn
CHIPID25	hardware digital ID25	FDF9H	00H								nnnn,nnnn
CHIPID26	hardware digital ID26	FDFAH	User program space end address (high byte)								nnnn,nnnn
CHIPID27	hardware digital ID27	FDFBH	Chip test time (year)								nnnn,nnnn
CHIPID28	hardware digital ID28	FDFCH	Chip test time (month)								nnnn,nnnn
CHIPID29	hardware digital ID29	FDFDH	Chip test time (day)								nnnn,nnnn
CHIPID30	hardware digital ID30	FDFEH	Chip package form number								nnnn,nnnn
CHIPID31	hardware digital ID31	FDFFH	5AH								nnnn,nnnn

7.4.1 Interpretation of Global Unique ID Number in CHIP

Symbol	Description	Address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID00	hardware digital ID00	FDE0H	Globally Unique ID Number (0th byte)								nnnn,nnnn
CHIPID01	hardware digital ID01	FDE1H	Globally Unique ID Number (1st byte)								nnnn,nnnn
CHIPID02	hardware digital ID02	FDE2H	Globally Unique ID Number (2nd byte)								nnnn,nnnn
CHIPID03	hardware digital ID03	FDE3H	Globally Unique ID Number (3rd byte)								nnnn,nnnn
CHIPID04	hardware digital ID04	FDE4H	Globally Unique ID Number (4th byte)								nnnn,nnnn
CHIPID05	hardware digital ID05	FDE5H	Globally Unique ID Number (5th byte)								nnnn,nnnn
CHIPID06	hardware digital ID06	FDE6H	Globally Unique ID Number (6th byte)								nnnn,nnnn

[CHIPID0, CHIPID1]: 16-bit MCU ID, which is used to distinguish different MCU models (higher bit first).
The commonly used MCU IDs of STC8A8K64D4 series are shown in the following table:

STC8A8K16D4 (F7F1)
STC8A8K32D4 (F7F2)
STC8A8K48D4 (F7F5)
STC8A8K60D4 (F7F3)
STC8A8K64D4 (F7F4)

[CHIPID2, CHIPID3]: 16-bit test machine number (higher bit first).
[CHIPID4, CHIPID5, CHIPID6]: 24-bit test serial number (higher bit first).

7.4.2 Interpretation of the internal reference signal source in CHIP

Symbol	Description	Address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID07	hardware digital ID07	FDE7H	Internal 1.19V reference signal source (high byte)								nnnn,nnnn
CHIPID08	hardware digital ID08	FDE8H	Internal 1.19V reference signal source (low byte)								nnnn,nnnn

[CHIPID7, CHIPID8]: 16-bit internal reference signal source voltage value (higher bit first).

The standard value is 1190 (04A6H), and the unit is mV, which is 1.19V. But there is error in the actual chip due to manufacturing errors. The voltage value of the internal reference signal source is not affected by the working voltage VCC, so the internal reference signal source can be combined with the ADC to calibrate the ADC, and can also be combined with the comparator to detect the operating voltage.

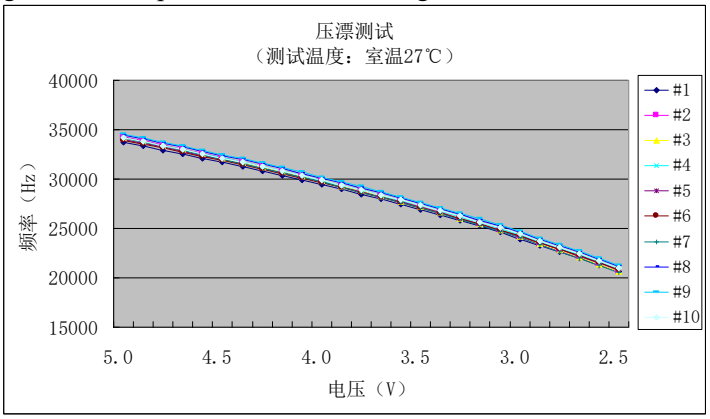
7.4.3 Interpretation of internal 32K IRC oscillation frequency in CHIP

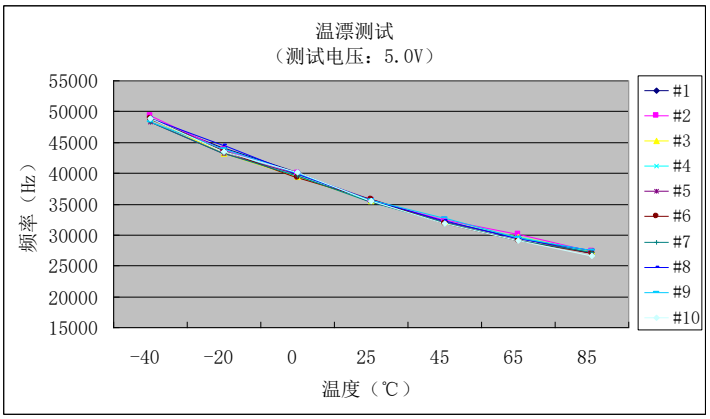
Symbol	Description	Address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID09	hardware digital ID09	FDE9H	32K Power-down wake-up timer frequency(high byte)								nnnn,nnnn
CHIPID10	hardware digital ID10	FDEAH	32K Power-down wake-up timer frequency (low byte)								nnnn,nnnn

[CHIPID9, CHIPID10]: 16-bit 32K IRC oscillator frequency value (higher bit first).

The standard value is 32768 (8000H), the unit is Hz, that is, 32.768KHz. However, the actual chip has manufacturing errors, and the temperature drift and pressure drift are relatively large.

The voltage drift test linear diagram and temperature drift linear diagram of the internal 32K oscillator are as follows:





7.4.4 Interpretation of high precision IRC parameters in CHIP

Symbol	Description	Address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID11	hardware digital ID11	FDEBH	IRC parameter of 22.1184MHz (27M band)								nnnn,nnnn
CHIPID12	hardware digital ID12	FDECH	IRC parameter of 24MHz (27M band)								nnnn,nnnn
CHIPID13	hardware digital ID13	FDEDH	IRC parameter of 20MHz (27M band)								nnnn,nnnn
CHIPID14	hardware digital ID14	FDEEH	IRC parameter of 27MHz (27M band)								nnnn,nnnn
CHIPID15	hardware digital ID15	FDEFH	IRC parameter of 30MHz (27M band)								nnnn,nnnn
CHIPID16	hardware digital ID16	FDF0H	IRC parameter of 33.1776MHz (27M band)								nnnn,nnnn
CHIPID17	hardware digital ID17	FDF1H	IRC parameter of 35MHz (44M band)								nnnn,nnnn
CHIPID18	hardware digital ID18	FDF2H	IRC parameter of 36.864MHz(44M band)								nnnn,nnnn
CHIPID19	hardware digital ID19	FDF3H	IRC parameter of 40MHz(44M band)								nnnn,nnnn

CHIPID20	hardware digital ID20	FDF4H	IRC parameter of 45MHz(44M band)	nnnn,nnnn
CHIPID21	hardware digital ID21	FDF5H	VRTRIM parameter of 6M band	
CHIPID22	hardware digital ID22	FDF6H	VRTRIM parameter of 10M band	
CHIPID23	hardware digital ID23	FDF7H	VRTRIM parameter of 27M band	
CHIPID24	hardware digital ID24	FDF8H	VRTRIM parameter of 44M band	

In the STC8A8K64D4 series MCUs, the integrated high-precision IRC is divided into 4 frequency bands, and the reference voltage value corresponding to every frequency band has been calibrated at the factory. When selecting different frequency bands, you only need to fill the calibrated voltage value of the corresponding frequency band in the VRTRIM register. The center frequencies of the 4 frequency bands are 6MHz, 10MHz, 27MHz and 44MHz respectively. Due to manufacturing errors, the center frequency may generally have a deviation of ±5%. In order to obtain accurate user frequencies, IRTRIM can be used to fine-tune the frequency. When using the download software provided by STC to download the user program, the system will automatically set the VRTRIM and IRTRIM registers according to the frequency set by the user. At the same time, the IRTRIM value of 10 common frequencies and the calibration value of the reference voltage value of 4 frequency bands are preset in CHIPID, so that the user can dynamically modify the working frequency during the running of the program.

[CHIPID11 : CHIPID20]: the IRTRIM value of 10 common frequencies. The annotations in parentheses are the corresponding frequency bands.

[CHIPID21 : CHIPID24]: the calibration value of the reference voltage value of 4 frequency bands.

When the user modifies the frequency dynamically, he only needs to read out a certain frequency calibration value in [CHIPID11 : CHIPID20] and write it into the IRTRIM register, and at the same time, according to the frequency band corresponding to the frequency, set a certain voltage calibration value in [CHIPID21 : CHIPID24] Just read and write to the VRTRIM register. For detailed operations, please refer to the sample programs in the following chapters.

7.4.5 Interpretation of test time parameters in CHIP

Symbol	Description	Address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID27	hardware digital ID27	FDFBH	Chip test time (year)								nnnn,nnnn
CHIPID28	hardware digital ID28	FDFCH	Chip test time (month)								nnnn,nnnn
CHIPID29	hardware digital ID29	FDFDH	Chip test time (day)								nnnn,nnnn

The year, month, and day parameters of the test time are all BCD codes. For example, CHIPID27=0x21, CHIPID28=0x11, CHIPID29=0x18, then the production test date of the target chip is November 18, 2021.

7.4.6 Interpretation of chip package form number in CHIP

Symbol	Description	Address	Bit address and symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID30	hardware digital ID30	FDFEH	Chip package form number								nnnn,nnnn

Chip package form number	Package		Chip package form number	Package
0x00	DIP8		0x50	SOP32
0x01	SOP8		0x51	LQFP32
0x02	DFN8		0x52	QFN32
0x10	DIP16		0x53	PLCC32
0x11	SOP16		0x54	QFN32S
0x20	DIP18		0x60	PDIP40
0x21	SOP18		0x70	LQFP44
0x30	DIP20		0x71	PLCC44
0x31	SOP20		0x72	PQFP44
0x32	TSSOP20		0x80	LQFP48
0x33	LSSOP20		0x81	QFN48
0x34	QFN20		0x90	LQFP64
0x40	SKDIP28		0x91	LQFP64S
0x41	SOP28		0x92	LQFP64L
0x42	TSSOP28		0x93	LQFP64M
0x43	QFN28		0x94	QFN64

7.5 Example Routines

7.5.1 Read Internal Referrence Voltage Value (Read from CHIPID)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

#define    CPUIDBASE        0xfde0

#define    ID_ADDR          ((unsigned char volatile xdata *)(CPUIDBASE + 0x00))
#define    VREF_ADDR        (*(unsigned int volatile xdata *)(CPUIDBASE + 0x07))
#define    F32K_ADDR        (*(unsigned int volatile xdata *)(CPUIDBASE + 0x09))
#define    T22M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0b))    //22.1184MHz
#define    T24M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0c))    //24MHz
#define    T20M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0d))    //20MHz
#define    T27M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0e))    //27MHz
#define    T30M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0f))    //30MHz
#define    T33M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x10))    //33.1776MHz
#define    T35M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x11))    //35MHz
#define    T36M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x12))    //36.864MHz
#define    T40M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x13))    //40MHz
#define    T45M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x14))    //45MHz
#define    VRT6M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x15))    //VRTRIM_6M
#define    VRT10M_ADDR      (*(unsigned char volatile xdata *)(CPUIDBASE + 0x16))    //VRTRIM_10M
#define    VRT27M_ADDR      (*(unsigned char volatile xdata *)(CPUIDBASE + 0x17))    //VRTRIM_27M
#define    VRT44M_ADDR      (*(unsigned char volatile xdata *)(CPUIDBASE + 0x18))    //VRTRIM_44M

sfr        AUXR        =    0x8e;
sfr        P_SW2       =    0xba;

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;

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;

        UartInit();
        ES = 1;
        EA = 1;
        P_SW2 = 0x80;
        UartSend(VREF_ADDR >> 8);                   // Read the high byte of the internal 1.19V reference signal source
        UartSend(VREF_ADDR);                        // Read the low byte of the internal 1.19V reference signal source

        while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CPUIDBASE    EQU   0FDE0H

ID_ADDR          EQU     CPUIDBASE + 00H
VREF_ADDR        EQU     CPUIDBASE + 07H
F32K_ADDR        EQU     CPUIDBASE + 09H
T22M_ADDR        EQU     CPUIDBASE + 0BH          ;22.1184MHz
```

```
T24M_ADDR       EQU     CPUIDBASE + 0CH         ;24MHz
T20M_ADDR       EQU     CPUIDBASE + 0DH         ;20MHz
T27M_ADDR       EQU     CPUIDBASE + 0EH         ;27MHz
T30M_ADDR       EQU     CPUIDBASE + 0FH         ;30MHz
T33M_ADDR       EQU     CPUIDBASE + 10H         ;33.1776MHz
T35M_ADDR       EQU     CPUIDBASE + 11H         ;35MHz
T36M_ADDR       EQU     CPUIDBASE + 12H         ;36.864MHz
T40M_ADDR       EQU     CPUIDBASE + 13H         ;40MHz
T45M_ADDR       EQU     CPUIDBASE + 14H         ;45MHz
VRT6M_ADDR      EQU     CPUIDBASE + 15H         ;VRTRIM_6M
VRT10M_ADDR     EQU     CPUIDBASE + 16H         ;VRTRIM_10M
VRT27M_ADDR     EQU     CPUIDBASE + 17H         ;VRTRIM_27M
VRT44M_ADDR     EQU     CPUIDBASE + 18H         ;VRTRIM_44M

AUXR        DATA        8EH
P_SW2       DATA        0BAH

BUSY        BIT         20H.0

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

            ORG         0000H
            LJMP        MAIN
            ORG         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     P_SW2,#80H
                MOV     DPTR,# VREF_ADDR
                CLR     A
                MOVX    A,@DPTR             ; Read the high byte of the internal 1.19V reference signal source
                LCALL   UART_SEND
                INC     DPTR
                MOVX    A,@DPTR             ; Read the low byte of the internal 1.19V reference signal source
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

7.5.2 Read Internal Referrence Voltage (from Flash)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr       AUXR        =   0x8e;

sfr       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;
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 *)0xeff7;                   // STC8A8K60S4
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                        //Read the high byte of the internal reference voltage
    UartSend(*BGV);                             //Read the low byte of the internal reference voltage

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
BGV         EQU     0EFF7H          ;STC8A8K60S4

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     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 the high byte of the internal reference voltage
        LCALL   UART_SEND
        MOV     A,#1
        MOVC    A,@A+DPTR       ;Read the low byte of the internal reference voltage
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

~~7.5.3 Read Internal Referrence Voltage (Read from RAM)~~

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr       AUXR        =   0x8e;

sfr       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;
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;

        P5M0 = 0x00;
        P5M1 = 0x00;

        BGV = (int idata *)0xef;
        UartInit();
        ES = 1;
        EA = 1;
        UartSend(*BGV >> 8);                    //Read the high byte of the internal reference voltage
        UartSend(*BGV);                         //Read the low byte of the internal reference voltage

        while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
BGV         DATA        0EFH

BUSY        BIT         20H.0

P0M1        DATA        093H
```

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

            ORG     0000H
            LJMP    MAIN
            ORG     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 the high byte of the internal reference voltage
        LCALL   UART_SEND
        INC     R0
        MOV     A,@R0                   ;Read the low byte of the internal reference voltage
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

7.5.4 Read the Unique ID (Read from CHIPID)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

#define   CPUIDBASE         0xfde0

#define   ID_ADDR           ((unsigned char volatile xdata *)(CPUIDBASE + 0x00))
#define   VREF_ADDR         (*(unsigned int volatile xdata *)(CPUIDBASE + 0x07))
#define   F32K_ADDR         (*(unsigned int volatile xdata *)(CPUIDBASE + 0x09))
#define   T22M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0b))    //22.1184MHz
#define   T24M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0c))    //24MHz
#define   T20M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0d))    //20MHz
#define   T27M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0e))    //27MHz
#define   T30M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0f))    //30MHz
#define   T33M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x10))    //33.1776MHz
#define   T35M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x11))    //35MHz
#define   T36M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x12))    //36.864MHz
#define   T40M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x13))    //40MHz
#define   T45M_ADDR         (*(unsigned char volatile xdata *)(CPUIDBASE + 0x14))    //45MHz
#define   VRT6M_ADDR        (*(unsigned char volatile xdata *)(CPUIDBASE + 0x15))    //VRTRIM_6M
#define   VRT10M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x16))    //VRTRIM_10M
#define   VRT27M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x17))    //VRTRIM_27M
#define   VRT44M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x18))    //VRTRIM_44M

sfr     AUXR    =   0x8e;
sfr     P_SW2   =   0xba;

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;

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;

    UartInit();
    ES = 1;
    EA = 1;

    P_SW2 = 0x80;
    for (i=0; i<7; i++)
    {
```

```
        UartSend(ID_ADDR[i]);
    }

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CPUIDBASE   EQU  0FDE0H

ID_ADDR         EQU   CPUIDBASE + 00H
VREF_ADDR       EQU   CPUIDBASE + 07H
F32K_ADDR       EQU   CPUIDBASE + 09H
T22M_ADDR       EQU   CPUIDBASE + 0BH       ;22.1184MHz
T24M_ADDR       EQU   CPUIDBASE + 0CH       ;24MHz
T20M_ADDR       EQU   CPUIDBASE + 0DH       ;20MHz
T27M_ADDR       EQU   CPUIDBASE + 0EH       ;27MHz
T30M_ADDR       EQU   CPUIDBASE + 0FH       ;30MHz
T33M_ADDR       EQU   CPUIDBASE + 10H       ;33.1776MHz
T35M_ADDR       EQU   CPUIDBASE + 11H       ;35MHz
T36M_ADDR       EQU   CPUIDBASE + 12H       ;36.864MHz
T40M_ADDR       EQU   CPUIDBASE + 13H       ;40MHz
T45M_ADDR       EQU   CPUIDBASE + 14H       ;45MHz
VRT6M_ADDR      EQU   CPUIDBASE + 15H       ;VRTRIM_6M
VRT10M_ADDR     EQU   CPUIDBASE + 16H       ;VRTRIM_10M
VRT27M_ADDR     EQU   CPUIDBASE + 17H       ;VRTRIM_27M
VRT44M_ADDR     EQU   CPUIDBASE + 18H       ;VRTRIM_44M

AUXR        DATA        8EH
P_SW2       DATA        0BAH

BUSY        BIT         20H.0

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

            ORG         0000H
            LJMP        MAIN
            ORG         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     P_SW2,#80H
            MOV     DPTR,#ID_ADDR
            MOV     R1,#7
NEXT:       CLR     A
            MOVX    A,@DPTR
            LCALL   UART_SEND
            INC     DPTR
            DJNZ    R1,NEXT

LOOP:
            JMP     LOOP

            END
```

7.5.5 Read the Unique ID (Read from Flash)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

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

sfr       AUXR          =   0x8e;

sfr       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      *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 *)0xeff9;                    // STC8A8K60S4
    UartInit();
    ES = 1;
    EA = 1;

    for (i=0; i<7; i++)
    {
        UartSend(ID[i]);
    }

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
ID          EQU         0EFF9H              ; STC8A8K60S4

BUSY        BIT         20H.0

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

            ORG         0000H
            LJMP        MAIN
            ORG         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
```

7.5.6 Read the Unique ID (Read from RAM)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

```
sfr         AUXR        =   0x8e;

sfr         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        *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

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
ID          DATA        0F1H

BUSY        BIT         20H.0

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

            ORG         0000H
            LJMP        MAIN
            ORG         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
```

7.5.7 Read the Frequency of 32K Power-down Wake-up Timer (Read from CHIPID)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

#define   CPUIDBASE     0xfde0
```

```
#define   ID_ADDR         ((unsigned char volatile xdata *)(CPUIDBASE + 0x00))
#define   VREF_ADDR       (*(unsigned int volatile xdata *)(CPUIDBASE + 0x07))
#define   F32K_ADDR       (*(unsigned int volatile xdata *)(CPUIDBASE + 0x09))
#define   T22M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0b))    //22.1184MHz
#define   T24M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0c))    //24MHz
#define   T20M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0d))    //20MHz
#define   T27M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0e))    //27MHz
#define   T30M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0f))    //30MHz
#define   T33M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x10))    //33.1776MHz
#define   T35M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x11))    //35MHz
#define   T36M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x12))    //36.864MHz
#define   T40M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x13))    //40MHz
#define   T45M_ADDR       (*(unsigned char volatile xdata *)(CPUIDBASE + 0x14))    //45MHz
#define   VRT6M_ADDR      (*(unsigned char volatile xdata *)(CPUIDBASE + 0x15))    //VRTRIM_6M
#define   VRT10M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x16))    //VRTRIM_10M
#define   VRT27M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x17))    //VRTRIM_27M
#define   VRT44M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x18))    //VRTRIM_44M

sfr       AUXR      =   0x8e;
sfr       P_SW2     =   0xba;

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;

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 *)0xeff5;                  // STC8A8K60S4
    UartInit();
    ES = 1;
    EA = 1;

    P_SW2 = 0x80;
    UartSend(F32K_ADDR >> 8);                   // Read high byte of 32K frequency
    UartSend(F32K_ADDR);                        // Read low byte of 32K frequency

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CPUIDBASE       EQU     0FDE0H

ID_ADDR         EQU     CPUIDBASE + 00H
VREF_ADDR       EQU     CPUIDBASE + 07H
F32K_ADDR       EQU     CPUIDBASE + 09H
T22M_ADDR       EQU     CPUIDBASE + 0BH         ;22.1184MHz
T24M_ADDR       EQU     CPUIDBASE + 0CH         ;24MHz
T20M_ADDR       EQU     CPUIDBASE + 0DH         ;20MHz
T27M_ADDR       EQU     CPUIDBASE + 0EH         ;27MHz
T30M_ADDR       EQU     CPUIDBASE + 0FH         ;30MHz
T33M_ADDR       EQU     CPUIDBASE + 10H         ;33.1776MHz
T35M_ADDR       EQU     CPUIDBASE + 11H         ;35MHz
T36M_ADDR       EQU     CPUIDBASE + 12H         ;36.864MHz
T40M_ADDR       EQU     CPUIDBASE + 13H         ;40MHz
T45M_ADDR       EQU     CPUIDBASE + 14H         ;45MHz
VRT6M_ADDR      EQU     CPUIDBASE + 15H         ;VRTRIM_6M
VRT10M_ADDR     EQU     CPUIDBASE + 16H         ;VRTRIM_10M
VRT27M_ADDR     EQU     CPUIDBASE + 17H         ;VRTRIM_27M
VRT44M_ADDR     EQU     CPUIDBASE + 18H         ;VRTRIM_44M

AUXR        DATA        8EH
P_SW2       DATA        0BAH
```

```
BUSY        BIT         20H.0

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

            ORG         0000H
            LJMP        MAIN
            ORG         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     P_SW2,#80H
        MOV     DPTR,# F32K_ADDR
        CLR     A
        MOVX    A,@DPTR             ; Read high byte of 32K frequency
        LCALL   UART_SEND
        INC     DPTR
        CLR     A
        MOVX    A,@ DPTR            ; Read low byte of 32K frequency
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

7.5.8 Read the Frequency of 32K Power-down Wake-up Timer (Read from Flash)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr       AUXR        =   0x8e;

sfr       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;
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 *)0xeff5;                  // STC8A8K60S4
    UartInit();
    ES = 1;
    EA = 1;

    UartSend(*F32K >> 8);                       //Read high byte of 32K frequency
    UartSend(*F32K);                            //Read low byte of 32K frequency

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
F32K        EQU         0EFF5H          ; STC8A8K60S4

BUSY        BIT         20H.0

P0M1        DATA        093H
```

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

            ORG     0000H
            LJMP    MAIN
            ORG     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 high byte of 32K frequency
        LCALL   UART_SEND
        INC     DPTR
        CLR     A
        MOVC    A,@A+DPTR           ;Read low byte of 32K frequency
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

~~7.5.9 Read the Frequency of 32K Power-down Wake-up Timer (Read from RAM)~~

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr     AUXR        =   0x8e;

sfr     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;
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 idata *)0xf8;
    UartInit();
    ES = 1;
    EA = 1;

    UartSend(*F32K >> 8);                   //Read high byte of 32K frequency
    UartSend(*F32K);                        //Read low byte of 32K frequency

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
F32K        DATA        0F8H

BUSY        BIT         20H.0

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
```

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

            ORG     0000H
            LJMP    MAIN
            ORG     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 high byte of 32K frequency
        LCALL   UART_SEND
        INC     R0
        MOV     A,@R0           ;Read low byte of 32K frequency
        LCALL   UART_SEND

LOOP:
        JMP     LOOP

        END
```

7.5.10 Read the User-defined internal IRC Frequency (Read from CHIPID)

C language code

```
//Operating frequency for test is 11.0592MHz

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

#define   CLKSEL        (*(unsigned char volatile xdata *)0xfe00)
#define   CLKDIV        (*(unsigned char volatile xdata *)0xfe01)

#define   CPUIDBASE     0xfde0

#define   ID_ADDR       ((unsigned char volatile xdata *)(CPUIDBASE + 0x00))
#define   VREF_ADDR     (*(unsigned int volatile xdata *)(CPUIDBASE + 0x07))
#define   F32K_ADDR     (*(unsigned int volatile xdata *)(CPUIDBASE + 0x09))
#define   T22M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0b))    //22.1184MHz
#define   T24M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0c))    //24MHz
#define   T20M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0d))    //20MHz
#define   T27M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0e))    //27MHz
#define   T30M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x0f))    //30MHz
#define   T33M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x10))    //33.1776MHz
#define   T35M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x11))    //35MHz
#define   T36M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x12))    //36.864MHz
#define   T40M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x13))    //40MHz
#define   T45M_ADDR     (*(unsigned char volatile xdata *)(CPUIDBASE + 0x14))    //45MHz
#define   VRT6M_ADDR    (*(unsigned char volatile xdata *)(CPUIDBASE + 0x15))    //VRTRIM_6M
#define   VRT10M_ADDR   (*(unsigned char volatile xdata *)(CPUIDBASE + 0x16))    //VRTRIM_10M
#define   VRT27M_ADDR   (*(unsigned char volatile xdata *)(CPUIDBASE + 0x17))    //VRTRIM_27M
#define   VRT44M_ADDR   (*(unsigned char volatile xdata *)(CPUIDBASE + 0x18))    //VRTRIM_44M

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;

//  //Select 20MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T20M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND = 0x02;
//  CLKDIV = 0x00;

//  // Select 22.1184MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T22M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND = 0x02;
//  CLKDIV = 0x00;

    // Select 24MHz
    P_SW2 = 0x80;
    CLKDIV = 0x04;
    IRTRIM = T24M_ADDR;
    VRTRIM = VRT27M_ADDR;
    IRCBAND = 0x02;
    CLKDIV = 0x00;

//  // Select 27MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T27M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND = 0x02;
//  CLKDIV = 0x00;

//  // Select 30MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T30M_ADDR;
//  VRTRIM = VRT27M_ADDR;
//  IRCBAND = 0x02;
//  CLKDIV = 0x00;

//  // Select 33.1776MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T33M_ADDR;
//  VRTRIM = VRT27M_ADDR;
```

```
//      IRCBAND = 0x02;
//      CLKDIV = 0x00;

//      // Select 35MHz
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T35M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND = 0x03;
//      CLKDIV = 0x00;

//      // Select 40MHz
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T40M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND = 0x03;
//      CLKDIV = 0x00;

//      // Select 45MHz
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T45M_ADDR;
//      VRTRIM = VRT44M_ADDR;
//      IRCBAND = 0x03;
//      CLKDIV = 0x00;

        while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CPUIDBASE       EQU     0FDE0H

ID_ADDR         EQU     CPUIDBASE + 00H
VREF_ADDR       EQU     CPUIDBASE + 07H
F32K_ADDR       EQU     CPUIDBASE + 09H
T22M_ADDR       EQU     CPUIDBASE + 0BH         ;22.1184MHz
T24M_ADDR       EQU     CPUIDBASE + 0CH         ;24MHz
T20M_ADDR       EQU     CPUIDBASE + 0DH         ;20MHz
T27M_ADDR       EQU     CPUIDBASE + 0EH         ;27MHz
T30M_ADDR       EQU     CPUIDBASE + 0FH         ;30MHz
T33M_ADDR       EQU     CPUIDBASE + 10H         ;33.1776MHz
T35M_ADDR       EQU     CPUIDBASE + 11H         ;35MHz
T36M_ADDR       EQU     CPUIDBASE + 12H         ;36.864MHz
T40M_ADDR       EQU     CPUIDBASE + 13H         ;40MHz
T45M_ADDR       EQU     CPUIDBASE + 14H         ;45MHz
VRT6M_ADDR      EQU     CPUIDBASE + 15H         ;VRTRIM_6M
VRT10M_ADDR     EQU     CPUIDBASE + 16H         ;VRTRIM_10M
VRT27M_ADDR     EQU     CPUIDBASE + 17H         ;VRTRIM_27M
VRT44M_ADDR     EQU     CPUIDBASE + 18H         ;VRTRIM_44M

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

;           ; Select 20MHz
;           MOV     P_SW2,#80H
;           MOV     A,#4
;           MOV     DPTR,#CLKDIV
;           MOVX    @DPTR,A
;           MOV     DPTR,#T20M_ADDR
;           CLR     A
;           MOVX    A,@DPTR
;           MOV     IRTRIM,A
;           MOV     DPTR,#VRT27M_ADDR
;           CLR     A
;           MOVX    A,@DPTR
;           MOV     VRTRIM,A
;           MOV     IRCBAND,#02H
;           MOV     A,#0
;           MOV     DPTR,#CLKDIV
;           MOVX    @DPTR,A
;           MOV     P_SW2,#00H

;           ; Select 22.1184MHz
;           MOV     P_SW2,#80H
;           MOV     A,#4
;           MOV     DPTR,#CLKDIV
;           MOVX    @DPTR,A
;           MOV     DPTR,#T22M_ADDR
;           CLR     A
;           MOVX    A,@DPTR
;           MOV     IRTRIM,A
```

```
;               MOV     DPTR,#VRT27M_ADDR
;               CLR     A
;               MOVX    A,@DPTR
;               MOV     VRTRIM,A
;               MOV     IRCBAND,#02H
;               MOV     A,#0
;               MOV     DPTR,#CLKDIV
;               MOVX    @DPTR,A
;               MOV     P_SW2,#00H

                ; Select 24MHz
                MOV     P_SW2,#80H
                MOV     A,#4
                MOV     DPTR,#CLKDIV
                MOV     @DPTR,A
                MOV     DPTR,#T24M_ADDR
                CLR     A
                MOVX    A,@DPTR
                MOV     IRTRIM,A
                MOV     DPTR,#VRT27M_ADDR
                CLR     A
                MOVX    A,@DPTR
                MOV     VRTRIM,A
                MOV     IRCBAND,#02H
                MOV     A,#0
                MOV     DPTR,#CLKDIV
                MOV     @DPTR,A
                MOV     P_SW2,#00H

;               ; Select 27MHz
;               MOV     P_SW2,#80H
;               MOV     A,#4
;               MOV     DPTR,#CLKDIV
;               MOVX    @DPTR,A
;               MOV     DPTR,#T27M_ADDR
;               CLR     A
;               MOVX    A,@DPTR
;               MOV     IRTRIM,A
;               MOV     DPTR,#VRT27M_ADDR
;               CLR     A
;               MOVX    A,@DPTR
;               MOV     VRTRIM,A
;               MOV     IRCBAND,#02H
;               MOV     A,#0
;               MOV     DPTR,#CLKDIV
;               MOVX    @DPTR,A
;               MOV     P_SW2,#00H

;               ; Select 30MHz
;               MOV     P_SW2,#80H
;               MOV     A,#4
;               MOV     DPTR,#CLKDIV
;               MOVX    @DPTR,A
;               MOV     DPTR,#T30M_ADDR
;               CLR     A
;               MOVX    A,@DPTR
;               MOV     IRTRIM,A
;               MOV     DPTR,#VRT27M_ADDR
;               CLR     A
;               MOVX    A,@DPTR
;               MOV     VRTRIM,A
```

```
;       MOV     IRCBAND,#02H
;       MOV     A,#0
;       MOV     DPTR,#CLKDIV
;       MOVX    @DPTR,A
;       MOV     P_SW2,#00H

;       ; Select 33.1776MHz
;       MOV     P_SW2,#80H
;       MOV     A,#4
;       MOV     DPTR,#CLKDIV
;       MOVX    @DPTR,A
;       MOV     DPTR,#T33M_ADDR
;       CLR     A
;       MOVX    A,@DPTR
;       MOV     IRTRIM,A
;       MOV     DPTR,#VRT27M_ADDR
;       CLR     A
;       MOVX    A,@DPTR
;       MOV     VRTRIM,A
;       MOV     IRCBAND,#02H
;       MOV     A,#0
;       MOV     DPTR,#CLKDIV
;       MOVX    @DPTR,A
;       MOV     P_SW2,#00H

;       ; Select 35MHz
;       MOV     P_SW2,#80H
;       MOV     A,#4
;       MOV     DPTR,#CLKDIV
;       MOVX    @DPTR,A
;       MOV     DPTR,#T35M_ADDR
;       CLR     A
;       MOVX    A,@DPTR
;       MOV     IRTRIM,A
;       MOV     DPTR,#VRT44M_ADDR
;       CLR     A
;       MOVX    A,@DPTR
;       MOV     VRTRIM,A
;       MOV     IRCBAND,#03H
;       MOV     A,#0
;       MOV     DPTR,#CLKDIV
;       MOVX    @DPTR,A
;       MOV     P_SW2,#00H

;       ; Select 36.864MHz
;       MOV     P_SW2,#80H
;       MOV     A,#4
;       MOV     DPTR,#CLKDIV
;       MOVX    @DPTR,A
;       MOV     DPTR,#T36M_ADDR
;       CLR     A
;       MOVX    A,@DPTR
;       MOV     IRTRIM,A
;       MOV     DPTR,#VRT44M_ADDR
;       CLR     A
;       MOVX    A,@DPTR
;       MOV     VRTRIM,A
;       MOV     IRCBAND,#03H
;       MOV     A,#0
;       MOV     DPTR,#CLKDIV
;       MOVX    @DPTR,A
```

```
;               MOV         P_SW2,#00H

;               ; Select 40MHz
;               MOV         P_SW2,#80H
;               MOV         A,#4
;               MOV         DPTR,#CLKDIV
;               MOVX        @DPTR,A
;               MOV         DPTR,#T40M_ADDR
;               CLR         A
;               MOVX        A,@DPTR
;               MOV         IRTRIM,A
;               MOV         DPTR,#VRT44M_ADDR
;               CLR         A
;               MOVX        A,@DPTR
;               MOV         VRTRIM,A
;               MOV         IRCBAND,#03H
;               MOV         A,#0
;               MOV         DPTR,#CLKDIV
;               MOVX        @DPTR,A
;               MOV         P_SW2,#00H

;               ; Select 45MHz
;               MOV         P_SW2,#80H
;               MOV         A,#4
;               MOV         DPTR,#CLKDIV
;               MOVX        @DPTR,A
;               MOV         DPTR,#T45M_ADDR
;               CLR         A
;               MOVX        A,@DPTR
;               MOV         IRTRIM,A
;               MOV         DPTR,#VRT44M_ADDR
;               CLR         A
;               MOVX        A,@DPTR
;               MOV         VRTRIM,A
;               MOV         IRCBAND,#03H
;               MOV         A,#0
;               MOV         DPTR,#CLKDIV
;               MOVX        @DPTR,A
;               MOV         P_SW2,#00H
;$ENDIF

                JMP         $

                END
```

7.5.11 Read the User-defined internal IRC Frequency (Read from Flash)

C language code

```
//Operating frequency for test is 11.0592MHz

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

#define     CKSEL           (*(unsigned char volatile xdata *)0xfe00)
#define     CLKDIV          (*(unsigned char volatile xdata *)0xfe01)

// The following table is the parameter list of STC8A8K60S4
#define     ID_ROMADDR              ((unsigned char code *)0xeff9)
#define     VREF_ROMADDR            (*(unsigned int code *)0xeff7)
```

```
#define   F32K_ROMADDR        (*(unsigned int code *)0xeff5)
#define   T22M_ROMADDR        (*(unsigned char code *)0xeff4)       //22.1184MHz
#define   T24M_ROMADDR        (*(unsigned char code *)0xeff3)       //24MHz
#define   T20M_ROMADDR        (*(unsigned char code *)0xeff2)       //20MHz
#define   T27M_ROMADDR        (*(unsigned char code *)0xeff1)       //27MHz
#define   T30M_ROMADDR        (*(unsigned char code *)0xeff0)       //30MHz
#define   T33M_ROMADDR        (*(unsigned char code *)0xefef)       //33.1776MHz
#define   T35M_ROMADDR        (*(unsigned char code *)0xefee)       //35MHz
#define   T36M_ROMADDR        (*(unsigned char code *)0xefed)       //36.864MHz
#define   VRT20M_ROMADDR      (*(unsigned char code *)0xefea)       //VRTRIM_20M
#define   VRT35M_ROMADDR      (*(unsigned char code *)0xefe9)       //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;

//  //Select 20MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T20M_ROMADDR;
//  VRTRIM = VRT20M_ROMADDR;
//  IRCBAND = 0x00;
//  CLKDIV = 0x00;

//  // Select 22.1184MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T22M_ROMADDR;
//  VRTRIM = VRT20M_ROMADDR;
//  IRCBAND = 0x00;
//  CLKDIV = 0x00;
```

```
    // Select 24MHz
    P_SW2 = 0x80;
    CLKDIV = 0x04;
    IRTRIM = T24M_ROMADDR;
    VRTRIM = VRT20M_ROMADDR;
    IRCBAND = 0x00;
    CLKDIV = 0x00;

//  // Select 27MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T27M_ROMADDR;
//  VRTRIM = VRT35M_ROMADDR;
//  IRCBAND = 0x01;
//  CLKDIV = 0x00;

//  // Select 30MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T30M_ROMADDR;
//  VRTRIM = VRT35M_ROMADDR;
//  IRCBAND = 0x01;
//  CLKDIV = 0x00;

//  // Select 33.1776MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T33M_ROMADDR;
//  VRTRIM = VRT35M_ROMADDR;
//  IRCBAND = 0x01;
//  CLKDIV = 0x00;

//  // Select 35MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T35M_ROMADDR;
//  VRTRIM = VRT35M_ROMADDR;
//  IRCBAND = 0x01;
//  CLKDIV = 0x00;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

; The following table is the parameter list of STC8A8K60S4
ID_ROMADDR          EQU     0EFF9H
VREF_ROMADDR        EQU     0EFF7H
F32K_ROMADDR        EQU     0EFF5H
T22M_ROMADDR        EQU     0EFF4H          //22.1184MHz
T24M_ROMADDR        EQU     0EFF3H          //24MHz
T20M_ROMADDR        EQU     0EFF2H          //20MHz
T27M_ROMADDR        EQU     0EFF1H          //27MHz
T30M_ROMADDR        EQU     0EFF0H          //30MHz
T33M_ROMADDR        EQU     0EFEFH          //33.1776MHz
T35M_ROMADDR        EQU     0EFEEH          //35MHz
T36M_ROMADDR        EQU     0EFEDH          //36.864MHz
VRT20M_ROMADDR      EQU     0EFEAH          //VRTRIM_20M
VRT35M_ROMADDR      EQU     0EFE9H          //VRTRIM_35M
```

```
P_SW2       DATA    0BAH
CKSEL       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

;           ;Select 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

;           ; Select 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

                ; Select 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

;               ; Select 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

;               ; Select 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

;               ; Select 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

;               ; Select 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

;               ; Select 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
```

7.5.12 Read the User-defined internal IRC Frequency (Read from RAM)

C language code

```
//Operating frequency for test 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;
//  IRCCR = *IRC22M;                    //Load 22.1184MHz IRC parameters
    IRCCR = *IRC24M;                    //Load 24MHz IRC parameters

    P_SW2 = 0x80;
    CLKDIV = 0;                         //No division to main clock
    P_SW2 = 0x00;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz
```

```
P_SW2       DATA    0BAH
CLKDIV      EQU     0FE01H

IRCCR       DATA    09FH

IRC22M      DATA    0FAH
IRC24M      DATA    0FBH

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     R0,#IRC22M          ;Load 22.1184MHz IRC parameters
;           MOV     IRCCR,@R0
            MOV     R0,#IRC24M          ;Load 24MHz IRC parameters
            MOV     IRCCR,@R0

            MOV     P_SW2,#80H
            MOV     A,#0                ;No division to main clock
            MOV     DPTR,#CLKDIV
            MOVX    @DPTR,A
            MOV     P_SW2,#00H

            JMP     $

            END
```

8 Special Function Registers

8.1 STC8A8K64D4-64Pin/48Pin family

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7	CH	CCAP0H	CCAP1H	CCAL2H	-	-	RSTCFG
F0H	B	PWMSET	PCA_PWM0	PCA_PWM1	PCA_PWM2	IAP_TPS	PWMCFG	-
E8H	P6	CL	CCAP0L	CCAP1L	CCAL2L	-	IP3H	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2	-	ADCCFG	IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0	P6M1	P6M0	SPSTAT	SPCTL	SPDAT
C0H	P4	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	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	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	S4CON	S4BUF	-	PCON

Bit addressable　　Not bit addressable

Note: The register address can be bit-addressable only if it is divisible by 8, and it cannot be bit-addressable if it is not divisible by 8.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FF48H	PWM7T1H	PWM7T1L	PWM7T2H	PWM7T2L	PWM7CR	PWM7HLD		
FF40H	PWM6T1H	PWM6T1L	PWM6T2H	PWM6T2L	PWM6CR	PWM6HLD		
FF38H	PWM5T1H	PWM5T1L	PWM5T2H	PWM5T2L	PWM5CR	PWM5HLD		
FF30H	PWM4T1H	PWM4T1L	PWM4T2H	PWM4T2L	PWM4CR	PWM4HLD		
FF28H	PWM3T1H	PWM3T1L	PWM3T2H	PWM3T2L	PWM1CR	PWM3HLD		
FF20H	PWM2T1H	PWM2T1L	PWM2T2H	PWM2T2L	PWM2CR	PWM2HLD		
FF18H	PWM1T1H	PWM1T1L	PWM1T2H	PWMT2L	PWM1CR	PWM1HLD		
FF10H	PWM0T1H	PWM0T1L	PWM0T2H	PWM0T2L	PWM0CR	PWM0HLD		
FF00H	PWMCH	PWMCL	PWMCKS	PWMTADCH	PWMTADCL	PWMIF	PWMFDCR	
FEA8H	ADCTIM					ADCEXCFG	CMPEXCFG	
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE50H	LCMIFCFG	LCMIFCFG2	LCMIFCR	LCMIFSTA	LCMIFDATL	LCMIFDATH		
FE30H	P0IE	P1IE	P2IE	P3IE	P4IE	P5IE	P6IE	P7IE
FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR	P6DR	P7DR
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR	P6SR	P7SR
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FDF8H	CHIPID24	CHIPID25	CHIPID26	CHIPID27	CHIPID28	CHIPID29	CHIPID30	CHIPID31
FDF0H	CHIPID16	CHIPID17	CHIPID18	CHIPID19	CHIPID20	CHIPID21	CHIPID22	CHIPID23
FDE8H	CHIPID8	CHIPID9	CHIPID10	CHIPID11	CHIPID12	CHIPID13	CHIPID14	CHIPID15
FDE0H	CHIPID0	CHIPID1	CHIPID2	CHIPID3	CHIPID4	CHIPID5	CHIPID6	CHIPID7
FD60H	PINIPL	PINIPH						
FD50H					CCAPM3	CCAP3L	CCAP3H	PCA_PWM3
FD40H	P0WKUE	P1WKUE	P2WKUE	P3WKUE	P4WKUE	P5WKUE	P6WKUE	P7WKUE
FD30H	P0IM1	P1IM1	P2IM1	P3IM1	P4IM1	P5IM1	P6IM1	P7IM1
FD20H	P0IM0	P1IM0	P2IM0	P3IM0	P4IM0	P5IM0	P6IM0	P7IM0
FD10H	P0INTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF	P6INTF	P7INTF
FD00H	P0INTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE	P6INTE	P7INTE
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON

FA78H	DMA_LCM_RXAL								
FA70H	DMA_LCM_CFG	DMA_LCM_CR	DMA_LCM_STA	DMA_LCM_AMT	DMA_LCM_DONE	DMA_LCM_TXAH	DMA_LCM_TXAL	DMA_LCM_RXAH	
FA68H	DMA_UR4R_CFG	DMA_UR4R_CR	DMA_UR4R_STA	DMA_UR4R_AMT	DMA_UR4R_DONE	DMA_UR4R_RXAH	DMA_UR4R_RXAL		
FA60H	DMA_UR4T_CFG	DMA_UR4T_CR	DMA_UR4T_STA	DMA_UR4T_AMT	DMA_UR4T_DONE	DMA_UR4T_TXAH	DMA_UR4T_TXAL		
FA58H	DMA_UR3R_CFG	DMA_UR3R_CR	DMA_UR3R_STA	DMA_UR3R_AMT	DMA_UR3R_DONE	DMA_UR3R_RXAH	DMA_UR3R_RXAL		
FA50H	DMA_UR3T_CFG	DMA_UR3T_CR	DMA_UR3T_STA	DMA_UR3T_AMT	DMA_UR3T_DONE	DMA_UR3T_TXAH	DMA_UR3T_TXAL		
FA48H	DMA_UR2R_CFG	DMA_UR2R_CR	DMA_UR2R_STA	DMA_UR2R_AMT	DMA_UR2R_DONE	DMA_UR2R_RXAH	DMA_UR2R_RXAL		
FA40H	DMA_UR2T_CFG	DMA_UR2T_CR	DMA_UR2T_STA	DMA_UR2T_AMT	DMA_UR2T_DONE	DMA_UR2T_TXAH	DMA_UR2T_TXAL		
FA38H	DMA_UR1R_CFG	DMA_UR1R_CR	DMA_UR1R_STA	DMA_UR1R_AMT	DMA_UR1R_DONE	DMA_UR1R_RXAH	DMA_UR1R_RXAL		
FA30H	DMA_UR1T_CFG	DMA_UR1T_CR	DMA_UR1T_STA	DMA_UR1T_AMT	DMA_UR1T_DONE	DMA_UR1T_TXAH	DMA_UR1T_TXAL		
FA28H	DMA_SPI_RXAL	DMA_SPI_CFG2							
FA20H	DMA_SPI_CFG	DMA_SPI_CR	DMA_SPI_STA	DMA_SPI_AMT	DMA_SPI_DONE	DMA_SPI_TXAH	DMA_SPI_TXAL	DMA_SPI_RXAH	
FA18H	DMA_ADC_RXAL	DMA_ADC_CFG2	DMA_ADC_CHSW0	DMA_ADC_CHSW1					
FA10H	DMA_ADC_CFG	DMA_ADC_CR	DMA_ADC_STA					DMA_ADC_RXAH	
FA08H	DMA_M2M_RXAL								
FA00H	DMA_M2M_CFG	DMA_M2M_CR	DMA_M2M_STA	DMA_M2M_AMT	DMA_M2M_DONE	DMA_M2M_TXAH	DMA_M2M_TXAL	DMA_M2M_RXAH	

8.2 List of Special Function Registers

Note: The register address can be bit-addressable only if it is divisible by 8, and it cannot be bit-addressable if it is not divisible by 8.
Registers in STC8A8K64D4 which can be bit-addressable are: P0 (80H), TCON (88H), P1 (90H), SCON (98H), P2 (A0H), IE (A8H), P3 (B0H), IP (B8H), P4 (C0H), P5 (C8H), PSW (D0H), ACC (E0H), B (F0H)

Symbol	Description	Address	Bit Address and Symbol								Value after Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	Port 0	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
SP	Stack Pointer	81H									0000,0111
DPL	Data pointer low byte register	82H									0000,0000
DPH	Data pointer high byte register	83H									0000,0000
S4CON	UART 4 control register	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	UART 4 data buffer register	85H									0000,0000
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	Timer 0 and 1 control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	Timer 0 and 1 mode register	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	Timer 0 low byte register	8AH									0000,0000
TL1	Timer 1 low byte register	8BH									0000,0000
TH0	Timer 0 high byte register	8CH									0000,0000
TH1	Timer 1 high byte register	8DH									0000,0000
AUXR	Auxiliary register 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	External interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	Port 1	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P1M1	Port 1 mode register 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	Port 1 mode register 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P0M1	Port 0 mode register 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	Port 0 mode register 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P2M1	Port 2 mode register 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	Port 2 mode register 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
SCON	UART1 control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	UART1 data buffer register	99H									0000,0000
S2CON	UART2 control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0x00,0000
S2BUF	UART2 data buffer registe	9BH									0000,0000
IRCBAND	IRC band selection detection	9DH	-	-	-	-	-	-	-	SEL	xxxx,xxxn
LIRTRIM	IRC frequency trim register	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		xxxx,xxnn
IRTRIM	IRC frequency adjustment register	9FH	IRTRIM[7:0]								nnnn,nnnn
P2	Port 2	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
BUS_SPEED	Bus speed control register	A1H	RW_S[1:0]						SPEED[2:0]		00xx,x000
P_SW1	Peripheral port switch register 1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-	nn00,000x
IE	Interrupt enable register	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
SADDR	UART1 slave address register	A9H									0000,0000
WKTCL	Wake-up Timer Control Register Low Byte	AAH									1111,1111
WKTCH	Wake-up Timer Control Register High Byte	ABH	WKTEN								0111,1111
S3CON	UART3 control register	ACH	S3SM0	S3ST4	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	UART3 data buffer register	ADH									0000,0000
TA	DPTR Timing control register	AEH									0000,0000
IE2	Interrupt enable register 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000
P3	Port 3	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P3M1	Port 3 mode register 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	1111,1100
P3M0	Port 3 mode register 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	0000,0000
P4M1	Port 4 mode register 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	Port 4 mode register 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
IP2	2nd Interrupt Priority register low byte	B5H	-	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	2nd Interrupt Priority register high byte	B6H	-	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H	0000,0000
IPH	Interrupt Priority High Byte	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP	Interrupt Priority Low Byte	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
SADEN	UART1 slave address enable register	B9H									0000,0000
P_SW2	Peripheral port switch register 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000
ADC_CONTR	ADC control register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				0000,0000
ADC_RES	ADC Result High Byte	BDH									0000,0000
ADC_RESL	ADC Result Low Byte	BEH									0000,0000
P4	Port 4	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
WDT_CONTR	Watchdog control register	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0x00,0000
IAP_DATA	IAP Flash Data Register	C2H									1111,1111
IAP_ADDRH	IAP Flash Address High Byte	C3H									0000,0000
IAP_ADDRL	IAP Flash Address Low Byte	C4H									0000,0000
IAP_CMD	IAP Flash Command Register	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP Flash Trigger register	C6H									0000,0000
IAP_CONTR	IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
P5	Port 5	C8H	-	-	P55	P54	P53	P52	P51	P50	xx11,1111
P5M1	Port 5 mode register 1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	Port 5 mode register 0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx00,0000
P6M1	Port 6 mode register 1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1	1111,1111
P6M0	Port 6 mode register 0	CCH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0	0000,0000

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 Register	D0H	CY	AC	F0	RS1	RS0	OV	F1	P	0000,00x0
T4T3M	Timer4 and Timer 3 Control Register	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	Timer 4 high byte register	D2H									0000,0000
T4L	Timer 4 low byte register	D3H									0000,0000
T3H	Timer 3 high byte register	D4H									0000,0000
T3L	Timer 3 low byte register	D5H									0000,0000
T2H	Timer 2 high byte register	D6H									0000,0000
T2L	Timer 2 low byte register	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 Control Register	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA Module 1 Control Register	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA Module 2 Control Register	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
ADCCFG	ADC Configuration Register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
IP3	3nd Interrupt Priority register low byte	DFH	-	-	-	-	-	-	PS4	PS3	xxxx,x000
ACC	Accumulator	E0H									0000,0000
P7M1	Port 7 mode register 1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1	1111,1111
P7M0	Port 7 mode register 0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0	0000,0000
DPS	DPTR Selection Register	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
DPL1	2nd Data pointer low byte	E4H									0000,0000
DPH1	2nd Data pointer high byte	E5H									0000,0000
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
P6	Port 6	E8H	P67	P66	P65	P64	P63	P62	P61	P60	1111,1111
CL	PCA counter low byte	E9H									0000,0000
CCAP0L	PCA module 0 low byte	EAH									0000,0000
CCAP1L	PCA module 1 low byte	EBH									0000,0000
CCAP2L	PCA module 2 low byte	ECH									0000,0000
IP3H	3nd Interrupt Priority Register High Byte	EEH	-	-	-	-	-	-	PS4H	PS3H	xxxx,x000
AUXINTIF	Extended External Interrupt Flag Register	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
B	B register	F0H									0000,0000
PWMSET	Enhanced PWM global configuration	F1H	-	PWMRST	-	-	-	-	-	ENPWM	x0xx,xxx0
PCA_PWM0	PWM mode register of PCA0	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PWM mode register of PCA1	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PWM mode register of PCA2	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
PWMCFG	Enhanced PWM Configuration Register	F6H	-	-	-	-	PWMCBIF	EPWMCBI	ENPWMTA	PWMCEN	xxxx,0000
P7	P7 Port	F8H	P77	P76	P75	P74	P73	P72	P71	P70	1111,1111
CH	PCA Counter High Byte	F9H									0000,0000
CCAP0H	PCA module 0 high byte	FAH									0000,0000
CCAP1H	PCA module 1 high byte	FBH									0000,0000
CCAP2H	PCA module 2 high byte	FCH									0000,0000
RSTCFG	Reset Configuration Register	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		xnxn,xxnn

The following special function registers are extended SFRs whose logical addresses are in the XDATA area. Before access them, the highest bit (EAXFR) of the P_SW2 (BAH) register needs to be set, and they can accessed by using the MOVX A, @DPTR and MOVX @ DPTR, A instructions.

Symbol	Description	Address	Bit Address and Symbol								Value after Reset
			B7	B6	B5	B4	B3	B2	B1	B0	
CKSEL	Clock Select Register	FE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	Clock Divide Register	FE01H									nnnn,nnnn
HIRCCR	Internal high-speed oscillator Control Register	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	External Oscillator Control Register	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	Internal 32K Oscillator Control Register	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	Main 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 Pull-up Resistor Control Register	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 Pull-up Resistor Control Register	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 Pull-up Resistor Control Register	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 Pull-up Resistor Control Register	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	P4 Pull-up Resistor Control Register	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	P5 Pull-up Resistor Control Register	FE15H	-	-	-	P54PU	P53PU	P52PU	P51PU	P50PU	xxx0,0000
P6PU	P6 Pull-up Resistor Control Register	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU	0000,0000

P7PU	P7 Pull-up Resistor Control Register	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU	0000,0000
P0NCS	P0 Schmitt Trigger Control Register	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 Schmitt Trigger Control Register	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 Schmitt Trigger Control Register	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 Schmitt Trigger Control Register	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P4NCS	P4 Schmitt Trigger Control Register	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	P5 Schmitt Trigger Control Register	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xx00,0000
P6NCS	P6 Schmitt Trigger Control Register	FE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS	0000,0000
P7NCS	P7 Schmitt Trigger Control Register	FE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS	0000,0000
P0SR	P0 Level Shift Rate Register	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	P1 Level Shift Rate Register	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	P2 Level Shift Rate Register	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	P3 Level Shift Rate Register	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	P4 Level Shift Rate Register	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	P5 Level Shift Rate Register	FE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xx11,1111
P6SR	P6 Level Shift Rate Register	FE26H	P57SR	P66SR	P65SR	P64SR	P63SR	P62SR	P61SR	P60SR	1111,1111
P7SR	P7 Level Shift Rate Register	FE27H	P77SR	P76SR	P75SR	P74SR	P73SR	P72SR	P71SR	P70SR	1111,1111
P0DR	P0 Drive Current Control Register	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 Drive Current Control Register	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 Drive Current Control Register	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 Drive Current Control Register	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P4DR	P4 Drive Current Control Register	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111
P5DR	P5 Drive Current Control Register	FE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xx11,1111
P6DR	P6 Drive Current Control Register	FE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR	1111,1111
P7DR	P7 Drive Current Control Register	FE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR	1111,1111
P0IE	P0 Input Enable Control Register	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE	1111,1111
P1IE	P1 Input Enable Control Register	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P2IE	P2 Input Enable Control Register	FE32H	P27IE	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE	1111,1111
P3IE	P3 Input Enable Control Register	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE	1111,1111
P4IE	P4 Input Enable Control Register	FE34H	P47IE	P46IE	P45IE	P44IE	P43IE	P42IE	P41IE	P40IE	1111,1111
P5IE	P5 Input Enable Control Register	FE35H	-	-	P55IE	P54IE	P53IE	P52IE	P51IE	P50IE	xx11,1111
P6IE	P6 Input Enable Control Register	FE36H	P67IE	P66IE	P65IE	P64IE	P63IE	P62IE	P61IE	P60IE	1111,1111
P7IE	P7 Input Enable Control Register	FE37H	P77IE	P76IE	P75IE	P74IE	P73IE	P72IE	P71IE	P70IE	1111,1111

LCMIFCFG	LCM Interface Configuration Register	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFCFG2	LCM Interface Configuration Register 2	FE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000
LCMIFCR	LCM Interface Control Register	FE52H	ENLCMIF	-	-	-	-	CMD[2:0]			0xxx,x000
LCMIFSTA	LCM Interface Status Register	FE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
LCMIDDATL	LCM interface low byte data	FE54H	LCMIFDAT[7:0]								0000,0000
LCMIDDATH	LCM interface high byte data	FE55H	LCMIFDAT[15:8]								0000,0000
I2CCFG	I²C Configuration Register	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I²C Master Control Register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C Master 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	SLADR[6:0]							MA	0000,0000
I2CTXD	I²C Data Transmission Register	FE86H									0000,0000
I2CRXD	I²C Data Receive Register	FE87H									0000,0000
I2CMSAUX	I²C Master Auxiliary Control Register	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
TM2PS	Timer2 Clock Prescaler Register	FEA2H									0000,0000
TM3PS	Timer3 Clock Prescaler Register	FEA3H									0000,0000
TM4PS	Timer4 Clock Prescaler Register	FEA4H									0000,0000
ADCTIM	ADC Timing Control Register	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010
ADCEXCFG	ADC Extended Configuration Register	FEADH	-	-	ADCETRS [1:0]		-	CVTIMESEL[2:0]			xx00,x000
CMPEXCFG	Comparator Extended Configuration Register	FEAEH	CHYS[1:0]		-	-	-	CMPNS	CMPPS[1:0]		00xx,x000

PWMCH	PWM0 Counter high byte	FF00H	-							x000,0000
PWMCL	PWM0 Counter low byte	FF01H								0000,0000
PWMCKS	PWM0 clock selection	FF02H	-	-	-	SELT2	PWM_PS[3:0]			xxx0,0000
PWMTADCH	PWM0 Trigger ADC count high byte	FF03H	-							x000,0000
PWMTADCL	PWM0 Trigger ADC count low byte	FF04H								0000,0000
PWMIF	PWM0 Interrupt Flag Register	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDCR	PWM0 Anomaly Detection Control Register	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM0T1H	PWM00T1 count value high byte	FF10H	-							x000,0000
PWM0T1L	PWM00T1 count value low byte	FF11H								0000,0000
PWM0T2H	PWM00T2 count value high byte	FF12H	-							x000,0000
PWM0T2L	PWM00T2 count value low byte	FF13H								0000,0000
PWM0CR	PWM00 control register	FF14H	ENO	INI	-	PWM0PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM0HLD	PWM00 Level Hold Control Register	FF15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM1T1H	PWM01T1 count value high byte	FF18H	-							x000,0000
PWM1T1L	PWM01T1 count value low byte	FF19H								0000,0000
PWM1T2H	PWM01T2 count value high byte	FF1AH	-							x000,0000
PWM1T2L	PWM01T2 count value low byte	FF1BH								0000,0000
PWM1CR	PWM01 control register	FF1CH	ENO	INI	-	PWMPS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM1HLD	PWM01 Level Hold Control Register	FF1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM2T1H	PWM02T1 count value high byte	FF20H	-							x000,0000
PWM2T1L	PWM02T1 count value low byte	FF21H								0000,0000
PWM2T2H	PWM02T2 count value high byte	FF22H	-							x000,0000
PWM2T2L	PWM02T2 count value low byte	FF23H								0000,0000
PWM2CR	PWM02 control register	FF24H	ENO	INI	-	PWM2PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM2HLD	PWM02 Level Hold Control Register	FF25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM3T1H	PWM03T1 count value high byte	FF28H	-							x000,0000
PWM3T1L	PWM03T1 count value low byte	FF29H								0000,0000
PWM3T2H	PWM03T2 count value high byte	FF2AH	-							x000,0000
PWM3T2L	PWM03T2 count value low byte	FF2BH								0000,0000
PWM3CR	PWM03 control register	FF2CH	ENO	INI	-	PWM3PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM3HLD	PWM03 Level Hold Control Register	FF2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM4T1H	PWM04T1 count value high byte	FF30H	-							x000,0000
PWM4T1L	PWM04T1 count value low byte	FF31H								0000,0000
PWM4T2H	PWM04T2 count value high byte	FF32H	-							x000,0000
PWM4T2L	PWM04T2 count value low byte	FF33H								0000,0000
PWM4CR	PWM04 control register	FF34H	ENO	INI	-	PWM4PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM4HLD	PWM04 Level Hold Control Register	FF35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM5T1H	PWM05T1 count value high byte	FF38H	-							x000,0000
PWM5T1L	PWM05T1 count value low byte	FF39H								0000,0000
PWM5T2H	PWM05T2 count value high byte	FF3AH	-							x000,0000
PWM5T2L	PWM05T2 count value low byte	FF3BH								0000,0000
PWM5CR	PWM05 control register	FF3CH	ENO	INI	-	PWM5PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM5HLD	PWM05 Level Hold Control Register	FF3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM6T1H	PWM06T1 count value high byte	FF40H	-							x000,0000
PWM6T1L	PWM06T1 count value low byte	FF41H								0000,0000
PWM6T2H	PWM06T2 count value high byte	FF42H	-							x000,0000
PWM6T2L	PWM06T2 count value low byte	FF43H								0000,0000
PWM6CR	PWM06 control register	FF44H	ENO	INI	-	PWM6PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM6HLD	PWM06 Level Hold Control Register	FF45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM7T1H	PWM07T1 count value high byte	FF48H	-							x000,0000
PWM7T1L	PWM07T1 count value low byte	FF49H								0000,0000
PWM7T2H	PWM07T2 count value high byte	FF4AH	-							x000,0000
PWM7T2L	PWM07T2 count value low byte	FF4BH								0000,0000
PWM7CR	PWM07 control register	FF4CH	ENO	INI	-	PWM7PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM7HLD	PWM07 Level Hold Control Register	FF4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00

MD3	MDU Data register	FCF0H	MD3[7:0]								0000,0000
MD2	MDU Data register	FCF1H	MD2[7:0]								0000,0000
MD1	MDU Data register	FCF2H	MD1[7:0]								0000,0000
MD0	MDU Data register	FCF3H	MD0[7:0]								0000,0000
MD5	MDU Data register	FCF4H	MD5[7:0]								0000,0000
MD4	MDU Data register	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU Mode Control Register	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU Operation Control Register	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000
P0INTE	P0 Interrupt enable register	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 Interrupt enable register	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 Interrupt enable register	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 Interrupt enable register	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 Interrupt enable register	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 Interrupt enable register	FD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	xx00,0000
P6INTE	P6 Interrupt enable register	FD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE	0000,0000
P7INTE	P7 Interrupt enable register	FD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE	0000,0000
P0INTF	P0 Interrupt flag register	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 Interrupt flag register	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000

P2INTF	P2 Interrupt flag register	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 Interrupt flag register	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P4INTF	P4 Interrupt flag register	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 Interrupt flag register	FD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	xx00,0000
P6INTF	P6 Interrupt flag register	FD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF	0000,0000
P7INTF	P7 Interrupt flag register	FD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF	0000,0000
P0IM0	P0 Interrupt mode register0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 Interrupt mode register0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 Interrupt mode register0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 Interrupt mode register0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P4IM0	P4 Interrupt mode register0	FD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0	0000,0000
P5IM0	P5 Interrupt mode register0	FD25H	-	-	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	xx00,0000
P6IM0	P6 Interrupt mode register0	FD26H	P67IM0	P66IM0	P65IM0	P64IM0	P63IM0	P62IM0	P61IM0	P60IM0	0000,0000
P7IM0	P7 Interrupt mode register0	FD27H	P77IM0	P76IM0	P75IM0	P74IM0	P73IM0	P72IM0	P71IM0	P70IM0	0000,0000
P0IM1	P0 Interrupt mode register1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1 Interrupt mode register1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 Interrupt mode register1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 Interrupt mode register1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P4IM1	P4 Interrupt mode register1	FD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1	0000,0000
P5IM1	P5 Interrupt mode register1	FD35H	-	-	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	xx00,0000
P6IM1	P6 Interrupt mode register1	FD36H	P67IM1	P66IM1	P65IM1	P64IM1	P63IM1	P62IM1	P61IM1	P60IM1	0000,0000
P7IM1	P7 Interrupt mode register1	FD37H	P77IM1	P76IM1	P75IM1	P74IM1	P73IM1	P72IM1	P71IM1	P70IM1	0000,0000
P0WKUE	P0 Interrupt Wake-Up Enable Register	FD40H	P07WKUE	P06WKUE	P05WKUE	P04WKUE	P03WKUE	P02WKUE	P01WKUE	P00WKUE	0000,0000
P1WKUE	P1 Interrupt Wake-Up Enable Register	FD41H	P17WKUE	P16WKUE	P15WKUE	P14WKUE	P13WKUE	P12WKUE	P11WKUE	P10WKUE	0000,0000
P2WKUE	P2 Interrupt Wake-Up Enable Register	FD42H	P27WKUE	P26WKUE	P25WKUE	P24WKUE	P23WKUE	P22WKUE	P21WKUE	P20WKUE	0000,0000
P3WKUE	P3 Interrupt Wake-Up Enable Register	FD43H	P37WKUE	P36WKUE	P35WKUE	P34WKUE	P33WKUE	P32WKUE	P31WKUE	P30WKUE	0000,0000
P4WKUE	P4 Interrupt Wake-Up Enable Register	FD44H	P47WKUE	P46WKUE	P45WKUE	P44WKUE	P43WKUE	P42WKUE	P41WKUE	P40WKUE	0000,0000
P5WKUE	P5 Interrupt Wake-Up Enable Register	FD45H	-	-	P55WKUE	P54WKUE	P53WKUE	P52WKUE	P51WKUE	P50WKUE	xx00,0000
P6WKUE	P6 Interrupt Wake-Up Enable Register	FD46H	P67WKUE	P66WKUE	P65WKUE	P64WKUE	P63WKUE	P62WKUE	P61WKUE	P60WKUE	0000,0000
P7WKUE	P7 Interrupt Wake-Up Enable Register	FD47H	P77WKUE	P76WKUE	P75WKUE	P74WKUE	P73WKUE	P72WKUE	P71WKUE	P70WKUE	0000,0000

CCAPM3	PCA3 Mode Control Register	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CCAP3L	PCA3 low byte	FD55H									0000,0000
CCAP3H	PCA3 high byte	FD56H									0000,0000
PCA_PWM3	PCA3 的 PWM mode register	FD57H	EBS3[1:0]		XCCAP3H[1:0]		XCCAP3L[1:0]		EPC3H	EPC3L	0000,0000
PINIPL	I/O port interrupt priority low register	FD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O port interrupt priority high register	FD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
CHIPID0	Hard ID0	FDE0H									nnnn,nnnn
CHIPID1	Hard ID1	FDE1H									nnnn,nnnn
CHIPID2	Hard ID2	FDE2H									nnnn,nnnn
CHIPID3	Hard ID3	FDE3H									nnnn,nnnn
CHIPID4	Hard ID4	FDE4H									nnnn,nnnn
CHIPID5	Hard ID5	FDE5H									nnnn,nnnn
CHIPID6	Hard ID6	FDE6H									nnnn,nnnn
CHIPID7	Hard ID7	FDE7H									nnnn,nnnn
CHIPID8	Hard ID8	FDE8H									nnnn,nnnn
CHIPID9	Hard ID9	FDE9H									nnnn,nnnn
CHIPID10	Hard ID10	FDEAH									nnnn,nnnn
CHIPID11	Hard ID11	FDEBH									nnnn,nnnn
CHIPID12	Hard ID12	FDECH									nnnn,nnnn
CHIPID13	Hard ID13	FDEDH									nnnn,nnnn
CHIPID14	Hard ID14	FDEEH									nnnn,nnnn
CHIPID15	Hard ID15	FDEFH									nnnn,nnnn
CHIPID16	Hard ID16	FDF0H									nnnn,nnnn
CHIPID17	Hard ID17	FDF1H									nnnn,nnnn
CHIPID18	Hard ID18	FDF2H									nnnn,nnnn
CHIPID19	Hard ID19	FDF3H									nnnn,nnnn
CHIPID20	Hard ID20	FDF4H									nnnn,nnnn
CHIPID21	Hard ID21	FDF5H									nnnn,nnnn
CHIPID22	Hard ID22	FDF6H									nnnn,nnnn
CHIPID23	Hard ID23	FDF7H									nnnn,nnnn
CHIPID24	Hard ID24	FDF8H									nnnn,nnnn
CHIPID25	Hard ID25	FDF9H									nnnn,nnnn
CHIPID26	Hard ID26	FDFAH									nnnn,nnnn
CHIPID27	Hard ID27	FDFBH									nnnn,nnnn
CHIPID28	Hard ID28	FDFCH									nnnn,nnnn
CHIPID29	Hard ID29	FDFDH									nnnn,nnnn
CHIPID30	Hard ID30	FDFEH									nnnn,nnnn
CHIPID31	Hard ID31	FDFFH									nnnn,nnnn

DMA_M2M_CR	M2M_DMA control register	FA01H	ENM2M	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA status register	FA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_M2M_AMT	M2M_DMA total bytes to be transferred	FA03H									0000,0000
DMA_M2M_DONE	M2M_DMA transfer completed bytes	FA04H									0000,0000
DMA_M2M_TXAH	M2M_DMA send address high byte	FA05H									0000,0000
DMA_M2M_TXAL	M2M_DMA send address low byte	FA06H									0000,0000
DMA_M2M_RXAH	M2M_DMA receive address high byte	FA07H									0000,0000
DMA_M2M_RXAL	M2M_DMA receive address low byte	FA08H									0000,0000
DMA_ADC_CFG	ADC_DMA configuration register	FA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_CR	ADC_DMA control register	FA11H	ENADC	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_ADC_STA	ADC_DMA status register	FA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_ADC_RXAH	ADC_DMA receive address high byte	FA17H									0000,0000
DMA_ADC_RXAL	ADC_DMA receive address low byte	FA18H									0000,0000
DMA_ADC_CFG2	ADC_DMA configuration register 2	FA19H	-	-	-	-	CVTIMESEL[3:0]				xxxx,0000
DMA_ADC_CHSW0	ADC_DMA channel enable 0	FA1AH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	1000,0000
DMA_ADC_CHSW1	ADC_DMA channel enable 1	FA1BH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0	0000,0001
DMA_SPI_CFG	SPI_DMA configuration register	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_CR	SPI_DMA control register	FA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO	000x,xxx0
DMA_SPI_STA	SPI_DMA status register	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_SPI_AMT	SPI_DMA total bytes to be transferred	FA23H									0000,0000
DMA_SPI_DONE	SPI_DMA transfer completed bytes	FA24H									0000,0000
DMA_SPI_TXAH	SPI_DMA send address high byte	FA25H									0000,0000
DMA_SPI_TXAL	SPI_DMA send address low byte	FA26H									0000,0000
DMA_SPI_RXAH	SPI_DMA receive address high byte	FA27H									0000,0000
DMA_SPI_RXAL	SPI_DMA receive address low byte	FA28H									0000,0000
DMA_SPI_CFG2	SPI_DMA configuration register 2	FA29H	-	-	-	-	-	WRPSS	SSS[1:0]		xxxx,x000
DMA_UR1T_CFG	UR1T_DMA configuration register	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_CR	UR1T_DMA control register	FA31H	ENUR1T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA status register	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1T_AMT	UR1T_DMA total bytes to be transferred	FA33H									0000,0000
DMA_UR1T_DONE	UR1T_DMA transfer completed bytes	FA34H									0000,0000
DMA_UR1T_TXAH	UR1T_DMA send address high byte	FA35H									0000,0000
DMA_UR1T_TXAL	UR1T_DMA send address low byte	FA36H									0000,0000
DMA_UR1R_CFG	UR1R_DMA configuration register	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_CR	UR1R_DMA control register	FA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0

DMA_UR1R_STA	UR1R_DMA status register	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR1R_AMT	UR1R_DMA total bytes to be transferred	FA3BH									0000,0000
DMA_UR1R_DONE	UR1R_DMA transfer completed bytes	FA3CH									0000,0000
DMA_UR1R_TXAH	UR1R_DMA send address high byte	FA3DH									0000,0000
DMA_UR1R_TXAL	UR1R_DMA send address low byte	FA3EH									0000,0000
DMA_UR2T_CFG	UR2T_DMA configuration register	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_CR	UR2T_DMA control register	FA41H	ENUR2T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA status register	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA total bytes to be transferred	FA43H									0000,0000
DMA_UR2T_DONE	UR2T_DMA transfer completed bytes	FA44H									0000,0000
DMA_UR2T_TXAH	UR2T_DMA send address high byte	FA45H									0000,0000
DMA_UR2T_TXAL	UR2T_DMA send address low byte	FA46H									0000,0000
DMA_UR2R_CFG	UR2R_DMA configuration register	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_CR	UR2R_DMA control register	FA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR2R_STA	UR2R_DMA status register	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR2R_AMT	UR2R_DMA total bytes to be transferred	FA4BH									0000,0000
DMA_UR2R_DONE	UR2R_DMA transfer completed bytes	FA4CH									0000,0000
DMA_UR2R_TXAH	UR2R_DMA send address high byte	FA4DH									0000,0000
DMA_UR2R_TXAL	UR2R_DMA send address low byte	FA4EH									0000,0000
DMA_UR3T_CFG	UR3T_DMA configuration register	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_CR	UR3T_DMA control register	FA51H	ENUR3T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA status register	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA total bytes to be transferred	FA53H									0000,0000
DMA_UR3T_DONE	UR3T_DMA transfer completed bytes	FA54H									0000,0000
DMA_UR3T_TXAH	UR3T_DMA send address high byte	FA55H									0000,0000
DMA_UR3T_TXAL	UR3T_DMA send address low byte	FA56H									0000,0000
DMA_UR3R_CFG	UR3R_DMA configuration register	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000
DMA_UR3R_CR	UR3R_DMA control register	FA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR3R_STA	UR3R_DMA status register	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR3R_AMT	UR3R_DMA total bytes to be transferred	FA5BH									0000,0000
DMA_UR3R_DONE	UR3R_DMA transfer completed bytes	FA5CH									0000,0000
DMA_UR3R_TXAH	UR3R_DMA send address high byte	FA5DH									0000,0000
DMA_UR3R_TXAL	UR3R_DMA send address low byte	FA5EH									0000,0000
DMA_UR4T_CFG	UR4T_DMA configuration register	FA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_CR	UR4T_DMA control register	FA61H	ENUR4T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA status register	FA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA total bytes to be transferred	FA63H									0000,0000
DMA_UR4T_DONE	UR4T_DMA transfer completed bytes	FA64H									0000,0000
DMA_UR4T_TXAH	UR4T_DMA send address high byte	FA65H									0000,0000
DMA_UR4T_TXAL	UR4T_DMA send address low byte	FA66H									0000,0000
DMA_UR4R_CFG	UR4R_DMA configuration register	FA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_CR	UR4R_DMA control register	FA69H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR4R_STA	UR4R_DMA status register	FA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_UR4R_AMT	UR4R_DMA total bytes to be transferred	FA6BH									0000,0000
DMA_UR4R_DONE	UR4R_DMA transfer completed bytes	FA6CH									0000,0000
DMA_UR4R_TXAH	UR4R_DMA send address high byte	FA6DH									0000,0000
DMA_UR4R_TXAL	UR4R_DMA send address low byte	FA6EH									0000,0000
DMA_LCM_CFG	LCM_DMA configuration register	FA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]		0xxx,0000
DMA_LCM_CR	LCM_DMA control register	FA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	-	0000,0xxx
DMA_LCM_STA	LCM_DMA status register	FA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_LCM_AMT	LCM_DMA total bytes to be transferred	FA73H									0000,0000
DMA_LCM_DONE	LCM_DMA transfer completed bytes	FA74H									0000,0000
DMA_LCM_TXAH	LCM_DMA send address high byte	FA75H									0000,0000
DMA_LCM_TXAL	LCM_DMA send address low byte	FA76H									0000,0000
DMA_LCM_RXAH	LCM_DMA receive address high byte	FA77H									0000,0000
DMA_LCM_RXAL	LCM_DMA receive address low byte	FA78H									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: The initial value is related to the hardware options when the ISP downloads;
x: This bit does not exist, the initial value is undefined.

9 I/O Ports

There are 4 modes for all GPIOs, quasi bidirectional or weak pull-up mode (standard 8051 output mode), push-pull output / strong pull-up mode, high-impedance input mode (where current can neither flow in nor out), open drain mode. It is easy to configure the I/O mode using software.

Note: All I/O ports except for P3.0 and P3.1 are in high-impedance input state after power-on. You must set the I/O port mode before using it.

9.1 Registers Related to I/O

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	Port 0	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
P1	Port 1	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	Port 2	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	Port 3	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P4	Port 4	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
P5	Port 5	C8H	-	-	P55	P54	P53	P52	P51	P50	xx11,1111
P6	Port 6	E8H	P67	P66	P65	P64	P63	P62	P61	P60	1111,1111
P7	Port 7	F8H	P77	P76	P75	P74	P73	P72	P71	P70	1111,1111
P0M1	Port 0 mode register 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	Port 0 mode register 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P1M1	Port 1 mode register 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	Port 1 mode register 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P2M1	Port 2 mode register 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	Port 2 mode register 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
P3M1	Port 3 mode register 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	n111,1100
P3M0	Port 3 mode register 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	n000,0000
P4M1	Port 4 mode register 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	Port 4 mode register 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
P5M1	Port 5 mode register 1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	Port 5 mode register 0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx00,0000
P6M1	P6 mode registe 1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1	0000,0000
P6M0	P6 mode registe 0	CCH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0	0000,0000
P7M1	P7 mode registe 1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1	0000,0000
P7M0	P7 mode registe 0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0	0000,0000

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0PU	P0 Pull-up resistor control register	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 Pull-up resistor control register	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 Pull-up resistor control register	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 Pull-up resistor control register	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	P4 Pull-up resistor control register	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	P5 Pull-up resistor control register	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xx00,0000
P6PU	P6 Pull-up resistor control register	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU	0000,0000
P7PU	P7 Pull-up resistor control register	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU	0000,0000
P0NCS	P0 Schmitt trigger control register	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 Schmitt trigger control registe	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 Schmitt trigger control registe	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 Schmitt trigger control registe	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P4NCS	P4 Schmitt trigger control registe	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	P5 Schmitt trigger control registe	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xx00,0000
P6NCS	P6 Schmitt trigger control registe	FE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS	0000,0000
P7NCS	P7 Schmitt trigger control registe	FE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS	0000,0000
P0SR	Port0 Level Shift Rate Register	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	Port1 Level Shift Rate Register	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	Port2 Level Shift Rate Register	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	Port3 Level Shift Rate Register	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	Port4 Level Shift Rate Register	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	Port5 Level Shift Rate Register	FE25H	-	-	-	P54SR	P53SR	P52SR	P51SR	P50SR	xx11,1111
P6SR	Port6 Level Shift Rate Register	FE26H	P67SR	P66SR	P65SR	P64SR	P63SR	P62SR	P61SR	P60SR	1111,1111
P7SR	Port7 Level Shift Rate Register	FE27H	P77SR	P76SR	P75SR	P74SR	P73SR	P72SR	P71SR	P70SR	1111,1111
P0DR	P0 Drive Current Control Register	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 Drive Current Control Register	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 Drive Current Control Register	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 Drive Current Control Register	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P4DR	P4 Drive Current Control Register	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111
P5DR	P5 Drive Current Control Register	FE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xx11,1111
P6DR	P6 Drive Current Control Register	FE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR	1111,1111
P7DR	P7 Drive Current Control Register	FE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR	1111,1111
P0IE	P0 Input Enable Control Register	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P01IE	P00IE	1111,1111
P1IE	P1 Input Enable Control Register	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111

P2IE	P2 Input Enable Control Register	FE32H	P27IE	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE	1111,1111
P3IE	P3 Input Enable Control Register	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE	1111,1111
P4IE	P4 Input Enable Control Register	FE34H	P47IE	P46IE	P45IE	P44IE	P43IE	P42IE	P41IE	P40IE	1111,1111
P5IE	P5 Input Enable Control Register	FE35H	-	-	P55IE	P54IE	P53IE	P52IE	P41IE	P50IE	xx11,1111
P6IE	P6 Input Enable Control Register	FE36H	P67IE	P66IE	P65IE	P64IE	P63IE	P62IE	P41IE	P60IE	1111,1111
P7IE	P7 Input Enable Control Register	FE37H	P77IE	P76IE	P75IE	P74IE	P73IE	P72IE	P41IE	P70IE	1111,1111

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
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
P6	E8H	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
P7	F8H	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

Read and write port status

Write 0: Output low to port buffer.
Write 1: Output high to port buffer.
Read: Read the level on the port pin directly.

9.1.2 Ports Mode Registers (PxM0, PxM1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P0M1	93H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P1M0	92H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P1M1	91H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P2M0	96H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P2M1	95H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P3M0	B2H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P3M1	B1H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P4M0	B4H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P4M1	B3H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P5M0	CAH	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1
P5M1	C9H	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0
P6M0	CCH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0
P6M1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1
P7M0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0
P7M1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1

Configure the mode of the ports as shown below.

PnM1.x	PnM0.x	Pn.x mode
0	0	quasi bidirectional mode
0	1	push-pull output mode
1	0	high-impedance input mode
1	1	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 input mode. In addition, if the ADC channel still needs to be enabled after the MCU enters the power-down mode/clock stop mode, you need to set the PxIE register to close the digital input to ensure that there will be no additional power consumption.

9.1.3 Pull-up Resistor Control Registers (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
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

Internal 4.1K pull-up resistor control bit. (Note: The pull-up resistors on the P3.0 and P3.1 ports may be slightly smaller.)

0: Disable 4.1K pull-up resistor inside the port

1: Enable 4.1K pull-up resistor inside the port

9.1.4 Schmitt Trigger Control Registers (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
P4NCS	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS
P5NCS	FE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS
P6NCS	FE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS
P7NCS	FE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS

Schmitt trigger control bit:

0: Enable schmitt trigger function on the port. (Schmitt trigger is enabled by default after power-on reset.)

1: Disable schmitt trigger function on the port.

9.1.5 Level Shifting Speed Control Registers (PxSR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0	Reset value
P0SR	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,111
P1SR	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,111
P2SR	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,111
P3SR	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,111
P4SR	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,111
P5SR	FE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xx11,111
P6SR	FE26H	P57SR	P66SR	P65SR	P64SR	P63SR	P62SR	P61SR	P60SR	1111,111
P7SR	FE27H	P77SR	P76SR	P75SR	P74SR	P73SR	P72SR	P71SR	P70SR	1111,111

Level shifting speed control bits:

0: Fast level shifting, and the corresponding up and down impact will be relatively large.

1: Slow level shifting, and the corresponding up and down impact will be relatively small.

9.1.6 Drive Current Control Registers (PxDR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0	Reset value
P0DR	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,111
P1DR	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,111
P2DR	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,111
P3DR	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,111
P4DR	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,111

P5DR	FE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR	xx11,111
P6DR	FE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR	1111,111
P7DR	FE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR	1111,111

Drive capability control bit:

0: Enhanced drive ability
1: General drive ability

9.1.7 Port digital signal input enable control register (PxIE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0IE	FE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE
P1IE	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE
P2IE	FE32H	P27IE	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE
P3IE	FE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE
P4IE	FE34H	P47IE	P46IE	P45IE	P44IE	P43IE	P42IE	P41IE	P40IE
P5IE	FE35H	-	-	P55IE	P54IE	P53IE	P52IE	P41IE	P50IE
P6IE	FE36H	P67IE	P66IE	P65IE	P64IE	P63IE	P62IE	P41IE	P60IE
P7IE	FE37H	P77IE	P76IE	P75IE	P74IE	P73IE	P72IE	P41IE	P70IE

Digital signal input enable control:

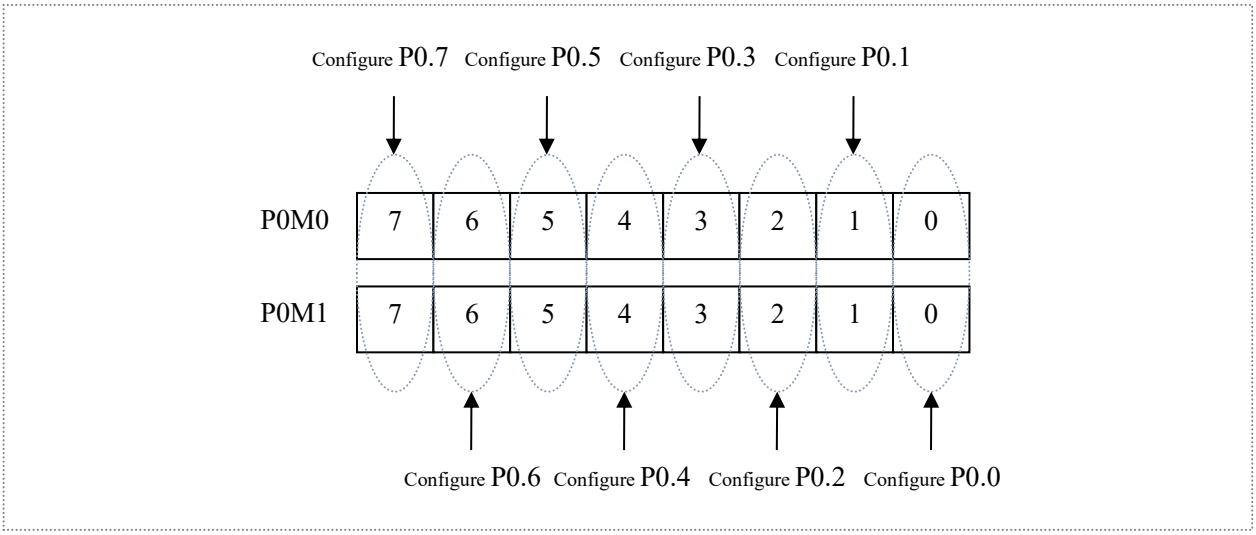
0: Disable digital signal input. If the I/O is used as an analog port such as a comparator input port, ADC input port, touch key input port or external crystal oscillator input pin, it must be set to 0 before entering the clock stop mode, otherwise there will be additional power consumption.

1: Enable digital signal input. If the I/O is used as a digital port, it must be set to 1, otherwise the MCU cannot read the level of the external port.

9.2 Configure I/O Ports

Two registers are used to configure each I/O mode.

Taking Port 0 as an example, two registers, P0M0 and P0M1, are used to configure Port 0, as shown in the following figure:



The combination of bit 0 of P0M0 and bit 0 of P0M1 is used to configure the mode of P0.0.
The combination of bit 1 of P0M0 and bit 1 of P0M1 is used to configure the mode of P0.1.
All other I/O lines configuration method is similar.

The combination of PnM0 and PnM1 to configure the I/O ports mode is as following.

PnM1	PnM0	I/O ports Mode
0	0	Quasi bidirectional (traditional 8051 I/O port, weak pull-up) Sink Current up to 20mA , Pull-up Current is 270~150μA (manufacturing error may be exist)
0	1	Push-pull output (strong pull-up output, current can be up to 20mA, resistors should be used to
1	0	high-impedance (where current can neither flow in nor out)
1	1	Open Drain mode. The internal pull-up resistors are disabled. The open drain mode can be used for both external status reading and output high or low. To read the external state correctly or output high level, the external pull-up resistors should be connected, otherwise the external state can not be read and the high level can not be output.

Note: n = 0,1,2,3,4,5,6,7

Note:

Any I/O port line can tolerate 20mA of sink current in weak pull-up mode (quasi-bidirectional mode) or strong push-pull output mode or open drain mode, and can output 20mA pull current in the strong push-pull output mode. Current limiting resistors should be connected in all I/O mode above, such as 1KΩ, 560Ω, 472Ω, etc. But the working current of the whole chip is recommended not to exceed 70mA, that is, the current flowing in from Vcc is not recommended to exceed 70mA, the current flowing from Gnd is not to exceed 70mA, and the overall current flowing in/out of it is not recommended to exceed 70mA.

9.3 I/O Ports Structure

9.3.1 Quasi-Bidirectional I/O (weak pull-up)

A quasi bidirectional port can be used as an input and output functions without the need to reconfigure the port. This is because the drive capability is weak when the port outputs a logic high level, allowing external devices to pull it low. When the pin outputs low, it has strong driving capability and able to sink a considerable current. There are three pull-up transistors in the quasi-bidirectional output to adapt different needs.

One of the three pull-up transistors, called “weak pull-up”, is turned on when the port register is logic “1” and the pin itself is logic “1”. This pull-up transistor provides the basic drive current to make the quasi-bidirectional port output logic “1”.

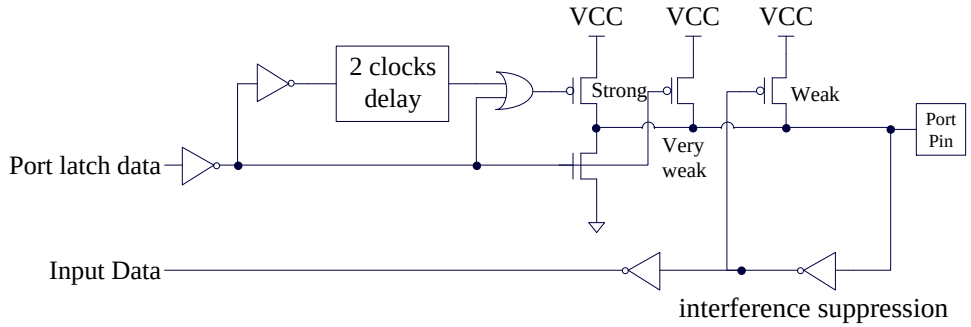
If one of the pin outputs logic “1” and the external device pulls it low, the weak pull-up transistor is off and the “very weak pull-up” maintains on. To pull the pin low, the external device must have sufficient sink capability to make the voltage on the pin drop below the threshold voltage. For a 5V microcontroller, the current of “weak pull-up” transistor is about 250uA; for a 3.3V microcontroller, the current of “weak pull-up” transistor is about 150uA.

The second pull-up transistor, called “very weak pull-up”, turns on when the port latch is “1”. When the pin is not connected, this very weak pull-up source produces a weak pull-up current that pulls the pin high. For a 5V microcontroller, the current of “weak pull-up” transistor is about 18uA; for 3.3V microcontrollers, the current of “weak pull-up” transistor is about 5uA.

The third pull-up transistor is called “strong pull-up”. This pull-up transistor is used to speed up the low-to-high transition for quasi-bidirectional port pin when the port latch changes from logic “0” to logic “1”. When this occurs, the strong pull-up transistor keeps on for about two clocks to quickly pull the pin high.

Quasi-bidirectional port (weak pull-up) has a Schmitt trigger and an interference suppression circuit. To read the correct external state, quasi-bidirectional port (weak pull-up) should latch to ‘1’ before reading.

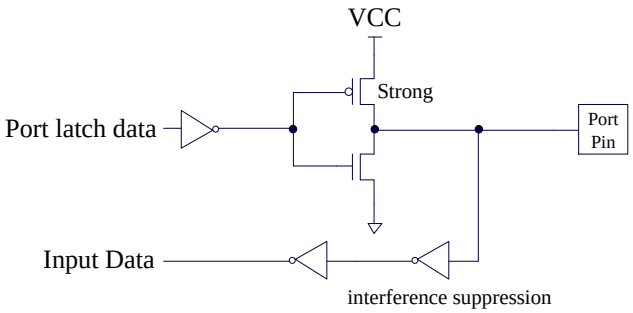
The structure of quasi-bidirectional port (weak pull-up) output is shown below:



9.3.2 Push-Pull Output

The pull-down structure of the strong push-pull output mode is the same as the pull-down structure of the open-drain output mode and quasi-bidirectional mode. However, the push-pull output mode can provide a sustained strong pull-up when the latch is logic “1”. Push-pull mode is generally used when more drive current is required.

The structure of strong push-pull pin configuration is shown below:

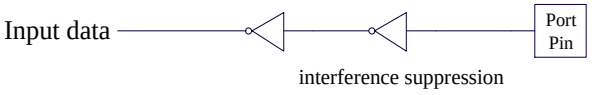


9.3.3 High Impedance Input

The current can neither flow in nor flow out.

The input port has a Schmitt trigger input and an interference suppression circuit.

The structure of high impedance input pin configuration is shown below:



9.3.4 Open-Drain Output

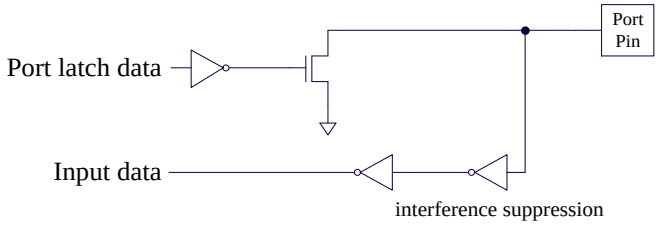
The open-drain mode can be used for both reading external status and 169utputting high or low level. To read the external state correctly or output a high level, the external pull-up resistor should be connected.

All pull-up transistors are turned off in the open-drain output configuration when the port latch is logic “0”. There must

be an external pull-up resistor in this configuration when the port outputs a logic high, typically the port pin is externally connected to VCC through a resistor. An open-drain I/O port pin can read the external state if the external pull-up resistor is connected, and the open-drain mode I/O port pin can be used as input mode. The pull-down structure in this way is the same as quasi-bidirectional mode.

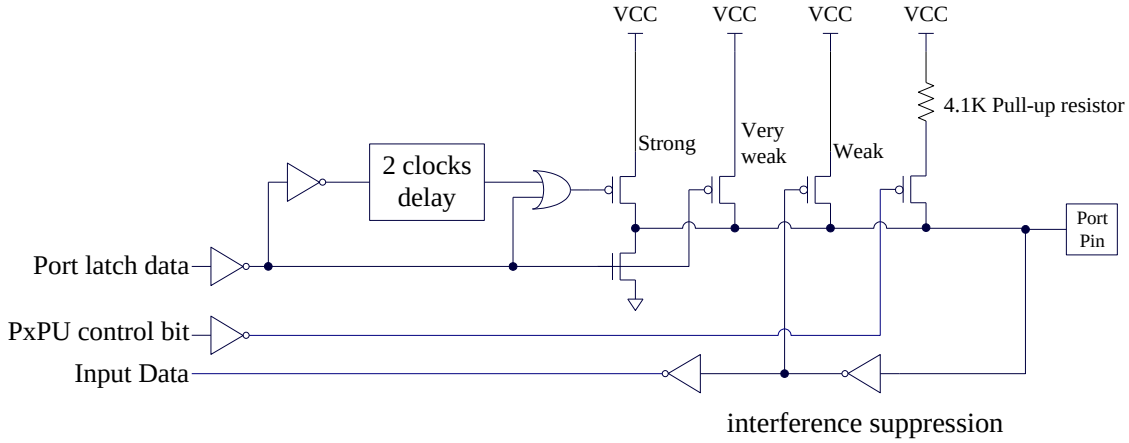
The open drain port has a Schmitt trigger input and an interference suppression circuit.

The structure of open drain port configuration is shown below:



9.3.5 4.1K Pull-up Resistor

A pull-up resistor of approximately 4.1K can be enabled internally in all I/O ports of the STC8 series (due to manufacturing errors, the range of the pull-up resistor may be 3K to 5K).



Pull-up Resistor Control Registers

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
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	-	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	FE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	FE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

Internal 4.1K pull-up resistor control bit (Note: The pull-up resistors on the P3.0 and P3.1 ports may be slightly smaller)

0: Disable 4.1K pull-up resistor inside the port

1: Enable 4.1K pull-up resistor inside the port

9.3.6 How to set the external output speed of the I/O port

When users need the I/O port to output a faster frequency, they can increase the I/O port drive current and increase the I/O port level conversion speed to increase the I/O port output speed.

Setting the PxSR register can be used to control the I/O port level conversion speed. When it is set to 0, the corresponding I/O port is fast flipping, and when it is set to 1, it is slow flipping.

Set the PxDR register, which can be used to control the drive current of the I/O port. When set to 1, I/O output is general drive current, and when set to 0, it is strong drive current.

9.3.7 How to set I/O port current drive capability

If you need to change the current drive capability of the I/O port, you can do so by setting the PxDR register

Set the PxDR register, which can be used to control the drive current of the I/O port. When set to 1, I/O output is general drive current, when set to 0, it is strong drive current

9.3.8 How to reduce the external radiation of I/O ports

Because the PxSR register is set, it can be used to control the I/O port level conversion speed, and the PxDR register can be used to control the I/O port drive current.

When the external radiation of the I/O port needs to be reduced, the PxSR register needs to be set to 1 to reduce the I/O port level conversion speed, and the PxDR register needs to be set to 1 to reduce the I/O drive current, and finally reduce I /O port external radiation

9.4 Example Routines

9.4.1 Port Mode Setting

C language code

```
//Operating frequency for test 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 P0.0 ~ P0.7 as bidirectional port mode
    P0M1 = 0x00;
    P1M0 = 0xff;                        //Set P1.0 ~ P1.7 as push-pull output mode
    P1M1 = 0x00;
    P2M0 = 0x00;                        //Set P2.0 ~ P2.7 as high impedance input mode
    P2M1 = 0xff;
    P3M0 = 0xff;                        //Set P3.0 ~ P3.7 as open-drain mode
    P3M1 = 0xff;

    while (1);
}
```

Assembly code

```
;Operating frequency for test 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 P0.0 ~ P0.7 as bidirectional port mode
            MOV         P0M1,#00H
            MOV         P1M0,#0FFH      ;Set P1.0 ~ P1.7 as push-pull output mode
            MOV         P1M1,#00H
            MOV         P2M0,#00H       ;Set P2.0 ~ P2.7 as high impedance input mode
            MOV         P2M1,#0FFH
            MOV         P3M0,#0FFH      ;Set P3.0 ~ P3.7 as open-drain mode
            MOV         P3M1,#0FFH

            JMP         $

            END
```

9.4.2 Reading and Writing Operation of Bidirection Port

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr         P0M0        =   0x94;
sfr         P0M1        =   0x93;
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        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;
```

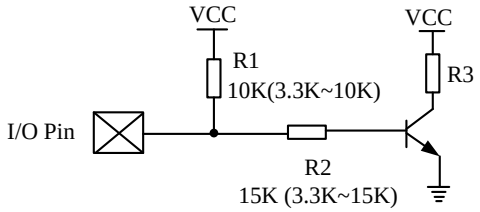
```
    P5M1 = 0x00;

    P0M0 = 0x00;                        //Set P0.0 ~ P0.7 as bidirectional port mode
    P0M1 = 0x00;

    P00 = 1;                            //P0.0 output high level
    P00 = 0;                            //P0.0 output low level

    P00 = 1;                            //Enable the internal weak pull-up resistor before reading the port
    _nop_();                            //Wait for two clocks
    _nop_();                            //
    CY = P00;                           //Read port status

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M0        DATA    094H
P0M1        DATA    093H
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     P0M0,#00H       ;Set P0.0 ~ P0.7 as bidirectional port mode
            MOV     P0M1,#00H

            SETB    P0.0            ;P0.0 output high level
            CLR     P0.0            ;P0.0 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
```

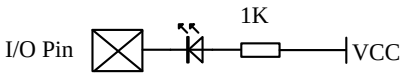
9.5 A Typical Circuit Controlled by Triode



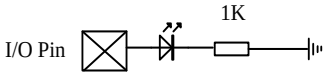
For pull-up control, it is recommended to add a pull-up resistor R1 (3.3K ～ 10K). If pull-up resistor R1 (3.3K ～ 10K) is not connected, it is recommended that the value of R2 be above 15K, or use strong push-pull output mode.

9.6 Typical Control Circuit of LED

For Quasi-Bidirectional (weak pull-up) I/O, you can drive the light-emitting diode using sink current mode, where the current limiting resistance should be greater than 1K oms, preferably not less than 470Ω.

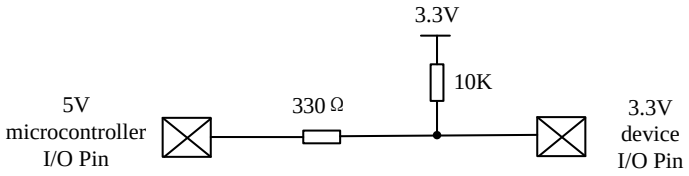


For push-pull (strong pull-up) I/O, you can drive the light-emitting diode with pull current mode.

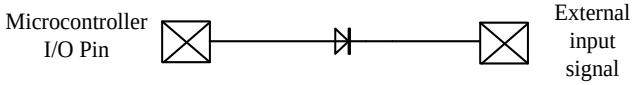


9.7 Interconnection of 3V/5V Devices in Mixed Voltage Power Supply System

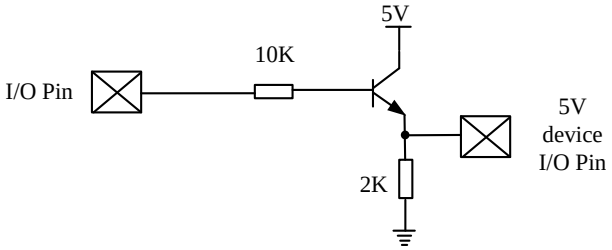
When STC's 5V microcontroller is connected to a 3.3V device, the corresponding I/O port of the 5V microcontroller can be connected with a 330Ω current limiting resistor to the 3.3V device I/O port in order to prevent the 3.3V device from withstanding 5V. The I/O port of the microcontroller is set to open-drain mode, and the internal pull-up resistor is disconnected. The corresponding 3.3V device I/O port is connected to 3.3V via with a 10K pull-up resistor. Then the high level is 3.3V, and the low level is 0V.



When STC's 3V microcontroller is connected to a 5V device and the corresponding I/O port is used as an input, an isolation diode can be connected in series to the I/O port to isolate the high voltage part in order to prevent the 3V microcontroller from bearing 5V. When the external signal voltage is higher than the microcontroller operating voltage, the isolation diode will cutoff, the state of the read I/O port is high because the I/O port is pulled up to a high level internally. When the external signal voltage is low, the isolation diode will turn on and the I/O port is clamped at 0.7V. The microcontroller reads I/O port low status as the voltage is less than 0.8V.



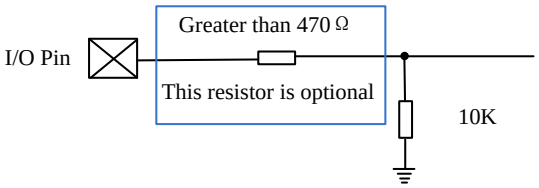
When STC's 3V microcontroller is connected to a 5V device and the corresponding I/O port is used as an output, it can be isolated with an NPN transistor in order to prevent the 3V microcontroller from bearing 5V. The circuit is as follows.



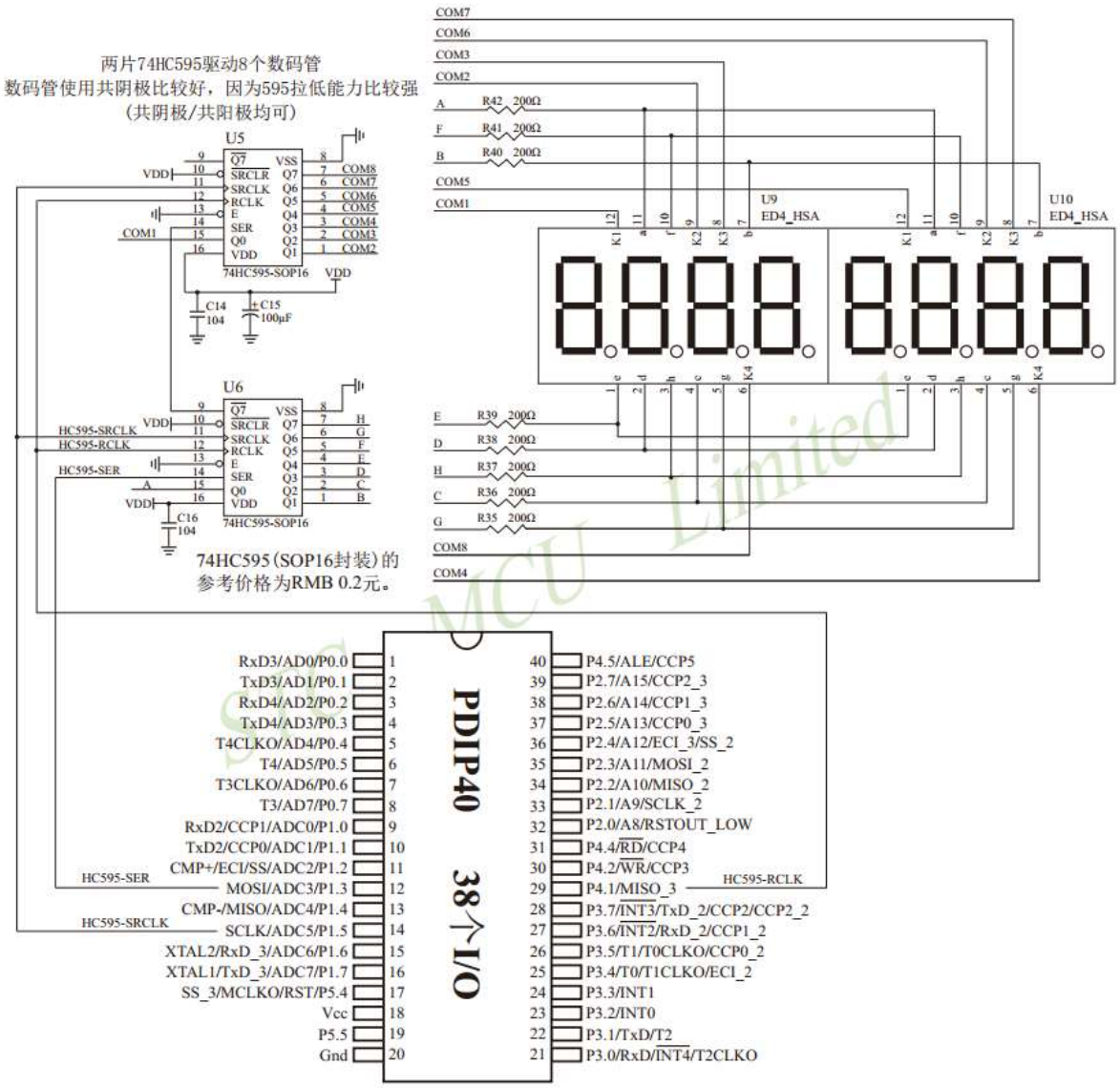
9.8 Make I/O Port Output Low When Power on Reset

The general I/O port which is a weak pull-up (quasi-bidirectional) port will output high level when the traditional 8051 microcontroller is powered on and reset. Many practical applications require that certain I/O ports be low level output when powered on, otherwise the system (such as the motor) controlled by the microcontroller will malfunction. The new generation of STC8A8K64D4 series and STC8H series microcontrollers are high-impedance inputs after all I/O resets (except for P3.0/P3.1 which are traditional weak pull-ups), adding a pull-down resistor can ensure low power when powered on If you want to change to high level later, you only need to change the mode of I/O to strong push-pull output and output high level to the outside.

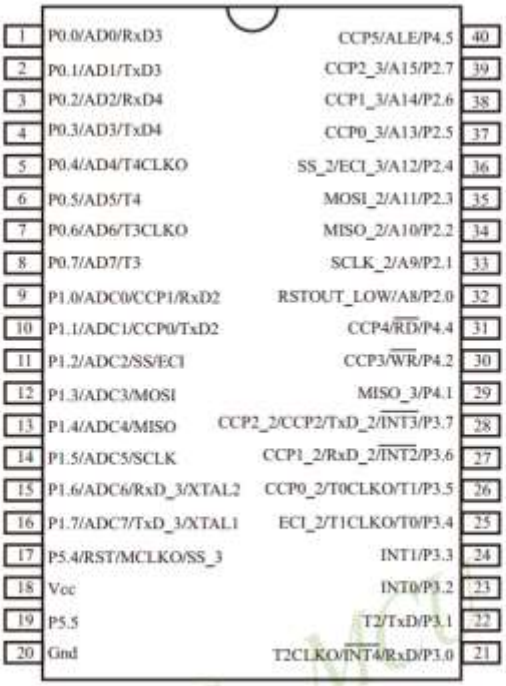
Now you can connect a pull-down resistor (about 10K) to the I/O port of STC microcontroller. In this way, during power-on reset, except the download ports P3.0 and P3.1 are weak pull-ups (quasi-bidirectional ports), other I/O ports are in high-impedance input mode, and there are external pull-down resistors, so this I/O When the port is powered on reset, the external level is low. If you want to drive this I/O port to a high level, you can set this I/O port as a strong push-pull output, and when it is a strong push-pull output, the driving current of the I/O port can reach 20mA, so you can definitely use this The port is driven as a high-level output.

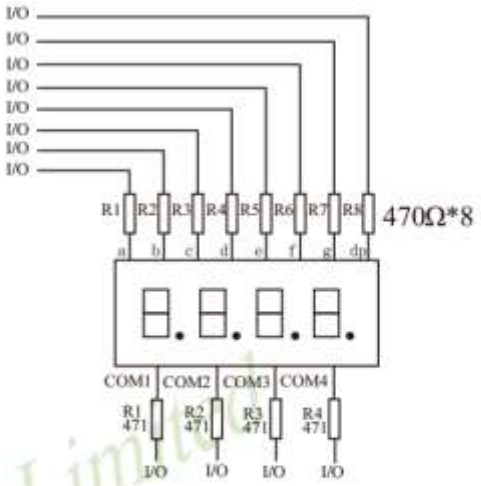


9.9 Circuit Diagram of Driving 8 Digital LEDs using 74HC595



9.10 Digital LEDs Driven Directly by I/O Port Circuit

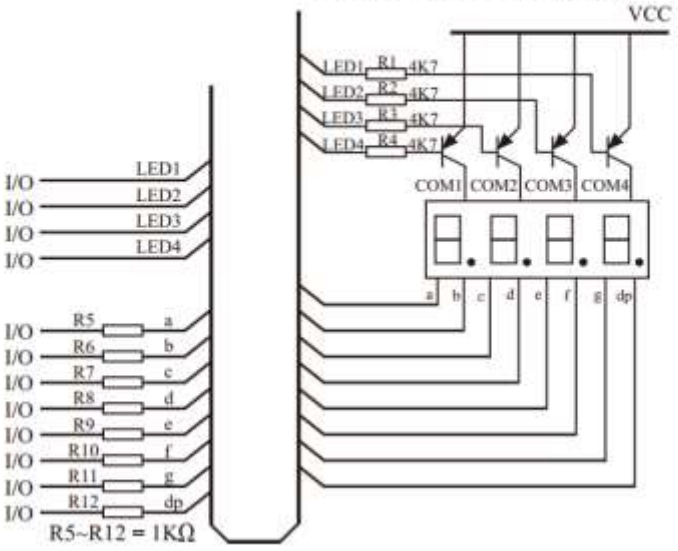




I/O口动态扫描驱动4个共阴极数码管参考电路图

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 LCD Segment LCD Driven Directly by I/O Port Circuit

An external LCD driver IC is needed when you design a product having segment LCD display requirement using MCU without LCD driver is used, which will increase cost. In fact, many small projects, such as a large number of small appliances, do not need to display a lot of segment codes. There are usually four 8 colons with a decimal point or clock: ":", so if you use the IO port to scan and display directly, it will reduce costs and work more reliably.

However, this solution is not suitable for driving too many segments (occupying too many IOs), and it is not suitable for very low power consumption occasions (driving will have hundreds of uA current).

The simple principle of segment code LCD driving is shown in Figure 1.

LCD is a special kind of liquid crystal. The arrangement direction of the crystal will be reversed under the action of an electric field, which changes its light transmittance, so that the display content can be seen. LCD has a torsional voltage threshold. The content will be displayed if the voltage across the LCD is higher than this threshold, and the content will not be displayed if the voltage is lower than this threshold. There are usually 3 parameters in LCD: working voltage, DUTY (corresponding to COM number) and BIAS (ie bias, corresponding threshold). For example, '3.0V, 1/4 DUTY, 1/3 BIAS' means that the LCD display voltage is 3.0V, 4 COM, the threshold is about 1.5V. The content will be displayed if the voltage across a certain LCD segment is 3.0V, and the content will not be displayed if the voltage is 1.0V. However, the LCD's response to the driving voltage is not very sensitive. For example, the display may be faint if the voltage is 2V. This is usually called a 'ghost image'. Therefore, it is necessary to ensure that the voltage is larger than the threshold value if you want to display something, and the voltage is smaller than the threshold value if you do not display anything.

Note: The LCD should be driven by AC, and DC voltage cannot be applied to the two ends of the LCD. Otherwise it will be damaged for a long time DC applying. The average voltage of the driving voltage applied to the LCD must be 0. Time-sharing is used to LCD. At any time, if one COM scan is valid, the other COM is invalid.

The scheme circuit for driving '1/4Duty, 1/ 2BIAS, 3V' is shown in Figure 1. The scanning principle of LCD is shown in Figure 3. The MCU works at 3.0V or 3.3V. Each COM is connected with a 20K resistor in series to a capacitor C1 aiming to obtain the midpoint voltage of 1/2VDD after RC filtering. When it is the turn of a COM scan, the connected IO is set to push-pull output mode, and the remaining COMs are set to high impedance. If the SEG connected to this COM is not used to display, the SEG output is in phase with the COM, and if it is not used to display, it is inverted. After scanning is finished, the I/O corresponding to this COM is set to high impedance. Each COM is connected to the 1/2VDD voltage on the capacitor C1 through a 20K resistor. The SEG outputs high or low level according to whether it is used to display or not. The voltage applied to the LCD segment is +-VDD when displayed, and +-1/2VDD when not displayed, which can ensure that the average DC voltage across the LCD is 0.

The circuit for driving the '1/4Duty, 1/3BIAS, 3V' is shown in Figure 4. The scanning principle of LCD is shown in Figure 5. The MCU works at 5V. The IOs connected to SEG output 1.5V and 3.5V through the resistor divider. The IOs connected to COM output 0.5V, 2.5V (at high impedance), 4.5V. The common point of the voltage-dividing resistor is connected to a capacitor C1 to abtain a mid-point voltage of 1/2VDD after RC filtering. When it is the turn of a certain COM scan, the IO is set to push-pull output mode. If the SEG connected to this COM is not used to display, the SEG output is in phase with COM, and if it is used to display, it is inverted. After scanning is finished, the I/O corresponding to this COM is set to high impedance. This COM is connected to a 2.5V voltage through a 47K resistor. The SEG outputs high or low level according to whether it is used to display or not. The voltage applied to the LCD is +-3.0V when displayed, and +-1.0V when not displaying, which can meet the LCD scanning requirements.

When sleep of power saving is required, all IOs used to dirve COMs and SEGs output low level, and the extra current will not appear in LCD drive part.

Figure 1 Circuit for driving a '1/4Duty, 1/2BIAS, 3V' LCD

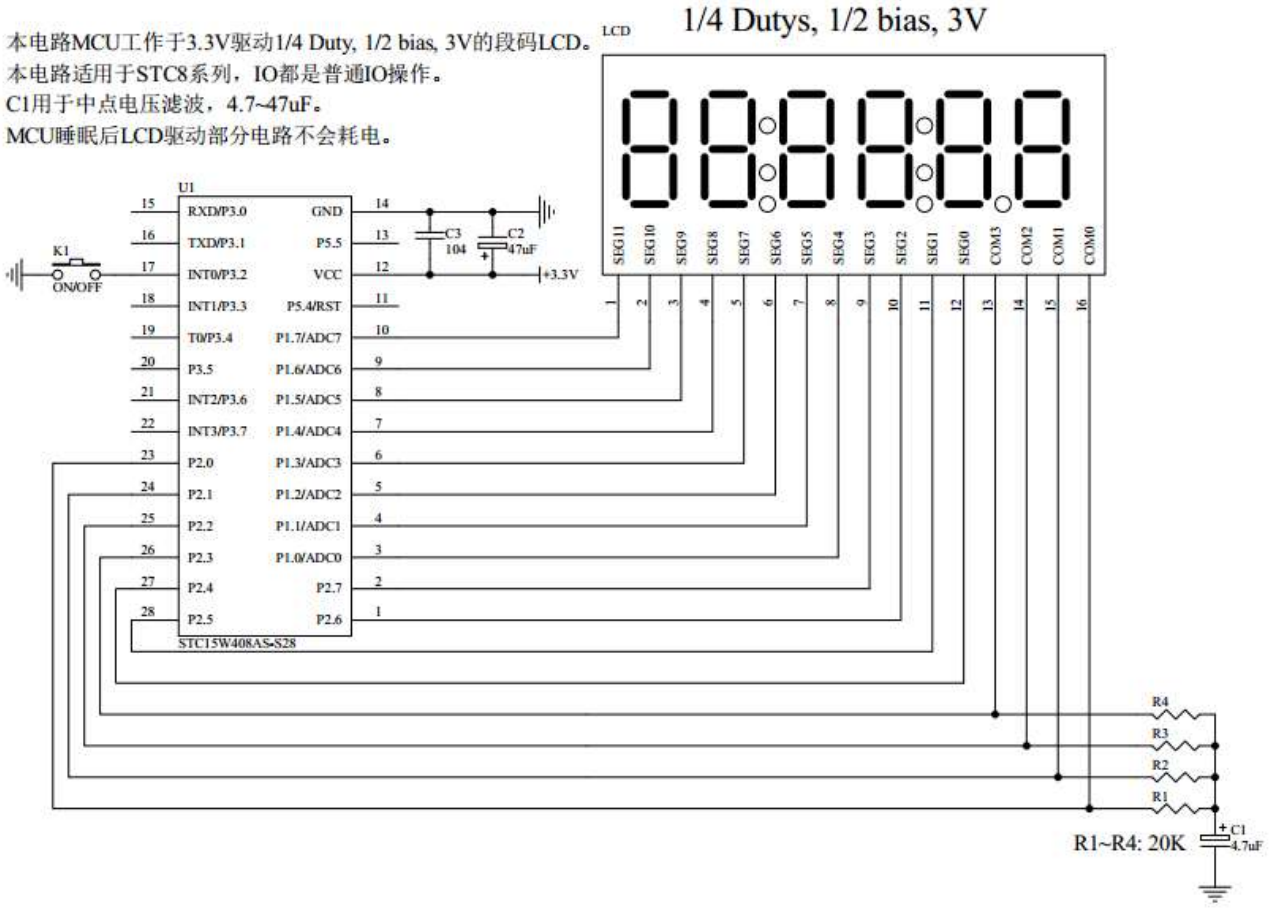


Figure 2 Segment name

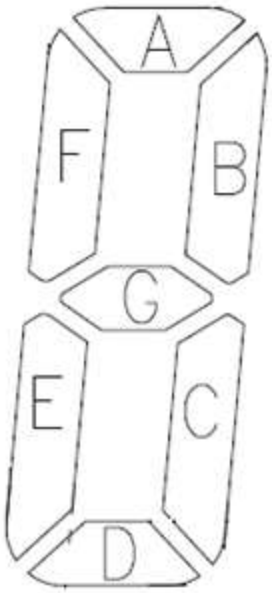


Figure 3 Principle of '1/4Duty, 1/2BIAS' scanning

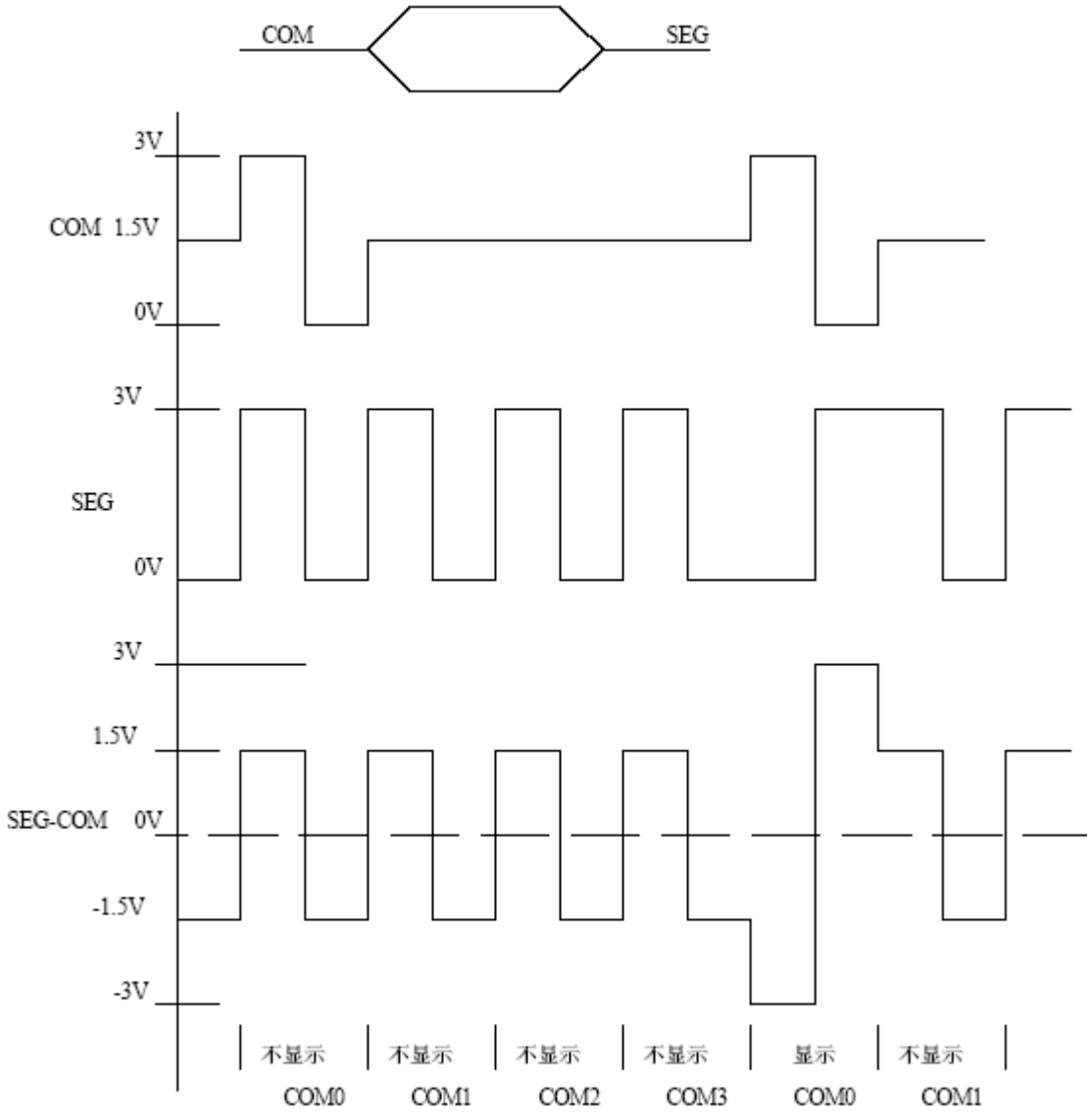


Figure 4 Driving circuit of '1/4Duty, 1/3BIAS, 3V' LCD

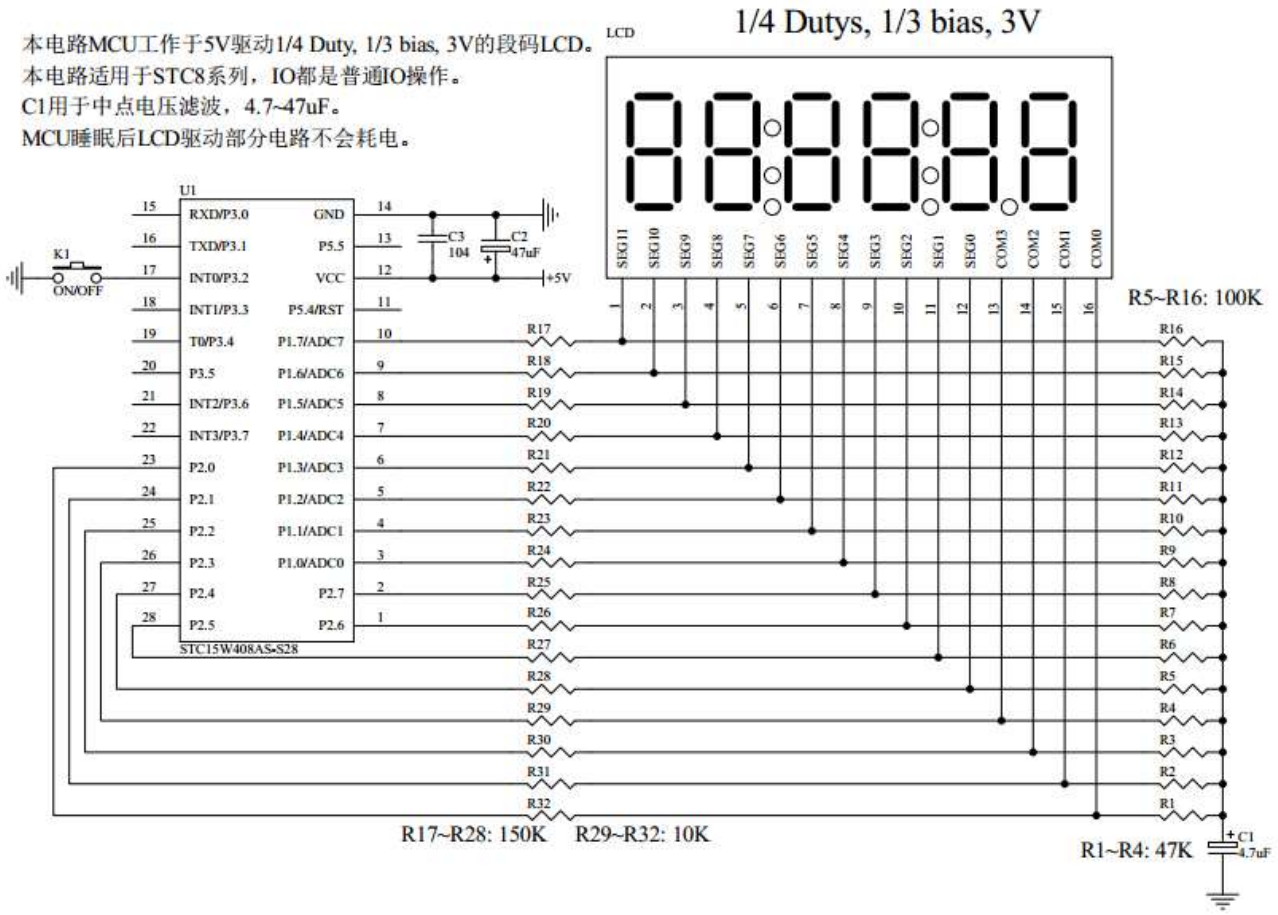
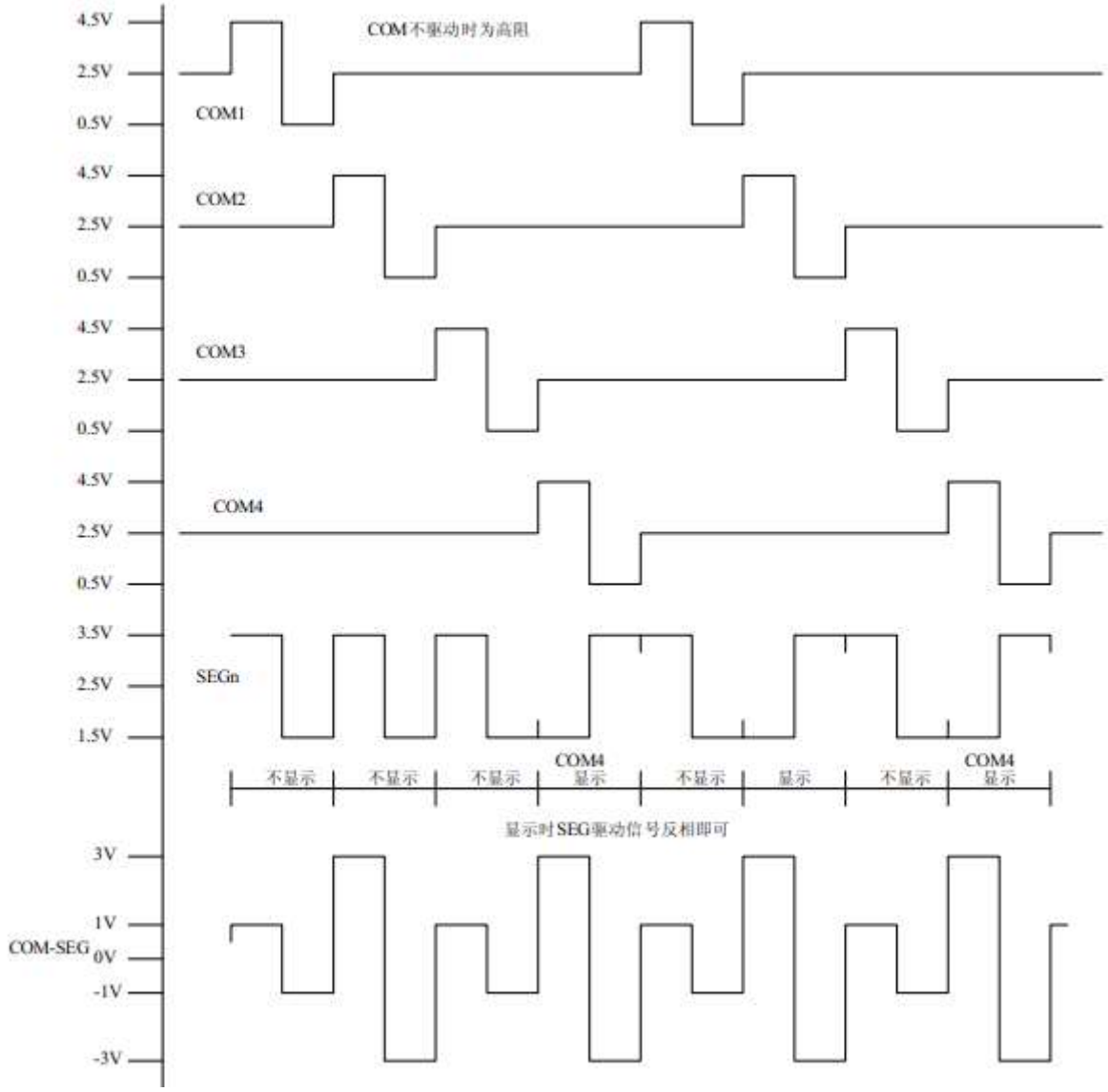


Figure 5 Principle of '1/4Duty, 1/3BIAS' scanning



For ease of use, the display contents are stored in display memory where each bit corresponds to the LCD segment one by one, as shown in Figure 6.

Figure 6 LCD truth table and memory mapping table

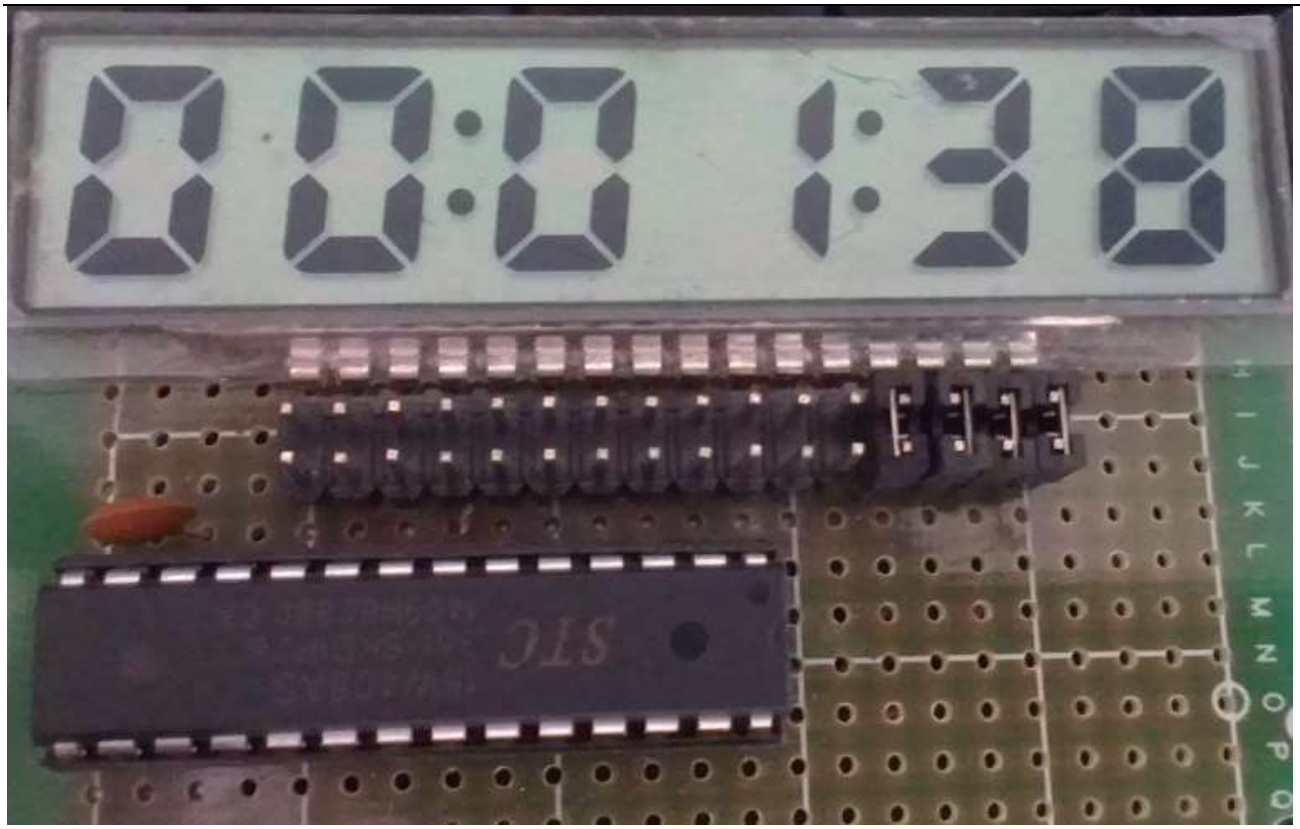
LCD真值表：

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]:	4.	5D	5.	6D	--	--	--	--
buff[5]:	5E	5C	6E	6C	--	--	--	--
buff[6]:	5G	5B	6G	6B	--	--	--	--
buff[7]:	5F	5A	6F	6A	--	--	--	--

Figure 7: Driving effect photo



Only two functions are required in the LCD scanning program:
1. LCD segment code scan function
  void LCD_scan(void)
  The program calls this function at a certain interval, and it will display the contents of the LCD display buffer on the LCD. It takes 8 calling cycles to scan all of them. The calling interval is generally 1~ 2ms. The scanning cycle is 8ms if 1ms is used. The refresh rate is 125Hz.
2. LCD segment code display buffer loading function
  void LCD_load(u8 n,u8 dat)
  This function is used to put the displayed numbers or characters in the LCD display buffer. For example, LCD_load (1,6) is to display the number 6 at the first digit position. It supports the display of 0 ~ 9, A ~ F. You can add them by yourself if you need other characters.
In addition, macros are used to display, extinguish, or flash colons or decimal points.

C language code

```
/*****************Function description*****************
STC15 series of microcontrollers are used to test segment LCD driven by I/O directly (6 8-word LCDs, 1/4 Dutys, 1/3 bias).
Time (hours, minutes and seconds) is displayed after power-on.
P3.2 is connected to ground via a switch to enter 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 main 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		LCD_buff[0] |= 0x08
#define LCD_CLR_DP2		LCD_buff[0] &= ~0x08
#define LCD_FLASH_DP2		LCD_buff[0] ^= 0x08

#define LCD_SET_DP4		LCD_buff[4] |= 0x80
#define LCD_CLR_DP4		LCD_buff[4] &= ~0x80
#define LCD_FLASH_DP4		LCD_buff[4] ^= 0x80

#define LCD_SET_2M		LCD_buff[0] |= 0x20
#define LCD_CLR_2M		LCD_buff[0] &= ~0x20
#define LCD_FLASH_2M		LCD_buff[0] ^= 0x20

#define LCD_SET_4M		LCD_buff[0] |= 0x02
#define LCD_CLR_4M		LCD_buff[0] &= ~0x02
#define LCD_FLASH_4M		LCD_buff[0] ^= 0x02

#define LCD_SET_DP5		LCD_buff[4] |= 0x20
#define LCD_CLR_DP5		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 memory
for(i=0; i<8; i++) LCD_buff[i] = 0;
P2n_push_pull(0xf0);
P1n_push_pull(0xff);                    //segment is set as push-pull output mode

LCD_SET_2M;                             //Display hour-minute division interval:
LCD_SET_4M;                             //Display minute-second division interval:
LoadRTC();                              //Display time

while (1)
{
    PCON |= 0x01;                       //Enter Idle mode, wake up and exit by Timer0 2ms
    _nop_();
    _nop_();
    _nop_();

    if(B_2ms)                           //2ms beat
    {
        B_2ms = 0;

        if(++cnt_500ms >= 250)          //reach to 500ms
        {
            cnt_500ms = 0;
        //  LCD_FLASH_2M;               //Flashing hour-minute interval:
        //  LCD_FLASH_4M;               //Flashing minute-second interval:

            B_Second = ~B_Second;
            if(B_Second)
            {
                if(++second >= 60)      //reach to 1 minute
                {
                    second = 0;
                    if(++minute >= 60)  //reach to 1 hour
                    {
                        minute = 0;
                        if(++hour >= 24) hour = 0;  //reach to 24 hours
                    }
                }
                LoadRTC();              //Display time
            }
        }

        if(!INT0)                       //key is pressed, ready to sleep
        {
            LCD_CLR_2M;                 //Display hour-minute division interval:
            LCD_CLR_4M;                 //Display minute-second division interval:
            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(!INT0) delay_ms(10);  //Waiting for the key to be released
```

```
                delay_ms(50);
                while(!INT0) delay_ms(10);          //Waiting for the key to be released once more

                TR0 = 0;                            //关闭定时器
                IE0 = 0;                            //外中断 0 标志位
                EX0 = 1;                            //INT0 Enable
                IT0 = 1;                            //INT0 下降沿中断

                P1n_push_pull(0xff);                //com 和 seg 全部输出 0
                P2n_push_pull(0xff);
                P1 = 0;
                P2 = 0;

                PCON |= 0x02;                       //Sleep
                _nop_();
                _nop_();
                _nop_();

                LCD_SET_2M;                         //Display hour-minute division interval:
                LCD_SET_4M;                         //Display minute-second division interval:
                LoadRTC();                          //Display time
                TR0 = 1;                            //Open the timer
                while(!INT0) delay_ms(10);          //Waiting for the key to be released
                delay_ms(50);
                while(!INT0) delay_ms(10);          //Waiting for the key to be released once more
            }
        }
    }
}

/*********************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 scan 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);     //All COM outputs are high impedance, and COM's voltage is the midpoint
    if(scan_index & 1)                                  // Reverse scan
    {
        P1 = ~LCD_buff[j];
        P2 = ~(LCD_buff[j]|4] & 0xf0);
    }
    else                                                // Normal phase scan
    {
        P1 = LCD_buff[j];
        P2 = LCD_buff[j]|4] & 0xf0;
    }
    P2n_push_pull(T_COM[j]);                            //A COM is set as push-pull output mode
    if(++scan_index >= 8) scan_index = 0;
}

/****************** Load display functions for numbers 1 to 6 ***************************/
void  LCD_load(u8 n, u8 dat)                            // n is the number，dat is the number to be displayed
{
    u8 code t_display[]={                               // Standard font
        // 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

```
;STC8 series of microcontrollers are used to test segment LCD driven by I/O directly (6 8-word LCDs, 1/4 Dutys, 1/3 bias).
;Time (hours, minutes and seconds) is displayed after power-on.

;*************************************************************
P0M1        DATA    0x93
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   EQU     010H
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.0               ;2ms signal
B_Second    BIT     20H.1               ;second signal
cnt_500ms   DATA    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 as a quasi-bidirectional port
                MOV         P3M0, A
                MOV         P5M1, A             ;Set as a quasi-bidirectional port
                MOV         P5M0, A

                MOV         P1M1, #0            ; segments are set as push-pull output mode
                MOV         P1M0, #0ffh
                ANL         P2M1, #NOT 0f0h     ; segments are set as push-pull output mode
                ORL         P2M0, #0f0h
                ORL         P2M1, #00fH  ; All COM outputs are high impedance, and COM's voltage is the midpoint
                ANL         P2M0, #0f0H
                MOV         SP, #0D0H
                MOV         PSW, #0
                USING       0                   ;Select bank0 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 hour-minute division interval:
;               ORL         LCD_buff, #002H     ;Display minute-second division interval:

                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
                CLR         B_2ms

                INC         cnt_500ms
                MOV         A, cnt_500ms
                CJNE        A, #250, L_Main_Loop
                                                ;reach to 500ms
                MOV         cnt_500ms, #0;
```

```
                XRL     LCD_buff, #020H         ;Flashing hour-minute interval:
                XRL     LCD_buff, #002H         ;Flashing minute-second interval:

                CPL     B_Second
                JNB     B_Second, L_Main_Loop

                INC     second
                MOV     A, second
                CJNE    A, #60, L_Main_Load
                MOV     second, #0              ; reach to 1 minute
                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                ;reach to 24 hours
L_Main_Load:
                LCALL   F_LoadRTC               ;Display time
                LJMP    L_Main_Loop

;**************************************************************

F_Timer0_init:
                CLR     TR0                     ; Stop counting
                ANL     TMOD, #0f0H
                SETB    ET0                     ; Enable interrupt
                ORL     TMOD, #0                ; Working mode 0: 16-bit auto-reload
                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 operation
                RET

;**************************************************************
F_Timer0_Interrupt:     ;Timer0 1ms interrupt function
                PUSH    PSW                     ;push PSW into stack
                PUSH    ACC                     ;push ACC into 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                     ;pop ACC from stack
                POP     PSW                     ;pop PSW from stac
                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 is the number, which is 1~6, 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 is the number, which is 1~6, 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 is the number, which is 1~6, 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 is the number, which is 1~6, 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 is the number, which is 1~6, 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 is the number, which is 1~6, 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 outputs are high impedance, and COM's voltage is the
midpoint
        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:                                    ; Normal 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 COM is set as push-pull output mode
                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*****************************
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

;******************** Load display functions for numbers 1 to 6, and the algorithm is simple ****************************
F_LCD_load:                                     ;R6 is the number, which is 1~6, 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                    ;the number to be displayed

                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
                MOV     A, @R0
                MOV     C, B.2                  ;C
```

```
        MOV     ACC.6, C
        MOV     C, B.4          ;E
        MOV     ACC.7, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.6, C
        MOV     C, B.6          ;G
        MOV     ACC.7, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.6, C
        MOV     C, B.5          ;F
        MOV     ACC.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
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.4, C
        MOV     C, B.4          ;E
        MOV     ACC.5, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.4, C
        MOV     C, B.6          ;G
        MOV     ACC.5, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.4, C
        MOV     C, B.5          ;F
        MOV     ACC.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
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.2, C
        MOV     C, B.4          ;E
        MOV     ACC.3, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.2, C
        MOV     C, B.6          ;G
        MOV     ACC.3, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.2, C
        MOV     C, B.5          ;F
        MOV     ACC.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
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.0, C
        MOV     C, B.4          ;E
        MOV     ACC.1, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.0, C
        MOV     C, B.6          ;G
        MOV     ACC.1, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.0, C
        MOV     C, B.5          ;F
        MOV     ACC.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
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.6, C
        MOV     C, B.4          ;E
        MOV     ACC.7, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.6, C
        MOV     C, B.6          ;G
        MOV     ACC.7, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.6, C
        MOV     C, B.5          ;F
        MOV     ACC.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
        MOV     A, @R0
        MOV     C, B.2          ;C
        MOV     ACC.4, C
        MOV     C, B.4          ;E
        MOV     ACC.5, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.1          ;B
        MOV     ACC.4, C
        MOV     C, B.6          ;G
        MOV     ACC.5, C
        MOV     @R0, A

        INC     R0
        MOV     A, @R0
        MOV     C, B.0          ;A
        MOV     ACC.4, C
        MOV     C, B.5          ;F
```

```
        MOV     ACC.5, C
        MOV     @R0, A
        RET
L_NotLoadChar6:
        RET

                                END
```

10 Instruction Set

Mnemonic		Description	Bytes	Cycle
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add ditect byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add ditect byte to Accumulator with Carry	2	1
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	1
ADDC	A,#data	Add immediate data to Accumulator with Carry	2	1
SUBB	A,Rn	Subtract Register from Accumulator with borrow	1	1
SUBB	A,direct	Subtract direct byte from Accumulator with borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from Accumulator with borrow	1	1
SUBB	A,#data	Substract immediate data from Accumulator with borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement Register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	1
MUL	AB	Multiply A & B, high byte of result is in B, low byte in A	1	2
DIV	AB	Divde A by B, quotient is in A, remainder is in B.	1	6
DA	A	Decimal Adjust Accumulator	1	3
ANL	A,Rn	AND Register to Accumulator	1	1
ANL	A,direct	AND direct btye to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	1
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	1
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	1
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	1

CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate Accumulator Left through the Carry	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate Accumulator Right through the Carry	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1
CLR	C	Clear Carry	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry	1	1
SETB	bit	Set direct bit	2	1
CPL	C	Complement Carry	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry	2	1
ANL	C,/bit	AND complement of direct bit to Carry	2	1
ORL	C,bit	OR direct bit to Carry	2	1
ORL	C,/bit	OR complement of direct bit to Carry	2	1
MOV	C,bit	Move direct bit to Carry	2	1
MOV	bit,C	Move Carry to direct bit	2	1
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Move immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	1
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	1
MOV	direct,direct	Move direct byte to direct	3	1
MOV	direct,@Ri	Move indirect RAM to direct byte	2	1
MOV	direct,#data	Move immediate data to direct byte	3	1
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	1
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Move 16-bit immdiate data to indirect RAM	3	1
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Accumulator	1	4
MOVC	A,@A+PC	Move Code byte relative to PC to Accumulator	1	3
MOVX	A,@Ri	Move extended RAM(8-bit addr) to Accumulator (Read)	1	3[1]
MOVX	A,@DPTR	Move extended RAM(16-bit addr) to Accumulator (Read)	1	2[1]
MOVX	@Ri,A	Move Accumulator to extended RAM(8-bit addr) (Write)	1	3[1]
MOVX	@DPTR,A	Move Accumulator to extended RAM(16-bit addr) (Write)	1	2[1]
PUSH	direct	Push direct byte onto stack	2	1
POP	direct	POP direct byte from stack	2	1
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	1
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Accumulator	1	1
ACALL	addr11	Absolute Subroutine Call	2	3
LCALL	addr16	Long Subroutine Call	3	3

RET		Return from Subroutine	1	3
RETI		Return from interrupt	1	3
AJMP	addr11	Absolute Jump	2	3
LJMP	addr16	Long Jump	3	3
SJMP	rel	Short Jump (relative addr)	2	3
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	4
JZ	rel	Jump if Accumulator is Zero	2	1/3[2]
JNZ	rel	Jump if Accumulator is not Zero	2	1/3[2]
JC	rel	Jump if Carry is set	2	1/3[2]
JNC	rel	Jump if Carry not set	2	1/3[2]
JB	bit,rel	Jump if direct bit is set	3	1/3[2]
JNB	bit,rel	Jump if direct bit is not set	3	1/3[2]
JBC	bit,rel	Jump if direct bit is set & clear bit	3	1/3[2]
CJNE	A,direct,rel	Compare direct byte to Accumulator and jump if not equal	3	2/3[3]
CJNE	A,#data,rel	Compare immediate data to Accumulator and Jump if not equal	3	1/3[2]
CJNE	Rn,#data,rel	Compare immediate data to register and Jump if not equal	3	2/3[3]
CJNE	@Ri,#data,rel	Compare immediate data to indirect and jump if not equal	3	2/3[3]
DJNZ	Rn,rel	Decrement register and jump if not Zero	2	2/3[3]
DJNZ	direct,rel	Decrement direct byte and Jump if not Zero	3	2/3[3]
NOP		No Operation	1	1

[1]: When accessing external extended RAM, the instruction execution cycle is related to the SPEED [1: 0] bits in the BUS_SPEED register.

[2]: For the conditional jump statement, the execution cycle will be different based on whether the conditions are met or not. When the conditions are not met, the jump will not occur and continue to execute the next instruction, then execution cycle of the conditional jump statement is 1 machine cycle. When the conditions are met, the jump will occur, the execution cycle of the conditional jump statement is 3 machine cycles.

[3]: For the conditional jump statement, the execution cycle will be different based on whether the conditions are met or not. When the conditions are not met, the jump will not occur and continue to execute the next instruction, then execution cycle of the conditional jump statement is 1 machine cycle. When the conditions are met, the jump will occur, the execution cycle of the conditional jump statement is 3 machine cycles.

11 Interrupt System

(An error will be reported when compiled in Keil when using an interrupt with an interrupt number greater than 31 in a C program. Please refer to Appendix for the solution.)

The interrupt system is set up to give the CPU real-time processing capabilities for external emergencies.

If an emergency request occurrs while CPU is dealing with something, the CPU is required to suspend the current work to handle the emergency. After the emergency processing is completed, the CPU returns to the place where it was interrupted and continues the original work. This process is called interrupt. The componet that implements this function is called the interrupt system. The request source that makes the CPU interrupt to suspend the current work is called the interrupt source. Microcontroller interrupt system generally allows multiple interrupt sources. When several interrupt sources simultaneously require the CPU to handle the requests, the CPU should response the interrupt source which has the highest priority. Usually the CPU handle the interrupt requests according to the priority of interrupt sources. The most urgent incidents have the highest priority. Each interrupt source has a priority level. The CPU always responds the highest priority interrupt request.

Another interrupt source request with a higher priority takes place while the CPU is processing an interrupt source request, that is, the CPU is executing the corresponding interrupt service routine, if the CPU can suspend the original interrupt service routine, and deal with the higher priority interrupt request source, and then return to the original low-level interrupt service routine after processing finished, this process is called interrupt nesting. Such an interrupt system is called a multi-level interrupt system, whereas an interrupt system without interrupt nesting is called a single-level interrupt system.

The corresponding interrupt request can be masked by turning off the general enable bit (EA / IE.7) or the corresponding interrupt enable bit. The CPU can be enabled to respond to the corresponding interrupt request by turning on the corresponding interrupt enable bit. Every interrupt source can be set or reset independently by software to interrupt enabled or disabled state. The priority of some interrupts can be set by software. Higher priority interrupt requests can interrupt lower priority interrupts, whereas lower priority interrupt requests can not interrupt higher priority interrupts. When two interrupts with the same priority ocuur simultaneously, the inquiry order determines which interrupt the system responds firstly.

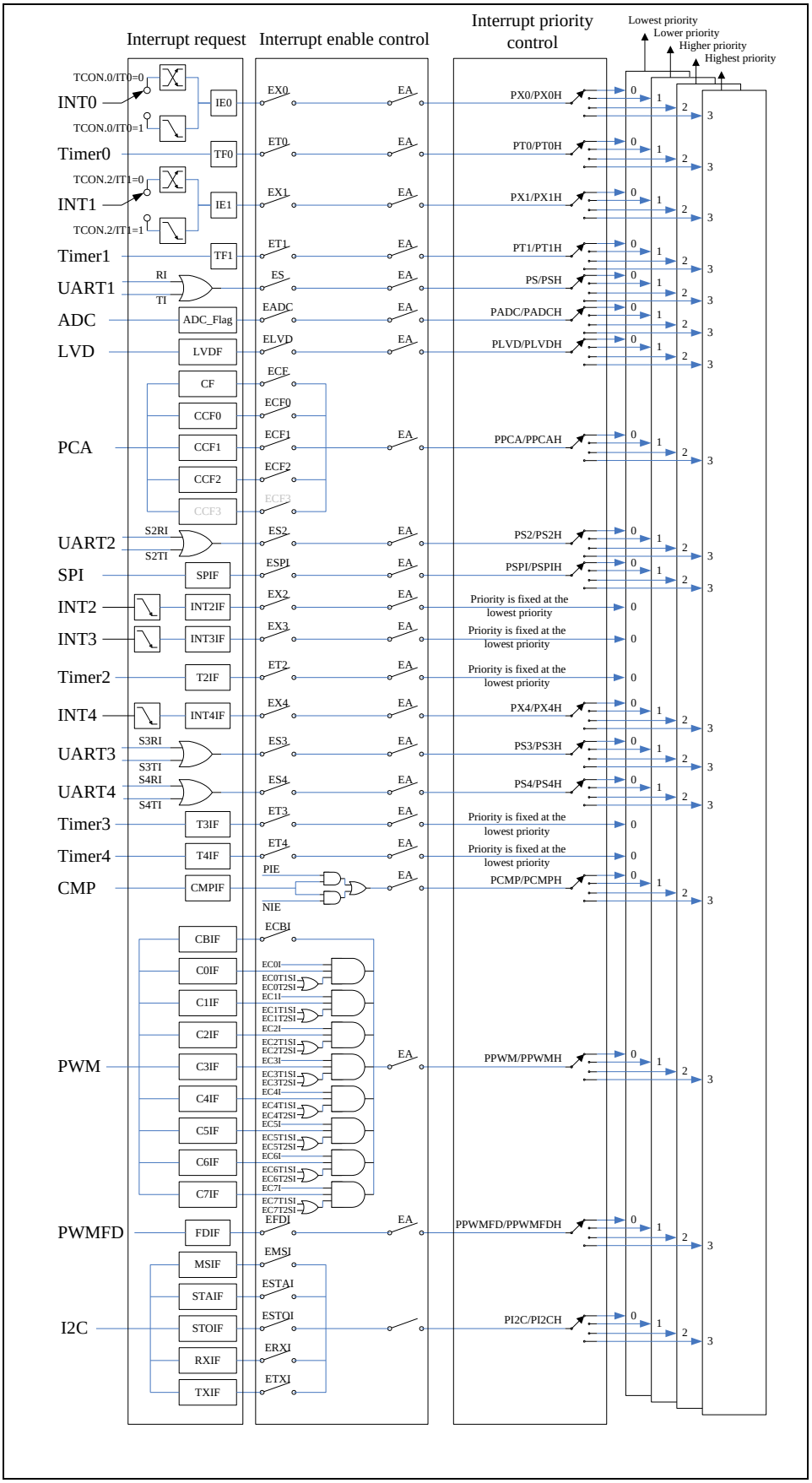
11.1 Interrupt sources of STC8A8K64D4 series

The √ in the following table indicates that the corresponding series have the corresponding interrupt source.

Interrupt sources	STC8A8K64S4 family
External interrupt 0 (INT0) Supports falling and edges interrupts	√
Timer 0 interrrupt (Timer0)	√
External interrupt 1 (INT1) Supports falling and edges interrupts	√
Timer 1 interrrupt (Timer1)	√
UART1 interrupt (UART1)	√
ADC interrupt (ADC)	√
Low voltage detection interrupt (LVD)	√
CCP/PCA/PWM interrupt (CCP/PCA) Supports falling, rising and edges interrupts	√
UART2 interrupt (UART2)	√
SPI interrupt (SPI)	√
External interrupt 2 (INT2) Supports falling edge interrupts	√
External interrupt 3 (INT3) Supports falling edge interrupts	√
Timer 2 interrrupt (Timer2)	√
External interrupt 4 (INT4)	√
UART3 interrupt (UART3)	√
UART4 interrupt (UART4)	√

Timer 3 interrrupt (Timer3)	√
Timer 4 interrrupt (Timer4)	√
Comparator interrupt (CMP)	√
Enhanced PWM interrupt	√
PWM abnormal detection interrupt	√
I2C interrupt	√
Port0 interrupt Supports falling edge, rising edge, high level and low level interrupts	√
Port1 interrupt Supports falling edge, rising edge, high level and low level interrupts	√
Port2 interrupt Supports falling edge, rising edge, high level and low level interrupts	√
Port3 interrupt Supports falling edge, rising edge, high level and low level interrupts	√
Port4 interrupt Supports falling edge, rising edge, high level and low level interrupts	√
Port5 interrupt Supports falling edge, rising edge, high level and low level interrupts	√
Port6 interrupt Supports falling edge, rising edge, high level and low level interrupts	
Port7 interrupt Supports falling edge, rising edge, high level and low level interrupts	
DMA_M2M interrupt	√
DMA_ADC interrupt	√
DMA_SPI interrupt	√
DMA_UR1T interrupt	√
DMA_UR1R interrupt	√
DMA_UR2T interrupt	√
DMA_UR2R interrupt	√
DMA_UR3T interrupt	√
DMA_UR3R interrupt	√
DMA_UR4T interrupt	√
DMA_UR4R interrupt	√
DMA_LCM interrupt	√
LCM interrupt	√

11.2 Structure of STC8A8K64D4 Interrupt



11.3 Interrupt List of STC8A8K64D4 Series

Interrupt source	Interrupt vector	Order	Priority level setup bit	Priority level	Interrupt request flag	Interrupt enable 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
UART3	008BH	17	PS3,PS3H	0/1/2/3	S3RI || S3TI	ES3
UART4	0093H	18	PS4,PS4H	0/1/2/3	S4RI || S4TI	ES4
Timer3	009BH	19		0	T3IF	ET3
Timer4	00A3H	20		0	T4IF	ET4
CMP	00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE|NIE
PWM	00B3H	22	PPWM,PPWMH	0/1/2/3	CBIF	ECBI
					C0IF	EC0I && EC0T1SI
						EC0I && EC0T2SI
					C1IF	EC1I && EC1T1SI
						EC1I && EC1T2SI
					C2IF	EC2I && EC2T1SI
						EC2I && EC2T2SI
					C3IF	EC3I && EC3T1SI
						EC3I && EC3T2SI
					C4IF	EC4I && EC4T1SI
						EC4I && EC4T2SI
					C5IF	EC5I && EC5T1SI
						EC5I && EC5T2SI
					C6IF	EC6I && EC6T1SI
						EC6I && EC6T2SI
					C7IF	EC7I && EC7T1SI
						EC7I && EC7T2SI
PWMFD	00BBH	23	PPWMFD,PPWMFDH	0/1/2/3	FDIF	EFDI

Interrupt source	Interrupt	Order	Priority level	Priority	Interrupt	Interrupt enable

	vector		setup bit	level	request flag	bit
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI
P0 interrupt	012BH	37	P0IP,P0IPH	0/1/2/3	P0INTF	P0INTE
P1 interrupt	0133H	38	P1IP,P1IPH	0/1/2/3	P1INTF	P1INTE
P2 interrupt	013BH	39	P2IP,P2IPH	0/1/2/3	P2INTF	P2INTE
P3 interrupt	0143H	40	P3IP,P3IPH	0/1/2/3	P3INTF	P3INTE
P4 interrupt	014BH	41	P4IP,P4IPH	0/1/2/3	P4INTF	P4INTE
P5 interrupt	0153H	42	P5IP,P5IPH	0/1/2/3	P5INTF	P5INTE
P6 interrupt	015BH	43	P6IP,P6IPH	0/1/2/3	P6INTF	P6INTE
P7 interrupt	0163H	44	P7IP,P7IPH	0/1/2/3	P7INTF	P7INTE
DMA_M2M interrupt	017BH	47	M2MIP[1:0]	0/1/2/3	M2MIF	M2MIE
DMA_ADC interrupt	0183H	48	ADCIP[1:0]	0/1/2/3	ADCIF	ADCIE
DMA_SPI interrupt	018BH	49	SPIIP[1:0]	0/1/2/3	SPIIF	SPIIE
DMA_UR1T interrupt	0193H	50	UR1TIP[1:0]	0/1/2/3	UR1TIF	UR1TIE
DMA_UR1R interrupt	019BH	51	UR1RIP[1:0]	0/1/2/3	UR1RIF	UR1RIE
DMA_UR2T interrupt	01A3H	52	UR2TIP[1:0]	0/1/2/3	UR2TIF	UR2TIE
DMA_UR2R interrupt	01ABH	53	UR2RIP[1:0]	0/1/2/3	UR2RIF	UR2RIE
DMA_UR3T interrupt	01B3H	54	UR3TIP[1:0]	0/1/2/3	UR3TIF	UR3TIE
DMA_UR3R interrupt	01BBH	55	UR3RIP[1:0]	0/1/2/3	UR3RIF	UR3RIE
DMA_UR4T interrupt	01C3H	56	UR4TIP[1:0]	0/1/2/3	UR4TIF	UR4TIE
DMA_UR4R interrupt	01CBH	57	UR4RIP[1:0]	0/1/2/3	UR4RIF	UR3RIE
DMA_LCM interrupt	01D3H	58	LCMIP[1:0]	0/1/2/3	LCMIF	LCMIE
LCM interrupt	01DBH	59	LCMIFIP[1:0]	0/1/2/3	LCMIFIF	LCMIFIE

Interrupt service routine may be declared in C language as the following,

```
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    UART3_Routine(void)         interrupt  17;
void    UART4_Routine(void)         interrupt  18;
void    TM3_Routine(void)           interrupt  19;
void    TM4_Routine(void)           interrupt  20;
void    CMP_Routine(void)           interrupt  21;
void    PWM_Routine(void)           interrupt  22;
void    PWMFD_Routine(void)         interrupt  23;
void    I2C_Routine(void)           interrupt  24;
```

Interrupt service routines with interrupt numbers greater than 31 cannot be directly declared in C language. Please refer to the processing method in "Appendix J". Assembly language is not affected.

11.4 Registers Related to Interrupt

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	Interrupt enable register	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	Interrupt enable register2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	0000,0000
INTCLKO	External interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
IP	Interrupt Priority Low Byte	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
IPH	Interrupt Priority High Byte	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP2	2nd Interrupt Priority register low byte	B5H	-	PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2	0000,0000
IP2H	2nd Interrupt Priority register high byte	B6H	-	PI2CH	PCMPH	PX4H	PPWMFDH	PPWMH	PSPIH	PS2H	0000,0000
IP3	3nd Interrupt Priority register low byte	DFH	-	-	-	-	-	-	PS4	PS3	xxxx,x000
IP3H	3nd Interrupt Priority Register High Byte	EEH	-	-	-	-	-	-	PS4H	PS3H	xxxx,x000
AUXINTIF	Extended External Interrupt Flag Register	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	UART1 control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	UART2 control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S3CON	UART3 control register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	Serial port 4 control register	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	Timer 0 and 1 control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
SPSTAT	SPI Status register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
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 Control Register	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA Module 1 Control Register	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA Module 2 Control Register	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CMPCR1	Comparator Control Register 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
PWMCFG	Enhanced PWM Configuration Register	F6H	-	-	-	-	PWMCBIF	EPWMCBI	ENPWMTA	PWMCEN	xxxx,0000

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
LCMIFCFG	LCM Interface Configuration Register	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFSTA	LCM Interface Status Register	FE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
I2CMSCR	I2C Master Control Register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I2C Master 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
PWMIF	PWM0 Interrupt Flag Register	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDCR	PWM0 Abnormal Detection Control Register	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM0CR	PWM00 Control Register	FF14H	ENO	INI	-	PWM0PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM1CR	PWM01 Control Register	FF1CH	ENO	INI	-	PWMPS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM2CR	PWM02 Control Register	FF24H	ENO	INI	-	PWM2PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM3CR	PWM03 Control Register	FF2CH	ENO	INI	-	PWM3PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM4CR	PWM04 Control Register	FF34H	ENO	INI	-	PWM4PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM5CR	PWM05 Control Register	FF3CH	ENO	INI	-	PWM5PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM6CR	PWM06 Control Register	FF44H	ENO	INI	-	PWM6PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
PWM7CR	PWM07 Control Register	FF4CH	ENO	INI	-	PWM7PS[1:0]		ENI	ENT2I	ENT1I	00x0,0000
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
P4INTE	P4 port interrupt enable register	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 port interrupt enable register	FD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	xx00,0000
P6INTE	P6 port interrupt enable register	FD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE	0000,0000
P7INTE	P7 port interrupt enable register	FD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE	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
P4INTF	P4 Port Interrupt Flag Register	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 Port Interrupt Flag Register	FD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	xx00,0000
P6INTF	P6 Port Interrupt Flag Register	FD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF	0000,0000
P7INTF	P7 Port Interrupt Flag Register	FD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF	0000,0000
CCAPM3	PCA Module 3 Mode Control Register	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
PINIPL	I/O port interrupt priority low register	FD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O port interrupt priority high register	FD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
DMA_M2M_CFG	M2M DMA configuration register	FA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_ADC_CFG	ADC_DMA configuration register	FA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_SPI_CFG	SPI_DMA configuration register	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_UR1T_CFG	UR1T_DMA configuration register	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1R_CFG	UR1R_DMA configuration register	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR2T_CFG	UR2T_DMA configuration register	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000

DMA_UR2R_CFG	UR2R_DMA configuration register	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR3T_CFG	UR3T_DMA configuration register	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3R_CFG	UR3R_DMA configuration register	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000
DMA_UR4T_CFG	UR4T_DMA configuration register	FA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4R_CFG	UR4R_DMA configuration register	FA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_LCM_CFG	LCM_DMA configuration register	FA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]		0xxx,0000
DMA_M2M_STA	M2M_DMA status register	FA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_ADC_STA	ADC_DMA status register	FA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_SPI_STA	SPI_DMA status register	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_UR1T_STA	UR1T_DMA status register	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1R_STA	UR1R_DMA status register	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR2T_STA	UR2T_DMA status register	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2R_STA	UR2R_DMA status register	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR3T_STA	UR3T_DMA status register	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3R_STA	UR3R_DMA status register	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR4T_STA	UR4T_DMA status register	FA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4R_STA	UR4R_DMA status register	FA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_LCM_STA	LCM_DMA status register	FA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00

11.4.1 Interrupt Enable Registers (Interrupt Enable bits)

IE (Interrupt Enable Rsgister)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

EA: The general or global interrupt enable control bit. The function of EA is to allow interrupts to be multi-level controlled. That is, every interrupt source is controlled by EA firstly and then by its own interrupt enable control bit.
0: All interrupts are masked.
1: Enable the CPU interrupt, every interrupt source would be individually enabled or disabled by setting or clearing its enable bit.

ELVD: Low volatge detection interrupt enable bit.
0: disable low voltage detection interrupt.
1: enable Low voltage detection interrupt.

EADC: ADC interrupt enable bit.
0: disable ADC interrupt.
1: enable ADC interrupt.

ES: UART1 interrupt enable bit.
0: disable UART1 interrupt.
1: enable UART1 interrupt.

ET1: Timer 1 interrupt enable bit.
0: disable Timer 1 interrupt.
1: enable Timer 1 interrupt.

EX1: External interrupt 1 enable bit.
0: disable external interrupt 1.
1: enable external interrupt 1.

ET0: Timer 0 interrupt enable bit.
0: disable Timer 0 interrupt.
1: enable Timer 0 interrupt.

EX0: External interrupt 0 enable bit.
0: disable external interrupt 0.
1: enable external interrupt 0.

IE2 (Interrupt Enable Rsgister 2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

ET4: Timer 4 interrupt enable bit.
0: disable Timer 4 interrupt.
1: enable Timer 4 interrupt.

ET3: Timer 3 interrupt enable bit.
0: disable Timer 3 interrupt.

1: enable Timer 3 interrupt.

ES4: UART4 interrupt enable bit.

0: disable UART4 interrupt.

1: enable UART4 interrupt.

ES3: UART3 interrupt enable bit.

0: disable UART3 interrupt.

1: enable UART3 interrupt.

ET2: Timer 2 interrupt enable bit.

0: disable Timer 2 interrupt.

1: enable Timer 2 interrupt.

ESPI: SPI interrupt enalbe bit.

0: disable SPI interrupt.

1: enable SPI interrupt.

ES2: UART2 interrupt enable bit.

0: disable UART2 interrupt.

1: enable UART2 interrupt.

INTCLKO (External interrupt and clock output control register)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4: External Interrupt 4 enabel bit.

0: disable External Interrupt 4.

1: enable External Interrupt 4.

EX3: External Interrupt 3 enabel bit.

0: disable External Interrupt 3.

1: enable External Interrupt 3.

EX2: External Interrupt 2 enabel bit.

0: disable External Interrupt 2.

1: enable External Interrupt 2.

PCA/CCP/PWM Interrupt Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CMOD	D9H	CIDL	-	-	-	CPS[2:0]			ECF
CCAPM0	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0
CCAPM1	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1
CCAPM2	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2
CCAPM3	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3

ECF: PCA counter interrupt enable bit.

0: disable PCA counter interrupt

1: enable PCA counter interrupt

ECCF0: PCA module 0 interrupt enable bit.

0: disable PCA module 0 interrupt

1: enable PCA module 0 interrupt

ECCF1: PCA module 1 interrupt enable bit.

0: disable PCA module 1 interrupt

1: enable PCA module 1 interrupt

ECCF2：PCA module 2 interrupt enable bit.

0: disable PCA module 2 interrupt

1: enable PCA module 2 interrupt

ECCF3: PCA module 3 interrupt enable bit.

0: disable PCA module 3 interrupt

1: enable PCA module 3 interrupt

CMPCR1 (Comparator Control Register 1)

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

PIE: Comparator rising-edge interrupt enable bit.

0: disable comparator rising-edge interrupt.

1: enable comparator rising-edge interrupt.

NIE: Comparator falling-edge interrupt enable bit.

0: disable comparator falling-edge interrupt.
1: enable comparator falling-edge interrupt.

Enhanced PWM Configuration Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG	F6H	-	-	-	-	PWMCBIF	EPWMCBI	ENPWMTA	PWMCEN

EPWMCBI: Enhanced PWM0 counter interrupt enable bit.

0: disable PWM0 counter interrupt
1: enable PWM0 counter interrupt

Enhanced PWM Abnormal Detection Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

EFDI: PWM external abnormal event interrupt enable bit.

0: disable PWM external abnormal event interrupt
1: enable PWM external abnormal event interrupt

Enhanced PWM Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF14H	ENO	INI	-	PWM0PS[1:0]		ENI	ENT2I	ENT1I
PWM1CR	FF1CH	ENO	INI	-	PWMPS[1:0]		ENI	ENT2I	ENT1I
PWM2CR	FF24H	ENO	INI	-	PWM2PS[1:0]		ENI	ENT2I	ENT1I
PWM3CR	FF2CH	ENO	INI	-	PWM3PS[1:0]		ENI	ENT2I	ENT1I
PWM4CR	FF34H	ENO	INI	-	PWM4PS[1:0]		ENI	ENT2I	ENT1I
PWM5CR	FF3CH	ENO	INI	-	PWM5PS[1:0]		ENI	ENT2I	ENT1I
PWM6CR	FF44H	ENO	INI	-	PWM6PS[1:0]		ENI	ENT2I	ENT1I
PWM7CR	FF4CH	ENO	INI	-	PWM7PS[1:0]		ENI	ENT2I	ENT1I

ENI: PWM channel interrupt enable bit.

0: disable PWM interrupt
1: enable PWM interrupt

ENT2SI: PWM channel 2nd trigger point interrupt enable bit.

0: disable the 2nd trigger point interrupt of PWM
1: enable the 2nd trigger point interrupt of PWM

ENT1SI: PWM channel 1st trigger point interrupt enable bit.

0: disable the 1st trigger point interrupt of PWM
1: enable the 1st trigger point interrupt of PWM

I2C Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	MSCMD[3:0]			
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI: I^2C master mode interrupt enable bit.

0: disable I^2C master mode interrupt.
1: enable I^2C master mode interrupt.

ESTAI: I^2C slave receives the START event interrupt enable bit.

0: disable I^2C slave receives the START event interrupt.
1: enable I^2C slave receives the START event interrupt.

ERXI: I^2C slave completes receiving data event interrupt enable bit.

0: disable I^2C slave completes receiving data event interrupt.
1: enable I^2C slave completes receiving data event interrupt.

ETXI: I^2C slave completes transmitting data event interrupt enable bit.

0: disable I^2C slave completes transmitting data event interrupt.
1: enable I^2C slave completes transmitting data event interrupt.

ESTOI: I^2C slave receives STOP event interrupt enable bit.

0: disable I^2C slave receives STOP event interrupt.
1: enable I^2C slave receives STOP event interrupt.

Port interrupt enable registers

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
P4INTE	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	FD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE
P6INTE	FD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE
P7INTE	FD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE

PnINTE.x: Port interrupt enable control bit (n=0~7, x=0~7)

0: disable Pn.x port interrupt function

1: Enable Pn.x port interrupt function

LCM interface configuration register

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

LCMIFIE: LCM interface interrupt enable bit.

0: disable LCM interface interrupt

1: Enable LCM interface interrupt

DMA interrupt enable registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	FA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	
DMA_ADC_CFG	FA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]	
DMA_SPI_CFG	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	
DMA_UR1T_CFG	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	
DMA_UR1R_CFG	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	
DMA_UR2T_CFG	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	
DMA_UR2R_CFG	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	
DMA_UR3T_CFG	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	
DMA_UR3R_CFG	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	
DMA_UR3R_CFG	FA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	
DMA_UR4R_CFG	FA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	
DMA_LCM_CFG	FA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]	

M2MIE: DMA_M2M (Memory-to-Memory DMA) interrupt enable bit.

0: disable DMA_M2M interrupt

1: enable DMA_M2M interrupt

ADCIE: DMA_ADC (ADC DMA) interrupt enable bit.

0: disable DMA_ADC interrupt

1: enable DMA_ADC interrupt

SPIIE: DMA_SPI (SPI DMA) interrupt enable bit.

0: disable DMA_SPI interrupt

1: enable DMA_SPI interrupt

UR1TIE: DMA_UR1T (UART 1 send DMA) interrupt enable bit.

0: disable DMA_UR1T interrupt

1: enable DMA_UR1T interrupt

UR1RIE: DMA_UR1R (UART 1 receive DMA) interrupt enable bit.

0: disable DMA_UR1R interrupt

1: enable DMA_UR1R interrupt

UR2TIE: DMA_UR2T (UART 2 send DMA) interrupt enable bit.

0: disable DMA_UR2T interrupt

1: enable DMA_UR2T interrupt

UR2RIE: DMA_UR2R (UART 2 receive DMA) interrupt enable bit.

0: disable DMA_UR2R interrupt

1: enable DMA_UR2R interrupt

UR3TIE: DMA_UR3T (UART 3 send DMA) interrupt enable bit.

0: disable DMA_UR3T interrupt

1: enable DMA_UR3T interrupt

UR3RIE: DMA_UR3R (UART 3 receive DMA) interrupt enable bit.

0: disable DMA_UR3R interrupt
1: enable DMA_UR3R interrupt

UR4TIE: DMA_UR4T (UART 4 send DMA) interrupt enable bit.

0: disable DMA_UR4T interrupt
1: enable DMA_UR4T interrupt

UR4RIE: DMA_UR4R (UART 4 receive DMA) interrupt enable bit.

0: disable DMA_UR4R interrupt
1: enable DMA_UR4R interrupt

LCMIE: DMA_LCM (LCM interface DMA) interrupt enable bit.

0: disable DMA_LCM interrupt
1: enable DMA_LCM interrupt

11.4.2 Interrupt Request Registers (Interrupt flags)

Timer 0 and 1 Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1: Timer/Counter 1 Overflow Flag. Set by hardware on Timer/Counter 1 overflow. It will be automatically cleared by the hardware when processor enters the Timer 1 interrupt service routine.

TF0: Timer/Counter 0 Overflow Flag. Set by hardware on Timer/Counter 0 overflow. It will be automatically cleared by the hardware when processor enters the Timer 1 interrupt service routine.

IE1: External Interrupt 1 request flag. It will be automatically cleared when the processor enters the external interrupt 1 service routine.

IE0: External Interrupt 0 request flag. It will be automatically cleared when the processor enters the external interrupt 0 service routine.

Auxiliary Interrupt Flag Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
AUXINTIF	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF

INT4IF: external interrupt 4 request flag, which must be cleared by software.

INT3IF: external interrupt 3 request flag, which must be cleared by software.

INT2IF: external interrupt 2 request flag, which must be cleared by software.

T4IF: timer 4 overflow interrupt flag, which must be cleared by software (Note: This bit is a write-only register, not readable).

T3IF: timer 3 overflow interrupt flag, which must be cleared by software (Note: This bit is a write-only register, not readable).

T2IF: timer 2 overflow interrupt flag, which must be cleared by software (Note: This bit is a write-only register, not readable).

Notice:

Ealy 1T 8051 MCU using 0.35um process, STC15 series added a 16-bit reload timer which was the world's first big innovation of 8051. Due to high manufacturing cost, STC 16-bit reloadable timer 2/3/4 did not design the interrupt request flag registers for users to access. The interrupt request flag register has only internal hidden flags. The method provided to the user software to clear the internal hidden flags is: when the user software disables the timer 2/3/4 interrupt, the hardware automatically clears the internal timer 2/3/4. Hide interrupt request flags.

For product consistency:

The STC8A/ STC8F and subsequent STC8G/STC8H/ STC8C/ STC12H series which adopt 0.18um process add an interrupt request flag register accessible by the timer 2/3/4 user, but when the timer 2/3/4 interrupt is disabled , the function of the internal hidden interrupt request flag bit of the hardware automatic clear timer 2/3/4 is still retained. Therefore, do not arbitrarily disable the timer 2/3/4 interrupt when the timer 2/3/4 does not stop counting, otherwise the hidden interrupt request flag that actually works will be cleared. It is possible that after the counter overflows again, there is also a case that after the

hidden interrupt request flag is set to 1, and request an interrupt and wait, it is mistakenly cleared by the user.

This is different from the traditional INTEL8048, 8051, but INTEL has been discontinued, so the new design of STC does not consider the specifications compatible with traditional INTEL.

This is the further development of 8051 by China STC.

UARTs Control Registers

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
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

TI: Transmit interrupt flag of UART1, which must be cleared by software.

RI: Receive interrupt flag of UART1, which must be cleared by software.

S2TI: Transmit interrupt flag of UART2, which must be cleared by software.

S2RI: Receive interrupt flag of UART2, which must be cleared by software.

S3TI: Transmit interrupt flag of UART3, which must be cleared by software.

S3RI: Receive interrupt flag of UART3, which must be cleared by software.

S4TI: Transmit interrupt flag of UART4, which must be cleared by software.

S4RI: Receive interrupt flag of UART4, which must be cleared by software.

Power Control 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 flag, which must be cleared by software.

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 completes conversion interrupt request flag, which must be cleared by software.

SPI Status Register

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

SPIF: SPI transmission completion interrupt request flag, which must be cleared by software.

PCA Control Register

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

CF: PCA counter interrupt request flag, which must be cleared by software.

CCF3: PCA module 3 interrupt request flag, which must be cleared by software.

CCF2: PCA module 2 interrupt request flag, which must be cleared by software.

CCF1: PCA module 1 interrupt request flag, which must be cleared by software.

CCF0: PCA module 0 interrupt request flag, which must be cleared by software.

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, which must be cleared by software.

I2C Status Registers

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

MSIF: I^2C master mode interrupt request flag, which must be cleared by software.

ESTAI: I^2C slave receives the START event interrupt request flag, which must be cleared by software.

ERXI: I²C slave completes receiving data event interrupt request flag, which must be cleared by software.

ETXI: I²C slave completes transmitting data event interrupt request flag, which must be cleared by software.

ESTOI: I²C slave receives the STOP event interrupt request flag, which should be must by software.

Enhanced PWM Configuration Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG	F6H	-	-	-	-	PWMCBIF	EPWMCBI	ENPWMTA	PWMCEN

PWMCBIF: Enhanced PWM0 counter interrupt request flag, which must be cleared by software.

Enhanced PWM Interrupt Flag Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMIF	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

C7IF: Enhanced PWM channel 7 interrupt request flag, which must be cleared by software.

C6IF: Enhanced PWM channel 6 interrupt request flag, which must be cleared by software.

C5IF: Enhanced PWM channel 5 interrupt request flag, which must be cleared by software.

C4IF: Enhanced PWM channel 4 interrupt request flag, which must be cleared by software.

C3IF: Enhanced PWM channel 3 interrupt request flag, which must be cleared by software.

C2IF: Enhanced PWM channel 2 interrupt request flag, which must be cleared by software.

C1IF: Enhanced PWM channel 1 interrupt request flag, which must be cleared by software.

C0IF: Enhanced PWM channel 0 interrupt request flag, which must be cleared by software.

Enhanced PWM Abnormal Detection Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

FDIF: Enhanced PWM abnormal detection interrupt request flag, which must be cleared by software.

Port interrupt flag registers

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
P4INTF	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF
P5INTF	FD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF
P6INTF	FD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF
P7INTF	FD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF

PnINTF.x: Port interrupt request flag (n=0~7, x=0~7)

0: No interrupt request for Pn.x port

1: Pn.x port has an interrupt request, if the interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software.

LCM Interface Status Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFSTA	FE53H	-	-	-	-	-	-	-	LCMIFIF

LCMIFIF: LCM interface interrupt request flag. Need to be cleared by software.

DMA interrupt flag registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	FA02H	-	-	-	-	-	-	-	M2MIF
DMA_ADC_STA	FA12H	-	-	-	-	-	-	-	ADCIF
DMA_SPI_STA	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF
DMA_UR1T_STA	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF
DMA_UR1R_STA	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF
DMA_UR2T_STA	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF

DMA_UR2R_STA	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF
DMA_UR3T_STA	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF
DMA_UR3R_STA	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF
DMA_UR4T_STA	FA62H	-	-	-	-	-	TXOVW	-	UR4TIF
DMA_UR4R_STA	FA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF
DMA_LCM_STA	FA72H	-	-	-	-	-	-	TXOVW	LCMIF

M2MIF: DMA_M2M(Memory-to-Memory DMA) interrupt request flag. Need to be cleared by software.

ADCIF: DMA_ADC(ADC DMA) interrupt request flag. Need to be cleared by software.

SPIIF: DMA_SPI(SPI DMA) interrupt request flag. Need to be cleared by software.。

UR1TIF: DMA_UR1T(UART1 send DMA) interrupt request flag. Need to be cleared by software.

UR1RIF: DMA_UR1R(UART1 receive DMA) interrupt request flag. Need to be cleared by software.

UR2TIF: DMA_UR2T(UART2 send DMA) interrupt request flag. Need to be cleared by software.

UR2RIF: DMA_UR2R(UART2 receive DMA) interrupt request flag. Need to be cleared by software.

UR3TIF: DMA_UR3T(UART3 send DMA) interrupt request flag. Need to be cleared by software.

UR3RIF: DMA_UR3R(UART3 receive DMA) interrupt request flag. Need to be cleared by software.

UR4TIF: DMA_UR4T(UART4 send DMA) interrupt request flag. Need to be cleared by software.

UR4RIF: DMA_UR4R(UART4 receive DMA) interrupt request flag. Need to be cleared by software.

LCMIF: DMA_LCM(LCM interface DMA) interrupt request flag. Need to be cleared by software.

11.4.3 Interrupt Priority Registers

Except INT2, INT3, Timer 2, Timer 3 and Timer 4, all other interrupts have 4 levels of interrupt priority that can be set.

Interrupt Priority Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	PPCAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	-	PI2C	PCMP	PX4	PPWMFD	PPWM	PSPI	PS2
IP2H	B6H	-	PI2CH	PCMPH	PX4H	PPWMFDH	PPWMH	PSPIH	PS2H
IP3	DFH	-	-	-	-	-		PS4	PS3
IP3H	EEH	-	-	-	-	-		PS4H	PS3H

PX0H,PX0: External interrupt 0 interrupt priority control bit.

00: INT0 interrupt priority level is 0 (lowest)
01: INT0 interrupt priority level is 1 (lower)
10: INT0 interrupt priority level is 2 (higher)
11: INT0 interrupt priority level is 3 (highest)

PT0H,PT0: Timer 0 interrupt priority control bit.

00: Timer 0 interrupt priority level is 0 (lowest)
01: Timer 0 interrupt priority level is 1 (lower)
10: Timer 0 interrupt priority level is 2 (higher)
11: Timer 0 interrupt priority level is 3 (highest)

PX1H,PX1: External interrupt 1 interrupt priority control bit.

00: INT1 interrupt priority level is 0 (lowest)
01: INT1 interrupt priority level is 1 (lower)
10: INT1 interrupt priority level is 2 (higher)
11: INT1 interrupt priority level is 3 (highest)

PT1H,PT1: Timer 1 interrupt priority control bit.

00: Timer 1 interrupt priority level is 0 (lowest)
01: Timer 1 interrupt priority level is 1 (lower)
10: Timer 1 interrupt priority level is 2 (higher)
11: Timer 1 interrupt priority level is 3 (highest)

PSH,PS: UART1 interrupt priority control bit.

00: UART1 interrupt priority level is 0 (lowest)
01: UART1 interrupt priority level is 1 (lower)
10: UART1 interrupt priority level is 2 (higher)
11: UART1 interrupt priority level is 3 (highest)

PADCH,PADC: ADC interrupt priority control bit.

00: ADC interrupt priority level is 0 (lowest)
01: ADC interrupt priority level is 1 (lower)
10: ADC interrupt priority level is 2 (higher)
11: ADC interrupt priority level is 3 (highest)

PLVDH,PLVD: Low voltage detection interrupt priority control bit.

00: LVD interrupt priority level is 0 (lowest)
01: LVD interrupt priority level is 1 (lower)
10: LVD interrupt priority level is 2 (higher)
11: LVD interrupt priority level is 3 (highest)

PPCAH,PPCA: CCP/PCA/PWM interrupt priority control bit.

00: CCP/PCA/PWM interrupt priority level is 0 (lowest)
01: CCP/PCA/PWM interrupt priority level is 1 (lower)
10: CCP/PCA/PWM interrupt priority level is 2 (higher)
11: CCP/PCA/PWM interrupt priority level is 3 (highest)

PS2H,PS2: UART2 interrupt priority control bit.

00: UART2 interrupt priority level is 0 (lowest)
01: UART2 interrupt priority level is 1 (lower)
10: UART2 interrupt priority level is 2 (higher)
11: UART2 interrupt priority level is 3 (highest)

PSPIH,PSPI: SPI interrupt priority control bit.

00: SPI interrupt priority level is 0 (lowest)
01: SPI interrupt priority level is 1 (lower)
10: SPI interrupt priority level is 2 (higher)
11: SPI interrupt priority level is 3 (highest)

PX4H,PX4: External interrupt 4 interrupt priority control bit.

00: INT4 interrupt priority level is 0 (lowest)
01: INT4 interrupt priority level is 1 (lower)
10: INT4 interrupt priority level is 2 (higher)
11: INT4 interrupt priority level is 3 (highest)

PCMPH,PCMP: Comparator interrupt priority control bit.

00: CMP interrupt priority level is 0 (lowest)
01: CMP interrupt priority level is 1 (lower)
10: CMP interrupt priority level is 2 (higher)
11: CMP interrupt priority level is 3 (highest)

PI2CH,PI2C: I2C interrupt priority control bit.

00: I2C interrupt priority level is 0 (lowest)
01: I2C interrupt priority level is 1 (lower)
10: I2C interrupt priority level is 2 (higher)
11: I2C interrupt priority level is 3 (highest)

PPWMH,PPWM: Advanced PWMA interrupt priority control bit.

00: Advanced PWM interrupt priority level is 0 (lowest)
01: Advanced PWM interrupt priority level is 1 (lower)
10: Advanced PWM interrupt priority level is 2 (higher)
11: Advanced PWM interrupt priority level is 3 (highest)

PPWMFDH,PPWMFD: Advanced PWM abnormal detection interrupt priority control bit.

00: PWMFD interrupt priority level is 0 (lowest)
01: PWMFD interrupt priority level is 1 (lower)
10: PWMFD interrupt priority level is 2 (higher)
11: PWMFD interrupt priority level is 3 (highest)

PS3H,PS3: UART3 interrupt priority control bit.

00: UART3 interrupt priority level is 0 (lowest)
01: UART3 interrupt priority level is 1 (lower)
10: UART3 interrupt priority level is 2 (higher)
11: UART3 interrupt priority level is 3 (highest)

PS4H,PS4: UART4 interrupt priority control bit.

00: UART4 interrupt priority level is 0 (lowest)
01: UART4 interrupt priority level is 1 (lower)

10: UART4 interrupt priority level is 2 (higher)
11: UART4 interrupt priority level is 3 (highest)

LCM Interface Configuration Register

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

LCMIFIP[1:0]: LCM interface interrupt priority control bits

00: LCM interface interrupt priority level is 0 (lowest level)
01: LCM interface interrupt priority is level 1 (lower level)
10: LCM interface interrupt priority is level 2 (higher level)
11: LCM interface interrupt priority level is 3 (the highest level)

Port Interrupt Priority Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PINIPL	FD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP
PINIPH	FD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH

P0IPH,P0IP: P0 interrupt priority control bits

00: P0 interrupt priority level is 0 (lowest level)
01: P0 interrupt priority is level 1 (lower level)
10: P0 interrupt priority is level 2 (higher level)
11: P0 interrupt priority level is 3 (the highest level)

P1IPH,P1IP: P1 interrupt priority control bits

00: P1 interrupt priority level is 0 (lowest level)
01: P1 interrupt priority is level 1 (lower level)
10: P1 interrupt priority is level 2 (higher level)
11: P1 interrupt priority level is 3 (the highest level)

P2IPH,P2IP: P2 interrupt priority control bits

00: P2 interrupt priority level is 0 (lowest level)
01: P2 interrupt priority is level 1 (lower level)
10: P2 interrupt priority is level 2 (higher level)
11: P2 interrupt priority level is 3 (the highest level)

P3IPH,P3IP: P3 interrupt priority control bits

00: P3 interrupt priority level is 0 (lowest level)
01: P3 interrupt priority is level 1 (lower level)
10: P3 interrupt priority is level 2 (higher level)
11: P3 interrupt priority level is 3 (the highest level)

P4IPH,P4IP: P4 interrupt priority control bits

00: P4 interrupt priority level is 0 (lowest level)
01: P4 interrupt priority is level 1 (lower level)
10: P4 interrupt priority is level 2 (higher level)
11: P4 interrupt priority level is 3 (the highest level)

P5IPH,P5IP: P5 interrupt priority control bits

00: P5 interrupt priority level is 0 (lowest level)
01: P5 interrupt priority is level 1 (lower level)
10: P5 interrupt priority is level 2 (higher level)
11: P5 interrupt priority level is 3 (the highest level)

P6IPH,P6IP: P6 interrupt priority control bits

00: P6 interrupt priority level is 0 (lowest level)
01: P6 interrupt priority is level 1 (lower level)
10: P6 interrupt priority is level 2 (higher level)
11: P6 interrupt priority level is 3 (the highest level)

P7IPH,P7IP: P7 interrupt priority control bits

00: P7 interrupt priority level is 0 (lowest level)
01: P7 interrupt priority is level 1 (lower level)
10: P7 interrupt priority is level 2 (higher level)
11: P7 interrupt priority level is 3 (the highest level)

DMA Interrupt Priority Control Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	FA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	
DMA_ADC_CFG	FA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]	
DMA_SPI_CFG	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	
DMA_UR1T_CFG	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	
DMA_UR1R_CFG	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	
DMA_UR2T_CFG	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	
DMA_UR2R_CFG	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	
DMA_UR3T_CFG	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	
DMA_UR3R_CFG	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	
DMA_UR3R_CFG	FA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	
DMA_UR4R_CFG	FA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	
DMA_LCM_CFG	FA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]	

M2MIP: DMA_M2M(Memory-to Memory DMA)interrupt priority control bits

00: DMA_M2M interrupt priority level is 0 (lowest level)
01: DMA_M2M interrupt priority is level 1 (lower level)
10: DMA_M2M interrupt priority is level 2 (higher level)
11: DMA_M2M interrupt priority level is 3 (the highest level)

ADCIP: DMA_ADC(ADC DMA)interrupt priority control bits

00: DMA_ADC interrupt priority level is 0 (lowest level)
01: DMA_ADC interrupt priority is level 1 (lower level)
10: DMA_ADC interrupt priority is level 2 (higher level)
11: DMA_ADC interrupt priority level is 3 (the highest level)

SPIIP: DMA_SPI(SPI DMA)interrupt priority control bits

00: DMA_SPI interrupt priority level is 0 (lowest level)
01: DMA_SPI interrupt priority is level 1 (lower level)
10: DMA_SPI interrupt priority is level 2 (higher level)
11: DMA_SPI interrupt priority level is 3 (the highest level)

UR1TIP: DMA_UR1T(UART 1 send DMA)interrupt priority control bits

00: DMA_UR1T interrupt priority level is 0 (lowest level)
01: DMA_UR1T interrupt priority is level 1 (lower level)
10: DMA_UR1T interrupt priority is level 2 (higher level)
11: DMA_UR1T interrupt priority level is 3 (the highest level)

UR1RIP: DMA_UR1R(UART 1 receive DMA)interrupt priority control bits

00: DMA_UR1R interrupt priority level is 0 (lowest level)
01: DMA_UR1R interrupt priority is level 1 (lower level)
10: DMA_UR1R interrupt priority is level 2 (higher level)
11: DMA_UR1R interrupt priority level is 3 (the highest level)

UR2TIP: DMA_UR2T(UART 2 send DMA)interrupt priority control bits

00: DMA_UR2T interrupt priority level is 0 (lowest level)
01: DMA_UR2T interrupt priority is level 1 (lower level)
10: DMA_UR2T interrupt priority is level 2 (higher level)
11: DMA_UR2T interrupt priority level is 3 (the highest level)

UR2RIP: DMA_UR2R(UART 2 receive DMA)interrupt priority control bits

00: DMA_UR2R interrupt priority level is 0 (lowest level)
01: DMA_UR2R interrupt priority is level 1 (lower level)
10: DMA_UR2R interrupt priority is level 2 (higher level)
11: DMA_UR2R interrupt priority level is 3 (the highest level)

UR3TIP: DMA_UR3T(UART 3 send DMA)interrupt priority control bits

00: DMA_UR3T interrupt priority level is 0 (lowest level)
01: DMA_UR3T interrupt priority is level 1 (lower level)
10: DMA_UR3T interrupt priority is level 2 (higher level)
11: DMA_UR3T interrupt priority level is 3 (the highest level)

UR3RIP: DMA_UR3R(UART 3 receive DMA)interrupt priority control bits

00: DMA_UR3R interrupt priority level is 0 (lowest level)
01: DMA_UR3R interrupt priority is level 1 (lower level)
10: DMA_UR3R interrupt priority is level 2 (higher level)
11: DMA_UR3R interrupt priority level is 3 (the highest level)

UR4TIP: DMA_UR4T(UART 4 send DMA)interrupt priority control bits

00: DMA_UR3R interrupt priority level is 0 (lowest level)

01: DMA_UR3R interrupt priority is level 1 (lower level)
10: DMA_UR3R interrupt priority is level 2 (higher level)
11: DMA_UR3R interrupt priority level is 3 (the highest level)

UR4RIP: DMA_UR4R(UART 4 receive DMA)interrupt priority control bits

00: DMA_UR4R interrupt priority level is 0 (lowest level)
01: DMA_UR4R interrupt priority is level 1 (lower level)
10: DMA_UR4R interrupt priority is level 2 (higher level)
11: DMA_UR4R interrupt priority level is 3 (the highest level)

LCMIP: DMA_LCM(LCM interfaceDMA)interrupt priority control bits

00: DMA_LCM interrupt priority level is 0 (lowest level)
01: DMA_LCM interrupt priority is level 1 (lower level)
10: DMA_LCM interrupt priority is level 2 (higher level)
11: DMA_LCM interrupt priority level is 3 (the highest level)

11.5 Example Routines

11.5.1 INT0 Interrupt (Rising and Falling Edges)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sbit        P10         =   P1^0;
sbit        P11         =   P1^1;

void INT0_Isr() interrupt 0
{
    if (INT0)                                   //Judging 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 edge and falling edge interrupts
    EX0 = 1;                                    //Enable INT0 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            JB      INT0,RISING         ;Judging 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 edge and falling edge interrupts
            SETB    EX0                 ;Enable INT0 interrupt
            SETB    EA
            JMP     $

                                        END
```

11.5.2 INT0 Interrupt (Falling Edge)

C language code

```
//Operating frequency for test is 11.0592MHz
```

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

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

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;                                    //Enable INT0 falling edge interrupt
    EX0 = 1;                                    //Enable INT0 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

```
            ORG     0000H
            LJMP    MAIN
            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 INT0 falling edge interrupt
            SETB    EX0                 ;Enable INT0 interrupt
            SETB    EA
            JMP     $

            END
```

11.5.3 INT1 Interrupt (Rising and Falling Edges)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void INT1_Isr() interrupt 2
{
```

```
    if (INT1)                                   //Judging 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 edge and falling edge interrupts
    EX1 = 1;                                    //Enable INT1 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     0013H
            LJMP    INT1ISR

            ORG     0100H
INT1ISR:
            JB      INT1,RISING         ;Judging 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 edge and falling edge interrupts
        SETB    EX1             ;Enable INT1 interrupt
        SETB    EA
        JMP     $

        END
```

11.5.4 INT1 Interrupt (Falling Edge)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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 INT1 falling edge interrupt
    EX1 = 1;                                    //Enable INT1 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     0013H
            LJMP    INT11SR

            ORG     0100H
INT11SR:
            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 INT1 falling edge interrupt
            SETB    EX1                     ;Enable INT1 interrupt
            SETB    EA
            JMP     $
```

```
END
```

11.5.5 INT2 Interrupt (Falling Edge only)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr         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

```
;Operating frequency for test is 11.0592MHz
```

```
INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

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     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)

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr         P0M1        =   0x93;
```

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

sfr     INTCLKO =   0x8f;
#define EX2         0x10
#define EX3         0x20
#define EX4         0x40
sbit    P10     =   P1^0;

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

```
;Operating frequency for test is 11.0592MHz

INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

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         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)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr         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

```
;Operating frequency for test is 11.0592MHz

INTCLKO     DATA    8FH
EX2         EQU     10H
EX3         EQU     20H
EX4         EQU     40H

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     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
```

11.5.8 Timer0 Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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 timer
    ET0 = 1;                                //Enable timer0 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            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 timer
        SETB    ET0             ;Enable timer0 interrupt
        SETB    EA

        JMP     $

        END
```

11.5.9 Timer1 Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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 timer
    ET1 = 1;                                    //Enable timer1 interrupt
    EA = 1;

    while (1);
```

```
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            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 timer
            SETB    ET1                 ;Enable timer1 interrupt
            SETB    EA

            JMP     $

            END
```

11.5.10 Timer2 Interrupt

C language code

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

sbit        P10             =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //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 timer
    IE2 = ET2;                                  //Enable timer2 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L             DATA            0D7H
```

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

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

        ORG     0000H
        LJMP    MAIN
        ORG     0063H
        LJMP    TM2ISR

        ORG     0100H
TM2ISR:
        CPL     P1.0            ;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 timer
        MOV     IE2,#ET2        ;Enable timer2 interrupt
        SETB    EA

        JMP     $

        END
```

11.5.11 Timer3 Interrupt

C language code

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

sbit        P10             =   P1^0;

void 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 timer
    IE2 = ET3;                          //Enable timer3 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T3L             DATA            0D5H
```

```
T3H         DATA    0D4H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET3         EQU     20H
AUXINTIF    DATA    0EFH
T3IF        EQU     02H

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     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 timer
            MOV     IE2,#ET3        ;Enable timer3 interrupt
            SETB    EA

            JMP     $

            END
```

11.5.12 Timer4 Interrupt

C language code

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

sbit        P10             =   P1^0;

void 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 timer
    IE2 = ET4;                          //Enable timer4 interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test 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

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     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 timer
            MOV     IE2,#ET4        ;Enable timer4 interrupt
            SETB    EA

            JMP     $
```

```
END
```

11.5.13 UART1 Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sbit        P10         =   P1^0;
sbit        P11         =   P1^1;

void UART1_Isr() interrupt 4
{
    if (TI)
    {
        TI = 0;                                 //Clear interrupt flag
        P10 = !P10;                             //Test port
    }
    if (RI)
    {
        RI = 0;                                 //Clear interrupt flag
        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 timer
    ES = 1;                             //Enable UART1 interrupt
    EA = 1;
    SBUF = 0x5a;                        // Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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     0100H
UART1ISR:
            JNB     TI,CHECKRI
            CLR     TI              ;Clear interrupt flag
            CPL     P1.0            ;Test port
CHECKRI:
            JNB     RI,ISREXIT
            CLR     RI              ;Clear interrupt flag
            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 timer
        SETB    ES              ;Enable UART1 interrupt
        SETB    EA
        MOV     SBUF,#5AH       ; Send test data

        JMP     $

        END
```

11.5.14 UART2 Interrupt

C language code

```
//Operating frequency for test 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     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    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART2_Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;                 //Clear interrupt flag
        P12 = !P12;                     //Test port
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;                 //Clear interrupt flag
        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 timer
    IE2 = ES2;                      //Enable UART2 interrupt
    EA = 1;
    S2BUF = 0x5a;                   // Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH
ES2         EQU     01H

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

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2ISR

            ORG     0100H
UART2ISR:
            PUSH    ACC
```

```
            PUSH    PSW
            MOV     A,S2CON
            JNB     ACC.1,CHECKRI
            ANL     S2CON,#NOT 02H      ;Clear interrupt flag
            CPL     P1.2                ;Test port
CHECKRI:
            MOV     A,S2CON
            JNB     ACC.0,ISREXIT
            ANL     S2CON,#NOT 01H      ;Clear interrupt flag
            CPL     P1.3                ;Test port
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 timer
            MOV     IE2,#ES2            ;Enable UART2 interrupt
            SETB    EA
            MOV     S2BUF,#5AH          ; Send test data

            JMP     $

            END
```

11.5.15 UART3 Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;
#define ES3         0x08

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    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART3_Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;                 //Clear interrupt flag
        P12 = !P12;                     //Test port
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;                 //Clear interrupt flag
        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;

    S3CON = 0x10;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                        //Start timer
    IE2 = ES3;                          //Enable UART3 interrupt
    EA = 1;
    S3BUF = 0x5a;                       //Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
```

```
AUXR        DATA    8EH
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH
ES3         EQU     08H

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    UART3ISR

            ORG     0100H
UART3ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S3CON
            JNB     ACC.1,CHECKRI
            ANL     S3CON,#NOT 02H      ;Clear interrupt flag
            CPL     P1.2                ;Test port
CHECKRI:
            MOV     A,S3CON
            JNB     ACC.0,ISREXIT
            ANL     S3CON,#NOT 01H      ;Clear interrupt flag
            CPL     P1.3                ;Test port
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     S3CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H           ;Start timer
```

```
        MOV     IE2,#ES3                ;Enable UART3 interrupt
        SETB    EA
        MOV     S3BUF,#5AH              ;Send test data

        JMP     $

        END
```

11.5.16 UART4 Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S4CON   =   0x84;
sfr     S4BUF   =   0x85;
sfr     IE2     =   0xaf;
#define ES4         0x10

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    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART4_Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;                 //Clear interrupt flag
        P12 = !P12;                     //Test port
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;                 //Clear interrupt flag
        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;

    S4CON = 0x10;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                        //Start timer
    IE2 = ES4;                          //Enable UART4 interrupt
    EA = 1;
    S4BUF = 0x5a;                       //Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
S4CON       DATA    84H
S4BUF       DATA    85H
IE2         DATA    0AFH
ES4         EQU     10H

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    UART4ISR

            ORG     0100H
UART4ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S4CON
            JNB     ACC.1,CHECKRI
            ANL     S4CON,#NOT 02H      ;Clear interrupt flag
            CPL     P1.2                ;Test port
CHECKRI:
            MOV     A,S4CON
```

```
                JNB     ACC.0,ISREXIT
                ANL     S4CON,#NOT 01H      ;Clear interrupt flag
                CPL     P1.3                ;Test port
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     S4CON,#10H
                MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#14H           ;Start timer
                MOV     IE2,#ES4            ;Enable UART4 interrupt
                SETB    EA
                MOV     S4BUF,#5AH          ;Send test data

                JMP     $

                END
```

11.5.17 LVD Interrupt

C language code

```
//Operating frequency for test 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     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
sbit    P10     =   P1^0;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;                  //Clear interrupt flag
    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;                  //Interrupt flag needs to be cleared at power-on
    RSTCFG = LVD3V0;                //Set the LVD voltage to 3.0V
    ELVD = 1;                       //Enable LVD interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test 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

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     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF      ;Clear interrupt flag
            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      ;Interrupt flag needs to be cleared at power-on
            MOV     RSTCFG,# LVD3V0     ;Set the LVD voltage to 3.0V
            SETB    ELVD                ;Enable LVD interrupt
            SETB    EA
            JMP     $

            END
```

11.5.18 SPI Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

```
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //Clear interrupt flag
    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;                   //Enable SPI master mode
    SPSTAT = 0xc0;                  //Clear interrupt flag
    IE2 = ESPI;                     //Enable SPI interrupt
    EA = 1;
    SPDAT = 0x5a;                   //Send test data

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
```

```
                LJMP    SPIISR

                ORG     0100H
SPIISR:
                MOV     SPSTAT,#0C0H        ;Clear interrupt flag
                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          ;Enable SPI master mode
                MOV     SPSTAT,#0C0H        ;Clear interrupt flag
                MOV     IE2,#ESPI           ;Enable SPI interrupt
                SETB    EA
                MOV     SPDAT,#5AH          ;Send test data

                JMP     $

                END
```

11.5.19 Comparator Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

sbit        P10 = P1^0;

void CMP_Isr() interrupt 21
{
            CMPCR1 &= ~0x40;                //clear interrupt flag
```

```
        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  is CMP+ input pin
        CMPCR1 |= 0x04;         //P3.7 is CMP- input pin
        CMPCR1 |= 0x02;         // Enable comparator output
        EA = 1;

        while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CMPCR1      DATA 0E6H
CMPCR2      DATA 0E7H

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

        ORG         0000H
        LJMP        MAIN
        ORG         00ABH
        LJMP        CMPISR

        ORG         0100H
CMPISR:
        ANL     CMPCR1,#NOT 40H     ; clear interrupt flag
        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 comparator module
        ORL     CMPCR1,#30H         ; Enable comparator edge interrupt
        ANL     CMPCR1,#NOT 08H     ;P3.6  is CMP+ input pin
        ORL     CMPCR1,#04H         ; P3.7 is CMP- input pin
        ORL     CMPCR1,#02H         ; Enable comparator output
        SETB    EA

        JMP     $

    END
```

11.5.21 I2C Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr     P_SW2       =   0xba;

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

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;
sbit    P10         =   P1^0;

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

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
```

```
        LJMP    I2CISR

        ORG     0100H
I2CISR:
        PUSH    ACC
        PUSH    DPL
        PUSH    DPH
        PUSH    P_SW2
        MOV     P_SW2,#80H
        MOV     DPTR,#I2CMSST
        MOVX    A,@DPTR
        ANL     A,#NOT 40H          ;Clear interrupt flag
        MOVX    @DPTR,A
        CPL     P1.0                ;Test port
        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 master 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

The STC8A8K64D4 series MCUs support all I/O interrupts, and support 4 interrupt modes: falling edge interrupt, rising edge interrupt, low-level interrupt, and high-level interrupt. Every group of I/O ports has an independent interrupt entry address, and each I/O can independently set the interrupt mode.

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 interrupt enable register	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 interrupt enable register	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 interrupt enable register	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 interrupt enable register	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 interrupt enable register	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 interrupt enable register	FD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	xx00,0000
P6INTE	P6 interrupt enable register	FD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE	0000,0000
P7INTE	P7 interrupt enable register	FD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE	0000,0000
P0INTF	P0 interrupt flag register	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 interrupt flag register	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 interrupt flag register	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 interrupt flag register	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P4INTF	P4 interrupt flag register	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 interrupt flag register	FD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	xx00,0000
P6INTF	P6 interrupt flag register	FD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF	0000,0000
P7INTF	P7 interrupt flag register	FD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF	0000,0000
P0IM0	P0 Interrupt mode register 0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 Interrupt mode register 0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 Interrupt mode register 0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 Interrupt mode register 0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P4IM0	P4 Interrupt mode register 0	FD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0	0000,0000
P5IM0	P5 Interrupt mode register 0	FD25H	-	-	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	xx00,0000
P6IM0	P6 Interrupt mode register 0	FD26H	P67IM0	P66IM0	P65IM0	P64IM0	P63IM0	P62IM0	P61IM0	P60IM0	0000,0000
P7IM0	P7 Interrupt mode register 0	FD27H	P77IM0	P76IM0	P75IM0	P74IM0	P73IM0	P72IM0	P71IM0	P70IM0	0000,0000
P0IM1	P0 Interrupt mode register 1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1 Interrupt mode register 1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 Interrupt mode register 1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 Interrupt mode register 1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P4IM1	P4 Interrupt mode register 1	FD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1	0000,0000
P5IM1	P5 Interrupt mode register 1	FD35H	-	-	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	xx00,0000
P6IM1	P6 Interrupt mode register 1	FD36H	P67IM1	P66IM1	P65IM1	P64IM1	P63IM1	P62IM1	P61IM1	P60IM1	0000,0000

P7IM1	P7 Interrupt mode register 1	FD37H	P77IM1	P76IM1	P75IM1	P74IM1	P73IM1	P72IM1	P71IM1	P70IM1	0000,0000
PINIPL	I/O port interrupt priority low register	FD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O port interrupt priority high register	FD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
P0WKUE	P0 interrupt wake-up enable register	FD40H	P07WKUE	P06WKUE	P05WKUE	P04WKUE	P03WKUE	P02WKUE	P01WKUE	P00WKUE	0000,0000
P1WKUE	P1 interrupt wake-up enable register	FD41H	P17WKUE	P16WKUE	P15WKUE	P14WKUE	P13WKUE	P12WKUE	P11WKUE	P10WKUE	0000,0000
P2WKUE	P2 interrupt wake-up enable register	FD42H	P27WKUE	P26WKUE	P25WKUE	P24WKUE	P23WKUE	P22WKUE	P21WKUE	P20WKUE	0000,0000
P3WKUE	P3 interrupt wake-up enable register	FD43H	P37WKUE	P36WKUE	P35WKUE	P34WKUE	P33WKUE	P32WKUE	P31WKUE	P30WKUE	0000,0000
P4WKUE	P4 interrupt wake-up enable register	FD44H	P47WKUE	P46WKUE	P45WKUE	P44WKUE	P43WKUE	P42WKUE	P41WKUE	P40WKUE	0000,0000
P5WKUE	P5 interrupt wake-up enable register	FD45H	-	-	P55WKUE	P54WKUE	P53WKUE	P52WKUE	P51WKUE	P50WKUE	xx00,0000
P6WKUE	P6 interrupt wake-up enable register	FD46H	P67WKUE	P66WKUE	P65WKUE	P64WKUE	P63WKUE	P62WKUE	P61WKUE	P60WKUE	0000,0000
P7WKUE	P7 interrupt wake-up enable register	FD47H	P77WKUE	P76WKUE	P75WKUE	P74WKUE	P73WKUE	P72WKUE	P71WKUE	P70WKUE	0000,0000

12.1.1 Port interrupt enable registers (PxINTE)

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
P4INTE	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	FD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE
P6INTE	FD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE
P7INTE	FD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE

PnINTE.x: Port interrupt enable control bit (n=0~7, x=0~7)

0: disable Pn.x port interrupt function

1: Enable Pn.x port interrupt function

12.1.2 Port interrupt flag registers (PxINTF)

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
P4INTF	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF
P5INTF	FD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF
P6INTF	FD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF
P7INTF	FD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF

PnINTF.x: Port interrupt request flag (n=0~7, x=0~7)

0: No interrupt request for Pn.x port

1: Pn.x port has an interrupt request, if the interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software.

12.1.3 Port interrupt mode configuration registers (PxIM0，PxIM1)

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
P4IM0	FD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0
P4IM1	FD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1
P5IM0	FD25H	-	-	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0
P5IM1	FD35H	-	-	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1
P6IM0	FD26H	P67IM0	P66IM0	P65IM0	P64IM0	P63IM0	P62IM0	P61IM0	P60IM0
P6IM1	FD36H	P67IM1	P66IM1	P65IM1	P64IM1	P63IM1	P62IM1	P61IM1	P60IM1
P7IM0	FD27H	P77IM0	P76IM0	P75IM0	P74IM0	P73IM0	P72IM0	P71IM0	P70IM0
P7IM1	FD37H	P77IM1	P76IM1	P75IM1	P74IM1	P73IM1	P72IM1	P71IM1	P70IM1

Configure port mode

PnIM1.x	PnIM0.x	Pn.x Interrupt mode
0	0	Falling edge interrupt
0	1	Rising edge interrupt
1	0	Low level interrupt
1	1	High level interrupt

12.1.4 Port interrupt priority control registers (PINIPL, PINIPH)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PINIPL	FD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP
PINIPH	FD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH

PxIPH,PxIP: Px interrupt priority control bit

00: Px interrupt priority level is 0 (lowest)
01: Px interrupt priority level is 1 (lower)
10: Px interrupt priority level is 2 (higher)
11: Px interrupt priority level is 3 (highest)

12.1.5 Port interrupt power-down wake-up enable registers (PxWKUE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P0WKUE	FD40H	P07WKUE	P06WKUE	P05WKUE	P04WKUE	P03WKUE	P02WKUE	P01WKUE	P00WKUE
P1WKUE	FD41H	P17WKUE	P16WKUE	P15WKUE	P14WKUE	P13WKUE	P12WKUE	P11WKUE	P10WKUE
P2WKUE	FD42H	P27WKUE	P26WKUE	P25WKUE	P24WKUE	P23WKUE	P22WKUE	P21WKUE	P20WKUE
P3WKUE	FD43H	P37WKUE	P36WKUE	P35WKUE	P34WKUE	P33WKUE	P32WKUE	P31WKUE	P30WKUE
P4WKUE	FD44H	P47WKUE	P46WKUE	P45WKUE	P44WKUE	P43WKUE	P42WKUE	P41WKUE	P40WKUE
P5WKUE	FD45H	-	-	P55WKUE	P54WKUE	P53WKUE	P52WKUE	P51WKUE	P50WKUE
P6WKUE	FD46H	P67WKUE	P66WKUE	P65WKUE	P64WKUE	P63WKUE	P62WKUE	P61WKUE	P60WKUE
P7WKUE	FD47H	P77WKUE	P76WKUE	P75WKUE	P74WKUE	P73WKUE	P72WKUE	P71WKUE	P70WKUE

PnxWKUE: Port interrupt power-down wake-up enable bit(n=0~7, x=0~7)

0: disable Pn.x interrupt power-down wake-up function

1: enable Pn.x interrupt power-down wake-up function

12.2 Example Routines

12.2.1 P0 Falling edge interrupt

C language code

```
// Operating frequency for test 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);
}
//Because the interrupt vector is greater than 31, it cannot be directly compiled in KEIL
//The 13th interrupt entry address must be borrowed
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 interrupt
                }
                if (intf & 0x02)
                {
                                                                //P0.1 interrupt
                }
                if (intf & 0x04)
                {
                                                                //P0.2 interrupt
                }
                if (intf & 0x08)
                {
                                                                //P0.3 interrupt
                }
                if (intf & 0x10)
                {
                                                                //P0.4 interrupt
                }
                if (intf & 0x20)
                {
                                                                //P0.5 interrupt
                }
                if (intf & 0x40)
                {
                                                                //P0.6 interrupt
                }
                if (intf & 0x80)
                {
                                                                //P0.7 interrupt
                }
        }
        P_SW2 = psw2_st;
}

// ISR.ASM
// Save the following code as ISP.ASM, and then add the file to the project

                CSEG            AT 012BH                        ;P0 interrupt entry address
                JMP             P0INT_ISR
```

```
P0INT_ISR:
        JMP     006BH                   ; Borrow the entry address of the 13th interrupt
        END
```

Assembly code

```
;Operating frequency for test 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 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 interrupt
CHECKP01:
        JNB     ACC.1,CHECKP02
        NOP                             ;P0.1 interrupt
CHECKP02:
```

```
            JNB     ACC.2,CHECKP03
            NOP                             ;P0.2 interrupt
CHECKP03
            JNB     ACC.3,CHECKP04
            NOP                             ;P0.3 interrupt
CHECKP04:
            JNB     ACC.4,CHECKP05
            NOP                             ;P0.4 interrupt
CHECKP05:
            JNB     ACC.5,CHECKP06
            NOP                             ;P0.5 interrupt
CHECKP06:
            JNB     ACC.6,CHECKP07
            NOP                             ;P0.6 interrupt
CHECKP07:
            JNB     ACC.7,P0ISREXIT
            NOP                             ;P0.7 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 interrupt
            ANL     P_SW2,#7FH

            SETB    EA

            JMP     $

            END
```

12.2.2 P1 rising edge interrupt

C language code

```
;Operating frequency for test 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 interrupt
    P_SW2 &= ~0x80;

    EA = 1;
```

```
    while (1);
}

//Because the interrupt vector is greater than 31, it cannot be directly compiled in KEIL
//The 13th interrupt entry address must be borrowed
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 interrupt
        }
        if (intf & 0x02)
        {
                                                    //P1.1 interrupt
        }
        if (intf & 0x04)
        {
                                                    //P1.2 interrupt
        }
        if (intf & 0x08)
        {
                                                    //P1.3 interrupt
        }
        if (intf & 0x10)
        {
                                                    //P1.4 interrupt
        }
        if (intf & 0x20)
        {
                                                    //P1.5 interrupt
        }
        if (intf & 0x40)
        {
                                                    //P1.6 interrupt
        }
        if (intf & 0x80)
        {
                                                    //P1.7 interrupt
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
// Save the following code as ISP.ASM, and then add the file to the project

            CSEG        AT 0133H                    ;P1 interrupt entry address
            JMP         P1INT_ISR
```

```
P1INT_ISR:
        JMP     006BH                   ; Borrow the entry address of the 13th interrupt
        END
```

Assembly code

```
;Operating frequency for test 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

P1INTE  XDATA   0FD01H
P1INTF  XDATA   0FD11H
P1IM0   XDATA   0FD21H
P1IM1   XDATA   0FD31H

        ORG     0000H
        LJMP    MAIN

        ORG     0133H               ;P1 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 interrupt
CHECKP11:
        JNB     ACC.1,CHECKP12
        NOP                         ;P1.1 interrupt
CHECKP12:
```

```
            JNB     ACC.2,CHECKP13
            NOP                             ;P1.2 interrupt
CHECKP13
            JNB     ACC.3,CHECKP14
            NOP                             ;P1.3 interrupt
CHECKP14:
            JNB     ACC.4,CHECKP15
            NOP                             ;P1.4 interrupt
CHECKP15:
            JNB     ACC.5,CHECKP16
            NOP                             ;P1.5 interrupt
CHECKP16:
            JNB     ACC.6,CHECKP17
            NOP                             ;P1.6 interrupt
CHECKP17:
            JNB     ACC.7,P1ISREXIT
            NOP                             ;P1.7 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 interrupt
            ANL     P_SW2,#7FH

            SETB    EA

            JMP     $

            END
```

12.2.3 P2 low level interrupt

C language code

```
;Operating frequency for test 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;                                   // low level interrupt
    P2IM1 = 0xff;
    P2INTE = 0xff;                                  // Enable P2 port interrupt
    P_SW2 &= ~0x80;

    EA = 1;

    while (1);
```

```
}

//Because the interrupt vector is greater than 31, it cannot be directly compiled in KEIL
//The 13th interrupt entry address must be borrowed
void common_isr() interrupt 13
{
    unsigned char psw2_st;
    unsigned char intf;

    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P2INTF;
    if (intf)
    {
        P2INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P2.0 interrupt
        }
        if (intf & 0x02)
        {
                                                    //P2.1 interrupt
        }
        if (intf & 0x04)
        {
                                                    //P2.2 interrupt
        }
        if (intf & 0x08)
        {
                                                    //P0.3 interrupt
        }
        if (intf & 0x10)
        {
                                                    //P2.4 interrupt
        }
        if (intf & 0x20)
        {
                                                    //P2.5 interrupt
        }
        if (intf & 0x40)
        {
                                                    //P2.6 interrupt
        }
        if (intf & 0x80)
        {
                                                    //P2.7 interrupt
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
// Save the following code as ISP.ASM, and then add the file to the project

            CSEG        AT 013BH                    ;P2 interrupt entry address
            JMP         P2INT_ISR
P2INT_ISR:
```

```
            JMP     006BH                   ; Borrow the entry address of the 13th interrupt
            END
```

Assembly code

```
;Operating frequency for test 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  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 interrupt
CHECKP21:
            JNB     ACC.1,CHECKP22
            NOP                                 ;P2.1 interrupt
CHECKP22:
            JNB     ACC.2,CHECKP23
            NOP                                 ;P2.2 interrupt
CHECKP23
            JNB     ACC.3,CHECKP24
            NOP                                 ;P2.3 interrupt
CHECKP24:
            JNB     ACC.4,CHECKP25
            NOP                                 ;P2.4 interrupt
CHECKP25:
            JNB     ACC.5,CHECKP26
            NOP                                 ;P2.5 interrupt
```

```
CHECKP26:
        JNB     ACC.6,CHECKP27
        NOP                             ;P2.6 interrupt
CHECKP27:
        JNB     ACC.7,P2ISREXIT
        NOP                             ;P2.7  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  interrupt
        ANL     P_SW2,#7FH

        SETB    EA

        JMP     $

        END
```

12.2.4 Port3 high level interrupt

C language code

```
//Operating frequency for test 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;                                   // high level interrupt
        P3IM1 = 0xff;
        P3INTE = 0xff;                                  //Enable P3 interrupt
        P_SW2 &= ~0x80;

        EA = 1;
        while (1);

}

//Because the interrupt vector is greater than 31, it cannot be directly compiled in KEIL
//The 13th interrupt entry address must be borrowed
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)
        {
                                                        //P3.0 interrupt
        }
        if (intf & 0x02)
        {
                                                        //P3.1 interrupt
        }
        if (intf & 0x04)
        {
```

```
                                                        //P3.2 interrupt
        }
        if (intf & 0x08)
        {
                                                        //P3.3 interrupt
        }
        if (intf & 0x10)
        {
                                                        //P3.4 interrupt
        }
        if (intf & 0x20)
        {
                                                        //P3.5 interrupt
        }
        if (intf & 0x40)
        {
                                                        //P3.6 interrupt
        }
        if (intf & 0x80)
        {
                                                        //P3.7 interrupt
        }
    }
    P_SW2 = psw2_st;
}
// ISR.ASM
//Save the following code as ISP.ASM, and then add the file to the project
            CSEG    AT 0143H        ; P3 port interrupt entry address
            JMP     P3INT_ISR
P3INT_ISR:
            JMP     006BH           ; Borrow the entry address of the 13th interrupt
            END
```

Assembly code

```
;Operating frequency for test 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 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 interrupt
CHECKP31:
            JNB     ACC.1,CHECKP32
            NOP                             ;P3.1 interrupt
CHECKP32:
            JNB     ACC.2,CHECKP33
            NOP                             ;P3.2 interrupt
CHECKP33
            JNB     ACC.3,CHECKP34
            NOP                             ;P3.3 interrupt
CHECKP34:
            JNB     ACC.4,CHECKP35
            NOP                             ;P3.4 interrupt
CHECKP35:
            JNB     ACC.5,CHECKP36
            NOP                             ;P3.5 interrupt
CHECKP36:
            JNB     ACC.6,CHECKP37
            NOP                             ;P3.6 interrupt
CHECKP37:
            JNB     ACC.7,P3ISREXIT
            NOP                             ;P3.7 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 interrupt
ANL     P_SW2,#7FH

SETB    EA

JMP     $

END
```

13 Timer/Counter

Five 16-bit Timers/Counters are integrated in STC8A8K64D4 series of microcontrollers: T0, T1, T2, T3 and T4. All of them can be used as timer or counter. For T0 and T1, the ‘timer’ or ‘counter’ function is selected by the control bits C/T in the special function register TMOD. For T2, the ‘timer’ or ‘counter’ function is selected by the control bit T2_C/T in the special function register AUXR. For T3, the ‘timer’ or ‘counter’ function is selected by the control bit T3_C/T in the special function register T4T3M. For T4, the ‘timer’ or ‘counter’ function is selected by the control bit T4_C/T in the special function register T4T3M. The core of the timer/counter is a up counter, the essence of which is counting pulses. The only difference of ‘timer’ mode and ‘counter’ mode is the different counting pulses sources. If the counting pulse is from the system clock, the timer/counter runs in the timing mode, it counts once every 12 clocks or one clock. If the counting pulse is from the microcontroller external pins, the timer/counter runs in counting mode, it counts once every pulse.

When T0, T1 and T2 are operating in ‘timer’ mode, T0x12, T1x12 and T2x12 in AUXR register are used to determine the clocks of T0, T1 and T2 are system clock/12 or system clock/1. When T3 and T4 are operating in ‘timer’ mode, T3x12 and T4x12 in the T4T3M register determine the clocks of T3 and T4 are system clock/12 or system clock/1. When the timer/counters are operating in ‘counter’ mode, the frequency of the external pulse is not divided.

T0 has four operating modes which are selected by bit-pairs (M1, M0) in TMOD. The four modes are mode 0 (16-bit auto-reload mode), mode 1 (16-bit non-auto-reload mode), mode 2 (8-bit auto-reload mode) and mode 3 (16-bit auto-reload mode whose interrupt can not be disabled). And for T1, all modes except mode 3 are the same as T0. The mode 3 of T1 is invalid and stops counting. For T2, T3 and T4, they only have one mode: 16-bit auto-reload mode. Besides being used as timer/counters, T2, T3 and T4 can also be as the baud-rate generators of UARTs and programmable clock outputs.

13.1 Registers Related to Timers

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	Timer 0 and 1 control register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	Timer 0 and 1 mode register	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	Timer 0 low byte register	8AH									0000,0000
TL1	Timer 1 low byte register	8BH									0000,0000
TH0	Timer 0 high byte register	8CH									0000,0000
TH1	Timer 1 high byte register	8DH									0000,0000
AUXR	Auxiliary register 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	External interrupt and clock output control register	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	Wake-up Timer Control Register low	AAH									1111,1111
WKTCH	Wake-up Timer Control Register high byte	ABH	WKTEN								0111,1111
T4T3M	Timer4 and Timer 3 Control Register	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	Timer 4 high byte register	D2H									0000,0000
T4L	Timer 4 low byte register	D3H									0000,0000
T3H	Timer 3 high byte register	D4H									0000,0000
T3L	Timer 3 low byte register	D5H									0000,0000
T2H	Timer 2 high byte register	D6H									0000,0000
T2L	Timer 2 low byte register	D7H									0000,0000

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
TM2PS	Timer 2 clock prescaler register	FEA2H									0000,0000
TM3PS	Timer 3 clock prescaler register	FEA3H									0000,0000
TM4PS	Timer 4 clock prescaler register	FEA4H									0000,0000

13.2 Timer 0/1

13.2.1 Timer 0 and 1 Control Register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1: T1 overflow flag. After T1 is enabled to count, it performs adding 1 count from the initial value. TF1 is set by hardware on T1 overflow and requests interrupt to CPU. It will keep the status until CPU responds the interrupt and is cleared by hardware automatically. It also can be cleared by software.

TR1: T1 run control bit. It is set or cleared by software to turn the timer on/off. If GATE (TMOD.7) = 0, T1 will start counting as soon as TR1=1 and stop counting when TR1=0. If GATE (TMOD.7) = 1, T1 is enabled to count only if TR1 = 1 and INT1 is high.

TF0: T0 overflow flag. After T0 is enabled to count, it performs adding 1 count from the initial value. TF0 is set by hardware on T0 overflow and requests interrupt to CPU. It will keep the status until CPU responds the interrupt and is cleared by hardware automatically. It also can be cleared by software.

TR0: T0 run control bit. It is set or cleared by software to turn the timer on/off. If GATE (TMOD.3) = 0, T0 will start counting as soon as TR0=1 and stop counting when TR0=0. If GATE (TMOD.0) = 1, T0 is enabled to count only if TR0 = 1 and INT0 is high.

IE1: External Interrupt 1 (INT1/P3.3) request flag. IE1=1 means external interrupt requests interrupt to CPU.It is cleared by hardware automatically when the CPU responds to the interrupt.

IT1: External Intenupt 1 trigger edge type control bit. If IT1 = 0, INT1 can be triggered by both rising and falling edges. If IT1 = 1, INT1 can be triggered only by falling edge.

IE0: External Interrupt 0 (INT0/P3.2) request flag. IE0=1 means external interrupt requests interrupt to CPU.It is cleared by hardware automatically when the CPU responds to the interrupt.

IT0: External Intenupt 0 trigger edge type control bit. If IT0 = 0, INT0 can be triggered by both rising and falling edges. If IT0 = 1, INT0 can be triggered only by falling edge.

13.2.2 Timer 0/1 Mode Register

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

T1_GATE: T1 gate control. If GATE/TMOD.7=1, T1 starts only when TR1 is set AND INT1 pin is high.

T0_GATE: T0 gate control. If GATE/TMOD.3 =1, T0 starts only when TR0 is set AND INT0 pin is high.

T1_C/T: T1 mode select bit. If it is reset, T1 is used as a timer (input pulse is from internal system clock). If it is set, T1 is used as a counter (input pulse is from external T1/P3.5 pin).

T0_C/T: T0 mode select bit. If it is reset, T0 is used as a timer (input pulse is from internal system clock). If it is set, T0 is used as a counter (input pulse is from external T0/P3.4 pin).

T1_M1/T1_M0: T1 mode select bits.

T1_M1	T1_M0	T1 operating mode
0	0	16-bit auto-reload mode. When the 16-bit counter [TH1, TL1] overflows, the system loads the reload value in the internal 16-bit reload register into [TH1, TL1] automatically.
0	1	16-bit non-auto-reload mode. When the 16-bit counter [TH1, TL1] overflows, timer 1 will count from 0.
1	0	8-bit auto-reload mode.

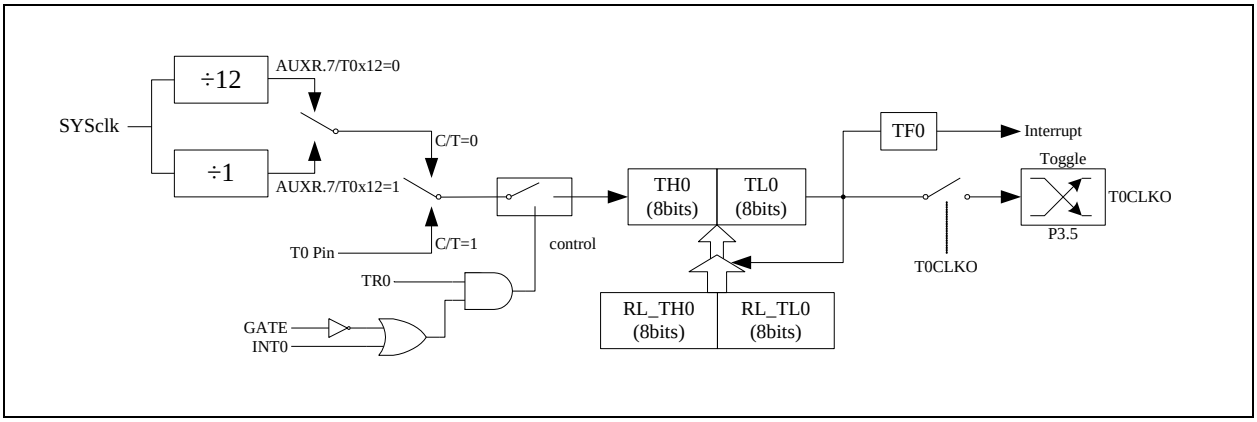
		When the 8-bit counter TL1 overflows, the system loads the reload value in TH1 into TL1 automatically.
1	1	T1 stops working.

T0_M1/T0_M0: T0 mode select bits.

T0_M1	T0_M0	T0 operating mode
0	0	16-bit auto-reload mode. When the 16-bit counter [TH0, TL0] overflows, the system loads the reload value in the internal 16-bit reload register into [TH0, TL0] automatically.
0	1	16-bit non-auto-reload mode. When the 16-bit counter [TH1, TL1] overflows, timer 1 will count from 0.
1	0	8-bit auto-reload mode. When the 8-bit counter TL0 overflows, the system loads the reload value in TH0 into TL0 automatically.
1	1	16-bit auto-reload mode. It is similar to mode 0, whose interrupt can not be disabled. The interrupt has the highest priority, higher than the priority of all other interrupts, and cannot be turned off. It can be used as the system tick timer of the operating system or the system monitoring timer. The only way to stop is to turn off the TR0 bit in the TCON register and stop supplying the clock to Timer 0.

13.2.3 Timer0 mode 0 (16-bit auto-reloadable mode)

In this mode, Timer/Counter 0 is used as a 16-bit counter that can be automatically reloaded, as shown in the figure below.



Timer/Counter0 mode 0: 16-bit auto-reload mode

When GATE=0 (TMOD.3), the timer will count if TR0=1. When GATE=1, it is allowed to control timer0 by external input INT0, so that pulse width measurement can be realized. TR0 is the control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock. T0 counts the internal system clock, and works in timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.4/T0, and T0 works in counting mode.

Timer0 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T0 is determined by T0x12 in the special function register AUXR. If T0x12=0, T0 works in 12T mode, and if T0x12=1, T0 works in 1T mode.

Timer0 has two hidden registers RL_TH0 and RL_TL0. RL_TH0 and TH0 share the same address, and RL_TL0 and TL0 share the same address. When TR0=0, that is, when Timer/Counter0 is disabled, the content written to TL0

will be written to RL_TL0 at the same time, and the content written to TH0 will also be written to RL_TH0 at the same time. When TR0=1, that is, when Timer/Counter0 is allowed to work, writing content to TL0 is not actually written to the current register TL0, but written to the hidden register RL_TL0, and writing content to TH0 is actually also it is not written into the current register TH0, but into the hidden register RL_TH0, which can cleverly realize the 16-bit reload timer. When reading the contents of TH0 and TL0, the contents be read are the contents of TH0 and TL0, not the contents of RL_TH0 and RL_TL0.

When Timer0 is working in mode 0 (TMOD[1:0]/[M1,M0]=00B), the overflow of [TH0,TL0] not only sets TF0, but also automatically reloads the contents of [RL_TH0,RL_TL0] to [TH0,TL0].

If T0CLKO/INT_CLKO.0=1, the P3.5/T1 pin is configured as timer 0's clock output T0CLKO. The output clock frequency is T0 overflow rate/2.

If C/T=0, the timer/counter 0 counts the internal system clock, then:

if T0 works in 1T mode (AUXR.7/T0x12=1), the output clock frequency = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

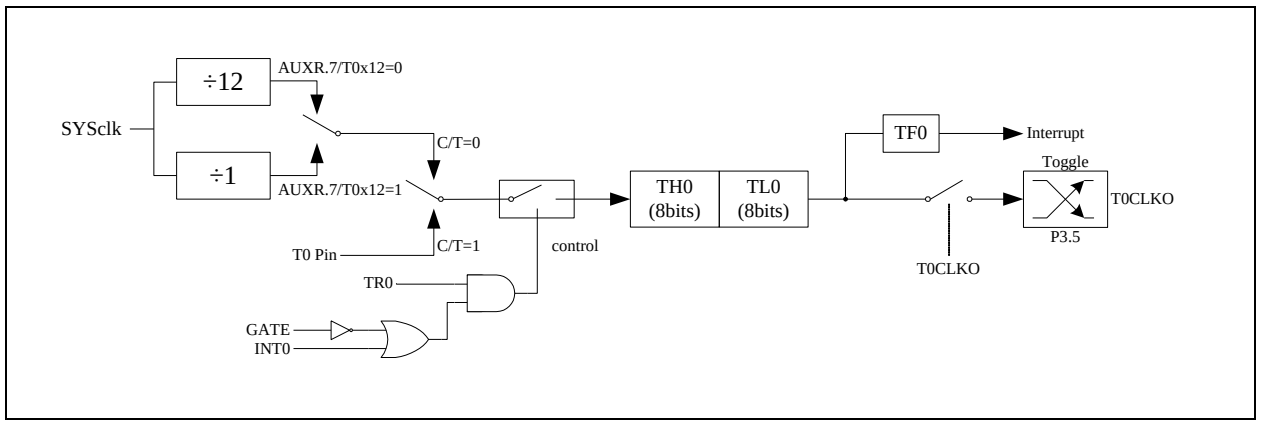
if T0 works in 12T mode (AUXR.7/T0x12=0), the output clock frequency = (SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2

If C/T=1, the timer/counter 0 counts the external pulse input (P3.4/T0), then:

the output clock frequency = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

13.2.4 Timer0 mode 1 (16-bit non-autoreloadable mode)

In this mode, Timer/Counter 0 works in 16-bit non-reloadable mode, as shown in the figure below.



Timer/counter 0 mode 1: 16-bit non-reloadable mode

In this mode, Timer/Counter 0 is configured as a 16-bit non-reloadable mode, which is composed of 8 bits of TL0 and 8 bits of TH0. The 8-bit overflow of TL0 carries over to TH0, and the overflow of TH0 counts the overflow flag TF0 in TCON.

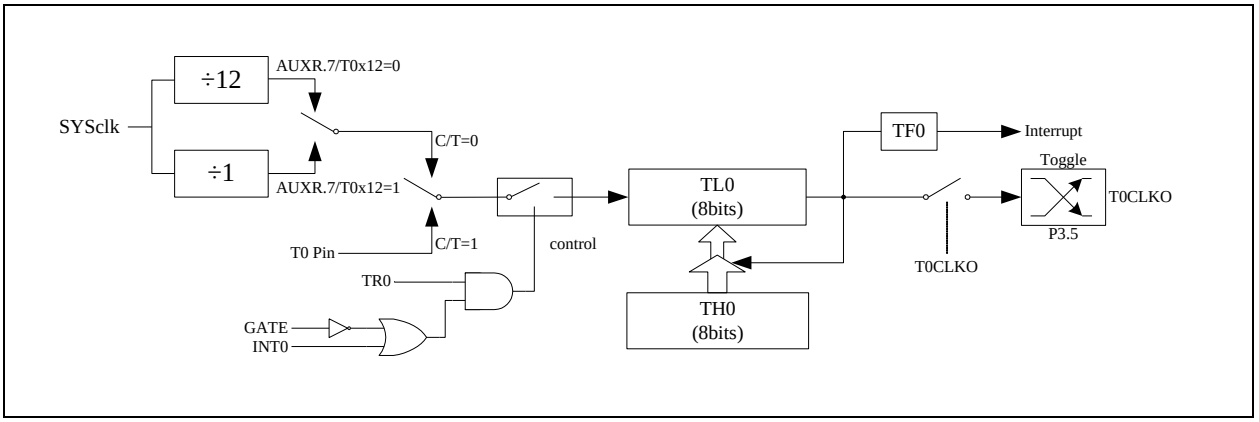
When GATE=0 (TMOD.3), the timer will count if TR0=1. When GATE=1, it is allowed to control timer 0 by external input INT0, so that pulse width measurement can be realized. TR0 is the control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock, T0 counts the internal system clock, and works in timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.4/T0, and T0 works in counting mode.

Timer0 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T0 is determined by T0x12 in the special function register AUXR. If T0x12=0, T0 works in 12T mode, and if T0x12=1, T0 works in 1T mode.

13.2.5 Timer 0 mode 2 (8-bit auto-reloadable mode)

In this mode, Timer/Counter 0 is an 8-bit counter that can be automatically reloaded, as shown in the figure below.



Timer/counter 0 mode 2: 8-bit auto-reloadable mode

The overflow of TL0 not only sets TF0, but also reloads the content of TH0 into TL0. The content of TH0 is preset by software, and its content remains unchanged during reloading.

When T0CLKO/INT_CLKO.0=1, the P3.5/T1 pin is configured as timer 0's clock output T0CLKO. The output clock frequency is T0 overflow rate/2.

If C/T=0, the timer/counter 0 counts the internal system clock, then:

if T0 works in 1T mode (AUXR.7/T0x12=1), the output clock frequency = (SYSclk)/(256-TH0)/2

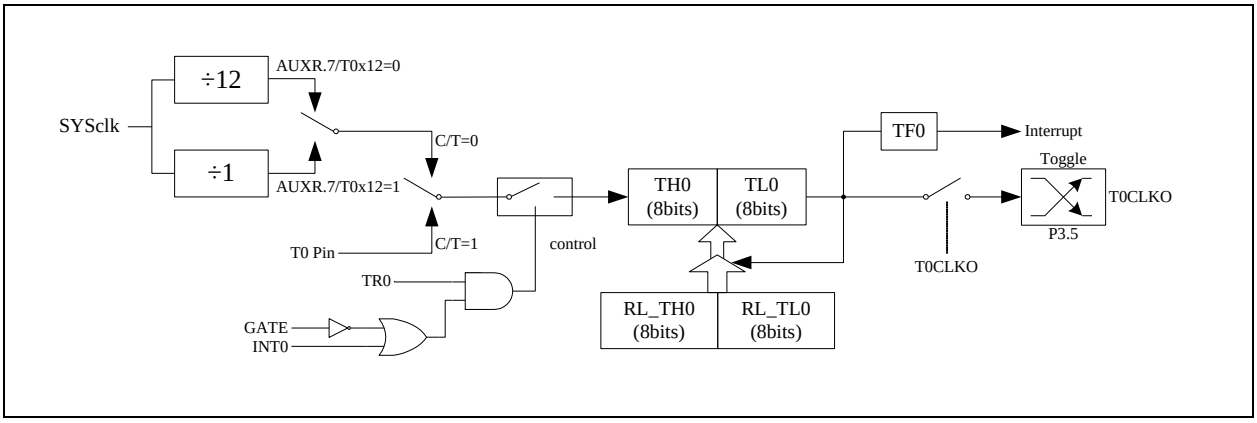
if T0 works in 12T mode (AUXR.7/T0x12=0), the output clock frequency = (SYSclk)/12/(256-TH0)/2

If C/T=1, the timer/counter T0 counts the external pulse input (P3.4/T0), then:

Output clock frequency = (T0_Pin_CLK) / (256-TH0)/2

13.2.6 Timer 0 mode 3 (16-bit auto-realoadable mode with non-maskable interrupt, which can be used as real-time operating system metronome)

For timer/counter 0, its working mode 3 is the same as working mode 0 (the schematic diagram of timer mode 3 in the figure below is the same as working mode 0). The only difference is: when timer/counter 0 is working in mode 3, its interrupt can be enabled just setting ET0/IE.1 (timer/counter 0 interrupt enable bit), and EA/IE.7 (total interrupt enable bit) is not required. The timer/counter 0 interrupt in this mode has nothing to do with the total interrupt enable bit EA. Once the timer/counter 0 interrupt working in mode 3 is enabled (ET0=1), then the interrupt is non-maskable, and the priority of the interrupt is the highest, that is, the interrupt cannot be interrupted by any interrupt, and the interrupt is neither controlled by EA/IE.7 nor controlled by ET0 after it is enabled, When EA=0 or ET0=0, this interrupt cannot be disabled. This mode is so called the 16-bit automatic reload mode with non-maskable interrupt.

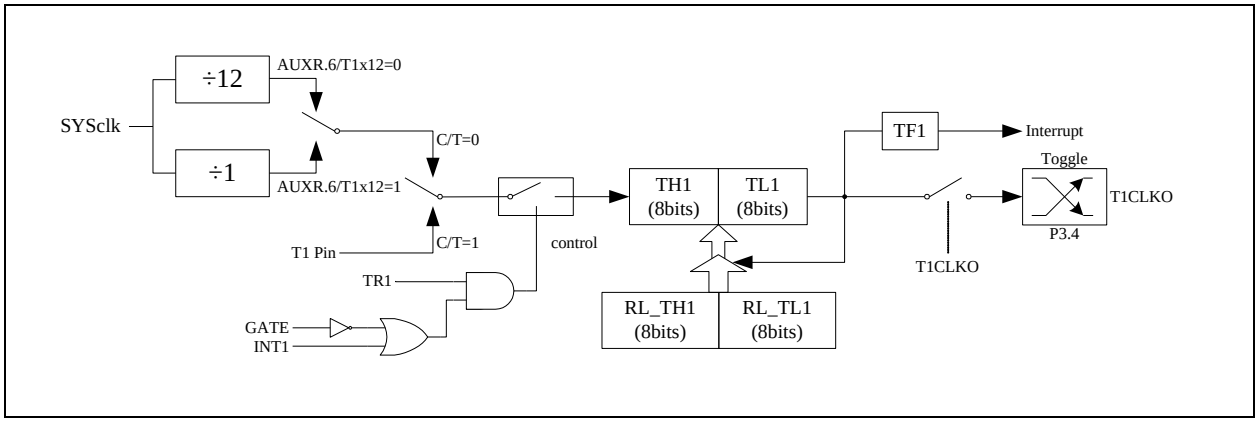


Timer/counter 0 mode 3: 16-bit auto-reload mode with non-maskable interrupt

Note: When Timer/Counter 0 works in mode 3 (16-bit auto-reload mode with non-maskable interrupt), it is not necessary to enable EA/IE.7 (total interrupt enable bit), only ET0/IE.1 is required. (Timer/counter 0 interrupt enable bit) can turn on the timer/counter 0 interrupt. The timer/counter 0 interrupt in this mode has nothing to do with the total interrupt enable bit EA. Once the timer/counter 0 interrupt in this mode is enabled, the timer/counter 0 interrupt priority is the highest, and it cannot be interrupted by any other interrupt (no matter it is lower than the timer/counter 0 interrupt priority). After the interrupt in this mode is enabled, it is neither controlled by EA/IE.7 nor controlled by ET0. Clearing EA nor ET0 can not disable this interrupt.

13.2.7 Timer 1 mode 0 (16-bit auto-reloadable mode)

In this mode, Timer/Counter 1 is used as a 16-bit counter that can be automatically reloaded, as shown in the figure below.



Timer/Counter 1 mode 0: 16-bit auto-reload mode

When GATE=0 (TMOD.7), the timer will count if TR1=1. When GATE=1, it is allowed to control timer1 by external input INT1, so that pulse width measurement can be realized. TR1 is the control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock. T1 counts the internal system clock, and works in timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.5/T1, and T1 works in counting mode.

Timer1 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T1 is determined by T1x12 in the special function register AUXR. If T1x12=0, T1 works in 12T mode, and if T1x12=1, T1 works in 1T mode.

Timer1 has two hidden registers RL_TH1 and RL_TL1. RL_TH1 and TH1 share the same address, and RL_TL1 and TL1 share the same address. When TR1=0, that is, when Timer/Counter1 is disabled, the content written to TL1 will be written to RL_TL1 at the same time, and the content written to TH1 will also be written to RL_TH1 at the same

time. When TR1=1, that is, when Timer/Counter1 starts to work, writing content to TL1 is not actually written to the current register TL1, but written to the hidden register RL_TL1, and writing content to TH1 is actually also it is not written into the current register TH1, but into the hidden register RL_TH1, which can cleverly realize the 16-bit reload timer. When reading the contents of TH1 and TL1, the contents be read are the contents of TH1 and TL1, not the contents of RL_TH1 and RL_TL1.

When Timer1 is working in mode 0 (TMOD[5:4]/[M1,M0]=00B), the overflow of [TH1,TL1] not only sets TF1, but also automatically reloads the contents of [RL_TH1,RL_TL1] to [TH1,TL1].

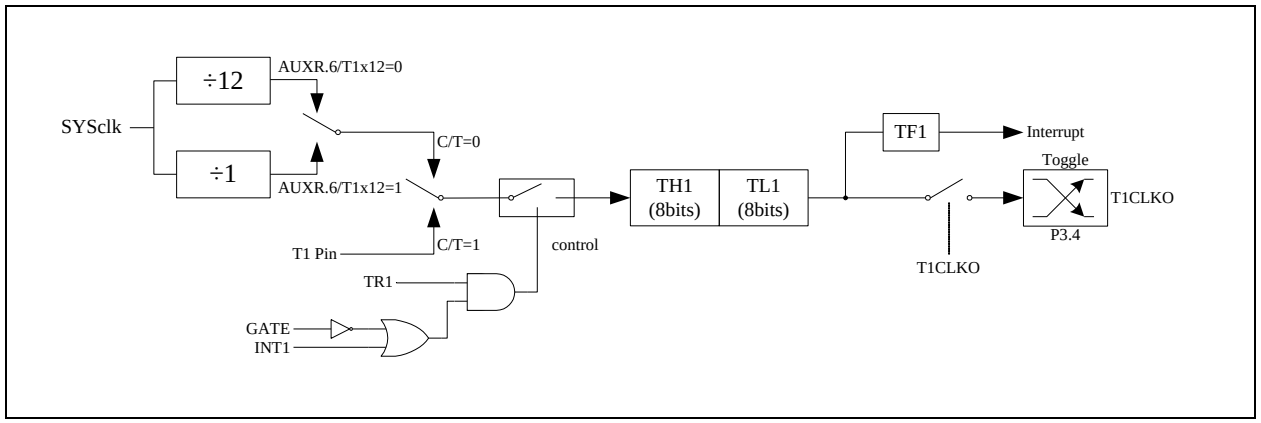
If T1CLKO/INT_CLKO.1=1, the P3.4/T1 pin is configured as timer 1's clock output T1CLKO. The output clock frequency is T1 overflow rate/2.

If C/T=0, the timer/counter 1 counts the internal system clock, then:
if T1 works in 1T mode (AUXR.6/T1x12=1), the output clock frequency = (SYSclk)/(65536-[RL_TH1, RL_TL1])/2
if T1 works in 12T mode (AUXR.6/T1x12=0), the output clock frequency = (SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2

If C/T=1, the timer/counter 1 counts the external pulse input (P3.5/T1), then:
the output clock frequency = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

13.2.8 Timer1 mode 1 (16-bit non-autoreloadable mode)

In this mode, Timer/Counter 1 works in 16-bit non-reloadable mode, as shown in the figure below.



Timer/counter 1 mode 1: 16-bit non-reloadable mode

In this mode, Timer/Counter 1 is configured as a 16-bit non-reloadable mode, which is composed of 8 bits of TL1 and 8 bits of TH1. The 8-bit overflow of TL1 carries over to TH1, and the overflow of TH1 counts the overflow flag TF1 in TCON.

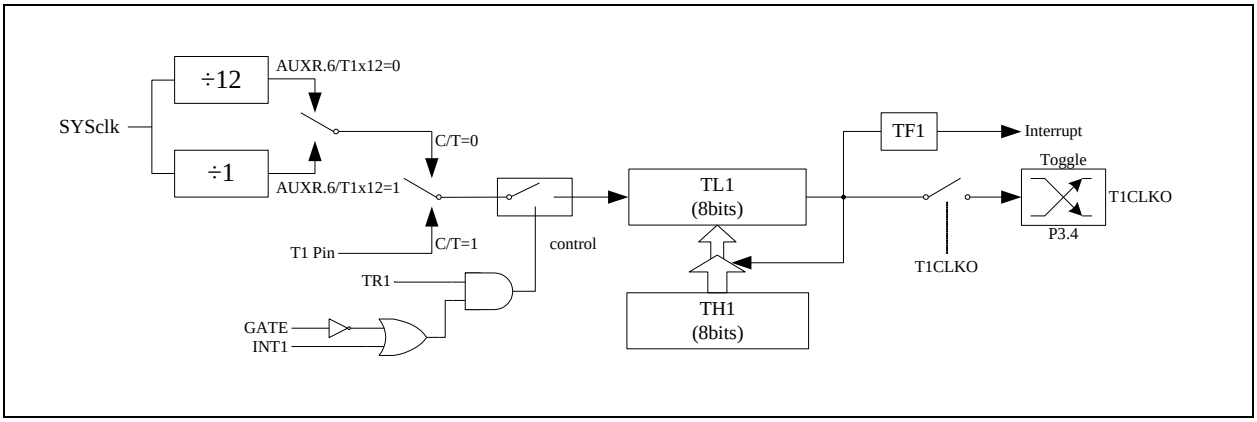
When GATE=0 (TMOD.7), the timer will count if TR1=1. When GATE=1, it is allowed to control timer 1 by external input INT1, so that pulse width measurement can be realized. TR1 is the control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock, T1 counts the internal system clock, and works in timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.5/T1, and T1 works in counting mode.

Timer1 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T1 is determined by T1x12 in the special function register AUXR. If T1x12=0, T1 works in 12T mode, and if T1x12=1, T1 works in 1T mode.

13.2.9 Timer 1 mode 2 (8-bit auto-reloadable mode)

In this mode, Timer/Counter 1 is an 8-bit counter that can be automatically reloaded, as shown in the figure below.



Timer/counter 1 mode 2: 8-bit auto-reloadable mode

The overflow of TL1 not only sets TF1, but also reloads the content of TH1 into TL1. The content of TH1 is preset by software, and its content remains unchanged during reloading.

When T1CLKO/INT_CLKO.1=1, the P3.4/T0 pin is configured as timer 1's clock output T1CLKO. The output clock frequency is T1 overflow rate/2.

If C/T=0, the timer/counter 1 counts the internal system clock, then:

if T1 works in 1T mode (AUXR.6/T1x12=1), the output clock frequency = (SYSclk)/(256-TH1)/2

if T1 works in 12T mode (AUXR.6/T1x12=0), the output clock frequency = (SYSclk)/12/(256-TH1)/2

If C/T=1, the timer/counter T1 counts the external pulse input (P3.5/T1), then:

Output clock frequency = (T1_Pin_CLK) / (256-TH1)/2

13.2.10 Timer 0 Counting Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

When T0 is operating in 16-bit mode (Mode 0, Mode 1, Mode 3), TL0 and TH0 combine into a 16-bit register with TL0 as the low byte and TH0 as the high byte. For 8-bit mode (mode 2), TL0 and TH0 are two independent 8-bit registers.

13.2.11 Timer 1 Counting Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

When T1 is operating in 16-bit mode (Mode 0, Mode 1, Mode 3), TL1 and TH1 combine into a 16-bit register with TL1 as the low byte and TH1 as the high byte. For 8-bit mode (mode 2), TL1 and TH1 are two independent 8-bit registers.

13.2.12 Auxiliary Register 1 (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: T0 speed control bit.

0: The clock source of T0 is SYSclk/12.

1: The clock source of T0 is SYSclk/1.

T1x12: T1 speed control bit.

0: The clock source of T1 is SYSclk/12.

1: The clock source of T1 is SYSclk/1.

13.2.13 External 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

T0CLKO: T0 clock out control bit.

0: Turn off the clock output.

1: P3.5 is configured for T0 clock output pin. When T0 overflows, P3.5 will flip automatically.

T1CLKO: T0 clock out control bit.

0: Turn off the clock output.

1: P3.4 is configured for T1 clock output pin. When T1 overflows, P3.4 will flip automatically.

13.2.14 Timer 0 calculation formula

Mode of Timer	Speed	Period calculation formula
Mode 0/3 (16-bit automatic reload)	1T	$\text{Timer period} = \frac{65536-[TH0,TL0]}{\text{SYSclk}}$ (Auto-reload)
	12T	$\text{Timer period} = \frac{65536-[TH0,TL0]}{\text{SYSclk}} * 12$ (Auto-reload)
Mode 1 (16-bit does not automatically reload)	1T	$\text{Timer period} = \frac{65536-[TH0,TL0]}{\text{SYSclk}}$ (Soft reload)
	12T	$\text{Timer period} = \frac{65536-[TH0,TL0]}{\text{SYSclk}} * 12$ (Soft reload)
Mode 2 (8-bit automatic reload)	1T	$\text{Timer period} = \frac{256-TH0}{\text{SYSclk}}$
	12T	$\text{Timer period} = \frac{256-TH0}{\text{SYSclk}} * 12$ (Auto-reload)

13.2.15 Timer 1 calculation formula

Mode of Timer	Speed	Period calculation formula
Mode 0 (16-bit automatic reload)	1T	$\text{Timer period} = \frac{65536-[TH1,TL1]}{\text{SYSclk}}$ (Auto-reload)
	12T	$\text{Timer period} = \frac{65536-[TH1,TL1]}{\text{SYSclk}} * 12$ (Auto-reload)

Mode 1 (16-bit does not automatically reload)	1T	Timer period $= \frac{65536-[TH1,TL1]}{\text{SYSclk}}$ (Soft reload)
	12T	Timer period $= \frac{65536-[TH1,TL1]}{\text{SYSclk}} * 12$ (Soft reload)
Mode 2 (8-bit automatic reload)	1T	Timer period $= \frac{256-TH1}{\text{SYSclk}}$ (Auto-reload)
	12T	Timer period $= \frac{256-TH1}{\text{SYSclk}} * 12$ (Auto-reload)

13.3 Timer 2 (24-bit timer, 8-bit prescaler + 16-bit timing)

13.3.1 Auxiliary Register 1 (AUXR)

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

TR2: T2 run control bit.

0: T2 stops counting.

1: T2 start counting.

T2_C/T: T2 mode select bit.

0: T2 is used as a timer (input pulse is from internal system clock);

1: T2 is used as a counter (input pulse is from external T2/P1.2 pin).

T2x12: T2 speed control bit.

0: The clock source of T2 is SYSclk/12.

1: The clock source of T2 is SYSclk/1.

13.3.2 External 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

T2CLKO: T2 clock out control bit.

0: Turn off the clock output.

1: P1.3 is configured for T2 clock output pin. When T2 overflows, P1.3 will flip automatically.

13.3.3 Timer 2 Counting Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

T2 operates in 16-bit auto-reload mode. T2L and T2H combine into a 16-bit register with T2L as the low byte and T2H as the high byte. When the 16-bit counter [T2H, T2L] overflows, the system loads the reload value in the internal 16-bit reload register into [T2H, T2L] automatically.

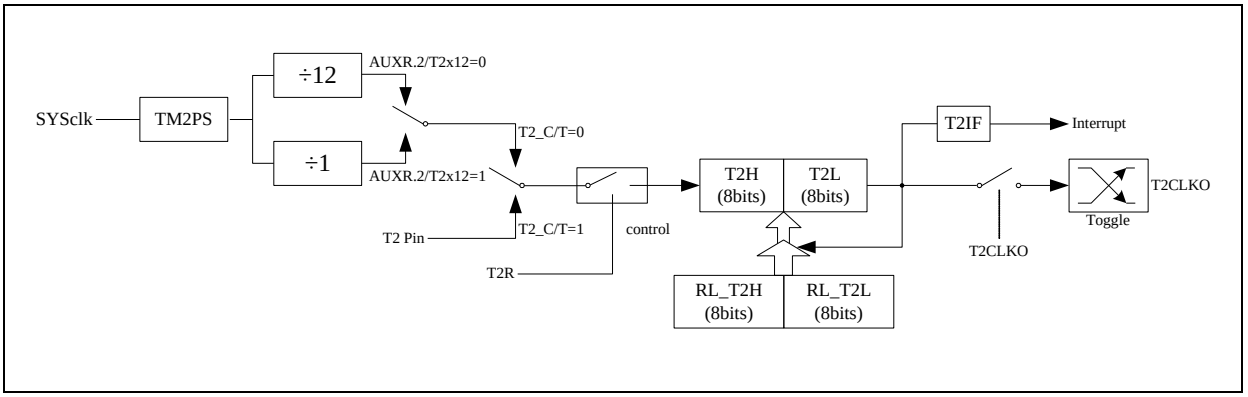
13.3.4 Timer 2 8-bit Prescaler Register (TM2PS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TM2PS	FEA2H								

Timer 2 clock = SYSclk ÷ (TM2PS + 1)

13.3.5 Timer 2 working mode

The functional block diagram of Timer/Counter 2 is as follows.



Timer/counter 2 working mode: 16-bit auto-reload mode

T2R/AUXR.4 is the control bit in the AUXR register. For the specific function description of each bit of the AUXR register, see the introduction of the AUXR register in the previous section.

When T2_C/T=0, the multiplexer is connected to the frequency division output of the system clock, T2 counts the internal system clock, and works in timing mode. When T2_C/T=1, the multiplexer is connected to the external pulse input T2, and T2 works in counting mode.

Timer2 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T2 is determined by T2x12 in the special function register AUXR. If T2x12=0, T2 works in 12T mode, and if T2x12=1, T1 works in 1T mode.

Timer2 has two hidden registers RL_T2H and RL_T2L. RL_T2H and T2H share the same address, and RL_T2L and T2L share the same address. When T2R=0, that is, when Timer/Counter2 is disabled, the content written to T2L will be written to RL_T2L at the same time, and the content written to T2H will also be written to RL_T2H at the same time. When T2R=1, that is, when Timer/Counter2 starts to work, writing content to T2L is not actually written to the current register T2L, but written to the hidden register RL_T2L, and writing content to T2H is actually also it is not written into the current register T2H, but into the hidden register RL_T2H, which can cleverly realize the 16-bit reload timer. When reading the contents of T2H and T2L, the contents be read are the contents of T2H and T2L, not the contents of RL_T2H and RL_T2L.

The overflow of [T2H, T2L] not only sets the interrupt request flag (T2IF), which causes the CPU to switch to the timer 2 interrupt routine, but also automatically reloads the contents of [RL_T2H, RL_T2L] into [T2H, T2L].

13.3.6 Timer 2 calculation formula

Speed	Period calculation formula
1T	$\text{Timer period} = \frac{65536-[T2H,T2L]}{\text{SYSclk}/(\text{TM2PS}+1)}$ (Auto-reload)
12T	$\text{Timer period} = \frac{65536-[T2H,T2L]}{\text{SYSclk}/(\text{TM2PS}+1)} * 12$ (Auto-reload)

13.4 Timer 3/4 (24-bit timer, 8-bit prescaler + 16-bit timing)

13.4.1 Timer4 and Timer 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

TR4: T4 run control bit.

0: T4 stops counting.
1: T4 start counting.

T4_C/T: T4 mode select bit.
0: T4 is used as a timer (input pulse is from internal system clock);
1: T4 is used as a counter (input pulse is from external T4/P0.6 pin).

T4x12: T4 speed control bit.
0: The clock source of T4 is SYSclk/12.
1: The clock source of T4 is SYSclk/1.

T4CLKO: T4 clock out control bit.
0: Turn off the clock output.
1: P0.7 is configured for T4 clock output pin. When T4 overflows, P0.7 will flip automatically.

TR3: T3 run control bit.
0: T3 stops counting.
1: T3 start counting.

T3_C/T: T3 mode select bit.
0: T3 is used as a timer (input pulse is from internal system clock);
1: T3 is used as a counter (input pulse is from external T3/P0.4 pin).

T3x12: T3 speed control bit.
0: The clock source of T3 is SYSclk/12.
1: The clock source of T3 is SYSclk/1.

T3CLKO: T3 clock out control bit.
0: Turn off the clock output.
1: P0.5 is configured for T3 clock output pin. When T3 overflows, P0.5 will flip automatically.

13.4.2 Timer 3 Counting Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

T3 operates in 16-bit auto-reload mode. T3L and T3H combine into a 16-bit register with T3L as the low byte and T3H as the high byte. When the 16-bit counter [T3H, T3L] overflows, the system loads the reload value in the internal 16-bit reload register into [T3H, T3L] automatically.

13.4.3 Timer 4 Counting Registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

T4 operates in 16-bit auto-reload mode. T4L and T4H combine into a 16-bit register with T4L as the low byte and T4H as the high byte. When the 16-bit counter [T4H, T4L] overflows, the system loads the reload value in the internal 16-bit reload register into [T4H, T4L] automatically.

13.4.4 Timer 3 8-bit Prescaler Register (TM3PS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TM3PS	FEA3H								

Timer 3 clock = SYSclk ÷ (TM3PS + 1)

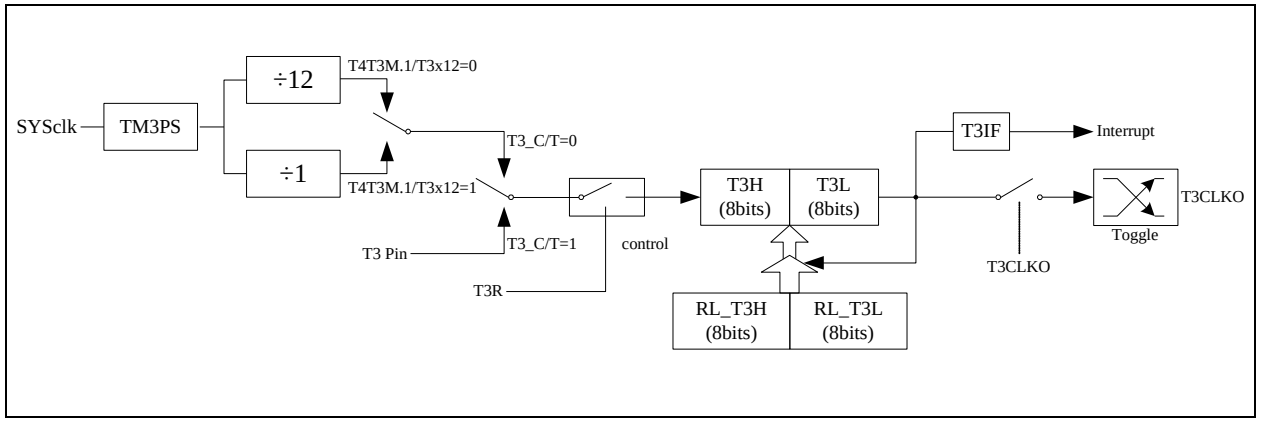
13.4.5 Timer 4 8-bit Prescaler Register (TM4PS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TM4PS	FEA4H								

Timer 4 clock = SYSclk ÷ (TM4PS + 1)

13.4.6 Timer 3 working mode

The functional block diagram of Timer/Counter 3 is as follows.



Timer/counter 3 working mode: 16-bit auto-reload mode

T3R/T4T3M.3 is the control bit in the T4T3M register. For the specific function description of each bit of the T4T3M register, see the introduction of the T4T3M register in the previous section.

When T3_C/T=0, the multiplexer is connected to the frequency division output of the system clock, T3 counts the internal system clock, and works in timing mode. When T3_C/T=1, the multiplexer is connected to the external pulse input T3, and T3 works in counting mode.

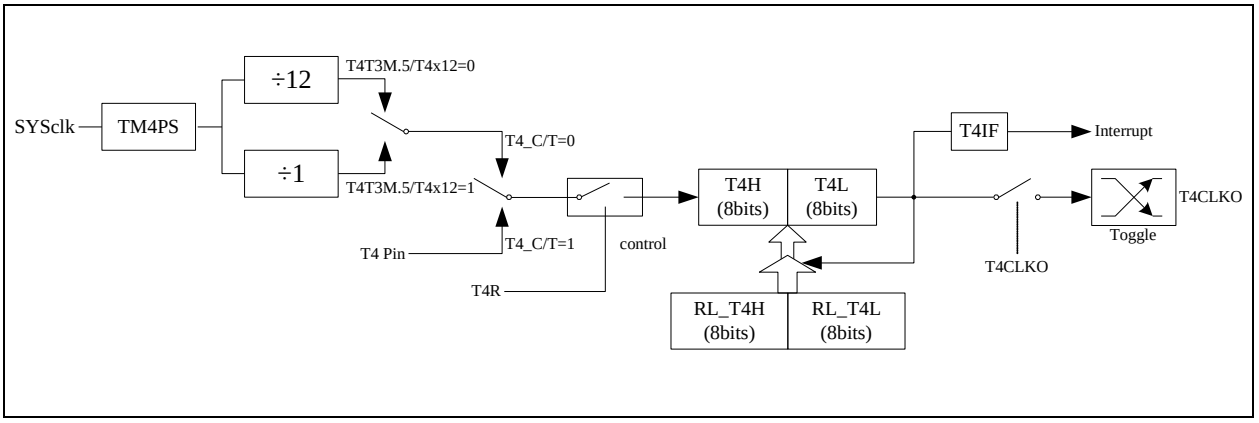
Timer3 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T3 is determined by T3x12 in the special function register T4T3M. If T3x12=0, T3 works in 12T mode, and if T3x12=1, T3 works in 1T mode.

Timer3 has two hidden registers RL_T3H and RL_T3L. RL_T3H and T3H share the same address, and RL_T3L and T3L share the same address. When T3R=0, that is, when Timer/Counter3 is disabled, the content written to T3L will be written to RL_T3L at the same time, and the content written to T3H will also be written to RL_T3H at the same time. When T3R=1, that is, when Timer/Counter3 starts to work, writing content to T3L is not actually written to the current register T3L, but written to the hidden register RL_T3L, and writing content to T3H is actually also it is not written into the current register T3H, but into the hidden register RL_T3H, which can cleverly realize the 16-bit reload timer. When reading the contents of T3H and T3L, the contents be read are the contents of T3H and T3L, not the contents of RL_T3H and RL_T3L.

The overflow of [T3H, T3L] not only sets the interrupt request flag (T3IF), which causes the CPU to switch to the timer 3 interrupt routine, but also automatically reloads the contents of [RL_T3H, RL_T3L] into [T3H, T3L].

13.4.7 Timer 4 working mode

The functional block diagram of Timer/Counter 4 is as follows.



Timer/counter 4 working mode: 16-bit auto-reload mode

T4R/T4T3M.7 is the control bit in the T4T3M register. For the specific function description of each bit of the T4T3M register, see the introduction of the T4T3M register in the previous section.

When T4_C/T=0, the multiplexer is connected to the frequency division output of the system clock, T4 counts the internal system clock, and works in timing mode. When T4_C/T=1, the multiplexer is connected to the external pulse input T4, and T4 works in counting mode.

Timer4 of STC microcontroller has two counting rates: one is 12T mode, which is increased by 1 for every 12 clocks, which is the same as traditional 8051 microcontroller, the other is 1T mode, which is increased by 1 for each clock, and the speed is 12 times of traditional 8051. The rate of T4 is determined by T4x12 in the special function register T4T3M. If T4x12=0, T4 works in 12T mode, and if T4x12=1, T4 works in 1T mode.

Timer4 has two hidden registers RL_T4H and RL_T4L. RL_T4H and T4H share the same address, and RL_T4L and T4L share the same address. When T4R=0, that is, when Timer/Counter4 is disabled, the content written to T4L will be written to RL_T4L at the same time, and the content written to T4H will also be written to RL_T4H at the same time. When T4R=1, that is, when Timer/Counter4 starts to work, writing content to T4L is not actually written to the current register T4L, but written to the hidden register RL_T4L, and writing content to T4H is actually also it is not written into the current register T4H, but into the hidden register RL_T4H, which can cleverly realize the 16-bit reload timer. When reading the contents of T4H and T4L, the contents be read are the contents of T4H and T4L, not the contents of RL_T4H and RL_T4L.

The overflow of [T4H, T4L] not only sets the interrupt request flag (T4IF), which causes the CPU to switch to the timer 4 interrupt routine, but also automatically reloads the contents of [RL_T4H, RL_T4L] into [T4H, T4L].

13.4.8 Timer 3 calculation formula

Speed of Timer	Period calculation formula
1T	$\text{Timer period} = \frac{65536-[T3H,T3L]}{\text{SYSclk}/(\text{TM3PS}+1)}$ (Auto-reload)
12T	$\text{Timer period} = \frac{65536-[T3H,T3L]}{\text{SYSclk}/(\text{TM3PS}+1)} * 12$ (Auto-reload)

13.4.9 Timer 4 calculation formula

Speed of Timer	Period calculation formula
1T	$\text{Timer period} = \frac{65536-[T4H,T4L]}{\text{SYSclk}/(\text{TM4PS}+1)}$ (Auto-reload)
12T	$\text{Timer period} = \frac{65536-[T4H,T4L]}{\text{SYSclk}/(\text{TM4PS}+1)} * 12$ (Auto − reload)

13.5 Example Routines

13.5.1 Timer 0 (Mode 0 – 16-bit auto reload)

C language code

```
//Operating frequency for test is 11.0592MHz

#include “reg51.h”
#include “intrins.h”

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

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;                                //Mode 0
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //Start timer
    ET0 = 1;                                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            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           ;Mode 0
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;Start timer
            SETB    ET0                 ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

13.5.2 Timer 0 (Mode 1 – 16-bit non-auto reload)

C language code

```
//Operating frequency for test is 11.0592MHz
```

```
#include “reg51.h”
#include “intrins.h”

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

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                             //Reset 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;                            //Mode 1
    TL0 = 0x66;                             //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                //Start timer
    ET0 = 1;                                //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            MOV     TL0,#66H            ;Reset 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           ;Mode 1
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;Start timer
            SETB    ET0                 ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

13.5.3 Timer 0 (Mode 2 - 8-bit auto reload)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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;                                //Mode 2
    TL0 = 0xf4;                                 //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;                                    //Start timer
    ET0 = 1;                                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
```

```
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            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           ;Mode 2
            MOV     TL0,#0F4H           ;256-11.0592M/12/76K
            MOV     TH0,#0F4H
            SETB    TR0                 ;Start timer
            SETB    ET0                 ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

13.5.4 Timer 0 (Mode 3 - 16-bit auto reload with non-maskable interrupt)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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;                                //Mode 3
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //Start timer
    ET0 = 1;                                    //Enable timer interrupt
//  EA = 1;                                     // Not controlled by EA

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

        ORG     0000H
        LJMP    MAIN
```

```
                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           ;Mode 3
                MOV         TL0,#66H            ;65536-11.0592M/12/1000
                MOV         TH0,#0FCH
                SETB        TR0                 ;Start timer
                SETB        ET0                 ;Enable timer interrupt
;               SETB        EA                  ; Not controlled by EA

                JMP         $

                END
```

13.5.5 Timer 0 (External count - T0 is extended for external falling edge interrupt)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

```
sfr         P5M0            =   0xca;

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 timer
    ET0 = 1;                                        //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG         0000H
            LJMP        MAIN
            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 timer
            SETB    ET0                 ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

13.5.6 Timer 0 (Pulse width measurement for high-level width of INT0)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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;

void INT0_Isr() interrupt 0
{
    P0 = TL0;                       //TL0 is 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;                                //1T mode
    TMOD = 0x08;                                //Enable GATE, and enable timing when INT0 is 1
    TL0 = 0x00;
    TH0 = 0x00;
    while (INT0);                               //Wait for INT0 to be low
    TR0 = 1;                                    //Start timer
    IT0 = 1;                                    //Enable INT0 falling edge interrupt
    EX0 = 1;
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
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         0003H
            LJMP        INT0ISR

            ORG         0100H
INT0ISR:
            MOV         P0,TL0                  ;TL0 is 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       ;1T mode
            MOV     TMOD,#08H       ;Enable GATE, and enable timing when INT0 is 1
            MOV     TL0,#00H
            MOV     TH0,#00H
            JB      INT0,$          ;Wait for INT0 to be low
            SETB    TR0             ;Start timer
            SETB    IT0             ;Enable INT0 falling edge interrupt
            SETB    EX0
            SETB    EA

            JMP     $

            END
```

13.5.7 Timer 0 (Mode 0, Divided clock output)

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr     INTCLKO     =   0x8f;

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

void main()
```

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

    TMOD = 0x00;                    //Mode 0
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //Start timer
    INTCLKO = 0x01;                 //Enable clock output

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

INTCLKO     DATA    8FH
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

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

```
        MOV     P5M1, #00H

        MOV     TMOD,#00H           ;Mode 0
        MOV     TL0,#66H            ;65536-11.0592M/12/1000
        MOV     TH0,#0FCH
        SETB    TR0                 ;Start timer
        MOV     INTCLKO,#01H        ;Enable clock output

        JMP     $

        END
```

13.5.8 Timer 1 (Mode 0 - 16-bit auto reload)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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;                                //Mode 0
    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //Start timer
    ET1 = 1;                                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            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               ;Mode 0
            MOV     TL1,#66H                ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                     ;Start timer
            SETB    ET1                     ;Enable timer interrupt
            SETB    EA
```

```
        JMP     $

        END
```

13.5.9 Timer 1 (Mode 1 - 16-bit non-auto reload)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;                             //Reset 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;                            //Mode 1
    TL1 = 0x66;                             //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                //Start timer
    ET1 = 1;                                //Enable timer interrupt
```

```
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

        ORG     0000H
        LJMP    MAIN
        ORG     001BH
        LJMP    TM1ISR

        ORG     0100H
TM1ISR:
        MOV     TL1,#66H            ;Reset 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           ;Mode 1
        MOV     TL1,#66H            ;65536-11.0592M/12/1000
        MOV     TH1,#0FCH
        SETB    TR1                 ;Start timer
        SETB    ET1                 ;Enable timer interrupt
        SETB    EA

        JMP     $
```

```
END
```

13.5.10 Timer 1 (Mode 2 - 8-bit auto reload)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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;                                //Mode 2
    TL1 = 0xf4;                                 //256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1;                                    //Start timer
    ET1 = 1;                                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            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           ;Mode 2
            MOV     TL1,#0F4H           ;256-11.0592M/12/76K
            MOV     TH1,#0F4H
            SETB    TR1                 ;Start timer
            SETB    ET1                 ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

13.5.11 Timer 1 (External count – T1 is extended for external falling edge interrupt)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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 timer
    ET1 = 1;                                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            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 timer
            SETB    ET1                 ;Enable timer interrupt
            SETB    EA

            JMP     $

              END
```

13.5.12 Timer 1 (Pulse width measurement for high-level width of INT1)

C language code

```
//Operating frequency for test is 11.0592MHz
```

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

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

sfr         AUXR        =   0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TL1;                                   //TL1 is 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;                                //1T mode
    TMOD = 0x80;                                //Enable GATE, and enable timing when INT1 is 1
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);                               //Wait for INT1 to be low
    TR1 = 1;                                    //Start timer
    IT1 = 1;                                    //Enable INT1 falling edge interrupt
    EX1 = 1;
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH
```

```
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     0013H
            LJMP    INT1ISR

            ORG     0100H
INT1ISR:
            MOV     P0,TL1          ;TL1 is 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       ;1T mode
            MOV     TMOD,#80H       ;Enable GATE, and enable timing when INT1 is 1
            MOV     TL1,#00H
            MOV     TH1,#00H
            JB      INT1,$          ;Wait for INT1 to be low
            SETB    TR1             ;Start timer
            SETB    IT1             ;Enable INT1 falling edge interrupt
            SETB    EX1
            SETB    EA

            JMP     $

            END
```

13.5.13 Timer 1 (Mode 0, Divided clock output)

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr         INTCLKO     =   0x8f;

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

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

    TMOD = 0x00;                    //Mode 0
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //Start timer
    INTCLKO = 0x02;                 //Enable clock output

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

INTCLKO     DATA    8FH
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
```

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     TMOD,#00H           ;Mode 0
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;Start timer
            MOV     INTCLKO,#02H        ;Enable clock output

            JMP     $

            END
```

13.5.14 Timer 1 (Mode 0) is used as baud rate generator of UART1

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr     AUXR    =   0x8e;

sfr     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

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH

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     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 1 (Mode 2) is used as baud rate generator of UART1

C language code

```
//Operating frequency for test 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

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA        8EH

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     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
```

13.5.16 Timer 2 (16-bit auto reload)

C language code

```
//Operating frequency for test 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         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
```

```
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        P10             =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                     //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 timer
    IE2 = ET2;                                      //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

```
            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;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 timer
            MOV     IE2,#ET2            ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

13.5.17 Timer 2 (External count – T2 is extended for external falling edge interrupt)

C language code

```
//Operating frequency for test 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     P0M1    =   0x93;
```

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

sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //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 external counting mode and start timer
    IE2 = ET2;                                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;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 external counting mode and start timer
            MOV     IE2,#ET2            ;Enable timer interrupt
            SETB    EA

            JMP     $

            END
```

13.5.18 Timer 2 (Divided clock output)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

```
sfr     AUXR    =   0x8e;
sfr     INTCLKO =   0x8f;

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

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

    T2L = 0x66;                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                //Start timer
    INTCLKO = 0x04;             //Enable clock output

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

```
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN

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

            MOV         T2L,#66H                ;65536-11.0592M/12/1000
            MOV         T2H,#0FCH
            MOV         AUXR,#10H               ;Start timer
            MOV         INTCLKO,#04H            ;Enable clock output

            JMP         $

            END
```

13.5.19 Timer 2 is used as baud rate generator of UART1

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

sfr         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

```
;Operating frequency for test 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
```

13.5.20 Timer 2 is used as baud rate generator of UART2

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

```
sfr         P5M1            =     0xc9;
sfr         P5M0            =     0xca;

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

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

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

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

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
```

```
                PUSH    ACC
                PUSH    PSW
                MOV     PSW,#08H

                MOV     A,S2CON
                JNB     ACC.1,CHKRI
                ANL     S2CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC.0,UART2ISR_EXIT
                ANL     S2CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S2BUF
                INC     WPTR
UART2ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART2_INIT:
                MOV     S2CON,#10H
                MOV     T2L,#0E8H               ;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 Timer 2 is used as baud rate generator of UART3

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
```

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

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

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

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

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

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

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

```
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR
```

```
            ORG     0100H

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

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART3_INIT:
            MOV     S3CON,#10H
            MOV     T2L,#0E8H           ;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 Timer 2 is used as baud rate generator of UART4

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

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

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

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

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

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

void main()
{
    P0M0 = 0x00;
```

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

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S4CON       DATA    84H
S4BUF       DATA    85H
IE2         DATA    0AFH

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

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

            ORG     0000H
            LJMP    MAIN
```

```
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

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

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#10H
            MOV     T2L,#0E8H           ;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 Timer 3 (16-bit auto reload)

C language code

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

sbit      P10         =   P1^0;

void 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 timer
    IE2 = ET3;                      //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
```

```
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
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

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     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 timer
            MOV     IE2,#ET3        ;Enable timer interrupt
            SETB    EA

            JMP     $
```

```
END
```

13.5.24 Timer 3 (External count – T3 is extended for external falling edge interrupt)

C language code

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

sbit        P10         =   P1^0;

void 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 external counting mode and start timer
    IE2 = ET3;                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz
T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
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

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     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 external counting mode and start timer
        MOV     IE2,#ET3            ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.25 Timer 3 (Divided clock output)

C language code

```
//Operating frequency for test 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     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
```

```
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

    T3L = 0x66;                         //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;                       //Enable clock output and start timer

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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     T3L,#66H                ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#09H              ;Enable clock output and start timer

        JMP     $

        END
```

13.5.26 Timer 3 is used as baud rate generator of UART3

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;

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

bit     busy;
```

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

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

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

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

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

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

```
    P5M1 = 0x00;

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
```

```
                PUSH    ACC
                PUSH    PSW
                MOV     PSW,#08H

                MOV     A,S3CON
                JNB     ACC.1,CHKRI
                ANL     S3CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC.0,UART3ISR_EXIT
                ANL     S3CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S3BUF
                INC     WPTR
UART3ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

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

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

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

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
```

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

                LCALL       UART3_INIT
                MOV         IE2,#08H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART3_SENDSTR

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

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

                END
```

13.5.27 Timer 4 (16-bit auto reload)

C language code

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

sbit    P10     =   P1^0;

void 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 timer
    IE2 = ET4;                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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         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 timer
                MOV         IE2,#ET4            ;Enable timer interrupt
                SETB        EA

                JMP         $

                END
```

13.5.28 Timer 4 (External count – T4 is extended for external falling edge interrupt)

C language code

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

sbit        P10         =   P1^0;

void 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 external counting mode and start timer
    IE2 = ET4;                  //Enable timer interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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     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 external counting mode and start timer
        MOV     IE2,#ET4                ;Enable timer interrupt
        SETB    EA

        JMP     $

        END
```

13.5.29 Timer 4 (Divided clock output)

C language code

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

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

    T4L = 0x66;                                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x90;                               //Enable clock output and start timer

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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     T4L,#66H                ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#90H              ;Enable clock output and start timer

        JMP     $

        END
```

13.5.30 Timer 4 is used as baud rate generator of UART4

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

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

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

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

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

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

    Uart4Init();
```

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

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

```
                MOV     A,S4CON
                JNB     ACC.1,CHKRI
                ANL     S4CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC.0,UART4ISR_EXIT
                ANL     S4CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S4BUF
                INC     WPTR
UART4ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART4_INIT:
                MOV     S4CON,#50H
                MOV     T4L,#0E8H               ;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 UART Communication

There are 4 full duplex asynchronous serial communication ports (UART in short) in STC8A8K64D4 series of microcontrollers. Each UART consists of two data buffers, a shift register, a serial control register and a baud rate generator. Each UART data buffer consists of two independent receive and transmit buffers, which can transmit and receive data simultaneously.

There are 4 modes for UART1 of STC8A8K64D4 series of microcontrollers, the baud rates of two modes of them are variable, and the baud rates of the other two modes are fixed. They can be chosen for different applications. There are only two modes in UART2, UART3 and UART4, and their baud rates are variable. Different baud rates and different modes can be set by software. It is flexible for the host to query the receiving or sending process, or use the interrupt method.

All the pins of UART1, UART2, UART3 and UART4 can be switched amoung multiple groups of ports using the pin switching function, so that a serial port can be multiplexed into severial serial ports in a time-sharing manner.

14.1 UART function pin switch

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

S1_S[1:0]: UART1 function pin selection bit

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

S4_S: UART4 function pin selection bit

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

S3_S: UART3 function pin selection bit

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

S2_S: UART2 function pin selection bit

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

14.2 Registers Related to UARTs

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	UART1 control register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	UART1 data buffer register	99H									0000,0000
S2CON	UART2 control register	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	UART2 data buffer registe	9BH									0000,0000
S3CON	UART3 control register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	UART3 data buffer register	ADH									0000,0000
S4CON	Serial port 4 control register	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	Serial port 4 data buffer register	85H									0000,0000
PCON	Power control register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	Auxiliary register 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	UART1 slave address register	A9H									0000,0000
SADEN	UART1 slave address enable register	B9H									0000,0000

14.3 UART1

14.3.1 UART1 control register

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

SM0/FE: If the SMOD0 bit in the PCON register is 1, this bit is the frame error detection flag. When the UART detects an invalid stop bit during reception, it is set by the UART receiver and must be cleared by software. If SMOD0 bit in PCON register is 0, this bit and SM1 specify the communication mode of UART1 as shown in the following table:

SM0	SM1	Mode of UART1	Function description
0	0	Mode 0	synchronous shift serial mode
0	1	Mode 1	8-bit UART, whose baud-rate is variable
1	0	Mode 2	9-bit UART, whose baud-rate is fixed
1	1	Mode 3	9-bit UART, whose baud-rate is variable

SM2: Mode 2 or mode 3 multi-machine communication enable control bit. When UART1 adopts mode 2 or mode 3, if the SM2 bit is 1 and the REN bit is 1, the receiver is in the Address Frame Filter state. In this case, the received 9th bit (RB8) can be used to filter the address frame. If RB8 = 1, it indicates that the frame is an address frame, the address information can enter SBUF and set RI bit. The address information is compared in the interrupt service routine. If RB8 = 0, it indicates that the frame is not an address frame, which should be discarded and keep RI = 0. In mode 2 or mode 3, if the SM2 bit is 0 and the REN bit is 1, the receiver is in a state where the address frame filtering is disabled. The received message can enter SBUF regardless of whether RB8 is 0 or 1, and make RI = 1. Here, RB8 is usually used as a check bit. Mode 1 and mode 0 are non-multi-machine communication modes. In these two modes, SM2 should be set to 0.

REN: Receive enable control bit.
0: disable UART1 receive data.
1: enable UART1 receive data.

TB8: The 9th bit be transmitted for UART1 in mode 2 and 3. It can be set or cleared by software. It is not used in mode 0 and mode 1.

RB8: The 9th bit received for UART1 in mode 2 and 3 which is usually used as a check bit or address frame/data frame flag. It is not used in mode 0 and mode 1.

TI: Transmit interrupt request flag of UART1. In mode 0, when the ransmission of the 8th bit completes, TI is set by the hardware automatically and requests the interrupt to the CPU. After the CPU responds the interrupt, TI must be cleared by software. In other modes, TI is set by the hardware automatically at the start of the stop bit transmittion and requests interrupts to the CPU. TI must be cleared by software after the interrupt is responded.

RI: Receive interrupt request flag of UART1. In mode 0, when the serial port receives the 8th bit of datum, RI is set by the hardware automatically and requests interrupt to the CPU. After the interrupt is responded, RI must be cleared by software. In other modes, RI is set by hardware automatically at the middle of stop bit the serial port received, and requests the interrupt to the CPU. After the interrupt is responded, RI must be cleared by software.

14.3.2 UART1 data register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF: It is used as the buffer in transmission and receiving of UART1. SBUF is actually two buffers, reading buffer and writing buffer. Two operations correspond to two different registers, one is write-only register (writing buffer), the other is read-only register (reading buffer). In fact, the CPU reads serial receive buffer when reads SBUF. When CPU writes to the SBUF will trigger the serial port to start sending data.

14.3.3 Power control register

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

SMOD: double Baud rate of UART1 control bit.

0: disable double baud rate of the UART1.

1: enable double baud rate of the UART1.

SMOD0: Frame error detection control bit.

0: No frame error detection function, SCON.7 is SM0 function.

1: enable frame error detection function. The function of SM0/FE is FE.

14.3.4 Auxiliary register 1

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

UART_M0x6: Baud rate select bit of UART1 while it works in mode 0.

0: The baud-rate of UART in mode 0 is SYSclk/12.

1: The baud-rate of UART in mode 0 is SYSclk/2.

S1ST2: UART1 baud rate generator select bit.

0: Select Timer 1 as the baud-rate generator of UART1.

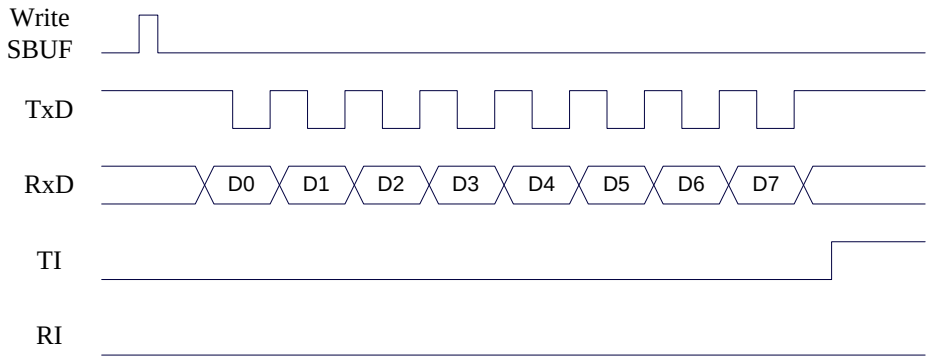
1: Select Timer 2 as the baud-rate generator of UART1.

14.3.5 UART1 Mode 0

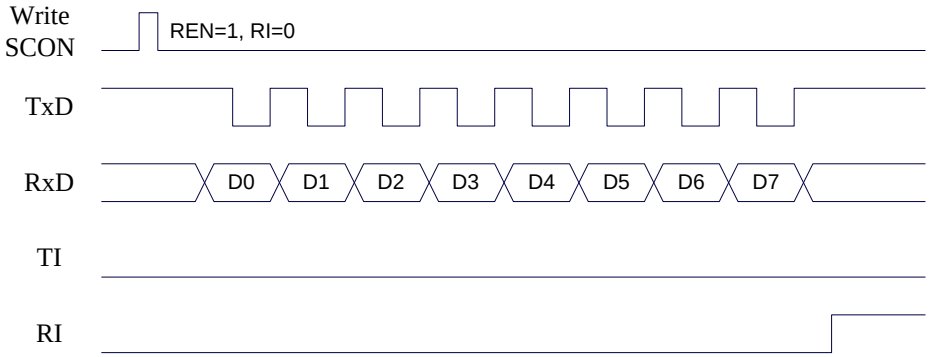
When mode 0 is selected for UART1, UART1 operates in synchronous shift register mode. When the serial port mode 0 communication speed setting bit UART_M0x6 is 0, the baud rate is fixed to SYSclk/12. When UART_M0x6 is 1, the baud rate is fixed to SYSclk/2. RxD is used as serial communication data pin, TxD is used as synchronous shift pulse output pin. 8-bit data are transmitted and received, LSB first.

Transmission process of mode 0: Transmission is initiated by any instruction that write data to SBUF. The 8-bit datum is output from the RxD pin at the baud rate of SYSclk/12 or SYSclk/2 (determined by the UART_M0x6 divided by 12 or 2), from LSB to MSB. The TxD pin outputs the synchronous shift pulse signal. The interrupt flag TI will be set when transmittion is completed. When the write signal is valid, the transmit control signal SEND is active (high) one clock apart, allowing RxD to send data while allowing the TxD output the synchronous shift pulse. When a frame (8 bits) of datum is sent, all control signals are reset to the original status, and only TI keeps high level and keeps the interrupt request status. TI must be cleared by software before sending data again.

Receiving process of mode 0: Receiving is initiated by setting REN and the receive interrupt request flag RI=0. After starting the receive process, RxD is the serial data input pin and TxD is the synchronous pulse output pin. The serial receiving baud rate is SYSclk/12 or SYSclk/2 (determined by UART_M0x6 is 12 or 2). After receiving a frame of datum (8 bits), the control signal is reset and the interrupt flag RI is set to 1, the interrupt requet status appears. RI must be cleared by software for the next receiving data.



Transmitting data (UART1 mode 0)



Receiving data (UART1 mode 0)

In mode 0, SM2 must be cleared so that TB8 and RB8 bits are not affected. Since the baud rate is fixed at SYSclk/12 or SYSclk/2, no timer is required and the clock of the microcontroller is used as the synchronous shift pulse directly.

The baud rates of UART1 mode 0 are shown in the following table, where SYSclk is the system operating frequency:

UART_M0x6	Baud rate calculation formula
0	$\text{Baud rate} = \frac{\text{SYSclk}}{12}$
1	$\text{Baud rate} = \frac{\text{SYSclk}}{2}$

14.3.6 UART1 Mode 1

If SM0 and SM1 of SCON are set to '01' by the software, UART1 will work in mode 1, which is a 8-bit UART mode. In mode 1, a frame of information consists 10 bits: 1 start bit, 8 data bits (LSB first) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD is the data transmitting pin, and RxD is the data receiving pin, the UART is a full duplex receiver/transmitter.

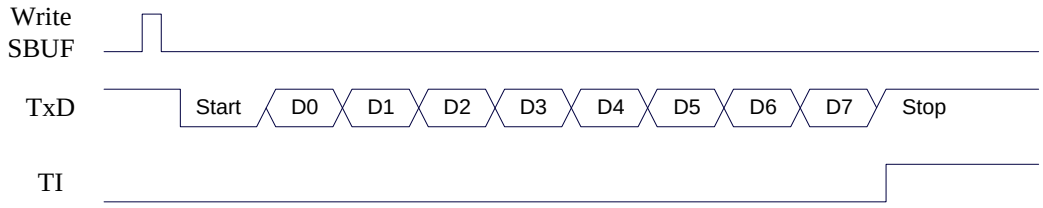
Transmission process of mode 1: TxD is used as data output pin when transmitting a datum. Transmission is initiated by writing SBUF. "1" is also written into the 9th bit of transmission shift register by the writing "SBUF" signal, and the TX control unit is notified to start sending. The shift register shifts the data right to TxD to send, and shifts "0" in the left to supplement. When the highest bit of data is shifted to the output of the shift register, it is followed by the 9th bit "1", and all bits to the left of it are "0". This state causes the TX control unit to make the last shift output, and then disables the transmission signal "SEND" to complete the transmission of a frame and sets TI, and requests interrupt processing to CPU.

Receiving process of mode 1: After the software sets the reception enable flag REN, i.e. REN = 1, the receiver will detect the RxD pin signal. The receiver is ready to receive data when a "1" → "0" falling edge is detected at RxD pin, and

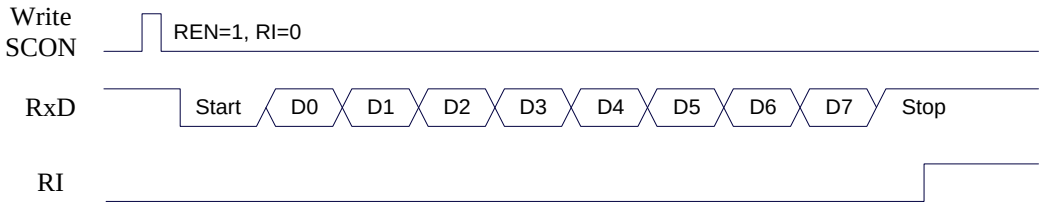
resets the receiving counter of the baud rate generator immediately, loads 1FFH into the shift register. The received datum is shifted in from the right of the receiving shift register, the loaded 1FFH is shifted out to the left. When the start bit "0" is shifted to the far left of the shift register, the RX controller shifts for the last time and completes a frame receiving. The received datum is valid only if the following two conditions are met:

- RI=0;
- SM2=0 or the stop bit received is 1.

The datum received is loaded into SBUF, the stop bit is loaded into RB8, RI flag is set to request interrupt to CPU. If the two conditions can not be met at the same time, the received data is invalid and is discarded. Regardless of the conditions are met or not, the receiver will re-test RxD pin of the "1" → "0" edge, and continue to receive the next frame. If the received datum is valid, the RI flag must be cleared by software in the interrupt service routine. Usually, SM2 is set to "0" when serial port is operating in mode 1.



Transmitting data (UART1 mode 1)



Receiving data (UART1 mode 1)

The baud rate of UART1 is variable. It can be generated by T1 or T2. If the timer is in 1T mode (12x speed), the corresponding baud rate is increased by 12 times.

The baud rate of UART1 mode 1 is calculated as follows, where SYSclk is the system operating frequency.

Timer selected	Speed of timer	Baud rate calculation formula
T2	1T	$\text{reload value of T2} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of timer 2} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$
T1 mode 0	1T	$\text{reload value of T1} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T1} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$
T1 mode 2	1T	$\text{reload value of T1} = 256 - \frac{2^{SMOD} \times \text{SYSclk}}{32 \times \text{baud rate}}$
	12T	$\text{reload value of T1} = 256 - \frac{2^{SMOD} \times \text{SYSclk}}{12 \times 32 \times \text{baud rate}}$

The reload value of the timers corresponding to the common frequency and the common baud rate are as following.

Frequency (MHz)	Baud rate	T2		T1 mode 0		T1 mode 2			
						SMOD=1		1T mode	
		1T mode	12T mode	1T mode	12T mode	1T mode	12T mode	1T mode	12T mode
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

14.3.7 UART1 Mode 2

If the two bits of SM0 and SM1 are '10', UART1 operates in mode 2. UART1 operating in mode 2 is a 9-bit data asynchronous communication UART. One frame of data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9th bit) and 1 stop bit. The transmitted programmable bit (9th bit) is supplied by TB8 in SCON, which can be confugred as either 1 or 0 by software. Or, the odd/even parity bit P in the PSW can be loaded into TB8. Not only can TB8 be used as a multi-machine communication address/data flag, but also it can be used as datum parity check bit. The 9th bit is received into RB8 of SCON. TxD is the transmitting pin, and RxD is the receiving pin, the serial port is a full duplex receiver/transmitter.

The baud rate of mode 2 is fixed to the system clock divided by 64 or 32 depending on the value of SMOD in PCON.

The baud rate of UART1 mode 2 is shown in the following table, where SYSclk is the system operating frequency.

SMOD	Baud rate calculation formula
0	$\text{baud rate} = \frac{\text{SYSclk}}{64}$
1	$\text{baud rate} = \frac{\text{SYSclk}}{32}$

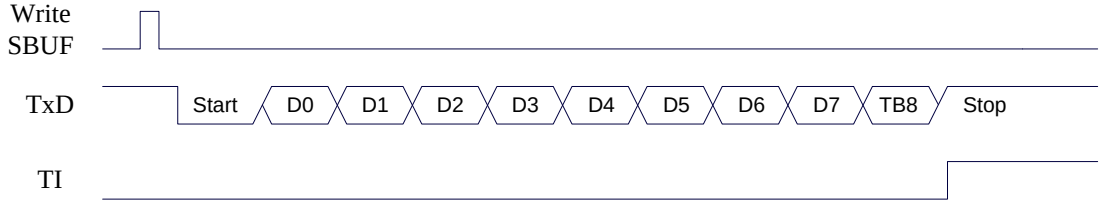
Except that the source of the baud rate is slightly different, and the 9th bit of the shift register supplied by TB8 while being sent is different, the functional and structure of mode 2 and mode 1 are basically the same, the receiving / sending operation and timing of mode 2 and mode 1are also basically the same.

After the receiver receives a frame of information, the following conditions must be met at the same time.

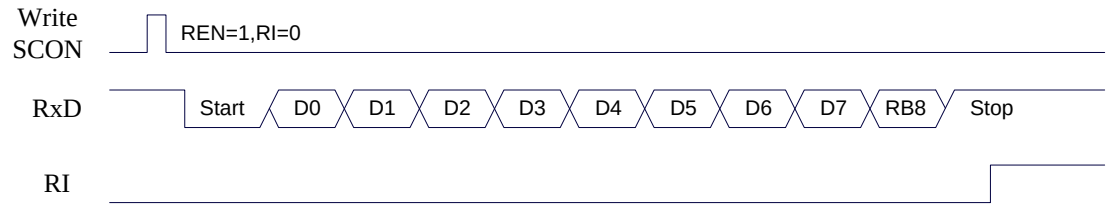
- RI=0
- SM2=0 or SM2=1 and the 9th bit received RB8=1.

Only when the two conditions above are met at the same time, the data received in shift register is loaded into SBUF and RB8. The RI flag is set to 1, and the interrupt is requested to CPU. If one of the above conditions is not met, the data received in the shift register is invalid and is discarded, and RI is not set. Regardless of the above conditions are met or not, the receiver begins to detect the RxD pin hopping information again to receive the next frame of information. In mode 2, the received stop bit is not related to SBUF, RB8 and RI.

It provides for the convenience of multi-machine communication by setting SM2, TB8 of SCON and communication protocol using the software.



Transmitting data (UART1 mode 2)



Receiving data (UART1 mode 2)

14.3.8 UART1 Mode 3

If the two bits of SM0 and SM1 are '11', UART1 operates in mode 3. UART1 operating in mode 3 is a 9-bit data asynchronous communication UART. One frame of data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9th bit) and 1 stop bit. The transmitted programmable bit (9th bit) is supplied by TB8 in SCON, which can be confugred as either 1 or 0 by software. Or, the odd/even parity bit P in the PSW can be loaded into TB8. Not only can TB8 be used as a multi-machine communication address/data flag, but also it can be used as datum parity check bit. The 9th bit is received into RB8 of SCON. TxD is the transmitting pin, and RxD is the receiving pin, the serial port is a full duplex receiver/transmitter.

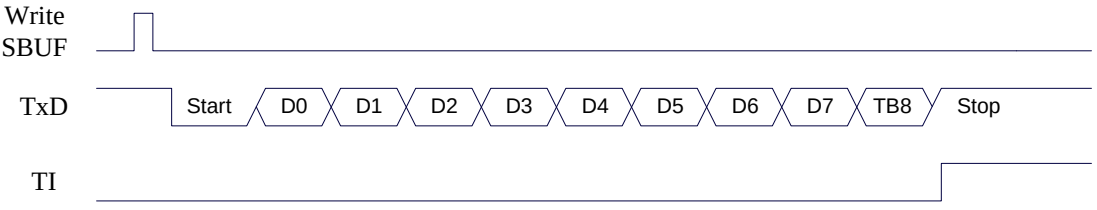
Except that the 9th bit of the shift register supplied by TB8 while being sent is different, the functional and structure of mode 3 and mode 1 are basically the same, the receiving / sending operation and timing of mode 3 and mode 1 are also basically the same.

After the receiver receives a frame of information, the following conditions must be met at the same time.

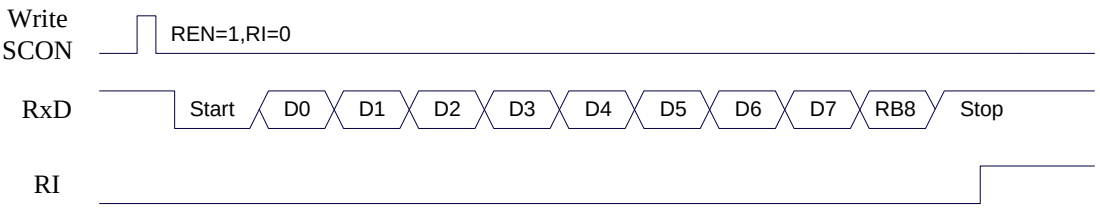
- RI=0
- SM2=0 or SM2=1 and the 9th bit received RB8=1.

Only when the two conditions above are met at the same time, the data received in shift register is loaded into SBUF and RB8. The RI flag is set to 1, and the interrupt is requested to CPU. If one of the above conditions is not met, the data received in the shift register is invalid and is discarded, and RI is not set. Regardless of the above conditions are met or not, the receiver begins to detect the RxD pin hopping information again to receive the next frame of information. In mode 3, the received stop bit is not related to SBUF, RB8 and RI.

It provides for the convenience of multi-machine communication by setting SM2, TB8 of SCON and communication protocol using the software.



Transmitting data (UART1 mode 3)



Receiving data (UART1 mode 3)

The baud rate calculation formula of UART1 mode 3 is exactly the same as that of mode 1. Please refer to the mode 1 baud rate calculation formula.

14.3.9 Automatic Address Recognition

14.3.10 UART1 slave address control registers

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR: Slave address register
SADEN: Slave address mask register

The automatic address recognition function is typically used in the field of multi-machine communications. Its main principle is that the slave system identifies the address information from the master serial port data stream through the hardware comparison function. The address of the slave is set by the registers SADDR and SADEN. The hardware filters the slave address automatically. The hardware will generate a serial port interrupt when the slave address information from the master matches the slave address set by the slave. Otherwise, the hardware will discard the serial port data automatically without any interruption. When a number of slaves in Idle mode are connected together, only the slave that matches the slave address will wake up from Idle mode. Then the power consumption of the slave MCU reduces greatly. Constantly entering the serial port interrupt which reduces the system execution efficiency can be avoided even if the slave is in normal operation.

To use the automatic address recognition feature of the serial port, mode 2 or mode 3 of the serial port of the MCU that participates in communication is selected. Usually the mode 3 with variable baud rate is selected because the baud rate of mode 2 is fixed, and it is inconvenient to adjust. SM2 bit of slave in SCON is set to 1. The 9th bit which is stored in RB8 is the address/data flag in mode 2 or 3. When the 9th bit is 1, it indicates the previous 8-bit datum stored in SBUF is the address information. If SM2 is set to 1, the slave MCU will filter out non-address data whose 9th bit is 0 automatically while the address data whose 9th bit is 1 in SBUF will automatically be matched with the address set in SADDR and SADEN. If the address matches, RI will be set to "1" and an interrupt will occur. Otherwise the received data is discarded.

The slave address is set by two registers, SADDR and SADEN. SADDR is the slave address register, where the slave address is stored. SADEN is the slave address mask register, which is used to set the ignore bit in the address information. The setting method is as follows.

For example
SADDR ＝ 11001010
SADEN ＝ 10000001
Then the matched address is 1xxxxxx0
That is, as long as bit 0 is 0 and bit 7 is 1 in the address data sent by the master, the address can be matched with the local address.
Another example
SADDR ＝ 11001010
SADEN ＝ 00001111
Then the matched address is xxxx1010
That is, as long as the low 4 bits are 1010 in the address data sent by the master, the address can be matched with the local address. The high 4 bits can be any value and are ignored.
The Broadcast Address (FFH) can be used by the master to select all the slaves simultaneously for communication.

14.4 UART2

14.4.1 UART2 control register

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

S2SM0: Serial port 2 mode select bit.

S2SM0	UART2 mode	Function description
0	Mode 0	8-bit UART, whose baud-rate is variable
1	Mode 1	9-bit UART, whose baud-rate is variable

S2SM2: UART2 multi-machine communication control enable bit. In mode 1, if the S2SM2 bit is 1 and the S2REN bit is 1, the receiver is in the address frame filter state. In this case, the received 9th bit (S2RB8) can be used to filter the address frame. If S2RB8 = 1, the frame is the address frame, address information can enter S2BUF, S2RI becomes 1, and then address can be compared in the interrupt service routine. If S2RB8 = 0, it indicates that the frame is not an address frame and should be discarded and keep S2RI = 0. In mode 1, if the S2SM2 bit is 0 and the S2REN bit is 1, the receiver is in the address frame filter disabled state. Regardless of the received S2RB8 is 0 or 1, the information received can enter into the S2BUF, and make S2RI = 1. Here, S2RB8 is usually used as check bit. Mode 0 is non-multi-machine communication mode, where S2SM2 should be 0.

S2REN: Receive enable control bit.

0: disable UART2 receive data.

1: enable UART2 receive data.

S2TB8: S2TB8 is the 9th bit of datum to be sent when UART2 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S2RB8: S2RB8 is the 9th bit of datum recieved when UART2 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S2TI: Transmit interrupt request flag of UART2. S2TI is set by the hardware automatically at the beginning of the stop bit transmittion and requests interrupts to the CPU. S2TI must be cleared by software after the interrupt is responded.

S2RI: Receive interrupt request flag of UART2. S2RI is set by hardware automatically at the middle of stop bit received, and requests the interrupt to the CPU. After the interrupt is responded, S2RI must be cleared by software.

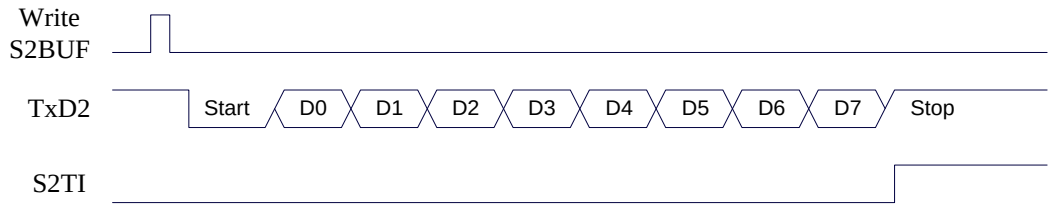
14.4.2 UART2 data register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

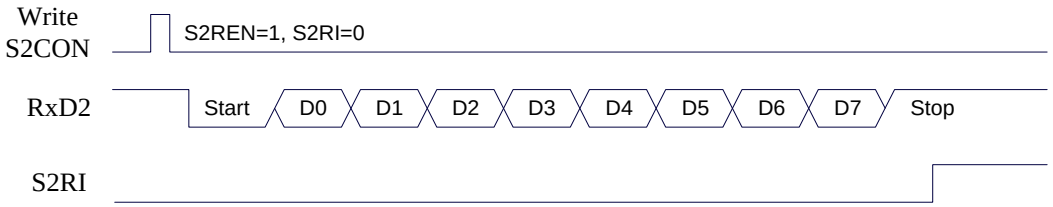
S2BUF: It is used as the buffer in transmission and receiving for UART2. S2BUF is actually two buffers, reading buffer and writing buffer. Two operations correspond to two different registers, one is write-only register (writing buffer), the other is read-only register (reading buffer). The CPU reads serial receiving buffer when reads S2BUF, and writes to the S2BUF will trigger the serial port to start sending data.

14.4.3 UART2 Mode 0

UART2 mode 0 is 8-bit UART mode with ariable baud rate. In this mode, a frame of data consists 10 bits: 1 start bit, 8 data bits (LSB first) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD2 is the data transmitting pin, and RxD2 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART2 mode 0)



Receiving data (UART2 mode 0)

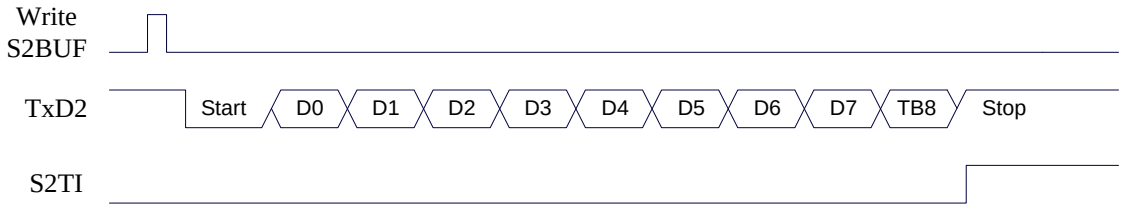
The baud rate of UART2 is variable. It is generated by T2. If the timer is in 1T mode (12x speed), the corresponding baud rate is increased by 12 times.

The baud rate of UART2 mode 0 is calculated as follows, where SYSclk is the system operating frequency.

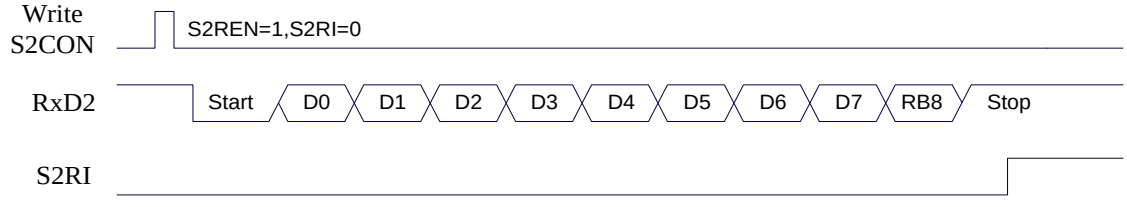
Timer selected	Speed of timer	Baud rate calculation formula
T2	1T	$\text{reload value of timer 2} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of timer 2} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$

14.4.4 UART2 Mode 1

UART2 operating in mode 1 is a 9-bit data UART mode with variable baud rate. One frame data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9th bit) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD2 is the data transmitting pin, and RxD2 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART2 mode 1)



Receiving data (UART2 mode 1)

The baud rate calculation formula of UART2 mode 1 is exactly the same as that of mode 0. Please refer to the mode 0 baud rate calculation formula.

14.5 UART3

14.5.1 UART3 control register

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

S3SM0: UART3 mode select bit.

S3SM0	UART3 mode	Function description
0	Mode 0	8-bit UART, whose baud-rate is variable
1	Mode 1	9-bit UART, whose baud-rate is variable

S3ST3: UART3 baud rate generator select bit.

0: Select T2 as the baud-rate generator of UART3.

1: Select T3 as the baud-rate generator of UART3.

S3SM2: UART3 multi-machine communication control bit. In mode 1, if the S3SM2 bit is 1 and the S3REN bit is 1, the receiver is in the address frame filter state. In this case, the received 9th bit (S3RB8) can be used to filter the address frame. If S3RB8 = 1, the frame is the address frame, address information can enter S3BUF, S3RI becomes 1, and then the address is compared with the slave address in the interrupt service routine. If S3RB8 = 0, it indicates that the frame is not an address frame and should be discarded and keep S3RI = 0. In mode 1, if the S3SM2 bit is 0 and the S3REN bit is 1, the receiver is in the address frame filter disabled state. Regardless of the received S3RB8 is 0 or 1, the information received can enter into the S3BUF, and make S3RI = 1. Here, S3RB8 is usually used as parity check bit. Mode 0 is non-multi-machine communication mode, where S3SM2 should be 0.

S3REN: Receive enable control bit.

0: disable UART3 receive data.

1: enable UART3 receive data.

S3TB8: S3TB8 is the 9^{th} bit of datum to be sent when UART3 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S3RB8: S3RB8 is the 9^{th} bit of datum recieved when UART3 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S3TI: Transmit interrupt request flag of UART3. S3TI is set by the hardware automatically at the beginning of the stop bit transmittion and requests interrupt to the CPU. S3TI must be cleared by software after the interrupt is responded.

S3RI: Receive interrupt request flag of UART3. S3RI is set by hardware automatically at the middle of stop bit the serial port received, and requests interrupt to the CPU. After the interrupt is responded, S3RI must be cleared by software.

14.5.2 UART3 data register

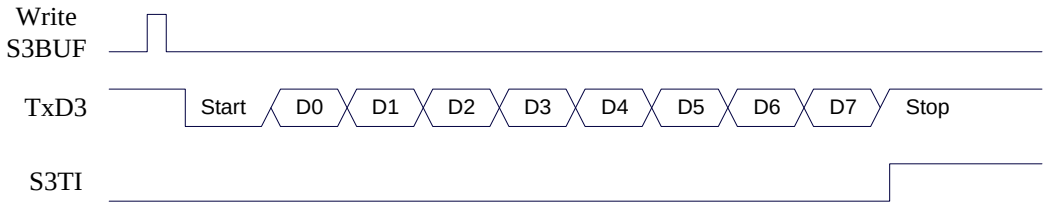
Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

S3BUF: It is used as the buffer in transmission and receiving for UART3. S3BUF is actually two buffers, reading buffer and writing buffer. Two operations correspond to two different registers, one is write-only register (writing buffer), the other is read-only register (reading buffer). The CPU reads serial receive buffer when reads S3BUF, and writes to the S3BUF will trigger the serial port to start sending data.

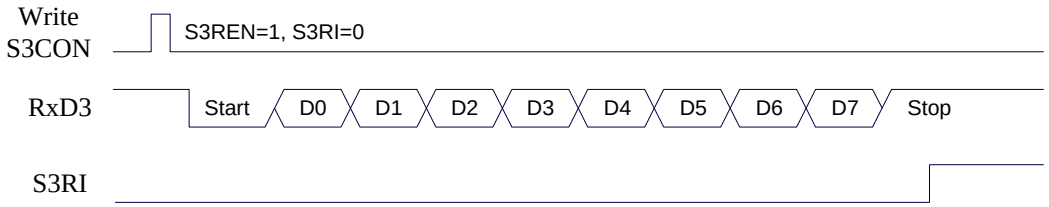
14.5.3 UART3 Mode 0

UART3 mode 0 is a 8-bit UART mode with variable baud rate, where a frame of data consists 10 bits: 1 start bit, 8 data bits (LSB first) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD3 is the

data transmitting pin, and RxD3 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART3 mode 0)



Receiving data (UART3 mode 0)

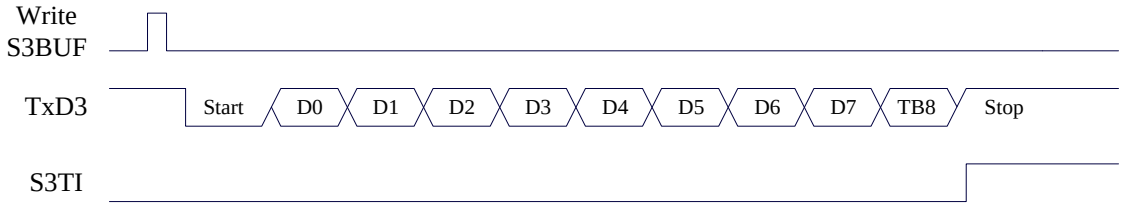
The baud rate of UART3 is variable. It is generated by T2 or T3. If the timer is in 1T mode (12x speed), the corresponding baud rate is increased by 12 times.

The baud rate of UART3 mode 0 is calculated as follows, where SYSclk is the system operating frequency.

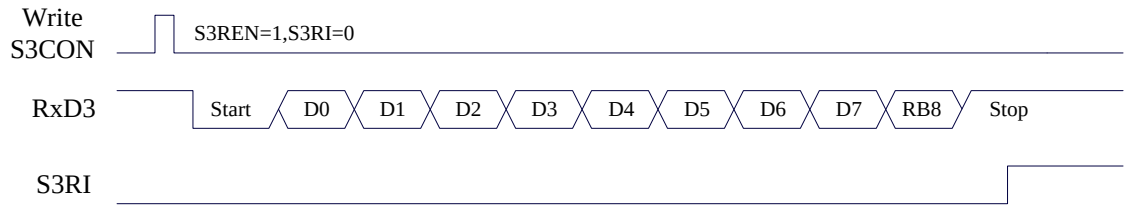
Timer selected	Speed of timer	Baud rate calculation formula
T2	1T	$\text{reload value of T2} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T2} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$
T3	1T	$\text{reload value of T3} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T3} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$

14.5.4 UART3 Mode 1

UART3 operating in mode 1 is a 9-bit data UART mode with variable baud rate. One frame data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9^{th} bit) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD3 is the data transmitting pin, and RxD3 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART3 mode 1)



Receiving data (UART3 mode 1)

The baud rate calculation formula of UART3 mode 1 is exactly the same as that of mode 0. Please refer to the mode 0 baud rate calculation formula.

14.6 UART4

14.6.1 UART4 control register

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

S4SM0: UART4 mode select bit.

S4SM0	UART4 mode	Function description
0	Mode 0	8-bit UART, whose baud-rate is variable
1	Mode 1	9-bit UART, whose baud-rate is variable

S4ST4: UART4 baud rate generator select bit.

0: Select T2 as the baud-rate generator of UART4.

1: Select T4 as the baud-rate generator of UART4.

S4SM2: UART4 multi-machine communication control bit. In mode 1, if the S4SM2 bit is 1 and the S4REN bit is 1, the receiver is in the address frame filter state. In this case, the received 9th bit (S4RB8) can be used to filter the address frame. If S4RB8 = 1, the frame is the address frame, address information can enter S4BUF, S4RI becomes 1, and then address can be compared with the slave address in the interrupt service routine. If S4RB8 = 0, it indicates that the frame is not an address frame and should be discarded and keep S4RI = 0. In mode 1, if the S4SM2 bit is 0 and the S4REN bit is 1, the receiver is in the address frame filter disabled state. Regardless of the received S4RB8 is 0 or 1, the information received can enter into the S4BUF, and make S4RI = 1. Here, S4RB8 is usually used as parity check bit. Mode 0 is non-multi-machine communication mode, where S4SM2 should be 0.

S4REN: Receive enable control bit.

0: disable UART4 receive data.

1: enable UART4 receive data.

S4TB8: S4TB8 is the 9th bit of datum to be sent when UART4 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S4RB8: S4RB8 is the 9th bit of datum recieved when UART4 is in mode 1, which is usually used as a parity check bit or an address frame / data frame flag. It can be set or cleared by software as required. In mode 0, this bit is not used.

S4TI: Transmit interrupt request flag of UART4. S4TI is set by the hardware automatically at the beginning of the stop bit transmittion and requests interrupt to the CPU. S4TI must be cleared by software after the interrupt is responded.

S4RI: Receive interrupt request flag of UART4. S4RI is set by hardware automatically at the middle of stop bit the serial port received, and requests interrupt to the CPU. After the interrupt is responded, S4RI must be cleared by software.

14.6.2 UART4 data register

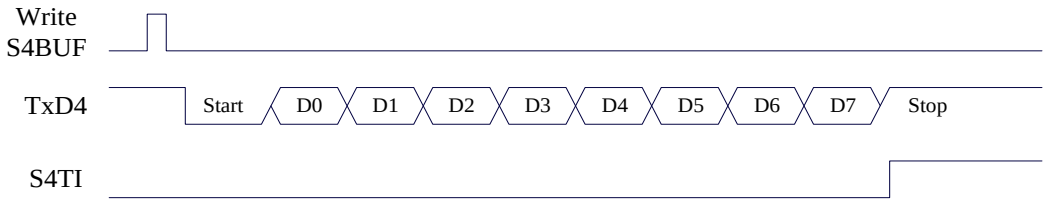
Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	85H								

S4BUF: It is used as the buffer in transmission and receiving for UART4. S4BUF is actually two buffers, reading buffer and writing buffer. Two operations correspond to two different registers, one is write-only register (writing buffer), the other is read-only register (reading buffer). The CPU reads serial receive buffer when reads S4BUF, and writes to the S4BUF will trigger the serial port to start sending data.

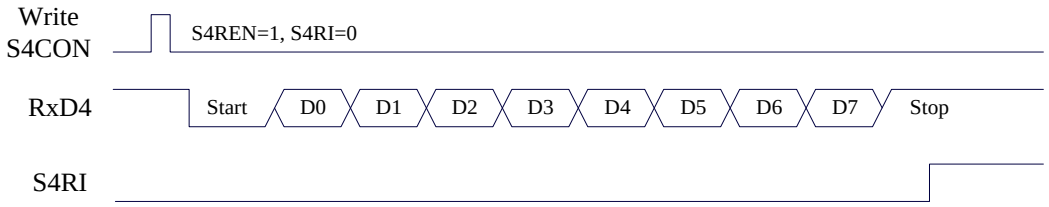
14.6.3 UART4 Mode 0

UART4 mode 0 is an 8-bit UART mode with variable baud rate, where a frame of data consists 10 bits: 1 start bit, 8 data bits (LSB first) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD4 is the

data transmitting pin, and RxD4 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART4 mode 0)



Receiving data (UART4 mode 0)

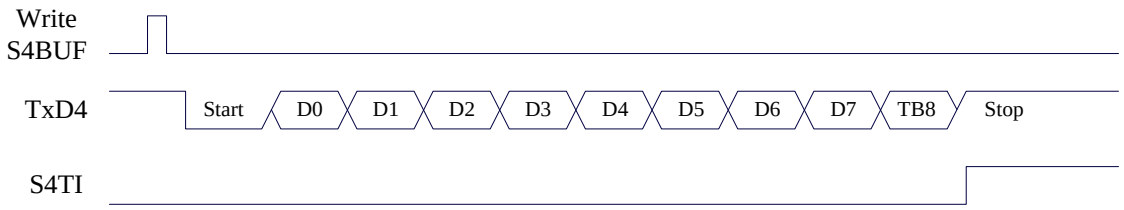
The baud rate of UART4 is variable. It is generated by T2 or T4. If the timer is in 1T mode (12x speed), the corresponding baud rate is increased by 12 times.

The baud rate of UART4 mode 0 is calculated as follows, where SYSclk is the system operating frequency.

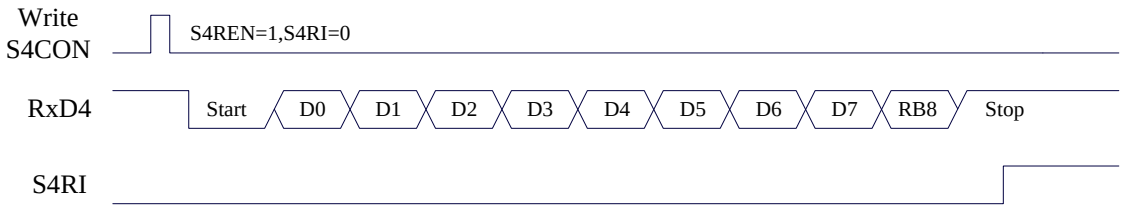
Timer selected	Speed of timer	Baud rate calculation formula
T2	1T	$\text{reload value of T2} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T2} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$
T4	1T	$\text{reload value of T4} = 65536 - \frac{\text{SYSclk}}{4 \times \text{baud rate}}$
	12T	$\text{reload value of T4} = 65536 - \frac{\text{SYSclk}}{12 \times 4 \times \text{baud rate}}$

14.6.4 UART4 Mode 1

UART4 operating in mode 1 is a 9-bit data UART mode with variable baud rate. One frame data consists of 11 bits: 1 start bit, 8 data bits (LSB first), 1 programmable bit (9^{th} bit) and 1 stop bit. The baud rate is variable, which can be set by the software as needed. TxD4 is the data transmitting pin, and RxD4 is the data receiving pin, the serial port is a full duplex receiver/transmitter.



Transmitting data (UART4 mode 1)



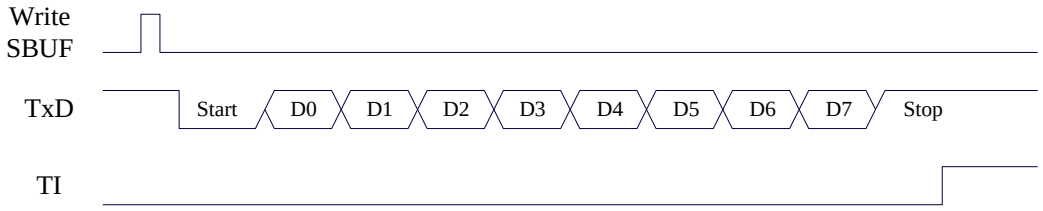
Receiving data (UART4 mode 1)

The baud rate calculation formula of UART4 mode 1 is exactly the same as that of mode 0. Please refer to the mode 0 baud rate calculation formula.

14.7 Precautions of UARTs

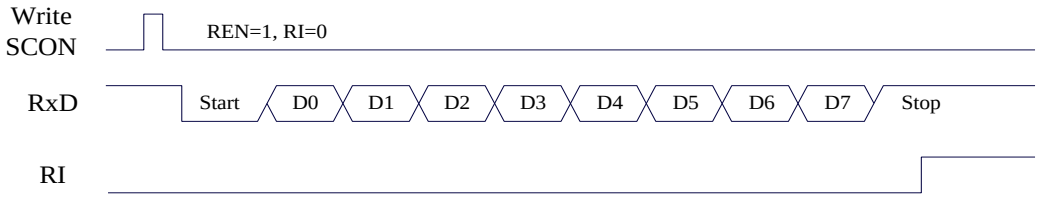
Regarding the UART interrupts requests, the following issues need to be noted. UART1, UART2, UART3, and UART4 are all similar, and serial port 1 is used as an example below.

In 8-bit data mode, TI interrupt request is generated after the entire stop bit is transmitted, as shown in the following figure:



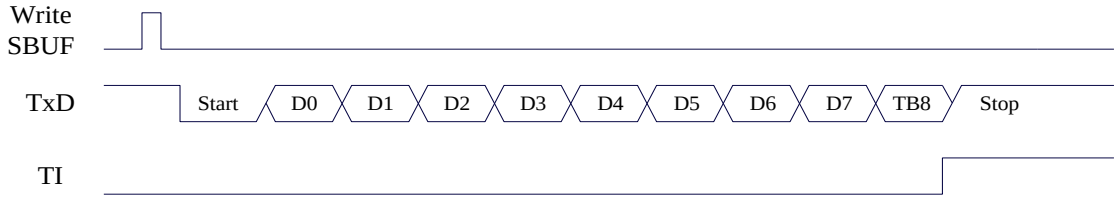
Transmission data (8-bit data)

In 8-bit data mode, RI interrupt request is generated after half of the stop bit is received, as shown in the following figure:



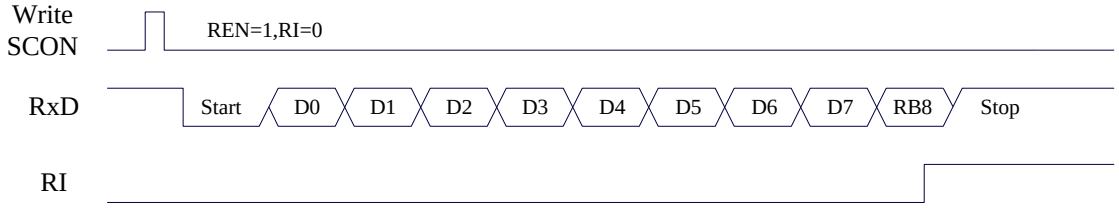
Receiving data (8-bit data)

In 9-bit data mode, TI interrupt request is generated after the entire 9th data bit is transmitted, as shown in the following figure:



Transmission data (9-bit data)

In 9-bit data mode, RI interrupt request is generated after receiving half of the 9th bit, as shown in the following figure:



Receiving data (9-bit data)

14.8 Example Routines

14.8.1 UART1 using T2 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz
```

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

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

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

sfr       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

```
;Operating frequency for test 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
```

14.8.2 UART1 using T1 (Mode 0) as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

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

#define     FOSC            11059200UL
```

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

sfr       AUXR        =   0x8e;

sfr       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

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH

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     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
```

14.8.3 UART1 using T1 (Mode 2) as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

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

#define    FOSC         11059200UL
#define    BRT          (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

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH

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     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
```

14.8.4 UART2 using T2 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

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

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

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

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

```
}

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

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR		DATA		8EH
T2H		DATA		0D6H
T2L		DATA		0D7H
S2CON		DATA		9AH
S2BUF		DATA		9BH
IE2		DATA		0AFH

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

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

```
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2_ISR

            ORG     0100H

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

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART2ISR_EXIT
            ANL     S2CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S2BUF
            INC     WPTR
UART2ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

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

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

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

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

        LCALL   UART2_INIT
        MOV     IE2,#01H
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART2_SENDSTR

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

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

        END
```

14.8.5 UART3 using T2 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

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

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
```

```
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

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

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

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

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

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

```
                JMP         UART3_SENDSTR
SEND3END:
                RET

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

                LCALL       UART3_INIT
                MOV         IE2,#08H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART3_SENDSTR

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

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

                END
```

14.8.6 UART3 using T3 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

```
sfr         T4T3M       =   0xd1;
sfr         T4L         =   0xd3;
sfr         T4H         =   0xd2;
sfr         T3L         =   0xd5;
sfr         T3H         =   0xd4;
sfr         T2L         =   0xd7;
sfr         T2H         =   0xd6;
sfr         S3CON       =   0xac;
sfr         S3BUF       =   0xad;
sfr         IE2         =   0xaf;

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

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

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

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

void Uart3Send(char dat)
{
    while (busy);
```

```
        busy = 1;
        S3BUF = dat;
}

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

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

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

T4T3M       DATA        0D1H
T4L         DATA        0D3H
T4H         DATA        0D2H
T3L         DATA        0D5H
T3H         DATA        0D4H
T2L         DATA        0D7H
T2H         DATA        0D6H
S3CON       DATA        0ACH
S3BUF       DATA        0ADH
IE2         DATA        0AFH

BUSY        BIT         20H.0
```

```
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

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

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

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

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

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

UART3_SEND:
```

```
                JB      BUSY,$
                SETB    BUSY
                MOV     S3BUF,A
                RET

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

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

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

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

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

                END
```

14.8.7 UART4 using T2 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

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

void Uart4Init()
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
```

```
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

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

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

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S4CON   DATA    84H
```

```
S4BUF       DATA    85H
IE2         DATA    0AFH

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

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

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

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

```
                MOV     RPTR,#00H
                RET

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

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

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

                LCALL   UART4_INIT
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

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

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

```
END
```

14.8.8 UART4 using T4 as baud rate generator

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

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

```
}

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

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

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

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

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

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

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

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
```

```
                INC         WPTR
UART4ISR_EXIT:
                POP         PSW
                POP         ACC
                RETI

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

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

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

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

                LCALL       UART4_INIT
                MOV         IE2,#10H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART4_SENDSTR

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

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

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

            END
```

14.8.9 Serial multi-MCUs communication

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

14.8.10 UART to LIN BUS

C language code

```
// Operating frequency for test is 22.1184MHz

/************* Function Description   **************
This routine is based on the experiment box 8 to program and test, whose main control chip is STC8H8K64U.
It is can be used for general reference when using STC8G and STC8H series chips.
Connect the LIN transceiver through the UART interface to realize the LIN bus signal transceiver test routine. UART1 is connected to the computer through the serial port tool.
UART2 is connected to an external LIN transceiver (TJA1020/1) and connected to the LIN bus.
Forward the data sent by the computer serial port to the LIN bus; forward the data received from the LIN bus to the computer serial port.
Default transmission rate: 9600 baud rate, switch baud rate before sending LIN data, send 13 dominant interval signals.
When downloading, select the clock 22.1184MHz (users can modify the frequency by themselves).
*****************************************/

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

#define    MAIN_Fosc        22118400L

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

sfr    AUXR    =    0x8E;
sfr    S2CON   =    0x9A;
sfr    S2BUF   =    0x9B;
sfr    TH2     =    0xD6;
sfr    TL2     =    0xD7;
sfr    IE2     =    0xAF;
sfr    INT_CLKO =   0x8F;
sfr    P_SW1   =    0xA2;
sfr    P_SW2   =    0xBA;

sfr    P4      =    0xC0;
sfr    P5      =    0xC8;
sfr    P6      =    0xE8;
sfr    P7      =    0xF8;
sfr    P1M1    =    0x91;
sfr    P1M0    =    0x92;
sfr    P0M1    =    0x93;
sfr    P0M0    =    0x94;
sfr    P2M1    =    0x95;
sfr    P2M0    =    0x96;
sfr    P3M1    =    0xB1;
sfr    P3M0    =    0xB2;
sfr    P4M1    =    0xB3;
sfr    P4M0    =    0xB4;
sfr    P5M1    =    0xC9;
sfr    P5M0    =    0xCA;
sfr    P6M1    =    0xCB;
sfr    P6M0    =    0xCC;
sfr    P7M1    =    0xE1;
sfr    P7M0    =    0xE2;
```

```
sbit    P00     =   P0^0;
sbit    P01     =   P0^1;
sbit    P02     =   P0^2;
sbit    P03     =   P0^3;
sbit    P04     =   P0^4;
sbit    P05     =   P0^5;
sbit    P06     =   P0^6;
sbit    P07     =   P0^7;
sbit    P10     =   P1^0;
sbit    P11     =   P1^1;
sbit    P12     =   P1^2;
sbit    P13     =   P1^3;
sbit    P14     =   P1^4;
sbit    P15     =   P1^5;
sbit    P16     =   P1^6;
sbit    P17     =   P1^7;
sbit    P20     =   P2^0;
sbit    P21     =   P2^1;
sbit    P22     =   P2^2;
sbit    P23     =   P2^3;
sbit    P24     =   P2^4;
sbit    P25     =   P2^5;
sbit    P26     =   P2^6;
sbit    P27     =   P2^7;
sbit    P30     =   P3^0;
sbit    P31     =   P3^1;
sbit    P32     =   P3^2;
sbit    P33     =   P3^3;
sbit    P34     =   P3^4;
sbit    P35     =   P3^5;
sbit    P36     =   P3^6;
sbit    P37     =   P3^7;
sbit    P40     =   P4^0;
sbit    P41     =   P4^1;
sbit    P42     =   P4^2;
sbit    P43     =   P4^3;
sbit    P44     =   P4^4;
sbit    P45     =   P4^5;
sbit    P46     =   P4^6;
sbit    P47     =   P4^7;
sbit    P50     =   P5^0;
sbit    P51     =   P5^1;
sbit    P52     =   P5^2;
sbit    P53     =   P5^3;
sbit    P54     =   P5^4;
sbit    P55     =   P5^5;
sbit    P56     =   P5^6;
sbit    P57     =   P5^7;

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

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

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

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

```
Signal

#define     UART1_BUF_LENGTH    32
#define     UART2_BUF_LENGTH    32

#define     LIN_ID          0x31

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

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

void UART1_config(u8 brt);
void UART2_config(u8 brt);
void PrintString1(u8 *puts);
void delay_ms(u8 ms);
void UART1_TxByte(u8 dat);
void UART2_TxByte(u8 dat);
void Lin_Send(u8 *puts);
void SetTimer2Baudraye(u16 dat);

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

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

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

    PrintString1("STC8A8K64D4 UART1 Test Programme!\r\n");      //UART1 sends a string

    while (1)
```

```
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)
        {
            if(--RX1_TimeOut == 0)                      // If it times out, the serial port reception ends
            {
                if(RX1_Cnt > 0)
                {
                    Lin_Send(RX1_Buffer);               // Send the data received by UART1 to the LIN bus
                }
                RX1_Cnt = 0;
            }
        }

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

//=========================================================================
// function: void delay_ms(unsigned      char ms)
// description: delay function
// parameters: ms, number of ms to delay, only support 1~255ms, and automatically adapt to master clock.
// return: none.
// version: VER1.0
// date: 2013-4-1
// remark:
//=========================================================================
void delay_ms(u8 ms)
{
    u16 i;
    do{
        i = MAIN_Fosc / 10000;
        while(--i);                                     //10T per loop
    }while(--ms);
}

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

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

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

	returnpid = id|P0|P1 ;

	return returnpid ;
}

//==========================================================================
// function: u8 LINCalcChecksum(u8 *dat)
// description: Calculate the checksum.
// parameters: The data transmitted by the data field.
// return: checksum.
// version: VER1.0
// date: 2020-12-2
// remark:
//==========================================================================
static u8 LINCalcChecksum(u8 *dat)
{
	u16 sum = 0;
	u8 i;

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

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

//==========================================================================
// function: void Lin_Send(u8 *puts)
```

```
// description: Send LIN bus message.
// parameters: The content of the data field to be sent.
// return: none.
// version: VER1.0
// date: 2020-12-2
// remark:
//========================================================================
void Lin_Send(u8 *puts)
{
    u8 i;

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

//========================================================================
// function: void UART1_TxByte(u8 dat)
// description: Send a byte.
// parameters: none.
// return: none..
// version: V1.0, 2014-6-30
//========================================================================
void UART1_TxByte(u8 dat)
{
    SBUF = dat;
    B_TX1_Busy = 1;
    while(B_TX1_Busy);
}

//========================================================================
// function: void UART2_TxByte(u8 dat)
// description: Send a byte.
// parameters: none.
// return: none.
// version: V1.0, 2014-6-30
//========================================================================
void UART2_TxByte(u8 dat)
{
    S2BUF = dat;
    B_TX2_Busy = 1;
    while(B_TX2_Busy);
}

//========================================================================
// function: void PrintString1(u8 *puts)
// description: UART1 sends a string function
// parameters: puts:            String pointer.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//========================================================================
```

```
void PrintString1(u8 *puts)
{
    for (; *puts != 0; puts++)                  // End with stop 0
    {
        SBUF = *puts;
        B_TX1_Busy = 1;
        while(B_TX1_Busy);
    }
}

//========================================================================
// function: void PrintString2(u8 *puts)
// description: UART1 sends a string function
// parameters: puts:            String pointer.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//========================================================================
//void PrintString2(u8 *puts)
//{
//    for (; *puts != 0;  puts++)               //End with stop 0
//    {
//        S2BUF = *puts;
//        B_TX2_Busy = 1;
//        while(B_TX2_Busy);
//    }
//}

//========================================================================
// function: SetTimer2Baudraye(u16 dat)
// description: Set Timer2 as baud rate generator.
// parameters: dat: Reload value of Timer2
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//========================================================================
void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                            //Timer stop
    AUXR &= ~(1<<3);                            //Timer2 set As Timer
    AUXR |= (1<<2);                             //Timer2 set as 1T mode
    TH2  = dat / 256;
    TL2 = dat % 256;
    IE2 &= ~(1<<2);                             //Disable interrupt
    AUXR |= (1<<4);                             //Timer run enable
}

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

```
void UART1_config(u8 brt)
{
    /********** select Timer2 as baud rate generator ****************/
    if(brt == 2)
    {
        AUXR |= 0x01;                       //S1 BRT Use Timer2;
        SetTimer2Baudraye(Baudrate1);
    }

    /********** select Timer1 as baud rate generator ****************/
    else
    {
        TR1  = 0;
        AUXR &= ~0x01;                      //S1 BRT Use Timer1;
        AUXR |= (1<<6);                     //Timer1 set as   1T mode
        TMOD &= ~(1<<6);                    //Timer1 set As   Timer
        TMOD &= ~0x30;                      //Timer1_16bitAutoReload;
        TH1 = (u8)(Baudrate1 / 256);
        TL1 = (u8)(Baudrate1 % 256);
        ET1 = 0;                            //diable interrupt
        INT_CLKO &= ~0x02;                  //does not output clock
        TR1 = 1;
    }
    /*************************************************/

    SCON = (SCON & 0x3f) | 0x40;            //UART1 mode:  0x00: Synchronous shift output,
                                            //             0x40: 8-bit data, variable baud rate,
                                            //             0x80: 9-bit data, fixed baud rate,
                                            //             0xc0: 9-bit data, variable baud rate
//  PS  = 1;                                //High priority
    ES  = 1;                                //enable interrupt
    REN = 1;                                //enable recieving
    P_SW1 &= 0x3f;
//  P_SW1 |= 0x80;                          //UART1switch to:    0x00: P3.0 P3.1,
                                            //                   0x40: P3.6 P3.7,
                                            //                   0x80: P1.6 P1.7,
                                            //                   0xC0: P4.3 P4.4

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

//========================================================================
// function: void UART2_config(u8 brt)
// description: UART2 initialization function
// parameters: brt: baud rate selected, 2: select Timer2 as baud rate generator, other values: not valid.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//========================================================================
void UART2_config(u8 brt)
{
    if(brt == 2)
    {
        SetTimer2Baudraye(Baudrate2);
```

```
        S2CON &= ~(1<<7);                                   //8-bit data, 1 start bit, 1 stop bit, no checking
        IE2 |= 1;                                           //enable interrupt
        S2CON |= (1<<4);                                    //enable recieving
        P_SW2 &= ~0x01;
//      P_SW2     |= 1;                                     //UART2 switch to: 0: P1.0/P1.1, 1: P4.6/P4.7

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

//========================================================================
// function: void UART1_int(void) interrupt UART1_VECTOR
// description: UART1 interrupt function。
// parameters: nine.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//========================================================================
void UART1_int (void) interrupt  4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUF_LENGTH) RX1_Cnt = 0;
        RX1_Buffer[RX1_Cnt] = SBUF;
        RX1_Cnt++;
        RX1_TimeOut = 5;
    }

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

//========================================================================
// function: void UART2_int(void) interrupt UART2_VECTOR
// description: UART2 interrupt function
// parameters: nine.
// return: none.
// version: VER1.0
// date: 2014-11-28
// remark:
//========================================================================
void UART2_int (void) interrupt  8
{
    if((S2CON & 1) != 0)
    {
        S2CON &= ~1;                                        //Clear Rx flag
        if(RX2_Cnt >= UART2_BUF_LENGTH) RX2_Cnt = 0;
        RX2_Buffer[RX2_Cnt] = S2BUF;
        RX2_Cnt++;
        RX2_TimeOut = 5;
    }
```

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

15 Comparator, Power-down Detection, Internal Reference Voltage

A comparator is integrated in STC8A8K64D4 series of microcontrollers. The positive terminal of the comparator can be P3.7, P5.0, P5.1 or ADC analog input, and the negative can be P3.6 or the REFV voltage of the internal BandGap after amplified. The application of multiple comparators can be realized through multiplexer and time division multiplexing.

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

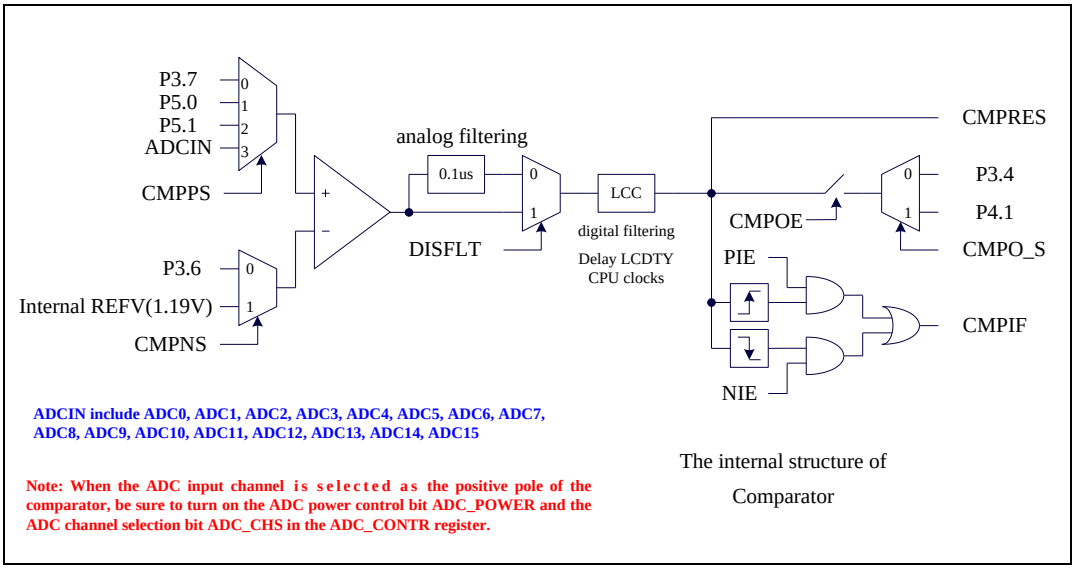
15.1 Comparator output pin switching

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

CMPO_S: Comparator output pin select bit

CMPO_S	CMPO
0	P3.4
1	P4.1

15.2 Internal Structure of Comparator



The internal structure of Comparator

15.3 Registers Related to Comparator

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	Comparator Control Register 1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES	0000,xx00
CMPCR2	Comparator Control Register 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000
CMPEXCFG	Comparator Extended Configuration Register	FEAEH	CHYS[1:0]		-	-	-	CMPNS	CMPPS[1:0]		00xx,x000

15.3.1 Comparator control register 1

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

CMPEN: Comparator enable bit

0: disable comparator

1: enable comparator

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

PIE: Comparator rising edge interrupt enable bit

0: disable comparator rising edge interrupt

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

NIE: Comparator falling edge interrupt enable bit

0: disable comparator falling edge interrupt

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

CMPOE: Comparator result output control bit

0: disable comparator result output.

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

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

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

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

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

15.3.2 Comparator control register 2

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

INVCMPO: Inverse comparator output control bit

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

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

DISFLT: Analog filtering function control bit

0: enable 0.1us analog filtering function

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

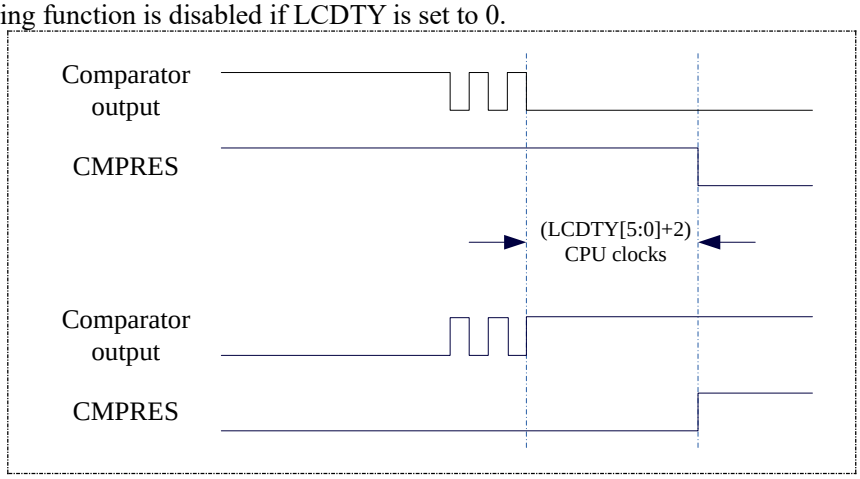
LCDTY[5:0]: Digital filtering function control bit

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

Note: When the digital filtering function is enabled, the actual waiting clock inside the chip needs to add two additional state machine switching times, that is, if LCDTY is set to 0, the digital filtering function is turned off;

if LCDTY is set to a non-zero value n (n =1~63), the actual digital filtering time is (n+2) system clocks

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



15.3.3 Comparator Extended Configuration Register (CMPEXCFG)

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

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

CHYS [1:0]	Comparator DC Hysteresis Input Selection
00	0mV
01	10mV
10	20mV
11	30mV

CMPNS：Comparator negative input selection

0：P3.6

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

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

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

15.4 Example Routines

15.4.1 Using Comparator (Interrupt Mode)

C language code

```
// Operating frequency for test is 11.0592MHz

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

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

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

sfr       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 CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                    // Clear interrupt flag
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;                     // Rising edge interrupt test port
    }
    else
    {
        P11 = !P11;                     // Falling edge interrupt test port
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

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

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

    EA = 1;

    while (1);
}
```

Assembly code

```
; Operating frequency for test is 11.0592MHz

CMPEXCFG    XDATA       0FEAEH

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

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

```
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            PUSH    ACC
            ANL     CMPCR1,#NOT 40H         ; Clear interrupt flag
            MOV     A,CMPCR1
            JB      ACC.0,RSING
FALLING:
            CPL     P1.0                    ; Falling edge interrupt test port
            POP     ACC
            RETI
RSING:
            CPL     P1.1                    ; Rising edge interrupt test port
            POP     ACC
            RETI

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

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

            MOV     CMPCR2,#00H
            ANL     CMPCR2,#NOT 80H         ; Comparator forward output
;           ORL     CMPCR2,#80H             ; Comparator inverted output
            ANL     CMPCR2,#NOT 40H         ; Enable 0.1us filtering
;           ORL     CMPCR2,#40H             ; Disable 0.1us filtering
;           ANL     CMPCR2,#NOT 3FH         ; Output comparator result directly
            ORL     CMPCR2,#10H             ; Output comparator result after 16 debounce clocks
            MOV     CMPCR1,#00H
```

```
                ORL     CMPCR1,#30H         ; Enable edge interrupt of comparator
;               ANL     CMPCR1,#NOT 20H     ; Disable comparator rising edge interrupt
;               ORL     CMPCR1,#20H         ; Enable comparator rising edge interrupt
;               ANL     CMPCR1,#NOT 10H     ; Disable comparator falling edge interrupt
;               ORL     CMPCR1,#10H         ; Enable comparator falling edge interrupt
                ANL     CMPCR1,#NOT 08H     ;P3.7 is CMP+ input pin
;               ORL     CMPCR1,#08H         ;ADC is CMP+ input pin
;               ANL     CMPCR1,#NOT 04H     ;internal 1.19Vreferrence voltage is CMP- input pin
                ORL     CMPCR1,#04H         ;P3.6 is CMP- input pin
;               ANL     CMPCR1,#NOT 02H     ; Disable comparator output
                ORL     CMPCR1,#02H         ; Enable Comparator output
                ORL     CMPCR1,#80H         ; Enable comparator module
                SETB    EA

                JMP     $

                END
```

15.4.2 Using Comparator (Polling Mode)

C language code

```
// Operating frequency for test is 11.0592MHz

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

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

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

sfr         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;

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

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

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

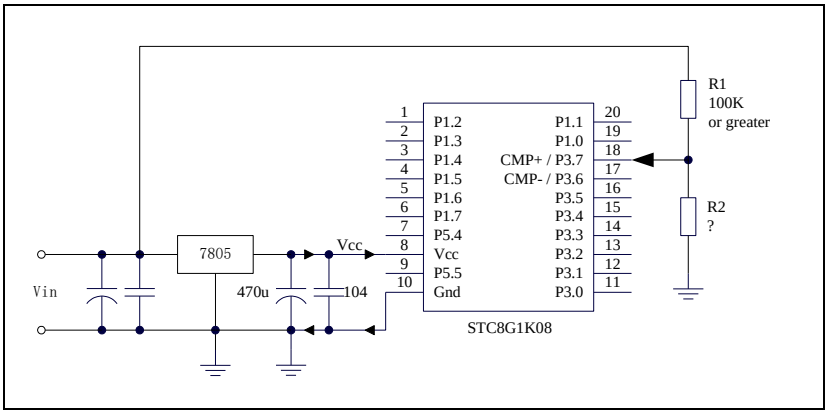
Assembly code

```
; Operating frequency for test is 11.0592MHz

CMPEXCFG    EQU         0FEFAH

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

P1M1        DATA        091H
P1M0        DATA        092H
P0M1        DATA        093H
P0M0        DATA        094H
P2M1        DATA        095H
P2M0        DATA        096H
```

```
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

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

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

LOOP:
            MOV     A,CMPCR1
            MOV     C,ACC.0
```

```
MOV     P1.0,C          ; Read comparator comparison result
JMP     LOOP

END
```

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

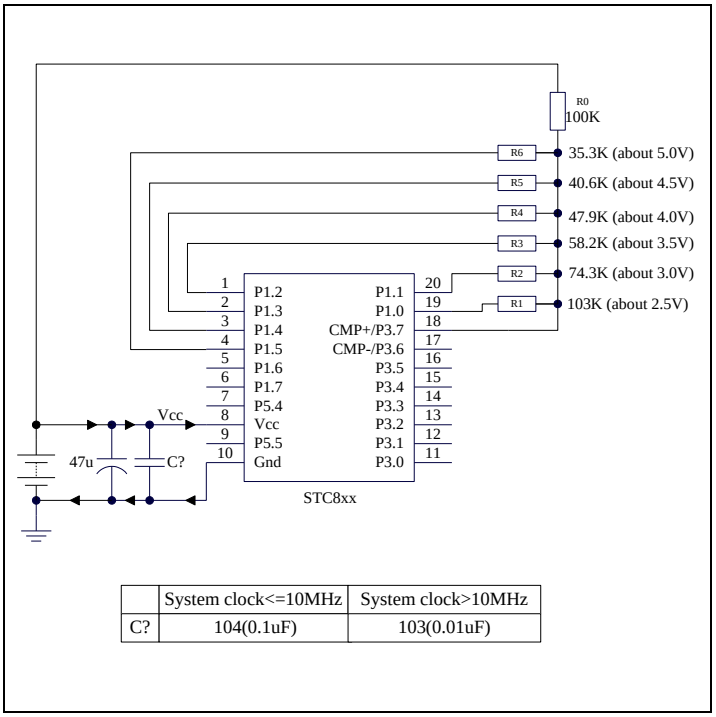


In the figure above, the resistor R1 and R2 divide the front-end voltage of the voltage regulator 7805. The divided voltage is used as the external input of the comparator CMP+ to compare with the internal reference voltage.

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

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

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



	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

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

The negative terminal of the comparator selects the internal reference voltage 1.19V, and the positive terminal selects the voltage value got from the voltage divided by a resistor as the input to the CMP+ pin.

In initialization, P1.5 ~ P1.0 are set to open-drain mode and output high. Firstly, P1.0 outputs a low level. At this time, if the VCC voltage is lower than 2.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 2.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 2.5V, then make the output of P1.0 high and the output of P1.1 low. At this time, if the VCC voltage is lower than 3.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 3.0V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 3.0V, then make the output of P1.1 high and the output of P1.2 low. At this time, if the VCC voltage is lower than 3.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 3.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 3.5V, then make the output of P1.2 high and the output of P1.3 low. At this time, if the VCC voltage is lower than 4.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 4.0V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 4.0V, then make the output of P1.3 high and the output of P1.4 low. At this time, if the VCC voltage is lower than 4.5V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 4.5V, the comparison value of the comparator is 1.

If you make sure that VCC is higher than 4.5V, then make the output of P1.4 high and the output of P1.5 low. At this time, if the VCC voltage is lower than 5.0V, the comparison value of the comparator is 0. Otherwise, if the VCC voltage is higher than 5.0V, the comparison value of the comparator is 1.

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

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

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

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

sfr       P2M0      =   0x96;
sfr       P2M1      =   0x95;

void delay ()
{
    char i;

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

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

    unsigned char v;

    P1M0 = 0x3f;                                    //P1.5 ~ P1.0 are initialized to open-drain mode
    P1M1 = 0x3f;
    P1 = 0xff;

    P_SW2 = 0x80;
    CMPEXCFG = 0x00;
    CMPEXCFG &= ~0x03;                              //P3.7 is CMP+ input pin
    CMPEXCFG |= 0x04;                               //internal 1.19V reference voltage is CMP- input pin
    P_SW2 = 0x00;
```

```
    CMPCR2 = 0x10;                          //Output comparator result after 16 debounce clocks
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x02;                        //Disable comparator output
    CMPCR1 |= 0x80;                         //Enable comparator module

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

CMPEXCFG    EQU     0FEFAH

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

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M0        DATA    096H
P2M1        DATA    095H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
```

```
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

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

            MOV     P_SW,#80H
            MOV     DPTR,#CMPEXCFG
            CLR     A
            ANL     A,#NOT 03H          ;P3.7 is CMP+ input pin
            ORL     A,#04H              ;internal 1.19V reference voltage is CMP- input pin
            MOVX    @DPTR,A
            MOV     P_SW,#00H

            MOV     P1M0,#00111111B     ;P1.5 ~ P1.0 are initialized to open-drain mode
            MOV     P1M1,#00111111B
            MOV     P1,#0FFH
            MOV     CMPCR2,#10H         ;Output comparator result after 16 debounce clocks
            MOV     CMPCR1,#00H
            ANL     CMPCR1,#NOT 02H     ;Disable comparator output
            ORL     CMPCR1,#80H         ;Enable comparator module
LOOP:
            MOV     R0,#00000000B       ;Voltage <2.5V
            MOV     P1,#11111110B       ;P1.0 outputs 0
            CALL    DELAY
            MOV     A,CMPCR1
            JNB     ACC.0,SKIP
            MOV     R0,#00000001B       ;Voltage>2.5V
            MOV     P1,#11111101B       ;P1.1 outputs 0
            CALL    DELAY
            MOV     A,CMPCR1
            JNB     ACC.0,SKIP
            MOV     R0,#00000011B       ;Voltage>3.0V
            MOV     P1,#11111011B       ;P1.2 outputs 0
            CALL    DELAY
            MOV     A,CMPCR1
            JNB     ACC.0,SKIP
            MOV     R0,#00000111B       ;Voltage>3.5V
            MOV     P1,#11110111B       ;P1.3 outputs 0
            CALL    DELAY
```

```
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
                MOV     R0,#00001111B           ;Voltage>4.0V
                MOV     P1,#11101111B           ;P1.4 outputs 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
                MOV     R0,#00011111B           ;Voltage>4.5V
                MOV     P1,#11011111B           ;P1.5 outputs 0
                CALL    DELAY
                MOV     A,CMPCR1
                JNB     ACC.0,SKIP
                MOV     R0,#00111111B           ;Voltage>5.0V
SKIP:
                MOV     P2,#11111111B
                MOV     A,R0
                CPL     A
                MOV     P0,A                    ;P0.5 ~ P0.0 display voltage
                JMP     LOOP

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

                END
```

16 IAP/EEPROM/DATA-FLASH

Large capacity of internal EEPROM is integrated in STC8A8K64D4 series of microcontrollers. The internal Data Flash can be used as EEPROM by using ISP / IAP technology. And it can be repeatedly erased more than 100,000 times. EEPROM can be divided into several sectors, each sector contains 512 bytes.

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

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

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

16.1 EEPROM operation time

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

The time required for EEPROM operation is automatically controlled by the hardware, and the user only needs to set the IAP_TPS register correctly.
IAP_TPS=System operating frequency/1000000 (the decimal part is rounded to the nearest whole number)
For example: the operating frequency of the system is 12MHz, then IAP_TPS is set to 12
Another example: the system operating frequency is 22.1184MHz, then IAP_TPS is set to 22
Another example: the system operating frequency is 5.5296MHz, then IAP_TPS is set to 6

16.2 Registers Related to EEPROM

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

16.2.1 EEPROM data register (IAP_DATA)

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

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

16.2.2 EEPROM address registers (IAP_ADDR)

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

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

16.2.3 EEPROM command register (IAP_CMD)

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

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

00: No operation.

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

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

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

16.2.4 EEPROM trigger register (IAP_TRIG)

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

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

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

16.2.5 EEPROM control register (IAP_CONTR)

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

IAPEN: EEPROM operation enable control bit.

0: disable EEPROM operation.

1: Enable EEPROM operation.

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

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

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

SWRST: Software reset control bit.

0: No operation.

1: Generate software reset.

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

0: EEPROM operation is right.

1: EEPROMoperation fails.

16.2.6 EEPROM erase wait time control register (IAP_TPS)

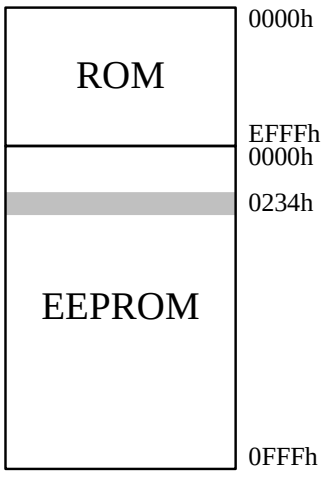
Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

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

16.3 EEPROM Size and Address

There is EEPROM for saving user data in all STC8A8K64D4 series of microcontrollers. There are three operation modes for the internal EEPROM: reading, writing, and erasing. The erasing operation is performed in sectors. Each sector has 512 bytes. That is, each time an erasing command will erase a sector when it executes. The reading and writing operations are in bytes, that is, only one byte can be read or written each time when a reading or writing command is executed.

There are two ways to access the internal EEPROM of STC8A8K64D4 series microcontrollers: IAP mode and MOVC mode. The IAP mode can perform reading, writing and erasing operations on the EEPROM. MOVC can only perform reading operations on the EEPROM, cannot perform writing and erasing operations. Regardless of whether IAP or MOVC is used to access the EEPROM, the correct target address must be set firstly. In IAP mode, the target address is the same as the actual physical address of the EEPROM. Both of them are accessed from address 0000H. However, when using MOVC instruction to read EEPROM data, the target address must be the actual physical address of the EEPROM plus a program size offset. STC8A8K64D4 is used as an example to describe the target address in detail as following:



STC8A8K60D4

The program space of STC8A8K60D4 is 60K bytes (0000h ~ EFFFh), and the EEPROM space is 4K bytes (0000h ~ 0FFFh). When you need to read, write, and erase the unit with EEPROM physical address 0234h, if you use IAP to access, set the target address to 0234h, that is, IAP_ADDRH is set to 02h, IAP_ADDRL is set to 34h, and then the corresponding trigger command can be set and the 0234h can be operated correctly. However, if the 0234h unit of the

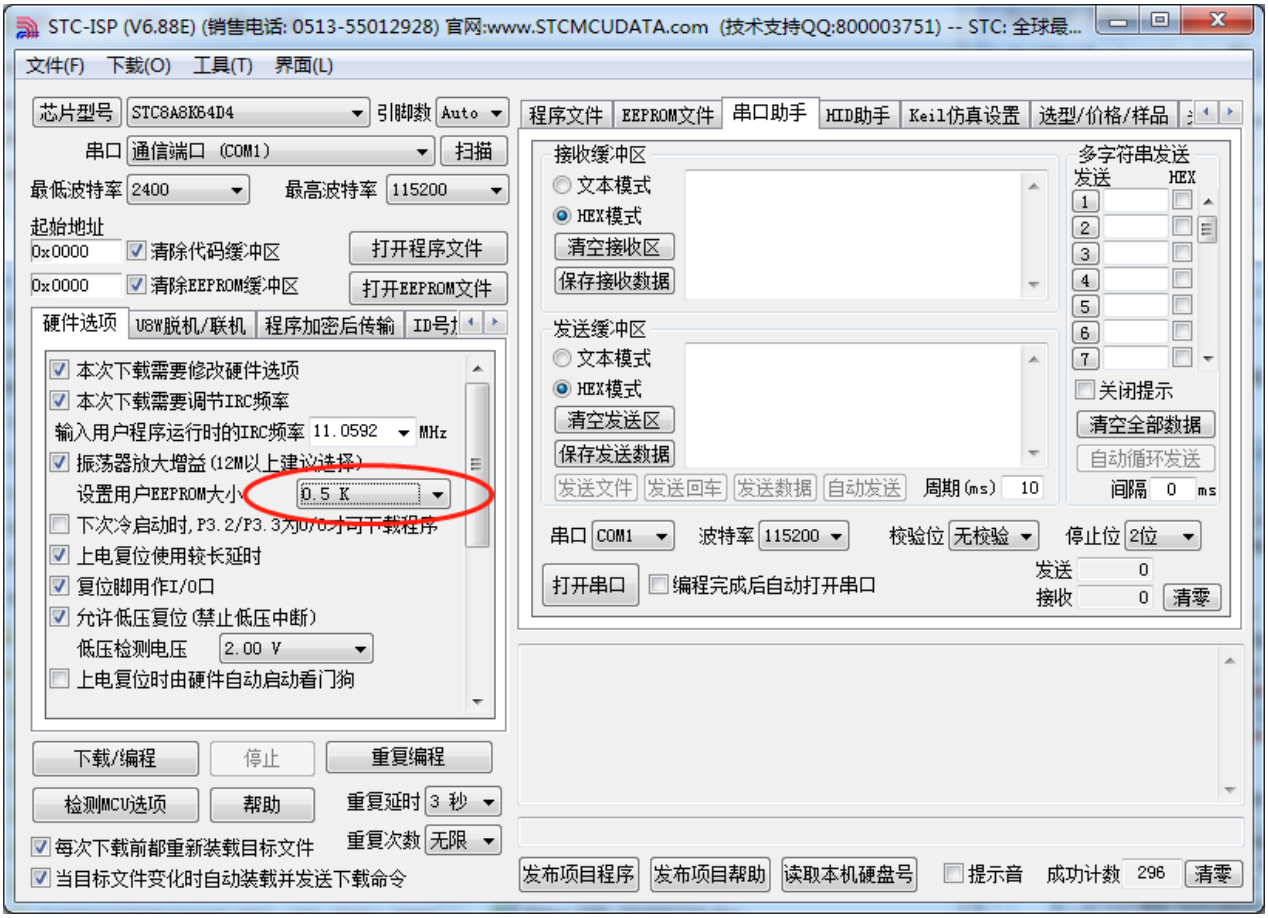
EEPROM is read by MOVC, the flash program memory (ROM) space must be added in addition to F000h. That is, the DPTR must be set to F234h before the MOVC instruction can be used for reading.

Note: Because the erasing is performed in 512-byte units, the lower 9 bits of the target address set when performing the erasing operation are meaningless. For example, if the target address is set to 0234H, 0200H, 0300H or 03FFH when executing the erasing command, the final erasing operation is the same, and the 512 bytes of 0200H ~ 03FFH are erased.

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

Model	Size	Sectors	Reading, writing, erasing in IAP mode		Reding using MOVC	
			Beginning adress	End address	Beginning adress	End address
STC8A8K16D4	48K	96	0000h	BFFFh	4000h	FFFFh
STC8A8K32D4	32K	64	0000h	7FFFh	8000h	FFFFh
STC8A8K60D4	4K	8	0000h	0FFFh	F000h	FFFFh
STC8A8K64D4	User defined[1]					

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



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

For example, the size of FLASH in STC8A8K64D4 is 64K. If user wants to allocate 4K of it as EEPROM, the physical address of EEPROM is the last 4K of 64K, and the physical address is F000h ~ FFFFh. Of course, if the user uses IAP to access, the target address still starts from 0000h and ends at 0FFFh. When using MOVC to read, the target address is in the range from F000h to FFFFh.

16.4 Example Routines

16.4.1 EEPROM Basic Operation

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

char IapRead(int addr)
{
    char dat;

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

    return dat;
```

```
}

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

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

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

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
```

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

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

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

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

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

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

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

            MOV     DPTR,#0400H
            LCALL   IAP_ERASE
            MOV     DPTR,#0400H
            LCALL   IAP_READ
            MOV     P0,A                    ;P0=0FFH
            MOV     DPTR,#0400H
            MOV     A,#12H
            LCALL   IAP_PROGRAM
            MOV     DPTR,#0400H
            LCALL   IAP_READ
            MOV     P1,A                    ;P1=12H

            SJMP    $

            END
```

16.4.2 Read EEPROM using MOVC

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

#define     IAP_OFFSET      0xF000H             // STC8A8K60S4

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

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

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

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

```
    _nop_();                                    //
    IapIdle();                                  //Disable IAP function
}

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

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

IAP_OFFSET  EQU     0F000H              ; STC8A8K60S4

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
```

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

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

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

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

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

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     DPTR,#0400H
        LCALL   IAP_ERASE
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P0,A                    ;P0=0FFH
        MOV     DPTR,#0400H
        MOV     A,#12H
        LCALL   IAP_PROGRAM
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P1,A                    ;P1=12H

        SJMP    $

        END
```

16.4.3 Send Out the Data in EEPROM Using UART

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

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

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

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

char IapRead(int addr)
{
    char dat;

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

    return dat;
}

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

void IapErase(int addr)
{
```

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

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

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

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

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
```

```
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

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

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

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

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

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

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

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

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

        SJMP    $

        END
```

16.4.4 UART1 reads and writes EEPROM - Read using MOVC

C language code (main.c)

```
//Operating frequency for test is 11.0592MHz

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

/************* Function description of this program    **************
STC8G family MCUs EEPROM general test program.
```

```
Please do not modify the program firstly, and download the "UART-EEPROM.hex" directly to test "02-UART 1 reads and writes EEPROM- using MOVC to read". Select the frequency 11.0592MHZ when downloading.
PC serial port setting: baud rate 115200,8,n,1.
Do sector erase, write 64 bytes, and read 64 bytes of EEPROM.

Command example:
E 0   Perform sector erasing operation on EEPROM, E means erasing, the number 0 is 0 sector (decimal, 0~126, see the specific IC).
W 0  Write operation to EEPROM, W means write, number 0 is 0 sector (decimal, 0~126, see the specific IC). Write 64 bytes continuously from the start address of the sector.
R 0  Perform IAP read operation on EEPROM, R means read out, the number 0 is 0 sector (decimal, 0~126, see the specific IC). Read 64 bytes continuously from the start address of the sector.
M 0  Perform MOVC read operation on EEPROM (operation address is sector*512+offset address), number 0 is sector 0 (decimal, 0~126, see the specific IC). Read 64 bytes continuously from the start address of the sector .


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

Date: 2019-6-10
*****************************************/

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

#define    Baudrate1                115200L
#define    UART1_BUF_LENGTH         10
#define    EEADDR_OFFSET            (8 * 1024)      //Define the offset added when EEPROM is accessed using MOVC
                                                    // Equal to the size of the FLASH ROM.
                                                    // For IAP or IRC at the beginning, the offset must be 0
#define    TimeOutSet1              5

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

/************* local variable declaration    **************/
u8    xdatatmp[70];
u8    xdataRX1_Buffer[UART1_BUF_LENGTH];
u8    RX1_Cnt;
u8    RX1_TimeOut;
bit   B_TX1_Busy;

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

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

/****************************************************/

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

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

    address = 0;
    if(RX1_Cnt <  3)return 65535;                   //error
    if(RX1_Cnt <= 5)                                //Within 5 bytes is sector operation, decimal,
                                                    //Command supported:       E 0, E 12, E 120
                                                    //                         W 0, W 12, W 120
                                                    //                         R 0, R 12, R 120
    {
        for(i=2; i<RX1_Cnt; i++)
        {
            if(CheckData(RX1_Buffer[i]) > 9)
                return 65535;                       //error
            address = address * 10 + CheckData(RX1_Buffer[i]);
        }
        if(address < 124)                           //Limited to sectors 0~123
        {
            address <<= 9;
            return (address);
        }
    }
    else if(RX1_Cnt == 8)                           //8 bytes direct address operation, hex,
                                                    //Command supported: E 0x1234, W 0x12b3, R 0x0A00
    {
        if((RX1_Buffer[2] == '0') && ((RX1_Buffer[3] == 'x') || (RX1_Buffer[3] == 'X')))
        {
            for(i=4; i<8; i++)
            {
                if(CheckData(RX1_Buffer[i]) > 0x0F)
                    return 65535;                   //error
                address = (address << 4) + CheckData(RX1_Buffer[i]);
            }
            if(address < 63488)
                return (address);                   //Limited to sectors 0~123
        }
    }

     return     65535;                              //error
}

//==========================================================================
// Function: void  delay_ms(u8 ms)
// Description: delay function
// Parameters: ms, the number of ms to delay, here only supports 1~255ms. Automatically adapt to the main clock.
// Return: none.
// Version: VER1.0
// Date: 2013-4-1
// Remark:
//==========================================================================
void  delay_ms(u8 ms)
{
    u16 i;
    do
    {
        i = MAIN_Fosc / 10000;
```

```
            while(--i)   ;
        }while(--ms);
}

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

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

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

        UART1_config();                                 // select baud rate, 2: Use Timer2 as baud rate generator,
                                                        //Other values: Use Timer1 as baud rate generator,
        EA = 1;                                         //Enable CPU interrupt

        PrintString1("STC8 familyMCU 用串口 1 测试 EEPROM 程序!\r\n");   //UART1send a string

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

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

                            else if(RX1_Buffer[0] == 'R')           //PC requests to return 64 bytes of EEPROM data
                            {
                                PrintString1("IAPData be read 如下：\r\n");
                                EEPROM_read_n(addr,tmp,64);
```

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

                    RX1_Cnt = 0;
                }
            }
        }
    }
/*********************************************/

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

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

//========================================================================
// Function: voidUART1_config(void)
// Description: UART1 initialization function.
// Parameters:none.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
```

```
// Remark:
//=========================================================================
void  UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01;                          //S1 BRT Use Timer1;
    AUXR |=  (1<<6);                        //Timer1 set as 1T mode
    TMOD &= ~(1<<6);                        //Timer1 set As Timer
    TMOD &= ~0x30;                          //Timer1_16bitAutoReload;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0;                                // Diable Timer1 interrupt
    INT_CLKO &= ~0x02;                      // Timer1 does not output high-speed clock
    TR1  = 1;                               // Start Timer1

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

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

    B_TX1_Busy = 0;
    RX1_Cnt = 0;
}

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

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

C language code (EEPROM.c)

```
//Operating frequency for test is 11.0592MHz

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

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

//=========================================================================
// Function: voidISP_Disable(void)
// Description: Disable access ISP/IAP.
// Parameters:non.
// Return: non.
// Version: V1.0, 2012-10-22
//=========================================================================
void  DisableEEPROM(void)
{
      ISP_CONTR = 0;                              //Disable ISP/IAP operation
      IAP_TPS   = 0;
      ISP_CMD   = 0;                        //Remove ISP/IAP commands
      ISP_TRIG  = 0;                        //Prevent false triggering of ISP/IAP commands
      ISP_ADDRH = 0xff;                           //Clear address high byte
      ISP_ADDRL = 0xff;                           //Clear address low byte, point to non-EEPROM area to prevent misoperation
}

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

```
            *DataAddress = ISP_DATA;                    //Data be read
            EE_address++;
            DataAddress++;
        }while(--number);

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

/********************* Sector Erase Function *****************/
//=========================================================================
// Function: void EEPROM_SectorErase(u16 EE_address)
// Description: Erase the EEPROM sector at the specified address.
// Parameters:EE_address:  The address of the sector EEPROM to be erased.
// Return: non.
// Version: V1.0, 2013-5-10
//=========================================================================
void EEPROM_SectorErase(u16 EE_address)
{
        EA = 0;                                         //Disable interrupts
                                                        //Only sector erase, no byte erase, 512 bytes per sector.
                                                        //Any byte address in a sector is sector address.
        ISP_ADDRH = EE_address / 256;                   //Send the high byte of the sector address (the address needs to be re-sent
when the address needs to be changed)
        ISP_ADDRL = EE_address % 256;                   //Send the low byte of the sector address
        ISP_CONTR = ISP_EN;                             //Allow ISP/IAP operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L);           //Working frequency setting
        ISP_ERASE();                                    //Send sector erase command. When the command does not need to be
changed, it is not necessary to send the command again
        ISP_TRIG();
        _nop_();
        _nop_();
        _nop_();
        DisableEEPROM();
        EA = 1;                                         //re-enable interrupt
}

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

        ISP_CONTR = ISP_EN;                             //Allow ISP/IAP operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L);           //Working frequency setting
        ISP_WRITE();                                    //Send byte write command. When the command does not need to be
changed, no need to send the command again
        do
        {
            ISP_ADDRH = EE_address / 256;               //Send the high byte of the address (the address needs to be re-sent when
the address needs to be changed)
```

```
            ISP_ADDRL = EE_address % 256;           //Send the low byte of the address
            ISP_DATA  = *DataAddress;               //Send data to ISP_DATA, and only need to send it again when the data
changes.
            ISP_TRIG();
            _nop_();
            _nop_();
            _nop_();
            EE_address++;
            DataAddress++;
        }while(--number);

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

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

C language code (main.c)

```
//Operating frequency for test is 11.0592MHz

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

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

Each writting writes 3 sectors, that is, redundant backup.
Write a record in each sector, after the writing is completed, read the saved data and check value and compare it with the source data and check value, and return the result (correct or wrong) from UART1 (P3.0 P3.1) .
Each record is self-checked, 64-byte data, 2-byte check value, check value = 64 bytes data's cumulative sum ^ 0x5555. ^0x5555 is to ensure that the written 66 data are not all 0.
If there is a sector error, the data of the correct sector will be written to the wrong sector, and if all three sectors are wrong, the default value will be written.
A password needs to be set before erasing, writing, and reading operations. If the password is incorrect, the operation will be exited, and the password will be cleared each time when an exit operation is performed.

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

PC serial port setting: baud rate 115200,8,n,1.
Do sector erase, write 64 bytes, and read 64 bytes of EEPROM.

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

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

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

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

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

Date: 2021-11-5
*****************************************/

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

#define         Baudrate1               115200L

/************* local constant declaration    **************/
u8    code  T_StringD[]={"去年今日此门中，人面桃花相映红。人面不知何处去，桃花依旧笑春风。"};
u8    code  T_StringW[]={"横看成岭侧成峰，远近高低各不同。不识庐山真面目，只缘身在此山中。"};


/************* local variable declaration    **************/
u8    xdatatmp[70];                             //General data
u8    xdataSaveTmp[70];                         // array to write

bit   B_TX1_Busy;
u8    cmd;                                      // single character command of UART


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


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

/*************  Read the EEPROM record, and verify, return the verification result, 0 is correct, 1 is wrong*****************/
u8    ReadRecord(u16 addr)
{
      u8    i;
      u16   ChckSum;                            //Calculated cumulative sum
      u16   j;                                  //Accumulated sum of reading

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

/*************  Write the EEPROM record, and verify, return the verification result, 0 is correct, 1 is wrong *****************/
u8    SaveRecord(u16 addr)
{
      u8    i;
      u16   ChckSum;                            //Calculated cumulative sum

      for(ChckSum=0, i=0; i<64; i++)
```

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

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

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

/********************** main function ************************/
void main(void)
{
    u8    i;
    u8    status;                               //Statuc

    UART1_config();                             // select baud rate, 2: Use Timer2 as baud rate generator,
                                                //Other values: Use Timer1 as baud rate generator,
    EA = 1;                                     //Enable CPU interrupt

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

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

    if(status == 0)                             //If all sectors are wrong, write default value
    {
```

```
        for(i=0; i<64; i++)
            SaveTmp[i] = T_StringD[i];                  //Read the default value
    }
    else PrintString1("上电读取 3 个扇区数据均正确!\r\n");     //UART1 send a string prompt

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


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

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

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

```
                if(SaveRecord(0x0400) == 0)
                    PrintString1("write a sector2 正确!\r\n");          // Writing record 2 sector is correct
                else
                    PrintString1("write a sector2 错误!\r\n");          // Writing record 2 sector is wrong
            }

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

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

                if(ReadRecord(0x0400) == 0)                             // Read data of sector 2
                {
                    PrintString1("读出扇区 2 的数据如下：\r\n");
                    for(i=0; i<64; i++)
                        TX1_write2buff(tmp[i]);                         // Return data to UART
                    TX1_write2buff(0x0d);                               // carriage return and line feed
                    TX1_write2buff(0x0a);
                }
                else   PrintString1("读出扇区 2 的数据错误!\r\n");
            }
            else   PrintString1("命令错误!\r\n");
            cmd = 0;
        }
    }
}
/*********************************************/

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

//========================================================================
// Function: void PrintString1(u8 *puts)
// Description: UART1 send string function.
// Parameters:puts:  String pointer.
// Return: none.
```

```
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//========================================================================
void PrintString1(u8 *puts)                    //send a string
{
    for (; *puts != 0;     puts++)             //End with stop 0
    {
        TX1_write2buff(*puts);
    }
}

//========================================================================
// Function: voidUART1_config(void)
// Description: UART1 initialization function.
// Parameters:none.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//========================================================================
void  UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01;                             //S1 BRT Use Timer1;
    AUXR |=  (1<<6);                           //Timer1 set as 1T mode
    TMOD &= ~(1<<6);                           //Timer1 set As Timer
    TMOD &= ~0x30;                             //Timer1_16bitAutoReload;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0;                                   // Disable Timer1 interrupt
    INT_CLKO &= ~0x02;                         // Timer1 does not output high-speed clock
    TR1  = 1;                                  // Start Timer1

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

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

    B_TX1_Busy = 0;
}

//========================================================================
// Function: void UART1_int (void) interrupt UART1_VECTOR
// Description: UART1 interrupt function
// Parameters:nine.
// Return: none.
// Version: VER1.0
// Date: 2014-11-28
// Remark:
//========================================================================
```

```
void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        cmd = SBUF;
    }

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

C language code (EEPROM.c)

```
//Operating frequency for test is 11.0592MHz

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

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

u32       PassWord;                                 // Password required for erasing and writing

//=========================================================================
// Function: voidISP_Disable(void)
// Description: Disable access ISP/IAP.
// Parameters:non.
// Return: non.
// Version: V1.0, 2012-10-22
//=========================================================================
void  DisableEEPROM(void)
{
    ISP_CONTR = 0;                                  //Disable ISP/IAP operation
    IAP_TPS   = 0;
    ISP_CMD   = 0;                                  //Remove ISP/IAP commands
    ISP_TRIG  = 0;                                  //Prevent false triggering of ISP/IAP commands
    ISP_ADDRH = 0xff;                               //Clear address high byte
    ISP_ADDRL = 0xff;                               //Clear address low byte, point to non-EEPROM area to prevent misoperation
}

//=========================================================================
// Function: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// Description: Read n bytes from the specified EEPROM first address and put them in the specified buffer.
// Parameters:EE_address:  The first address of the EEPROM to read.
//      DataAddress: The first address of the data buffer.
//      number:      The length of bytes to read.
// Return: non.
// Version: V1.0, 2012-10-22
//=========================================================================
void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
{
    if(PassWord == D_PASSWORD)                      // Only the correct password is allowed to operate the EEPROM
    {
        EA = 0;                                     //Disable interrupts
```

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

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

/******************** Sector Erase Function *****************/
//=========================================================================
// Function: void EEPROM_SectorErase(u16 EE_address)
// Description: Erase the EEPROM sector at the specified address.
// Parameters:EE_address:  The address of the sector EEPROM to be erased.
// Return: non.
// Version: V1.0, 2013-5-10
//=========================================================================
void EEPROM_SectorErase(u16 EE_address)
{
        if(PassWord == D_PASSWORD)              // Only the password is correct, the EEPROM will be operated
        {
            EA = 0;                             //Disable interrupts
                                                //Only sector erase, no byte erase, 512 bytes per sector.
                                                //Any byte address in a sector is sector address.
            ISP_ADDRH = EE_address / 256;       //Send the high byte of the sector address (the address needs to be re-sent
when the address needs to be changed)
            ISP_ADDRL = EE_address % 256;       //Send the low byte of the sector address
            ISP_CONTR = ISP_EN;                 //Allow ISP/IAP operation
            IAP_TPS = (u8)(MAIN_Fosc / 1000000L);   //Working frequency setting
            ISP_ERASE();                        //Send sector erase command. When the command does not need to be
changed, it is not necessary to send the command again
                if(PassWord == D_PASSWORD)      // If the password is correct, trigger the operation
                {
                    ISP_TRIG = 0x5A;            //Send 5AH firstly, then send A5H to the ISP/IAP trigger register,
                                                //Do this every time
                    ISP_TRIG = 0xA5;            //After sending A5H, the ISP/IAP command is triggered to start
```

```
immediately
        }                                   //The CPU waits for the IAP to complete before continuing to execute
the program.
        _nop_();
        _nop_();
        _nop_();
        DisableEEPROM();
        EA = 1;                             //re-enable interrupt
    }
    PassWord = 0;                           //Clear password
}

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

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

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

```
}
```

17 ADC, Internal Reference Voltage

A 12-bit high-speed Analog to Digital Converter is integrated in STC8A8K64D4 family of microcontrollers. The system frequency is divided by 2 and then divided again by the user-set division ratio as the clock frequency of the ADC. The range of ADC clock frequency is SYSclk/2/1 ~ SYSclk/2/16.

The fastest ADC speed of STC8A8K64D4 series: 12-bit ADC is 800K (800,000 ADC conversions per second).

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

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

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

17.1 Registers Related to ADC

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC control register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC Result High Byte	BDH									0000,0000
ADC_RESL	ADC Result Low Byte	BEH									0000,0000
ADCCFG	ADC Configuration Register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
ADCTIM	ADC Timing Control Register	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010,1010
ADCEXCFG	ADC Extended configuration registers	FEADH	-	-	ADCETRS[1:0]		-	CVTIMESEL[2:0]			xx00,x000

17.1.1 ADC control register (ADC_CONTR)

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

ADC_POWER: ADC power supply control bit.

0: turn off the power supply of ADC.

1: turn on the power supply of ADC.

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

Pay attention:

1. After the power supply to the internal ADC module of the MCU is turned on, wait for about 1ms, and wait for the ADC power supply inside the MCU to stabilize before allowing the ADC to work;

2. Properly lengthening the sampling time of the external signal is the charging or discharging time of the internal sampling and holding capacitor of the ADC. If the time is enough, the internal can be equal to the external potential.

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

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

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

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

ADC_EPWMT: enable PWM synchronous trigger ADC function.

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

(Note: The I/O port selected as the ADC input channel must be set to the PxM0/PxM1 register to set the I/O port mode to high-impedance input mode. In addition, if the MCU enters the power-down mode/clock stop mode,

it still needs To enable the ADC channel, you need to set the PxIE register to close the digital input channel to prevent the external analog input signal from fluctuating high and low and causing additional power consumption)

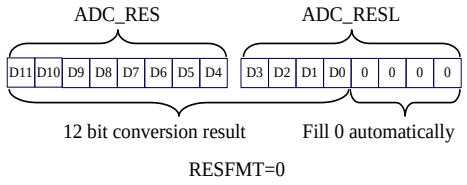
ADC_CHS[3:0]	ADC channel
0000	P1.0/ADC0
0001	P1.1/ADC1
0010	P5.4/ADC2
0011	P1.3/ADC3
0100	P1.4/ADC4
0101	P1.5/ADC5
0110	P6.2/ADC6
0111	P6.3/ADC7
1000	P0.0/ADC8
1001	P0.1/ADC9
1010	P0.2/ADC10
1011	P0.3/ADC11
1100	P0.4/ADC12
1101	P0.5/ADC13
1110	P0.6/ADC14
1111	Test internal 1.19V

17.1.2 ADC configuration register (ADCCFG)

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

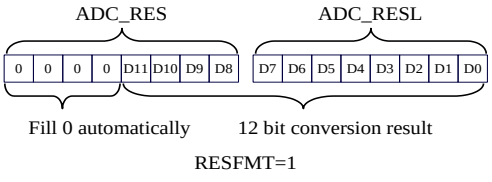
RESFMT: ADC conversion result format control bit

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



RESFMT=0

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



RESFMT=1

SPEED[3:0]: ADC clock control bits {F_{ADC}=SYSclk/2/(SPEED+1)}

SPEED[3:0]	ADC clock frequency
0000	SYSclk/2/1
0001	SYSclk/2/2
0010	SYSclk/2/3
...	. . .
1101	SYSclk/2/14
1110	SYSclk/2/15

1111	SYSclk/2/16

17.1.3 ADC result registers (ADC_RES, ADC_RESL)

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

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

17.1.4 ADC timing control register (ADCTIM)

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

CSSETUP: ADC channel selection time control T_{setup}

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

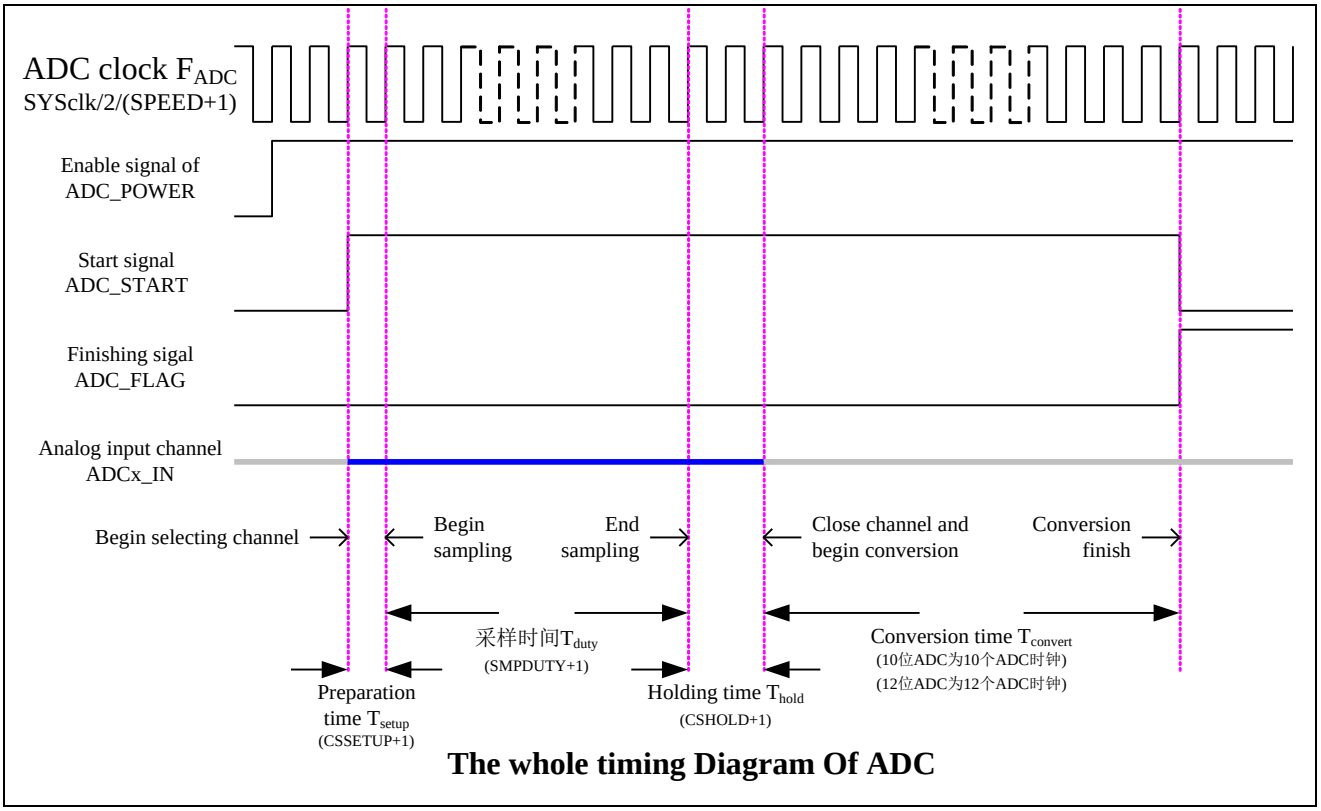
CSHOLD[1:0]: ADC Channel selection hold time control T_{hold}

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

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

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

ADC digital-to-analog conversion time: $T_{convert}$
The conversion time of 10-bit ADC is fixed at 10 ADC working clocks
The conversion time of 12-bit ADC is fixed at 12 ADC working clocks
A complete ADC conversion time is: Tsetup + Tduty + Thold + Tconvert, as shown in the figure below



The whole timing Diagram Of ADC

17.1.5 ADC Extended configuration registers (ADCEXCFG)

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

ADCETRS[1:0] : ADC external trigger pin ADC_ETR control bit

ADCETRS[1:0]	ADC_ETR configure
0x	Diable ETR function
10	Enable ADC_ETR rising edge to trigger ADC
11	Enable ADC_ETR falling edge to trigger ADC

Note: Before using this function, the ADC power switch in ADC_CONTR must be turned on, and the corresponding ADC channel must be set

CVTIMESEL[2:0] : ADC automatic conversion times selection

CVTIMESEL [2:0]	ADC automatic conversion times
0xx	One time
100	Convert 2 times and take the average
101	Convert 4 times and take the average
110	Convert 8 times and take the average
111	Convert 16 times and take the average

Note: When the ADC automatic many conversions function is enabled, the ADC interrupt flag will only be set to 1 after the ADC automatically converts to the set number of times (for example, set CVTIMESEL to 101B, that is, the ADC automatically converts 4 times and takes the average value, the ADC interrupt flag will be set to 1 after every 4 ADC conversions completed)

17.2 ADC related calculation formula

17.2.1 ADC speed calculation formula

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

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

Note:

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

17.2.2 ADC conversion result calculation formula

$$\text{12bit ADC conversion result} = 4096 * \frac{\text{The input voltage Vin of the ADC converted channel}}{\text{ADC external reference source voltage}} (has\ independent\ ADC_{Vref} + pin)$$

17.2.3 Reverse calculation formula for ADC input voltage

$$\text{input voltage Vin of the ADC converted channel} = \text{ADC external reference source voltage} * \frac{\text{12bit ADC conversion result}}{4096} (has\ independent\ ADC_{Vref} + pin)$$

17.2.4 Reverse working voltage calculation formula

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

$$\text{MCU working voltage Vcc} = 4096 * \frac{\text{input voltage Vin of the ADC converted channel}}{\text{12bit ADC conversion result}}$$

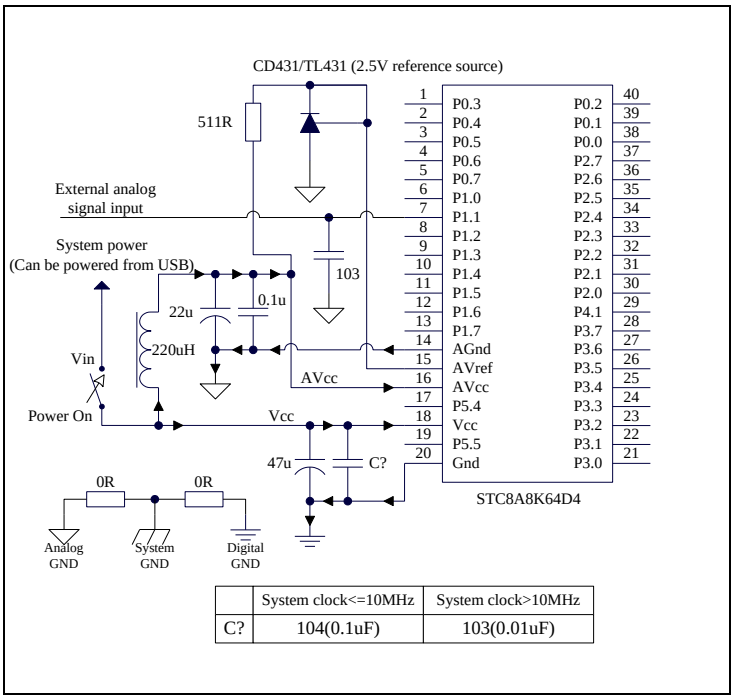
17.4 12 BIT ADC Static Characteristics

Symbol	Description	Minimum	Typical	Max	Unit
RES	Resolution	-	12	-	Bits

E_T	Overall error	-	0.5	1	LSB
E_O	Offset error	-	-0.1	1	LSB
E_G	Gain error	-	0	1	LSB
E_D	Differential nonlinear error	-	0.7	1.5	LSB
E_I	Integral nonlinear error	-	1	2	LSB
R_{AIN}	Channel equivalent resistance	-	∞	-	Ohm
RESD	Antistatic resistance connected in series before the sample and hold capacitor	-	700	-	Ohm
C_{ADC}	Internal sample and hold capacitor	-	16.5	-	pF

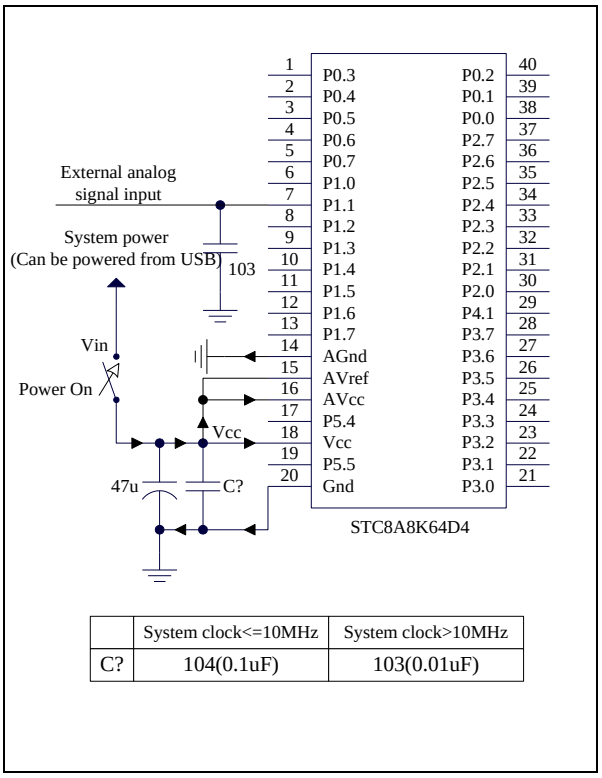
17.4 ADC application reference circuit diagram

17.4.1 High-precision ADC reference circuit diagram



	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

17.4.2 General precision ADC reference circuit diagram



	System clock<=10MHz	System clock>10MHz
C?	104(0.1uF)	103(0.01uF)

17.5 Example Routines

17.5.1 ADC Basic Operation (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

```
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

    P1M0 = 0x00;                    //Set P1.0 as ADC input
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                  // Set ADC internal timing
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;                  //Set the ADC clock to the system clock/2/16
    ADC_CONTR = 0x80;               //Enable ADC module

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

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

P_SW2       DATA    0BAH
ADCTIM      XDATA   0FEA8H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
```

```
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

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

            MOV     P1M0,#00H           ;Set P1.0 as ADC input
            MOV     P1M1,#01H
            MOV     P_SW2,#80H
            MOV     DPTR,#ADCTIM        ; Set ADC internal timing
            MOV     A,#3FH
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
            MOV     ADCCFG,#0FH         ;Set the ADC clock to the system clock/2/16
            MOV     ADC_CONTR,#80H      ;Enable ADC module
LOOP:
            ORL     ADC_CONTR,#40H      ;Start AD conversion
            NOP
            NOP
            MOV     A,ADC_CONTR         ;Query ADC completion flag
            JNB     ACC.5,$-2
            ANL     ADC_CONTR,#NOT 20H  ;Clear completion flag
            MOV     P2,ADC_RES          ;Read ADC results

            SJMP    LOOP

            END
```

17.5.2 ADC Basic Operation (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr         ADC_CONTR   =   0xbc;
```

```
sfr         ADC_RES     =   0xbd;
sfr         ADC_RESL    =   0xbe;
sfr         ADCCFG      =   0xde;

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

sbit        EADC        =   IE^5;

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

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

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

    P1M0 = 0x00;                        //Set P1.0 as ADC input
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                      // Set ADC internal timing
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;                      //Set the ADC clock to the system clock/2/16
    ADC_CONTR = 0x80;                   //Enable ADC module
    EADC = 1;                           //Enable ADC interrupt
    EA = 1;
    ADC_CONTR |= 0x40;                  //Start AD conversion

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

P_SW2       DATA    0BAH
ADCTIM      XDATA   0FEA8H

EADC        BIT     IE.5

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

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
            LJMP    ADCISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT 20H      ;Clear completion flag
            MOV     P2,ADC_RES              ;Read ADC results
            ORL     ADC_CONTR,#40H          ;Continue AD conversion
            RETI

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

            MOV     P1M0,#00H               ;Set P1.0 as ADC input
            MOV     P1M1,#01H
            MOV     P_SW2,#80H
```

```
        MOV     DPTR,#ADCTIM        ; Set ADC internal timing
        MOV     A,#3FH
        MOVX    @DPTR,A
        MOV     P_SW2,#00H
        MOV     ADCCFG,#0FH         ;Set the ADC clock to the system clock/2/16
        MOV     ADC_CONTR,#80H      ;Enable ADC module
        SETB    EADC                ;Enable ADC interrupt
        SETB    EA
        ORL     ADC_CONTR,#40H      ;Start AD conversion

        SJMP    $

        END
```

17.5.3 Format ADC Conversion Result

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

```
    P5M0 = 0x00;
    P5M1 = 0x00;

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

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

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

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

P_SW2       DATA    0BAH
ADCTIM      XDATA   0FEA8H

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
```

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

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

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

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

;       MOV     ADCCFG,#20H             ;Set result to align right
;       MOV     A,ADC_RES ;A [1: 0] stores the upper 2 bits of the 10-bit result of the ADC, and A [7: 2] is 0
;       MOV     B,ADC_RESL              ;B stores the lower 8 bits of the ADC's 10-bit result

        SJMP    $

        END
```

17.5.4 ADC converts multiple times to take average automatically

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

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

    P1M0 = 0x00;                                    // Set P1.0 as ADC input
    P1M1 = 0x01;
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                                  // Set ADC internal timing
//  ADCEXCFG = 0x04;                                //Set the ADC to convert 2 times to take the average automatically
//  ADCEXCFG = 0x05;                                //Set the ADC to convert 4 times to take the average automatically
    ADCEXCFG = 0x06;                                //Set the ADC to convert 8 times to take the average automatically
//  ADCEXCFG = 0x07;                                //Set the ADC to convert 16 times to take the average automatically
    P_SW2 &= 0x7f;
    ADCCFG = 0x0f;                                  // Set the ADC clock to the system clock/2/16
    ADC_CONTR = 0x80;                               // Enable ADC module

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

Assembly code

```
;Operating frequency for test is 11.0592MHz
```

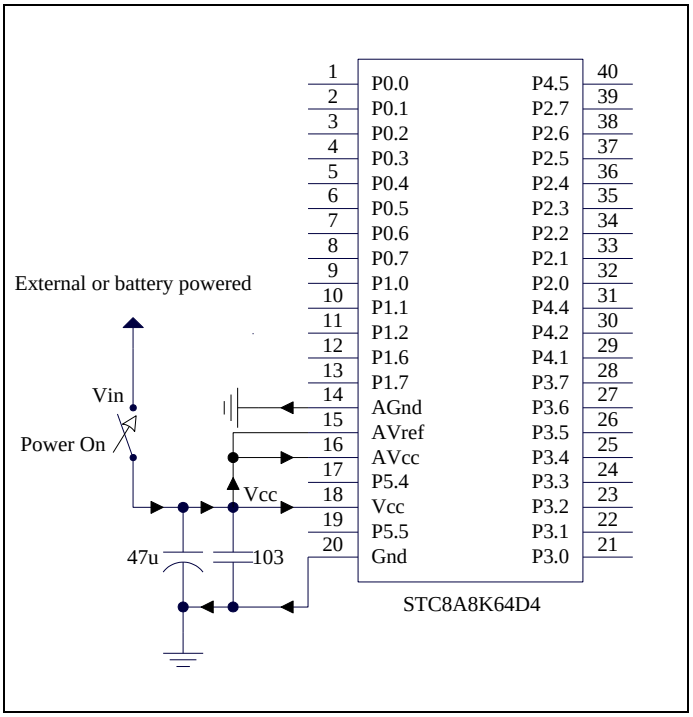
```
ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P_SW2       DATA    0BAH
ADCTIM      XDATA   0FEA8H
ADCEXCFG    XDATA   0FEAdH

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     P1M0,#00H           ; Set P1.0 as ADC input
            MOV     P1M1,#01H
            MOV     P_SW2,#80H
            MOV     DPTR,#ADCTIM        ; Set ADC internal timing
            MOV     A,#3FH
            MOVX    @DPTR,A
            MOV     DPTR,#ADCEXCFG
;           MOV     A,#04H              ; Set the ADC to convert 2 times to take the average automatically
            MOV     A,#05H              ; Set the ADC to convert 4 times to take the average automatically
;           MOV     A,#06H              ; Set the ADC to convert 8 times to take the average automatically
;           MOV     A,#07H              ; Set the ADC to convert 2 times to take the average automatically
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
            MOV     ADCCFG,#0FH         ; Set the ADC clock to the system clock/2/16
            MOV     ADC_CONTR,#80H      ; Enable ADC module
LOOP:
```

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

SJMP    LOOP

END
```

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

The 15th channel of ADC in the STC8A8K64D4 family of microcontrollers is used to measure the internal reference voltage. The internal reference voltage is stable, about 1.19V, and does not change with the chip's working voltage. So you can measure the internal reference voltage and use it to deduce the external voltage or external battery voltage through the value of the ADC.

The following figure is a reference circuit diagram:



C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr       AUXR        =     0x8e;
```

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

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

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

int         *BGV;                           //The internal reference voltage value is stored in idata
                                            //The high byte is stored in idata's EFH address
                                            //The low byte is stored in idata's F0H address
                                            //Voltage unit is millivolt (mV)
bit         busy;

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    busy = 0;
}

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

void ADCInit()
```

```
{
    P_SW2 |= 0x80;
    ADCTIM = 0x3f;                  // Set ADC internal timing
    P_SW2 &= 0x7f;

    ADCCFG = 0x2f;                  //Set the ADC clock to the system clock/2/16
    ADC_CONTR = 0x8f;               //Enable ADC module, and select channel 15
}

int   ADCRead()
{
    int res;

    ADC_CONTR |= 0x40;              //Start AD conversion
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));    //Query ADC completion flag
    ADC_CONTR &= ~0x20;             //Clear completion flag
    res = (ADC_RES << 8) | ADC_RESL;//Read ADC results

    return res;
}

void main()
{
    int res;
    int vcc;
    int i;

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

    BGV = (int idata *)0xef;
    ADCInit();                      //ADC initialization
    UartInit();                     //UART initialization

    ES = 1;
    EA = 1;

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

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

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

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

    while (1);
}
```

The method above uses the 15th channel of the ADC to invert the external battery voltage. In the ADC measurement range, the external measurement voltage of the ADC is proportional to the measurement value of the ADC, so the 15th channel of the ADC can also be used to reverse the input voltage of the external channel. Assuming that the current internal reference signal source voltage is BGV, the ADC measurement value of the internal reference signal source is res_{bg}, and the ADC measurement value of the external channel input voltage is res_x, then the external channel input voltage $Vx=BGV / res_{bg} * res_x$;

17.5.6 Using ADC as Capacitive Sensing Touch Keys

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

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

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

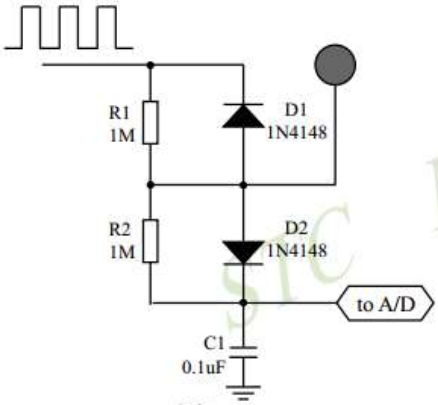


图1

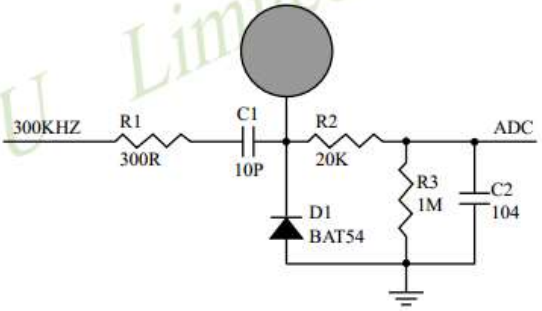


图2

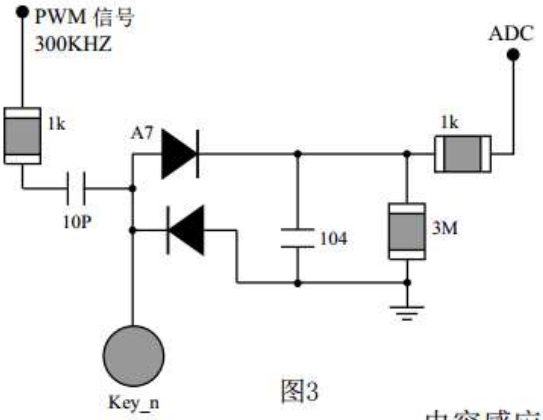


图3

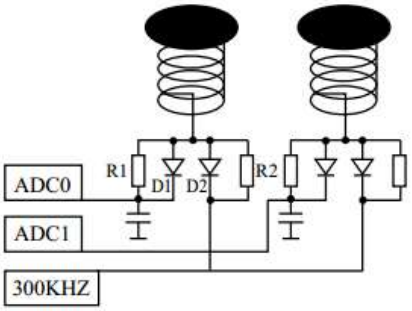
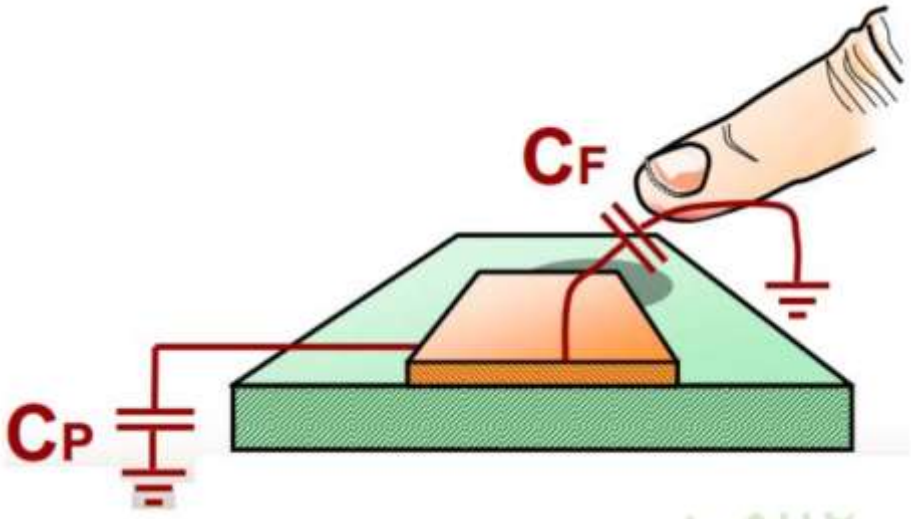


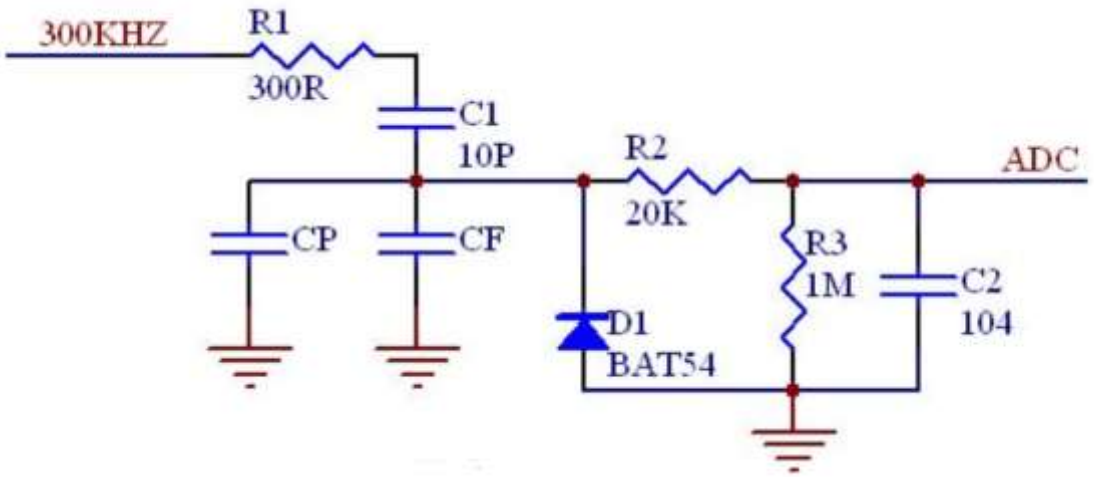
图4 加了感应弹簧

电容感应按键取样电路

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



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



C language code

```
//Operating frequency for testing is 24MHz

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

#define   MAIN_Fosc     24000000UL                  //Define the main clock
#define   Timer0_Reload (65536UL -(MAIN_Fosc / 600000))  //Timer 0 reload value corresponds to 300KHz

typedef   unsigned char  u8;
typedef   unsigned int   u16;
typedef   unsigned long  u32;

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

sfr       ADC_CONTR =    0xBC;                    // microcontrollers with ADC
sfr       ADC_RES   =    0xBD;                    / microcontrollers with ADC
sfr       ADC_RESL  =    0xBE;                    // microcontrollers with ADC
sfr       AUXR      =    0x8E;
sfr       AUXR2     =    0x8F;

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

sbit      P_LED7    =    P2^7;
sbit      P_LED6    =    P2^6;
```

```
sbit        P_LED5      =   P2^5;
sbit        P_LED4      =   P2^4;
sbit        P_LED3      =   P2^3;
sbit        P_LED2      =   P2^2;
sbit        P_LED1      =   P2^1;
sbit        P_LED0      =   P2^0;

u16 idata adc[TOUCH_CHANNEL];                   //Current ADC value
u16 idata adc_prev[TOUCH_CHANNEL];              //Previous ADC value
u16 idata TouchZero[TOUCH_CHANNEL];             //ADC value of 0-point
u8 idata TouchZeroCnt[TOUCH_CHANNEL];           //Automatic tracking count for 0-point
u8 cnt_250ms;

void delay_ms(u8 ms);
void ADC_init(void);
u16 Get_ADC10bitResult(u8 channel);
void AutoZero(void);
u8 check_adc(u8 index);
void ShowLED(void);

void main(void)
{
    u8 i;

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

    delay_ms(50);
    ET0 = 0;                                    //Initialize Timer0 to output a 300KHz clock
    TR0 = 0;
    AUXR |= 0x80;                               //Timer0 set as 1T mode
    AUXR2 |= 0x01;                              //Enable clock output
    TMOD = 0;                                   //Timer0 set as Timer, 16 bits Auto Reload.
    TH0 = (u8)(Timer0_Reload >> 8);
    TL0 = (u8)Timer0_Reload;
    TR0 = 1;
    ADC_init();                                 //ADC initialization
    delay_ms(50);                               //Delay 50ms
    for (i=0; i<TOUCH_CHANNEL; i++)  // Initialize the 0-point, the previous value and the 0-point auto-tracking count
    {
        adc_prev[i] = 1023;
        TouchZero[i] = 1023;
        TouchZeroCnt[i] = 0;
    }
    cnt_250ms = 0;
    while (1)
    {
        delay_ms(50);                           //Process key once every 50ms
```

```
            ShowLED();
            if (++cnt_250ms >= 5)
            {
                cnt_250ms = 0;
                AutoZero();                                 //Process 0-point auto-tracking every 250ms
            }
        }
}

void delay_ms(u8 ms)
{
        unsigned int i;

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

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

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

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

        for(i=0; i<TOUCH_CHANNEL; i++)                      //Process 8 channels
        {
            j = adc[i];
            k = j - adc_prev[i];                            // Subtract previous reading
            F0 = 0;                                         //Pressed
            if(k & 0x8000) F0 = 1, k = 0 - k;               //Release, get the difference between two samples
            if(k >= 20)                                     // Big change
            {
                TouchZeroCnt[i] = 0;                        // If the change is large, clear the counter
                if(F0) TouchZero[i] = j;    // If it is released, and the change is relatively large, then directly replace
            }
```

```
            else                        // If the change is relatively small, then creep, track 0-point automatically
            {
                if(++TouchZeroCnt[i] >= 20)                 // Continuously detect small changes 20 times/4 = 5 seconds.
                {
                    TouchZeroCnt[i] = 0;
                    TouchZero[i] = adc_prev[i];             // Use slowly changing values as 0 points                   }
            }
            adc_prev[i] = j;                                // Save this time's sample value
        }
}

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

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

void ShowLED(void)
{
    u8 i;

    i = check_adc(0);
    if(i == 0) P_LED0 = 1;                                  //Light off
    if(i == 1) P_LED0 = 0;                                  //Light on
    i = check_adc(1);
    if(i == 0) P_LED1 = 1;                                  //Light off
    if(i == 1) P_LED1 = 0;                                  //Light on
    i = check_adc(2);
    if(i == 0) P_LED2 = 1;                                  //Light off
    if(i == 1) P_LED2 = 0;                                  //Light on
    i = check_adc(3);
    if(i == 0) P_LED3 = 1;                                  //Light off
    if(i == 1) P_LED3 = 0;                                  //Light on
    i = check_adc(4);
    if(i == 0) P_LED4 = 1;                                  //Light off
    if(i == 1) P_LED4 = 0;                                  //Light on
    i = check_adc(5);
    if(i == 0) P_LED5 = 1;                                  //Light off
    if(i == 1) P_LED5 = 0;                                  //Light on
    i = check_adc(6);
    if(i == 0) P_LED6 = 1;                                  //Light off
    if(i == 1) P_LED6 = 0;                                  //Light on
    i = check_adc(7);
    if(i == 0) P_LED7 = 1;                                  //Light off
    if(i == 1) P_LED7 = 0;                                  //Light on
}
```

Assembly code

```
;Operating frequency for testing is 24MHz
```

```
Fosc_KHZ    EQU     24000                   ;Define the main clock KHZ
Reload      EQU     (65536 - Fosc_KHZ/600)  ;Timer 0 reload value, corresponding to 300KHz

ADC_CONTR   DATA    0xBC                    ; microcontrollers with ADC
ADC_RES     DATA    0xBD                    ; microcontrollers with ADC
ADC_RESL    DATA    0xBE                    ; microcontrollers with ADC
AUXR        DATA    0x8E
AUXR2       DATA    0x8F

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

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

P_LED7      BIT     P2.7;
P_LED6      BIT     P2.6;
P_LED5      BIT     P2.5;
P_LED4      BIT     P2.4;
P_LED3      BIT     P2.3;
P_LED2      BIT     P2.2;
P_LED1      BIT     P2.1;
P_LED0      BIT     P2.0;
adc         EQU     30H     ; Current ADC value in 30H ~ 3FH, two bytes constitute one value

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#0D0H
            MOV     P0M0,#00H
            MOV     P0M1,#00H
            MOV     P1M0,#00H
            MOV     P1M1,#00H
            MOV     P2M0,#00H
            MOV     P2M1,#00H
```

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

        MOV     R7,#50
        LCALL   F_delay_ms
        CLR     ET0                     ;Initialize Timer0 to output a 300KHz clock
        CLR     TR0
        ORL     AUXR,#080H              ;Timer0 set as 1T mode
        ORL     AUXR2,#01H              ;Enable clock output
        MOV     TMOD,#0                 ;Timer0 set as Timer,16 bits Auto Reload.
        MOV     TH0,#HIGH Reload
        MOV     TL0,#LOW Reload
        SETB    TR0
        LCALL   F_ADC_init
        MOV     R7,#50
        LCALL   F_delay_ms
        MOV     R0,#adc_prev            ;Initialize the previous ADC value
L_Init_Loop1:
        MOV     @R0,#03H
        INC     R0
        MOV     @R0,#0FFH
        INC     R0
        MOV     A,R0
        CJNE    A,#(adc_prev + CHANNEL * 2),L_Init_Loop1
        MOV     R0,#TouchZero           ;Initialize the ADC 0-point value
L_Init_Loop2:
        MOV     @R0,#03H
        INC     R0
        MOV     @R0,#0FFH
        INC     R0
        MOV     A,R0
        CJNE    A,#(TouchZero+CHANNEL * 2),L_Init_Loop2
        MOV     R0,#TouchZeroCnt        ;Initialize the automatic tracking count value
L_Init_Loop3:
        MOV     @R0,#0
        INC     R0
        MOV     A,R0
        CJNE    A,#(TouchZeroCnt + CHANNEL),L_Init_Loop3
        MOV     cnt_250ms,#5
L_MainLoop:
        MOV     R7,#50                  ;Delay 50ms
        LCALL   F_delay_ms
        LCALL   F_ShowLED               ;Handle key value once
        DJNZ    cnt_250ms,L_MainLoop
        MOV     cnt_250ms,#5    ;Processing once 0-point automatic tracking value every 250ms
        LCALL   F_AutoZero              ; Zero tracking
        SJMP    L_MainLoop

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

```
F_Get_ADC10bitResult:
        MOV     ADC_RES,#0
        MOV     ADC_RESL,#0
        MOV     A,R7
        ORL     A,#0E8H                 ;Trigger ADC
        MOV     ADC_CONTR,A
        NOP
        NOP
        NOP
        NOP
L_10bitADC_Loop1:
        MOV     A,ADC_CONTR
        JNB     ACC.4,L_10bitADC_Loop1  ;Wait for the ADC conversion complement
        MOV     ADC_CONTR,#080H         ;Clear flag
        MOV     A,ADC_RES
        MOV     B,#04H
        MUL     AB
        MOV     R7,A
        MOV     R6,B
        MOV     A,ADC_RESL
        ANL     A,#03H
        ORL     A,R7
        MOV     R7,A
        RET

F_AutoZero:                             ; Call once every 250ms
        ; This is detected using the sum of the absolute values of the differences between two adjacent samples.
        CLR     A
        MOV     R5,A
L_AutoZero_Loop:
        MOV     A,R5
        ADD     A,ACC
        ADD     A,#LOW (adc)
        MOV     R0,A
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
        MOV     R7,A
        MOV     A,R5
        ADD     A,ACC
        ADD     A,#LOW (adc_prev+01H)
        MOV     R0,A
        CLR     C
        MOV     A,R7
        SUBB    A,@R0
        MOV     R3,A
        MOV     A,R6
        DEC     R0
        SUBB    A,@R0
        MOV     R2,A
        CLR     F0                      ;Push down
        JNB     ACC.7,L_AutoZero_1
        SETB    F0
        CLR     C
        CLR     A
        SUBB    A,R3
        MOV     R3,A
```

```
                MOV     A,R3
                CLR     A
                SUBB    A,R2
                MOV     R2,A
L_AutoZero_1:
                CLR     C                       ;Calculate [R2 R3] -   #20,if(k >= 20)
                MOV     A,R3
                SUBB    A,#20
                MOV     A,R2
                SUBB    A,#00H
                JC      L_AutoZero_2            ;[R2 R3] ,20, Jump
                MOV     A,#LOW (TouchZeroCnt)   ; If the change is large, clear the counter TouchZeroCnt[i] = 0;
                ADD     A,R5
                MOV     R0,A
                MOV     @R0,#0
                JNB     F0,L_AutoZero_3
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (TouchZero)
                MOV     R0,A
                MOV     @R0,6
                INC     R0
                MOV     @R0,7
                SJMP    L_AutoZero_3
L_AutoZero_2:           ; If the change is relatively small, then creep, track 0-point automatically
                                                ; Continuously detect small changes 20 times/4 = 5 seconds.
                MOV     A,#LOW (TouchZeroCnt)
                ADD     A,R5
                MOV     R0,A
                INC     @R0
                MOV     A,@R0
                CLR     C
                SUBB    A,#20
                JC      L_AutoZero_3            ;if(TouchZeroCnt[i] < 20), jump
                MOV     @R0,#0                  ;TouchZeroCnt[i]= 0;
                MOV     A,R5                    ; Use slowly changing values as 0 points
                ADD     A,ACC
                ADD     A,#LOW (adc_prev)
                MOV     R0,A
                MOV     A,@R0
                MOV     R2,A
                INC     R0
                MOV     A,@R0
                MOV     R3,A
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (TouchZero)
                MOV     R0,A
                MOV     @R0,2
                INC     R0
                MOV     @R0,3
L_AutoZero_3:                                   ; Save the sampled value adc_prev[i] = j;
                MOV     A,R5
                ADD     A,ACC
                ADD     A,#LOW (adc_prev)
                MOV     R0,A
                MOV     @R0,6
                INC     R0
```

```
                MOV     @R0,7
                INC     R5
                MOV     A,R5
                XRL     A,#08H
                JZ      $ + 5H
                LJMP    L_AutoZero_Loop
                RET

F_check_adc:                                ; Judge key is pressed or released, with hysteresis control
                MOV R4,7
                LCALL   F_Get_ADC10bitResult    ; The ADC value returned is [R6 R7]
                CLR     C
                MOV     A,#0FFH
                SUBB    A,R7
                MOV     R7,A
                MOV     A,#03H
                SUBB    A,R6
                MOV     R6,A
                MOV     A,R4                ;Save adc[index]
                ADD     A,ACC
                ADD     A,#LOW (adc)
                MOV     R0,A
                MOV     @R0,6
                INC     R0
                MOV     @R0,7
                MOV     A,R4
                ADD     A,ACC
                ADD     A,#LOW (TouchZero+01H)
                MOV     R1,A
                MOV     A,R4
                ADD     A,ACC
                ADD     A,#LOW (adc)
                MOV     R0,A
                MOV     A,@R0
                MOV     R6,A
                INC     R0
                MOV     A,@R0
                CLR     C
                SUBB    A,@R1               ;Calculate adc[index] - TouchZero[index]
                MOV     A,R6
                DEC     R1
                SUBB    A,@R1
                JNC     L_check_adc_1
                MOV     R7,#00H
                RET
L_check_adc_1:
                MOV     A,R4
                ADD     A,ACC
                ADD     A,#LOW (TouchZero+01H)
                MOV     R1,A
                MOV     A,R4
                ADD     A,ACC
                ADD     A,#LOW (adc+01H)
                MOV     R0,A
                CLR     C
                MOV     A,@R0
                SUBB    A,@R1
                MOV     R7,A
```

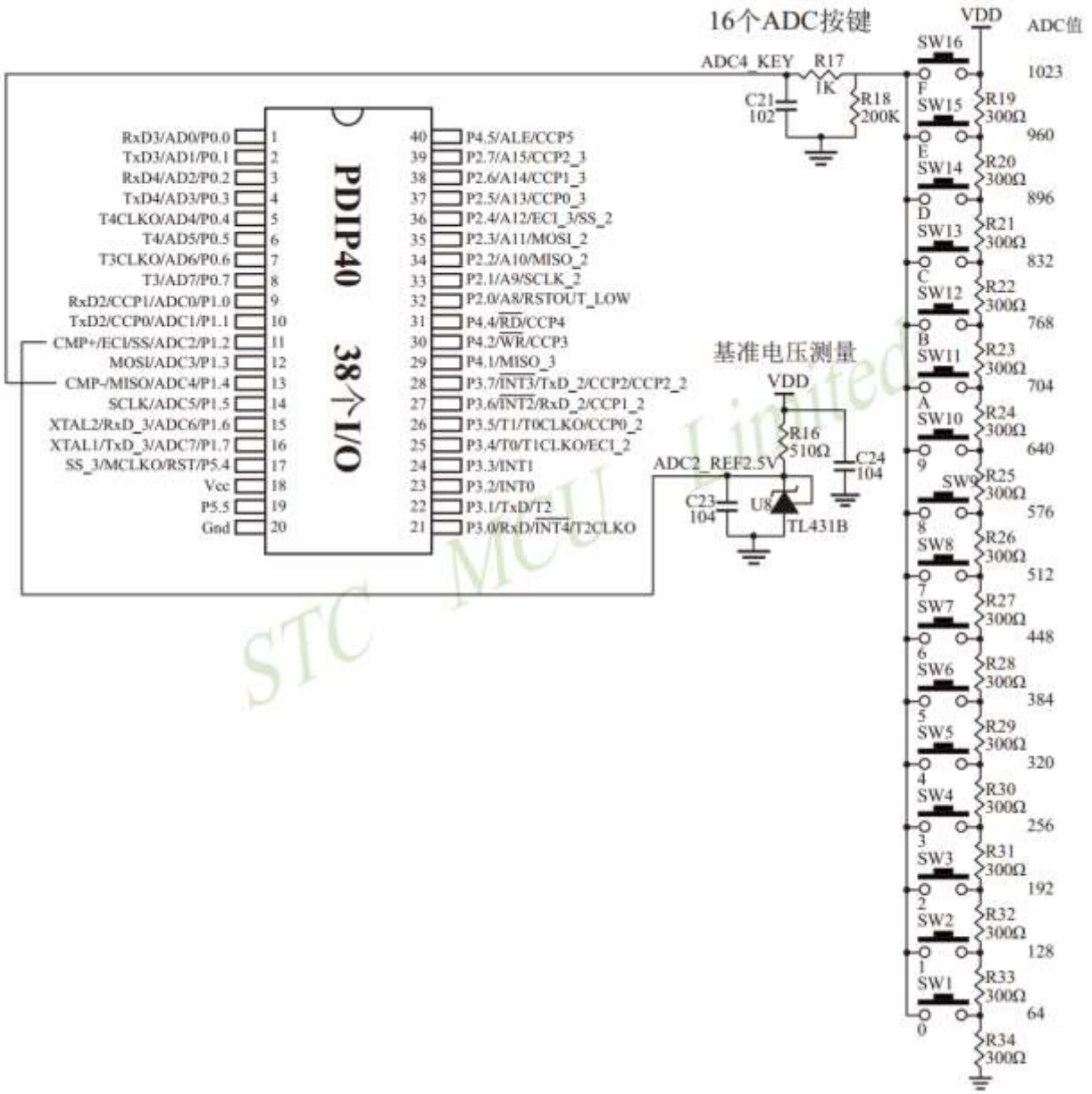
```
                DEC     R0
                MOV     A,@R0
                DEC     R1
                SUBB    A,@R1
                MOV     R6,A
                CLR     C
                MOV     A,R7
                SUBB    A,#40
                MOV     A,R6
                SUBB    A,#00H
                JC      L_check_adc_2           ;if(delta < 40), jump
                MOV     R7,#1                   ;if(delta >= 40) return 1; //Key pressed, return 1
                RET
L_check_adc_2:
                SETB    C
                MOV     A,R7
                SUBB    A,#20
                MOV     A,R6
                SUBB    A,#00H
                JNC     L_check_adc_3
                MOV     R7,#0
                RET
L_check_adc_3:
                MOV     R7,#2
                RET

F_ShowLED:
                MOV     R7,#0
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck0
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED0,C
L_QuitCheck0:
                MOV     R7,#1
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck1
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED1,C
L_QuitCheck1:
                MOV     R7,#2
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck2
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED2,C
L_QuitCheck2:
                MOV     R7,#3
```

```
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck3
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED3,C
L_QuitCheck3:
                MOV     R7,#4
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck4
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED4,C
L_QuitCheck4:
                MOV     R7,#5
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck5
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED5,C
L_QuitCheck5:
                MOV     R7,#6
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck6
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED6,C
L_QuitCheck6:
                MOV     R7,#7
                LCALL   F_check_adc
                MOV     A,R7
                ANL     A,#0FEH
                JNZ     L_QuitCheck7
                MOV     A,R7
                MOV     C,ACC.0
                CPL     C
                MOV     P_LED7,C
L_QuitCheck7:
                RET

F_delay_ms:
                PUSH    3
                PUSH    4
L_delay_ms_1:
                MOV     R3,#HIGH (Fosc_KHZ / 13)
                MOV     R4,#LOW (Fosc_KHZ / 13)
L_delay_ms_2:
```

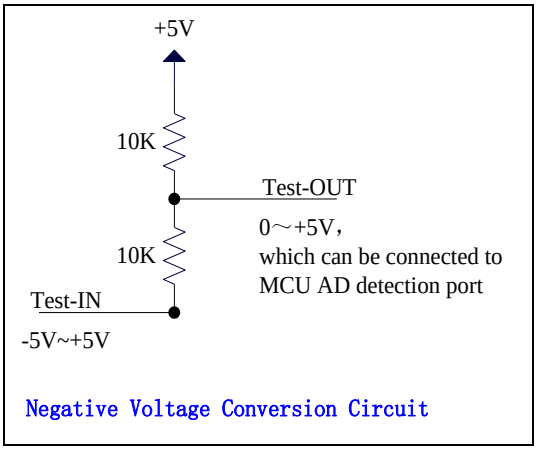
```
            MOV     A,R4
            DEC     R4
            JNZ     L_delay_ms_3
            DEC     R3
L_delay_ms_3:
            DEC     A
            ORL     A,R3
            JNZ     L_delay_ms_2
            DJNZ    R7,L_delay_ms_1
            POP     4
            POP     3
            RET

            END
```

17.5.7 Key-scan Application Circuit Diagram using ADC

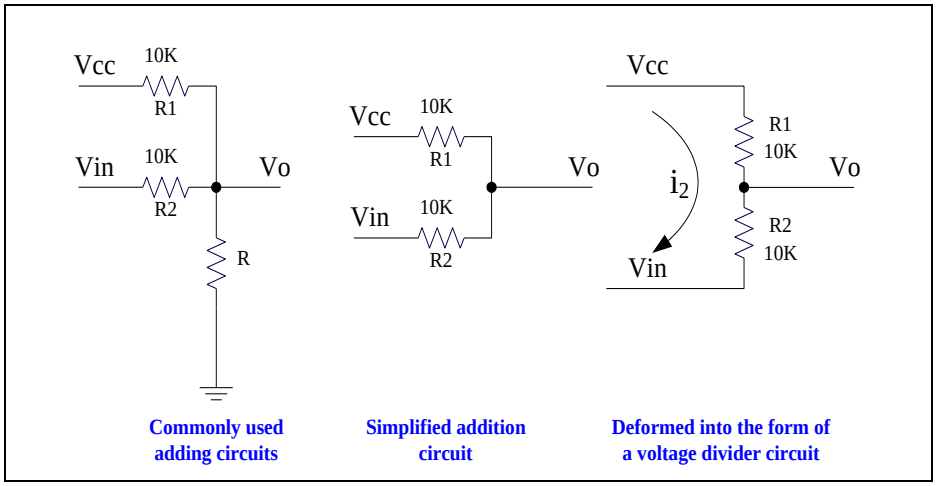
Method for reading the ADC key is reading the ADC value every 10ms or so and saving the last 3 readings. If the change is relatively small, judge the key. When the key is judged be valid, a certain deviation is allowed, such as a deviation of ±16 words.



17.5.8 Reference circuit diagram for detecting negative voltage



17.5.9 The application of common addition circuit in ADC



Refer to the voltage divider circuit to get formula 1
Formula 1: Vo = Vin + i2 * R2
Formula 2: i2 = (Vcc-Vin) / (R1 + R2) {Condition: the current flowing to Vo ≈0}

Substituting R1=R2 into formula 2 gives formula 3
Formula 3: i2 = (Vcc-Vin) / 2R2

Substituting formula 3 into formula 1 gives formula 4
Formula 4: Vo = (Vcc + Vin) / 2
According to formula 4, the above circuit can be regarded as an addition circuit.

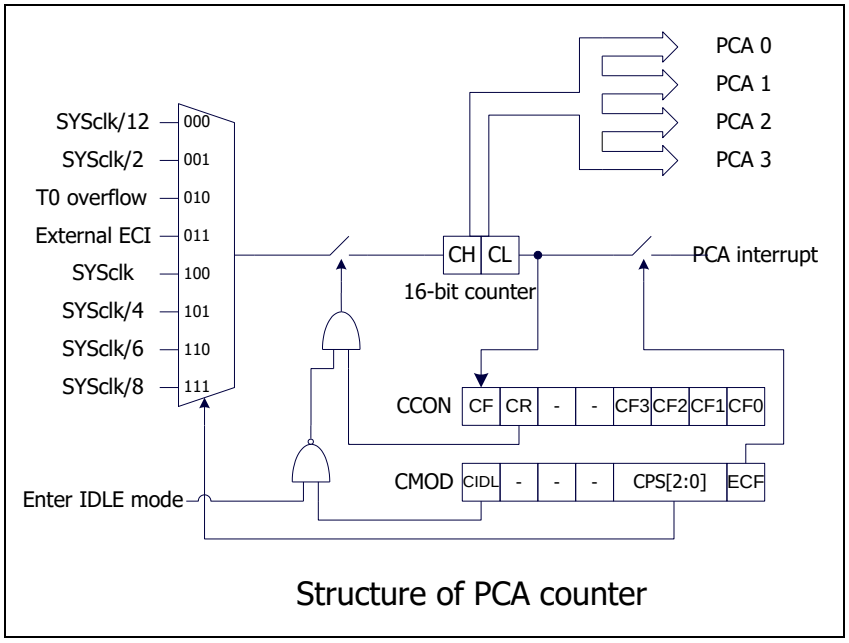
In the analog-to-digital conversion measurement of the microcontroller, the measured voltage is required to be greater than 0 and less than VCC. If the measured voltage is less than 0V, an addition circuit can be used to increase the measured voltage to above 0V. At this time, there are certain requirements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

Substituting the above conditions into formula 4, the following formula 2 can be obtained
(Vcc + Vin) / 2> 0 means Vin> -Vcc
(Vcc + Vin) / 2 <Vcc means Vin <Vcc
The above 2 formulas can be combined: -Vcc <Vin <Vcc

18 Application of PCA/CCP/PWM

Four groups of programmable counter array (PCA/CCP/PWM) modules are integrated in STC8A8K64D4 series of microcontrollers, which can be used for software timer, external pulse capture, high-speed pulse output and pulse width modulation (PWM) output.

PCA contains a special 16-bit counter, with which four groups of PCA modules are connected. The structure of PCA counter is as follows:



18.1 PCA function pin switch

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-

CCP_S[1:0]: PCA pin selection bit

CCP_S[1:0]	ECI	CCP0	CCP1	CCP2	CCP3
00	P1.2	P1.7	P1.6	P1.5	P1.4
01	P2.2	P2.3	P2.4	P2.5	P2.6
10	P7.4	P7.0	P7.1	P7.2	P7.3
11	P3.5	P3.3	P3.2	P3.1	P3.0

18.2 Registers Related to PCA

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
CCON	PCA Control Register	D8H	CF	CR	-	-	CCF3	CCF2	CCF1	CCF0	00xx,x000
CMOD	PCA Mode Register	D9H	CIDL	-	-	-	CPS[2:0]			ECF	0xxx,0000
CCAPM0	PCA 0 Mode Control Register	DAH	-	ECOM0	CCAPP0	CCAPN0	MAT0	TOG0	PWM0	ECCF0	x000,0000
CCAPM1	PCA 1 Mode Control Register	DBH	-	ECOM1	CCAPP1	CCAPN1	MAT1	TOG1	PWM1	ECCF1	x000,0000
CCAPM2	PCA 2 Mode Control Register	DCH	-	ECOM2	CCAPP2	CCAPN2	MAT2	TOG2	PWM2	ECCF2	x000,0000
CCAPM3	PCA 3 Mode Control Register	FD54H	-	ECOM3	CCAPP3	CCAPN3	MAT3	TOG3	PWM3	ECCF3	x000,0000
CL	PCA Counter Low Byte	E9H									0000,0000
CCAP0L	PCA 0 Capture Register Low Byte	EAH									0000,0000
CCAP1L	PCA 1 Capture Register Low Byte	EBH									0000,0000
CCAP2L	PCA 2 Capture Register Low Byte	ECH									0000,0000
CCAP3L	PCA 3 Capture Register Low Byte	FD55H									0000,0000
PCA_PWM0	PCA0 PWM Mode Register	F2H	EBS0[1:0]		XCCAP0H[1:0]		XCCAP0L[1:0]		EPC0H	EPC0L	0000,0000
PCA_PWM1	PCA1 PWM Mode Register	F3H	EBS1[1:0]		XCCAP1H[1:0]		XCCAP1L[1:0]		EPC1H	EPC1L	0000,0000
PCA_PWM2	PCA2 PWM Mode Register	F4H	EBS2[1:0]		XCCAP2H[1:0]		XCCAP2L[1:0]		EPC2H	EPC2L	0000,0000

PCA_PWM3	PCA3 PWM Mode Register	FD57H	EBS3[1:0]	XCCAP3H[1:0]	XCCAP3L[1:0]	EPC3H	EPC3L	0000,0000
CH	PCA Counter High Byte	F9H						0000,0000
CCAP0H	PCA 0 Capture Register High Byte	FAH						0000,0000
CCAP1H	PCA 1 Capture Register High Byte	FBH						0000,0000
CCAP2H	PCA 2 Capture Register High Byte	FCH						0000,0000
CCAP3H	PCA 3 Capture Register High Byte	FD56H						0000,0000

18.2.1 PCA control register (CCON)

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

CF: PCA Counter overflow flag. It is set by hardware when the 16-bit counter of PCA overflows, and requests interrupt to CPU. It must be cleared by software.

CR: PCA counter enable bit.

0: Stop PCA counting.

1: Start PCA counting.

CCFn(n=0,1,2,3): PCA interrupt flag. When a match or a capture occurs on the PCA module, the corresponding flag bit is set by the hardware automatically and requests an interrupt to CPU. These flags should be cleared by software.

18.2.2 PCA mode register (CMOD)

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

CIDL: PCA Counter control bit in Idle mode.

0: the PCA counter will continue counting in idle mode.

1: the PCA counter will stop counting in idle mode.

CPS[2:0]: PCA Counter pulse source select bits.

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

ECF: PCA counter overflow interrupt enable bit

0: disable PCA counter overflow interrupt

1: enable PCA counter overflow interrupt

18.2.3 PCA counter registers (CL, CH)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
CL	E9H								
CH	F9H								

The 16-bit counter is the combination of CL and CH, where CL is the low 8-bit counter and CH is the high 8-bit counter. The 16-bit counter of PCA increments automatically every one PCA clock.

18.2.4 PCA mode control registers (CCAPMn)

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

ECOMn: PCAn Comparator enable bit

CCAPPn: PCAn Capture on rising edge enable bit

CCAPNn: PCAn Capture on falling edge enable bit
MATn: PCAn match function enable bit
TOGn: PCAn high speed pulse output function enable bit
PWMn: PCAn PWM output function enable bit
ECCFn: PCAn match/capture interrupt enable bit

18.2.5 PCA capture value/compare value registers (CCAPnL, CCAPnH)

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

When the PCA capture function is enabled, CCAPnL and CCAPnH are used to save the count value (CL and CH) of the PCA at the time of capture. When the PCA comparison function is enabled, the PCA controller compares the current value in [CH,CL] and the value in [CCAPnH, CCAPnL], and the comparison result is given. When the PCA match function is enabled, the PCA controller compares the current value in [CH, CL] with the value stored in [CCAPnH, CCAPnL], and checks if they match (equal), then gives a match result.

18.2.6 PCA PWM mode control registers (PCA_PWMn)

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

EBSn[1:0]: PCAn PWM number of bits control

EBSn[1:0]	PWM bits	Reload value	Comparison value
00	8-bits PWM	{EPCnH, CCAPnH[7:0]}	{EPCnL, CCAPnL[7:0]}
01	7-bits PWM	{EPCnH, CCAPnH[6:0]}	{EPCnL, CCAPnL[6:0]}
10	6-bits PWM	{EPCnH, CCAPnH[5:0]}	{EPCnL, CCAPnL[5:0]}
11	10-bits PWM	{EPCnH, XCCAPnH[1:0], CCAPnH[7:0]}	{EPCnL, XCCAPnL[1:0], CCAPnL[7:0]}

XCCAPnH[1:0]: The 9th bit and 10th bit of reload value of 10-bit PWM
XCCAPnL[1:0]: The 9th bit and 10th bit of comparison value of 10-bit PWM
EPCnH: The MSB of reload vaule in PWM mode (i.e. the 9th bit of 8-bit PWM, the 8th bit of 7-bit PWM, the 7th bit of 6-bit PWM, the 11th bit of 10-bit PWM)
EPCnL: The MSB of comparison vaule in PWM mode (i.e. the 9th bit of 8-bit PWM, the 8th bit of 7-bit PWM, the 7th bit of 6-bit PWM, the 11th bit of 10-bit PWM)
Note: When updating the reload value of 10-bit PWM, write the upper two bits of XCCAPnH [1: 0] firstly and then the lower 8 bits of CCAPnH [7: 0].

18.3 PCA Operation Mode

There are 4 groups of PCA modules in STC12Hseries of microcontrollers, and operation mode of each module can be set independently. The mode settings are as follows:

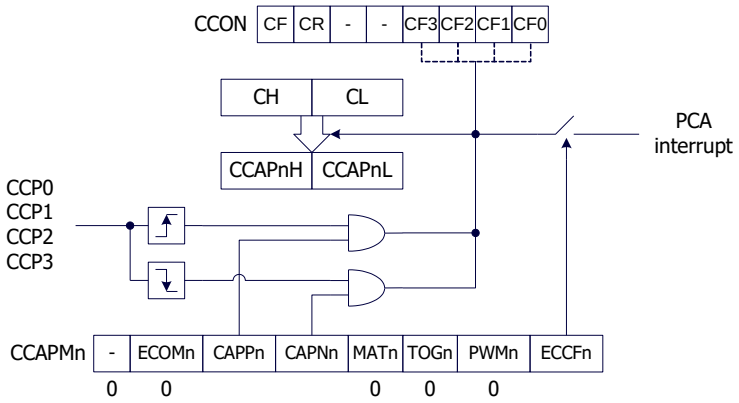
CCAPMn								Function of module
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	
-	0	0	0	0	0	0	0	No operation
-	1	0	0	0	0	1	0	6/7/8/10 bit PWM mode, no interrupt
-	1	1	0	0	0	1	1	6/7/8/10 bit PWM mode, rising edge interrupt
-	1	0	1	0	0	1	1	6/7/8/10 bit PWM mode, falling edge

								interrupt
-	1	1	1	0	0	1	1	6/7/8/10 bit PWM mode, rising and falling edge interrup
-	0	1	0	0	0	0	x	16 bit rising edge capture mode
-	0	0	1	0	0	0	x	16 bit falling edge capture mode
-	0	1	1	0	0	0	x	16 bit rising and falling edge capture mode
-	1	0	0	1	0	0	x	16 bit software timer mode
-	1	0	0	1	1	0	x	16 bit high speed pulse output mode

18.3.1 Capture Mode

At least one of CAPNn and CAPPn bits in CCAPMn must be set (or all of them are set) for a PCA module to operate in capture mode. When a PCA module is operating in capture mode, the input hoppings on the external CCP0/CCP1/CCP2 pins of the module are sampled. When a valid hopping is sampled, the PCA controller immediately loads the counter values in the PCA counters, CH and CL, into the module's capture registers, CCAPnL and CCAPnH, and sets the corresponding CCFn in the CCON register. If the ECCFn bit in CCAPMn is set to 1, an interrupt will be generated. Since all PCA modules' interrupt entry addresses are shared, it is necessary to determine which module generated an interrupt in the interrupt service routine and note that the interrupt flag bit must be cleared by software.

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

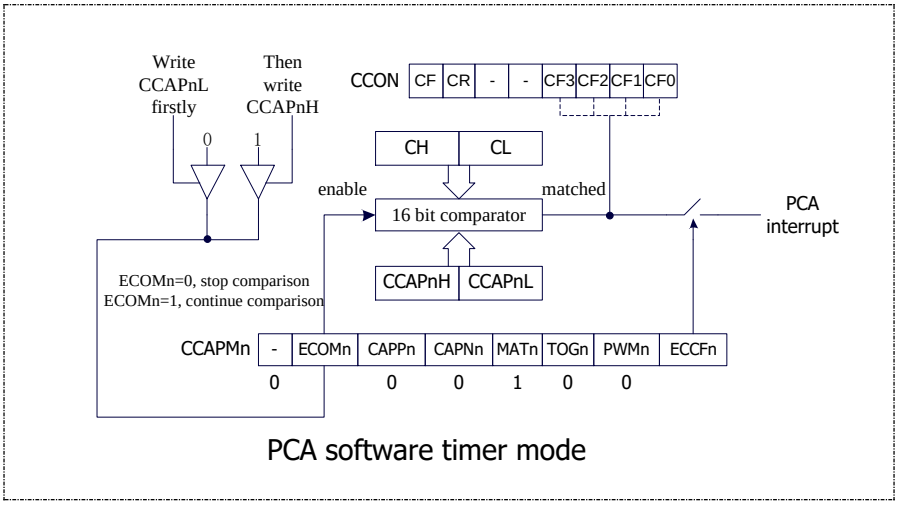


PCA capture mode

18.3.2 Software Timer Mode

The PCA module can be used as a software timer by setting the ECOM and MAT bits in the CCAPMn register. The PCA counter value in CL and CH is compared with the capture registers value in CCAPnL and CCAPnH. When they are equal, CCFn in CCON is set and an interrupt is generated if ECCFn in CCAPMn is set to 1. CCFn flag bit should be cleared by software.

The structure of PCA module working in software timer mode is shown in the following figure.

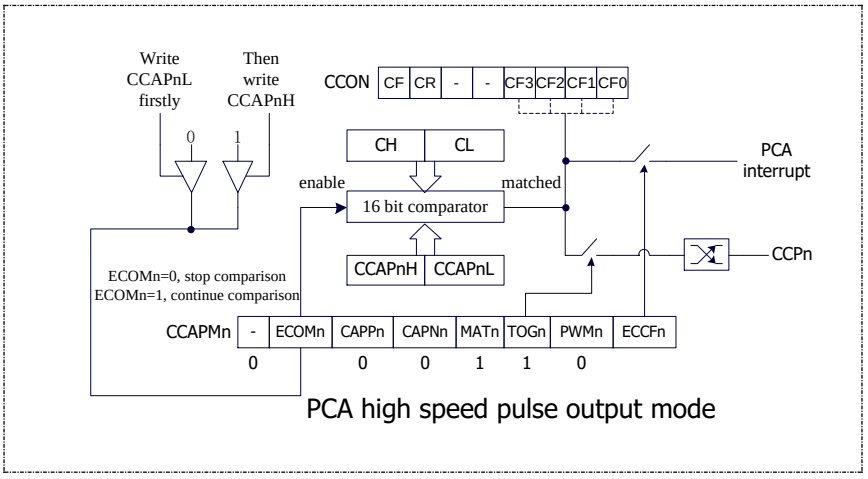


PCA software timer mode

18.3.3 High Speed Pulse Output Mode

When the count value of the PCA counter matches the value of the capture register, the CCPn output of the PCA module will hop. To activate the high speed pulse output mode, the TOGn, MATn, and ECOMn bits of the CCAPMn register must be set.

The structure of PCA module working in high-speed pulse output mode is shown below.



PCA high speed pulse output mode

18.3.4 Pulse Width Modulation Mode (PWM mode)

18.3.4.1 8-bit PWM Mode

Pulse width modulation is a technique that uses a program to control the duty ratio, cycle and phase of a waveform. It is widely used in applications such as three-phase motor driving and D/A conversion. The PCA modules of the STC8G series of microcontrollers can be configured to operate in 8-bit, 7-bit, 6-bit or 10-bit PWM mode by setting corresponding PCA_PWMn registers. To enable the PWM function of the PCA module, the PWMn and ECOMn bits of the module register CCAPMn must be set.

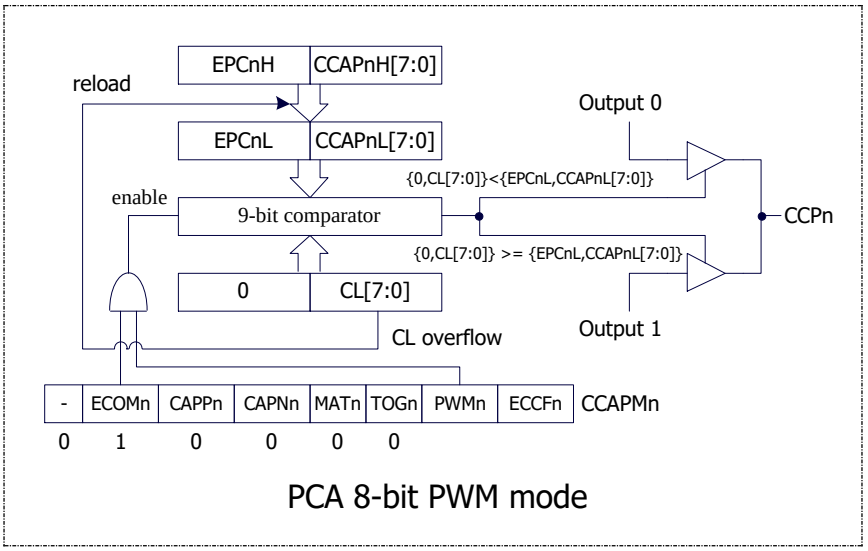
When EBSn [1:0] in the PCA_PWMn register is set to 00, PCAn operates in 8-bit PWM mode, where {0, CL [7: 0]} is compared with the capture registers {EPCnL, CCAPnL [7: 0]}. When PCA modules are operating in 8-bit PWM mode, the output frequencies of them are the same because all the modules share a single PCA counter. The output duty ratio of each module is set using the registers {EPCnL, CCAPnL [7: 0]}. The output is low when the value of {0, CL [7: 0]} is less than {EPCnL, CCAPnL [7: 0]}, and the output is high when the value of {0, CL [7: 0]} is equal to or greater than {EPCnL , CCAPnL [7: 0]}. When CL [7: 0] overflows from FF to 00, the contents of {EPCnH, CCAPnH

[7: 0]} are reloaded into {EPCnL, CCAPnL [7: 0]}. This makes it possible to update the PWM without interference.

$$\text{PWM frequency in 8-bit mode} = \frac{\text{PCA input clock source frequency}}{256}$$

When EPCnH=0 and CCAPnH=00H, PWM fixed output high
When EPCnH=1 and CCAPnH=FFH, PWM fixed output low

The structure of PCA module working in 8-bit PWM mode is shown below.



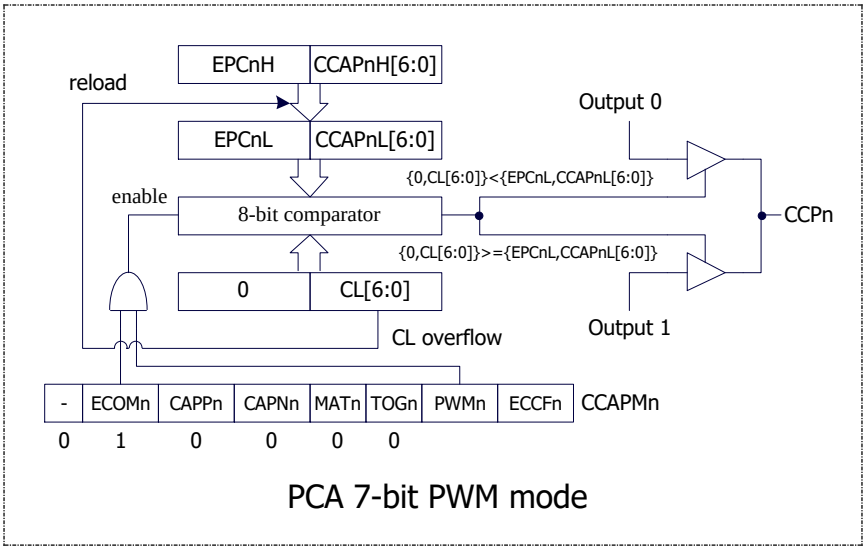
18.3.4.2 7-bit PWM Mode

When EBSn [1:0] in the PCA_PWMn register is set to 01, the PCAn operates in 7-bit PWM mode, where {0, CL [6: 0]} is compared with the capture registers {EPCnL, CCAPnL [6: 0]}. When PCA modules are operating in 7-bit PWM mode, the output frequencies of them are the same because all the modules share a single PCA counter. The output duty ratio of each module is set using the registers {EPCnL, CCAPnL [6: 0]}. The output is low when the value of {0, CL [6: 0]} is less than {EPCnL, CCAPnL [6: 0]}, and the output is high when the value of {0, CL [6: 0]} is equal to or greater than {EPCnL , CCAPnL [6: 0]}. When CL [6: 0] overflows from 7F to 00, the contents of {EPCnH, CCAPnH [6: 0]} are reloaded into {EPCnL, CCAPnL [6: 0]}. This makes it possible to update the PWM without interference.

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

When EPCnH=0 and CCAPnH=00H, PWM fixed output high
When EPCnH=1 and CCAPnH=FFH, PWM fixed output low

The structure of PCA module working in 7-bit PWM mode is shown below.



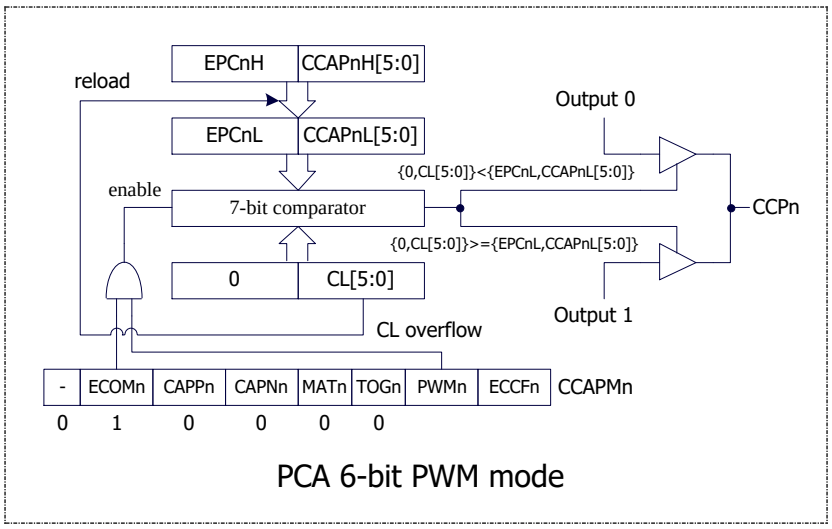
18.3.4.3 6-bit PWM Mode

When EBSn [1: 0] in the PCA_PWMn register is set to 10, PCAn operates in 6-bit PWM mode, where {0, CL [5: 0]} is compared with the capture registers {EPCnL, CCAPnL [5: 0] }. When PCA modules are operating in 6-bit PWM mode, the output frequencies of them are the same because all the modules share a single PCA counter. The output duty ratio of each module is set using the registers {EPCnL, CCAPnL [5: 0]}. The output is low when the value of {0, CL [5: 0]} is less than {EPCnL, CCAPnL [5: 0]}, and the output is high when the value of {0, CL [5: 0]} is equal to or greater than {EPCnL , CCAPnL [5: 0]}. When CL [5: 0] overflows from 3F to 00, the contents of {EPCnH, CCAPnH [5: 0]} are reloaded into {EPCnL, CCAPnL [5: 0]}. This makes it possible to update the PWM without interference.

$$\text{PWM frequency in 6-bit mode} = \frac{\text{PCA input clock source frequency}}{64}$$

When EPCnH=0 and CCAPnH=00H, PWM fixed output high
When EPCnH=1 and CCAPnH=FFH, PWM fixed output low

The structure of PCA module working in 6-bit PWM mode is shown below.



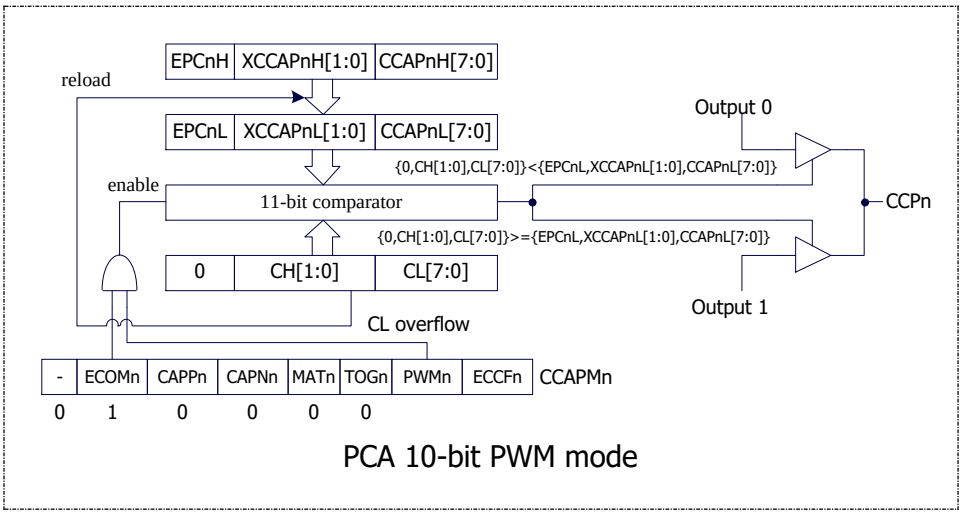
18.3.4.4 10-bit PWM Mode

When EBSn [1: 0] in the PCA_PWMn register is set to 11, PCAn operates in 10-bit PWM mode, where {CH[1:0],CL[7:0]} is compared with the capture registers{EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}. When PCA modules are operating in 10-bit PWM mode, the output frequencies of them are the same because all the modules share a single PCA counter. The output duty ratio of each module is set using the registers {EPCnL, XCCAPnL[1:0],CCAPnL[7:0]}. The output is low when the value of {CH[1:0],CL[7:0]} is less than {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}, and the output is high when the value of {CH[1:0],CL[7:0]} is equal to or greater than {EPCnL,XCCAPnL[1:0],CCAPnL[7:0]}. When {CH[1:0],CL[7:0]} overflows from 3FF to 00, the contents of {EPCnH,XCCAPnH[1:0],CCAPnH[7:0]} are reloaded into {EPCnL,XCCAPnL[1:0], CCAPnL[7:0]}. This makes it possible to update the PWM without interference.

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

When EPCnH=0 and CCAPnH=00H, PWM fixed output high
When EPCnH=1 and CCAPnH=FFH, PWM fixed output low

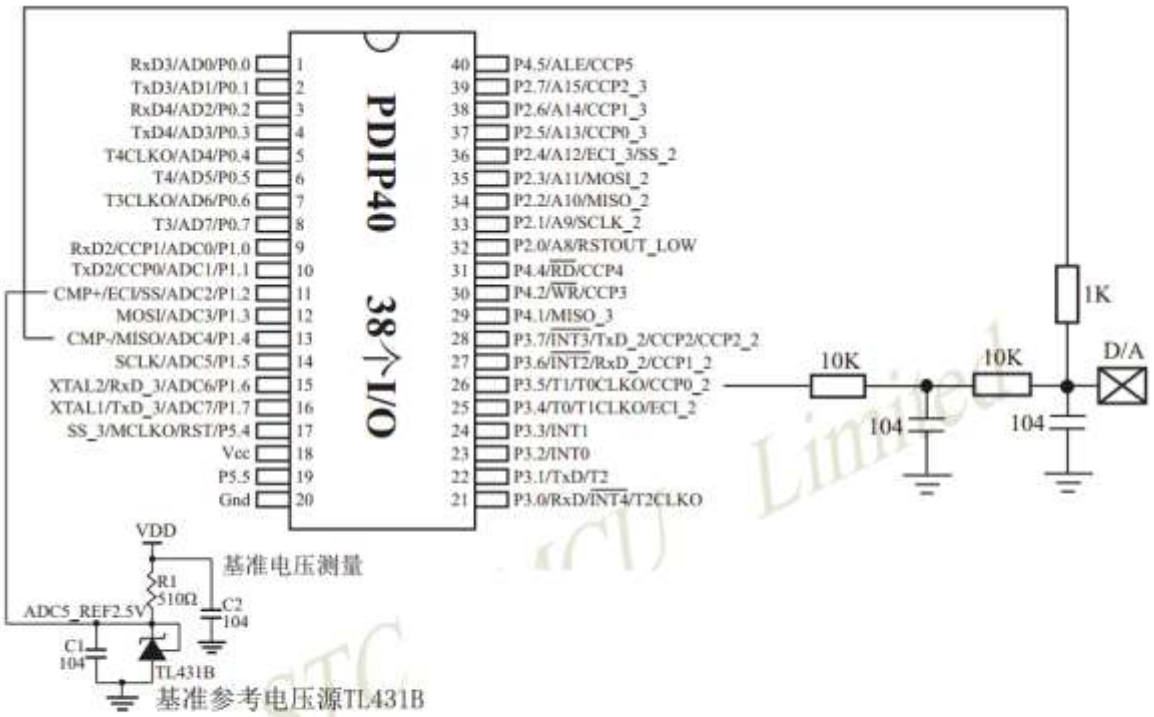
The structure of PCA module working in 10-bit PWM mode is shown below.



18.3.4.5 How to control PWM fixed output high / low

If PCA_PWMn &= 0xC0 and CCAPnH = 0x00, PWM fixed output high.
If PCA_PWMn |= 0x3F and CCAPnH = 0xFF, PWM fixed output low.

18.4 Reference Circuit for Implementing 8 ~ 16-bit DAC using CCP / PCA module



如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

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

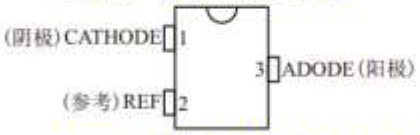
提示：

（1）PWM频率越高，输出波形越平滑。

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

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

基准参考电压源TL431B



SOT23-3封装，RMB￥0.15~0.3

基准参考电压源TL431B的符号

如应用简单，可无需基准参考电压源，直接与Vcc比较即可。

18.5 Example Routines

18.5.1 PCA Output PWM (6/7/8/10 bit)

C language code

```
//Operating frequency for test is 11.0592MHz

#include “reg51.h”
#include “intrins.h”

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

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CCON = 0x00;
    CMOD = 0x08;                    //PCA clock is the system clock
    CL = 0x00;
    CH = 0x00;
//——6 bit PWM——
    CCAPM0 = 0x42;                  //PCA 0 is in PWM mode
    PCA_PWM0 = 0x80;                //PCA 0 outputs 6-bit PWM
    CCAP0L = 0x20;                  //PWM duty cycle is 50%[(40H-20H)/40H]
    CCAP0H = 0x20;
//——7 bit PWM——
    CCAPM1 = 0x42;                  //PCA 1 is in PWM mode
    PCA_PWM1 = 0x40;                //PCA 1 outputs 7-bit PWM
    CCAP1L = 0x20;                  //PWM duty cycle is 75%[(80H-20H)/80H]
    CCAP1H = 0x20;
//——8 bit PWM——
//  CCAPM2 = 0x42;                  //PCA 2 is in PWM mode
//  PCA_PWM2 = 0x00;                //PCA 2 outputs 8-bit PWM
//  CCAP2L = 0x20;                  //PWM duty cycle is 87.5%[(100H-20H)/100H]
//  CCAP2H = 0x20;
//——10 bit PWM——
    CCAPM2 = 0x42;                  //PCA 2 is in PWM mode
    PCA_PWM2 = 0x00;                //PCA 2 outputs 10-bit PWM
    CCAP2L = 0x20;                  // PWM duty cycle is 96.875%[(400H-20H)/400H]
    CCAP2H = 0x20;
    CR = 1;                         //Start PCA timer

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

```
PCA_PWM2    DATA    0F4H

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     CCON,#00H
            MOV     CMOD,#08H       ;PCA clock is the system clock
            MOV     CL,#00H
            MOV     CH,#0H
;――6 bit PWM――
            MOV     CCAPM0,#42H     ;PCA 0 is in PWM mode
            MOV     PCA_PWM0,#80H   ;PCA 0 outputs 6-bit PWM
            MOV     CCAP0L,#20H     ;PWM duty cycle is 50%[(40H-20H)/40H]
            MOV     CCAP0H,#20H
;――7 bit PWM――
            MOV     CCAPM1,#42H     ;PCA 1 is in PWM mode
            MOV     PCA_PWM1,#40H   ;PCA 1 outputs 7-bit PWM
            MOV     CCAP1L,#20H     ;PWM duty cycle is 75%[(80H-20H)/80H]
            MOV     CCAP1H,#20H
;――8 bit PWM――
;           MOV     CCAPM2,#42H     ;PCA 2 is in PWM mode
;           MOV     PCA_PWM2,#00H   ;PCA 2 outputs 8-bit PWM
;           MOV     CCAP2L,#20H     ;PWM duty cycle is 87.5%[(100H-20H)/100H]
;           MOV     CCAP2H,#20H
;――10 bit PWM――
            MOV     CCAPM2,#42H     ;PCA 2 is in PWM mode
            MOV     PCA_PWM2,#0C0H  ;PCA 2 outputs 10-bit PWM
            MOV     CCAP2L,#20H     ;PWM duty cycle is 96.875%[(400H-20H)/400H]
            MOV     CCAP2H,#20H
```

```
        SETB    CR                  ;Start PCA timer
        JMP     $
        END
```

18.5.2 PCA Capture and Measure Pulse Width

C language code

```
//Operating frequency for test is 11.0592MHz

#include “reg51.h”
#include “intrins.h”

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

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

unsigned char   cnt;        //Store PCA timing overflow times
unsigned long   count0;     //Record the last captured value
unsigned long   count1;     //Record the current captured value
unsigned long   length;     //Store the time length of signal
```

```
void PCA_Isr() interrupt 7
{
    if (CF)
    {
        CF = 0;
        cnt++;                                  //PCA timing overflow times+1
    }
    if (CCF0)
    {
        CCF0 = 0;
        count0 = count1;                        //Back up the last captured value
        ((unsigned char *)&count1)[3] = CCAP0L;
        ((unsigned char *)&count1)[2] = CCAP0H;
        ((unsigned char *)&count1)[1] = cnt;
        ((unsigned char *)&count1)[0] = 0;
        length = count1 – count0;               //length saved is the captured pulse width
    }
}

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

    cnt = 0;                                    //User variable initialization
    count0 = 0;
    count1 = 0;
    length = 0;
    CCON = 0x00;
    CMOD = 0x09;                                //PCA clock is the system clock, enable PCA timing interrupt
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                              //PCA module 0 is 16-bit capture mode (falling edge capture)
    CCAPM0 = 0x21;                              //PCA module 0 is 16-bit capture mode (falling edge capture)
    CCAPM0 = 0x31;                              //PCA module 0 is 16-bit capture mode (falling edge capture)
    CCAP0L = 0x00;
    CCAP0H = 0x00;
    CR = 1;                                     //Start PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON        DATA        0D8H
```

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

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

            ORG     0100H
PCAISR:
            PUSH    ACC
            PUSH    PSW
            JNB     CF,CHECKCCF0
            CLR     CF          ;Clear interrupt flag
            INC     CNT         ; PCA Timing overflow times+1
CHECKCCF0:
            JNB     CCF0,ISREXIT
            CLR     CCF0
            MOV     COUNT0,COUNT1       ;Back up the last captured value
            MOV     COUNT0+1,COUNT1+1
            MOV     COUNT0+2,COUNT1+2
            MOV     COUNT1,CNT          ;Save the current captured value
```

```
                MOV     COUNT1+1,CCAP0H
                MOV     COUNT1+2,CCAP0L
                CLR     C                       ;Calculate two captures' differences
                MOV     A,COUNT1+2
                SUBB    A,COUNT0+2
                MOV     LENGTH+2,A
                MOV     A,COUNT1+1
                SUBB    A,COUNT0+1
                MOV     LENGTH+1,A
                MOV     A,COUNT1
                SUBB    A,COUNT0
                MOV     LENGTH,A                ;LENGTH saved is the captured pulse width
ISREXIT:
                POP     PSW
                POP     ACC
                RETI

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

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

                MOV     CCON,#00H
                MOV     CMOD,#09H               ;PCA clock is the system clock, enable PCA timing interrupt
                MOV     CL,#00H
                MOV     CH,#0H
                MOV     CCAPM0,#11H             ;PCA module 0 is 16-bit capture mode (falling edge capture)
;               MOV     CCAPM0,#21H             ;PCA module 0 is 16-bit capture mode (rising edge capture)
;               MOV     CCAPM0,#31H     ;PCA module 0 is 16-bit capture mode (falling and rising edge capture)
                MOV     CCAP0L,#00H
                MOV     CCAP0H,#00H
                SETB    CR                      ;Start PCA timer
                SETB    EA

                JMP     $
```

```
END
```

18.5.3 PCA Implements 16-bit Software Timing

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

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

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

sbit       P10         =   P1^0;

unsigned int               value;

void PCA_Isr() interrupt 7
{
    CCF0 = 0;
```

```
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;

    P10 = !P10;                                 //Test port
}

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

    CCON = 0x00;
    CMOD = 0x00;                                //PCA clock is the system clock/12
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x49;                              //PCA module 0 is 16-bit timer mode
    value = T50HZ;
    CCAP0L = value;
    CCAP0H = value >> 8;
    value += T50HZ;
    CR = 1;                                     //Start PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

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

```
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

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

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

            MOV     CCON,#00H
            MOV     CMOD,#00H           ;PCA clock is the system clock/12
```

```
        MOV     CL,#00H
        MOV     CH,#0H
        MOV     CCAPM0,#49H             ;PCA module 0 is 16-bit timer mode
        MOV     CCAP0L,#LOW T50HZ
        MOV     CCAP0H,#HIGH T50HZ
        SETB    CR                      ;Start PCA timer
        SETB    EA

        JMP     $

        END
```

18.5.4 PCA Output High-speed Pulse

C language code

```
//Operating frequency for test is 11.0592MHz

#include “reg51.h”
#include “intrins.h”

#define   T38K4HZ       (11059200L / 2 / 38400)

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

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

```
sfr         P5M1            =     0xc9;
sfr         P5M0            =     0xca;

unsigned int                value;

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

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

        CCON = 0x00;
        CMOD = 0x08;                                    //PCA clock is the system clock
        CL = 0x00;
        CH = 0x00;
        CCAPM0 = 0x4d;              //PCA module 0 is 16-bit timer mode, and enable pulse output
        value = T38K4HZ;
        CCAP0L = value;
        CCAP0H = value >> 8;
        value += T38K4HZ;
        CR = 1;                                         //Start PCA timer
        EA = 1;

        while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON        DATA        0D8H
CF          BIT         CCON.7
CR          BIT         CCON.6
CCF2        BIT         CCON.2
CCF1        BIT         CCON.1
CCF0        BIT         CCON.0
CMOD        DATA        0D9H
CL          DATA        0E9H
CH          DATA        0F9H
CCAPM0      DATA        0DAH
CCAP0L      DATA        0EAH
```

```
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

T38K4HZ     EQU     90H                 ;11059200/2/38400

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

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

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

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

```
        MOV     P5M0, #00H
        MOV     P5M1, #00H

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

        JMP     $

        END
```

18.5.5 PCA Extends External Interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

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

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

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
```

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

sbit    P10     =   P1^0;

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

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

    CCON = 0x00;
    CMOD = 0x08;                                    //PCA clock is the system clock
    CL = 0x00;
    CH = 0x00;
    CCAPM0 = 0x11;                                  //Extend external port CCP0 as a falling edge interrupt port
//  CCAPM0 = 0x21;                                  //Extend external port CCP0 as a rising edge interrupt port
//  CCAPM0 = 0x31;          //Extend external port CCP0 as a rising and falling edge interrupt port
    CCAP0L = 0;
    CCAP0H = 0;
    CR = 1;                                         //Start PCA timer
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CCON    DATA    0D8H
CF      BIT     CCON.7
CR      BIT     CCON.6
CCF2    BIT     CCON.2
CCF1    BIT     CCON.1
CCF0    BIT     CCON.0
CMOD    DATA    0D9H
CL      DATA    0E9H
CH      DATA    0F9H
CCAPM0  DATA    0DAH
```

```
CCAP0L      DATA    0EAH
CCAP0H      DATA    0FAH
PCA_PWM0    DATA    0F2H
CCAPM1      DATA    0DBH
CCAP1L      DATA    0EBH
CCAP1H      DATA    0FBH
PCA_PWM1    DATA    0F3H
CCAPM2      DATA    0DCH
CCAP2L      DATA    0ECH
CCAP2H      DATA    0FCH
PCA_PWM2    DATA    0F4H

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

            ORG     0000H
            LJMP    MAIN
            ORG     003BH
            LJMP    PCAISR

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

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

            MOV     CCON,#00H
            MOV     CMOD,#08H           ;PCA clock is the system clock
            MOV     CL,#00H
            MOV     CH,#0H
            MOV     CCAPM0,#11H         ;Extend external port CCP0 as a falling edge interrupt port
;           MOV     CCAPM0,#21H         ;Extend external port CCP0 as a rising edge interrupt port
;           MOV     CCAPM0,#31H   ;Extend external port CCP0 as a rising and falling edge interrupt port
```

```
MOV     CCAP0L,#0
MOV     CCAP0H,#0
SETB    CR                  ;Start PCA timer
SETB    EA

JMP     $

END
```

19 Enhanced PWM with 15-bit Accuracy

The STC8A8K64D4-64Pin/48Pin series microcontrollers integrate a group enhanced PWM waveform generator which can generate 8 independent PWMs.The clock source of PWM can be selected. There is a 15-bit PWM counter in the PWM waveform generator which is used for 8 channel PWMs. The initial level of each PWM can be set. In addition, two counters T1 and T2 are designed in the PWM waveform generator to control the waveform hopping for each PWM. The width of high and low level of each PWM can be set very flexibly, so that the PWM duty ratio and PWM output delay can be controlled. Because the 8 PWMs are independent and the initial state of each PWM can be set, any two of them can be used together to achieve special applications such as complementary symmetrical output and dead band control. Note: Enhanced PWM only has output function. If you need to measure pulse width, please use the PCA/CCP/PWM function of this series microcontroller. The enhanced PWM waveform generators also feature the ability to monitor external abnormal events, such as external port P3.5 abnormal level and the abnormal comparator results. It can be used to shutdown PWM outputs immediately. The PWM waveform generator can also be associated with ADC. Any point in the PWM cycle can be set to trigger the ADC conversion event.

Comparison of STC three kinds of hardware PWM:

Compatible with traditional 8051 PCA/CCP/PWM: It can output PWM waveforms, capture external input signals and output high-speed pulses. It can output 6-bit/7-bit/8-bit/10-bit PWM waveform externally. The frequency of the 6-bit PWM waveform is the PCA module clock source frequency/64. The frequency of the 7-bit PWM waveform is the PCA module clock source frequency/128. The frequency of the 8-bit PWM waveform is the PCA module clock source frequency/256. The frequency of the 10-bit PWM waveform is the PCA module clock source frequency/1024. For the external input signal capture, you can capture the rising edge, the falling edge or the rising edge and the falling edge at the same time.

15-bit enhanced PWM: It can only output PWM waveform externally, without input capture function. The frequency and duty cycle of PWM output can be set arbitrarily. Through software intervention, multiple complementary/symmetrical with dead-time PWM waveforms can be realized. There are external abnormality detection function and real-time trigger ADC conversion function.

The 16-bit advanced PWM timer: It is the PWM with the strongest function in STC products at present, which can output PWM waveforms of any frequency and any duty. It can output complementary/symmetrical with dead-time PWM waveform without software intervention. It can capture the external signal for the rising edge, the falling edge or the rising edge and the falling edge at the same time. When measuring the external waveform, the period value and the duty ratio value of the waveform can be measured at the same time. There are quadrature encoding function, external anomaly detection function and real-time trigger ADC conversion function.

19.1 Enhanced PWM output function pin switching

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF14H	ENC0O	C0INI	-	C0_S[1:0]		EC0I	EC0T2SI	EC0T1SI
PWM1CR	FF1CH	ENC1O	C1INI	-	C1_S[1:0]		EC1I	EC1T2SI	EC1T1SI
PWM2CR	FF24H	ENC2O	C2INI	-	C2_S[1:0]		EC2I	EC2T2SI	EC2T1SI
PWM3CR	FF2CH	ENC3O	C3INI	-	C3_S[1:0]		EC3I	EC3T2SI	EC3T1SI
PWM4CR	FF34H	ENC4O	C4INI	-	C4_S[1:0]		EC4I	EC4T2SI	EC4T1SI
PWM5CR	FF3CH	ENC5O	C5INI	-	C5_S[1:0]		EC5I	EC5T2SI	EC5T1SI
PWM6CR	FF44H	ENC6O	C6INI	-	C6_S[1:0]		EC6I	EC6T2SI	EC6T1SI
PWM7CR	FF4CH	ENC7O	C7INI	-	C7_S[1:0]		EC7I	EC7T2SI	EC7T1SI

C0_S[1:0]: Enhanced PWM channel 0 output pin selection bit

C0_S[1:0]	PWM0
00	P2.0
01	P1.0
10	P6.0
11	Reserved

C1_S[1:0] : Enhanced PWM channel 1 output pin selection bit

C1_S[1:0]	PWM1
00	P2.1
01	P1.1
10	P6.1
11	Reserved

C2_S[1:0] : Enhanced PWM channel 2 output pin selection bit

C2_S[1:0]	PWM2
00	P2.2
01	P1.2
10	P6.2
11	Reserved

C3_S[1:0] : Enhanced PWM channel 3 output pin selection bit

C3_S[1:0]	PWM3
00	P2.3
01	P1.3
10	P6.3
11	Reserved

C4_S[1:0] : Enhanced PWM channel 4 output pin selection bit

C4_S[1:0]	PWM4
00	P2.4
01	P1.4
10	P6.4
11	Reserved

C5_S[1:0] : Enhanced PWM channel 5 output pin selection bit

C5_S[1:0]	PWM5
00	P2.5
01	P1.5
10	P6.5
11	Reserved

C6_S[1:0] : Enhanced PWM channel 6 output pin selection bit

C6_S[1:0]	PWM6
00	P2.6
01	P1.6
10	P6.6
11	Reserved

C7_S[1:0] : Enhanced PWM channel 7 output pin selection bit

C7_S[1:0]	PWM7
00	P2.7
01	P1.7
10	P6.7
11	Reserved

19.2 Registers Related to PWM

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMSET	Enhanced PWM Global Configuration Register	F1H	-	PWMRST	-	-	-	-	-	ENPWM	x0xx,xxx0
PWMCFG	Enhanced PWM Configuration Register	F6H	-	-	-	-	PWMCBIF	EPWMCBI	ENPWMTA	PWMCEN	xxxx,0000

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMCH	PWM Counter High Byte	FF00H	-								x000,0000
PWMCL	PWM Counter Low Byte	FF01H									0000,0000
PWMCKS	PWM Clock Select Register	FF02H	-	-	-	SELT2	PWM_PS[3:0]				xxx0,0000
PWMTADCH	PWM Trigger ADC Counter High Byte	FF03H	-								x000,0000
PWMTADCL	PWM Trigger ADC Counter Low Byte	FF04H									0000,0000

PWMIF	PWM Interrupt Flag Register	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF	0000,0000
PWMFDCR	PWM Fault Detection Control Register	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF	0000,0000
PWM0T1H	PWM0 T1 High Byte	FF10H	-								x000,0000
PWM0T1L	PWM0 T1 Low Byte	FF11H									0000,0000
PWM0T2H	PWM0 T2 High Byte	FF12H	-								x000,0000
PWM0T2L	PWM0 T2 Low Byte	FF13H									0000,0000
PWM0CR	PWM0 Control Register	FF14H	ENO	INI	-	C0_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM0HLD	PWM0 Level Holding Control Register	FF15H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM1T1H	PWM1 T1 High Byte	FF18H	-								x000,0000
PWM1T1L	PWM1 T1 Low Byte	FF19H									0000,0000
PWM1T2H	PWM1 T2 High Byte	FF1AH	-								x000,0000
PWM1T2L	PWM1 T2 Low Byte	FF1BH									0000,0000
PWM1CR	PWM1 Control Register	FF1CH	ENO	INI	-	C1_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM1HLD	PWM1 Level Holding Control Register	FF1DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM2T1H	PWM2 T1 High Byte	FF20H	-								x000,0000
PWM2T1L	PWM2 T1 Low Byte	FF21H									0000,0000
PWM2T2H	PWM2 T2 High Byte	FF22H	-								x000,0000
PWM2T2L	PWM2 T2 Low Byte	FF23H									0000,0000
PWM2CR	PWM2 Control Register	FF24H	ENO	INI	-	C2_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM2HLD	PWM2 Level Holding Control Register	FF25H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM3T1H	PWM3 T1 High Byte	FF28H	-								x000,0000
PWM3T1L	PWM3 T1 Low Byte	FF29H									0000,0000
PWM3T2H	PWM3 T2 High Byte	FF2AH	-								x000,0000
PWM3T2L	PWM3 T2 Low Byte	FF2BH									0000,0000
PWM3CR	PWM3 Control Register	FF2CH	ENO	INI	-	C3_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM3HLD	PWM3 Level Holding Control Register	FF2DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM4T1H	PWM4 T1 High Byte	FF30H	-								x000,0000
PWM4T1L	PWM4 T1 Low Byte	FF31H									0000,0000
PWM4T2H	PWM4 T2 High Byte	FF32H	-								x000,0000
PWM4T2L	PWM4 T2 Low Byte	FF33H									0000,0000
PWM4CR	PWM4 Control Register	FF34H	ENO	INI	-	C4_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM4HLD	PWM4 Level Holding Control Register	FF35H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM5T1H	PWM5 T1 High Byte	FF38H	-								x000,0000
PWM5T1L	PWM5 T1 Low Byte	FF39H									0000,0000
PWM5T2H	PWM5 T2 High Byte	FF3AH	-								x000,0000
PWM5T2L	PWM5 T2 Low Byte	FF3BH									0000,0000
PWM5CR	PWM5 Control Register	FF3CH	ENO	INI	-	C5_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM5HLD	PWM5 Level Holding Control Register	FF3DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM6T1H	PWM6 T1 High Byte	FF40H	-								x000,0000
PWM6T1L	PWM6 T1 Low Byte	FF41H									0000,0000
PWM6T2H	PWM6 T2 High Byte	FF42H	-								x000,0000
PWM6T2L	PWM6 T2 Low Byte	FF43H									0000,0000
PWM6CR	PWM6 Control Register	FF44H	ENO	INI	-	C6_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM6HLD	PWM6 Level Holding Control Register	FF45H	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00
PWM7T1H	PWM7 T1 High Byte	FF48H	-								x000,0000
PWM7T1L	PWM7 T1 Low Byte	FF49H									0000,0000
PWM7T2H	PWM7 T2 High Byte	FF4AH	-								x000,0000
PWM7T2L	PWM7 T2 Low Byte	FF4BH									0000,0000
PWM7CR	PWM7 Control Register	FF4CH	ENO	INI	-	C7_S[1:0]		ENI	ENT2I	ENT1I	00xx,x000
PWM7HLD	PWM7 Level Holding Control Register	FF4DH	-	-	-	-	-	-	HLDH	HLDL	xxxx,xx00

19.2.1 Enhanced PWM global configuration register (PWMSET)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMSET	F1H	-	PWMRST	-	-	-	-	-	ENPWM

PWMRST: Software resets 6 groups of PWM.

0: No effect.

1: Reset the XFR registers for all PWMs, but do not reset SFRs which are required software reset.

ENPWM: PWM enable bit (including PWM0~PWM7).

0: disable PWM

1: enable PWM

19.2.2 Enhanced PWM configuration registers (PWMCFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMCFG01	F6H	-	-	-	-	PWMCBIF	EPWMCBI	ENPWMTA	PWMCEN

PWMCBIF: The interrupt flag bit of PWM when the PWM counter returns to zero (n=0~5).

When the 15-bit PWM counter overflows and returns to zero, this bit is set by hardware automatically and an interrupt is requested to CPU. This flag bit needs to be cleared by software.

EPWMCBI: PWM counter returns to zero interrupt enable bit. (n= 0 ~5)

0: disable PWM counter returns to zero interrupt (PWMCBIF is still set by hardware)
1: enable PWM counter returns to zero interrupt

EPWMTA: Whether the PWM is associated with the ADC or not.
0: PWM is not associated with the ADC.
1: PWM is associated with the ADC. A/D conversion is enabled to be triggered at a certain point in the PWM cycle. PWMTADCH and PWMTADCL are used to set the counter value..
(**Note: The ADC_POWER and ADC_EPWMT bits in the ADC_CONTR register need to be set at the same time to enable the PWM triger function. PWM only sets ADC_START to 1 automatically.**)

PWMCEN: PWMn waveform generator starts counting control bit.
0: PWM stops counting
1: PWM starts counting

Important note about the PWMCEN control bit:

- Once PWMCEN is enabled, the internal PWM counter will start counting immediately and compare the counting value with the value of T1/T2. So PWMCEN must be enabled after all other PWM settings (including T1 / T2 settings, initial level setting, PWM fault detection setting and PWM interrupt setting) are completed.
- During the PWM counter counting, when the PWMCEN control bit is turned off, the PWM counting will stop immediately. When the PWMCEN control bit is enabled again, the PWM counting will start counting from 0 again without remembering the count value before the PWM stopped counting.
- Special attention: When PWMCEN changes from 0 to 1, the internal PWM counter starts to count again after returning to zero from the previous uncertain value, so a reset interrupt will be generated immediately at this time. When the user needs to use the reset to zero interrupt of PWM, special attention should be paid to this point, that is, the first reset to zero interrupt is not generated by the reset to zero after the real PWM cycle is full.

19.2.3 PWM Interrupt Flag Register (PWMIF)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMIF	FF05H	C7IF	C6IF	C5IF	C4IF	C3IF	C2IF	C1IF	C0IF

CiIF: The interrupt flag bit of i-channel of PWM. (i＝0~7)
T1 and T2 of each PWM can be set. When a match event occurs at the set point, this bit is set by hardware automatically and an interrupt is requested to the CPU. This flag bit must be cleared by software.

19.2.4 PWM Fault Detection Control Register (PWMFDCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMFDCR	FF06H	INVCMP	INVIO	ENFD	FLTFLIO	EFDI	FDCMP	FDIO	FDIF

INVCMP: Fault signal of comparator result selection bit
0: the fault signal is the comparator result changing from low to high.
1: the fault signal is the comparator result changing from high to low.

INVIO: Fault signal of external port PWMFLT selection bit
0: the fault signal is the external port PWMFLT signal changing from low to high.
1: the fault signal is the external port PWMFLT signal changing from high to low.

ENFD: PWM external fault detection enable bit.
0: disable the PWM external fault detection.
1: enable the PWM external fault detection.

FLTFLIO: PWM output port control bit when external PWM fault occurs.
0: the PWM output port does not change when external PWM fault occurs.
1: the PWM outport port is set as high impedance input mode when external PWM default occurs.
Note: Only the port whose corresponding ENO = 1 is forcibly in high impedance state.

EFDI: PWM fault detection interrupt enable bit.
0: disable PWM fault detection interrupt (FDIF will still be set by hardware.)
1: enable PWM fault detection interrupt

FDCMP: fault detection of comparator output enable bit.
0: the comparator is not associated with PWM.
1: the source of PWM fault detection is comparator output. (The fault type is set by INVCMP.)

FDIO: PWMFLT level fault detection enable bit.

0: PWMFLT level is not associated with PWM
1: the source of PWM fault detection is PWMFLT. (The fault type is set by INVIO.)

FDIF: the interrupt flag bit of PWM fault detection.
It is set automatically by the hardware when a PWM fault occurs. If EFDI=1, the program will jump to the corresponding interrupt entry to execute interrupt service routine. It should be cleared by software.

19.2.5 PWM Counter Registers (PWMCH, PWMCL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMCH	FF00H	-							
PWMCL	FF01H								

PWMCH: upper 7 bits of PWM counter period value.
PWMCL: lower 8 bits of PWM counter period value.
The PWM counter is a 15-bit register that can be set to any value between 1 and 32767 as the PWM cycle. The counter inside the PWM waveform generator counts from 0 and increments by 1 every PWM clock cycle. When the internal counter reaches the PWM cycle set by [PWMCH, PWMCL], the internal counter of the PWM waveform generator will count from 0 again, and the PWM return-to-zero interrupt flag bit PWMCBIF will be set by hardware automatically. If ECBI = 1, the program will jump to the corresponding interrupt entry address to execute the interrupt service routine.

19.2.6 PWM Clock Selection Register (PWMCKS)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWMCKS	FF02H	-	-	-	SELT2	PWM_PS[3:0]			

SELT2: PWM clock source selection bit.
0: The PWM clock source is the clock generated by the system clock being divided by the frequency divider.
1: The PWM clock source is the overflow pulse of timer 2.

PWM_PS[3:0]: System clock prescaler parameter select bits

SELT2	PWM_PS[3:0]	PWM input clock frequency
1	xxxx	Overflow pulse of Timer 2
0	0000	SYSclk/1
0	0001	SYSclk/2
0	0010	SYSclk/3
…	…	…
0	x	SYSclk/(x+1)
…	…	…
0	1111	SYSclk/16

PWM output frequency calculation formula
The output frequency calculation formula of the 6 groups of PWM is the same, and each group can be set with a different frequency.

Clock source selection (SELT2)	PWM output frequency calculation formula
SELT2＝0 (System clock)	PWM output frequency $= \frac{\text{System operating frequency SYSclk}}{(\text{PWM\_PS} + 1) \times (\text{[PWMCH, PWMCL]} + 1)}$
SELT2＝1 (Timer 2 overflow pulse)	PWM output frequency $= \frac{\text{Timer 2 overflow pulse frequency}}{(\text{[PWMCH, PWMCL]} + 1)}$

19.2.7 PWM Trigger ADC counter Registers (PWMTADC)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0

PWMTADCH	FF03H	-	
PWMTADCL	FF04H		

PWMTADCH: the upper 7 bits of the PWM triggers ADC time.
PWMTADCL: the lower 8 bits of the PWM triggers ADC time.
If EPWMTA =1 and ADC_POWER=1, {PWMTADCH, PWMTADCL} forms a 15-bit register. In the PWM counting cycle, the hardware will trigger A/D conversion automatically when the internal PWM counting value equals to the value of {PWMTADCH, PWMTADCL}.

19.2.8 PWM Level output setting count value Registers (PWMnT1，PWMnT2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0T1H	FF10H	-							
PWM0T1L	FF11H								
PWM0T2H	FF12H	-							
PWM0T2L	FF13H								
PWM1T1H	FF18H	-							
PWM1T1L	FF19H								
PWM1T2H	FF1AH	-							
PWM1T2L	FF1BH								
PWM2T1H	FF20H	-							
PWM2T1L	FF21H								
PWM2T2H	FF22H	-							
PWM2T2L	FF23H								
PWM3T1H	FF28H	-							
PWM3T1L	FF29H								
PWM3T2H	FF2AH	-							
PWM3T2L	FF2BH								
PWM4T1H	FF30H	-							
PWM4T1L	FF31H								
PWM4T2H	FF32H	-							
PWM4T2L	FF33H								
PWM5T1H	FF38H	-							
PWM5T1L	FF39H								
PWM5T2H	FF3AH	-							
PWM5T2L	FF3BH								
PWM6T1H	FF40H	-							
PWM6T1L	FF41H								
PWM6T2H	FF42H	-							
PWM6T2L	FF43H								
PWM7T1H	FF48H	-							
PWM7T1L	FF49H								
PWM7T2H	FF4AH	-							
PWM7T2L	FF4BH								

PWMiT1H: The upper 7 bits of T1 counter value of channel i of PWM. (i= 0~7)
PWMiT1L: The lower 8 bits of T1 counter value of channel i of PWM. (i= 0~7)
PWMiT2H: The upper 7 bits of T2 counter value of channel i of PWM. (i= 0~7)
PWMiT2L: The lower 8 bits of T2counter value of channel i of PWM. (i= 0~7)
{PWMiT1H, PWMiT1L} and {PWMiT2H, PWMiT2L} of every channel of every PWM are combined into two 15-bit registers, which are used to control the two flip points of the PWM output waveform in every PWM cycle of each PWM. During the counting cycle of PWM, the output of PWM will be low level when the internal counting value of PWMn is equal to the value of T1 set by{PWMiT1H, PWMiT1L}. And the output of the PWM will be high level when the internal counting value of PWM is queal to the value of the T2 set by {PWMT2H, PWMT2L}.
Note: When the values of {PWMiT1H, PWMiT1L} and {PWMiT2H, PWMiT2L} are set equal, and if the internal count value of PWM is equal to the set value of T1 / T2, it will output low level.

19.2.9 PWM Channel Control Registers (PWMnCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0CR	FF14H	ENO	INI	-	C0_S[1:0]		ENI	ENT2I	ENT1I
PWM1CR	FF1CH	ENO	INI	-	C1_S[1:0]		ENI	ENT2I	ENT1I
PWM2CR	FF24H	ENO	INI	-	C2_S[1:0]		ENI	ENT2I	ENT1I
PWM3CR	FF2CH	ENO	INI	-	C3_S[1:0]		ENI	ENT2I	ENT1I
PWM4CR	FF34H	ENO	INI	-	C4_S[1:0]		ENI	ENT2I	ENT1I
PWM5CR	FF3CH	ENO	INI	-	C5_S[1:0]		ENI	ENT2I	ENT1I
PWM6CR	FF44H	ENO	INI	-	C6_S[1:0]		ENI	ENT2I	ENT1I
PWM7CR	FF4CH	ENO	INI	-	C7_S[1:0]		ENI	ENT2I	ENT1I

ENO: PWMi output enable bit. (i= 0~7)

0: the corresponding port of PWM channel i is GPIO.

1: the corresponding port of PWM channel i is PWM output port, which is controlled by the PWM waveform generator.

INI: the initial level of PWMi output. (i= 0~7)

0: the initial level of PWM channel i is low.

1: the initial level of PWM channel i is high.

Ci_S[1:0] : Switch PWMi output pin (i=0～7)

For details, please refer to the chapter "Function Foot Switching".

ENI: interrupt enable bit of PWM channel i. (i= 0~7)

0: disable PWM channel i interrupt.

1: enable PWM channel i interrupt.

ENT2I: interrupt enable bit of the second flip point of PWM channel i. (i= 0~7)

0: disable the interrupt of the second flip point of PWM channel i.

1: enable the interrupt of the second flip point of PWM channel i.

ENT1I: interrupt enable bit of the first flip point of PWM channel i. (i= 0~7)

0: disable the interrupt of the first flip point of PWM channel i.

1: enable the interrupt of the first flip point of PWM channel i.

19.2.10 PWM Channel Level Holding Control Registers (PWMnHLD)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
PWM0HLD	FF15H	-	-	-	-	-	-	HLDH	HLDL
PWM1HLD	FF1DH	-	-	-	-	-	-	HLDH	HLDL
PWM2HLD	FF25H	-	-	-	-	-	-	HLDH	HLDL
PWM3HLD	FF2DH	-	-	-	-	-	-	HLDH	HLDL
PWM4HLD	FF35H	-	-	-	-	-	-	HLDH	HLDL
PWM5HLD	FF3DH	-	-	-	-	-	-	HLDH	HLDL
PWM6HLD	FF45H	-	-	-	-	-	-	HLDH	HLDL
PWM7HLD	FF4DH	-	-	-	-	-	-	HLDH	HLDL

HLDH: PWM channel i outputs high compulsively control bit. (i= 0~7)

0: PWM channel i output normally.

1: PWM channel i outputs high compulsively.

HLDL: PWM channel i outputs low compulsively control bit. (i= 0~7)

0: PWM channel i output normally.

1: PWM channel i outputs low compulsively.

19.3 Example Routines

19.3.1 Output waveforms with arbitrary period and arbitrary duty

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr         P_SW2           =   0xba;

sfr         PWMSET          =   0xF1;
sfr         PWMCFG1         =   0xF6;

#define     PWMC                (*(unsigned int  volatile xdata *)0xFF00)
#define     PWMCH               (*(unsigned char volatile xdata *)0xFF00)
#define     PWMCL               (*(unsigned char volatile xdata *)0xFF01)
#define     PWMCKS              (*(unsigned char volatile xdata *)0xFF02)
#define     PWMTADC             (*(unsigned int  volatile xdata *)0xFF03)
#define     PWMTADCH            (*(unsigned char volatile xdata *)0xFF03)
#define     PWMTADCL            (*(unsigned char volatile xdata *)0xFF04)
#define     PWMIF               (*(unsigned char volatile xdata *)0xFF05)
#define     PWMFDCR             (*(unsigned char volatile xdata *)0xFF06)
#define     PWM0T1              (*(unsigned int  volatile xdata *)0xFF10)
#define     PWM0T1H             (*(unsigned char volatile xdata *)0xFF10)
#define     PWM0T1L             (*(unsigned char volatile xdata *)0xFF11)
#define     PWM0T2H             (*(unsigned char volatile xdata *)0xFF12)
#define     PWM0T2              (*(unsigned int  volatile xdata *)0xFF12)
#define     PWM0T2L             (*(unsigned char volatile xdata *)0xFF13)
#define     PWM0CR              (*(unsigned char volatile xdata *)0xFF14)
#define     PWM0HLD             (*(unsigned char volatile xdata *)0xFF15)

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

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

```
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMSET = 0x01;    //Enable PWM0 module (The configuration is effective only after the module is enabled.)

    P_SW2 = 0x80;
    PWMCKS = 0x00;                                  //The clock of PWM is the system clock
    PWMC = 0x1000;                                  //Set the PWM0 cycle  to 1000H PWM clocks
    PWM0T1= 0x0100;             //At the count value of 100H, the PWM0 channel outputs low level.
    PWM0T2= 0x0500;             //At the count value of 500H, the PWM0 channel outputs high level.
    PWM0CR= 0x80;                                   //enable PWM00 output
    P_SW2 = 0x00;

    PWMCFG = 0x01;                                  //Start PWM module

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA        0BAH

PWMSET      DATA        0F1H
PWMCFG      DATA        0F6H

PWMCH       EQU         0FF00H
PWMCL       EQU         0FF01H
PWMCKS      EQU         0FF02H
PWMTADCH    EQU         0FF03H
PWMTADCL    EQU         0FF04H
PWMIF       EQU         0FF05H
PWMFDCR     EQU         0FF06H
PWM0T1H     EQU         0FF10H
PWM0T1L     EQU         0FF11H
PWM0T2H     EQU         0FF12H
PWM0T2L     EQU         0FF13H
PWM0CR      EQU         0FF14H
PWM0HLD     EQU         0FF15H

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

            ORG         0000H
```

```
        LJMP    MAIN

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

        MOV     PWMSET,#01H     ;Enable PWM module (The configuration is effective only after the module is
enabled.)

        MOV     P_SW2,#80H
        CLR     A
        MOV     DPTR,#PWMCKS
        MOVX    @DPTR,A                 ;The clock of PWM is the system clock
        MOV     A,#10H
        MOV     DPTR,#PWMCH             ;Set the PWM cycle  to 1000H PWM clocks
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWMCL
        MOVX    @DPTR,A
        MOV     A,#01H
        MOV     DPTR,#PWM0T1H  ;At the count value of 100H, the PWM0 channel outputs low level.
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T1L
        MOVX    @DPTR,A
        MOV     A,#05H
        MOV     DPTR,#PWM0T2H ;At the count value of 500H, the PWM0 channel outputs high level.
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T2L
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM0CR            ;enable PWM0 output
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG,#01H             ;Start PWM module

        JMP     $

        END
```

19.3.2 Two-channel PWMs realize complementary symmetrical waveform with dead-time control

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr         P_SW2       =   0xba;

sfr         PWMSET      =   0xF1;
sfr         PWMCFG      =   0xF6;

#define     PWMC            (*(unsigned int  volatile xdata *)0xFF00)
#define     PWMCH           (*(unsigned char volatile xdata *)0xFF00)
#define     PWMCL           (*(unsigned char volatile xdata *)0xFF01)
#define     PWMCKS          (*(unsigned char volatile xdata *)0xFF02)
#define     PWMTADC         (*(unsigned int  volatile xdata *)0xFF03)
#define     PWMTADCH        (*(unsigned char volatile xdata *)0xFF03)
#define     PWMTADCL        (*(unsigned char volatile xdata *)0xFF04)
#define     PWMIF           (*(unsigned char volatile xdata *)0xFF05)
#define     PWMFDCR         (*(unsigned char volatile xdata *)0xFF06)
#define     PWM0T1          (*(unsigned int  volatile xdata *)0xFF10)
#define     PWM0T1H         (*(unsigned char volatile xdata *)0xFF10)
#define     PWM0T1L         (*(unsigned char volatile xdata *)0xFF11)
#define     PWM0T2          (*(unsigned int  volatile xdata *)0xFF12)
#define     PWM0T2H         (*(unsigned char volatile xdata *)0xFF12)
#define     PWM0T2L         (*(unsigned char volatile xdata *)0xFF13)
#define     PWM0CR          (*(unsigned char volatile xdata *)0xFF14)
#define     PWM0HLD         (*(unsigned char volatile xdata *)0xFF15)
#define     PWM1T1          (*(unsigned int  volatile xdata *)0xFF18)
#define     PWM1T1H         (*(unsigned char volatile xdata *)0xFF18)
#define     PWM1T1L         (*(unsigned char volatile xdata *)0xFF19)
#define     PWM1T2          (*(unsigned int  volatile xdata *)0xFF1A)
#define     PWM1T2H         (*(unsigned char volatile xdata *)0xFF1A)
#define     PWM1T2L         (*(unsigned char volatile xdata *)0xFF1B)
#define     PWM1CR          (*(unsigned char volatile xdata *)0xFF1C)
#define     PWM1HLD         (*(unsigned char volatile xdata *)0xFF1D)

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

void main()
```

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

    PWMSET = 0x01;                  //Enable PWM module (The configuration is effective only after
the module is enabled.)

    P_SW2 = 0x80;
    PWMCKS = 0x00;                  //The clock of PWM is the system clock
    PWMC = 0x0800;                  //Set the PWM cycle to 0800H PWM clocks
    PWM0T1= 0x0100;                 //At the count value of 100H, PWM0 outputs low level.
    PWM0T2= 0x0700;                 //At the count value of 700H, PWM0 outputs high level.
    PWM1T2= 0x0080;                 //At the count value of 0080H, PWM1 outputs high level.
    PWM1T1= 0x0780;                 //At the count value of 0780H, PWM1 outputs low level.
    PWM0CR= 0x80;                   //enable PWM0 output
    PWM1CR= 0x80;                   //enable PWM1 output
    P_SW2 = 0x00;

    PWMCFG = 0x01;                  //Start PWM module

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG      DATA    0F6H

PWMCH       EQU     0FF00H
PWMCL       EQU     0FF01H
PWMCKS      EQU     0FF02H
PWMTADCH    EQU     0FF03H
PWMTADCL    EQU     0FF04H
PWMIF       EQU     0FF05H
PWMFDCR     EQU     0FF06H
PWM0T1H     EQU     0FF10H
PWM0T1L     EQU     0FF11H
PWM0T2H     EQU     0FF12H
PWM0T2L     EQU     0FF13H
PWM0CR      EQU     0FF14H
PWM0HLD     EQU     0FF15H
PWM1T1H     EQU     0FF18H
PWM1T1L     EQU     0FF19H
PWM1T2H     EQU     0FF1AH
```

```
PWM1T2L     EQU     0FF1BH
PWM1CR      EQU     0FF1CH
PWM1HLD     EQU     0FF1DH

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     PWMSET,#01H         ;Enable PWM module (The configuration is effective only after the
module is enabled.)

            MOV     P_SW2,#80H
            CLR     A
            MOV     DPTR,#PWMCKS
            MOVX    @DPTR,A             ;The clock of PWM0 is the system clock
            MOV     A,#08H
            MOV     DPTR,#PWMCH         ;Set the PWM0 cycle to 0800H PWM clocks
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWMCL
            MOVX    @DPTR,A
            MOV     A,#01H
            MOV     DPTR,#PWM0T1H       ;At the count value of 0100H, PWM0 outputs low level.
            MOVX    @DPTR,A
            MOV     A,#00H
            MOV     DPTR,#PWM0T1L
            MOVX    @DPTR,A
            MOV     A,#07H
            MOV     DPTR,#PWM0T2H       ;At the count value of 0700H, PWM0 outputs high level.
            MOVX    @DPTR,A
```

```
        MOV     A,#00H
        MOV     DPTR,#PWM0T2L
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM1T2H       ;At the count value of 0080H, PWM1 outputs high level.
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM1T2L
        MOVX    @DPTR,A
        MOV     A,#07H
        MOV     DPTR,#PWM1T1H       ;At the count value of 0780H, PWM1 outputs low level.
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM1T1L
        MOVX    @DPTR,A
        MOV     A,#080H
        MOV     DPTR,#PWM0CR        ;enable PWM0 output
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM1CR        ;enable PWM1 output
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG,#01H         ;Start PWM module

        JMP     $

        END
```

19.3.3 PWM Implements Gradient Light (Breathing Light)

C language code

```
//Operating frequency for test is 11.0592MHz

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

#define   CYCLE             0x1000

sfr       P_SW2       =     0xba;

sfr       PWMSET      =     0xF1;
sfr       PWMCFG      =     0xF6;

#define   PWMC              (*(unsigned int  volatile xdata *)0xFF00)
#define   PWMCH             (*(unsigned char volatile xdata *)0xFF00)
#define   PWMCL             (*(unsigned char volatile xdata *)0xFF01)
#define   PWMCKS            (*(unsigned char volatile xdata *)0xFF02)
#define   PWMTADC           (*(unsigned int  volatile xdata *)0xFF03)
#define   PWMTADCH          (*(unsigned char volatile xdata *)0xFF03)
#define   PWMTADCL          (*(unsigned char volatile xdata *)0xFF04)
#define   PWMIF             (*(unsigned char volatile xdata *)0xFF05)
#define   PWMFDCR           (*(unsigned char volatile xdata *)0xFF06)
#define   PWM0T1            (*(unsigned int  volatile xdata *)0xFF10)
```

```
#define   PWM0T1H       (*(unsigned char volatile xdata *)0xFF10)
#define   PWM0T1L       (*(unsigned char volatile xdata *)0xFF11)
#define   PWM0T2H       (*(unsigned char volatile xdata *)0xFF12)
#define   PWM0T2        (*(unsigned int  volatile xdata *)0xFF12)
#define   PWM0T2L       (*(unsigned char volatile xdata *)0xFF13)
#define   PWM0CR        (*(unsigned char volatile xdata *)0xFF14)
#define   PWM0HLD       (*(unsigned char volatile xdata *)0xFF15)

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

void PWM0_Isr() interrupt 22
{
    static bit dir = 1;
    static int val = 0;

    if (PWMCFG & 0x08)
    {
        PWMCFG &= ~0x08;                    //Clear interrupt flag
        if (dir)
        {
            val++;
            if (val >= CYCLE) dir = 0;
        }
        else
        {
            val--;
            if (val <= 1) dir = 1;
        }
        _push_(P_SW2);
        P_SW2 |= 0x80;
        PWM0T2 = val;
        _pop_(P_SW2);
    }
}

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

```
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMSET = 0x01;                          //Enable PWM module (The configuration is effective only after
the module is enabled.)

    P_SW2 = 0x80;
    PWMCKS = 0x00;                          // The clock of PWM is the system clock
    PWMC = CYCLE;                           //Set PWM period
    PWM0T1= 0x0000;
    PWM0T2= 0x0001;
    PWM0CR= 0x80;                           //enable PWM0 output
    P_SW2 = 0x00;

    PWMCFG = 0x05;                          //Start PWM module and enable PWM interrput
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

CYCLE       EQU     1000H

P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG      DATA    0F6H

PWMCH       EQU     0FF00H
PWMCL       EQU     0FF01H
PWMCKS      EQU     0FF02H
PWMTADCH    EQU     0FF03H
PWMTADCL    EQU     0FF04H
PWMIF       EQU     0FF05H
PWMFDCR     EQU     0FF06H
PWM0T1H     EQU     0FF10H
PWM0T1L     EQU     0FF11H
PWM0T2H     EQU     0FF12H
PWM0T2L     EQU     0FF13H
PWM0CR      EQU     0FF14H
PWM0HLD     EQU     0FF15H

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

```
DIR             BIT             20H.0
VALL            DATA            21H
VALH            DATA            22H

                ORG             0000H
                LJMP            MAIN
                ORG             00B3H
                LJMP            PWM0ISR

                ORG             0100H
PWM0ISR:
                PUSH            ACC
                PUSH            PSW
                PUSH            DPL
                PUSH            DPH
                PUSH            P_SW2

                MOV             P_SW2,#80H
                MOV             A,PWMCFG
                JNB             ACC.3,ISREXIT
                ANL             PWMCFG,#NOT 08H         ;Clear interrupt flag
                JNB             DIR,PWMDN
PWMUP:
                MOV             A,VALL
                ADD             A,#1
                MOV             VALL,A
                MOV             A,VALH
                ADDC            A,#0
                MOV             VALH,A
                CJNE            A,#HIGH CYCLE,SETPWM
                MOV             A,VALL
                CJNE            A,#LOW CYCLE,SETPWM
                CLR             DIR
                JMP             SETPWM
PWMDN:
                MOV             A,VALL
                ADD             A,#0FFH
                MOV             VALL,A
                MOV             A,VALH
                ADDC            A,#0FFH
                MOV             VALH,A
                JNZ             SETPWM
                MOV             A,VALL
                CJNE            A,#1,SETPWM
                SETB            DIR
SETPWM:
                MOV             A,VALH
                MOV             DPTR,#PWM0T2H
                MOVX            @DPTR,A
                MOV             A,VALL
                MOV             DPTR,#PWM0T2L
                MOVX            @DPTR,A
ISREXIT:
                POP             P_SW2
                POP             DPH
                POP             DPL
                POP             PSW
                POP             ACC
```

```
        RETI

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

        SETB    DIR
        MOV     VALH,#00H
        MOV     VALL,#01H

        MOV     PWMSET,#01H             ;Enable PWM module (The configuration is effective only after the
module is enabled.)

        MOV     P_SW2,#80H
        CLR     A
        MOV     DPTR,#PWMCKS
        MOVX    @DPTR,A                 ;The clock of PWM is the system clock
        MOV     A,#HIGH CYCLE
        MOV     DPTR,#PWMCH             ;Set PWM period
        MOVX    @DPTR,A
        MOV     A,#LOW CYCLE
        MOV     DPTR,#PWMCL
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T1H
        MOVX    @DPTR,A
        MOV     A,#00H
        MOV     DPTR,#PWM0T1L
        MOVX    @DPTR,A
        MOV     A,VALH
        MOV     DPTR,#PWM0T2H
        MOVX    @DPTR,A
        MOV     A,VALL
        MOV     DPTR,#PWM0T2L
        MOVX    @DPTR,A
        MOV     A,#80H
        MOV     DPTR,#PWM0CR            ;enable PWM0 output
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG,#05H             ;Start PWM module and enable PWM interrupt
        SETB    EA
        JMP     $

        END
```

19.3.4 Use PWM to trigger ADC conversion

C language code

```
//Operating frequency for test is 11.0592MHz

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

sfr         P_SW2           =   0xba;

sfr         PWMSET          =   0xF1;
sfr         PWMCFG          =   0xF6;

#define     PWMC                (*(unsigned int  volatile xdata *)0xFF00)
#define     PWMCH               (*(unsigned char volatile xdata *)0xFF00)
#define     PWMCL               (*(unsigned char volatile xdata *)0xFF01)
#define     PWMCKS              (*(unsigned char volatile xdata *)0xFF02)
#define     PWMTADC             (*(unsigned int  volatile xdata *)0xFF03)
#define     PWMTADCH            (*(unsigned char volatile xdata *)0xFF03)
#define     PWMTADCL            (*(unsigned char volatile xdata *)0xFF04)
#define     PWMIF               (*(unsigned char volatile xdata *)0xFF05)
#define     PWMFDCR             (*(unsigned char volatile xdata *)0xFF06)
#define     PWM0T1              (*(unsigned int  volatile xdata *)0xFF10)
#define     PWM0T1H             (*(unsigned char volatile xdata *)0xFF10)
#define     PWM0T1L             (*(unsigned char volatile xdata *)0xFF11)
#define     PWM0T2H             (*(unsigned char volatile xdata *)0xFF12)
#define     PWM0T2              (*(unsigned int  volatile xdata *)0xFF12)
#define     PWM0T2L             (*(unsigned char volatile xdata *)0xFF13)
#define     PWM0CR              (*(unsigned char volatile xdata *)0xFF14)
#define     PWM0HLD             (*(unsigned char volatile xdata *)0xFF15)

sfr         ADC_CONTR       =   0xbc;
#define     ADC_POWER           0x80
#define     ADC_START           0x40
#define     ADC_FLAG            0x20
#define     ADC_EPWMT           0x10
sfr         ADC_RES         =   0xbd;
sfr         ADC_RESL        =   0xbe;

sbit        EADC            =   IE^5;

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

void delay()
```

```
{
    int i;
    for (i=0; i<100; i++);
}

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

    ADC_CONTR = ADC_POWER | ADC_EPWMT | 0;       // Select P1.0 as the ADC input channel
    delay();                    // Wait for the ADC power supply to stabilize
    EADC = 1;

     PWMSET = 0x01;                                          //Enable PWM module (The configuration is effective only after the module is enabled.)

     P_SW2 = 0x80;
     PWMCKS = 0x00;                                          // The clock of PWM is the system clock
      PWMC = 0x1000;     // Set the period of PWM to 1000H PWM clocks
    PWM0T1= 0x0100;     // When the count value is 100H, the PWM0 channel outputs low level
    PWM0T2= 0x0500;     // When the count value is 500H, the PWM0 channel outputs high level
    PWMTADC = 0x0200; // Set ADC trigger point
     PWM0CR= 0x80;                                           //Enable PWM0 output
     P_SW2 = 0x00;

     PWMCFG = 0x07;      // Start the PWM module and enable the PWM interrupt and ADC trigger
     EA = 1;

     while (1);
}

void pwm0_isr() interrupt 22
{
    if (PWMCFG & 0x08)
    {
        PWMCFG &= ~0x08;
    }
}

void ADC_ISR() interrupt 5
{
    ADC_CONTR &= ~ADC_FLAG;
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz
```

```
P_SW2       DATA    0BAH

PWMSET      DATA    0F1H
PWMCFG      DATA    0F6H

PWMCH       EQU     0FF00H
PWMCL       EQU     0FF01H
PWMCKS      EQU     0FF02H
PWMTADCH    EQU     0FF03H
PWMTADCL    EQU     0FF04H
PWMIF       EQU     0FF05H
PWMFDCR     EQU     0FF06H
PWM0T1H     EQU     0FF10H
PWM0T1L     EQU     0FF11H
PWM0T2H     EQU     0FF12H
PWM0T2L     EQU     0FF13H
PWM0CR      EQU     0FF14H
PWM0HLD     EQU     0FF15H

ADC_CONTR   DATA    0BCH
ADC_POWER   EQU     080H
ADC_START   EQU     040H
ADC_FLAG    EQU     020H
ADC_EPWMT   EQU     010H
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH

EADC        BIT     IE.5

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

            ORG     0000H
            LJMP    MAIN
            ORG     002BH
            LJMP    ADCISR
            ORG     00B3H
            LJMP    PWM0ISR

            ORG     0100H
ADCISR:
            ANL     ADC_CONTR,#NOT ADC_FLAG
            RETI

PWM0ISR:
            PUSH    ACC
            MOV     A,PWMCFG
```

```
                JNB     ACC.3,ISREXIT
                ANL     PWMCFG,#NOT 08H
ISREXIT:
                POP     ACC
                RETI

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

                MOV     ADC_CONTR,# ADC_POWER | ADC_EPWMT
                SETB    EADC

                MOV     PWMSET,#01H             ;Enable PWM module (The configuration is effective only after the module is enabled.)

                MOV     P_SW2,#80H
                CLR     A
                MOV     DPTR,#PWMCKS
                MOVX    @DPTR,A                 ; The clock of PWM is the system clock
                MOV     A,#10H
                MOV     DPTR,#PWMCH             ; Set the period of PWM to 1000H PWM clocks
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWMCL
                MOVX    @DPTR,A
                MOV     A,#01H
                MOV     DPTR,#PWM0T1H           ; When the count value is 100H, the PWM0 channel outputs low level
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWM0T1L
                MOVX    @DPTR,A
                MOV     A,#05H
                MOV     DPTR,#PWM0T2H           ; When the count value is 500H, the PWM0 channel outputs high level
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWM0T2L
                MOVX    @DPTR,A
                MOV     A,#02H
                MOV     DPTR,#PWMTADCH          ; Set ADC trigger point
                MOVX    @DPTR,A
                MOV     A,#00H
                MOV     DPTR,#PWMTADCL
                MOVX    @DPTR,A
                MOV     A,#80H
```

```
        MOV     DPTR,#PWM0CR        ; Enable PWM0 output
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        MOV     PWMCFG,#07H         ; Start the PWM module and enable the PWM interrupt and ADC
trigger
        SETB    EA
        JMP     $

        END
```

20 Sysnchronous Serial Peripheral Interface (SPI)

A high-speed serial communication interface, SPI, is integrated in STC8A8K64D4 series of microcontrollers. SPI is a full-duplex high-speed synchronous communication bus. SPI interface integrated in the STC8A8K64D4 series of microcontrollers offers two operation modes: master mode and slave mode.

20.1 SPI function pin switch

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CCP_S[1:0]		SPI_S[1:0]		0	-

SPI_S[1:0]: SPI function pin selection bit

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P7.4	P7.5	P7.6	P7.7
11	P3.5	P3.4	P3.3	P3.2

20.2 Registers Related to SPI

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

20.2.1 SPI Status register (SPSTAT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF: SPI transfer completion flag.

When SPI completes sending / receiving 1 byte of data, the hardware will automatically set this bit and request interrupt to CPU. If the SSIG bit is set to 0, this flag will also be automatically set by hardware to indicate a mode change of device when the master / slave mode of the device changes due to changes in the SS pin level.

Note: This bit must be cleared using software writing 1 to it.

WCOL: SPI write collision flag bit.

This bit is set by hardware when the SPI is writing to the SPDAT register during data transfer.

Note: This bit must be cleared using software by writing 1 to it.

20.2.2 SPI Control Register (SPCTL)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG: Control bit of whether SS pin is ignored or not.

0: the SS pin decides whether the device is a master or slave.

1: the function of SS pin is ignored. MSTR decides whether the device is a master or slave.

SPEN: SPI enable bit.

0: the SPI is disabled.

1: the SPI is enabled.

DORD: Set the transmitted or received SPI data order.

0: The MSB of the data is transmitted firstly.

1: The LSB of the data is transmitted firstly.

MSTR: Master/Slave mode select bit.

To set the mastert mode:

If SSIG = 0, the SS pin must be high and set MSTR to 1.

If SSIG = 1, it only needs to set MSTR to 1 (ignoring the SS pin level).

To set the slave mode:

If SSIG = 0, the SS pin must be low (regardless of the MSTR bit).

If SSIG = 1, it only needs to set MSTR to 0 (ignoring the SS pin level).

CPOL: SPI clock polarity select bit.

0：SCLK is low when idle. The leading edge of SCLK is the rising edge and the trailing edge is the falling edge.

1：SCLK is high when idle. The leading edge of SCLK is the falling edge and the trailing edge is the rising edge.

CPHA: SPI clock phase select bit.

0: The first bit of datum is driven when SS pin is low. The datum changes on the trailing edge of SCLK and is sampled on the leading edge of SCLK. (SSIG must be 0.)

1: The datum is driven on the leading edge of SCLK, and is sampled on the trailing edge.

SPR[1:0]: SPI clock frequency select bits

SPR[1:0]	SCLK frequency
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/2

20.2.3 SPI Data Register (SPDAT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

The SPDAT holds the data to be transmitted or the data received.

20.3 SPI Communication Modes

There are three SPI communication modes: single master and single slave mode, dual devices configuration mode (any one of them can be a master or slave), single master and multiple slaves mode.

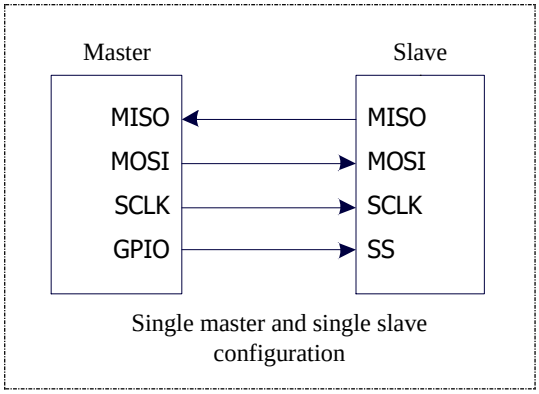
20.3.1 Single Master and Single Slave Mode

Two devices are connected, one of which is fixed as a master and the other as a slave.

Master settings: SSIG set to 1, MSTR set to 1, fixed to be master mode. The master can use any port to connect the slave SS pin, pull down the slave SS pin to enable the slave.

Slave settings: SSIG is set to 0, SS pin as the chip select signal of the slave.

Single master single slave connection configuration diagram is shown as follows:



Single master and single slave configuration

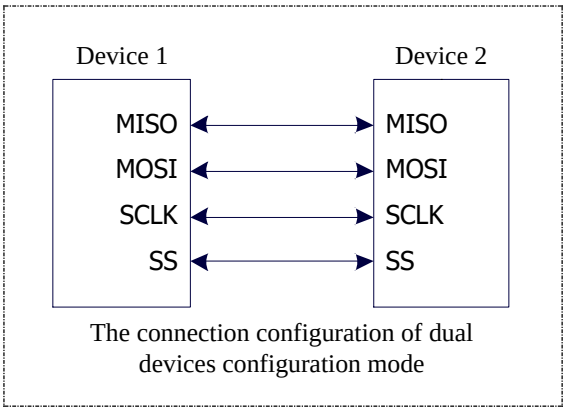
20.3.2 Dual Devices Configuration Mode

Two devices are connected, the master and the slave are not fixed.

Setting Method 1: Both devices are initialized with SSIG set to 0, MSTR set to 1, and SS pin set to bi-directional mode and output high. Now the both devices are in master mode with not ignoring SS. When one of the devices needs to initiate a transfer, set its own SS pin to output mode and output low to pull down the other device's SS pin so that the other device is forcibly set to slave mode.

Set Method 2: Both devices are initialized as slave mode with ignoring SS, where SIG is set to 1 and MSTR is set to 0. When one of the devices needs to initiate a transfer, detect the SS pin's level firstly. If SS is high, the device sets itself to master mode with ignoring SS, then starts the data transfer.

The connection configuration of dual devices configuration mode is shown as follows:



The connection configuration of dual devices configuration mode

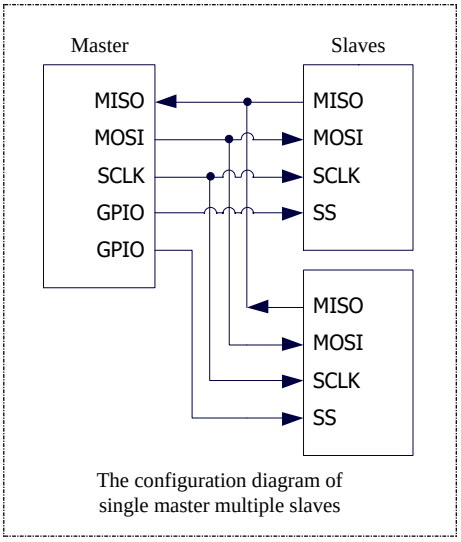
20.3.3 Single Master and Multiple Slaves Mode

Multiple devices are connected, one of which is fixed as a master and others are fixed as slaves.

Master settings: SSIG set to 1, MSTR set to 1, fixed to master mode. The master can use any port to connect with the SS pins of each slave respectively, and pull down the SS pin of one slave to enable the corresponding slave device.

Slave settings: SSIG is set to 0, SS pin is used as the chip select signal of the slave.

The configuration diagram of single master multiple slaves is as follows:



The configuration diagram of single master multiple slaves

20.4 Configure SPI

Control bits			Communication port pins				Descriptions
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	input	input	input	SPI is disabled, SS/MOSI/MISO/SCLK are used as general I/O ports
1	0	0	0	output	input	input	**Selected as slave**
1	0	0	1	high impedance	input	input	**Selected as slave,** not selected.
1	0	1→0	0	output	input	input	**Slavce mode**, master mode with notignoring SS and MSTR is 1. When SS pin is pulled low, MSTR will be automatically cleared by hardware and the operating mode will be passively set to slave mode.
1	0	1	1	input	high impedance	high impedance	**Master mode,** idle state
					output	output	**Master mode, ative state**
1	1	0	x	output	input	input	**Slave mode**
1	1	1	x	input	output	output	**Master mode**

Considerations for a Slave

When CPHA = 0, SSIG must be 0 (i.e. SS pin can not be ignored). The SS pin must be pulled low before each serial byte begins transfer and must be reset to high after the transfer completes. The SPDAT register can not be written while the SS pin is low, otherwise a write collision error will occur. Operation with CPHA = 0 and SSIG = 1 is undefined.

When CPHA = 1, SSIG may be set to 1 (i.e. the SS pin can be ignored). If SSIG = 0, the SS pin may remain active low (i.e., stay low all the way) for consecutive transfers. This method is suitable for fixed single master single slave system.

Considerations for a Master

In SPI, transfers are always initiated by the master. If the SPI is enabled (SPEN = 1) and selected as the master, the master will initiate SPI clock generator and data transfer by writing to SPI data register, SPDAT. The data will appear on the MOSI pin a half to one SPI bit-time later after the data is written to SPDAT. The data written to the SPDAT register of the master is shifted out from the MOSI pin and sent to the MOSI pin of the slave. And, at the same time the data in SPDAT register of the selected slave is shifted out on MISO pin to the MISO pin of the master.

After one byte has been transmitted, the SPI clock generator is stopped, the transfer completion flag (SPIF) is set, and an SPI interrupt is generated if the SPI interrupt is enabled. The two shift registers for the master and slave CPUs

can be considered as a 16-bit cyclic shift register. As data is shifted from the master to the slave, data is also shifted in the opposite direction simultaneously. This means that the data of the master and the slave are exchanged with each other in one shift period.

Change Mode using SS pin

If SPEN = 1, SSIG = 0 and MSTR = 1, SPI is enabled in master mode and the SS pin can be configured for input mode or quasi-bidirectional port mode. In this case, another master can drive this pin low to select the device as an SPI slave and send data to it. To avoid bus contention, the SPI system clears the slave's MSTR, forces MOSI and SCLK to be input mode, and MISO changes to output mode. The SPIF flag in SPSTAT is set, and if the SPI interrupt is enabled, an SPI interrupt will occur.

The user software must always detect the MSTR bit. If this bit is cleared by a slave selection action and the user wants to continue using the SPI as a master, the MSTR bit must be set again, otherwise it will remain in slave mode.

Write Collision

The SPI is single buffered in the transmition process and double buffered in receiving process. New data for transmission can not be written to the shift register until the previous transmission is complete. The WCOL bit will be set to indicate that a data write collision error has occurred when the data register SPDAT is written during transmission. In this case, the data currently being transmitted will continue to be transmitted, and the newly written data will be lost.

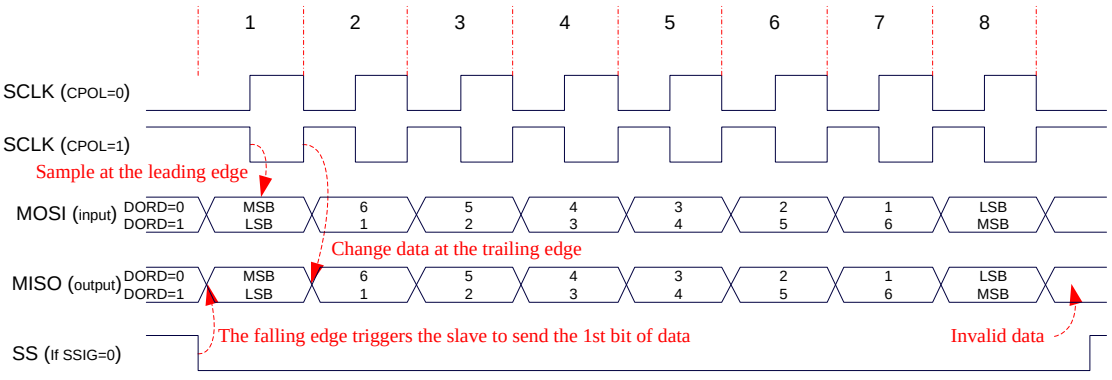
A write collision condition on the master is rare when write collision detection is performed on a master or slave because the master has full control of the data transfer. However, a write collision may occur on the slave because the slave can not control it when the master initiates the transfer.

When receiving data, the received data is transferred to a parallel read data buffer, which will release the shift register for the next data reception. However, the received data must be read from the data register before the next character is completely shifted in. Otherwise, the previous received data will be lost.

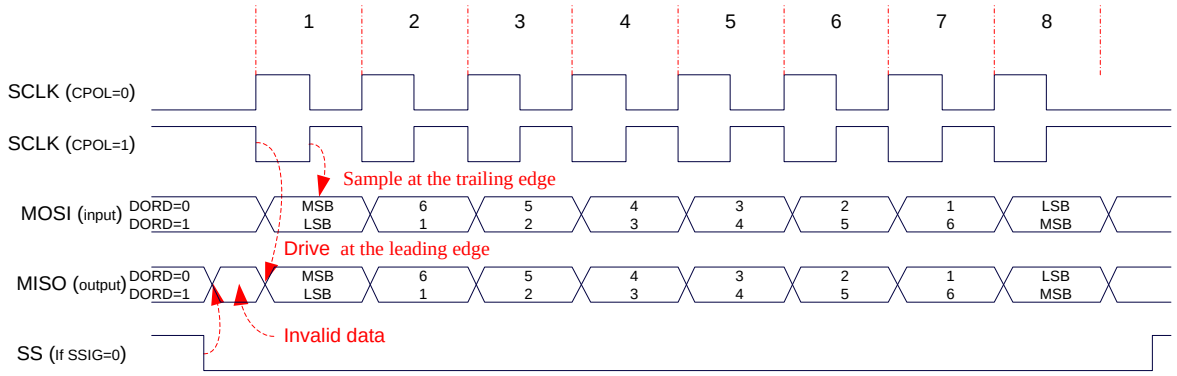
WCOL can be cleared by software by writing "1" to it.

20.5 Data Format

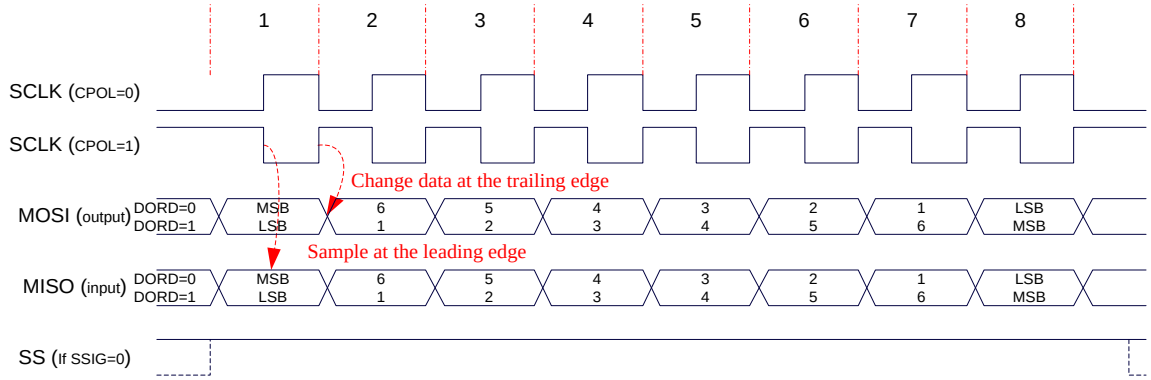
The clock phase control bit, CPHA, of the SPI allows the user to set the clock edge when the data is sampled and changed. The clock polarity bit, CPOL, allows the user to set the clock polarity. The following illustrations show the SPI communication timing under different clock phases and polarity settings.



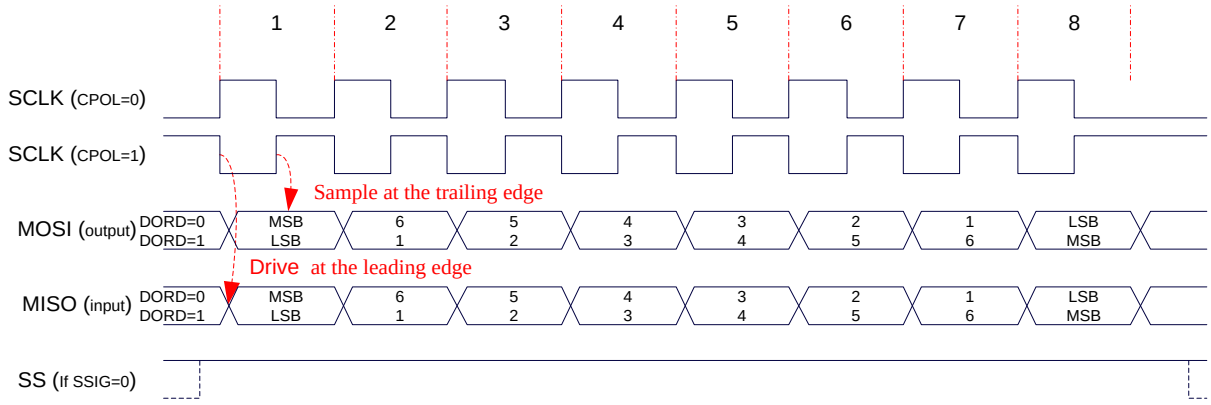
SPI slave transfer format with CPHA=0



SPI slave transfer format with CPHA=1



SPI master transfer format with CPHA=0



SPI master transfer format with CPHA=1

20.6 Example Routines

20.6.1 Master Routine of Single Master Single Slave Mode (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz
```

```
#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;

bit         busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //Clear interrupt flag
    SS = 1;                         //Pull up the SS pin of the slave
    busy = 0;
    LED = !LED;                     //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;                   //Enable SPI master mode
    SPSTAT = 0xc0;                  //Clear interrupt flag
    IE2 = ESPI;                     //Enable SPI interrupt
    EA = 1;
```

```
    while (1)
    {
        while (busy);
        busy = 1;
        SS = 0;                                 //Pull down the slave SS pin
        SPDAT = 0x5a;                           //Send test data
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

BUSY        BIT     20H.0
SS          BIT     P1.0
LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            SETB    SS                  ;Pull up the SS pin of the slave
            CLR     BUSY
            CPL     LED
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
```

```
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                SETB    LED
                SETB    SS
                CLR     BUSY

                MOV     SPCTL,#50H          ;Enable SPI master mode
                MOV     SPSTAT,#0C0H        ;Clear interrupt flag
                MOV     IE2,#ESPI           ;Enable SPI interrupt
                SETB    EA

LOOP:
                JB      BUSY,$
                SETB    BUSY
                CLR     SS                  ;Pull down the slave SS pin
                MOV     SPDAT,#5AH          ;Send test data
                JMP     LOOP

                END
```

20.6.2 Slave Routine of Single Master Single Slave Mode (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    LED         =   P1^1;

void SPI_Isr() interrupt 9
```

```
{
    SPSTAT = 0xc0;                          //Clear interrupt flag
    SPDAT = SPDAT;                          //Transmit the received data back to the master
    LED = !LED;                             //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                           //Enable SPI slave mode
    SPSTAT = 0xc0;                          //Clear interrupt flag
    IE2 = ESPI;                             //Enable SPI interrupt
    EA = 1;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR
```

```
            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            MOV     SPDAT,SPDAT         ;Transmit the received data back to the master
            CPL     LED
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;Enable SPI slave mode
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            MOV     IE2,#ESPI           ;Enable SPI interrupt
            SETB    EA

            JMP     $

            END
```

20.9.3 Master Routine of Single Master Single Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
```

```
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;

    SPCTL = 0x50;                       //Enable SPI master mode
    SPSTAT = 0xc0;                      //Clear interrupt flag

    while (1)
    {
        SS = 0;                         //Pull down the slave SS pin
        SPDAT = 0x5a;                   //Send test data
        while (!(SPSTAT & 0x80));       //Query completion flag
        SPSTAT = 0xc0;                  //Clear interrupt flag
        SS = 1;                         //Pull up the SS pin of the slave
        LED = !LED;                     //Test port
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
```

```
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    LED
            SETB    SS

            MOV     SPCTL,#50H          ;Enable SPI master mode
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag

LOOP:
            CLR     SS                  ;Pull down the slave SS pin
            MOV     SPDAT,#5AH          ;Send test data
            MOV     A,SPSTAT            ;Query completion flag
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            SETB    SS
            CPL     LED
            JMP     LOOP

            END
```

20.6.4 Slave Routine of Single Master Single Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
```

```
sfr         IE2             =   0xaf;
#define     ESPI                0x02

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        LED             =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                                  //Clear interrupt flag
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                                   //Enable SPI slave mode
    SPSTAT = 0xc0;                                  //Clear interrupt flag

    while (1)
    {
        while (!(SPSTAT & 0x80));                   //Query completion flag
        SPSTAT = 0xc0;                              //Clear interrupt flag
        SPDAT = SPDAT;                              //Transmit the received data back to the master
        LED = !LED;                                 //Test port
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA        0CDH
SPCTL       DATA        0CEH
SPDAT       DATA        0CFH
IE2         DATA        0AFH
```

```
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;Enable SPI slave mode
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag

LOOP:
            MOV     A,SPSTAT            ;Query completion flag
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            MOV     SPDAT,SPDAT         ;Transmit the received data back to the master
            CPL     LED
            JMP     LOOP

            END
```

20.6.5 Routine of Mutual Master-Slave Mode (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;
sbit        KEY         =   P0^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                          //Clear interrupt flag
    if (SPCTL & 0x10)
    {                                       //Master mode
        SS = 1;                             //Pull up the SS pin of the slave
        SPCTL = 0x40;                       //Reset to slave and standby
    }
    else
    {                                       //Slave mode
        SPDAT = SPDAT;                      //Transmit the received data back to the master
    }
    LED = !LED;                             //Test port
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;
```

```
    SPCTL = 0x40;                           //Enable SPI slave modeand standby
    SPSTAT = 0xc0;                          //Clear interrupt flag
    IE2 = ESPI;                             //Enable SPI interrupt
    EA = 1;

    while (1)
    {
        if (!KEY)                           //Wait for the key to trigger
        {
            SPCTL = 0x50;                   //Enable SPI master mode
            SS = 0;                         //Pull down the slave SS pin
            SPDAT = 0x5a;                   //Send test data
            while (!KEY);                   //Wait for the keys to be released
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            PUSH    ACC
            MOV     SPSTAT,#0C0H        ;Clear interrupt flag
            MOV     A,SPCTL
            JB      ACC.4,MASTER
SLAVE:
            MOV     SPDAT,SPDAT         ;Transmit the received data back to the master
```

```
            JMP     ISREXIT
MASTER:
            SETB    SS              ;Pull up the SS pin of the slave
            MOV     SPCTL,#40H      ;Reset to slave and standby
ISREXIT:
            CPL     LED
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    SS
            SETB    LED
            SETB    KEY

            MOV     SPCTL,#40H      ;Enable SPI slave mode and standby
            MOV     SPSTAT,#0C0H    ;Clear interrupt flag
            MOV     IE2,#ESPI       ;Enable SPI interrupt
            SETB    EA

LOOP:
            JB      KEY,LOOP        ;Wait for the key to trigger
            MOV     SPCTL,#50H      ;Enable SPI master mode
            CLR     SS              ;Pull down the slave SS pin
            MOV     SPDAT,#5AH      ;Send test data
            JNB     KEY,$           ;Wait for the keys to be released
            JMP     LOOP

            END
```

20.6.6 Routine of Mutual Master-Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
```

```
#define   ESPI          0x02

sfr       P0M1    =     0x93;
sfr       P0M0    =     0x94;
sfr       P1M1    =     0x91;
sfr       P1M0    =     0x92;
sfr       P2M1    =     0x95;
sfr       P2M0    =     0x96;
sfr       P3M1    =     0xb1;
sfr       P3M0    =     0xb2;
sfr       P4M1    =     0xb3;
sfr       P4M0    =     0xb4;
sfr       P5M1    =     0xc9;
sfr       P5M0    =     0xca;

sbit      SS      =     P1^0;
sbit      LED     =     P1^1;
sbit      KEY     =     P0^0;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                       //Enable SPI slave mode and standby
    SPSTAT = 0xc0;                      //Clear interrupt flag

    while (1)
    {
        if (!KEY)                       //Wait for the key to trigger
        {
            SPCTL = 0x50;               //Enable SPI master mode
            SS = 0;                     //Pull down the slave SS pin
            SPDAT = 0x5a;               //Send test data
            while (!KEY);               //Wait for the keys to be released
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;              //Clear interrupt flag
            if (SPCTL & 0x10)
            {                           //Master mode
                SS = 1;                 //Pull up the SS pin of the slave
                SPCTL = 0x40;           //Reset to slave and standby
            }
```

```
            else
            {                                   //Slave mode
                SPDAT = SPDAT;                  //Transmit the received data back to the master
            }
            LED = !LED;                         //Test port
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    SS
            SETB    LED
            SETB    KEY
```

```
        MOV     SPCTL,#40H          ;Enable SPI slave mode and standby
        MOV     SPSTAT,#0C0H        ;Clear interrupt flag

LOOP:
        JB      KEY,SKIP            ;Wait for the key to trigger
        MOV     SPCTL,#50H          ;Enable SPI master mode
        CLR     SS                  ;Pull down the slave SS pin
        MOV     SPDAT,#5AH          ;Send test data
        JNB     KEY,$               ;Wait for the keys to be released
SKIP:
        MOV     A,SPSTAT
        JNB     ACC.7,LOOP
        MOV     SPSTAT,#0C0H        ;Clear interrupt flag
        MOV     A,SPCTL
        JB      ACC.4,MASTER
SLAVE:
        MOV     SPDAT,SPDAT         ;Transmit the received data back to the master
        CPL     LED
        JMP     LOOP
MASTER:
        SETB    SS                  ;Pull up the SS pin of the slave
        MOV     SPCTL,#40H          ;Reset to slave and standby
        CPL     LED
        JMP     LOOP

        END
```

21 I²C Bus

An I²C serial bus controller is integrated in the STC8A8K64D4 series of microcontrollers. I²C is a high-speed synchronous communication bus, which uses SCL (clock line) and SDA (data line) to carry out two-wire synchronous communication. For the pin allocation of SCL and SDA, STC8A8K64D4 series of microcontrollers provide pin switch mode that can switch SCL and SDA to different I/O pins. Therefor, it is convenience to use a set of I²C as multiple sets of I²C buses through time sharing.

Compared with the standard I²C protocol, the following two mechanisms are ignored:

- No arbitration will be performed after the start signal (START) is sent.
- No timeout detection when the clock signal (SCL) stays at low level.

The I²C bus of the STC8A8K64D4 series of microcontrollers offer two modes of operation: master mode (SCL is the output port, which is used to transmit synchronous clock signal) and slave mode (SCL is the input port, which is used to receive the synchronous clock signal).

STC innovation: When the I²C serial bus controller of STC works in slave mode, the falling edge signal of SDA pin can wake up the MCU which is in power-down mode. (Note: Due to the fast I²C transmission speed, the first packet of data after the MCU wakes up is generally incorrect.)

21.1 I2C function pin switch

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S

I2C_S[1:0]: I²C function pin selection bit

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P2.5	P2.4
10	P7.7	P7.6
11	P3.2	P3.3

21.2 Registers Related to I²C

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I2C Configuration Register	FE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I²C Master Control Register	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C Master 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	SLADR[6:0]							MA	0000,0000
I2CTXD	I²C Data Transmission Register	FE86H									0000,0000
I2CRXD	I²C Data Receive Register	FE87H									0000,0000
I2CMSAUX	I²C Master Auxiliary Control Register	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

21.3 I²C Master Mode

21.3.1 I2C Configuration Register (I2CCFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL	MSSPEED[5:0]					

ENI2C: I²C function enable bit

0: disable I²C function

1: enable I²Cfunction

MSSL: I²C mode selection bit

0: Salve mode

1: Master mode

MSSPEED[5:0]: I²C bus speed control bits (clocks to wait), **I2C bus speed＝F_{OSC} / 2 / (MSSPEED * 2 + 4)**

MSSPEED[5:0]	Corresponding clocks
0	4
1	6
2	8
…	…
x	2x+4
…	…
62	128
63	130

The waiting parameter set by the MSSPEED is valid only when the I²C module is operating in the master mode. The waiting parameter is mainly used for the following signals in master mode:

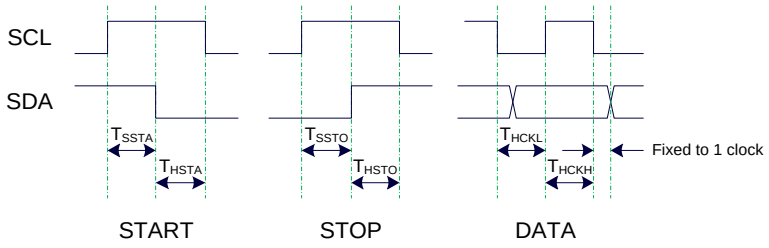
T_{SSTA}: Setup Time of START

T_{HSTA}: Hold Time of START

T_{SSTO}: Setup Time of STOP

T_{HSTO}: Hold Time of STOP

T_{HCKL}: Hold Time of SCL Low



Example 1: When MSSPEED＝10, T_{SSTA}＝T_{HSTA}＝T_{SSTO}＝T_{HSTO}＝T_{HCKL}＝24/FOSC

Example 2: When 400K I2C bus speed is required at 24MHz operating frequency, MSSPEED＝(24M / 400K / 2 - 4) / 2＝13

21.3.2 I²C Master Control Register (I2CMSCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	MSCMD[3:0]			

EMSI: Master mode interrupt enable control bit

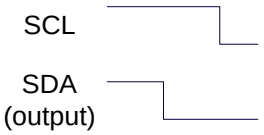
0: disable master mode interrupt

1: enable master mode interrupt

MSCMD[3:0]: master command bits

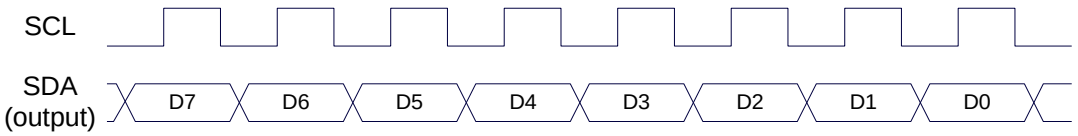
0000: Standby, no action

0001: START command. Send a START signal. If the I²C controller is in idle state currently, i.e. MSBUSY (I2CMSST.7) is 0, writing this command will make the controller enter the busy status, and the hardware will set the MSBUSY status bit automatically and start sending START signal. **If the I²C controller is busy currently**, writing this command will triger to send the START signal. Sending the START signal waveform is shown below:



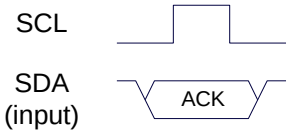
0010: Send data command.

After writing this command, the I²C bus controller will generate 8 clocks on the SCL pin and send the data in the I2CTXD register bit by bit to the SDA pin (send MSB firstly). The waveform of the transmitting data is shown in the following figure:



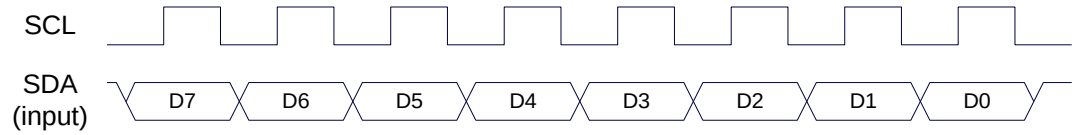
0011: Receive ACK command.

After writing this command, the I²C bus controller will generate a clock on the SCL pin and save the data bit read from SDA to MSACKI (I2CMSST.1). The waveform of the receiving ACK is shown below:



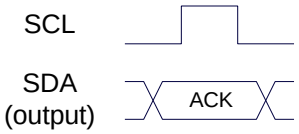
0100: Receive data command.

After writing this command, the I²C bus controller will generate 8 clocks on the SCL pin, and sequentially shift the data bit read from SDA to the I2CRXD register (receiving MSB firstly). The waveform of the receiving data is as shown in the figure below:



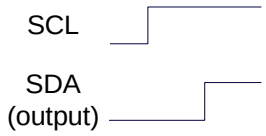
0101: Send ACK command.

After writing this command, the I²C bus controller will generate a clock on the SCL pin and send the data bit in MSACKO (I2CMSST.0) to SDA. The waveform of sending ACK is shown below:



0110: Send STOP signal.

After writing this command, the I²C bus controller starts sending STOP signal. After the signal is sent, the hardware clears the MSBUSY status bit automatically. The waveform of STOP signal is shown below:



0111: Reserved.
1000: Reserved.
1001: Start command + send data command + receive ACK command.
This command is a combination of command 0001, command 0010 and command 0011. After wrting this command, the controller will execute these three commands in sequence.
1010: Send data command + receive ACK command.
This command is a combination of command 0010 and command 0011. After writing this command, the controller will execute these two commands in sequence.
1011: Receive data command + send ACK (0) command.
This command is a combination of command 0100 and command 0101. After writing this command, the controller will execute these two commands in sequence.
Note: The response signal returned by this command is fixed as ACK (0) and is not affected by the MSACKO bit.
1100: Receive data command + send NAK (1) command.
This command is a combination of command 0100 and command 0101. After writing this command, the controller will execute these two commands in sequence.
Note: The response signal returned by this command is fixed to NAK (1), and is not affected by the MSACKO bit.

21.3.3 I²C Master Auxiliary Control Register (I2CMSAUX)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA: I2C data automatic transmission enable bit in master mode.
0: disable automatic transmission
1: enable automatic transmission
If the automatic transmission function is enabled, when the MCU finishes writing to the I2CTXD data register, the I²C controller will trigger the "1010" command automatically, that is, it will send data automatically and receive the ACK signal.

21.3.4 I²C Master Status Register (I2CMSST)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY: status bit of I²C controller in master mode. (Read-only)
0: the controller is in idle state.
1: the controller is in busy state.
When the I²C controller is in master mode, the controller will enter the busy state after sending the START signal in the idle state. The busy state will be maintained until the STOP signal is successfully transmitted, and the state will return to the idle.
MSIF: master mode interrupt request bit (interrupt flag bit). When the I²C controller in the master mode executes the MSCMD command in the completion register I2CMSCR, it generates an interrupt signal. This bit is set to 1 by hardware automatically to request an interrupt to CPU. The MSIF bit must be cleared by software after responding to the interrupt.
MSACKI: In master mode, it is the ACK datum received after sending the “0011” command to the MSCMD bit in I2CMSCR.

MSACKO: In master mode, it is the ACK signal ready to be transmitted. When the “0101” command is sent to the MSCMD bit of I2CMSCR, the controller will read the datum of this bit automatically and send it as ACK to SDA.

21.4 I²C Slave Mode

21.4.1 I²C Slave Control Register (I2CSLCR)

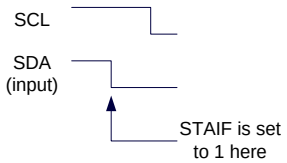
Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI: interrupt enable bit when receiving START signal in slave mode.
0: disable interrupt when receiving START signal in slave mode.
1: enable interrupt when receiving START signal in slave mode.
ERXI: interrupt enable bit after 1 byte datum is received in slave mode
0: disable interrupt after a datum is received in slave mode.
1: enable interrupt after 1 byte datum is reveived in slave mode.
ERXI: interrupt enable bit after 1 byte datum is sent in slave mode
0: disable interrupt after a datum is sent in slave mode.
1: enable interrupt after 1 byte datum is sent in slave mode.
ESTOI: interrupt enable bit after STOP signal is received in slave mode.
0: disable interrupt after STOP signal is received in slave mode.
1: enable interrupt after STOP signal is received in slave mode.
SLRST: reset slave mode

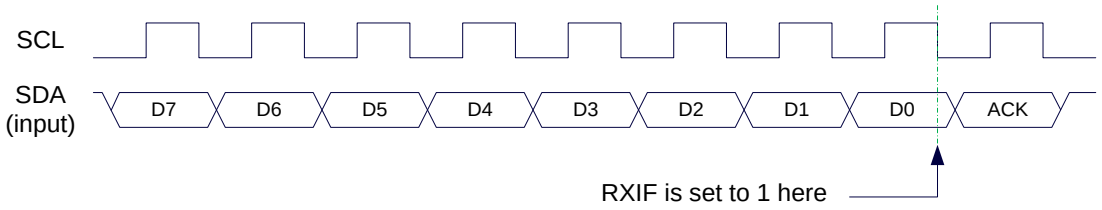
21.4.2 I²C Slave Status Register (I2CSLST)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

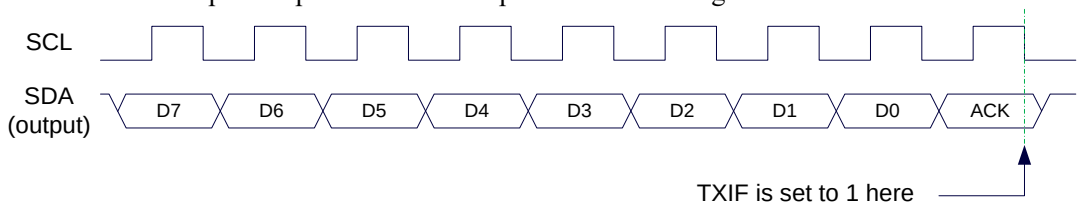
SLBUSY: status bit of I²C controller in slave mode. (Read-only)
0: the controller is in idle state.
1: the controller is in busy state.
When the I²C controller is in slave mode, the controller will continue to detect the subsequent device address data when it receives the START signal from the master in idle state. If the device address matches the slave address set in the current I2CSLADR register, the controller will enter the busy state. And the busy state will be maintained until receives a STOP signal sent by the master successfully, and then the state will return to idle.
STAIF: interrupt request bit after START signal is received in slave mode. After the I²C controller in slave mode receives the START signal, this bit is set by hardware automatically and requests interrupt to CPU. The STAIF bit must be cleared by software after the interrupt is responded. The time point of STAIF being set is shown below:



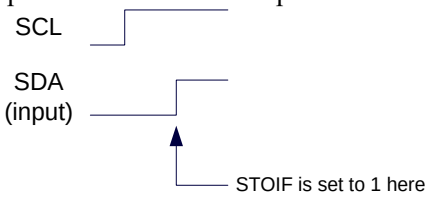
RXIF: interrupt request bit after 1-byte datum is received in slave mode. After the I²C controller in slave mode receives a 1-byte datum, this bit is set by hardware automatically at the falling edge of the 8th clock and will request interrupt to CPU. The RXIF bit must be cleared by software after the interrupt is responded. The time point of RXIF being set is shown in the figure below:



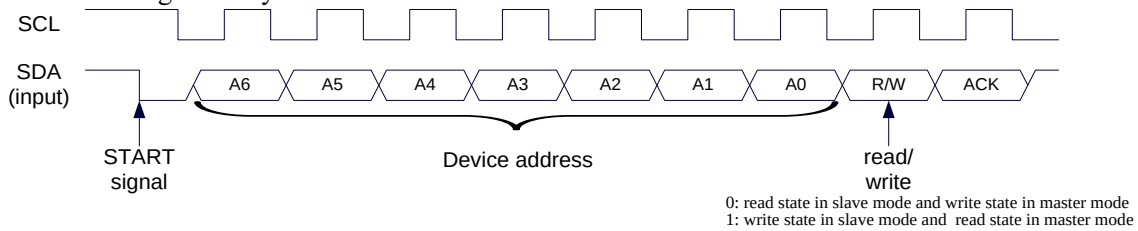
TXIF: interrupt request bit after 1-byte datum transmission is completed in slave mode. After the I²C controller in slave mode completes sending 1 byte of datum and receives a 1-bit ACK signal successfully, this bit is set by hardware automatically at the falling edge of the 9th clock and requests an interrupt to CPU. TXIF bit must be cleared by software after the interrupt is responded. The time point of TXIF being set is shown below:



STOIF: interrupt request bit after STOP signal is received in slave mode. After the I²C controller in slave mode receives the STOP signal, this bit is set by hardware automatically and requests interrupt to CPU. The STOIF bit must be cleared by software after the interrupt is serviced. The time point of STOIF being set is shown below:



SLACKI: ACK data received in slave mode
SLACKO: the ACK signal ready to send out in slave mode.



21.4.3 I²C Slave Address Register (I2CSLADR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H	I2CSLADR[7:1]							MA

I2CSLADR[7:1]: the slave device address

When the I²C controller is in slave mode, the controller will continue to detect the device address and read / write signals sent by the master after it receives the START signal. If the device address sent by the master matches the slave device address set in SLADR[6: 0], the controller will request an interrupt to CPU to process the I²C event. Otherwise, if the device address does not match, the I²C controller continues to monitor, wait for the next START signal, and match the next device address.

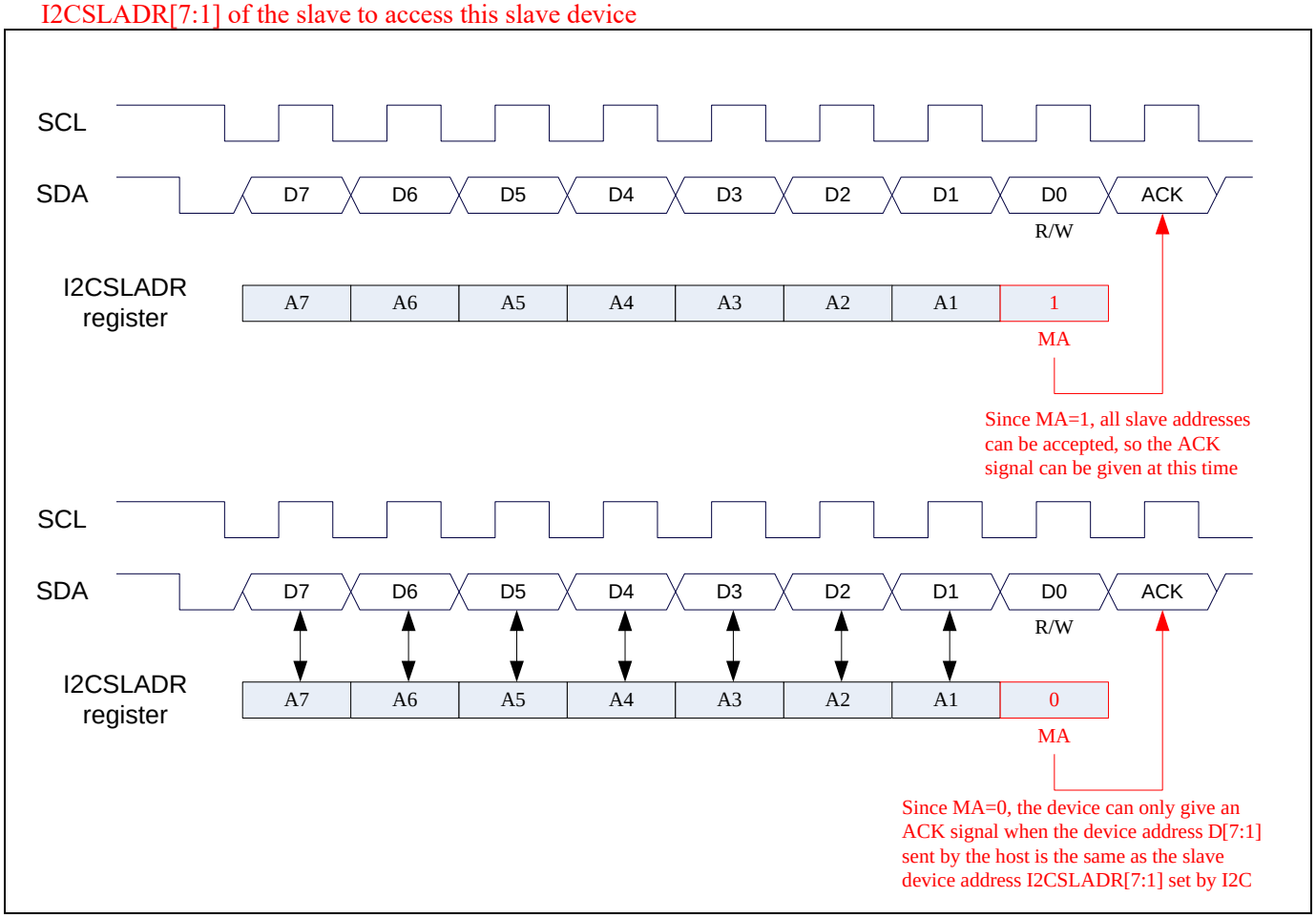
MA: Slave device address matching control bit

0: The device address must be the same as I2CSLADR[7:1].

1: Ignore the settings in I2CSLADR[7:1] and match all device addresses.

Note: The I2C bus protocol stipulates that a maximum of 128 I2C devices (theoretical value) can be mounted on the I2C bus, and different I2C devices are identified by different I2C slave device addresses. After the I2C master sends the start signal, the upper 7 bits of the first data (DATA0) sent are the slave device address (DATA0[7:1] is the I2C device address), and the lowest bit is the read and write signal. When the I2C device

slave address register MA (I2CSLADR.0) is 1, it means that the I2C slave can accept all device addresses. At this time, any device address sent by the host, that is, DATA0[7:1] is any value, the slave Can respond. When I2C device slave address register MA (I2CSLADR.0)
When it is 0, the device address DATA0[7:1] sent by the host must be the same as the device address I2CSLADR[7:1] of the slave to access this slave device



21.4.4 I²C data registers (I2CTXD, I2CRXD)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD is the I²C transmit data register that holds the I²C data to be transmitted.
I2CRXD is the I²C receive data register that holds the I²C data received.

21.5 Example Routines

21.5.1 I²C is Used to Access AT24C256 in Master Mode (Interrupt Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

bit         busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                       //Clear interrupt flag
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81;                             //Send START command
    while (busy);
```

```
}

void SendData(char dat)
{
    I2CTXD = dat;                       //Write data to the data buffer
    busy = 1;
    I2CMSCR = 0x82;                     //Send a SEND command
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;                     //Send read ACK command
    while (busy);
}

char RecvData()
{
    busy = 1;
    I2CMSCR = 0x84;                     //Send RECV command
    while (busy);
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                     //Setup the ACK signal
    busy = 1;
    I2CMSCR = 0x85;                     //Send ACK command
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                     //Setup the NAK signal
    busy = 1;
    I2CMSCR = 0x85;                     //Send ACK command
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86;                     //Send STOP command
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
```

```
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //Enable I2C master mode
    I2CMSST = 0x00;
    EA = 1;

    Start();    //Send start command
    SendData(0xa0);                         //Send device address + write command
    RecvACK();
    SendData(0x00);                         //Send storage address high byte
    RecvACK();
    SendData(0x00);                         //Send storage address low byte
    RecvACK();
    SendData(0x12);                         //Write test data 1
    RecvACK();
    SendData(0x78);                         //Write test data 2
    RecvACK();
    Stop();                                 //Send stop command

    Delay();                                //Waiting for the device to write data

    Start();                                //Send start command
    SendData(0xa0);                         //Send device address + write command
    RecvACK();
    SendData(0x00);                         //Send storage address high byte
    RecvACK();
    SendData(0x00);                         //Send storage address low byte
    RecvACK();
    Start();                                //Send start command
    SendData(0xa1);                         //Send device address + read command
    RecvACK();
    P0 = RecvData();                        //Read data 1
    SendACK();
    P2 = RecvData();                        //Read data 2
    SendNAK();
    Stop();                                 //Send stop command

    P_SW2 = 0x00;

    while (1);
```

```
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

BUSY        BIT     20H.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH

            MOV     DPTR,#I2CMSST       ;Clear interrupt flag
            MOVX    A,@DPTR
            ANL     A,#NOT 40H
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            CLR     BUSY                ;Reset busy flag

            POP     DPH
            POP     DPL
            POP     ACC
            RETI
```

```
START:
        SETB    BUSY
        MOV     A,#10000001B        ;Send START command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDDATA:
        MOV     DPTR,#I2CTXD        ;Write data to the data buffer
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000010B        ;Send a SEND command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVACK:
        SETB    BUSY
        MOV     A,#10000011B        ;Send read ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVDATA:
        SETB    BUSY
        MOV     A,#10000100B        ;Send RECV command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        CALL    WAIT
        MOV     DPTR,#I2CRXD        ;Read data from the data buffer
        MOVX    A,@DPTR
        RET
SENDACK:
        MOV     A,#00000000B        ;Setup the ACK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000101B        ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDNAK:
        MOV     A,#00000001B        ;Setup the NAK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000101B        ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
STOP:
        SETB    BUSY
        MOV     A,#10000110B        ;Send STOP command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
WAIT:
        JB      BUSY,$              ;Wait for the command to be sent
        RET

DELAY:
```

```
        MOV     R0,#0
        MOV     R1,#0
DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B        ;Set the I2C module as master mode
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    EA

        CALL    START               ;Send start command
        MOV     A,#0A0H
        CALL    SENDDATA            ;Send device address + write command
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address high byte
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address low byte
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;Write test data 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;Write test data 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;Send stop command

        CALL    DELAY               ;Waiting for the device to write data

        CALL    START               ;Send start command
        MOV     A,#0A0H             ;Send device address + write command
```

```
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address high byte
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;Send storage address low byte
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START               ;Send start command
        MOV     A,#0A1H             ;Send device address + read command
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA            ;Read data 1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA            ;Read data 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                ;Send stop command

        JMP     $

        END
```

21.5.2 I²C is Used to Access AT24C256 in Master Mode AT24C256 (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR         (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST         (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST         (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
```

```
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                         //Send START command
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                           //Write data to the data buffer
    I2CMSCR = 0x02;                         //Send a SEND command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                         //Send read ACK command
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                         //Send RECV command
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //Setup the ACK signal
    I2CMSCR = 0x05;                         //Send ACK command
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //Setup the NAK signal
    I2CMSCR = 0x05;                         //Send ACK command
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //Send STOP command
    Wait();
}
```

```
void Delay()
{
	int i;

	for (i=0; i<3000; i++)
	{
		_nop_();
		_nop_();
		_nop_();
		_nop_();
	}
}

void main()
{
	P0M0 = 0x00;
	P0M1 = 0x00;
	P1M0 = 0x00;
	P1M1 = 0x00;
	P2M0 = 0x00;
	P2M1 = 0x00;
	P3M0 = 0x00;
	P3M1 = 0x00;
	P4M0 = 0x00;
	P4M1 = 0x00;
	P5M0 = 0x00;
	P5M1 = 0x00;

	P_SW2 = 0x80;

	I2CCFG = 0xe0;				//Enable I2C master mode
	I2CMSST = 0x00;

	Start();				//Send start command
	SendData(0xa0);				//Send device address + write command
	RecvACK();
	SendData(0x00);				//Send storage address high byte
	RecvACK();
	SendData(0x00);				//Send storage address low byte
	RecvACK();
	SendData(0x12);				//Write test data 1
	RecvACK();
	SendData(0x78);				//Write test data 2
	RecvACK();
	Stop();					//Send stop command

	Delay();				//Waiting for the device to write data

	Start();				//Send start command
	SendData(0xa0);				//Send device address + write command
	RecvACK();
	SendData(0x00);				//Send storage address high byte
	RecvACK();
	SendData(0x00);				//Send storage address low byte
	RecvACK();
	Start();				//Send start command
	SendData(0xa1);				//Send device address + read command
	RecvACK();
```

```
    P0 = RecvData();                    //Read data 1
    SendACK();
    P2 = RecvData();                    //Read data 2
    SendNAK();
    Stop();                             //Send stop command

    P_SW2 = 0x00;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;Send START command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;Write data to the data buffer
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;Send a SEND command
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
```

```
RECVACK:
        MOV     A,#00000011B        ;Send read ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVDATA:
        MOV     A,#00000100B        ;Send RECV command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        CALL    WAIT
        MOV     DPTR,#I2CRXD        ;Read data from the data buffer
        MOVX    A,@DPTR
        RET
SENDACK:
        MOV     A,#00000000B        ;Setup the ACK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B        ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDNAK:
        MOV     A,#00000001B        ;Setup the NAK signal
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        MOV     A,#00000101B        ;Send ACK command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
STOP:
        MOV     A,#00000110B        ;Send STOP command
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
WAIT:
        MOV     DPTR,#I2CMSST       ;Clear interrupt flag
        MOVX    A,@DPTR
        JNB     ACC.6,WAIT
        ANL     A,#NOT 40H
        MOVX    @DPTR,A
        RET

DELAY:
        MOV     R0,#0
        MOV     R1,#0
DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
```

```
MOV     P1M0, #00H
MOV     P1M1, #00H
MOV     P2M0, #00H
MOV     P2M1, #00H
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     P_SW2,#80H

MOV     A,#11100000B        ;Set the I2C module as master mode
MOV     DPTR,#I2CCFG
MOVX    @DPTR,A
MOV     A,#00000000B
MOV     DPTR,#I2CMSST
MOVX    @DPTR,A

CALL    START               ;Send start command
MOV     A,#0A0H
CALL    SENDDATA            ;Send device address + write command
CALL    RECVACK
MOV     A,#000H             ;Send storage address high byte
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;Send storage address low byte
CALL    SENDDATA
CALL    RECVACK
MOV     A,#12H              ;Write test data 1
CALL    SENDDATA
CALL    RECVACK
MOV     A,#78H              ;Write test data 2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP                ;Send stop command

CALL    DELAY               ;Waiting for the device to write data

CALL    START               ;Send start command
MOV     A,#0A0H             ;Send device address + write command
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;Send storage address high byte
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H             ;Send storage address low byte
CALL    SENDDATA
CALL    RECVACK
CALL    START               ;Send start command
MOV     A,#0A1H             ;Send device address + read command
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA            ;Read data 1
MOV     P0,A
CALL    SENDACK
CALL    RECVDATA            ;Read data 2
```

```
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                    ;Send stop command

        JMP     $

        END
```

21.5.3 I²C is Used to Access PCF8563 in Master Mode

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR         (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST         (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST         (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                         //Send START command
    Wait();
}
```

```
void SendData(char dat)
{
    I2CTXD = dat;                       //Write data to the data buffer
    I2CMSCR = 0x02;                     //Send a SEND command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                     //Send read ACK command
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                     //Send RECV command
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                     //Setup the ACK signal
    I2CMSCR = 0x05;                     //Send ACK command
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                     //Setup the NAK signal
    I2CMSCR = 0x05;                     //Send ACK command
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                     //Send STOP command
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
```

```
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //Enable I2C master mode
    I2CMSST = 0x00;

    Start();                                //Send start command
    SendData(0xa2);                         //Send device address + write command
    RecvACK();
    SendData(0x02);                         //Send storage address
    RecvACK();
    SendData(0x00);                         //Set second value
    RecvACK();
    SendData(0x00);                         //Set minute value
    RecvACK();
    SendData(0x12);                         //Set hour value
    RecvACK();
    Stop();                                 //Send stop command

    while (1)
    {
        Start();                            //Send start command
        SendData(0xa2);                     //Send device address + write command
        RecvACK();
        SendData(0x02);                     //Send storage address
        RecvACK();
        Start();                            //Send start command
        SendData(0xa3);                     //Send device address + read command
        RecvACK();
        P0 = RecvData();                    //Read second value
        SendACK();
        P2 = RecvData();                    //Read minute value
        SendACK();
        P3 = RecvData();                    //Read hour value
        SendNAK();
        Stop();                             //Send stop command

        Delay();
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
```

```
I2CMSST     XDATA     0FE82H
I2CSLCR     XDATA     0FE83H
I2CSLST     XDATA     0FE84H
I2CSLADR    XDATA     0FE85H
I2CTXD      XDATA     0FE86H
I2CRXD      XDATA     0FE87H

SDA         BIT       P1.4
SCL         BIT       P1.5

P0M1        DATA      093H
P0M0        DATA      094H
P1M1        DATA      091H
P1M0        DATA      092H
P2M1        DATA      095H
P2M0        DATA      096H
P3M1        DATA      0B1H
P3M0        DATA      0B2H
P4M1        DATA      0B3H
P4M0        DATA      0B4H
P5M1        DATA      0C9H
P5M0        DATA      0CAH

            ORG       0000H
            LJMP      MAIN

            ORG       0100H
START:
            MOV       A,#00000001B          ;Send START command
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            JMP       WAIT
SENDDATA:
            MOV       DPTR,#I2CTXD          ;Write data to the data buffer
            MOVX      @DPTR,A
            MOV       A,#00000010B          ;Send a SEND command
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            JMP       WAIT
RECVACK:
            MOV       A,#00000011B          ;Send read ACK command
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            JMP       WAIT
RECVDATA:
            MOV       A,#00000100B          ;Send RECV command
            MOV       DPTR,#I2CMSCR
            MOVX      @DPTR,A
            CALL      WAIT
            MOV       DPTR,#I2CRXD          ;Read data from the data buffer
            MOVX      A,@DPTR
            RET
SENDACK:
            MOV       A,#00000000B          ;Setup the ACK signal
            MOV       DPTR,#I2CMSST
            MOVX      @DPTR,A
            MOV       A,#00000101B          ;Send ACK command
            MOV       DPTR,#I2CMSCR
```

```
                MOVX    @DPTR,A
                JMP     WAIT
SENDNAK:
                MOV     A,#00000001B        ;Setup the NAK signal
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                MOV     A,#00000101B        ;Send ACK command
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                MOV     A,#00000110B        ;Send STOP command
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST       ;Clear interrupt flag
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
                NOP
                DJNZ    R1,DELAY1
                DJNZ    R0,DELAY1
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     P_SW2,#80H

                MOV     A,#11100000B        ;Set the I2C module as master mode
                MOV     DPTR,#I2CCFG
                MOVX    @DPTR,A
                MOV     A,#00000000B
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
```

```
            CALL    START           ;Send start command
            MOV     A,#0A2H
            CALL    SENDDATA        ;Send device address + write command
            CALL    RECVACK
            MOV     A,#002H         ;Send storage address
            CALL    SENDDATA
            CALL    RECVACK
            MOV     A,#00H          ;Set second value
            CALL    SENDDATA
            CALL    RECVACK
            MOV     A,#00H          ;Set minute value
            CALL    SENDDATA
            CALL    RECVACK
            MOV     A,#12H          ;Set hour value
            CALL    SENDDATA
            CALL    RECVACK
            CALL    STOP            ;Send stop command
LOOP:
            CALL    START           ;Send start command
            MOV     A,#0A2H         ;Send device address + write command
            CALL    SENDDATA
            CALL    RECVACK
            MOV     A,#002H         ;Send storage address
            CALL    SENDDATA
            CALL    RECVACK
            CALL    START           ;Send start command
            MOV     A,#0A3H         ;Send device address + read command
            CALL    SENDDATA
            CALL    RECVACK
            CALL    RECVDATA        ;Read second value
            MOV     P0,A
            CALL    SENDACK
            CALL    RECVDATA        ;Read minute value
            MOV     P2,A
            CALL    SENDACK
            CALL    RECVDATA        ;Read hour value
            MOV     P3,A
            CALL    SENDNAK
            CALL    STOP            ;Send stop command

            CALL    DELAY

            JMP     LOOP

            END
```

21.5.4 I²C Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

bit         isda;                                   //Device address flag
bit         isma;                                   //Storage address flag
unsigned char           addr;
unsigned char pdata     buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                           //Handle the START event
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                           //Handle the RECV event
        if (isda)
        {
            isda = 0;                               //Handle the RECV event (RECV DEVICE ADDR)
        }
        else if (isma)
        {
            isma = 0;                               //Handle the RECV event (RECV MEMORY ADDR)
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;                //Handle the RECV event (RECV DATA)
        }
```

```
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                       //Handle the SEND event
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                      //Stop receiving data when receiving NAK
        }
        else
        {
            I2CTXD = buffer[++addr];            //Continue reading data when receiving ACK
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;                       //Handle the STOP event
        isda = 1;
        isma = 1;
    }

    _pop_(P_SW2);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                              //Enable I2C slave mode
    I2CSLADR = 0x5a;                            //Set the slave device address to 5A
                                                //That is, I2CSLADR[7:1]=010_1101B, MA=0B.
                                                //Since MA is 0, the device address sent by the host must be
                                                // the same as I2CSLADR[7:1] to access this I2C slave device.
                                                //If the host needs to write data, it will send 5AH(0101_1010B)
    I2CSLST = 0x00;
    I2CSLCR = 0x78;                             //Enable interrupt of slave mode
    EA = 1;

    isda = 1;                                   //User variable initialization
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5
ISDA        BIT     20H.0       ;Device address flag
ISMA        BIT     20H.1       ;Storage address flag

ADDR        DATA    21H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    PSW
            PUSH    DPL
            PUSH    DPH
            MOV     DPTR,#I2CSLST   ;Detect slave status
            MOVX    A,@DPTR
            JB      ACC.6,STARTIF
            JB      ACC.5,RXIF
            JB      ACC.4,TXIF
            JB      ACC.3,STOPIF
ISREXIT:
            POP     DPH
            POP     DPL
            POP     PSW
            POP     ACC
```

```
                RETI
STARTIF:
                ANL     A,#NOT 40H          ;Handle the START event
                MOVX    @DPTR,A
                JMP     ISREXIT
RXIF:
                ANL     A,#NOT 20H          ;Handle the RECV event
                MOVX    @DPTR,A
                MOV     DPTR,#I2CRXD
                MOVX    A,@DPTR
                JBC     ISDA,RXDA
                JBC     ISMA,RXMA
                MOV     R0,ADDR             ;Handle the RECV event (RECV DATA)
                MOVX    @R0,A
                INC     ADDR
                JMP     ISREXIT
RXDA:
                JMP     ISREXIT             ;Handle the RECV event (RECV DEVICE ADDR)
RXMA:
                MOV     ADDR,A              ;Handle the RECV event (RECV MEMORY ADDR)
                MOV     R0,A
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
TXIF:
                ANL     A,#NOT 10H          ;Handle the SEND event
                MOVX    @DPTR,A
                JB      ACC.1,RXNAK
                INC     ADDR
                MOV     R0,ADDR
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
RXNAK:
                MOVX    A,#0FFH
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
STOPIF:
                ANL     A,#NOT 08H          ;Handle the STOP event
                MOVX    @DPTR,A
                SETB    ISDA
                SETB    ISMA
                JMP     ISREXIT

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
```

```
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#10000001B        ;Enable I2C slave mode
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#01011010B        ;Set the slave device address to 5A
                                    ;That is, I2CSLADR[7:1]=010_1101B, MA=0B.
                                    ;Since MA is 0, the device address sent by the host must be
                                    ; the same as I2CSLADR[7:1] to access this I2C slave device.
                                    ;If the host needs to write data, it will send 5AH(0101_1010B)
        MOV     DPTR,#I2CSLADR
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CSLST
        MOVX    @DPTR,A
        MOV     A,#01111000B        ;Enable interrupt of slave mode
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                ;User variable initialization
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

        SETB    EA

        SJMP    $

        END
```

21.5.5 I²C Slave Mode (Polling Mode)

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
```

```
#define   I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define   I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr       P0M1      =     0x93;
sfr       P0M0      =     0x94;
sfr       P1M1      =     0x91;
sfr       P1M0      =     0x92;
sfr       P2M1      =     0x95;
sfr       P2M0      =     0x96;
sfr       P3M1      =     0xb1;
sfr       P3M0      =     0xb2;
sfr       P4M1      =     0xb3;
sfr       P4M0      =     0xb4;
sfr       P5M1      =     0xc9;
sfr       P5M0      =     0xca;

sbit      SDA       =     P1^4;
sbit      SCL       =     P1^5;

bit       isda;                                 //Device address flag
bit       isma;                                 //Storage address flag
unsigned char           addr;
unsigned char pdata     buffer[256];

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                            //Enable I2C slave mode
    I2CSLADR = 0x5a;                          //Set the slave device address to 5A
                                              //That is, I2CSLADR[7:1]=010_1101B, MA=0B.
                                              //Since MA is 0, the device address sent by the host must be
                                              // the same as I2CSLADR[7:1] to access this I2C slave device.
                                              //If the host needs to write data, it will send 5AH(0101_1010B)
                                              //If the host needs to read data, it will send 5BH (0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x00;                           //Disable interrupt of slave mode

    isda = 1;                                 //User variable initialization
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
```

```
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40;                   //Handle the START event
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;                   //Handle the RECV event
            if (isda)
            {
                isda = 0;                       //Handle the RECV event (RECV DEVICE ADDR)
            }
            else if (isma)
            {
                isma = 0;                       //Handle the RECV event (RECV MEMORY ADDR)
                addr = I2CRXD;
                I2CTXD = buffer[addr];
            }
            else
            {
                buffer[addr++] = I2CRXD;        //Handle the RECV event (RECV DATA)
            }
        }
        else if (I2CSLST & 0x10)
        {
            I2CSLST &= ~0x10;                   //Handle the SEND event
            if (I2CSLST & 0x02)
            {
                I2CTXD = 0xff;                  //Stop receiving data when receiving NAK
            }
            else
            {
                I2CTXD = buffer[++addr];        //Continue reading data when receiving ACK
            }
        }
        else if (I2CSLST & 0x08)
        {
            I2CSLST &= ~0x08;                   //Handle the STOP event
            isda = 1;
            isma = 1;
        }
    }
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H
```

```
SDA             BIT             P1.4
SCL             BIT             P1.5
ISDA            BIT             20H.0           ;Device address flag
ISMA            BIT             20H.1           ;Storage address flag

ADDR            DATA            21H

P0M1            DATA            093H
P0M0            DATA            094H
P1M1            DATA            091H
P1M0            DATA            092H
P2M1            DATA            095H
P2M0            DATA            096H
P3M1            DATA            0B1H
P3M0            DATA            0B2H
P4M1            DATA            0B3H
P4M0            DATA            0B4H
P5M1            DATA            0C9H
P5M0            DATA            0CAH

                ORG             0000H
                LJMP            MAIN

                ORG             0100H
MAIN:
                MOV             SP, #5FH
                MOV             P0M0, #00H
                MOV             P0M1, #00H
                MOV             P1M0, #00H
                MOV             P1M1, #00H
                MOV             P2M0, #00H
                MOV             P2M1, #00H
                MOV             P3M0, #00H
                MOV             P3M1, #00H
                MOV             P4M0, #00H
                MOV             P4M1, #00H
                MOV             P5M0, #00H
                MOV             P5M1, #00H

                MOV             P_SW2,#80H

                MOV             A,#10000001B    ;Enable I2C slave mode
                MOV             DPTR,#I2CCFG
                MOVX            @DPTR,A
                MOV             A,#01011010B    ;Set the slave device address to 5A
                                                ;/That is, I2CSLADR[7:1]=010_1101B, MA=0B.
                                                ;Since MA is 0, the device address sent by the host must be
                                                ; the same as I2CSLADR[7:1] to access this I2C slave device.
                                                ;If the host needs to write data, it will send 5AH(0101_1010B)
                                                ;If the host needs to read data, it will send 5BH (0101_1011B)

                MOV             DPTR,#I2CSLADR
                MOVX            @DPTR,A
                MOV             A,#00000000B
                MOV             DPTR,#I2CSLST
                MOVX            @DPTR,A
                MOV             A,#00000000B    ;Disable interrupt of slave mode
                MOV             DPTR,#I2CSLCR
```

```
                MOVX    @DPTR,A

                SETB    ISDA            ;User variable initialization
                SETB    ISMA
                CLR     A
                MOV     ADDR,A
                MOV     R0,A
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A

LOOP:
                MOV     DPTR,#I2CSLST   ;Detect slave status
                MOVX    A,@DPTR
                JB      ACC.6,STARTIF
                JB      ACC.5,RXIF
                JB      ACC.4,TXIF
                JB      ACC.3,STOPIF
                JMP     LOOP
STARTIF:
                ANL     A,#NOT 40H      ;Handle the START event
                MOVX    @DPTR,A
                JMP     LOOP
RXIF:
                ANL     A,#NOT 20H      ;Handle the RECV event
                MOVX    @DPTR,A
                MOV     DPTR,#I2CRXD
                MOVX    A,@DPTR
                JBC     ISDA,RXDA
                JBC     ISMA,RXMA
                MOV     R0,ADDR         ;Handle the RECV event (RECV DATA)
                MOVX    @R0,A
                INC     ADDR
                JMP     LOOP
RXDA:
                JMP     LOOP            ;Handle the RECV event (RECV DEVICE ADDR)
RXMA:
                MOV     ADDR,A          ;Handle the RECV event (RECV MEMORY ADDR)
                MOV     R0,A
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     LOOP
TXIF:
                ANL     A,#NOT 10H      ;Handle the SEND event
                MOVX    @DPTR,A
                JB      ACC.1,RXNAK
                INC     ADDR
                MOV     R0,ADDR
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     LOOP
RXNAK:
                MOVX    A,#0FFH
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     LOOP
```

```
STOPIF:
        ANL     A,#NOT 08H          ;Handle the STOP event
        MOVX    @DPTR,A
        SETB    ISDA
        SETB    ISMA
        JMP     LOOP

        END
```

21.5.6 Master Codes for testing I²C Slave Mode

C language code

```
//Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                 //Send START command
    Wait();
}
```

```
void SendData(char dat)
{
    I2CTXD = dat;                       //Write data to the data buffer
    I2CMSCR = 0x02;                     //Send a SEND command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                     //Send read ACK command
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                     //Send RECV command
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                     //Setup the ACK signal
    I2CMSCR = 0x05;                     //Send ACK command
    Wait();
}
void SendNAK()
{
    I2CMSST = 0x01;                     //Setup the NAK signal
    I2CMSCR = 0x05;                     //Send ACK command
    Wait();
}
void Stop()
{
    I2CMSCR = 0x06;                     //Send STOP command
    Wait();
}
void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //Enable I2C master mode
    I2CMSST = 0x00;

    Start();                                //Send start command
    SendData(0x5a);                         //Send device address (010_1101B) + write command (0b)
    RecvACK();
    SendData(0x00);                         //Send storage address
    RecvACK();
    SendData(0x12);                         //Write test data 1
    RecvACK();
    SendData(0x78);                         //Write test data 2
    RecvACK();
    Stop();                                 //Send stop command

    Start();                                //Send start command
    SendData(0x5a);                         //Send device address (010_1101B) + write command (0b)
    RecvACK();
    SendData(0x00);                         //Send storage address high byte
    RecvACK();
    Start();                                //Send start command
    SendData(0x5b);                         //Send device address (010_1101B) + read command (1b)
    RecvACK();
    P0 = RecvData();                        //Read data 1
    SendACK();
    P2 = RecvData();                        //Read data 2
    SendNAK();
    Stop();                                 //Send stop command

    P_SW2 = 0x00;

    while (1);
}
```

Assembly code

```
;Operating frequency for test is 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H
```

```
SDA             BIT             P1.4
SCL             BIT             P1.5

P0M1            DATA            093H
P0M0            DATA            094H
P1M1            DATA            091H
P1M0            DATA            092H
P2M1            DATA            095H
P2M0            DATA            096H
P3M1            DATA            0B1H
P3M0            DATA            0B2H
P4M1            DATA            0B3H
P4M0            DATA            0B4H
P5M1            DATA            0C9H
P5M0            DATA            0CAH

                ORG             0000H
                LJMP            MAIN

                ORG             0100H
START:
                MOV             A,#00000001B        ;Send START command
                MOV             DPTR,#I2CMSCR
                MOVX            @DPTR,A
                JMP             WAIT
SENDDATA:
                MOV             DPTR,#I2CTXD        ;Write data to the data buffer
                MOVX            @DPTR,A
                MOV             A,#00000010B        ;Send a SEND command
                MOV             DPTR,#I2CMSCR
                MOVX            @DPTR,A
                JMP             WAIT
RECVACK:
                MOV             A,#00000011B        ;Send read ACK command
                MOV             DPTR,#I2CMSCR
                MOVX            @DPTR,A
                JMP             WAIT
RECVDATA:
                MOV             A,#00000100B        ;Send RECV command
                MOV             DPTR,#I2CMSCR
                MOVX            @DPTR,A
                CALL            WAIT
                MOV             DPTR,#I2CRXD        ;Read data from the data buffer
                MOVX            A,@DPTR
                RET
SENDACK:
                MOV             A,#00000000B        ;Setup the ACK signal
                MOV             DPTR,#I2CMSST
                MOVX            @DPTR,A
                MOV             A,#00000101B        ;Send ACK command
                MOV             DPTR,#I2CMSCR
                MOVX            @DPTR,A
                JMP             WAIT
SENDNAK:
                MOV             A,#00000001B        ;Setup the NAK signal
                MOV             DPTR,#I2CMSST
                MOVX            @DPTR,A
                MOV             A,#00000101B        ;Send ACK command
```

```
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                MOV     A,#00000110B        ;Send STOP command
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST       ;Clear interrupt flag
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
                NOP
                DJNZ    R1,DELAY1
                DJNZ    R0,DELAY1
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     P_SW2,#80H

                MOV     A,#11100000B        ;Set the I2C module as master mode
                MOV     DPTR,#I2CCFG
                MOVX    @DPTR,A
                MOV     A,#00000000B
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A

                CALL    START               ;Send start command
                MOV     A,#5AH              ;Slave address is 5A
                CALL    SENDDATA            ;Send device address (010_1101B) + write command (0b)
                CALL    RECVACK
                MOV     A,#000H             ;Send storage address
                CALL    SENDDATA
```

```
CALL    RECVACK
MOV     A,#12H          ;Write test data 1
CALL    SENDDATA
CALL    RECVACK
MOV     A,#78H          ;Write test data 2
CALL    SENDDATA
CALL    RECVACK
CALL    STOP            ;Send stop command

CALL    DELAY           ;Waiting for the device to write data

CALL    START           ;Send start command
MOV     A,#5AH          ;Send device address (010_1101B) + write command (0b)
CALL    SENDDATA
CALL    RECVACK
MOV     A,#000H         ;Send storage address
CALL    SENDDATA
CALL    RECVACK
CALL    START           ;Send start command
MOV     A,#5BH          ;Send device address (010_1101B) + read command (1b)
CALL    SENDDATA
CALL    RECVACK
CALL    RECVDATA        ;Read data 1
MOV     P0,A
CALL    SENDACK
CALL    RECVDATA        ;Read data 2
MOV     P2,A
CALL    SENDNAK
CALL    STOP            ;Send stop command

JMP     $

END
```

22 LCM interface

STC8A8K64D4 series of microcontrollers integrate an LCM interface controller, which can be used to drive the current popular LCD modules. It can drive I8080 interface and M6800 interface color screen, support 8-bit and 16-bit data width.

26.1 LCM interface function pin switch

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80
LCMIFCFG2	FE51H		LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

LCMIFCPS[1:0]: LCM interface control pin selection bit

LCMIFCPS [1:0]	RS	RD signal of I8080 E signal of M6800	WR signal of I8080 RW signal of M6800
00	P4.1	P4.4	P4.3
01	P4.1	P3.7	P3.6
10	P4.1	P4.2	P4.0
11	P4.0	P3.7	P3.6

LCMIFDPS[1:0]: 8-bit data LCM interface data pin selection bit

LCMIFDPS [1:0]	D16_D8	Data byte DAT[7:0]
00	0	P2[7:0]
01	0	P6[7:0]
10	0	P2[7:0]
11	0	P6[7:0]

LCMIFDPS[1:0]: 16-bit data LCM interface data pin selection bit

LCMIFDPS [1:0]	D16_D8	High byte DAT[15:8]	Low byte DAT[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	P7[7:0]
11	1	P6[7:0]	P7[7:0]

22.2 Registers Related to LCM

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
LCMIFCFG	LCM Interface Configuration Register	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFCFG2	LCM Interface Configuration Register 2	FE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000
LCMIFCR	LCM Interface Control Register	FE52H	ENLCMIF	-	-	-	-	CMD[2:0]			0xxx,x000
LCMIFSTA	LCM Interface Status Register	FE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
LCMIDDATL	LCM Interface low byte data	FE54H	LCMIFDAT[7:0]								0000,0000
LCMIDDATH	LCM Interface high byte data	FE55H	LCMIFDAT[15:8]								0000,0000

22.2.1 LCM Interface Configuration Register (LCMIFCFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	FE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80

LCMIFIE: LCM interrupt enable control bit

0: disable LCM interface interrupt

1: enable LCM interface interrupt

LCMIFIP[1:0]: LCM interface interrupt priority control bits

LCMIFIP[1:0]	interrupt priority

00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

LCMIFDPS[1:0]: LCM interface data pin selection bit

LCMIFDPS [1:0]	D16_D8	High byte DAT[15:8]	Low byte DAT[7:0]
00	0	N/A	P2[7:0]
01	0	N/A	P6[7:0]
10	0	N/A	P2[7:0]
11	0	N/A	P6[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	P7[7:0]
11	1	P6[7:0]	P7[7:0]

D16_D8: LCM interface data width control bit

0: 8-bit data width

1: 16-bit data width

M68_I80: LCM interface mode selection bit

0: I8080 mode

1: M6800 mode

22.2.2 LCM Interface Configuration Register 2 (LCMIFCFG2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG2	FE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

LCMIFCPS[1:0]: LCM interface control pin selection bit

LCMIFCPS [1:0]	RS	RD signal of I8080 E signal of M6800	WR signal of I8080 RW signal of M6800
00	P4.1	P4.4	P4.3
01	P4.1	P3.7	P3.6
10	P4.1	P4.2	P4.0
11	P4.0	P3.7	P3.6

SETUPT[2:0]: Data setup time control bit for LCM interface communication (see timing diagrams in subsequent chapters for details)

HOLDT[1:0]: Data hold time control bit for LCM interface communication (see the timing diagram in the subsequent chapters for details)

22.2.3 LCM Interface Control Register (LCMIFCR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCR	FE52H	ENLCMIF	-	-	-	-	CMD[2:0]		

ELCMIF: LCM interface enable control bit

0: Disable LCM interface function

1: Enable LCM interface function

CMD[2:0]: LCM interface trigger command

CMD[2:0]	trigger command
100	Write command
101	Write data
110	Read command/status
111	Read data

22.2.4 LCM Interface Status Register (LCMIFSTA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0

LCMIFSTA	FE53H	-	-	-	-	-	-	-	LCMIFIF

LCMIFIF: LCM interface interrupt request flag, needs to be cleared by software

22.2.5 LCM interface Data Registers (LCMIFDATL, LCMIFDATH)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFDATL	FE54H	LCMIFDAT[7:0]							
LCMIFDATH	FE55H	LCMIFDAT[15:8]							

LCMIFDAT: LCM interface data register

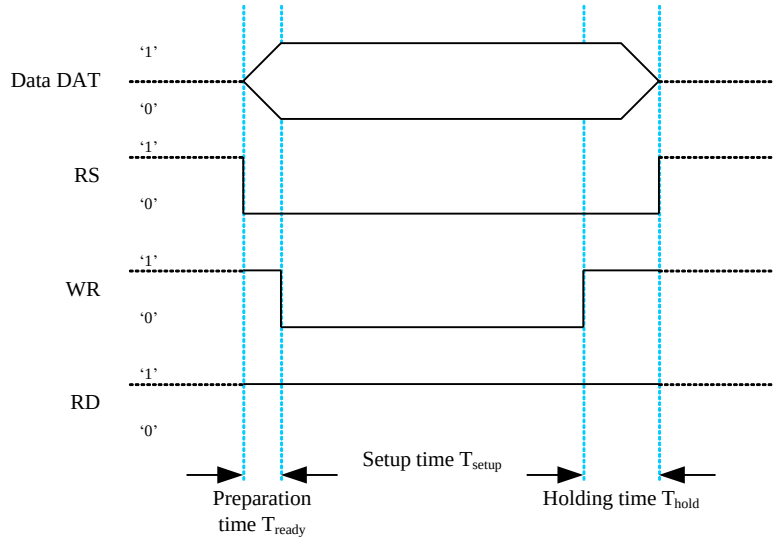
When the data width is 8 bits, only the LCMDATL data is valid.

When the data width is 16 bits, it is combined into 16-bit data by LCMDATL and LCMDATH.

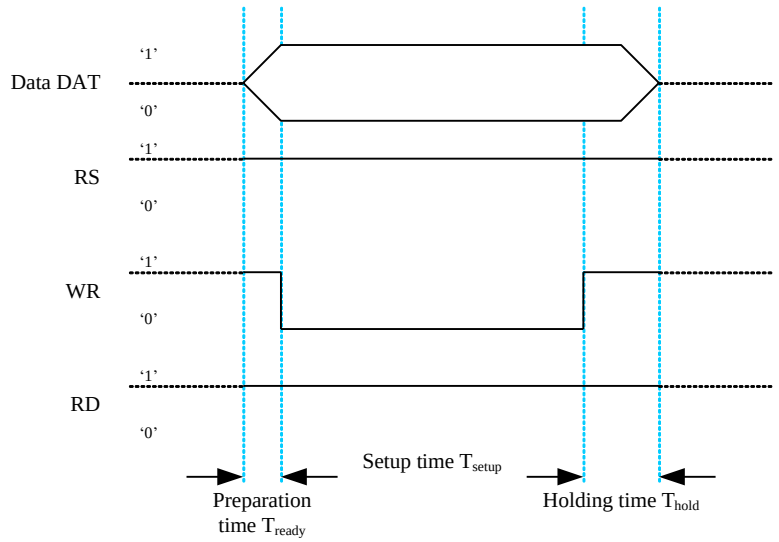
22.3 LCM interface timing diagram

Note: T_{ready} ＝1 system clock
T_{setup} ＝ (SETUPT +1) system clocks
T_{hold} ＝ (HOLDT+1) system clocks

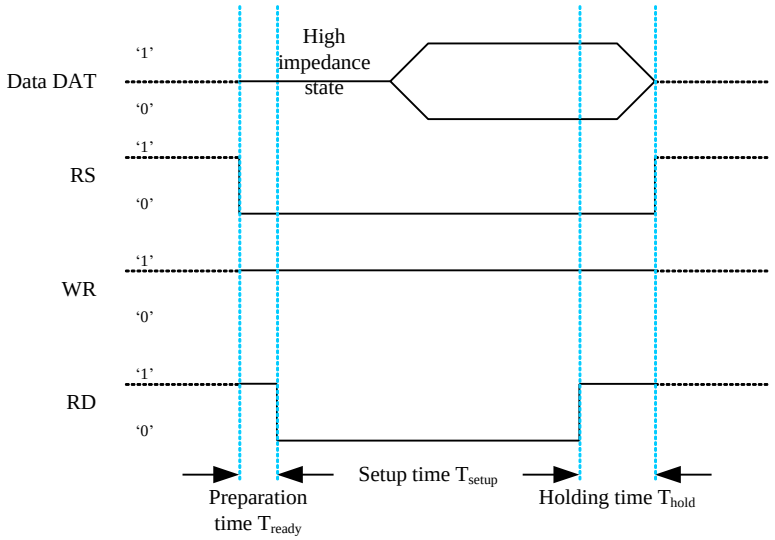
22.3.1 I8080 mode



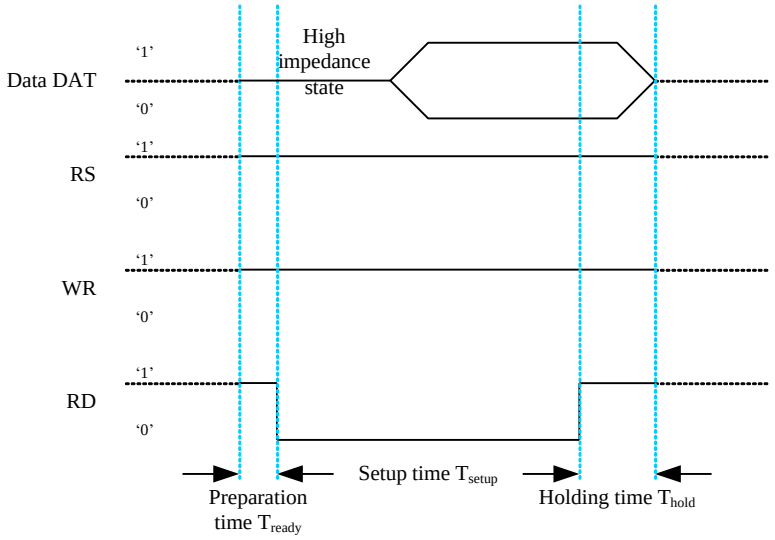
Send command (CMD=100B)



Send data (CMD=101B)

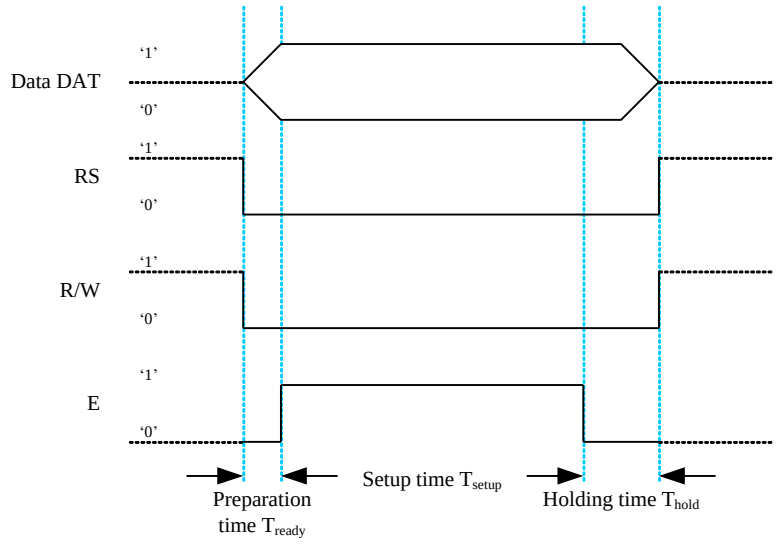


Read command (CMD=110B)

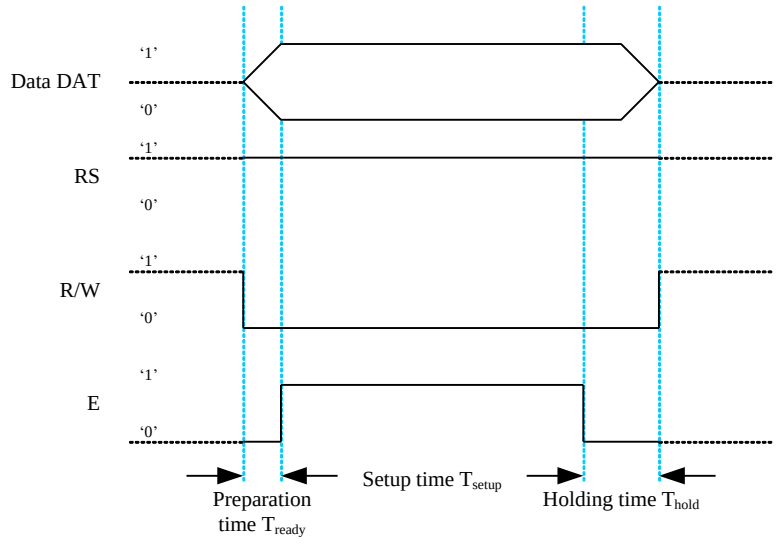


Read data (CMD=111B)

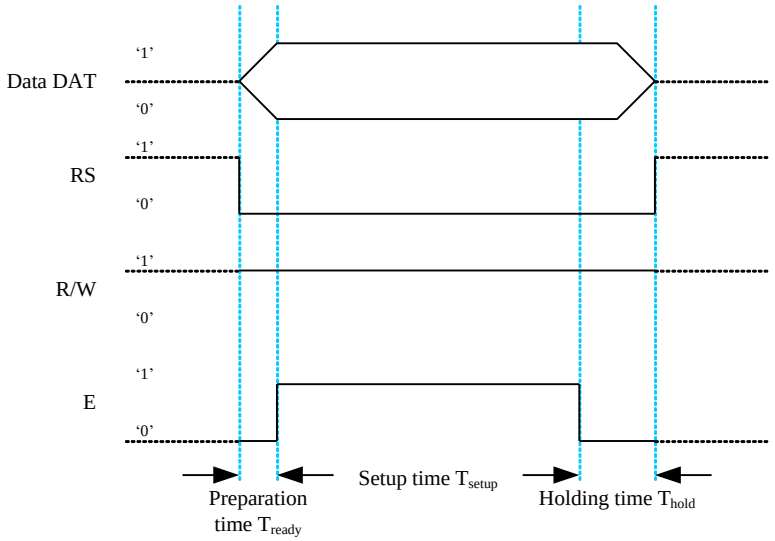
22.3.2 M6800 mode



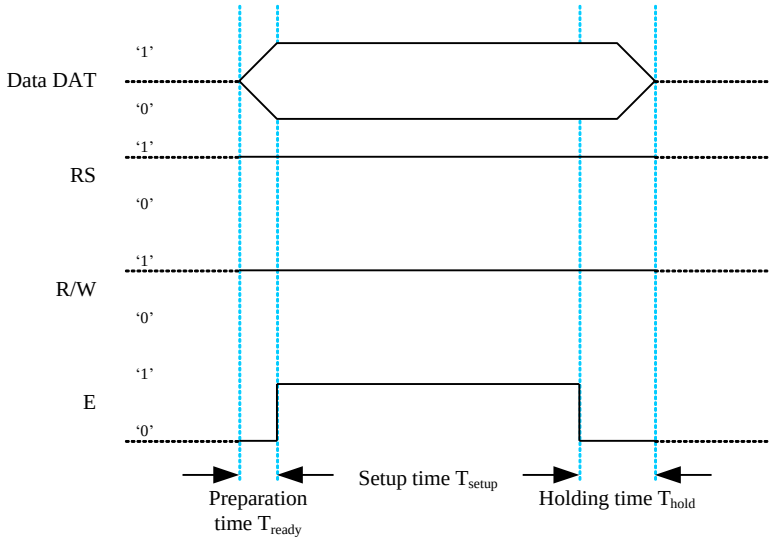
Send command (CMD=100B)



Send data (CMD=101B)



Read command (CMD=110B)



Read data (CMD=111B)

23 DMA

STC8A8K64D4 series of microcomputers support the function of batch data storage, that is, traditional DMA.
The following DMA operations are supported:

- M2M_DMA: read and write data from XRAM memory to XRAM memory
- ADC_DMA: automatically scan the enabled ADC channels and automatically store the converted ADC data into XRAM
- SPI_DMA: automatically exchange data between XRAM data and SPI peripherals
- UR1T_DMA: automatically send the data in XRAM through UART1
- UR1R_DMA: automatically store the data received from UART1 into XRAM
- UR2T_DMA: automatically send the data in XRAM through UART2
- UR2R_DMA: automatically store the data received from UART2 into XRAM
- UR3T_DMA: automatically send the data in XRAM through UART3
- UR3R_DMA: automatically store the data received from UART3 into XRAM
- UR4T_DMA: automatically send the data in XRAM through UART 4
- UR4R_DMA: automatically store the data received from UART4 into XRAM
- LCM_DMA: automatically exchange data between the data in XRAM and the LCM device

The maximum size of each DMA data transfer is 256 bytes.
Each DMA read and write operation to XRAM can be set to 4-level access priority, and the hardware will automatically perform the access arbitration of the XRAM bus, which will not affect CPU access to XRAM. Under the same priority, the access order of different DMAs to XRAM is as follows: SPI_DMA, UR1R_DMA, UR1T_DMA, UR2R_DMA, UR2T_DMA, UR3R_DMA, UR3T_DMA, UR4R_DMA, UR4T_DMA, LCM_DMA, M2M_DMA, ADC_DMA

23.1 Registers Related to DMA

Symbol	Description	Address	Bit Address and Symbol								Reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_M2M_CFG	M2M_DMA Configuration Register	FA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_CR	M2M_DMA Control Register	FA01H	ENM2M	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA Status Register	FA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_M2M_AMT	M2M_DMA Total Bytes Need to be Transferred	FA03H									0000,0000
DMA_M2M_DONE	M2M_DMA Transfer Completed Bytes	FA04H									0000,0000
DMA_M2M_TXAH	M2M_DMA Send High Address	FA05H									0000,0000
DMA_M2M_TXAL	M2M_DMA Send Low Address	FA06H									0000,0000
DMA_M2M_RXAH	M2M_DMA Receive High Address	FA07H									0000,0000
DMA_M2M_RXAL	M2M_DMA Receive Low Address	FA08H									0000,0000
DMA_ADC_CFG	ADC_DMA Configuration Register	FA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_CR	ADC_DMA Control Register	FA11H	ENADC	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_ADC_STA	ADC_DMA Status Register	FA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_ADC_RXAH	ADC_DMA Receive High Address	FA17H									0000,0000
DMA_ADC_RXAL	ADC_DMA Receive Low Address	FA18H									0000,0000
DMA_ADC_CFG2	ADC_DMA Configuration Register2	FA19H	-	-	-	-	CVTIMESEL[3:0]				xxxx,0000
DMA_ADC_CHSW0	ADC_DMA Channel Enable	FA1AH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	1000,0000
DMA_ADC_CHSW1	ADC_DMA Channel Enable	FA1BH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0	0000,0001
DMA_SPI_CFG	SPI_DMA Configuration Register	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_CR	SPI_DMA Control Register	FA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO	000x,xxx0
DMA_SPI_STA	SPI_DMA Status Register	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_SPI_AMT	SPI_DMA Total Bytes Need to be Transferred	FA23H									0000,0000
DMA_SPI_DONE	SPI_DMA Transfer Completed Bytes	FA24H									0000,0000
DMA_SPI_TXAH	SPI_DMA Send High Address	FA25H									0000,0000
DMA_SPI_TXAL	SPI_DMA Send Low Address	FA26H									0000,0000
DMA_SPI_RXAH	SPI_DMA Receive High Address	FA27H									0000,0000
DMA_SPI_RXAL	SPI_DMA Receive Low Address	FA28H									0000,0000
DMA_SPI_CFG2	SPI_DMA Configuration Register2	FA29H	-	-	-	-	-	WRPSS	SSS[1:0]		xxxx,x000
DMA_UR1T_CFG	UR1T_DMA Configuration Register	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_CR	UR1T_DMA Control Register	FA31H	ENUR1T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA Status Register	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0

DMA_UR1T_AMT	UR1T_DMA Total Bytes Need to be Transferred	FA33H									0000,0000
DMA_UR1T_DONE	UR1T_DMA Transfer Completed Bytes	FA34H									0000,0000
DMA_UR1T_TXAH	UR1T_DMA Send High Address	FA35H									0000,0000
DMA_UR1T_TXAL	UR1T_DMA Send Low Address	FA36H									0000,0000
DMA_UR1R_CFG	UR1R_DMA Configuration Register	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_CR	UR1R_DMA Control Register	FA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR1R_STA	UR1R_DMA Status Register	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR1R_AMT	UR1R_DMA Total Bytes Need to be Transferred	FA3BH									0000,0000
DMA_UR1R_DONE	UR1R_DMA Transfer Completed Bytes	FA3CH									0000,0000
DMA_UR1R_TXAH	UR1R_DMA Send High Address	FA3DH									0000,0000
DMA_UR1R_TXAL	UR1R_DMA Send Low Address	FA3EH									0000,0000
DMA_UR2T_CFG	UR2T_DMA Configuration Register	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_CR	UR2T_DMA Control Register	FA41H	ENUR2T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA Status Register	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA Total Bytes Need to be Transferred	FA43H									0000,0000
DMA_UR2T_DONE	UR2T_DMA Transfer Completed Bytes	FA44H									0000,0000
DMA_UR2T_TXAH	UR2T_DMA Send High Address	FA45H									0000,0000
DMA_UR2T_TXAL	UR2T_DMA Send Low Address	FA46H									0000,0000
DMA_UR2R_CFG	UR2R_DMA Configuration Register	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_CR	UR2R_DMA Control Register	FA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR2R_STA	UR2R_DMA Status Register	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR2R_AMT	UR2R_DMA Total Bytes Need to be Transferred	FA4BH									0000,0000
DMA_UR2R_DONE	UR2R_DMA Transfer Completed Bytes	FA4CH									0000,0000
DMA_UR2R_TXAH	UR2R_DMA Send High Address	FA4DH									0000,0000
DMA_UR2R_TXAL	UR2R_DMA Send Low Address	FA4EH									0000,0000
DMA_UR3T_CFG	UR3T_DMA Configuration Register	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_CR	UR3T_DMA Control Register	FA51H	ENUR3T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA Status Register	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA Total Bytes Need to be Transferred	FA53H									0000,0000
DMA_UR3T_DONE	UR3T_DMA Transfer Completed Bytes	FA54H									0000,0000
DMA_UR3T_TXAH	UR3T_DMA Send High Address	FA55H									0000,0000
DMA_UR3T_TXAL	UR3T_DMA Send Low Address	FA56H									0000,0000
DMA_UR3R_CFG	UR3R_DMA Configuration Register	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000
DMA_UR3R_CR	UR3R_DMA Control Register	FA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR3R_STA	UR3R_DMA Status Register	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR3R_AMT	UR3R_DMA Total Bytes Need to be Transferred	FA5BH									0000,0000
DMA_UR3R_DONE	UR3R_DMA Transfer Completed Bytes	FA5CH									0000,0000
DMA_UR3R_TXAH	UR3R_DMA Send High Address	FA5DH									0000,0000
DMA_UR3R_TXAL	UR3R_DMA Send Low Address	FA5EH									0000,0000
DMA_UR4T_CFG	UR4T_DMA Configuration Register	FA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_CR	UR4T_DMA Control Register	FA61H	ENUR4T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA Status Register	FA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA Total Bytes Need to be Transferred	FA63H									0000,0000
DMA_UR4T_DONE	UR4T_DMA Transfer Completed Bytes	FA64H									0000,0000
DMA_UR4T_TXAH	UR4T_DMA Send High Address	FA65H									0000,0000
DMA_UR4T_TXAL	UR4T_DMA Send Low Address	FA66H									0000,0000
DMA_UR4R_CFG	UR4R_DMA Configuration Register	FA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_CR	UR4R_DMA Control Register	FA69H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR4R_STA	UR4R_DMA Status Register	FA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_UR4R_AMT	UR4R_DMA Total Bytes Need to be Transferred	FA6BH									0000,0000
DMA_UR4R_DONE	UR4R_DMA Transfer Completed Bytes	FA6CH									0000,0000
DMA_UR4R_TXAH	UR4R_DMA Send High Address	FA6DH									0000,0000
DMA_UR4R_TXAL	UR4R_DMA Send Low Address	FA6EH									0000,0000
DMA_LCM_CFG	LCM_DMA Configuration Register	FA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]		0xxx,0000
DMA_LCM_CR	LCM_DMA Control Register	FA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	-	0000,0xxx
DMA_LCM_STA	LCM_DMA Status Register	FA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_LCM_AMT	LCM_DMA Total Bytes Need to be Transferred	FA73H									0000,0000
DMA_LCM_DONE	LCM_DMA Transfer Completed Bytes	FA74H									0000,0000

DMA_LCM_TXAH	LCM_DMA Send High Address	FA75H		0000,0000
DMA_LCM_TXAL	LCM_DMA Send Low Address	FA76H		0000,0000
DMA_LCM_RXAH	LCM_DMA Receive High Address	FA77H		0000,0000
DMA_LCM_RXAL	LCM_DMA Receive Low Address	FA78H		0000,0000

23.2 Data read and write between memory and memory (M2M_DMA)

23.2.1 M2M_DMA Configuration Register (DMA_M2M_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	FA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	

M2MIE: M2M_DMA interrupt enable control bit

0: Disable M2M_DMA interrupt

1: Enable M2M_DMA interrupt

TXACO: M2M_DMA source address (read address) changes direction

0: The address is automatically incremented after the data read is completed

1: The address is automatically decremented after the data read is completed

RXACO: M2M_DMA target address (write address) changed direction

0: The address is automatically incremented after data writing is completed

1: The address is automatically decremented after data writing is completed

M2MIP[1:0]: M2M_DMA interrupt priority control bits

M2MIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

M2MPTY[1:0]: M2M_DMA Data bus access priority control bits

M2MPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.2.2 M2M_DMA Control Register (DMA_M2M_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CR	FA01H	ENM2M	TRIG	-	-	-	-	-	-

ENM2M: M2M_DMA function enable control bit

0: Disable M2M_DMA function

1: Enable M2M_DMA function

TRIG: M2M_DMA data read and write trigger control bit

0: Write 0 is invalid

1: Write 1 to start M2M_DMA operation.

23.2.3 M2M_DMA Status Register (DMA_M2M_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	FA02H	-	-	-	-	-	-	-	M2MIF

M2MIF: M2M_DMA interrupt request flag bit. When the M2M_DMA operation is completed, the hardware automatically sets M2MIF to 1, and if the M2M_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

23.2.4 M2M_DMA transfer total byte register (DMA_M2M_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0

DMA_M2M_AMT	FA03H	

DMA_M2M_AMT: Set the number of bytes that need to read and write data.

Note: The actual number of bytes read and written is (DMA_M2M_AMT+1), that is, when DMA_M2M_AMT is set to 0, 1 byte is read and written, and when DMA_M2M_AMT is set to 255, 256 bytes are read and written.

23.2.5 M2M_DMA transfer complete byte register (DMA_M2M_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_DONE	FA04H								

DMA_M2M_DONE: The number of bytes that have been read and written currently.

23.2.6 M2M_DMA Send Address Registers (DMA_M2M_TXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_TXAH	FA05H	ADDR[15:8]							
DMA_M2M_TXAL	FA06H	ADDR[7:0]							

DMA_M2M_TXA: Set the source address when reading and writing data. Data will be read from this address when the M2M_DMA operation is performed.

23.2.7 M2M_DMA Receive Address Registers (DMA_M2M_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_RXAH	FA07H	ADDR[15:8]							
DMA_M2M_RXAL	FA08H	ADDR[7:0]							

DMA_M2M_RXA: Set the target address when reading and writing data. Data will be written from this address when the M2M_DMA operation is performed.

23.3 ADC Automatic Data Storage (ADC_DMA)

23.3.1 ADC_DMA Configuration Register (DMA_ADC_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG	FA10H	ADCIE	-			ADCIP[1:0]		ADCPTY[1:0]	

ADCIE: ADC_DMA interrupt enable control bit

0: Disable ADC_DMA interrupt

1: Enable ADC_DMA interrupt

ADCIP[1:0]: ADC_DMA interrupt priority control bits

ADCIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

ADCPTY[1:0]：ADC_DMA Data bus access priority control bits

ADCPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.3.2 ADC_DMA Control Register (DMA_ADC_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CR	FA11H	ENADC	TRIG	-	-	-	-	-	-

ENADC: ADC_DMA function enable control bit

0: Disable ADC_DMA function

1: Enable ADC_DMA function

TRIG: ADC_DMA operation trigger control bit

0: Write 0 is invalid

1: Write 1 to start ADC_DMA operation.

23.3.3 ADC_DMA Status Register (DMA_ADC_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_STA	FA12H	-	-	-	-	-	-	-	ADCIF

ADCIF: ADC_DMA interrupt request flag bit. After ADC_DMA completes scanning all enabled ADC channels, the hardware automatically sets ADCIF to 1. If the ADC_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

23.3.4 ADC_DMA Receive Address Registers (DMA_ADC_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_RXAH	FA17H	ADDR[15:8]							
DMA_ADC_RXAL	FA18H	ADDR[7:0]							

DMA_ADC_RXA: Set the storage address of ADC conversion data during ADC_DMA operation.

23.3.5 ADC_DMA Configuration Register 2 (DMA_ADC_CFG2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG2	FA19H	-	-	-	-	CVTIMESEL[3:0]			

CVTIMESEL[3:0]: Set the number of ADC conversions for each ADC channel during ADC_DMA operation.

CVTIMESEL[3:0]	Number of ADC conversions
0xxx	1
1000	2
1001	4
1010	8
1011	16
1100	32
1101	64
1110	128
1111	256

23.3.6 ADC_DMA Channel Enable Registers (DMA_ADC_CHSWx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CHSW0	FA1AH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
DMA_ADC_CHSW1	FA1BH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

CHn: The ADC channel to be scanned automatically when setting ADC_DMA operation. Channel scanning always starts from the lower-numbered channel.

23.3.7 Data storage format of ADC_DMA

Note: ADC conversion speed and conversion result alignment are set by ADC related registers.
XRAM[DMA_ADC_RXA+0] = high byte of the 1st ADC conversion result of the 1st enabled channel;
XRAM[DMA_ADC_RXA+1] = low byte of the 1st ADC conversion result of the 1st enabled channel;
XRAM[DMA_ADC_RXA+2] = high byte of the 2nd ADC conversion result of the 1st enabled channel;
XRAM[DMA_ADC_RXA+3] = low byte of the 2nd ADC conversion result of the 1st enabled channel;
...
XRAM[DMA_ADC_RXA+2n-2] = high byte of the nth ADC conversion result of the 1st enabled channel;
XRAM[DMA_ADC_RXA+2n-1] = low byte of the nth ADC conversion result of the 1st enabled channel;
XRAM[DMA_ADC_RXA+2n] = ADC channel number of 1st channel;
XRAM[DMA_ADC_RXA+2n+1] = remainder after the average value of the n ADC conversion results of the 1st channel;
XRAM[DMA_ADC_RXA+2n+2] = high byte of the average value of the n ADC conversion results of the 1st channel;
XRAM[DMA_ADC_RXA+2n+3] = low byte of the average value of the n ADC conversion results of the 1st channel;

XRAM[DMA_ADC_RXA+(2n+3)+0] = high byte of the 1st ADC conversion result of the 2nd enabled channel;
XRAM[DMA_ADC_RXA+(2n+3)+1] = low byte of the 1st ADC conversion result of the 2nd enabled channel;
XRAM[DMA_ADC_RXA+(2n+3)+2] = high byte of the 2nd ADC conversion result of the 2nd enabled channel;
XRAM[DMA_ADC_RXA+(2n+3)+3] = low byte of the 2nd ADC conversion result of the 2nd enabled channel;
...
XRAM[DMA_ADC_RXA+(2n+3)+2n-2] = high byte of the nth ADC conversion result of the enabled 2nd channel;
XRAM[DMA_ADC_RXA+(2n+3)+2n-1] = low byte of the nth ADC conversion result of the enabled 2nd channel;
XRAM[DMA_ADC_RXA+(2n+3)+2n] = ADC channel number of 2nd channel;
XRAM[DMA_ADC_RXA+(2n+3)+2n+1] = remainder after the average value of the n ADC conversion results of 2nd channel;
XRAM[DMA_ADC_RXA+(2n+3)+2n+2] = high byte of the average value of n ADC conversion results of 2nd channel;
XRAM[DMA_ADC_RXA+(2n+3)+2n+3] = low byte of the average value of n ADC conversion results of 2nd channel;
...
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+0] = high byte of the 1st ADC conversion result of the enabled mth channel;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+1] = low byte of the 1st ADC conversion result of the enabled mth channel;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2] = high byte of the 2nd ADC conversion result of the enabled mth channel;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+3] = low byte of the 2nd ADC conversion result of the enabled mth channel;
...
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n-2] = high byte of the nth ADC conversion result of the enabled mth channel;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n-1] = low byte of the nth ADC conversion result of the enabled mth channel;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n] = ADC channel number of the mth channel;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n+1] = remainder after the average value of the n ADC conversion results of the mth channel;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n+2] = high byte of the average value of the n ADC conversion results of the mth channel;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n+3] = low byte of the average value of the n ADC conversion results of the mth channel;

The form is as follows:

ADC channel	Offset address	Data
1st channel	0	high byte of the 1st ADC conversion result of the 1st enabled channel
	1	low byte of the 1st ADC conversion result of the 1st enabled channel
	2	high byte of the 2nd ADC conversion result of the 1st enabled channel
	3	low byte of the 2nd ADC conversion result of the 1st enabled channel
	...	...
	2n-2	high byte of the nth ADC conversion result of the 1st enabled channel
	2n-1	low byte of the nth ADC conversion result of the 1st enabled channel
	2n	ADC channel number of 1st channel
	2n+1	remainder after the average value of the n ADC conversion results of the 1st channel
	2n+2	high byte of the average value of the n ADC conversion results of the 1st channel
	2n+3	low byte of the average value of the n ADC conversion results of the 1st channel
2nd channel	(2n+3) + 0	high byte of the 1st ADC conversion result of the 2nd enabled channel
	(2n+3) + 1	low byte of the 1st ADC conversion result of the 2nd enabled channel
	(2n+3) + 2	high byte of the 2nd ADC conversion result of the 2nd enabled channel
	(2n+3) + 3	low byte of the 2nd ADC conversion result of the 2nd enabled channel
	...	...
	(2n+3) + 2n-2	high byte of the nth ADC conversion result of the enabled 2nd channel
	(2n+3) + 2n-1	low byte of the nth ADC conversion result of the enabled 2nd channel
	(2n+3) + 2n	ADC channel number of 2nd channel
	(2n+3) + 2n+1	remainder after the average value of the n ADC conversion results of 2nd channel
	(2n+3) + 2n+2	high byte of the average value of n ADC conversion results of 2nd channel
	(2n+3) + 2n+3	low byte of the average value of n ADC conversion results of 2nd channel
	...	...
mth channel	(m-1)(2n+3) + 0	high byte of the 1st ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 1	low byte of the 1st ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 2	high byte of the 2nd ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 3	low byte of the 2nd ADC conversion result of the enabled mth channel
	...	...
	(m-1)(2n+3) + 2n-2	high byte of the nth ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 2n-1	low byte of the nth ADC conversion result of the enabled mth channel
	(m-1)(2n+3) + 2n	ADC channel number of the mth channel
	(m-1)(2n+3) + 2n+1	remainder after the average value of the n ADC conversion results of the mth channel
	(m-1)(2n+3) + 2n+2	high byte of the average value of the n ADC conversion results of the mth channel
	(m-1)(2n+3) + 2n+3	low byte of the average value of the n ADC conversion results of the mth channel

23.4 Data exchange between SPI and memory (SPI_DMA)

23.4.1 SPI_DMA Configuration Register（DMA_SPI_CFG）

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG	FA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	

SPIIE: SPI_DMA interrupt enable control bit

0: Disable SPI_DMA interrupt

1: Enable SPI_DMA interrupt

ACT_TX: SPI_DMA transmit data control bit

0: Disable SPI_DMA to send data. In master mode, SPI only sends clock to SCLK port, but does not read data from XRAM, nor send data to MOSI port; in slave mode, SPI does not read data from XRAM, nor send data to MISO port.
1: Enable SPI_DMA to send data. In master mode, SPI sends clock to SCLK port, and reads data from XRAM and sends data to MOSI port; in slave mode, SPI reads data from XRAM and sends data to MISO port.

ACT_RX: SPI_DMA receive data control bit
0: Disable SPI_DMA to receive data. In master mode, SPI only sends clock to SCLK port, but does not read data from MISO port, nor write data to XRAM; in slave mode, SPI does not read data from MOSI port, nor write data to XRAM.
1: Enable SPI_DMA to receive data. In master mode, SPI sends clock to SCLK port, and reads data from MISO port and writes data to XRAM; in slave mode, SPI reads data from MOSI port and writes XRAM.

SPIIP[1:0]: SPI_DMA interrupt priority control bits

SPIIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

SPIPTY[1:0]：SPI_DMA Data bus access priority control bits

SPIPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.4.2 SPI_DMA Control Register (DMA_SPI_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CR	FA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO

ENSPI: SPI_DMA function enable control bit
0: Disable SPI_DMA function
1: Enable SPI_DMA function

TRIG_M: SPI_DMA master mode trigger control bit
0: Write 0 is invalid
1: Write 1 to start SPI_DMA master mode operation.

TRIG_S: SPI_DMA slave mode trigger control bit
0: Write 0 is invalid
1: Write 1 to start SPI_DMA slave mode operation.

CLRFIFO: Clear SPI_DMA receive FIFO control bit
0: Write 0 is invalid
1: Before starting the SPI_DMA operation, clear the built-in FIFO of SPI_DMA firstly.

23.4.3 SPI_DMA Status Register (DMA_SPI_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_STA	FA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF

SPIIF: SPI_DMA interrupt request flag bit. After the SPI_DMA data exchange is completed, the hardware automatically sets SPIIF to 1. If the SPI_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

RXLOSS: SPI_DMA receive data discard flag. During the SPI_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the SPI_DMA and the data received by the SPI_DMA is automatically discarded, the hardware automatically sets RXLOSS to 1. The flag bit needs to be cleared by software.

TXOVW: SPI_DMA data coverage flag. During the data transfer process of SPI_DMA, when the host mode SPI writes the SPDAT register to trigger the SPI data transfer again, the data transfer will fail, and the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

23.4.4 SPI_DMA transfer total byte register (DMA_SPI_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_AMT	FA23H								

DMA_SPI_AMT: Set the number of bytes that need to read and write data.

Note: The actual number of bytes read and written is (DMA_SPI_AMT+1), that is, when DMA_SPI_AMT is set to 0, 1 byte is transferred, and when DMA_SPI_AMT is set to 255, 256 bytes are transferred.

23.4.5 SPI_DMA transfer complete byte register (DMA_SPI_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_DONE	FA24H								

DMA_SPI_DONE: The number of bytes that have been read and written currently.

23.4.6 SPI_DMA Send Address Registers (DMA_SPI_TXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_TXAH	FA25H	ADDR[15:8]							
DMA_SPI_TXAL	FA26H	ADDR[7:0]							

DMA_SPI_TXA: Set the source address when reading and writing data. Data will be read from this address when the SPI_DMA operation is performed.

23.4.7 SPI_DMA Receive Address Registers (DMA_SPI_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_RXAH	FA27H	ADDR[15:8]							
DMA_SPI_RXAL	FA28H	ADDR[7:0]							

DMA_SPI_RXA: Set the target address when reading and writing data. Data will be written from this address when the SPI_DMA operation is performed.

23.4.8 SPI_DMA Configuration Register 2 (DMA_SPI_CFG2)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG2	FA29H	-	-	-	-	-	WRPSS	SSS[1:0]	

WRPSS: Enable SS pin control bit during SPI_DMA process

0: During the SPI_DMA transfer process, the SS pin is not automatically controlled

1: During the SPI_DMA transfer process, the SS pin is automatically pulled down, and the original state is automatically restored after the transfer is completed.

SSS[1:0]: During the SPI_DMA process, control the SS selection bit automatically

SSS[1:0]	SS pin
00	P1.2
01	P2.2
10	P7.4
11	P3.5

23.5 Data exchange between UART1 and memory (UR1T_DMA, UR1R_DMA)

23.5.1 UR1T_DMA Configuration Register (DMA_UR1T_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CFG	FA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	

UR1TIE: UR1T_DMA interrupt enable control bit
0: Disable UR1T_DMA interrupt
1: Enable UR1T_DMA interrupt

UR1TIP[1:0]: UR1T_DMA interrupt priority control bits

UR1TIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR1TPTY[1:0]: UR1T_DMA Data bus access priority control bits

UR1TPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.5.2 UR1T_DMA Control Register (DMA_UR1T_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CR	FA31H	ENUR1T	TRIG	-	-	-	-	-	-

ENUR1T: UR1T_DMA function enable control bit
0: Disable UR1T_DMA function
1: Enable UR1T_DMA function

TRIG: UR1T_DMA UART1 transmit trigger control bit
0: Write 0 is invalid
1: Write 1 to start UR1T_DMA automatically sending data.

23.5.3 UR1T_DMA Status Register (DMA_UR1T_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_STA	FA32H	-	-	-	-	-	TXOVW	-	UR1TIF

UR1TIF: UR1T_DMA interrupt request flag bit. When the UR1T_DMA data transmission is completed, the hardware automatically sets UR1TIF to 1, and if the UR1T_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

TXOVW: UR1T_DMA data coverage flag. When UR1T_DMA is in the process of data transmission, and the UART writes the SBUF register to trigger the UART to send data again, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

23.5.4 UR1T_DMA transfer total byte register (DMA_UR1T_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_AMT	FA33H								

DMA_UR1T_AMT: Set the number of bytes of data that needs to be automatically sent.
Note: The actual number of bytes is (DMA_UR1T_AMT+1), that is, when DMA_UR1T_AMT is set to 0, 1 byte is transferred, and when DMA_UR1T_AMT is set to 255, 256 bytes are transferred.

23.5.5 UR1T_DMA transfer complete byte register (DMA_UR1T_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_DONE	FA34H								

DMA_UR1T_DONE: The number of bytes that have been sent so far.

23.5.6 UR1T_DMA Send Address Registers (DMA_UR1T_TXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_TXAH	FA35H	ADDR[15:8]							
DMA_UR1T_TXAL	FA36H	ADDR[7:0]							

DMA_UR1T_TXA: Set the source address of automatic data transmission. Data is read from this address when performing a UR1T_DMA operation.

23.5.7 UR1R_DMA Configuration Register (DMA_UR1R_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CFG	FA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	

UR1RIE: UR1R_DMA interrupt enable control bit

0: Disable UR1R_DMA interrupt

1: Enable UR1R_DMA interrupt

UR1RIP[1:0]: UR1R_DMA interrupt priority control bits

UR1RIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR1RPTY[1:0]: UR1R_DMA Data bus access priority control bits

UR1RPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.5.8 UR1R_DMA Control Register (DMA_UR1R_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CR	FA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO

ENUR1R: UR1R_DMA function enable control bit

0: Disable UR1R_DMA function

1: Enable UR1R_DMA function

TRIG: UR1R_DMA UART1 receive trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR1R_DMA receiving data automatically

CLRFIFO: Clear UR1R_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the UR1R_DMA operation, clear the built-in FIFO of the UR1R_DMA firstly

23.5.9 UR1R_DMA Status Register (DMA_UR1R_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_STA	FA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF

UR1RIF: UR1R_DMA interrupt request flag bit. When UR1R_DMA receives data, the hardware will automatically set

UR1RIF to 1. If the UR1R_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR1R_DMA receive data discard flag. During the UR1R_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the UR1R_DMA and the data received by the UR1R_DMA is automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

23.5.10 UR1R_DMA transfer total byte register (DMA_UR1R_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_AMT	FA3BH								

DMA_UR1R_AMT: Set the number of data bytes that need to automatically receive. **Note: The actual number of bytes is (DMA_UR1R_AMT+1), that is, when DMA_UR1R_AMT is set to 0, 1 byte is transferred, and when DMA_UR1R_AMT is set to 255, 256 bytes are transferred.**

23.5.11 UR1R_DMA transfer complete byte register (DMA_UR1R_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_DONE	FA3CH								

DMA_UR1R_DONE: The number of bytes that have been received currently.

23.5.12 UR1R_DMA Receive Address Registers (DMA_UR1T_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_RXAH	FA3DH	ADDR[15:8]							
DMA_UR1R_RXAL	FA3EH	ADDR[7:0]							

DMA_UR1R_RXA: Set the target address for automatically receiving data. Data will be written from this address when performing a UR1R_DMA operation.

23.6 Data exchange between UART2 and memory (UR2T_DMA, UR2R_DMA)

23.6.1 UR2T_DMA Configuration Register (DMA_UR2T_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CFG	FA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	

UR2TIE: UR2T_DMA interrupt enable control bit
0: Disable UR2T_DMA interrupt
1: Enable UR2T_DMA interrupt

UR2TIP[1:0]: UR2T_DMA interrupt priority control bits

UR2TIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR2TPTY[1:0]: UR2T_DMA Data bus access priority control bits

UR2TPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.6.2 UR2T_DMA Control Register (DMA_UR2T_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CR	FA41H	ENUR2T	TRIG	-	-	-	-	-	-

ENUR2T: UR2T_DMA function enable control bit
0: Disable UR2T_DMA function
1: Enable UR2T_DMA function

TRIG: UR2T_DMA UART1 transmit trigger control bit
0: Write 0 is invalid
1: Write 1 to start UR2T_DMA automatically sending data.

23.6.3 UR2T_DMA Status Register (DMA_UR2T_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_STA	FA42H	-	-	-	-	-	TXOVW	-	UR2TIF

UR2TIF: UR2T_DMA interrupt request flag bit. When the UR2T_DMA data transmission is completed, the hardware automatically sets UR2TIF to 1, and if the UR2T_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

TXOVW: UR2T_DMA data coverage flag. When UR2T_DMA is in the process of data transmission, and the UART writes the SBUF register to trigger the UART to send data again, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

23.6.4 UR2T_DMA transfer total byte register (DMA_UR2T_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_AMT	FA43H								

DMA_UR2T_AMT: Set the number of bytes of data that needs to be automatically sent.
Note: The actual number of bytes is (DMA_UR2T_AMT+1), that is, when DMA_UR2T_AMT is set to 0, 1 byte is transferred, and when DMA_UR2T_AMT is set to 255, 256 bytes are transferred.

23.6.5 UR2T_DMA transfer complete byte register (DMA_UR2T_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_DONE	FA44H								

DMA_UR2T_DONE: The number of bytes that have been sent so far.

23.6.6 UR2T_DMA Send Address Registers (DMA_UR2T_TXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_TXAH	FA45H	ADDR[15:8]							
DMA_UR2T_TXAL	FA46H	ADDR[7:0]							

DMA_UR2T_TXA: Set the source address of automatic data transmission. Data is read from this address when performing a UR2T_DMA operation.

23.6.7 UR2R_DMA Configuration Register (DMA_UR2R_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CFG	FA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	

UR2RIE: UR2R_DMA interrupt enable control bit

0: Disable UR2R_DMA interrupt

1: Enable UR2R_DMA interrupt

UR2RIP[1:0]: UR2R_DMA interrupt priority control bits

UR2RIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR2RPTY[1:0]：UR2R_DMA Data bus access priority control bits

UR2RPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.6.8 UR2R_DMA Control Register (DMA_UR2R_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CR	FA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO

ENUR2R: UR2R_DMA function enable control bit

0: Disable UR2R_DMA function

1: Enable UR2R_DMA function

TRIG: UR2R_DMA UART1 receive trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR2R_DMA receiving data automatically

CLRFIFO: Clear UR2R_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the UR2R_DMA operation, clear the built-in FIFO of the UR2R_DMA firstly

23.6.9 UR2R_DMA Status Register (DMA_UR2R_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_STA	FA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF

UR2RIF: UR2R_DMA interrupt request flag bit. When UR2R_DMA receives data, the hardware will automatically set UR2RIF to 1. If the UR2R_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR2R_DMA receive data discard flag. During the UR2R_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the UR2R_DMA and the data received by the UR2R_DMA is automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

23.6.10 UR2R_DMA transfer total byte register (DMA_UR2R_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_AMT	FA4BH								

DMA_UR2R_AMT: Set the number of data bytes that need to automatically receive. **Note: The actual number of bytes is (DMA_UR2R_AMT+1), that is, when DMA_UR2R_AMT is set to 0, 1 byte is transferred, and when DMA_UR2R_AMT is set to 255, 256 bytes are transferred.**

23.6.11 UR2R_DMA transfer complete byte register (DMA_UR2R_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_DONE	FA4CH								

DMA_UR2R_DONE: The number of bytes that have been received currently.

23.6.12 UR2R_DMA Receive Address Registers (DMA_UR2T_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_RXAH	FA4DH	ADDR[15:8]							
DMA_UR2R_RXAL	FA4EH	ADDR[7:0]							

DMA_UR2R_RXA: Set the target address for automatically receiving data. Data will be written from this address when performing a UR2R_DMA operation.

23.7 Data exchange between UART3 and memory (UR3T_DMA, UR3R_DMA)

23.7.1 UR3T_DMA Configuration Register (DMA_UR3T_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CFG	FA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	

UR3TIE: UR3T_DMA interrupt enable control bit
0: Disable UR3T_DMA interrupt
1: Enable UR3T_DMA interrupt

UR3TIP[1:0]: UR3T_DMA interrupt priority control bits

UR3TIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR3TPTY[1:0]: UR3T_DMA Data bus access priority control bits

UR3TPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.7.2 UR3T_DMA Control Register (DMA_UR3T_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CR	FA51H	ENUR3T	TRIG	-	-	-	-	-	-

ENUR3T: UR3T_DMA function enable control bit
0: Disable UR3T_DMA function
1: Enable UR3T_DMA function

TRIG: UR3T_DMA UART1 transmit trigger control bit
0: Write 0 is invalid
1: Write 1 to start UR3T_DMA automatically sending data.

23.7.3 UR3T_DMA Status Register (DMA_UR3T_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_STA	FA52H	-	-	-	-	-	TXOVW	-	UR3TIF

UR3TIF: UR3T_DMA interrupt request flag bit. When the UR3T_DMA data transmission is completed, the hardware automatically sets UR3TIF to 1, and if the UR3T_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

TXOVW: UR3T_DMA data coverage flag. When UR3T_DMA is in the process of data transmission, and the UART writes the SBUF register to trigger the UART to send data again, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

23.7.4 UR3T_DMA transfer total byte register (DMA_UR3T_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_AMT	FA53H								

DMA_UR3T_AMT: Set the number of bytes of data that needs to be automatically sent.
Note: The actual number of bytes is (DMA_UR3T_AMT+1), that is, when DMA_UR3T_AMT is set to 0, 1 byte is transferred, and when DMA_UR3T_AMT is set to 255, 256 bytes are transferred.

23.7.5 UR3T_DMA transfer complete byte register (DMA_UR3T_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_DONE	FA54H								

DMA_UR3T_DONE: The number of bytes that have been sent so far.

23.7.6 UR3T_DMA Send Address Registers (DMA_UR3T_TXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_TXAH	FA55H	ADDR[15:8]							
DMA_UR3T_TXAL	FA56H	ADDR[7:0]							

DMA_UR3T_TXA: Set the source address of automatic data transmission. Data is read from this address when performing a UR3T_DMA operation.

23.7.7 UR3R_DMA Configuration Register (DMA_UR3R_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CFG	FA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	

UR3RIE: UR3R_DMA interrupt enable control bit

0: Disable UR3R_DMA interrupt

1: Enable UR3R_DMA interrupt

UR3RIP[1:0]: UR3R_DMA interrupt priority control bits

UR3RIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR3RPTY[1:0]: UR3R_DMA Data bus access priority control bits

UR3RPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.7.8 UR3R_DMA Control Register (DMA_UR3R_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CR	FA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO

ENUR3R: UR3R_DMA function enable control bit

0: Disable UR3R_DMA function

1: Enable UR3R_DMA function

TRIG: UR3R_DMA UART1 receive trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR3R_DMA receiving data automatically

CLRFIFO: Clear UR3R_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the UR3R_DMA operation, clear the built-in FIFO of the UR3R_DMA firstly

23.7.9 UR3R_DMA Status Register (DMA_UR3R_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_STA	FA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF

UR3RIF: UR3R_DMA interrupt request flag bit. When UR3R_DMA receives data, the hardware will automatically set

UR3RIF to 1. If the UR3R_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR3R_DMA receive data discard flag. During the UR3R_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the UR3R_DMA and the data received by the UR3R_DMA is automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

23.7.10 UR3R_DMA transfer total byte register (DMA_UR3R_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_AMT	FA5BH								

DMA_UR3R_AMT: Set the number of data bytes that need to automatically receive. **Note: The actual number of bytes is (DMA_UR3R_AMT+1), that is, when DMA_UR3R_AMT is set to 0, 1 byte is transferred, and when DMA_UR3R_AMT is set to 255, 256 bytes are transferred.**

23.7.11 UR3R_DMA transfer complete byte register (DMA_UR3R_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_DONE	FA5CH								

DMA_UR3R_DONE: The number of bytes that have been received currently.

23.7.12 UR3R_DMA Receive Address Registers (DMA_UR3T_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_RXAH	FA5DH	ADDR[15:8]							
DMA_UR3R_RXAL	FA5EH	ADDR[7:0]							

DMA_UR3R_RXA: Set the target address for automatically receiving data. Data will be written from this address when performing a UR3R_DMA operation.

23.8 Data exchange between UART4 and memory (UR4T_DMA, UR4R_DMA)

23.8.1 UR4T_DMA Configuration Register (DMA_UR4T_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CFG	FA50H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	

UR4TIE: UR4T_DMA interrupt enable control bit
0: Disable UR4T_DMA interrupt
1: Enable UR4T_DMA interrupt

UR4TIP[1:0]: UR4T_DMA interrupt priority control bits

UR4TIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR4TPTY[1:0]: UR4T_DMA Data bus access priority control bits

UR4TPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.8.2 UR4T_DMA Control Register (DMA_UR4T_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CR	FA51H	ENUR4T	TRIG	-	-	-	-	-	-

ENUR4T: UR4T_DMA function enable control bit
0: Disable UR4T_DMA function
1: Enable UR4T_DMA function

TRIG: UR4T_DMA UART1 transmit trigger control bit
0: Write 0 is invalid
1: Write 1 to start UR4T_DMA automatically sending data.

23.8.3 UR4T_DMA Status Register (DMA_UR4T_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_STA	FA52H	-	-	-	-	-	TXOVW	-	UR4TIF

UR4TIF: UR4T_DMA interrupt request flag bit. When the UR4T_DMA data transmission is completed, the hardware automatically sets UR4TIF to 1, and if the UR4T_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software.

TXOVW: UR4T_DMA data coverage flag. When UR4T_DMA is in the process of data transmission, and the UART writes the SBUF register to trigger the UART to send data again, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

23.8.4 UR4T_DMA transfer total byte register (DMA_UR4T_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_AMT	FA53H								

DMA_UR4T_AMT: Set the number of bytes of data that needs to be automatically sent.
Note: The actual number of bytes is (DMA_UR4T_AMT+1), that is, when DMA_UR4T_AMT is set to 0, 1 byte is transferred, and when DMA_UR4T_AMT is set to 255, 256 bytes are transferred.

23.8.5 UR4T_DMA transfer complete byte register (DMA_UR4T_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_DONE	FA54H								

DMA_UR4T_DONE: The number of bytes that have been sent so far.

23.8.6 UR4T_DMA Send Address Registers (DMA_UR4T_TXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_TXAH	FA55H	ADDR[15:8]							
DMA_UR4T_TXAL	FA56H	ADDR[7:0]							

DMA_UR4T_TXA: Set the source address of automatic data transmission. Data is read from this address when performing a UR4T_DMA operation.

23.8.7 UR4R_DMA Configuration Register (DMA_UR4R_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CFG	FA58H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	

UR4RIE: UR4R_DMA interrupt enable control bit

0: Disable UR4R_DMA interrupt

1: Enable UR4R_DMA interrupt

UR4RIP[1:0]: UR4R_DMA interrupt priority control bits

UR4RIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

UR4RPTY[1:0]: UR4R_DMA Data bus access priority control bits

UR4RPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.8.8 UR4R_DMA Control Register (DMA_UR4R_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CR	FA59H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO

ENUR4R: UR4R_DMA function enable control bit

0: Disable UR4R_DMA function

1: Enable UR4R_DMA function

TRIG: UR4R_DMA UART1 receive trigger control bit

0: Write 0 is invalid

1: Write 1 to start UR4R_DMA receiving data automatically

CLRFIFO: Clear UR4R_DMA receive FIFO control bit

0: Write 0 is invalid

1: Before starting the UR4R_DMA operation, clear the built-in FIFO of the UR4R_DMA firstly

23.8.9 UR4R_DMA Status Register (DMA_UR4R_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_STA	FA5AH	-	-	-	-	-	-	RXLOSS	UR4RIF

UR4RIF: UR4R_DMA interrupt request flag bit. When UR4R_DMA receives data, the hardware will automatically set

UR4RIF to 1. If the UR4R_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR4R_DMA receive data discard flag. During the UR4R_DMA operation, when the XRAM bus is too busy to clear the receive FIFO of the UR4R_DMA and the data received by the UR4R_DMA is automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

23.8.10 UR4R_DMA transfer total byte register (DMA_UR4R_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_AMT	FA5BH								

DMA_UR4R_AMT: Set the number of data bytes that need to automatically receive. **Note: The actual number of bytes is (DMA_UR4R_AMT+1), that is, when DMA_UR4R_AMT is set to 0, 1 byte is transferred, and when DMA_UR4R_AMT is set to 255, 256 bytes are transferred.**

23.8.11 UR4R_DMA transfer complete byte register (DMA_UR4R_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_DONE	FA5CH								

DMA_UR4R_DONE: The number of bytes that have been received currently.

23.8.12 UR4R_DMA Receive Address Registers (DMA_UR4T_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_RXAH	FA5DH	ADDR[15:8]							
DMA_UR4R_RXAL	FA5EH	ADDR[7:0]							

DMA_UR4R_RXA: Set the target address for automatically receiving data. Data will be written from this address when performing a UR4R_DMA operation.

23.9 Data exchange between LCM and memory (LCM_DMA)

23.9.1 LCM_DMA Configuration Register (DMA_LCM_CFG)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CFG	FA70H	LCMIE	ACT_TX	ACT_RX	-	LCMIP[1:0]		LCMPTY[1:0]	

LCMIE: LCM_DMA interrupt enable control bit

0: Disable LCM_DMA interrupt

1: Enable LCM_DMA interrupt

LCMIP [1:0]: LCM_DMA interrupt priority control bits

LCMIP[1:0]	Interrupt priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

LCMPTY[1:0]: LCM_DMA Data bus access priority control bits

LCMPTY [1:0]	Bus access priority
00	Lowest (0)
01	Lower (1)
10	Higher (2)
11	Highest (3)

23.9.2 LCM_DMA Control Register (DMA_LCM_CR)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CR	FA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	CLRFIFO

ENLCM: LCM_DMA function enable control bit

0: Disable LCM_DMA function

1: Enable LCM_DMA function

TRIGWC: LCM_DMA send command mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start LCM_DMA send command mode operation

TRIGWD: LCM_DMA send data mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start LCM_DMA send data mode operation

TRIGRC: LCM_DMA read command mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start LCM_DMA read command mode operation

TRIGRD: LCM_DMA read data mode trigger control bit

0: Write 0 is invalid

1: Write 1 to start LCM_DMA read data mode operation

23.9.3 LCM_DMA Status Register (DMA_LCM_STA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_STA	FA72H	-	-	-	-	-	-	TXOVW	LCMIF

LCMIF: LCM_DMA interrupt request flag bit. After the LCM_DMA data exchange is completed, the hardware automatically sets LCMIF to 1. If the LCM_DMA interrupt is enabled, the interrupt service routine is entered. The flag bit needs to be cleared by software

TXOVW: LCM_DMA data coverage flag. When LCM_DMA is in the process of data transmission, and LCMIF writes the LCMIFDATL and LCMIDDATH registers, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software

23.9.4 LCM_DMA transfer total byte register (DMA_LCM_AMT)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_AMT	FA73H								

DMA_LCM_AMT: Set the number of bytes that need to be read and written.

Note: The actual number of bytes read and written is (DMA_LCM_AMT+1), that is, when DMA_LCM_AMT is set to 0, 1 byte is transferred, and when DMA_LCM_AMT is set to 255, 256 bytes are transferred.

23.9.5 LCM_DMA transfer complete byte register (DMA_LCM_DONE)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_DONE	FA74H								

DMA_LCM_DONE: The number of bytes that have been transferred so far.

23.9.6 LCM_DMA Send Address Registers (DMA_LCM_TXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_TXAH	FA75H	ADDR[15:8]							
DMA_LCM_TXAL	FA76H	ADDR[7:0]							

DMA_LCM_TXA: Set the source address of automatic data transmission. Data is read from this address when performing an LCM_DMA operation.

23.9.7 LCM_DMA Receive Address Registers (DMA_LCM_RXAx)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_RXAH	FA77H	ADDR[15:8]							
DMA_LCM_RXAL	FA78H	ADDR[7:0]							

DMA_LCM_RXA: Set the target address for data transfer. Data is written from this address when performing an LCM_DMA operation.

23.10 Example Routines

23.10.1 UART1 interrupt mode and computer transceiver test - DMA receive timeout interrupt

C language code

```
//Operating frequency for test is 11.0592MHz

/************* Function Description    **************
UART1 works in full-duplex interrupt mode to send and receive data. PC sends data to the MCU, and the MCU will automatically store the received data in the DMA space. When the content received at one time is full of the set DMA space, the data in the storage space are output through the DMA automatic sending function of UART 1. Use UART receive interrupt to judge timeout, if no new data is received and timeout, it means that a string of data has been received, then outputs the received content, and clear the DMA space. If a timer is uased as baud rate generator, it is recommended to use 1T mode (unless 12T is used for low baud rates), and select a clock frequency that is divisible by the baud rate to improve accuracy.
When downloading, select the clock 22.1184MHz (users can modify the frequency by themselves).
*****************************************/

#include "stdio.h"
#include "stc8a8k64d4.h"

#define    MAIN_Fosc        22118400L          // Define the main clock (accurately calculate 115200 baud rate)
#define    Baudrate1        115200L
#define    Timer0_Reload    (65536UL -(MAIN_Fosc / 1000))

#define    DMA_AMT_LEN      255                //Set total bytes to be transferred(0~255)：DMA_AMT_LEN+1

bit        B_1ms;                              //1ms flag
bit        DMATxFlag;
bit        DMARxFlag;
bit        BusyFlag;
u8         Rx_cnt;
u8         RX1_TimeOut;

u8         xdata DMABuffer[256];

void UART1_config(u8 brt);
void DMA_Config(void);

void UartPutc(unsigned char dat)
{
    BusyFlag = 1;
    SBUF = dat;
    while(BusyFlag);
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}
```

```
void main(void)
{
    u16 i;

    P0M1 = 0x00;   P0M0 = 0x00;                 //set as quasi-bidirectional port
    P1M1 = 0x00;   P1M0 = 0x00;                 //set as quasi-bidirectional port
    P2M1 = 0x00;   P2M0 = 0x00;                 //set as quasi-bidirectional port
    P3M1 = 0x00;   P3M0 = 0x00;                 //set as quasi-bidirectional port
    P4M1 = 0x00;   P4M0 = 0x00;                 //set as quasi-bidirectional port
    P5M1 = 0x00;   P5M0 = 0x00;                 //set as quasi-bidirectional port
    P6M1 = 0x00;   P6M0 = 0x00;                 //set as quasi-bidirectional port
    P7M1 = 0x00;   P7M0 = 0x00;                 //set as quasi-bidirectional port

    for(i=0; i<256; i++)
    {
        DMABuffer[i] = i;
    }

    AUXR = 0x80;                                //Timer0 set as 1T, 16 bits timer auto-reload,
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;                                    //Timer0 interrupt enable
    TR0 = 1;                                    //Tiner0 run

    UART1_config(1);                            //Use Timer1 as baud rate generator.
    DMA_Config();
    EA = 1;                     //enable CPU interrupt

    printf("UART1 DMA Timeout Programme!\r\n");  //UART1 sends a string
    DMATxFlag = 0;
    DMARxFlag = 0;

    while (1)
    {
        if((DMATxFlag) && (DMARxFlag))          //Judging the send completion flag and the reception completion
flag
        {
            Rx_cnt = 0;
            RX1_TimeOut = 0;
            printf("\r\nUART1 DMA FULL!\r\n");  //UART1 sends a string
            DMATxFlag = 0;
            DMA_UR1T_CR = 0xc0;                 //bit7 1:Enable UART1_DMA,
                                                //bit6 1:Start UART1_DMA automatic transmission
            DMARxFlag = 0;
            DMA_UR1R_CR = 0xa1;                 //bit7 1:Enable UART1_DMA,
                                                //bit5 1:Start UART1_DMA automatic reception,
                                                //bit0 1:clear FIFO
        }

        if(B_1ms)                               //reach 1ms
        {
            B_1ms = 0;
            if(RX1_TimeOut > 0)                 //timeout count
            {
                if(--RX1_TimeOut == 0)
                {
                    DMA_UR1R_CR = 0x00;         //Disable UART1_DMA
                    printf("\r\nUART1 Timeout!\r\n");  //UART1 sends a string
```

```
                        for(i=0;i<Rx_cnt;i++) UartPutc(DMABuffer[i]);
                        printf("\r\n");

                        Rx_cnt = 0;
                        DMA_UR1R_CR = 0xa1;             //bit7 1:Enable UART1_DMA,
                                                        //bit5 1:Start UART1_DMA automatic reception,
                                                        //bit0 1:clear FIFO
                    }
                }
            }
        }
}

void DMA_Config(void)
{
    P_SW2 = 0x80;
    DMA_UR1T_CFG = 0x80;                //bit7 1:Enable Interrupt
    DMA_UR1T_STA = 0x00;
    DMA_UR1T_AMT = DMA_AMT_LEN;         //Set total bytes to be transferred：n+1
    DMA_UR1T_TXA = DMABuffer;
    DMA_UR1T_CR = 0xc0;                 //bit7 1:Enable UART1_DMA,
                                        //bit6 1:Start UART1_DMA automatic transmission

    DMA_UR1R_CFG = 0x80;                //bit7 1:Enable Interrupt
    DMA_UR1R_STA = 0x00;
    DMA_UR1R_AMT = DMA_AMT_LEN;         //Set total bytes to be transferred：n+1
    DMA_UR1R_RXA = DMABuffer;
    DMA_UR1R_CR = 0xa1;                 //bit7 1:Enable UART1_DMA,
                                        //bit5 1:Start UART1_DMA automatic reception,
                                        //bit0 1:clear FIFO
}

void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                    //Timer stop
    AUXR &= ~(1<<3);                    //Timer2 set As Timer
    AUXR |=  (1<<2);                    //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2  &= ~(1<<2);                    //Disable interrupts
    AUXR |=  (1<<4);                    //Timer run enable
}

void UART1_config(u8 brt)               //select baud rate:
                                        //2: Use Timer2 as baud rate generator,
                                        //Other values: Use Timer1 as baud rate generator.
{
    /********** Use Timer2 as baud rate generator **********/
    if(brt == 2)
    {
        AUXR |= 0x01;                   //S1 BRT Use Timer2;
        SetTimer2Baudraye(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /********** Use Timer1 as baud rate generator **********/
    else
    {
```

```
        TR1 = 0;
        AUXR &= ~0x01;                          //S1 BRT Use Timer1;
        AUXR |=  (1<<6);                        //Timer1 set as 1T mode
        TMOD &= ~(1<<6);                        //Timer1 set As Timer
        TMOD &= ~0x30;                          //Timer1_16bitAutoReload;
        TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
        TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
        ET1 = 0;                                //Disable interrupts
        INTCLKO &= ~0x02;                       //Does not output clock
        TR1  = 1;
    }
    /*********************************************/

    SCON = (SCON & 0x3f) | 0x40;                //UART1 mode:
                                                //0x00: Synchronous shift output,
                                                //0x40: 8-bit data, variable baud rate,
                                                //0x80: 9-bit data, fixed baud rate,
                                                //0xc0: 9-bit data, variable baud rate
//  PS = 1;                                     //high priority interrupt
    ES = 1;                                     //enable interrupt
    REN = 1;                                    //enable to receive
    P_SW1 &= 0x3f;
    P_SW1 |= 0x00;                              //UART1 switch to:
                                                //0x00: P3.0 P3.1,
                                                //0x40: P3.6 P3.7,
                                                //0x80: P1.6 P1.7,
                                                //0xC0: P4.3 P4.4

    RX1_TimeOut = 0;
}

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        Rx_cnt++;
        if(Rx_cnt >= DMA_AMT_LEN) Rx_cnt = 0;
        RX1_TimeOut = 5;                        //If no new data is received within 5ms, it is determined that a string
of data has been received.
    }

    if(TI)
    {
        TI = 0;
        BusyFlag = 0;
    }
}

void timer0 (void) interrupt 1
{
    B_1ms = 1;                                  //1ms flag
}

void UART1_DMA_Interrupt(void) interrupt 13
{
    if (DMA_UR1T_STA & 0x01)                    //send completed
    {
```

```
        DMA_UR1T_STA &= ~0x01;
        DMATxFlag = 1;
    }
    if (DMA_UR1T_STA & 0x04)                    //data coverage
    {
        DMA_UR1T_STA &= ~0x04;
    }

    if (DMA_UR1R_STA & 0x01)                    //Receive complete
    {
        DMA_UR1R_STA &= ~0x01;
        DMARxFlag = 1;
    }
    if (DMA_UR1R_STA & 0x02)                    //data is discarded
    {
        DMA_UR1R_STA &= ~0x02;
    }
}
```

```
//File: ISR.ASM
//Interrupts with interrupt numbers greater than 31 require interrupt entry address remapping processing
        CSEG    AT      012BH           ;P0INT_VECTOR
        JMP             P0INT_ISR
        CSEG    AT      0133H           ;P1INT_VECTOR
        JMP             P1INT_ISR
        CSEG    AT      013BH           ;P2INT_VECTOR
        JMP             P2INT_ISR
        CSEG    AT      0143H           ;P3INT_VECTOR
        JMP             P3INT_ISR
        CSEG    AT      014BH           ;P4INT_VECTOR
        JMP             P4INT_ISR
        CSEG    AT      0153H           ;P5INT_VECTOR
        JMP             P5INT_ISR
        CSEG    AT      015BH           ;P6INT_VECTOR
        JMP             P6INT_ISR
        CSEG    AT      0163H           ;P7INT_VECTOR
        JMP             P7INT_ISR
        CSEG    AT      016BH           ;P8INT_VECTOR
        JMP             P8INT_ISR
        CSEG    AT      0173H           ;P9INT_VECTOR
        JMP             P9INT_ISR
        CSEG    AT      017BH           ;M2MDMA_VECTOR
        JMP             M2MDMA_ISR
        CSEG    AT      0183H           ;ADCDMA_VECTOR
        JMP             ADCDMA_ISR
        CSEG    AT      018BH           ;SPIDMA_VECTOR
        JMP             SPIDMA_ISR
        CSEG    AT      0193H           ;U1TXDMA_VECTOR
        JMP             U1TXDMA_ISR
        CSEG    AT      019BH           ;U1RXDMA_VECTOR
        JMP             U1RXDMA_ISR
        CSEG    AT      01A3H           ;U2TXDMA_VECTOR
        JMP             U2TXDMA_ISR
        CSEG    AT      01ABH           ;U2RXDMA_VECTOR
        JMP             U2RXDMA_ISR
        CSEG    AT      01B3H           ;U3TXDMA_VECTOR
        JMP             U3TXDMA_ISR
```

```
            CSEG    AT      01BBH                   ;U3RXDMA_VECTOR
            JMP             U3RXDMA_ISR
            CSEG    AT      01C3H                   ;U4TXDMA_VECTOR
            JMP             U4TXDMA_ISR
            CSEG    AT      01CBH                   ;U4RXDMA_VECTOR
            JMP             U4RXDMA_ISR
            CSEG    AT      01D3H                   ;LCMDMA_VECTOR
            JMP             LCMDMA_ISR
            CSEG    AT      01DBH                   ;LCMIF_VECTOR
            JMP             LCMIF_ISR

P0INT_ISR:
P1INT_ISR:
P2INT_ISR:
P3INT_ISR:
P4INT_ISR:
P5INT_ISR:
P6INT_ISR:
P7INT_ISR:
P8INT_ISR:
P9INT_ISR:
M2MDMA_ISR:
ADCDMA_ISR:
SPIDMA_ISR:
U1TXDMA_ISR:
U1RXDMA_ISR:
U2TXDMA_ISR:
U2RXDMA_ISR:
U3TXDMA_ISR:
U3RXDMA_ISR:
U4TXDMA_ISR:
U4RXDMA_ISR:
LCMDMA_ISR:
LCMIF_ISR:

            JMP             006BH

            END
```

23.10.2 UART1 interrupt mode and computer transceiver test - DMA data check

C language code

```
// Operating frequency for test is 11.0592MHz

/************* Function Description    **************
UART1 works in full-duplex interrupt mode to send and receive data. PC sends data to the MCU, and the MCU will automatically store the received data in the DMA space. The last two bytes of the data packet are used as the check digit, and the routine performs the check using the crc16_ccitt algorithm. When the DMA space is full of the content of the set size, the valid data is checked and calculated, and then compared with the last two check digits. The data in the storage space is output through the DMA automatic sending function of UART1. If a timer is uased as baud rate generator, it is recommended to use 1T mode (unless 12T is used for low baud rates), and select a clock frequency that is divisible by the baud rate to improve accuracy.
```

```
When downloading, select the clock 22.1184MHz (users can modify the frequency by themselves).
*****************************************/

#include "stdio.h"
#include "STC8Hxxxxx.h"
#include "crc16.h"

#define   MAIN_Fosc       22118400L          // Define the main clock (accurately calculate 115200 baud rate)
#define   Baudrate1       115200L

#define   DMA_AMT_LEN     255                ///Set total bytes to be transferred(0~255) ：DMA_AMT_LEN+1

bit       DMATxFlag;
bit       DMARxFlag;

u8        xdata DMABuffer[256];

void UART1_config(u8 brt);
void DMA_Config(void);

void UartPutc(unsigned char dat)
{
    SBUF = dat;
    while(TI == 0);
    TI = 0;
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

/*****CRC calculation function********/
u16 crc16_ccitt(u8 *pbuf, u16 len)
{
    unsigned short code crc16_ccitt_table[256] =
    {
        0x0000,  0x1021,  0x2042,  0x3063,  0x4084,  0x50A5,  0x60C6,  0x70E7,
        0x8108,  0x9129,  0xA14A,  0xB16B,  0xC18C,  0xD1AD,  0xE1CE,  0xF1EF,
        0x1231,  0x0210,  0x3273,  0x2252,  0x52B5,  0x4294,  0x72F7,  0x62D6,
        0x9339,  0x8318,  0xB37B,  0xA35A,  0xD3BD,  0xC39C,  0xF3FF,  0xE3DE,
        0x2462,  0x3443,  0x0420,  0x1401,  0x64E6,  0x74C7,  0x44A4,  0x5485,
        0xA56A,  0xB54B,  0x8528,  0x9509,  0xE5EE,  0xF5CF,  0xC5AC,  0xD58D,
        0x3653,  0x2672,  0x1611,  0x0630,  0x76D7,  0x66F6,  0x5695,  0x46B4,
        0xB75B,  0xA77A,  0x9719,  0x8738,  0xF7DF,  0xE7FE,  0xD79D,  0xC7BC,
        0x48C4,  0x58E5,  0x6886,  0x78A7,  0x0840,  0x1861,  0x2802,  0x3823,
        0xC9CC,  0xD9ED,  0xE98E,  0xF9AF,  0x8948,  0x9969,  0xA90A,  0xB92B,
        0x5AF5,  0x4AD4,  0x7AB7,  0x6A96,  0x1A71,  0x0A50,  0x3A33,  0x2A12,
        0xDBFD,  0xCBDC,  0xFBBF,  0xEB9E,  0x9B79,  0x8B58,  0xBB3B,  0xAB1A,
        0x6CA6,  0x7C87,  0x4CE4,  0x5CC5,  0x2C22,  0x3C03,  0x0C60,  0x1C41,
        0xEDAE,  0xFD8F,  0xCDEC,  0xDDCD,  0xAD2A,  0xBD0B,  0x8D68,  0x9D49,
        0x7E97,  0x6EB6,  0x5ED5,  0x4EF4,  0x3E13,  0x2E32,  0x1E51,  0x0E70,
        0xFF9F,  0xEFBE,  0xDFDD,  0xCFFC,  0xBF1B,  0xAF3A,  0x9F59,  0x8F78,
        0x9188,  0x81A9,  0xB1CA,  0xA1EB,  0xD10C,  0xC12D,  0xF14E,  0xE16F,
        0x1080,  0x00A1,  0x30C2,  0x20E3,  0x5004,  0x4025,  0x7046,  0x6067,
        0x83B9,  0x9398,  0xA3FB,  0xB3DA,  0xC33D,  0xD31C,  0xE37F,  0xF35E,
        0x02B1,  0x1290,  0x22F3,  0x32D2,  0x4235,  0x5214,  0x6277,  0x7256,
```

```
        0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
        0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
        0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
        0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
        0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
        0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
        0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
        0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
        0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
        0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
        0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
        0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
    };

    u16 crc16 = 0x0000;
    u16 crc_h8, crc_l8;

    while( len-- ) {
        crc_h8 = (crc16 >> 8);
        crc_l8 = (crc16 << 8);
        crc16 = crc_l8 ^ crc16_ccitt_table[crc_h8 ^ *pbuf];
        pbuf++;
    }

    return crc16;
}

void main(void)
{
    u16 i;
    u16 CheckSum;

    P0M1 = 0x00;   P0M0 = 0x00;                    //set as quasi-bidirectional port
    P1M1 = 0x00;   P1M0 = 0x00;                    //set as quasi-bidirectional port
    P2M1 = 0x00;   P2M0 = 0x00;                    //set as quasi-bidirectional port
    P3M1 = 0x00;   P3M0 = 0x00;                    //set as quasi-bidirectional port
    P4M1 = 0x00;   P4M0 = 0x00;                    //set as quasi-bidirectional port
    P5M1 = 0x00;   P5M0 = 0x00;                    //set as quasi-bidirectional port
    P6M1 = 0x00;   P6M0 = 0x00;                    //set as quasi-bidirectional port
    P7M1 = 0x00;   P7M0 = 0x00;                    //set as quasi-bidirectional port

    for(i=0; i<256; i++)
    {
        DMABuffer[i] = i;
    }

    P_SW2 = 0x80;
    DMA_UR1T_STA = 0x00;
    UART1_config(1);
    printf("UART1 DMA CRC Programme!\r\n");

    DMA_Config();
    EA = 1;                                        //enable CPU interrupt

    DMATxFlag = 0;
    DMARxFlag = 0;

    while (1)
```

```
    {
        if((DMATxFlag) && (DMARxFlag))
        {
            CheckSum = crc16_ccitt(DMABuffer,DMA_AMT_LEN-1);
            if(((u8)CheckSum == DMABuffer[DMA_AMT_LEN-1]) &&
            ((u8)(CheckSum>>8) == DMABuffer[DMA_AMT_LEN]))
            {
                printf("\r\nOK! CheckSum = %04x\r\n",CheckSum);
            }
            else
            {
                printf("\r\nERROR! CheckSum = %04x\r\n",CheckSum);
            }
            DMATxFlag = 0;
            DMA_UR1T_CR = 0xc0;                 //bit7 1:Enable UART1_DMA,
                                                //bit6 1:Start UART1_DMA automatic transmission
            DMARxFlag = 0;
            DMA_UR1R_CR = 0xa1;                 //bit7 1:Enable UART1_DMA,
                                                //bit5 1:Start UART1_DMA automatic reception,
                                                //bit0 1:clear FIFO
        }
    }
}

void DMA_Config(void)
{
    P_SW2 = 0x80;
    DMA_UR1T_CFG = 0x80;                        //bit7 1:Enable Interrupt
    DMA_UR1T_STA = 0x00;
    DMA_UR1T_AMT = DMA_AMT_LEN;                 // Set total bytes to be transferred：n+1
    DMA_UR1T_TXA = DMABuffer;
    DMA_UR1T_CR = 0xc0;                         //bit7 1:Enable UART1_DMA,
                                                //bit6 1:Start UART1_DMA automatic transmission

    DMA_UR1R_CFG = 0x80;                        //bit7 1:Enable Interrupt
    DMA_UR1R_STA = 0x00;
    DMA_UR1R_AMT = DMA_AMT_LEN;                 // Set total bytes to be transferred：n+1
    DMA_UR1R_RXA = DMABuffer;
    DMA_UR1R_CR = 0xa1;                         //bit7 1:Enable UART1_DMA,
                                                //bit5 1:Start UART1_DMA automatic reception, bit0 1:clear
FIFO
}

void SetTimer2Baudraye(u16 dat)                 //select baud rate:
                                                //2: Use Timer2 as baud rate generator,
                                                //Other values: Use Timer1 as baud rate generator.
{
    AUXR &= ~(1<<4);                            //Timer stop
    AUXR &= ~(1<<3);                            //Timer2 set As Timer
    AUXR |=  (1<<2);                            //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2  &= ~(1<<2);                            //Disable interrupts
    AUXR |=  (1<<4);                            //Timer run enable
}

void UART1_config(u8 brt)                       //select baud rate:
                                                //2: Use Timer2 as baud rate generator
```

```
                                                    //Other values: Use Timer1 as baud rate generator.
{
    /************  Use Timer2 as baud rate generator *****************/
    if(brt == 2)
    {
        AUXR |= 0x01;                               //S1 BRT Use Timer2;
        SetTimer2Baudraye(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /************ Use Timer1 as baud rate generator *****************/
    else
    {
        TR1 = 0;
        AUXR &= ~0x01;                              //S1 BRT Use Timer1;
        AUXR |=  (1<<6);                            //Timer1 set as 1T mode
        TMOD &= ~(1<<6);                            //Timer1 set As Timer
        TMOD &= ~0x30;                              //Timer1_16bitAutoReload;
        TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
        TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
        ET1 = 0;                                    //Disable interrupts
        INTCLKO &= ~0x02;                           //Does not output clock
        TR1  = 1;
    }
    /************************************************/

    SCON = (SCON & 0x3f) | 0x40;                    //UART1 mode,
                                                    //0x00: Synchronous shift output,
                                                    //0x40: 8-bit data, variable baud rate,
                                                    //0x80: 9-bit data, fixed baud rate,
                                                    //0xc0: 9-bit data, variable baud rate
//  PS  = 1;                                        //high priority interrupt
//  ES  = 1;                                        //enable interrupt
    REN = 1;                                        //enable to receive
    P_SW1 &= 0x3f;
    P_SW1 |= 0x00;
}

void UART1_DMA_Interrupt(void) interrupt 13
{
    if (DMA_UR1T_STA & 0x01)                        //send completed
    {
        DMA_UR1T_STA &= ~0x01;
        DMATxFlag = 1;
    }
    if (DMA_UR1T_STA & 0x04)                        //data coverage
    {
        DMA_UR1T_STA &= ~0x04;
    }

    if (DMA_UR1R_STA & 0x01)                        //Receive complete
    {
        DMA_UR1R_STA &= ~0x01;
        DMARxFlag = 1;
    }
    if (DMA_UR1R_STA & 0x02)                        //data is discarded
    {
        DMA_UR1R_STA &= ~0x02;
    }
```

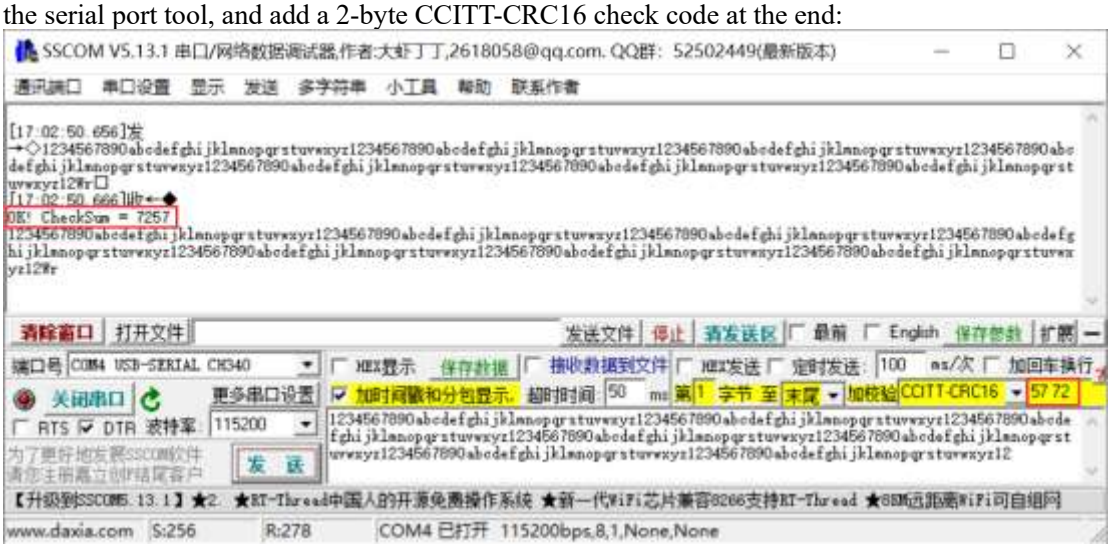
```
}
```

```
//File: ISR.ASM
//Interrupts with interrupt numbers greater than 31 require interrupt entry address remapping processing
            CSEG    AT      012BH               ;P0INT_VECTOR
            JMP             P0INT_ISR
            CSEG    AT      0133H               ;P1INT_VECTOR
            JMP             P1INT_ISR
            CSEG    AT      013BH               ;P2INT_VECTOR
            JMP             P2INT_ISR
            CSEG    AT      0143H               ;P3INT_VECTOR
            JMP             P3INT_ISR
            CSEG    AT      014BH               ;P4INT_VECTOR
            JMP             P4INT_ISR
            CSEG    AT      0153H               ;P5INT_VECTOR
            JMP             P5INT_ISR
            CSEG    AT      015BH               ;P6INT_VECTOR
            JMP             P6INT_ISR
            CSEG    AT      0163H               ;P7INT_VECTOR
            JMP             P7INT_ISR
            CSEG    AT      016BH               ;P8INT_VECTOR
            JMP             P8INT_ISR
            CSEG    AT      0173H               ;P9INT_VECTOR
            JMP             P9INT_ISR
            CSEG    AT      017BH               ;M2MDMA_VECTOR
            JMP             M2MDMA_ISR
            CSEG    AT      0183H               ;ADCDMA_VECTOR
            JMP             ADCDMA_ISR
            CSEG    AT      018BH               ;SPIDMA_VECTOR
            JMP             SPIDMA_ISR
            CSEG    AT      0193H               ;U1TXDMA_VECTOR
            JMP             U1TXDMA_ISR
            CSEG    AT      019BH               ;U1RXDMA_VECTOR
            JMP             U1RXDMA_ISR
            CSEG    AT      01A3H               ;U2TXDMA_VECTOR
            JMP             U2TXDMA_ISR
            CSEG    AT      01ABH               ;U2RXDMA_VECTOR
            JMP             U2RXDMA_ISR
            CSEG    AT      01B3H               ;U3TXDMA_VECTOR
            JMP             U3TXDMA_ISR
            CSEG    AT      01BBH               ;U3RXDMA_VECTOR
            JMP             U3RXDMA_ISR
            CSEG    AT      01C3H               ;U4TXDMA_VECTOR
            JMP             U4TXDMA_ISR
            CSEG    AT      01CBH               ;U4RXDMA_VECTOR
            JMP             U4RXDMA_ISR
            CSEG    AT      01D3H               ;LCMDMA_VECTOR
            JMP             LCMDMA_ISR
            CSEG    AT      01DBH               ;LCMIF_VECTOR
            JMP             LCMIF_ISR

P0INT_ISR:
P1INT_ISR:
P2INT_ISR:
P3INT_ISR:
P4INT_ISR:
P5INT_ISR:
```

```
P6INT_ISR:
P7INT_ISR:
P8INT_ISR:
P9INT_ISR:
M2MDMA_ISR:
ADCDMA_ISR:
SPIDMA_ISR:
U1TXDMA_ISR:
U1RXDMA_ISR:
U2TXDMA_ISR:
U2RXDMA_ISR:
U3TXDMA_ISR:
U3RXDMA_ISR:
U4TXDMA_ISR:
U4RXDMA_ISR:
LCMDMA_ISR:
LCMIF_ISR:

        JMP     006BH

        END
```

Code testing method

According to the predefined DMA packet length (for example: 256 bytes), send a packet of data (254 bytes) through the serial port tool, and add a 2-byte CCITT-CRC16 check code at the end:



After the MCU receives the entire packet of data (256 bytes), it performs CRC16 check on the first 254 bytes of data, and the obtained check code is compared with the last two bytes. If the values are equal, print "OK!" and calculate the check code, and then output the content read from the DMA space.

If the checksum values are not equal, print "ERROR!" and the calculated checksum.

24 Enhanced Dual Data Pointer

Two 16-bit data pointers are integrated in STC8A8K64D4 series of microcontrollers. The data pointers can be increased or decreased automatically by the program control, and they can be switched automatically.

24.1 Related special function registers

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	Data pointer low byte register	82H									0000,0000
DPH	Data pointer high byte register	83H									0000,0000
DPL1	2nd Data pointer low byte	E4H									0000,0000
DPH1	2nd Data pointer high byte	E5H									0000,0000
DPS	DPTR Selection Register	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR Timing control register	AEH									0000,0000

24.1.1 1st 16-bit Data Pointer Registers (DPTR0)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL is Data pointer 0 low byte.

DPH is Data pointer 0 high byte.

The combination of DPL and DPH is the first 16-bit data pointer register DPTR0.

24.1.2 2nd 16-bit Data Pointer Registers (DPTR1)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1 is Data pointer 1 low byte.

DPH1 is Data pointer 1 low byte.

The combination of DPL1 and DPH1 is the second 16-bit data pointer register DPTR1.

24.1.3 DPTR control register

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1: DPTR1 auto-increment or auto-decrement mode control bit
- 0: DPTR1 auto-increment mode
- 1: DPTR1 auto-decrement mode

ID0: DPTR0 auto-increment or auto-decrement mode control bit
- 0: DPTR0 auto-increment mode
- 1: DPTR0 auto-decrement mode

TSL: DPTR0/DPTR1 auto-switch control bit (invert SEL automatically)
- 0: DPTR0/DPTR1 auto switch is disabled.
- 1: DPTR0/DPTR1 auto switch is enabled.

If the TSL bit is set, the SEL bit will be inverted automatically after the relevant instruction is executed.

Instructions related to TSL include:

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

AU1/AU0: Enable DPTR1 / DPTR0 Automatic increment / decrement control bit

0: disable Automatic increment / decrement function

1: enable Automatic increment / decrement function

Note: In write-protect mode, AU0 and AU1 can not be enabled individually. AU0 will be enabled automatically if AU1 is enabled. If AU0 is enabled alone, there is no effect to AU1. To enable AU1 or AU0 independently, the TA register must be used to trigger the DPS protection mechanism. For more detail, please refer to the description of the TA register. In addition, DPTR0 / DPTR1 will be incremented / decremented automatically only after executing the following three instructions.

```
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

SEL: DPTR register select bit.

0: Default. DPTR0 is selected as current Data pointer.

1: DPTR1 is selected as current Data pointer.

The selection of current DPTR using SEL is valid for the following instructions,

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
JMP     @A+DPTR
```

24.1.4 Data Pointer control register (TA)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

AU1 and AU0 in the DPS register is write-protected by TA register. Since the program can not write AU1 and AU0 separately, TA register must be used to trigger enabling AU1 or AU0 independently. TA is a write-only register.

The following steps must be executed if you need to enable AU1 or AU0 separately.

```
CLR     EA          ; disable interrupt (it is necessary to disable interrupt)
MOV     TA,#0AAH    ; write the trigger command sequence 1
                    ; any other instructions can not be here
MOV     TA,#55H     ; write the trigger command sequence 2
                    ; any other instructions can not be here
MOV     DPS,#xxH    ; Write-protection is temporarily disabled,
                    ; and any value can be written to the DPS
                    ; DSP enters the write-protected mode again
SETB    EA          ; enable interrupt if necessary
```

24.2 Example Routines

24.2.1 Example Routine 1

Copy 4 bytes of data stored in program space 1000H to 1003H in reverse to 0100H to 0103H of the extended RAM, that is,

C:1000H -> X:0103H
C:1001H -> X:0102H
C:1002H -> X:0101H
C:1003H -> X:0100H

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     DPS,#00100000B          ; enable TSL and select DPTR0
            MOV     DPTR,#1000H     ; write 1000H to DPTR0, and then select DPTR1 as current DPTR
            MOV     DPTR,#0103H             ; write 0103H to DPTR1
            MOV     DPS,#10111000B          ;set DPTR1 in auto decreasing mode,
                                    ;DPTR0 in auto increasing mode,enable TSL, AU0 and AU1,
                                    ;select DPTR0 as the current DPTR
            MOV     R7,#4                   ; set the counter of copies
COPY_NEXT:
            CLR     A                       ;
            MOVC    A,@A+DPTR               ; Read data from the program space indicated by DPTR0,
                    ; When done, DPTR0 increments automatically and select DPTR1 as the current DPTR
            MOVX    @DPTR,A                 ; write the content of ACC to XDARA indicated by DPTR1,
```

```
                    ; When done, DPTR1 decrements automatically and select DPTR0 as the current DPTR
            DJNZ    R7,COPY_NEXT            ;

            SJMP    $

            END
```

24.2.2 Example Routine 2

Send the data stored in the extended RAM 0100H to 0103H to P0 port successively.

Assembly code

```
;Operating frequency for test is 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            CLR     EA                  ; disable interrupt
            MOV     TA,#0AAH            ; Write DPS write-protection trigger command 1
            MOV     TA,#55H             ; Write DPS write-protection trigger command 2
            MOV     DPS,#00001000B      ; set DPTR0 in increasing mode,enable AU0 independently,
                                        ;and select DPTR0

            SETB    EA                  ; enable interrupt
            MOV     DPTR,#0100H         ; write 0100H to DPTR0
            MOVX    A,@DPTR             ; Read data from XRAM indicated by DPTR0,
                                        ;and then DPTR0 increments automatically
            MOV     P0,A                ; output the datum to Port0
            MOVX    A,@DPTR             ; Read data from XRAM indicated by DPTR0,
```

```
                                ;and then DPTR0 increments automatically
MOV         P0,A                ; output the datum to Port0
MOVX        A,@DPTR             ; Read data from XRAM indicated by DPTR0,
                                ;and then DPTR0 increments automatically
MOV         P0,A                ; output the datum to Port0
MOVX        A,@DPTR             ; Read data from XRAM indicated by DPTR0,
                                ;and then DPTR0 increments automatically
MOV         P0,A                ; output the datum to Port0

SJMP        $

END
```

25 MDU16 Hardware 16-bit Multiplier and Divider

A 16-bit hardware multiply / divide unit MDU16 is integrated in some microcontrollers of the STC8A8K64D4 series.

The following data operations are supported:

- Data standardization (need 3-20 clocks of computing time)
- Logic left shift (need 3～18 clocks of operation time)
- Logic shift right (need 3～18 clocks of operation time)
- 16 bits multiplied by 16 bits (it takes 10 clocks of operation time)
- 16 bits divided by 16 bits (need 9 clocks of operation time)
- 32 bits divided by 16 bits (requires 17 clocks of operation time)

All operations are based on unsigned integer data types.

25.1 Registers Related to MDU16

Symbol	Description	Address	Bit Address and Symbol								Reset Value
			B7	B6	B5	B4	B3	B2	B1	B0	
MD3	MDU Data Register	FCF0H	MD3[7:0]								0000,0000
MD2	MDU Data Register	FCF1H	MD2[7:0]								0000,0000
MD1	MDU Data Register	FCF2H	MD1[7:0]								0000,0000
MD0	MDU Data Register	FCF3H	MD0[7:0]								0000,0000
MD5	MDU Data Register	FCF4H	MD5[7:0]								0000,0000
MD4	MDU Data Register	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU Mode Control Register	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU Operation Control Register	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000

25.1.1 Operand 1 Data Registers (MD0~MD3)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
MD3	FCF0H	MD3[7:0]							
MD2	FCF1H	MD2[7:0]							
MD1	FCF2H	MD1[7:0]							
MD0	FCF3H	MD0[7:0]							

25.1.2 Operand 2 Data Registers (MD4~MD5)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
MD5	FCF4H	MD5[7:0]							
MD4	FCF5H	MD4[7:0]							

32-bit division by 16-bit division:

Dividend: {MD3,MD2,MD1,MD0}
Divisor: {MD5,MD4}
Quotient: {MD3,MD2,MD1,MD0}
Remainder: {MD5,MD4}

16-bit division by 16-bit division:

Dividend: {MD1,MD0}
Divisor: {MD5,MD4}
Quotient: {MD1,MD0}
Remainder: {MD5,MD4}

16-bit multiplication by 16-bit multiplication:

Multiplicand: {MD1,MD0}

Multiplier: {MD5,MD4}
Product: {MD3,MD2,MD1,MD0}

32-bit logical shift left / logical shift right
Operand: {MD3,MD2,MD1,MD0}

32-bit data normalization:
Operand: {MD3,MD2,MD1,MD0}

25.1.3 MDU Mode Control Register (ARCON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
ARCON	FCF6H	MODE[2:0]			SC[4:0]				

MODE[2:0]: MDU mode selection

MODE[2:0]	Mode	Clocks	Desciption
1	Logical right shift	3～18	Shift the data in {MD3, MD2, MD1, MD0} right by SC [4: 0] bits, MD3 high-order complement 0
2	Logical left shift	3～18	Shift the data in {MD3, MD2, MD1, MD0} SC [4: 0] bits to the left, low-order complement of MD0
3	Data normalization	3～20	Perform logical left shift on the data in {MD3, MD2, MD1, MD0}, shift out all the high-order 0s of the data, make the highest bit of MD3 be 1, and the number of logical left shifts is recorded in SC [4: 0].
4	16-bit multiplication	10	{MD1,MD0}×{MD5,MD4}={MD3,MD2,MD1,MD0}
5	16-bit division	9	{MD1,MD0}÷{MD5,MD4}= {MD1,MD0}…{MD5,MD4}
6	32-bit division	17	{MD3,MD2,MD1,MD0}÷{MD5,MD4}= {MD3,MD2,MD1,MD0}…{MD5,MD4}
Others	Invalid		

SC[4:0]: Data shift bits
When the MDU is in shift mode, SC is used to set the number of bits for left/right shift
When MDU is in data normalization mode, SC is the actual number of bits moved by the data after data normalization

25.1.4 MDU Operation Control Register (OPCON)

Symbol	Address	B7	B6	B5	B4	B3	B2	B1	B0
OPCON	FCF7H	-	MDOV	-	-	-	-	RST	ENOP

MDOV: MDU Overflow flag (read-only flag)
MDOV is set by hardware automatically in the following situations:
1. When the divisor is 0;
2. When the product of multiplication is greater than 0FFFFH;
When software writes OPCON.0 (EN) or writes ARCON, MDOV is cleared by the hardware automatically.

RST: Software resets the MDU multiplication and division unit. Writing 1 to it will trigger a software reset. It is cleared by the hardware automatically after the MDU reset is complete.
Note: The value of the ARCON register is cleared when software resets the MDU multiply and divide unit.

ENOP: MDU enable bit.
Writing 1 to this bit will trigger the MDU module to start calculation. When the MDU calculation is completed, ENOP is cleared to 0 by the hardware automatically. After setting ENOP to 1, the software can query ENOP cyclically. When ENOP changes from 1 to 0, the calculation is completed.

29.2 Netizens' application of MDU16 (provide ideas, for reference only)

Netizen 1: "Data normalization is illustrated with the following simple example"

There is a 7-digit decimal precision data: 0.0000123. Due to the limited data bit width, if you need to use the bit width effectively, you need to shift the previous data to the left. For example, the data after the left shift is 0.123e-4, and the exponent -4 is stored in the Another register that records the number of left shifts is the size of the record index. The original register data is converted to 0.123. This frees up the bit width on the right side of the data to ensure the accuracy of subsequent calculations. The above is just a simple explanation of the principle of normalization in decimal, and the principle of binary is the same. Among them, the conversion between floating point and fixed point (integer) must use the principle of normalization. If the exponents of the two floating point numbers are different when adding and subtracting, they also need to be normalized (this process is called pair order). If the exponents of two floating-point numbers are very different, there will be a problem of large numbers eating decimals when adding and subtracting. For example: 0.123e+4 - 0.12e-4 = 0.123e+4 - 0.0000000012e+4 = 0.123e+4. The result is the minuend, because the exponents of the two floating-point numbers need to be exactly the same before the subtraction operation (for the order), you need to shift the floating point number with a small exponent to make the exponent become +4. However, the data width is limited to 7 decimal places, and the data to the right of the number 0.0000000012e+4 will be truncated to 0.0000000e +4 = 0.

Netizen 2: "About the MDU function of STC8C, I would like to share a little bit of my own experience. If there is anything wrong, please criticize and give advice to improve it together."

1. Functions 1 and 2 are efficient for reducing and expanding integer data. First of all, when performing a double-operand operation, if the lengths of the two numbers are different, they need to be converted to the same length before the operation is performed. For example, when multiplying a 32-bit integer by an 8-bit integer, it is necessary to convert the 8-bit to 32-bit. Secondly, the result of AD sampling also needs displacement when converting to the specified digit precision. Finally, for example, for network communication, it is necessary to extract certain bits of data for command parsing or data decomposition and synthesis, and displacement is very important. Since the 8051 only has an instruction to move 1 bit, the multi-bit movement requires additional loop codes and requires many instruction cycles, so using the MDU will be several times faster than the 51 assembly instruction.
2. Function 3 is necessary for converting integers to floating-point numbers. For full-precision 32-bit integers, it generally takes more than 100 instruction cycles to implement this function, so the improvement of the MDU counter-rotation speed is relatively large. Since the output of AD devices and various three-axis acceleration outputs are generally integers (such as 16-bit), to perform real number operations and trigonometric function operations, the output of integers must be converted to floating point numbers, and each time This data type conversion is required to collect data, and the number of conversions is required. For high-speed data acquisition and applications like drone control, the overall performance improvement with a DMU can be substantial.
3. Function 6 is the necessary division function for fixed-point real number operations, and function 4 is the multiplication operation corresponding to 16-bit x 16-bit result of function 6 as a 32-bit result. The most common application of function 6 is scale conversion in data processing. For example, for the integer conversion of a 10-bit AD collected with a reference voltage of 5 volts, the 2-bit fixed decimal point of a 3-digit digital tube is displayed. The formula is: N32=ADN *500/1023. At this time, as long as (1) AND the AD sample value to MX (DM1MD0), (2) to send 500 to NX (MD5MD4), (3) to execute function 4, the result is 32 bits, (4) to send 1023 to NX (MD5MD4), (3) Execute function 6, the 16-bit result is in MX, just get it back. Another common application is to draw dots and lines on a dot matrix screen such as TFT, such as a digital oscilloscope. These require multiplication and division of coordinate transformation - first multiply to a 32-bit integer, and then divide by a 16-bit integer to get 16 bits result.
4. The combination of function 4 and function 6 is the hardware basis for implementing discrete convolution. If the floating-point acceleration hardware is not used, the four arithmetic operations of floating-point numbers are an order of magnitude slower than that of integers. Therefore, the predecessors invented the method of using integer variables to implement convolution. First of all, for example, when we commonly convert JPG image data to RGB image data or vice versa, Fourier transform is required. Since the length of image data is fixed (8 bits or 16 bits), discrete Fourier transform can be used. To achieve, which basically only uses 8-bit or

16-bit integer multiplication and a very small amount of 32-bit multiplication and division. In this way, our early digital cameras were possible. Secondly, the various template processing common in PS image processing also uses the two-dimensional matrix convolution method, which also requires a huge amount of integers (8-bit image image size requires 16-bit and 32-bit intermediate calculation results) Multiplication Adding calculations, using discrete convolution will greatly improve the operation speed. Therefore, the STC8 single-chip microcomputer with MDU can not only be used to collect and display images in real time, but also process images in real time. Finally, artificial intelligence also involves a large number of vector and matrix operations, such as neural network convolution, which can be implemented with the combination of function 4 and function 6. MDU should be able to be applied in small intelligent scenarios. Just to realize these functions, it needs the cooperation of the enhanced double data pointer of STC8, and needs a special knowledge structure, and a function library is specially compiled to provide users with the huge advantage of STC8 MDU.

25.3 Example Routines

C Language code

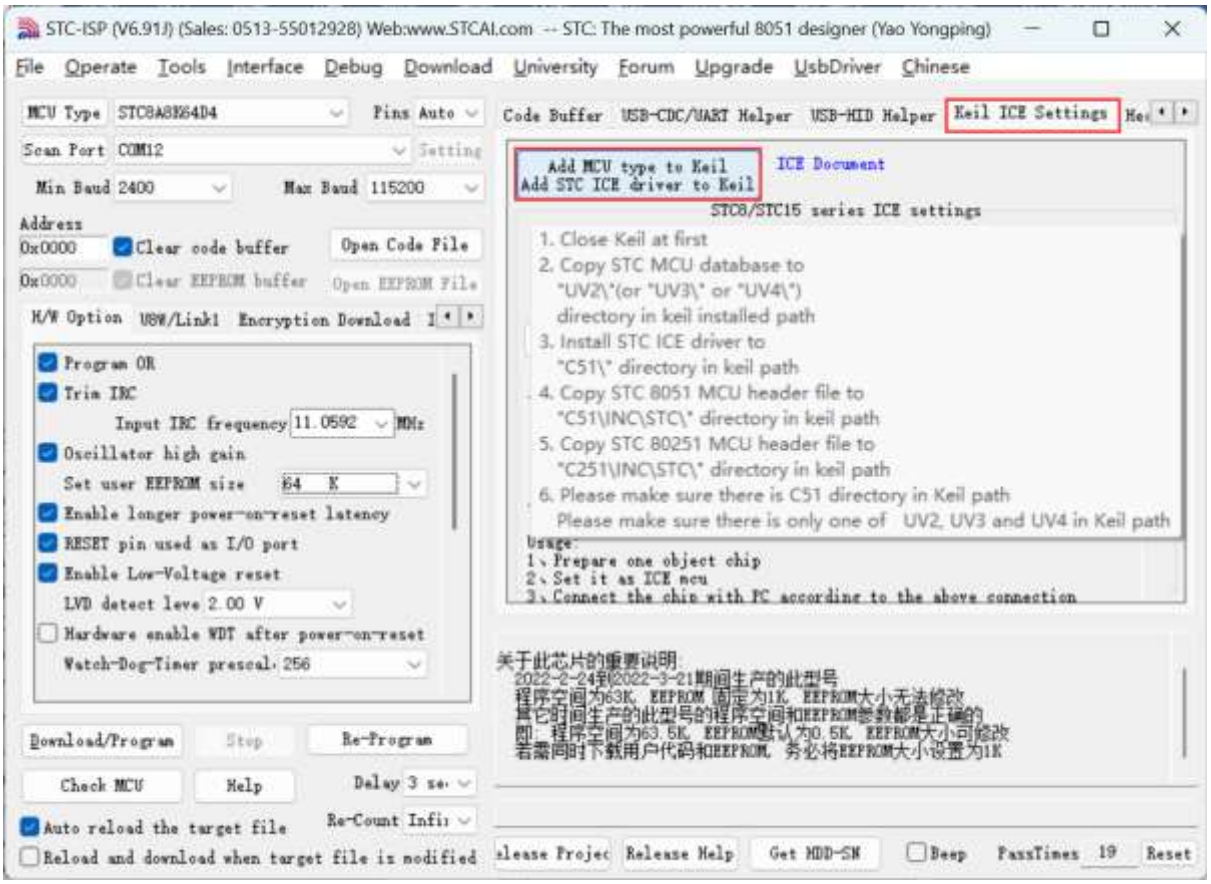
```
;Operating frequency for test is 11.0592MHz

#include  "reg51.h"
#include  "intrins.h"

#define     MD3U32              (*(unsigned long volatile xdata *)0xfcf0)
#define     MD3U16              (*(unsigned int volatile xdata *)0xfcf0)
#define     MD1U16              (*(unsigned int volatile xdata *)0xfcf2)
#define     MD5U16              (*(unsigned int volatile xdata *)0xfcf4)

#define     MD3                 (*(unsigned char volatile xdata *)0xfcf0)
#define     MD2                 (*(unsigned char volatile xdata *)0xfcf1)
#define     MD1                 (*(unsigned char volatile xdata *)0xfcf2)
#define     MD0                 (*(unsigned char volatile xdata *)0xfcf3)
#define     MD5                 (*(unsigned char volatile xdata *)0xfcf4)
#define     MD4                 (*(unsigned char volatile xdata *)0xfcf5)
#define     ARCON               (*(unsigned char volatile xdata *)0xfcf6)
#define     OPCON               (*(unsigned char volatile xdata *)0xfcf7)

sfr         P_SW2           =   0xBA;
//////////////////////////////////////////////////////////////////////////////
//16 bits by 16 bits
//////////////////////////////////////////////////////////////////////////////
unsigned  long  res;
unsigned int dat1, dat2;
P_SW2 |= 0x80;                          // Access the extension register xsfr
MD1U16 = dat1;                          //dat1 User given
MD5U16 = dat2;                          //dat2 User given
ARCON = 4 << 5;                         //16 bits*16 bits, multiplication mode
OPCON = 1;                              // Start calculation
while((OPCON & 1) != 0);                // Wait for the calculation to complete
res = MD3U32;                           //32-bit result
//////////////////////////////////////////////////////////////////////////////
//32 bits divided by 16 bits
//////////////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned int dat2;
P_SW2 |= 0x80;                          // Access the extension register xsfr
MD3U32 = dat1;                          //dat1 User given
MD5U16 = data2;                         //dat2 User given
ARCON = 6 << 5;                         //32-bit/16-bit, division mode
OPCON = 1;                              // Start calculation
while((OPCON & 1) != 0);                // Wait for the calculation to complete
res = MD3U32;                           //32-bit quotient, 16-bit remainder in MD5U16
unsigned long res;
unsigned long dat1;
unsigned char num;                      // The number of bits to shift, User given
MD3U32 = dat1;                          //dat1 User given
ARCON = (2 << 5) + num;                 //32-bit left shift mode
//ARCON = (1 << 5) + num;               //32-bit right shift mode
OPCON = 1;                              // Start calculation
while((OPCON & 1) != 0);                // Wait for the calculation to complete
res = MD3U32;                           //32-bit result
```

Appendix A STC Emulator User Guide

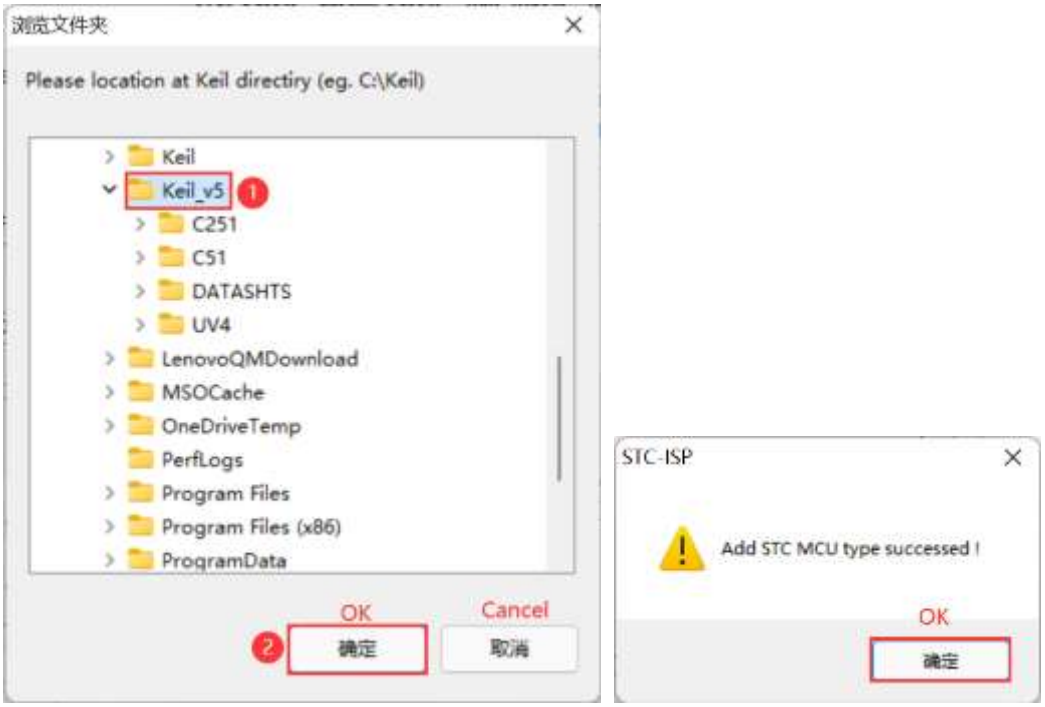
A: What kind of compiler/assembler should be used for STC MCU?
Q: Any old 8051 compiler/assembler can support it. Keil C51 is popular now

A: How to include header files in Keil environment
Q: After installing the driver and header files according to the steps shown below, select the STC corresponding MCU model when creating a new project, and directly use "#include <stc8a8k64d4.h>" in the source file to complete the inclusion of the header file. If you select Intel's 8052/87C52/87C54/87C58 or Philips P87C52/P87C54/P87C58 compile, the header file contains <reg51.h>, but the new STC special function register needs to be declared by the user.

1.Install Keil version of emulation driver.

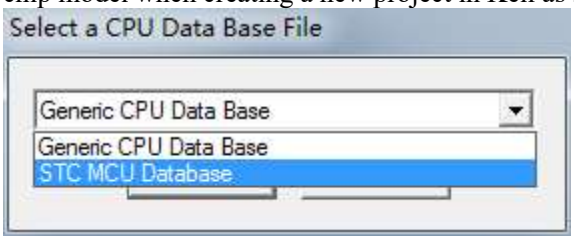


As shown above, select the "Keil Simulation Settings" page firstly, click "Add MCU model to Keil", and in the following directory selection window that appears, navigate to the installation directory of Keil (usually "C: \ Keil \") , After press "OK" button, the prompt message shown on the right in the following figure appears, indicating that the installation was successful. The STC Monitor51 emulation driver STCMON51.DLL will also be installed when adding the header file. The installation directory of the driver and header file is shown above.

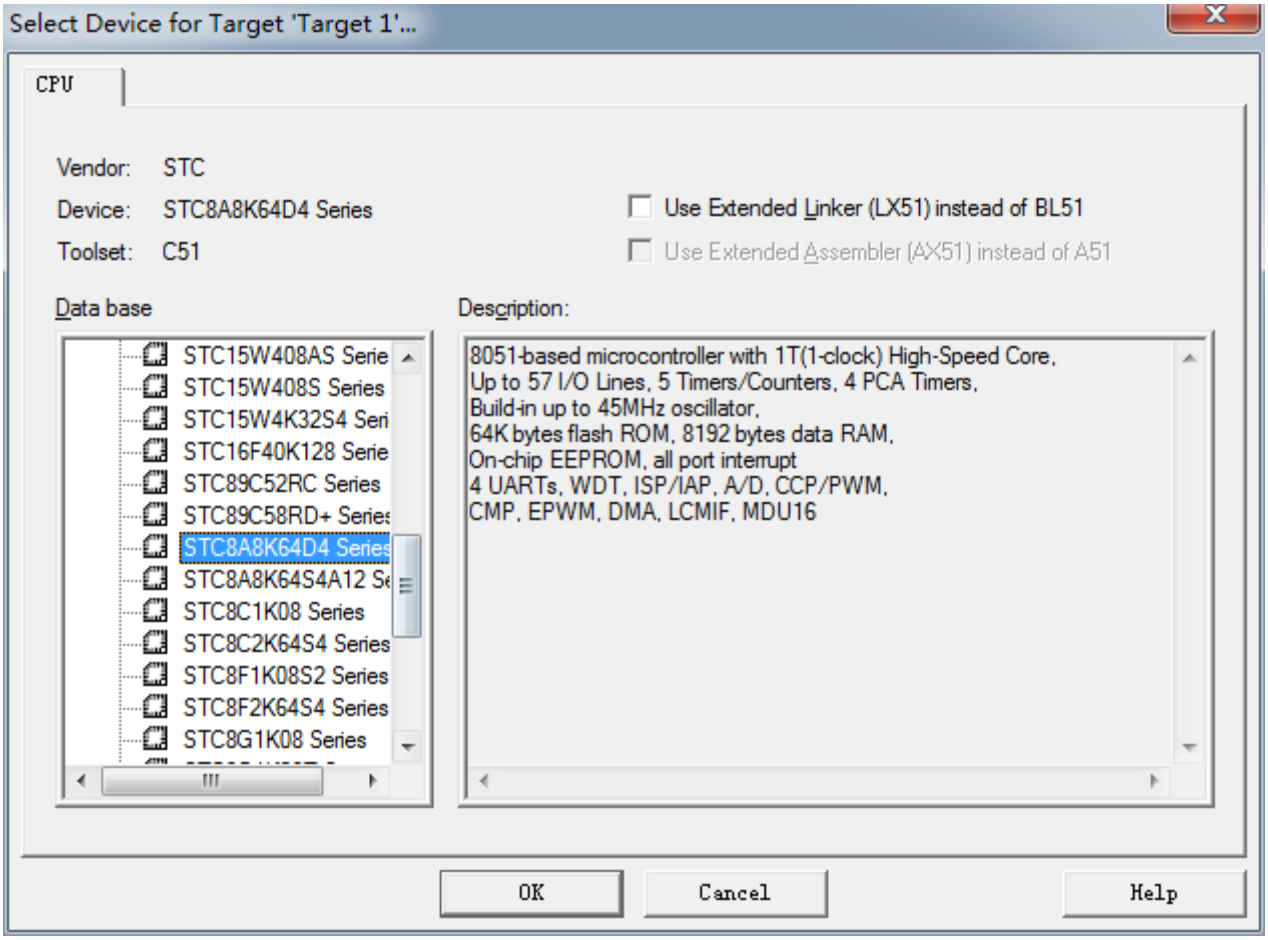


2. Create a project in Keil

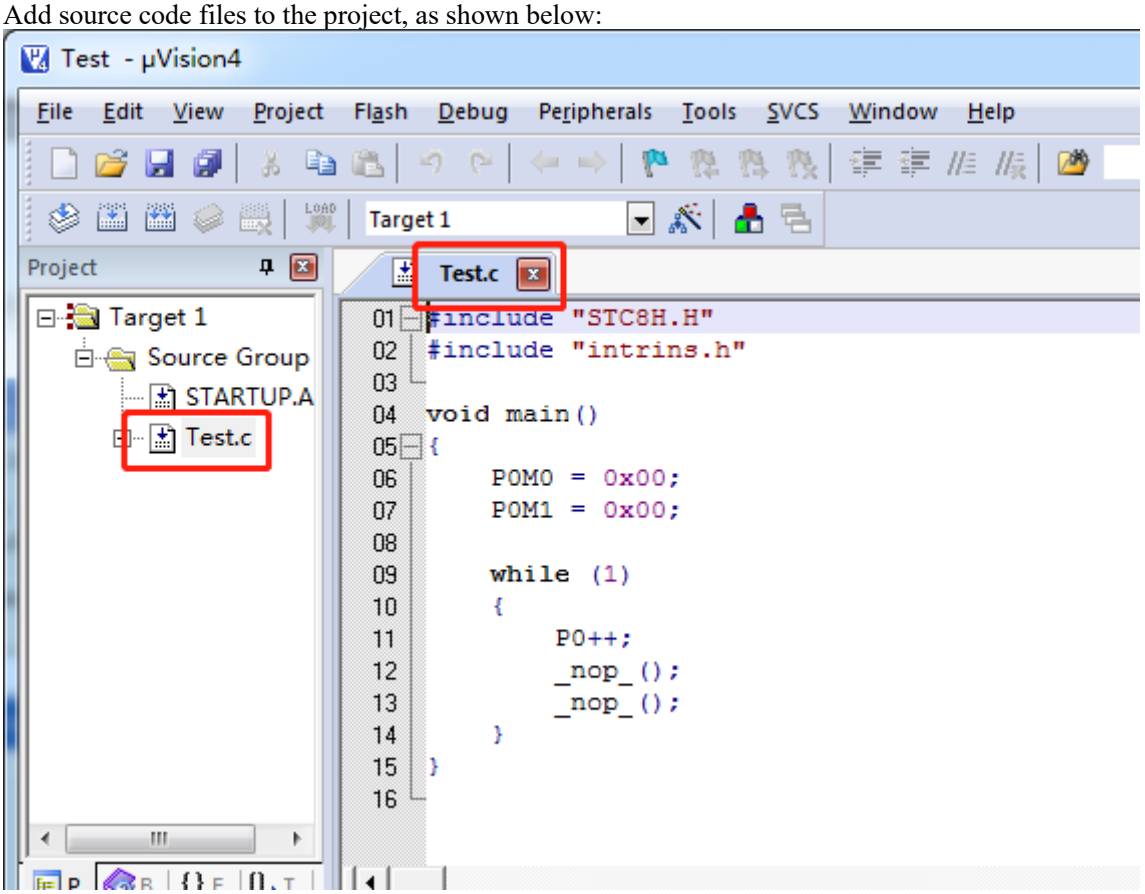
If the driver installation is successful in the first step, there will be an option of "STC MCU Database" in selecting the chip model when creating a new project in Keil as shown below.



Then select the responding MCU model from the list. Here we select the model of "STC8A8K64D4" and click "OK" to complete the selection.

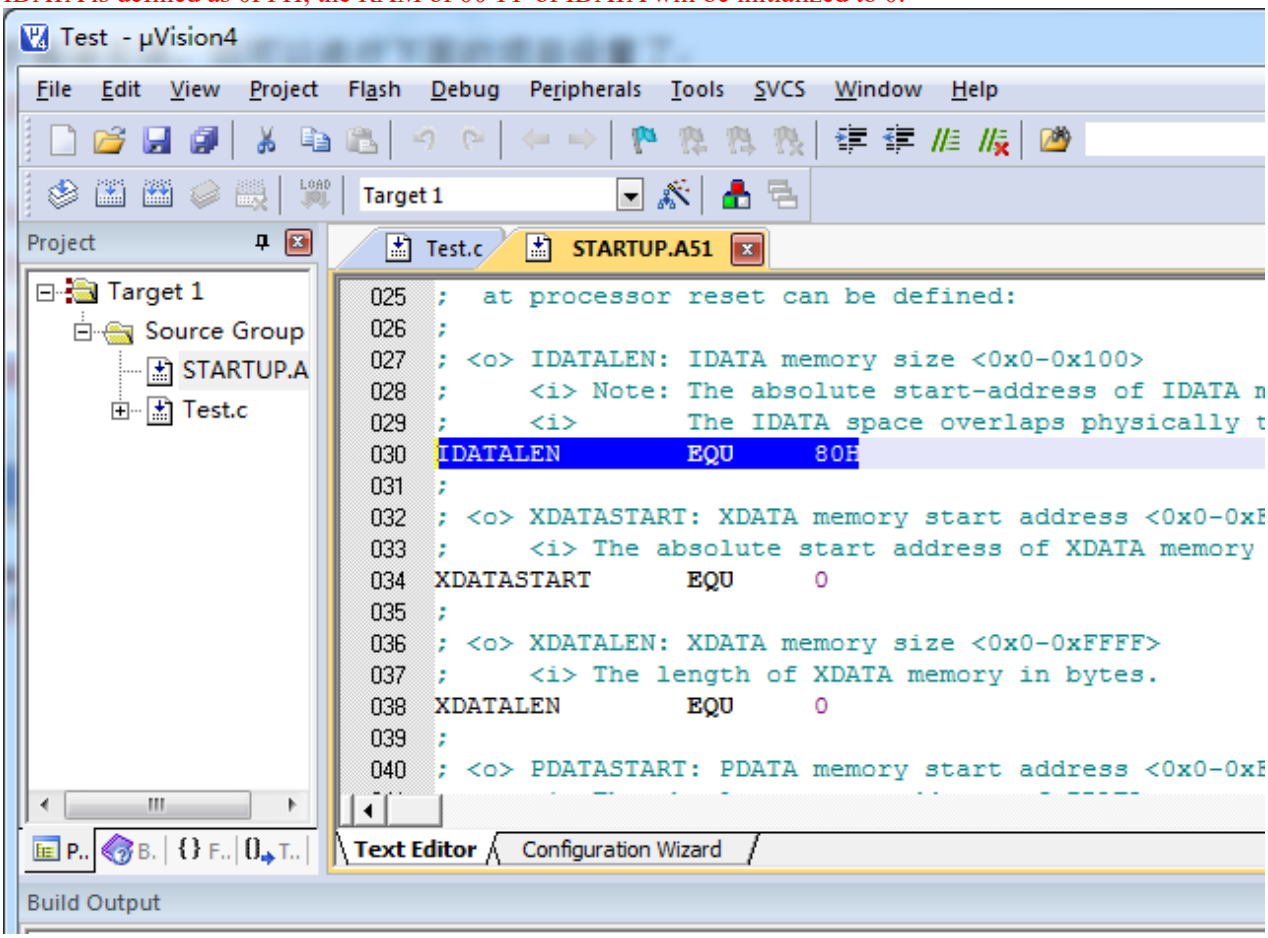


Add source code files to the project, as shown below:



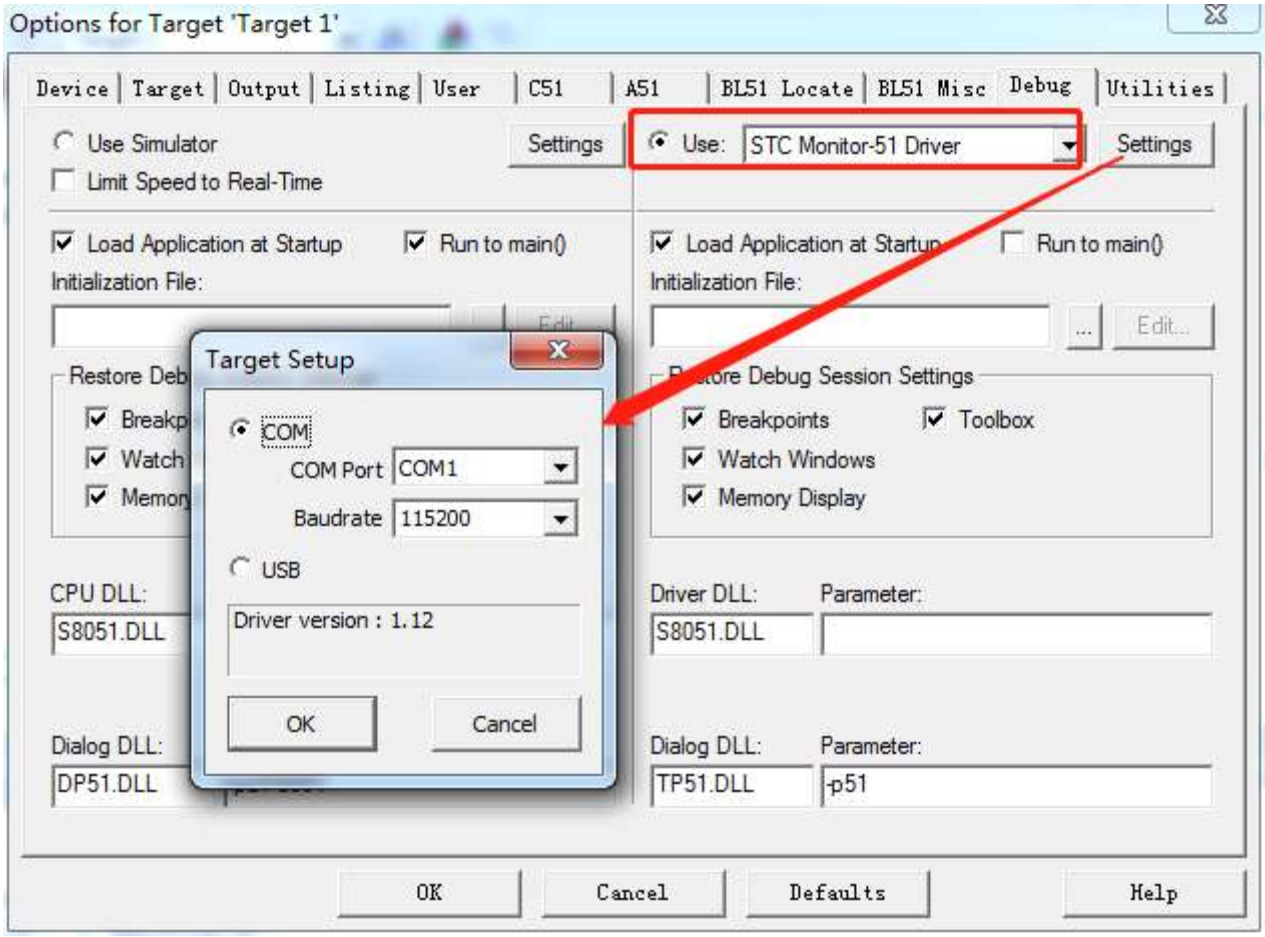
Save the project. If there is no error while compiling the project, you can set the following projects.

One additional note:
When a C language project is created and a startup file "STARTUP.A51" is added to the project, there is a macro definition named "IDATALEN", which is a macro used to define the size of the IDATA. The default value is 128, which is 80H in hexadecimal, and it is also the size of IDATA in the startup file that needs to be initialized to 0. Therefor if IDATA is defined as 80H, the code in STARTUP.A51 will initialize the RAM of 00-7F of IDATA to 0. Similarly, if IDATA is defined as 0FFH, the RAM of 00-FF of IDATA will be initialized to 0.



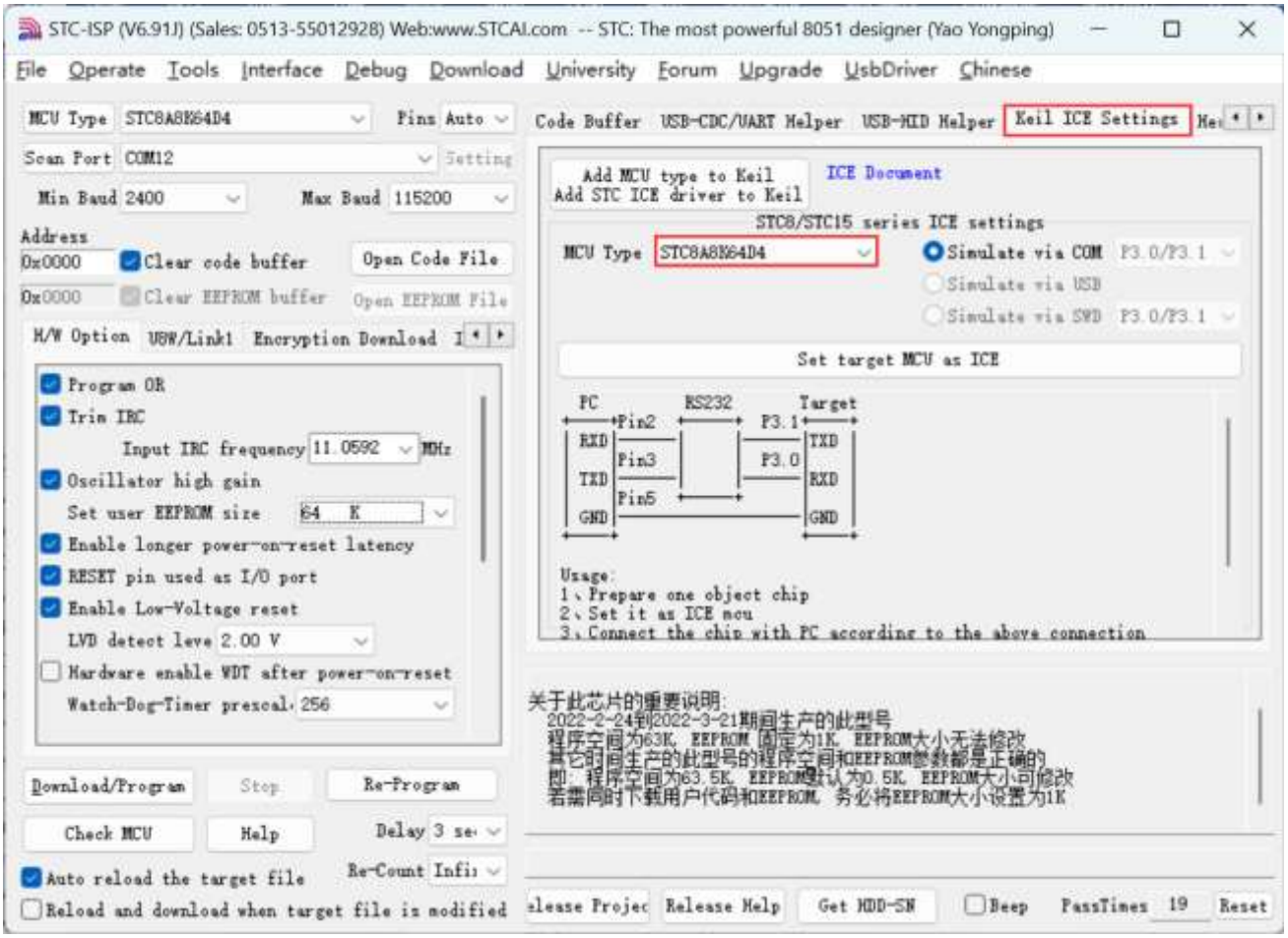
The IDATA size of the STC8 series of microcontrollers is 256 bytes (DATA of 00-7F and IDATA of 80H-FFH). Because the last 17 bytes of RAM have the ID number and related test parameters, if you need to use this part of the data in the program, you must not define IDATALEN as 256.

3. Project settings, select STC simulation driver.



As shown above, enter the project setting page firstly, select the "Debug" setting page, select the hardware emulation "Use…" on the right, and select "STC Monitor-51 Driver" in the emulation driver drop-down list. And then click the "Settings" button to enter the following setting screen. Set the port number and baud rate of the serial port. The baud rate is generally 115200. Then complete the setup.

4.Create simulation chip



Prepare a STC8A series or STC8F series chip, and connect it to the serial port of the computer through the download board. Then select the correct chip model as shown above, and enter the "Keil simulation settings" page, click the button of the corresponding model. After the program downloading completes, the simulator is ready for use.

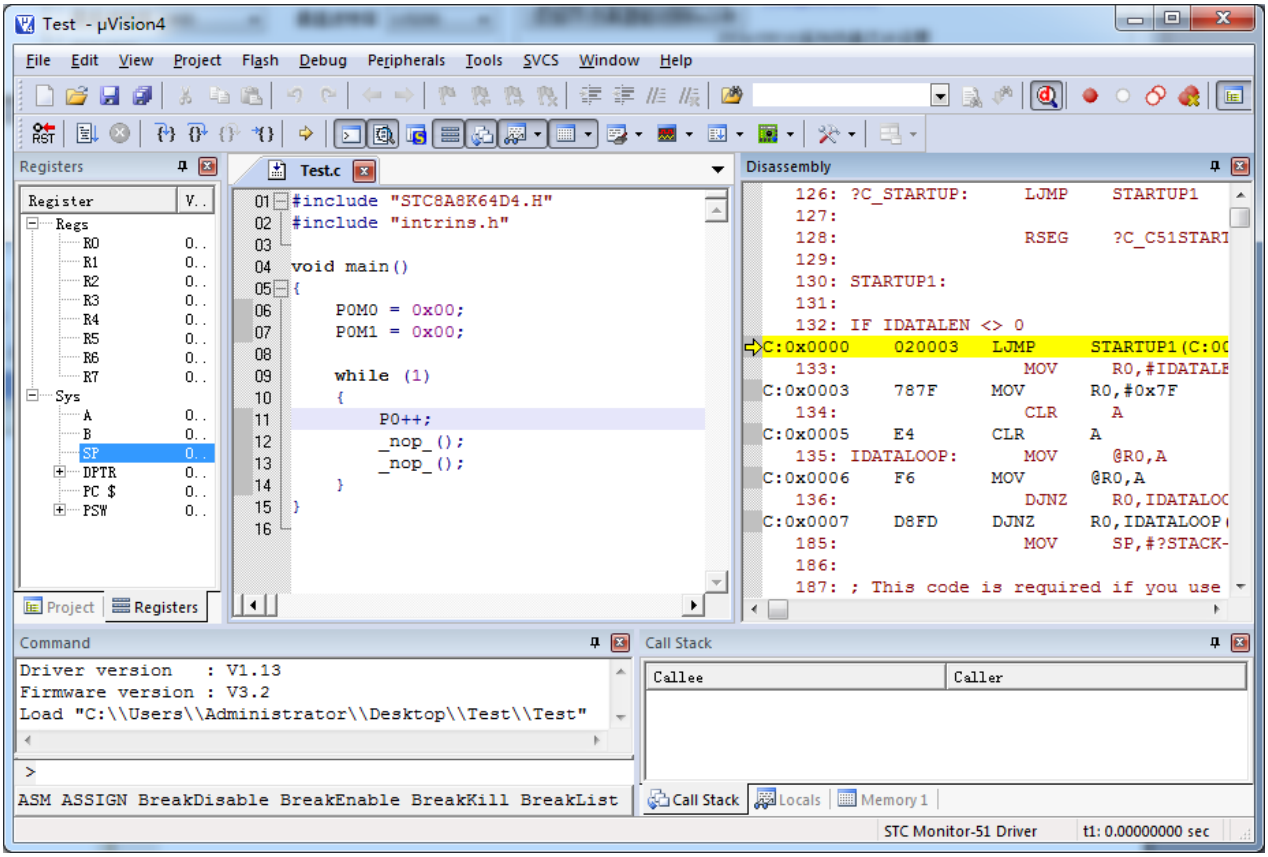
5.Start simulation

Connect the completed simulation chip to the computer through the serial port.

After compiling the project we created before without errors, press "Ctrl + F5" to start debugging.

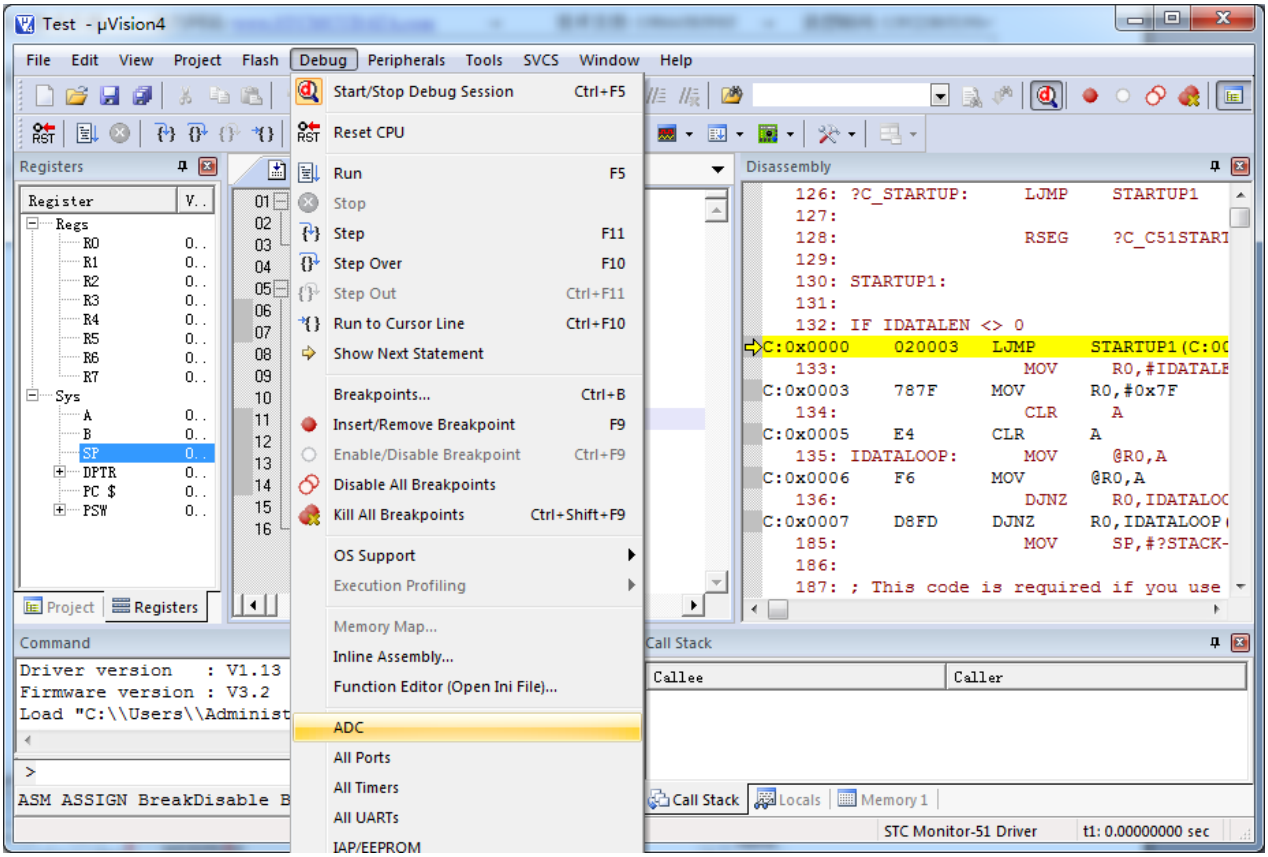
If the hardware connection is correct, you will enter a debugging interface similar to the following, and display the current simulation driver version number and the current simulation monitoring code firmware version number in the command output window.

The current maximum number of breakpoints is 20 (in theory, any number of breakpoints can be set, but too many breakpoints will affect the speed of debugging).

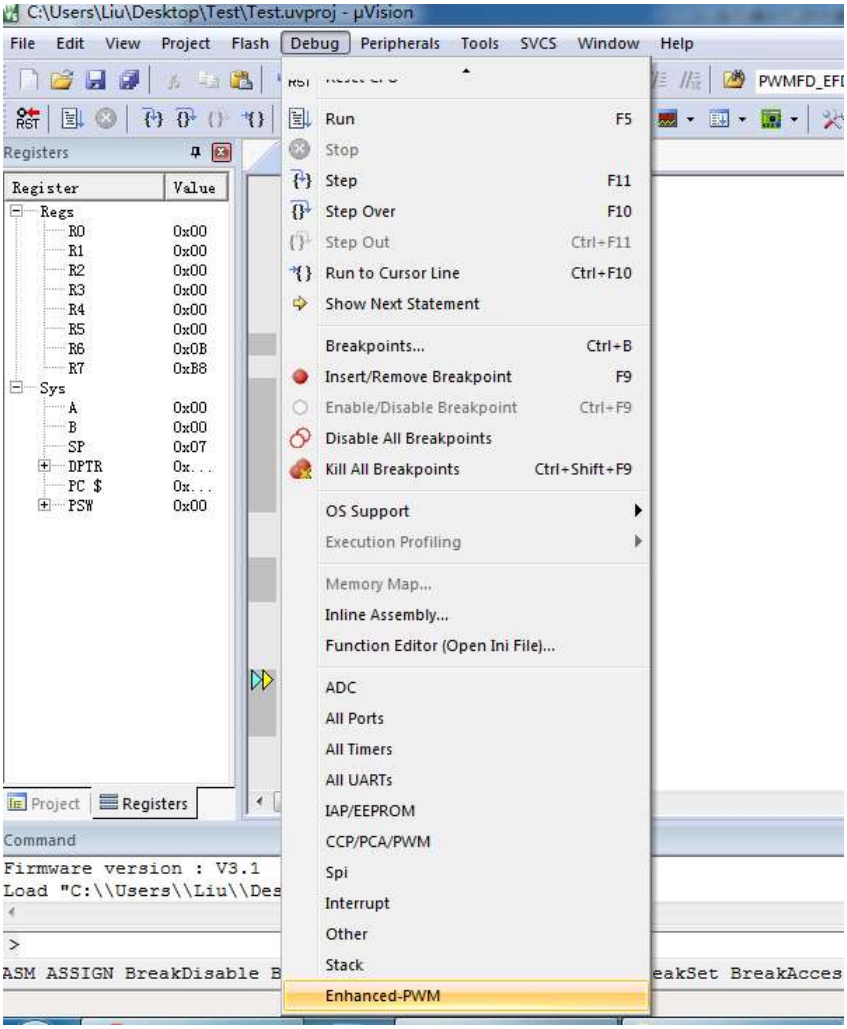


6. During the simulation process, the viewing of the register

During the simulation process, you can view the MCU-related registers. All registers are listed at the bottom of the "Debug" menu. As shown below:



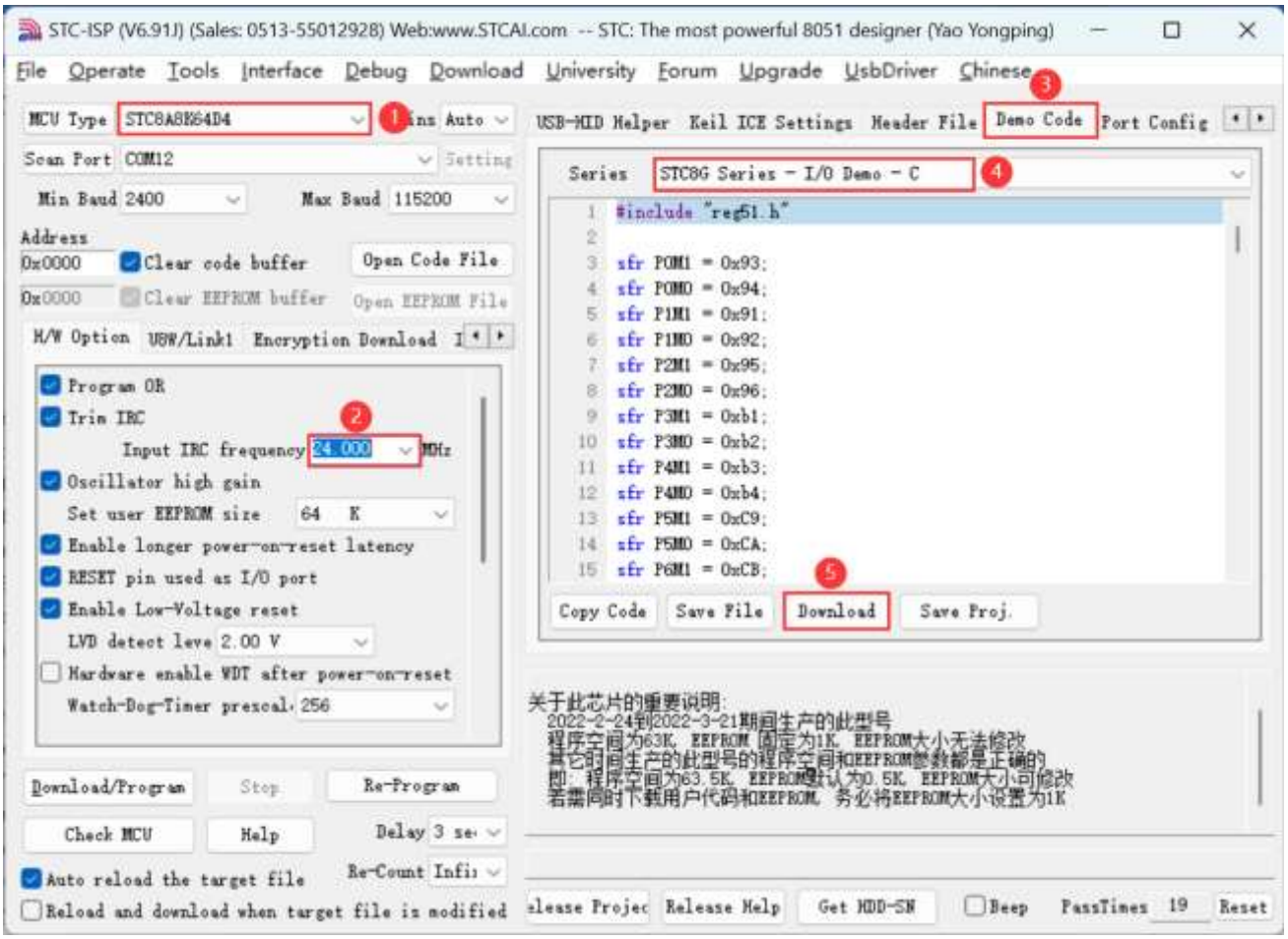
At the bottom of the "Debug" menu in the picture above, there is also a small black triangle, which indicates that there are hidden items (mainly due to the display layout size)
Drag the mouse on the small triangle to automatically drag out all the items, as shown below:



Simulation considerations:

1. The simulation monitoring program occupies the two ports P3.0/P3.1, but does not occupy the serial port 1. The user can switch the serial port 1 to P3.6/P3.7 or P1.6/P1.7 to be used.
2. The simulation monitoring program occupies the last 768 bytes of the internal extended RAM (XDATA), and the user cannot write to XDATA in this area .

Appendix B How to Test I/O Ports



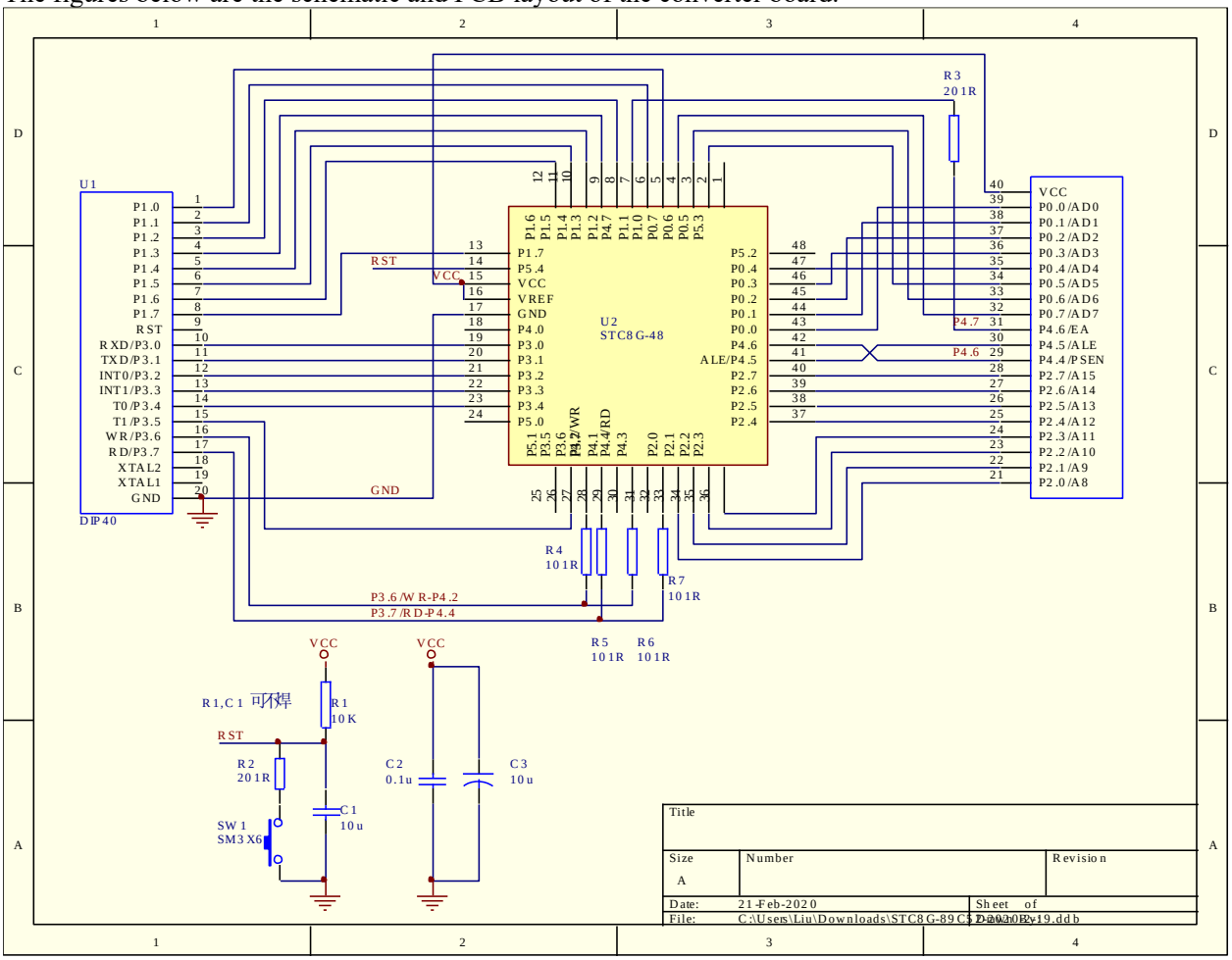
Test I/O port steps:
1. Select the microcontroller model
2. Set the operating frequency of the test program (24MHz)
3. Open the "Example Program" page
4. Select the "I/O port test" program of STC8G or STC8H series
5. Click "Download Hex Directly" on the page

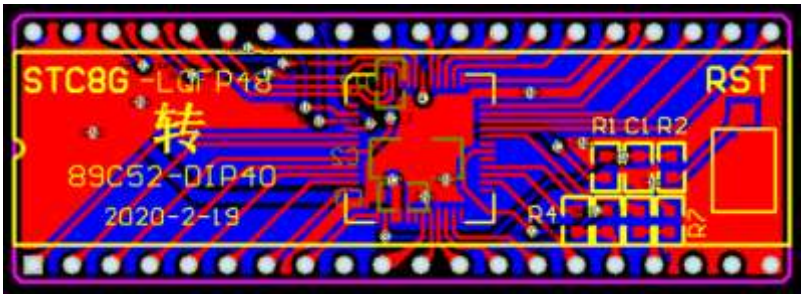
After the download is completed, the running water light program will be executed on all I/O ports. At this time, you can connect LEDs to the I/O ports or use an oscilloscope to see the waveform.

Appendix C How to Make the Traditional 8051 MCU EVB Emulatable

The traditional 8051 microcontroller EVB does not have simulation function. To enable the traditional 8051 microcontroller EVB to be simulated, a conversion board is needed. The physical picture of the conversion board is shown below. The converted pin arrangement is basically the same as that of the traditional 8051. Therefore, the simulation function of the standard 8051 learning board can be realized.

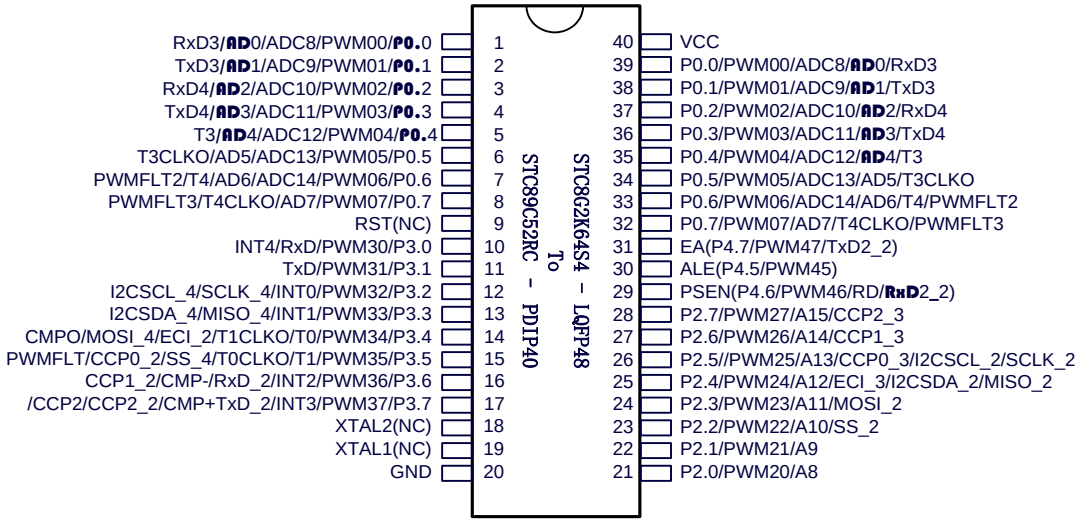
The figures below are the schematic and PCB layout of the converter board.





This conversion board can be used for STC8H series LQFP48 to STC89C52RC / STC89C58RD + series simulation.

The following figure is a functional diagram of the conversion board.



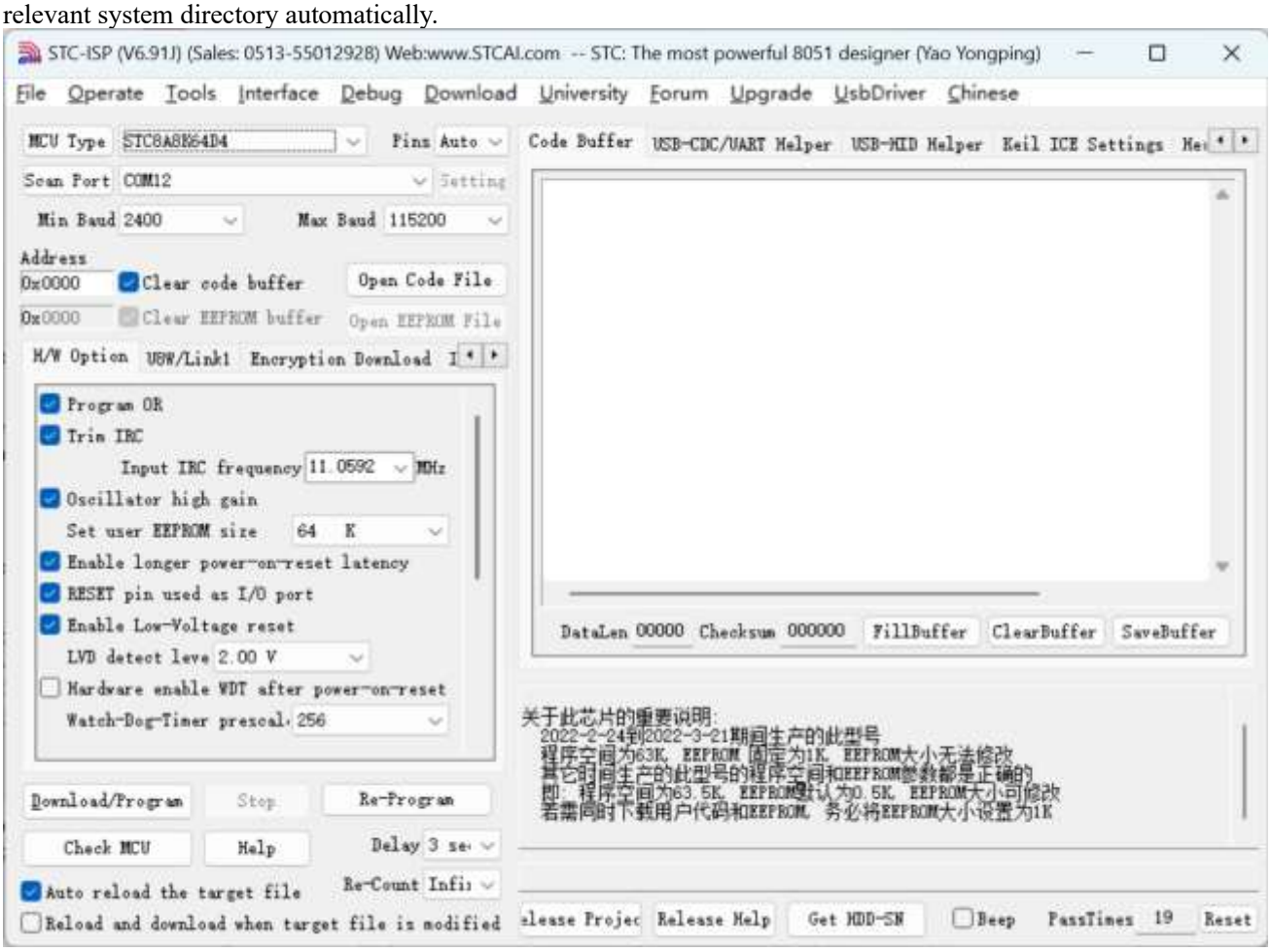
Note:

- ✓ Due to the built-in high-precision R/C clock, no external crystal is needed, XTAL1 and XTAL2 can be empty.
- ✓ WR and RD are P4.2/ WR and P4.4/ RD respectively, not traditional WR/P3.6 and RD/P3.7. **(In the conversion board, P4.2 and P3.6 are connected together, and P4.4 and P3.7 are connected together. When this conversion board is used to access the external bus, P3.6 and P3.7 should be set to high-impedance input mode, so that P4.2 and P4.4 can normally output the bus read and write signals. If the external bus is not needed to be accessed, P4.2 and P4.4 should be set to high-impedance input mode, and P3.6 and P3.7 are ordinary I/O.)**
- ✓ The STC8A8K64D4 series MCUs are low-level reset, it is not compatible with the high-level reset of the traditional 8051, so the RST pin is left floating, and replaced by the reset button and reset circuit on the conversion board.

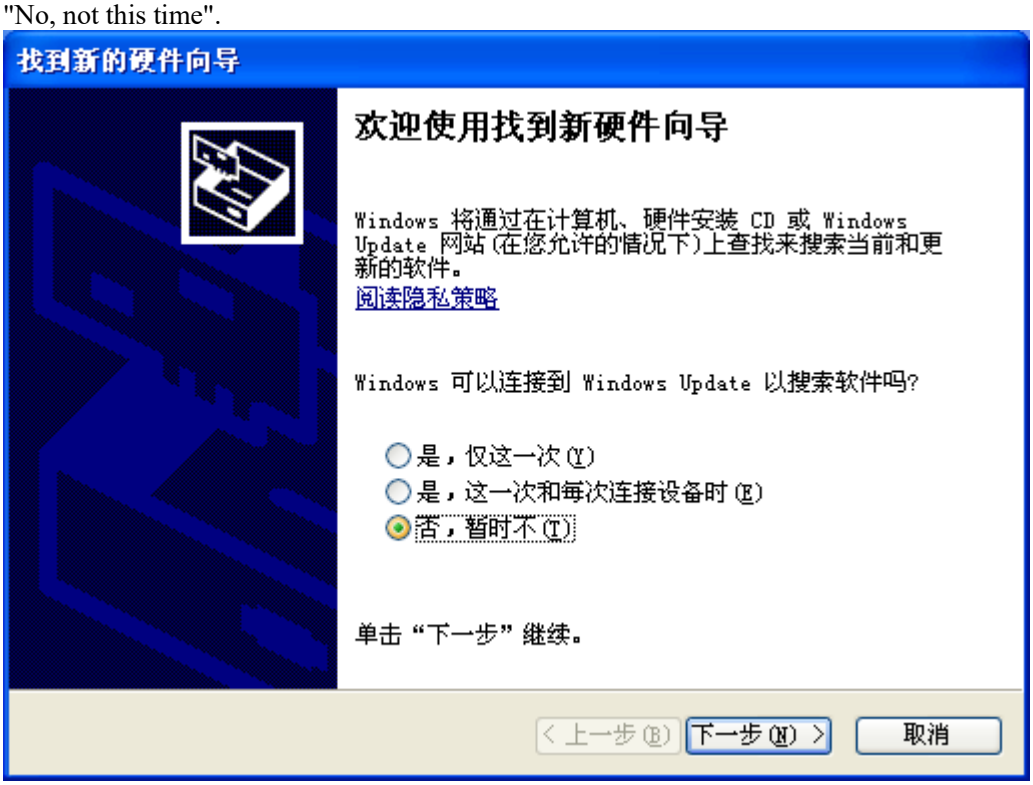
Appendix D STC-USB Driver Installation Instructions

Installation Instructions in Windows XP

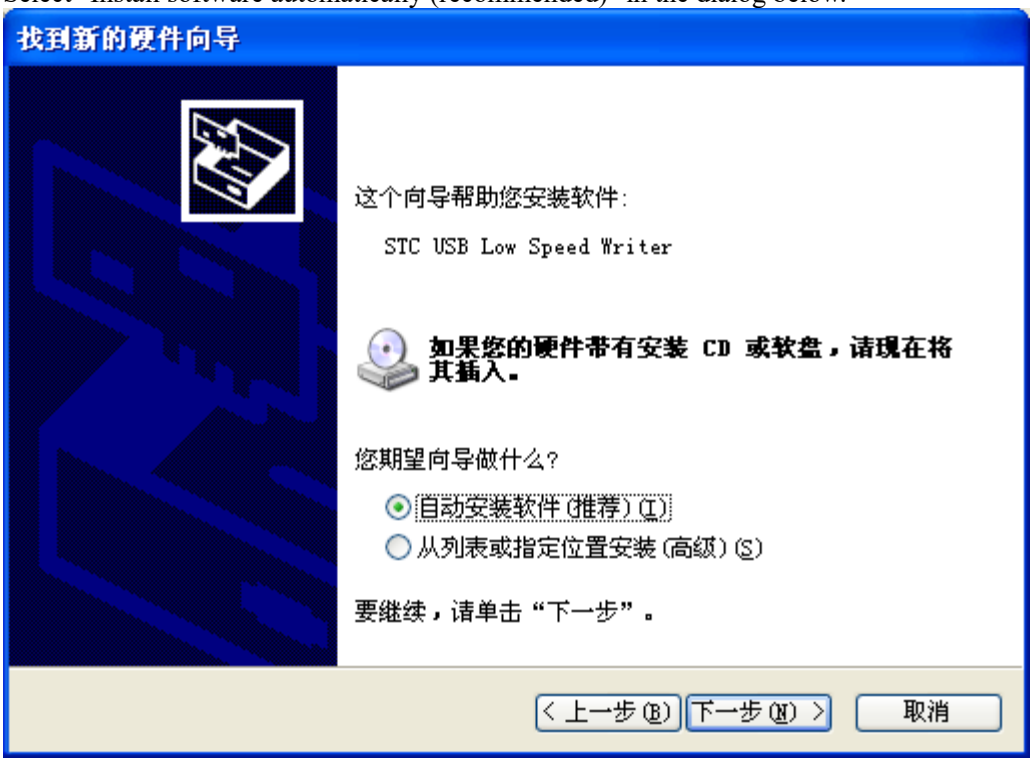
Open the STC-ISP download software of V6.79 (or later). The download software will copy the driver files to the relevant system directory automatically.



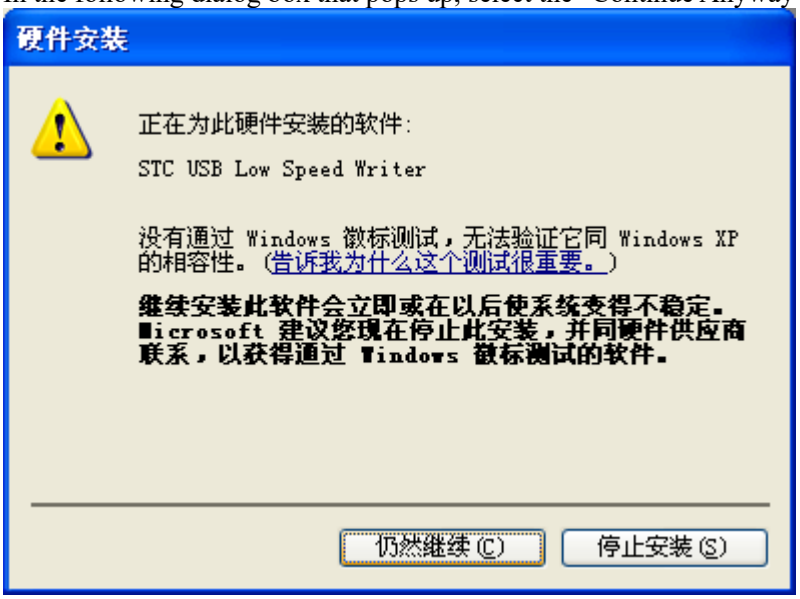
Plug in the USB device, the system will pop up the following dialog box automatically after finding the device, select "No, not this time".



Select "Install software automatically (recommended)" in the dialog below.



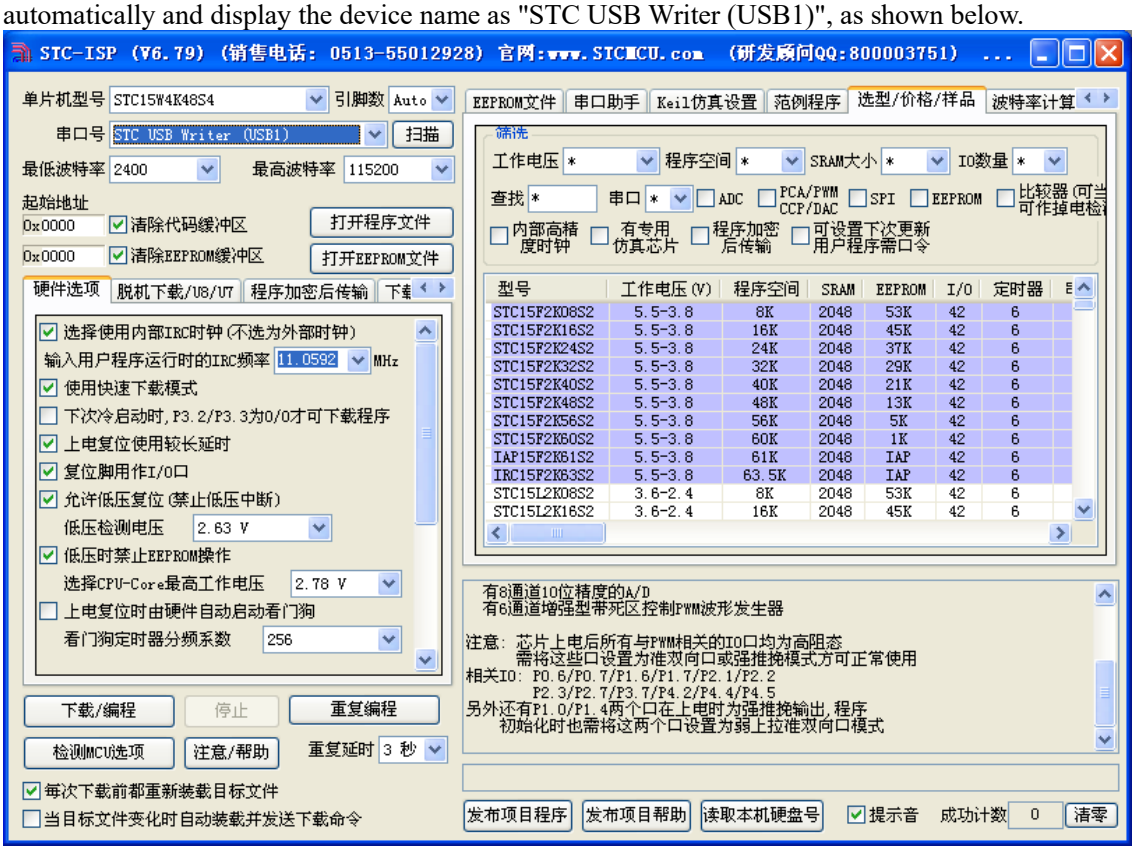
In the following dialog box that pops up, select the "Continue Anyway" button.



The system will automatically install the driver when connected, as shown below

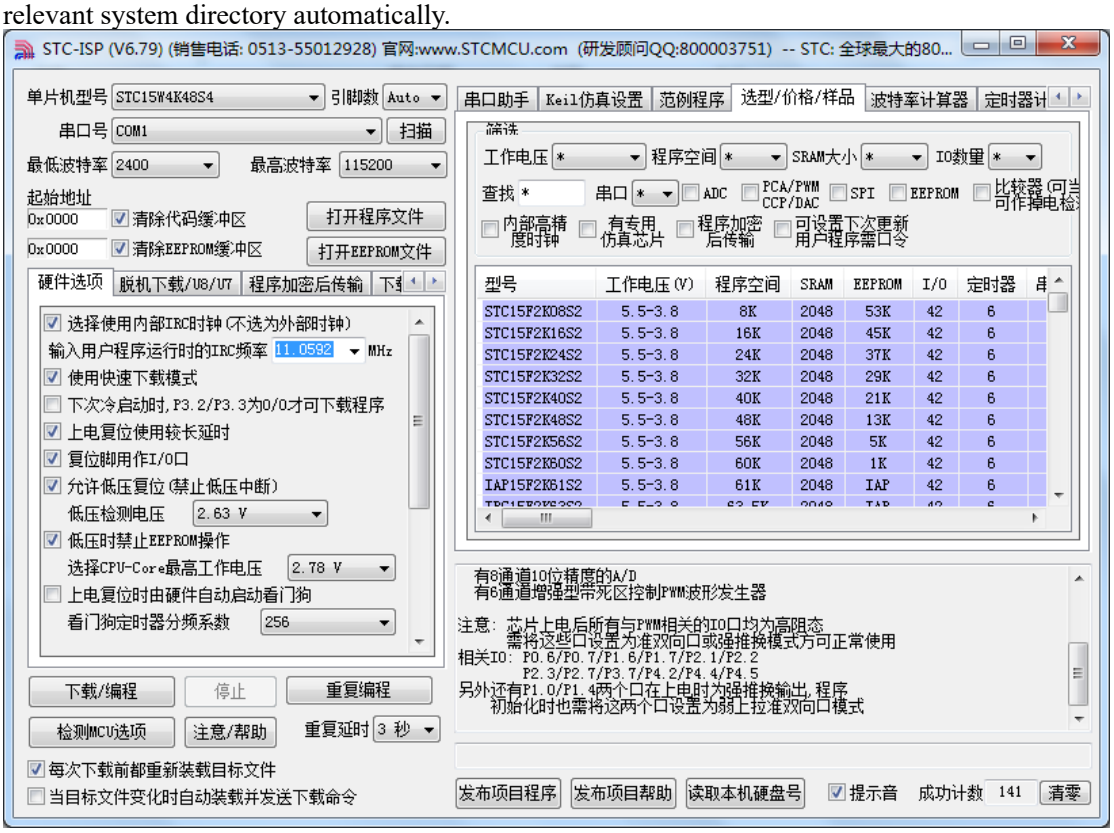
The following dialog box appears to indicate that the driver installation is complete.

Now, the serial number list in the previously opened STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (USB1)", as shown below.



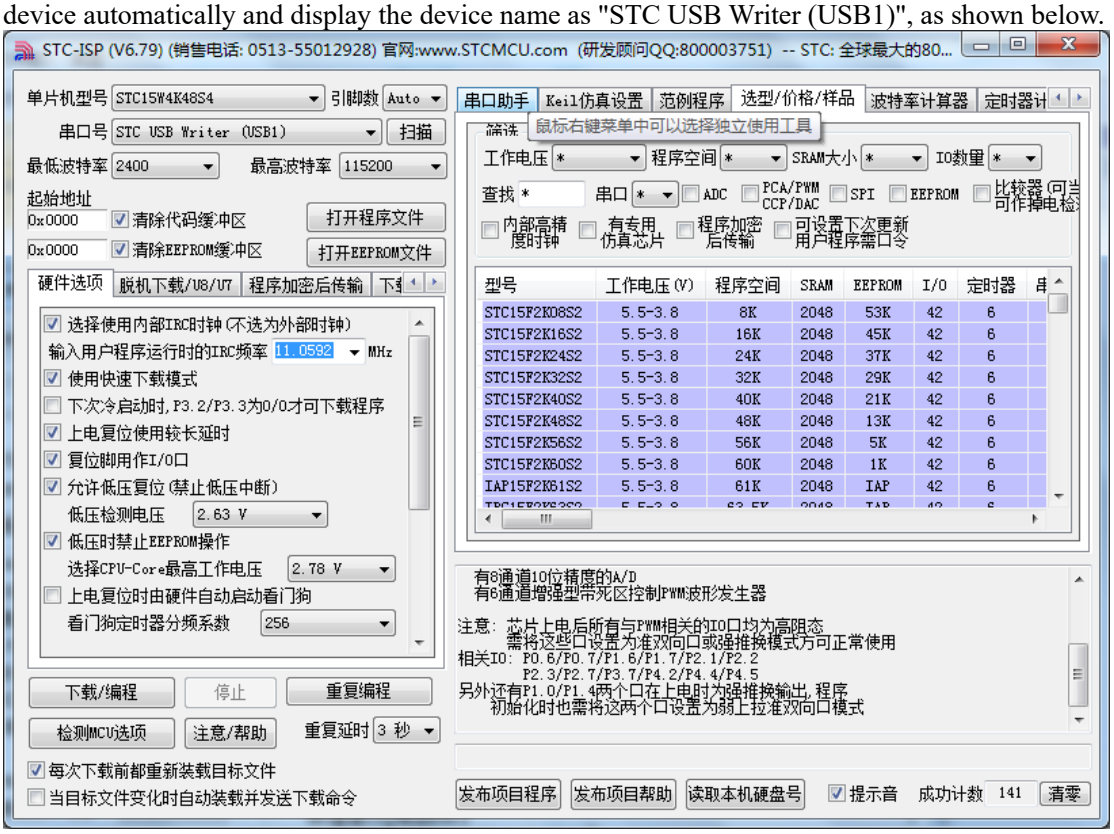
Installation Instructions in Windows 7 (32-bit)

Open the STC-ISP download software of V6.79 (or later). The download software will copy the driver files to the relevant system directory automatically.



Plug in the USB device, and the system will install the driver automatically when it finds the device. After the installation is complete, the following prompt box will appear.

Now, the serial port number list in the previously opened STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (USB1)", as shown below.

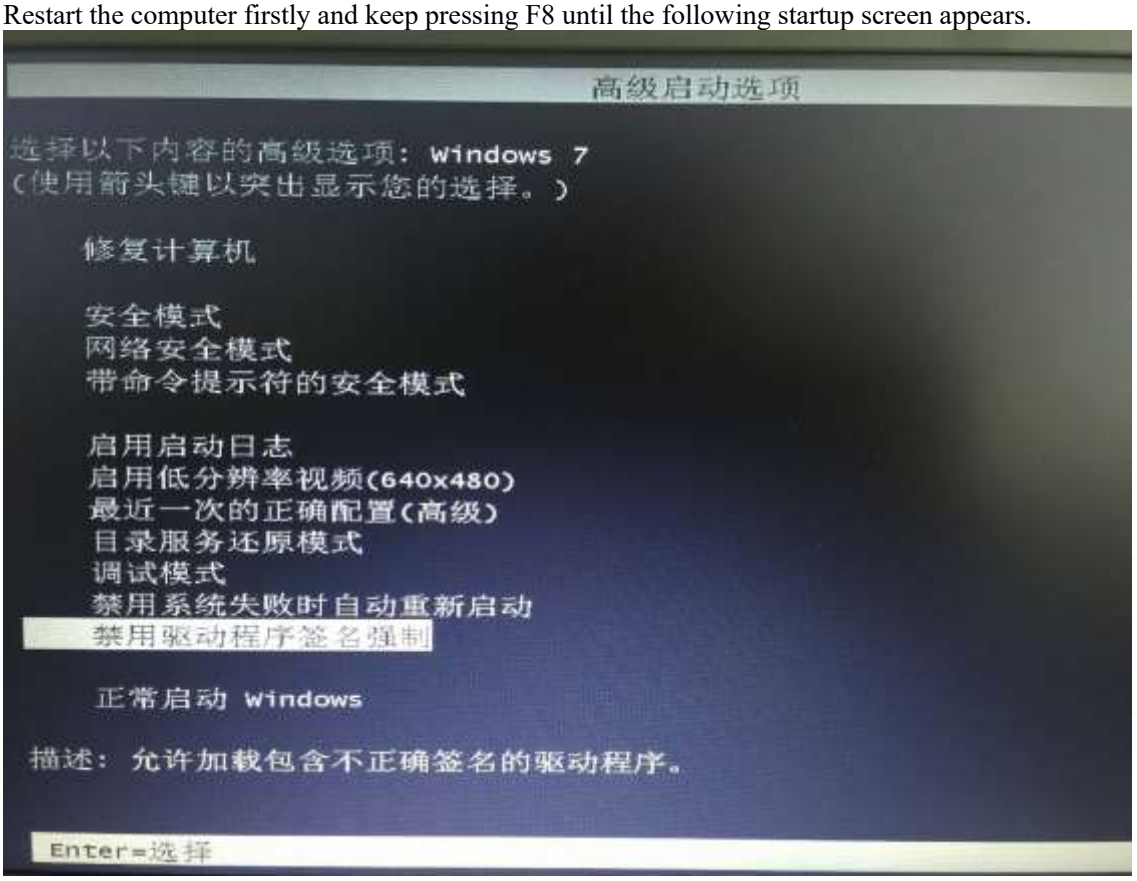


Note: If the system does not install the driver automatically in Windows 7, please refer to the installation method of Windows 8 (32-bit) for the driver installation method.

Installation Instructions in Windows 7 (64-bit)

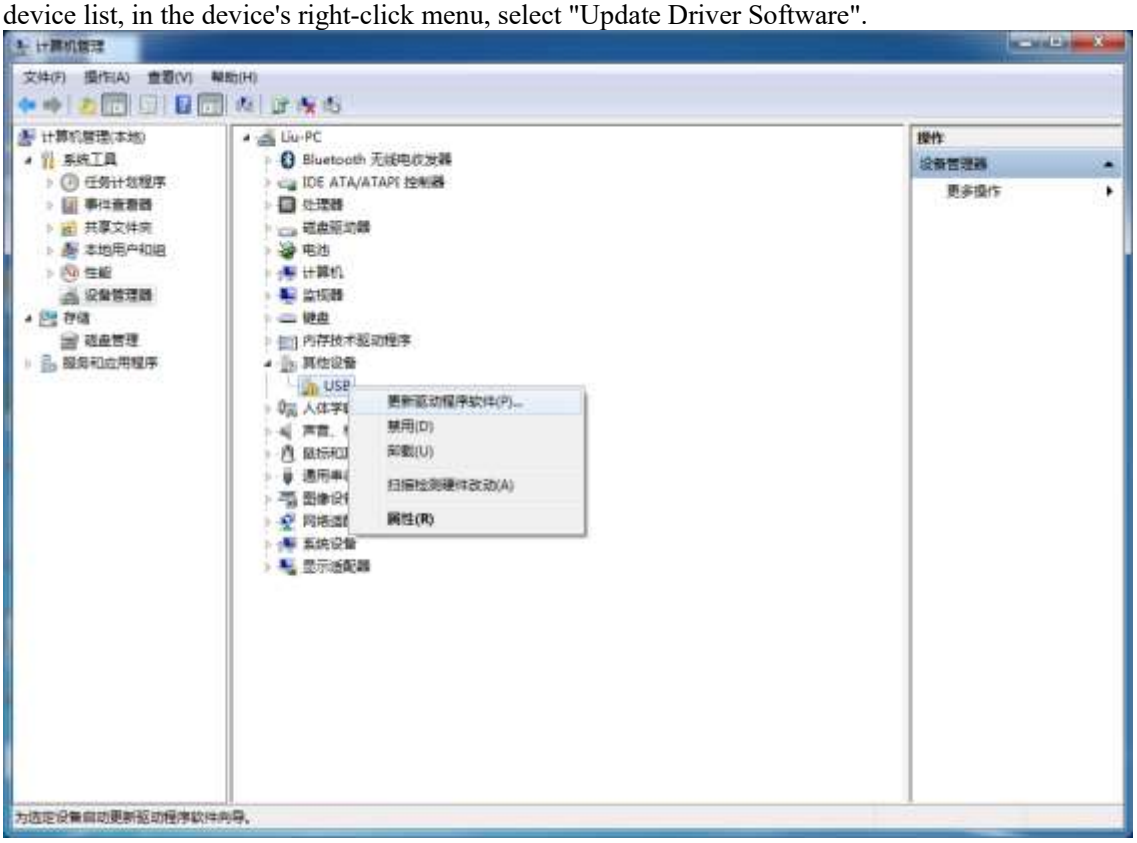
By default, the driver without digital signature cannot be successfully installed in Windows 7 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

Restart the computer firstly and keep pressing F8 until the following startup screen appears.



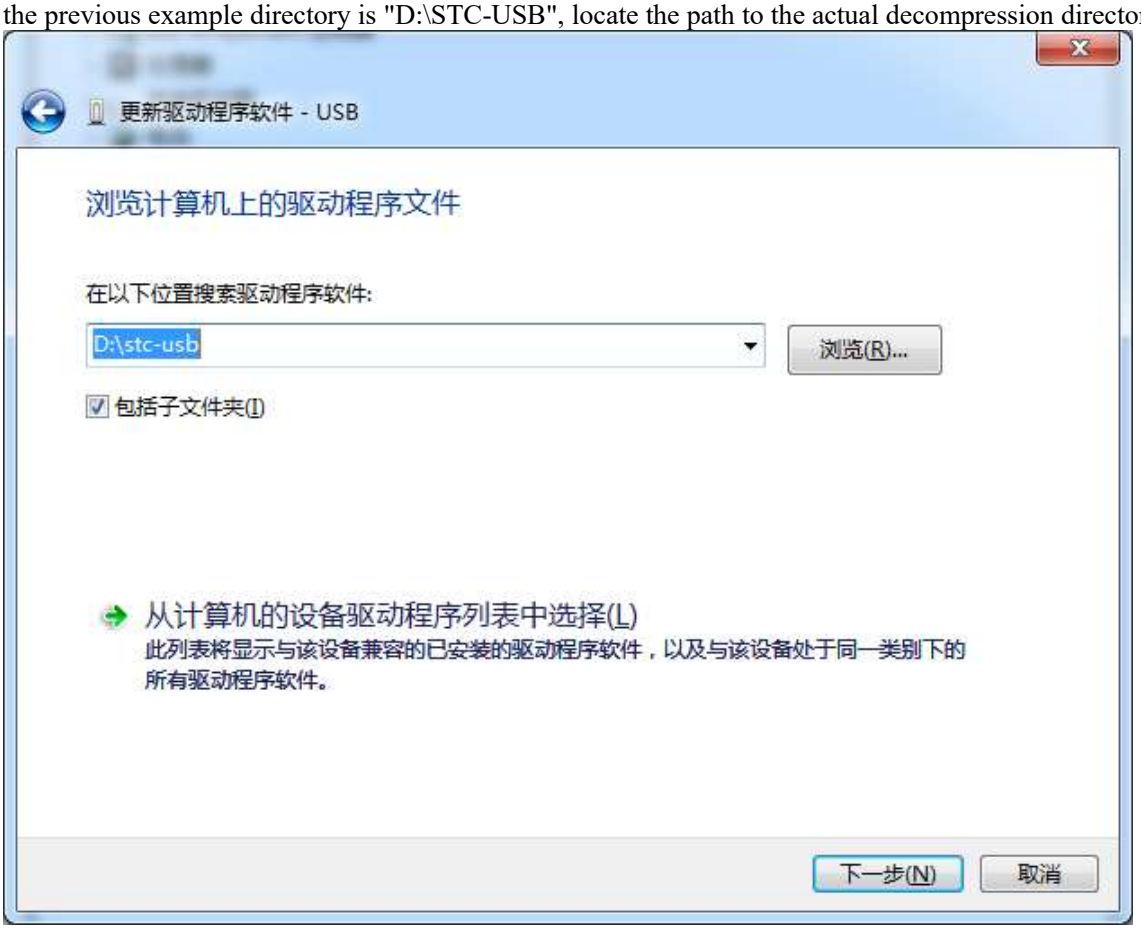
Select ‘Disable Driver Signature Enforcement’. The digital signature verification function is temporarily turn off after startup.

Plug in the USB device and open the Device Manager. Find the USB device with a yellow exclamation mark in the device list, in the device's right-click menu, select "Update Driver Software".

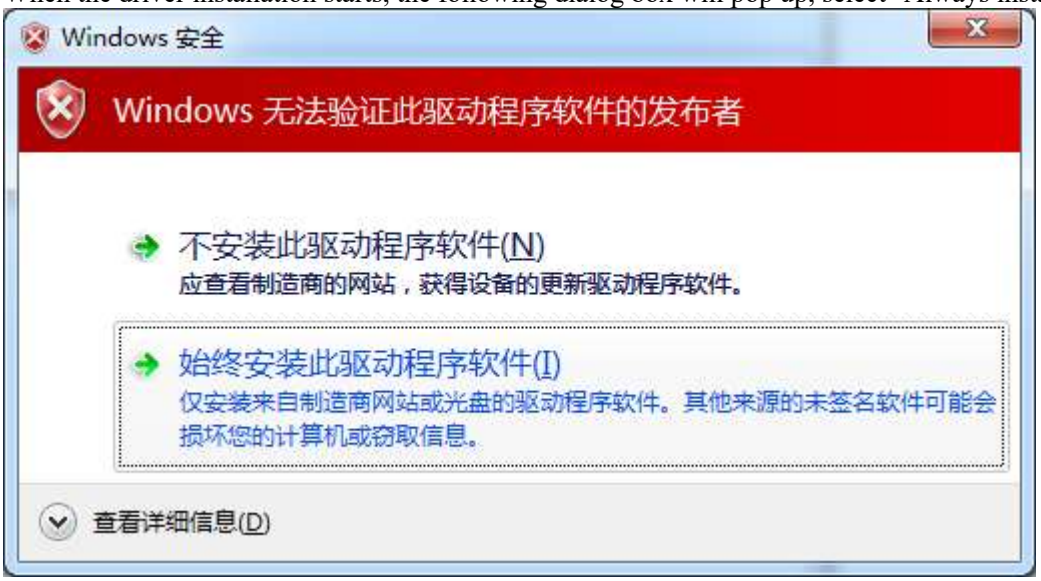


Select "Browse my computer for driver software" in the dialog below.

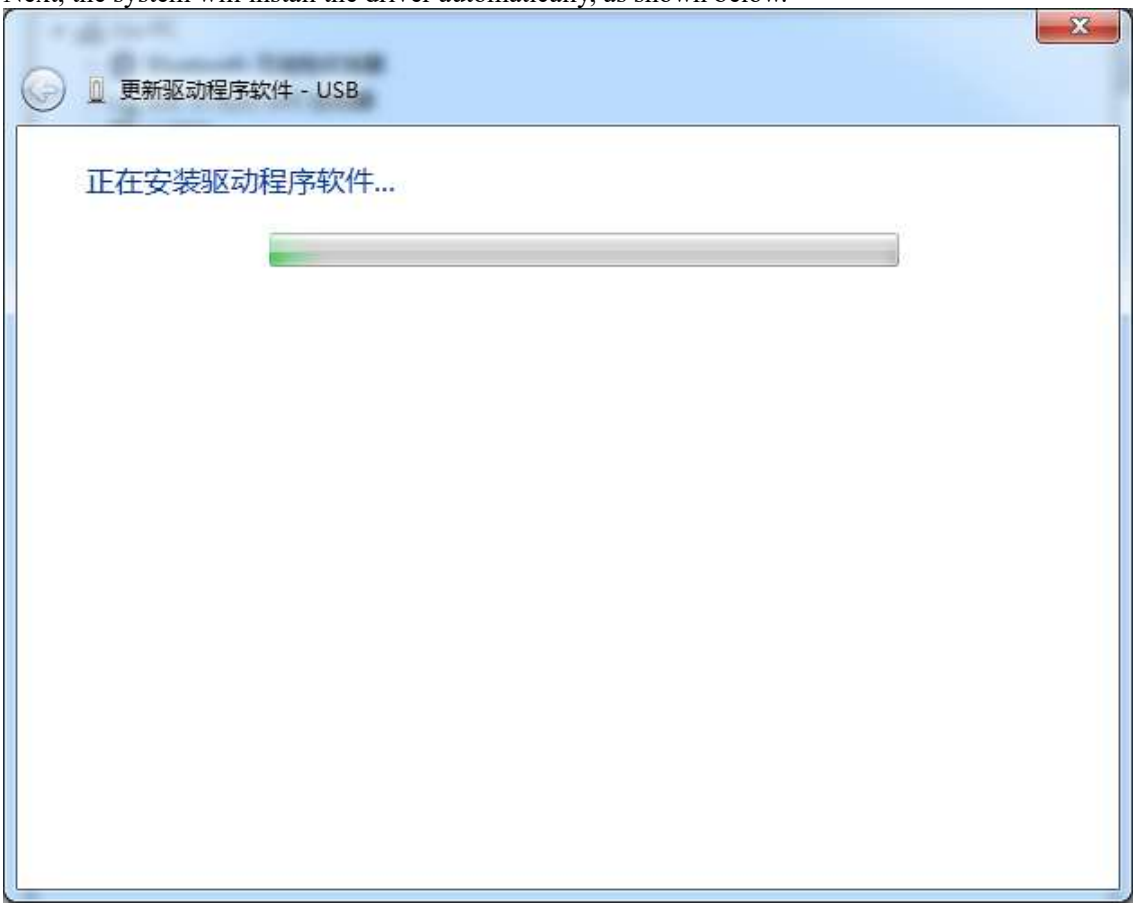
Click the "Browse" button in the dialog below to find the directory of the previous STC-USB driver stored (for example: the previous example directory is "D:\STC-USB", locate the path to the actual decompression directory).



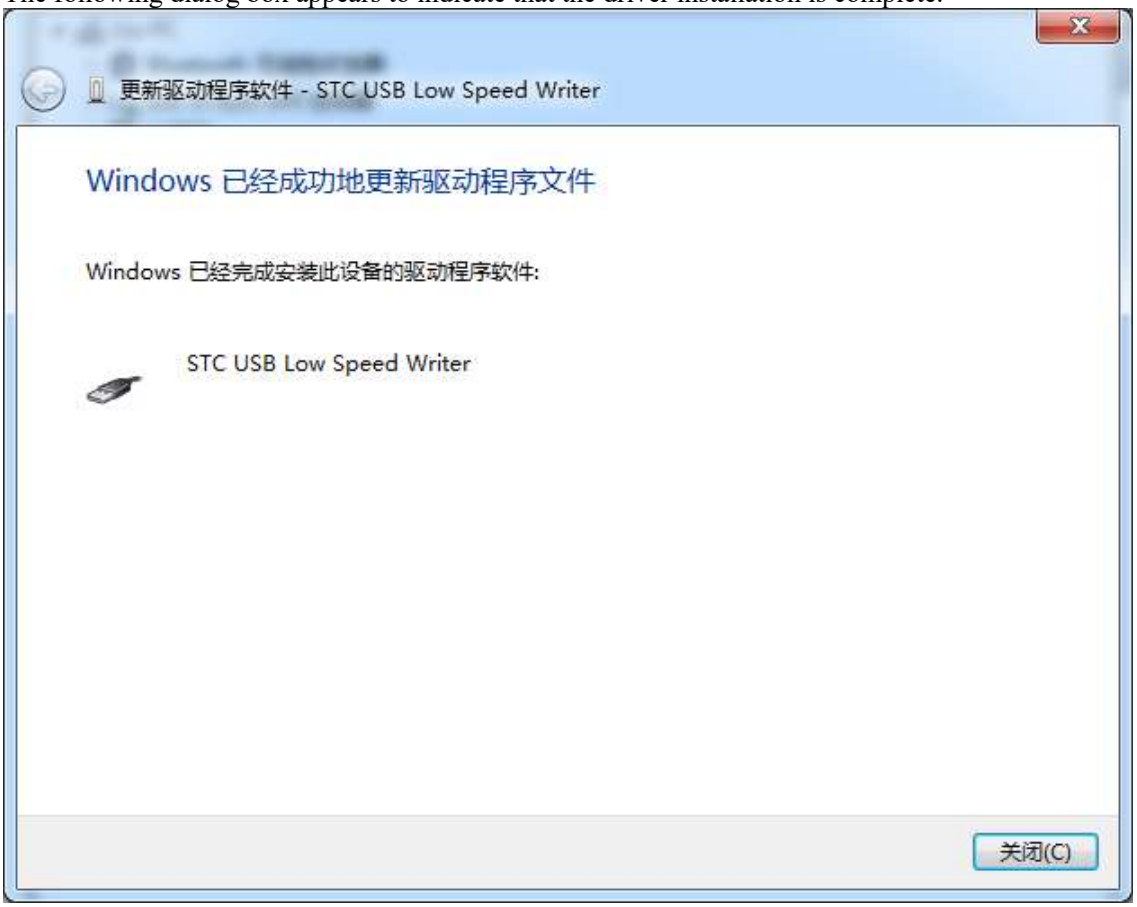
When the driver installation starts, the following dialog box will pop up, select "Always install this driver software".



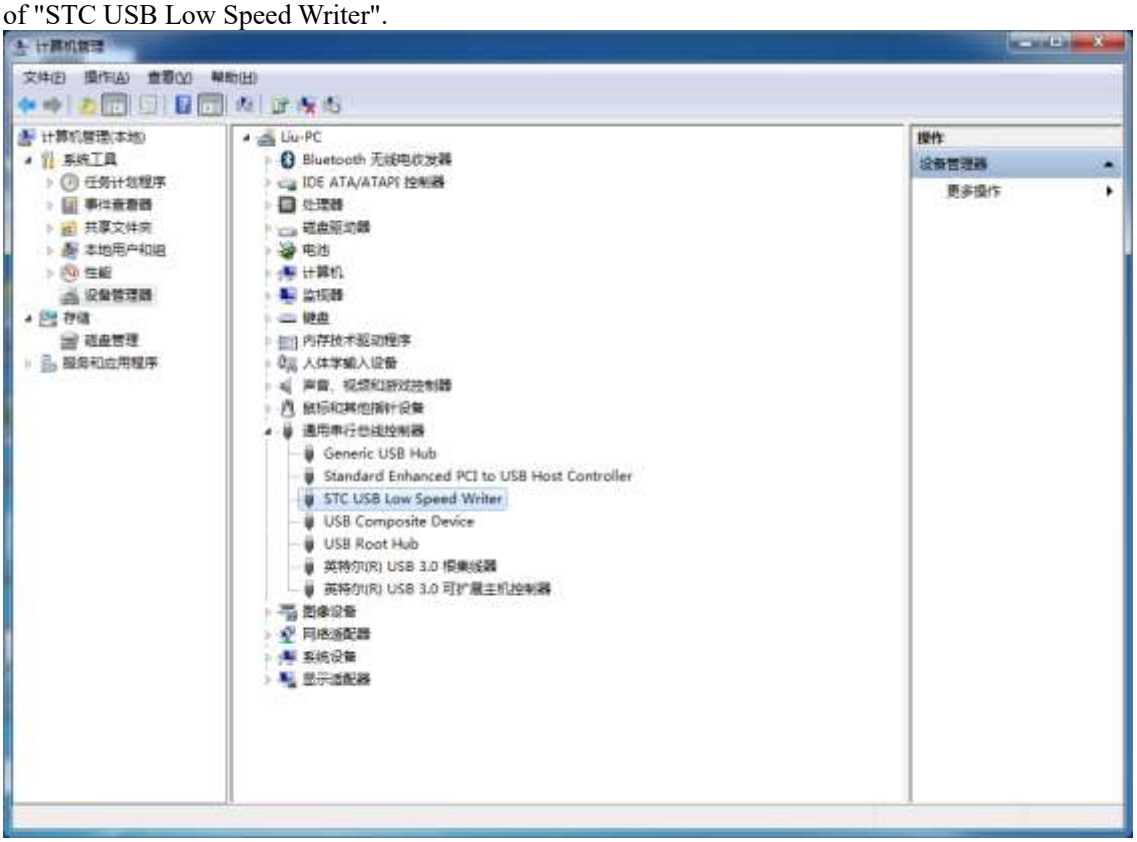
Next, the system will install the driver automatically, as shown below.



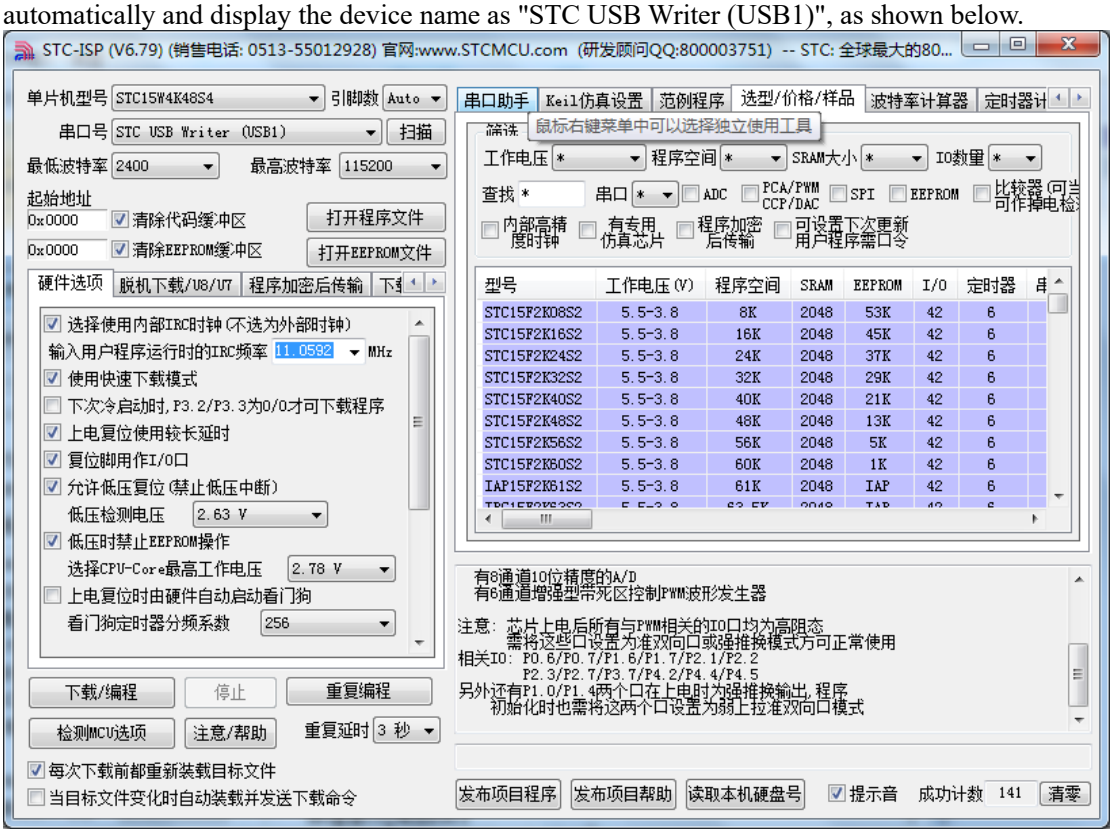
The following dialog box appears to indicate that the driver installation is complete.



Now in the device manager, the device with the yellow exclamation mark before will be displayed as the device name of "STC USB Low Speed Writer".

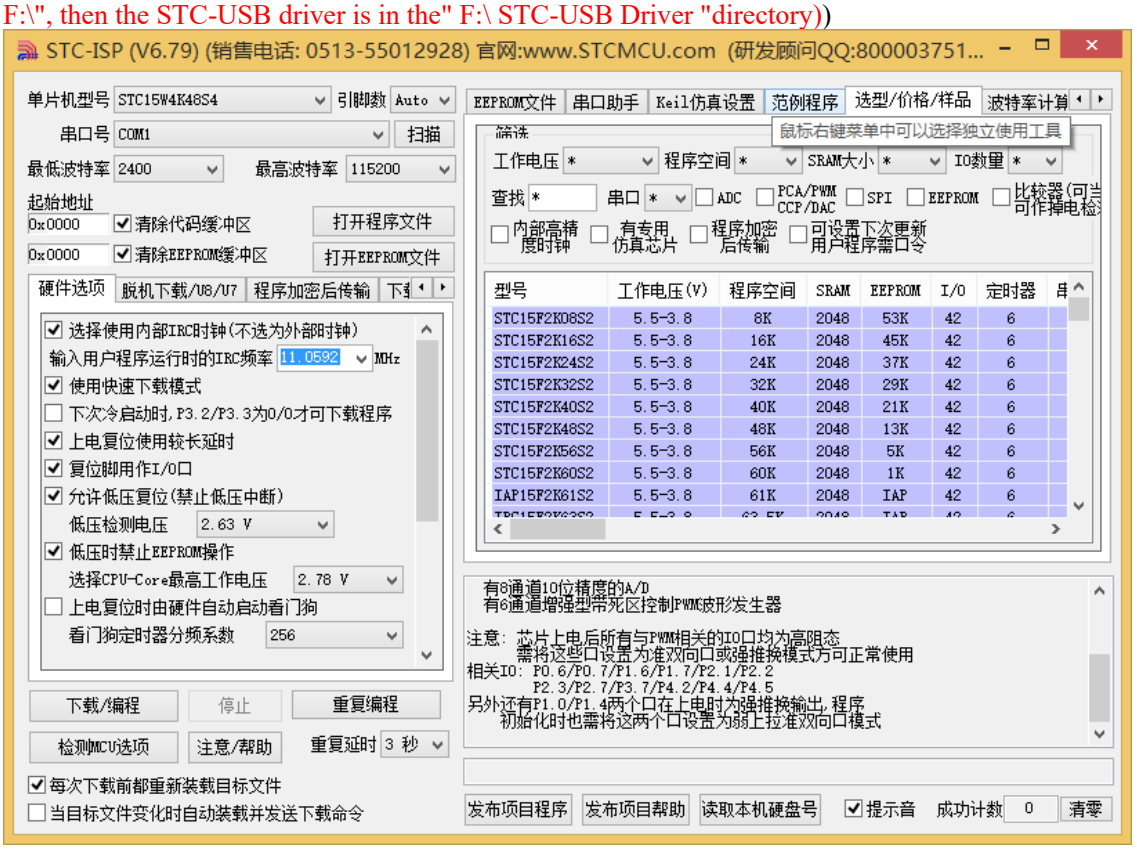


The serial number list in the previously downloaded STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (USB1)", as shown below.

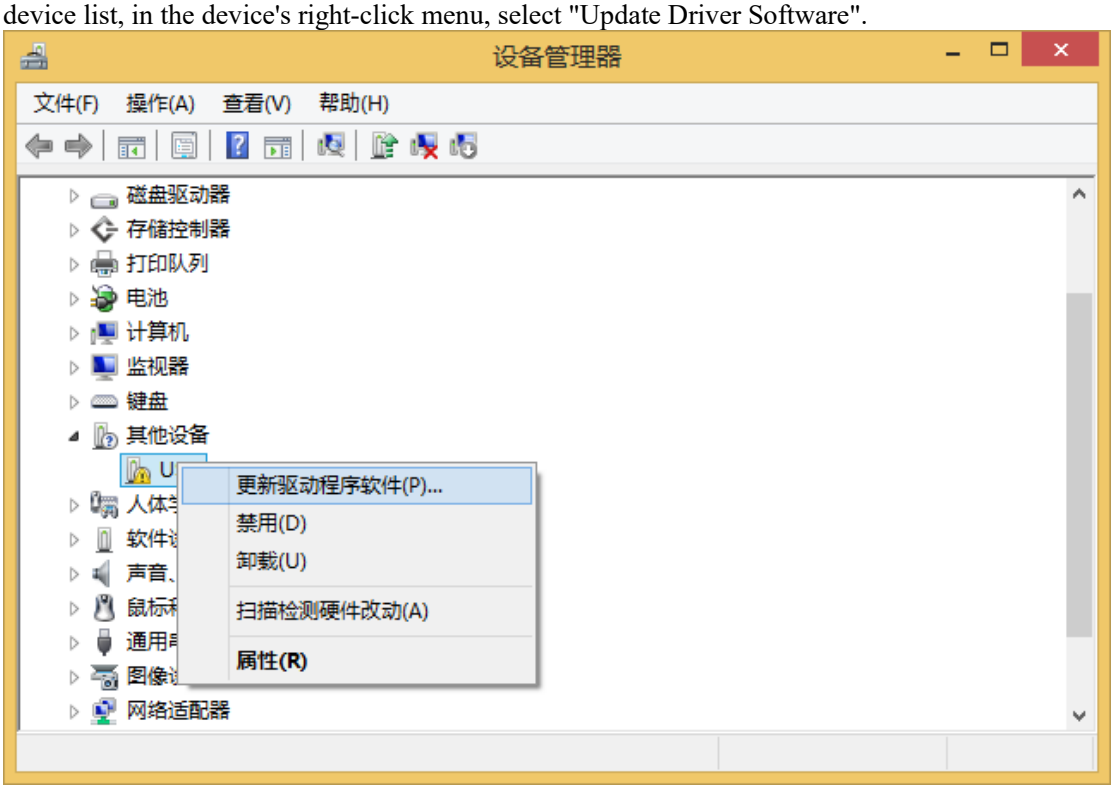


Installation Instructions in Windows 8 (32-bit)

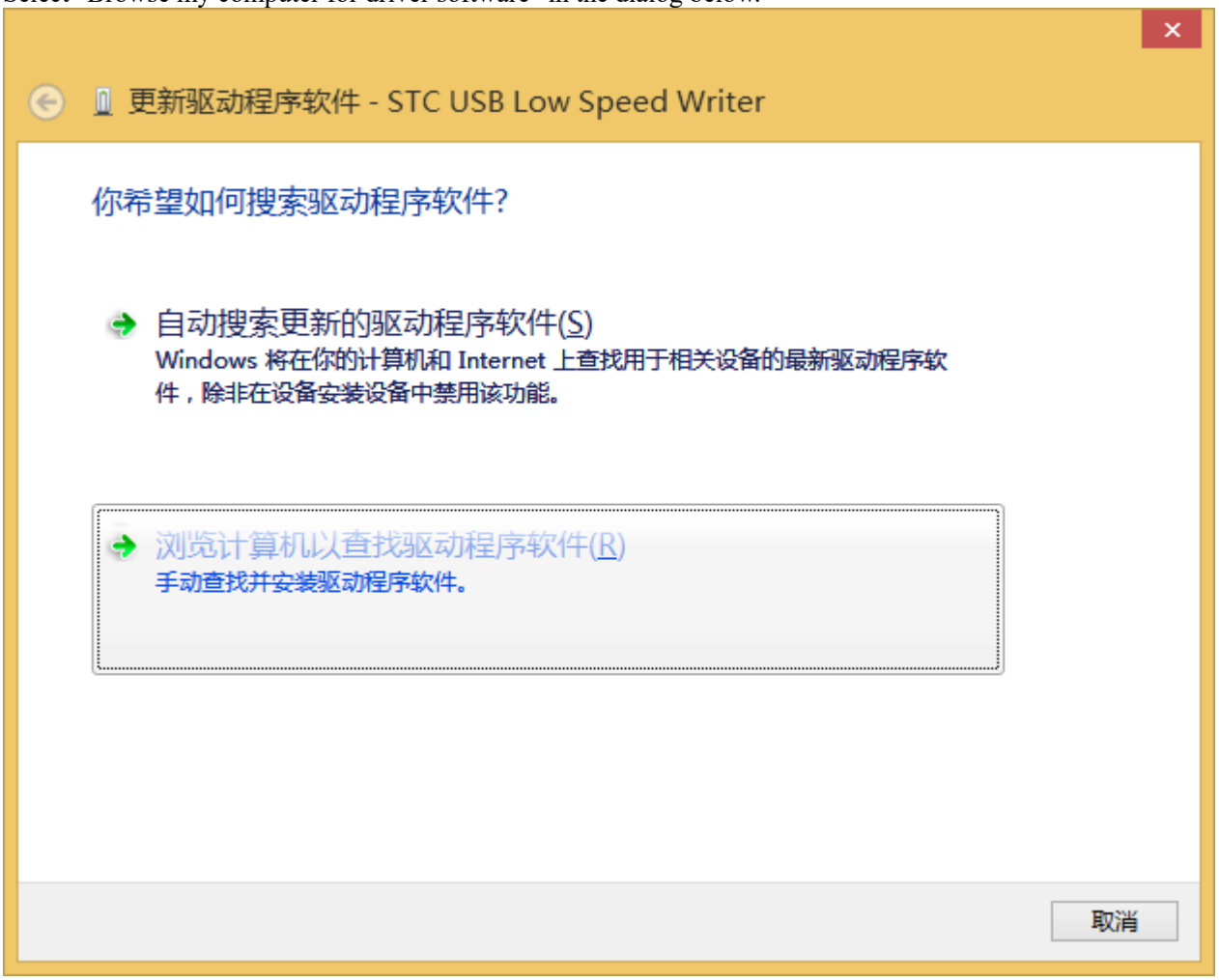
Open the STC-ISP download software of V6.79 (or newer version) (Due to permission reasons, downloading the software in Windows 8 will not copy the driver files to the relevant system directory. It requires manual installation by the user. Firstly, download "stc-isp-15xx-v6.79.zip" (or newer version) from the STC official website, and decompress it to the local disk after downloading, then the STC-USB driver file will also be decompressed to the "STC-USB Driver" folder of the current folder. (For example, decompress the downloaded compressed file" stc-isp-15xx-v6.79.zip "to" F:\", then the STC-USB driver is in the" F:\ STC-USB Driver "directory))



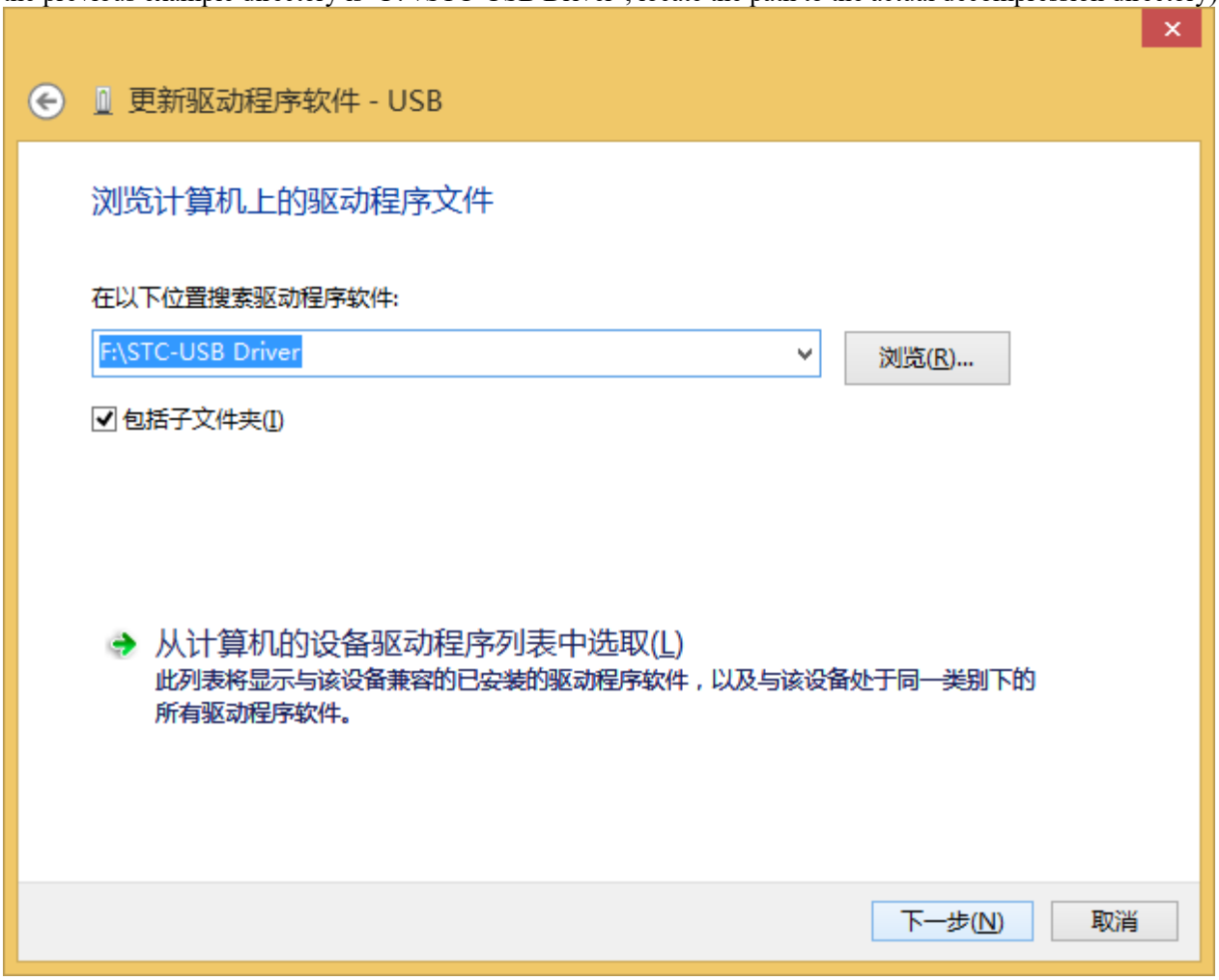
Plug in the USB device and open the Device Manager. Find the USB device with a yellow exclamation mark in the device list, in the device's right-click menu, select "Update Driver Software".



Select "Browse my computer for driver software" in the dialog below.

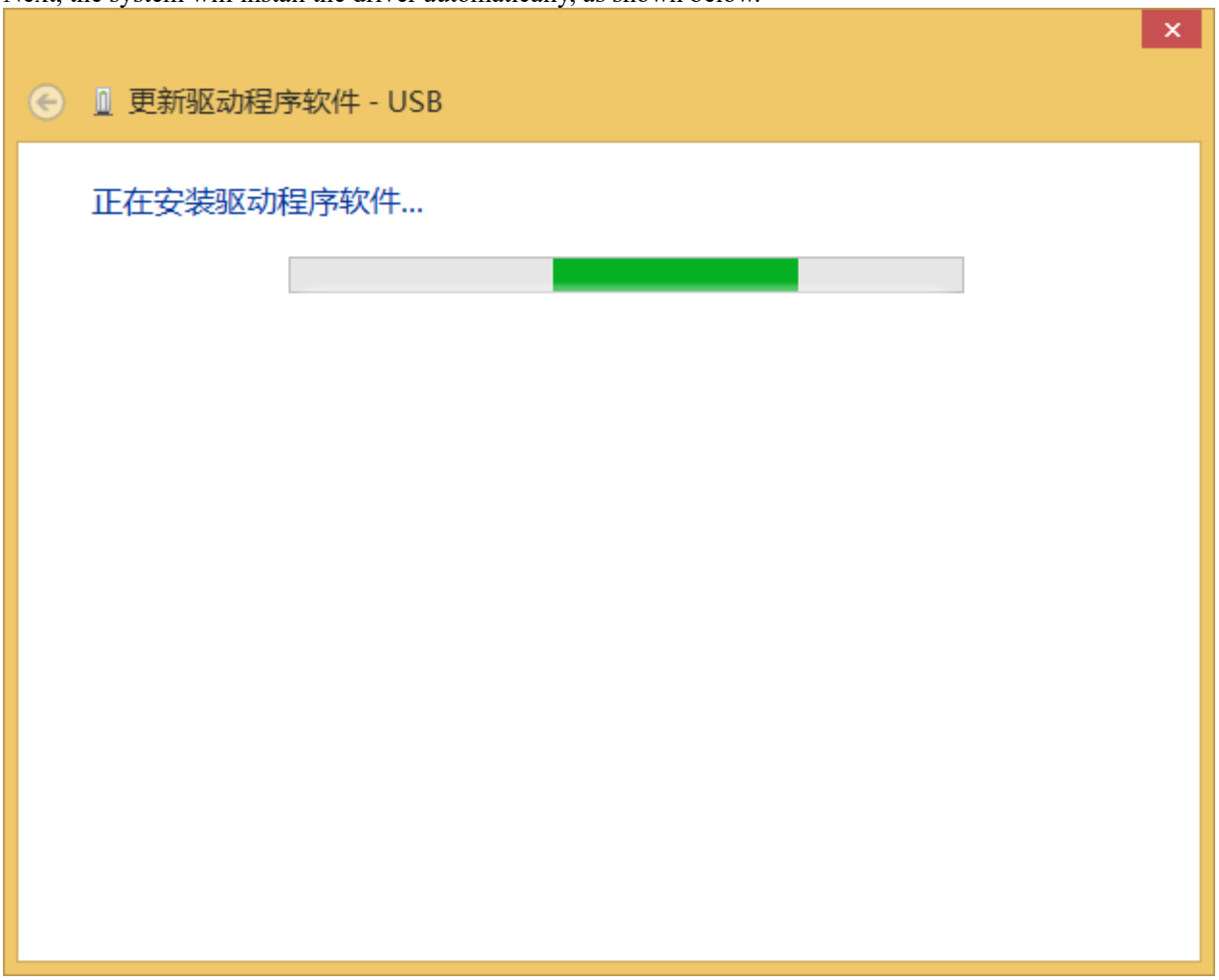


Click the "Browse" button in the dialog below to find the directory where the STC-USB driver was stored (for example: the previous example directory is "F: \ STC-USB Driver", locate the path to the actual decompression directory) .

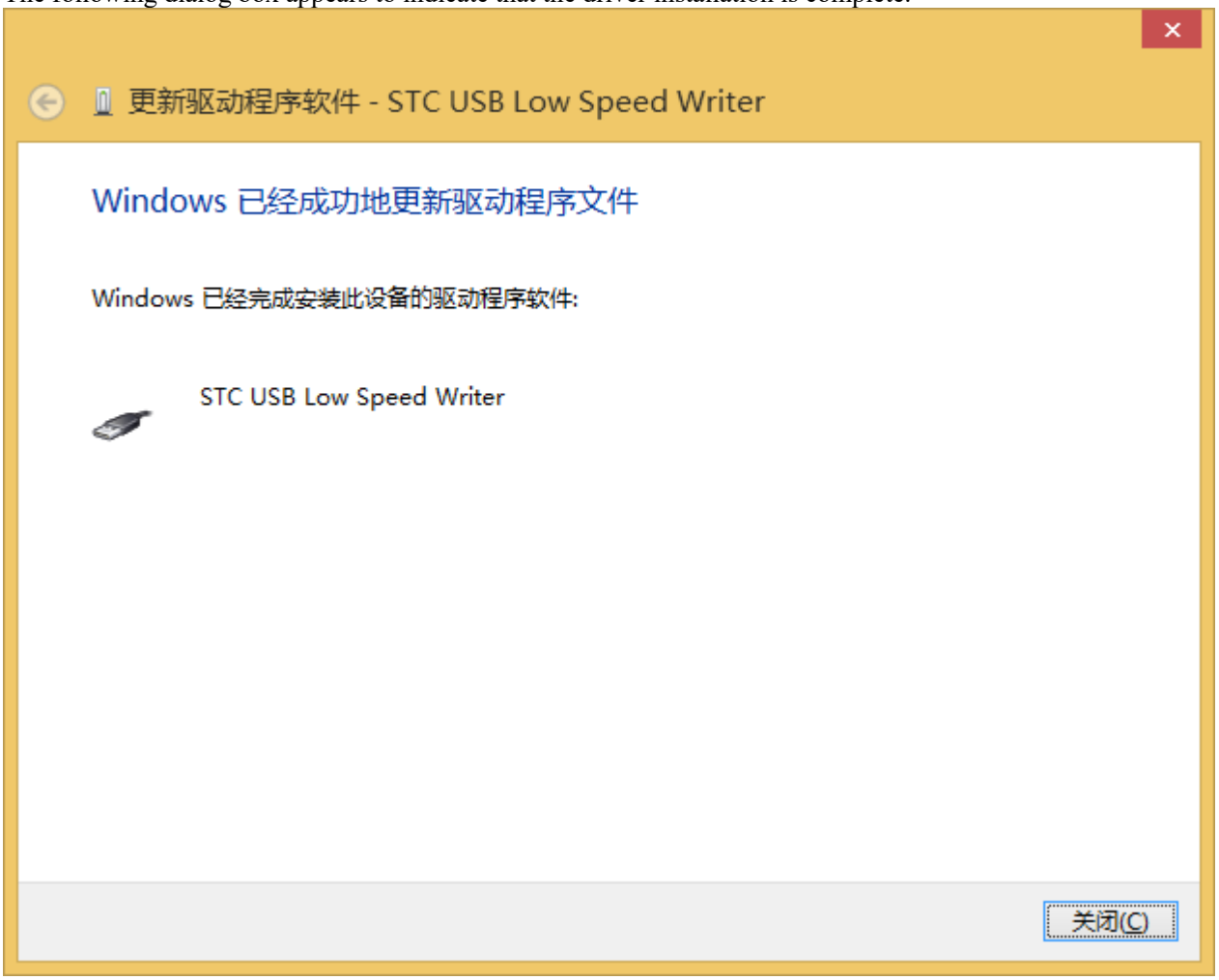


When the driver installation begins, the following dialog box will pop up, select "Always install this driver software".

Next, the system will install the driver automatically, as shown below.

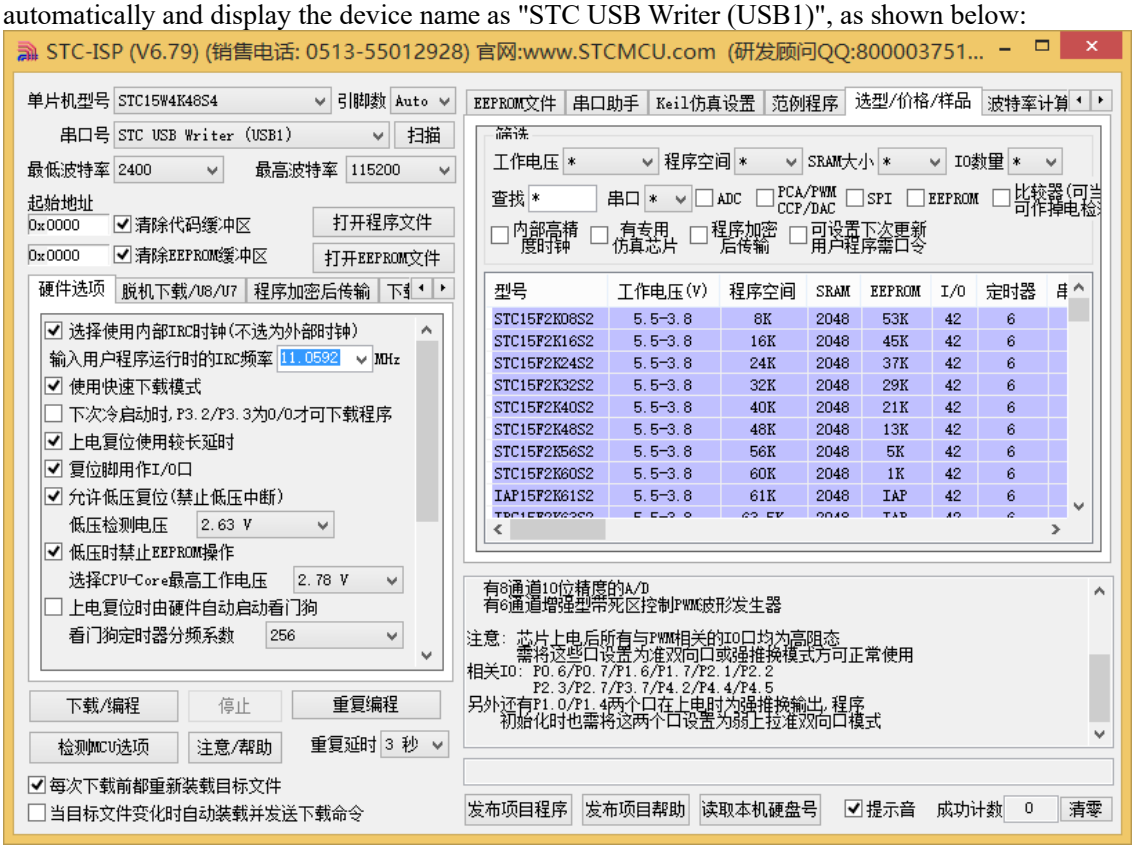


The following dialog box appears to indicate that the driver installation is complete.



Now in the device manager, the device with the yellow exclamation mark before will be displayed as the device name of "STC USB Low Speed Writer".

The serial number list in the previously downloaded STC-ISP download software will select the inserted USB device automatically and display the device name as "STC USB Writer (USB1)", as shown below:

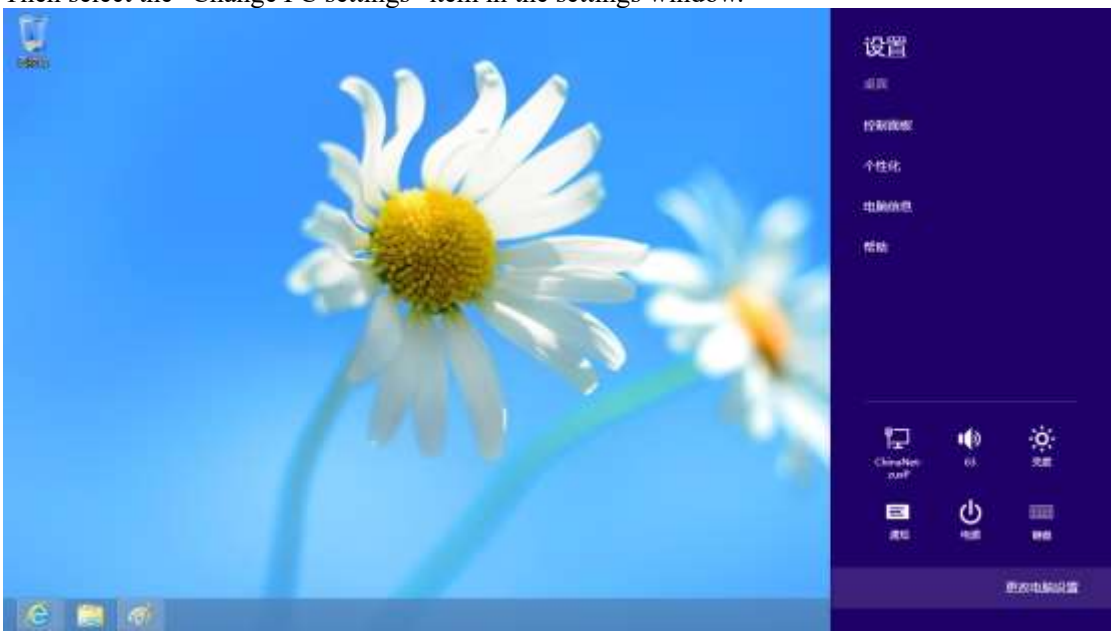


Installation Instructions in Windows 8 (64-bit)

By default, the driver without digital signature cannot be successfully installed in Windows 8 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

Firstly, move the mouse to the lower right corner of the screen and select the "Settings" button.

Then select the "Change PC settings" item in the settings window.



In the computer settings, select the "Start Now" button under the "Advanced Startup" item in the "General" property page.

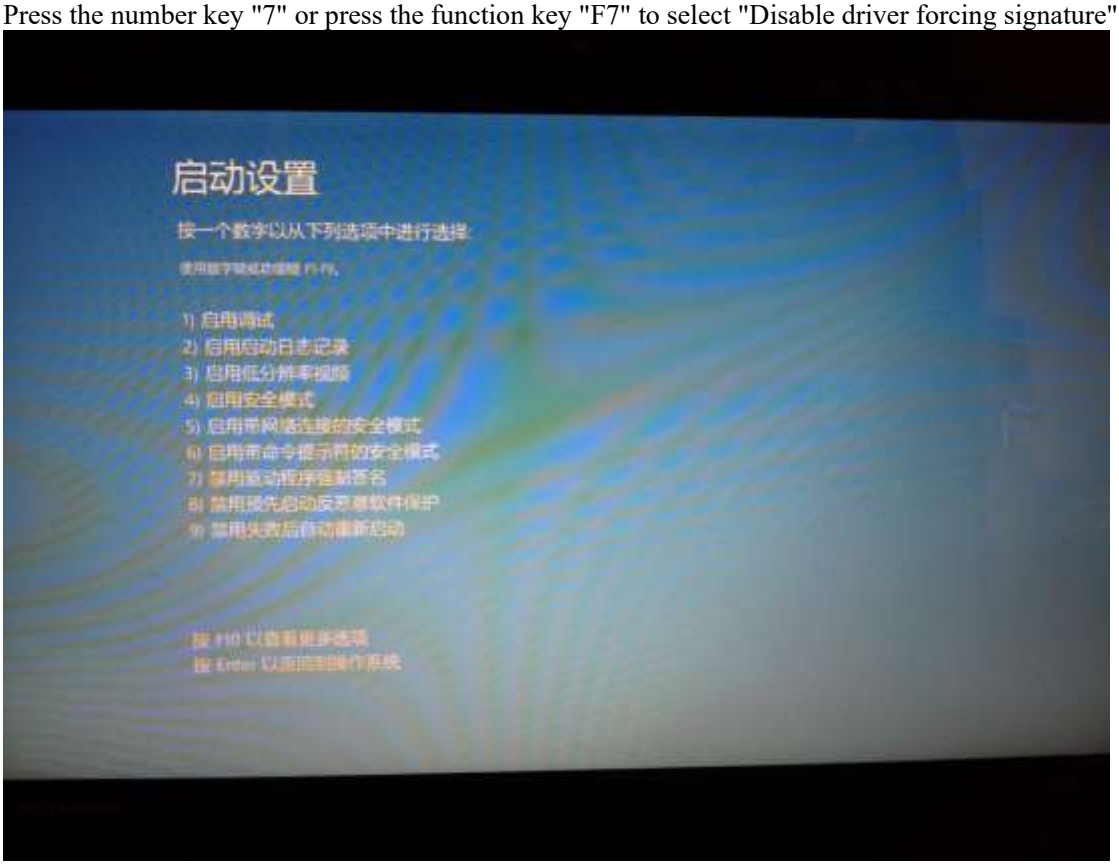
In the window below, select the "Troubleshooting" item.

Then select "Advanced Options" in "Troubleshooting".

In the "Advanced Options" window below, select "Startup Settings".

In the "Startup Settings" window below, click the "Restart" button to restart the computer.

After the computer restarts, it will enter the "Startup Settings" window automatically as shown in the figure below. Press the number key "7" or press the function key "F7" to select "Disable driver forcing signature" to start.



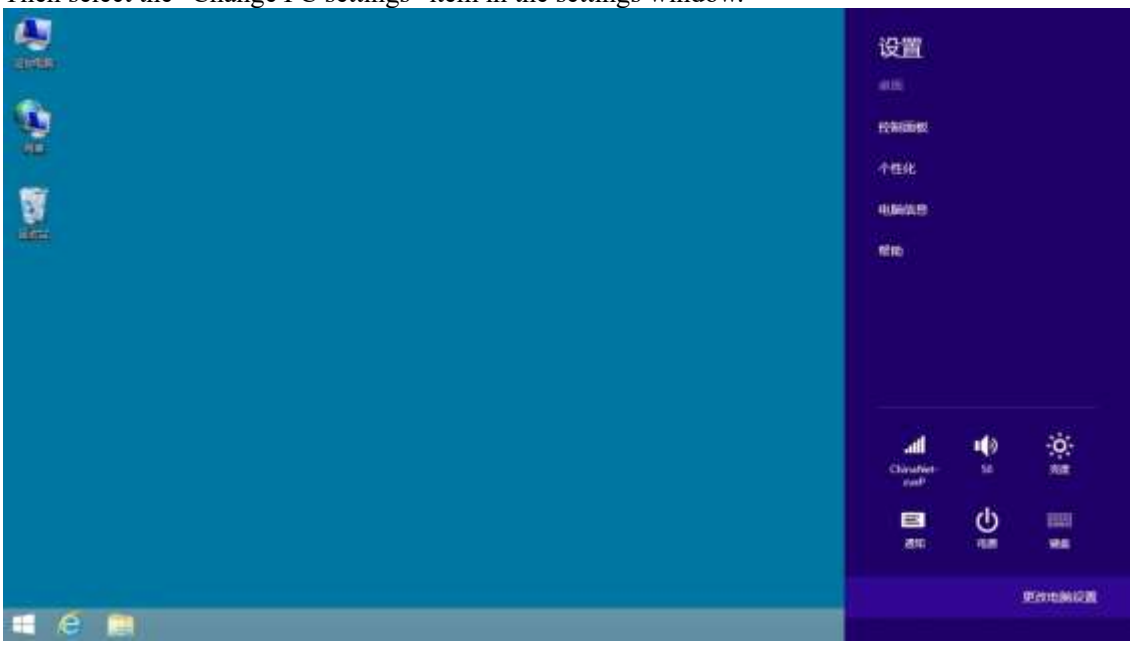
After booting to Windows 8, follow the Windows 8 (32-bit) installation method to complete the driver installation.

Installation Instructions in Windows 8.1 (64-bit)

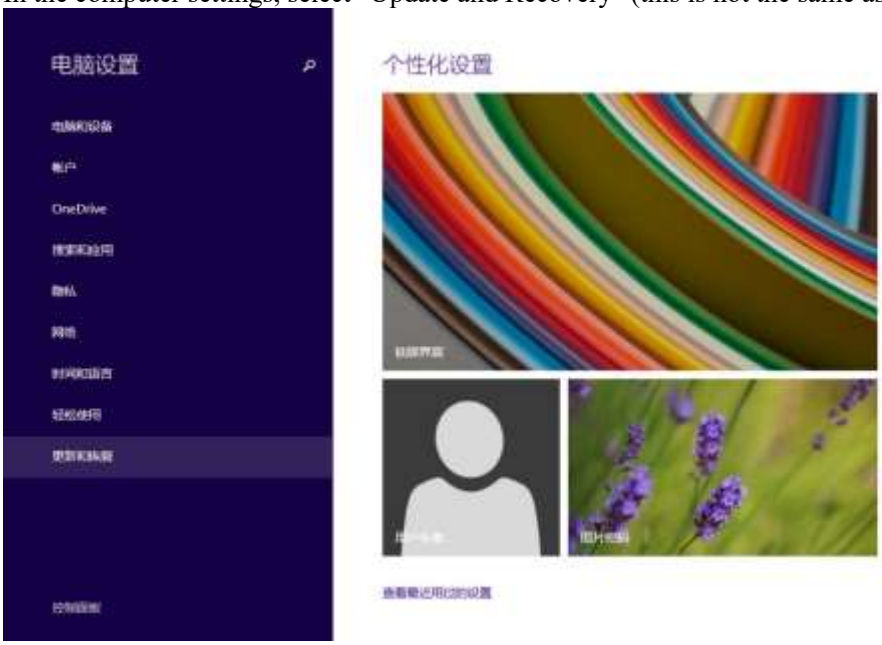
Windows 8.1 has different method for entering the advanced boot menu with respect to Windows 8, which is specifically explained here.

Firstly, move the mouse to the lower right corner of the screen and select the "Settings" button.

Then select the "Change PC settings" item in the settings window.



In the computer settings, select "Update and Recovery" (this is not the same as Windows 8, which is "General").



Select the "Restore" property page in the update and recovery page, and click the "Start Now" button under the "Advanced Startup" item.

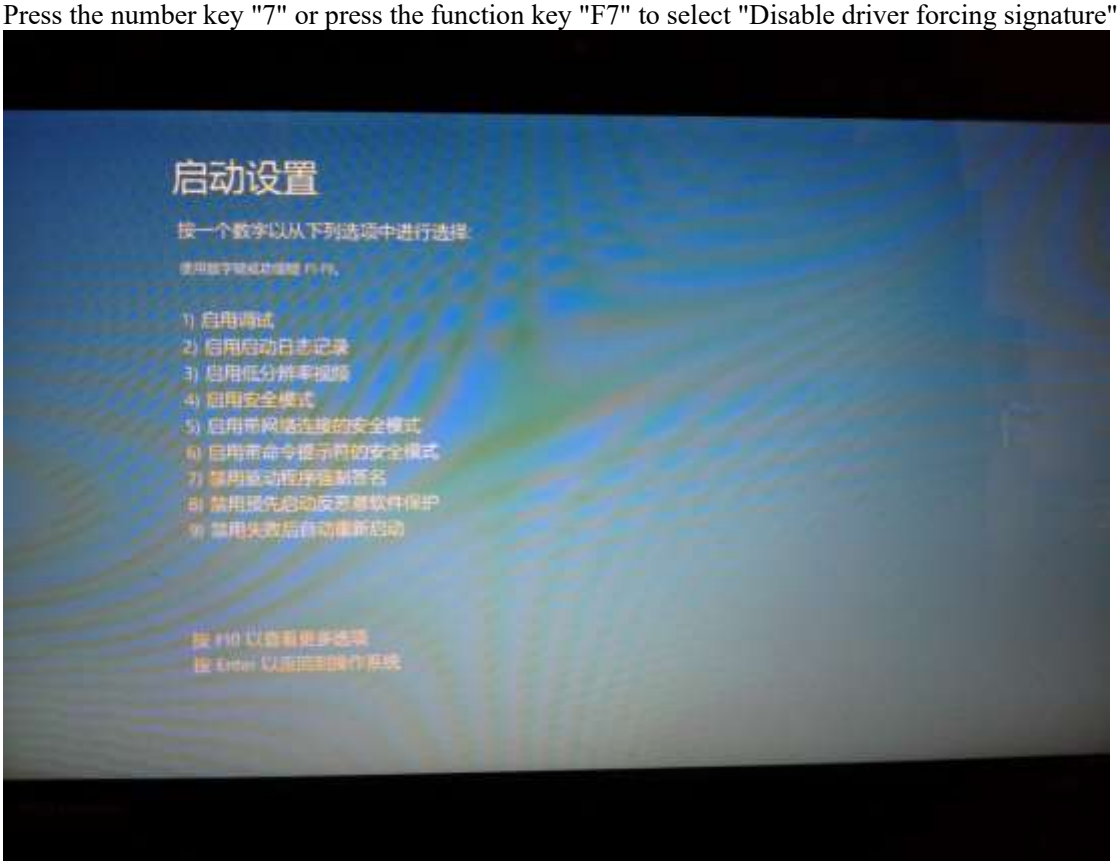
The following steps are the same as those of Window 8.
In the window below, select the "Troubleshooting" item.

Then select "Advanced Options" in "Troubleshooting".

In the "Advanced Options" window below, select "Startup Settings".

In the "Startup Settings" window below, click the "Restart" button to restart the computer.

After the computer restarts, it will enter the "Startup Settings" window automatically as shown in the figure below. Press the number key "7" or press the function key "F7" to select "Disable driver forcing signature" to start.



After booting to Windows 8, follow the Windows 8 (32-bit) installation method to complete the driver installation.

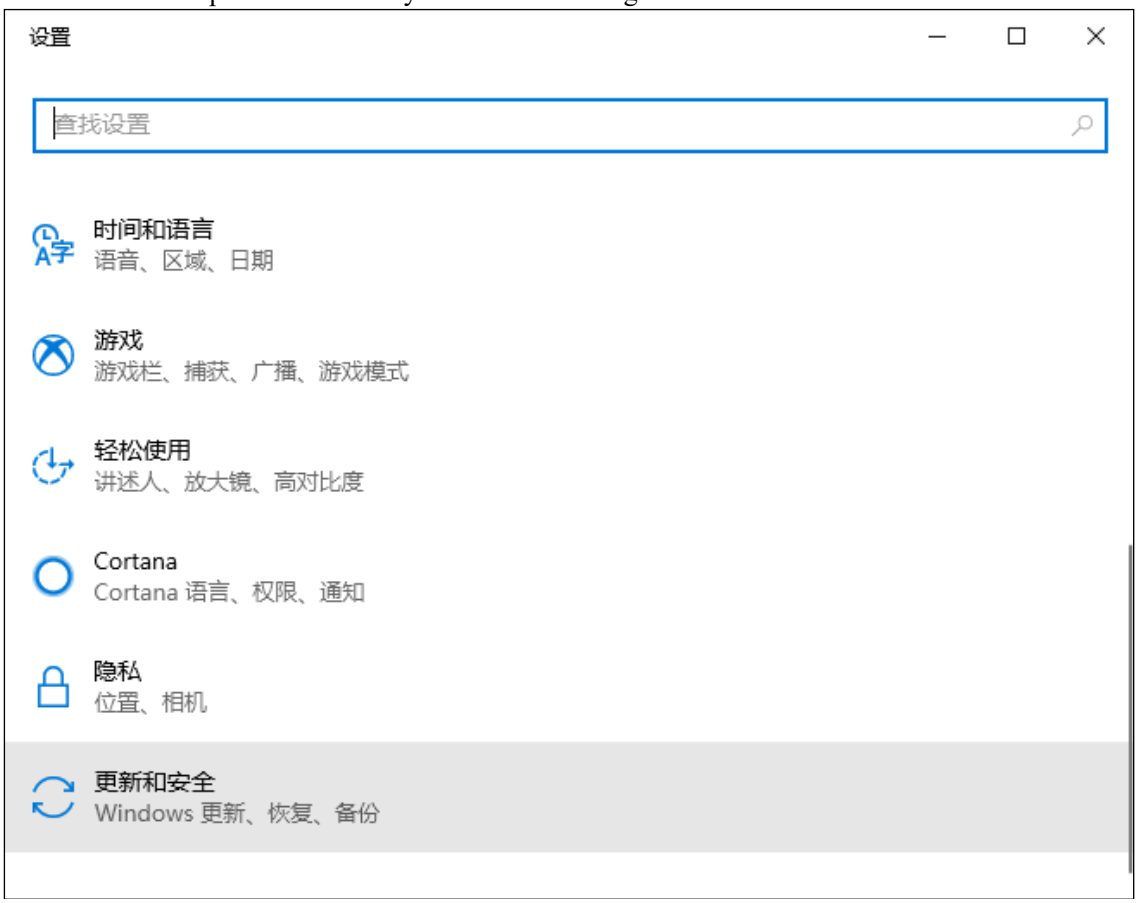
Installation Instructions in Windows 10 (64-bit)

By default, the driver without digital signature cannot be successfully installed in Windows 10 64-bit operating system. So, you need to follow the steps below before installing the STC-USB driver, skip the digital signature temporarily, and the installation will be successful.

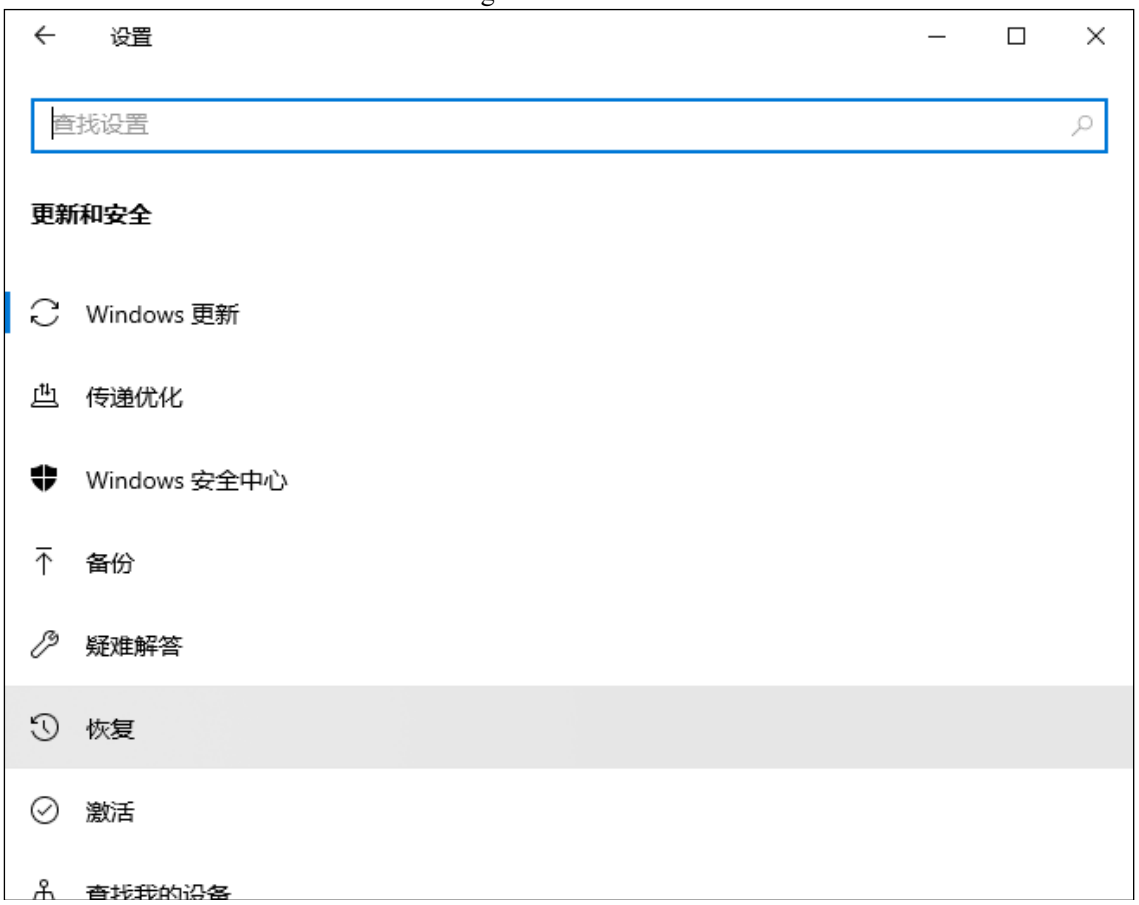
Before installing the driver, you need to extract the "STC-USB Driver" folder to the hard disk from the STC-ISP download software package downloaded from the STC official website. Prepare the chip with USB download function, but don't connect the computer firstly.

Right-click on the "Start" menu and select the "Settings" option.

Then select the "Update and Security" item in the settings window.



Then select the "Restore" item in the settings window.



In the recovery window, click the "Restart Now" button in the "Advanced Startup" item.

Before the computer restarts, the system will enter the following boot menu firstly and select the "Troubleshooting" item.

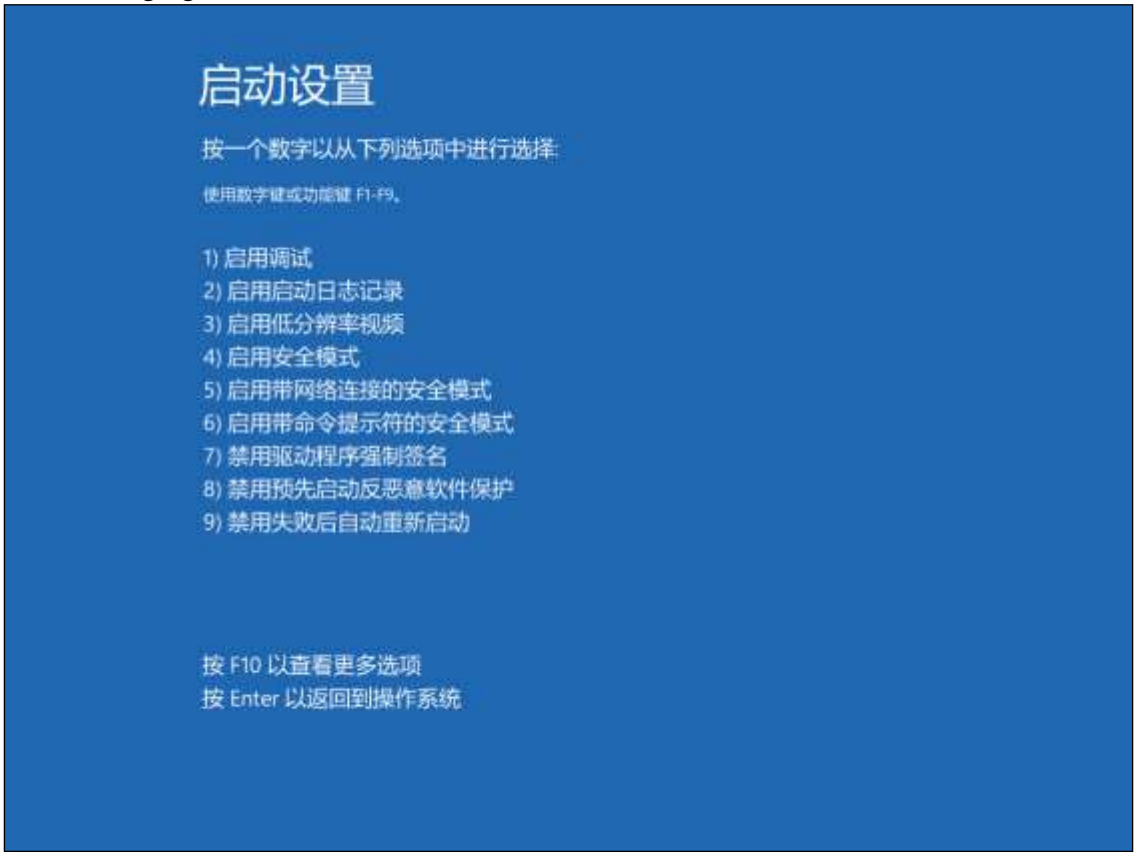
Select "Advanced Options" in the troubleshooting window.

Then select "View more recovery options".

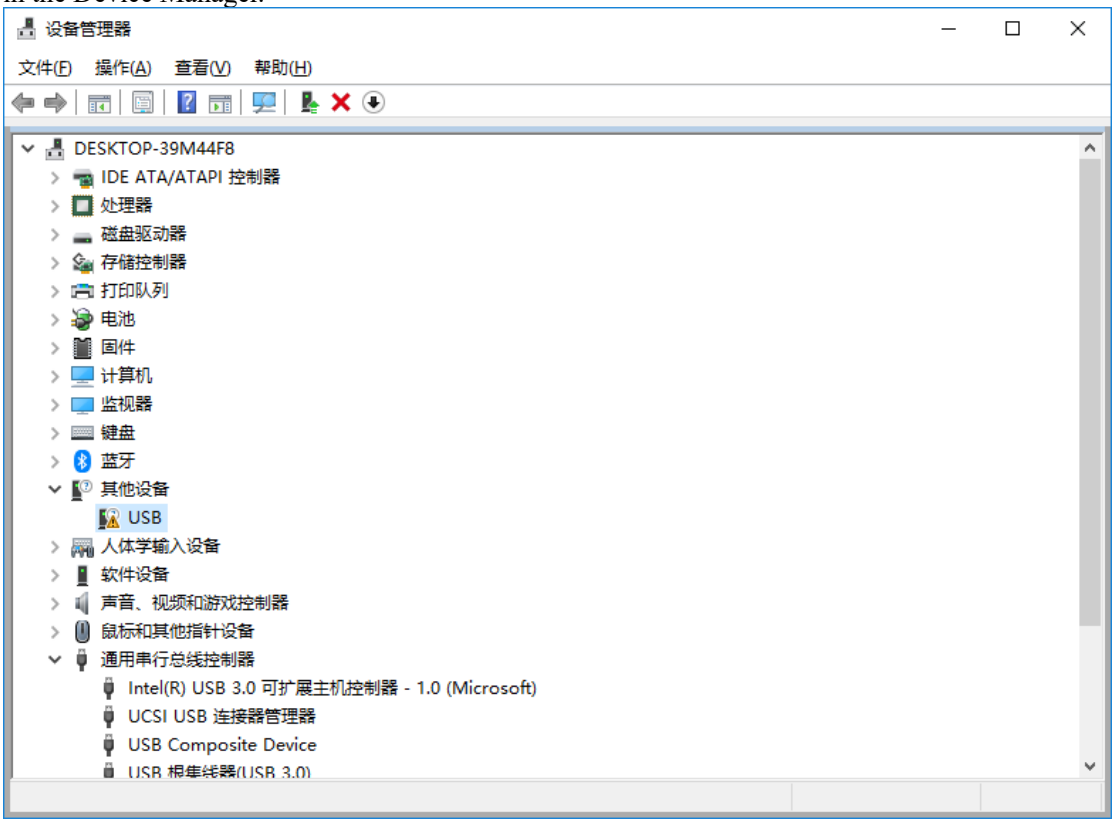
Select the "Startup Settings" item.

When the following screen appears, click the "Restart" button to restart the computer.

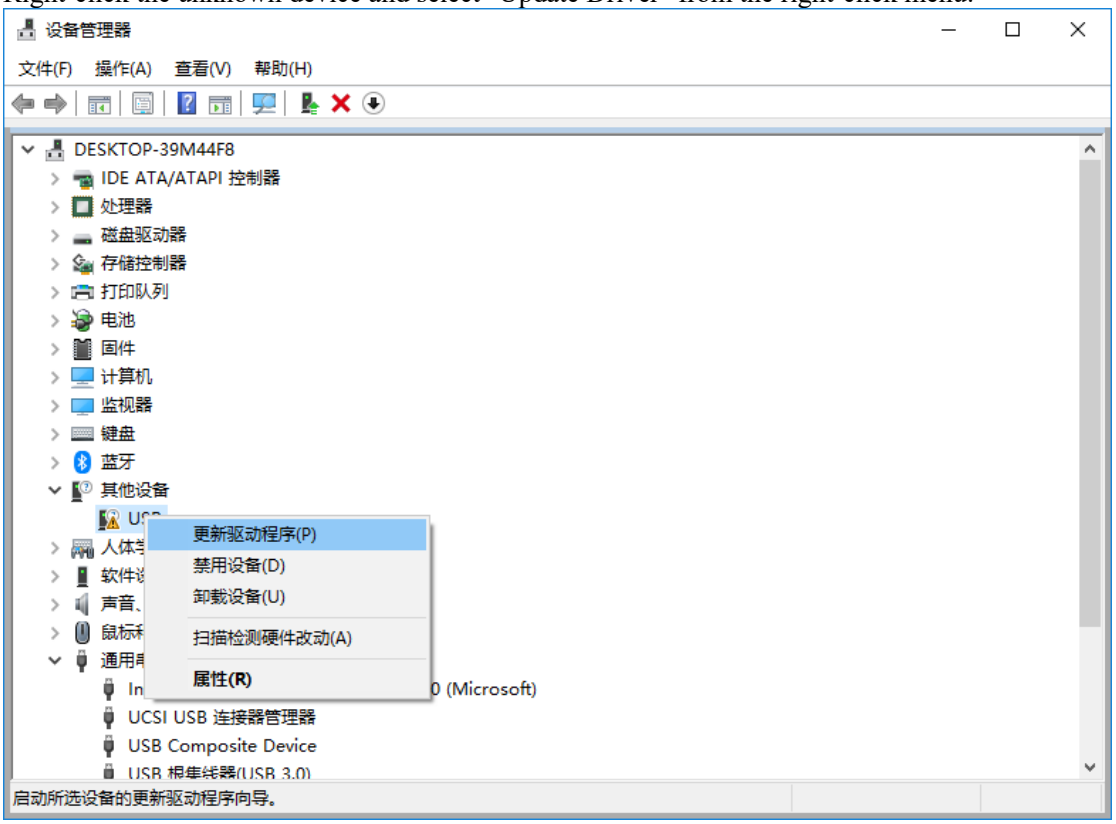
After the computer restarts, the "Startup Settings" window will pop up. Press the "F7" button to select the "Prohibit driver forcing signature" item.



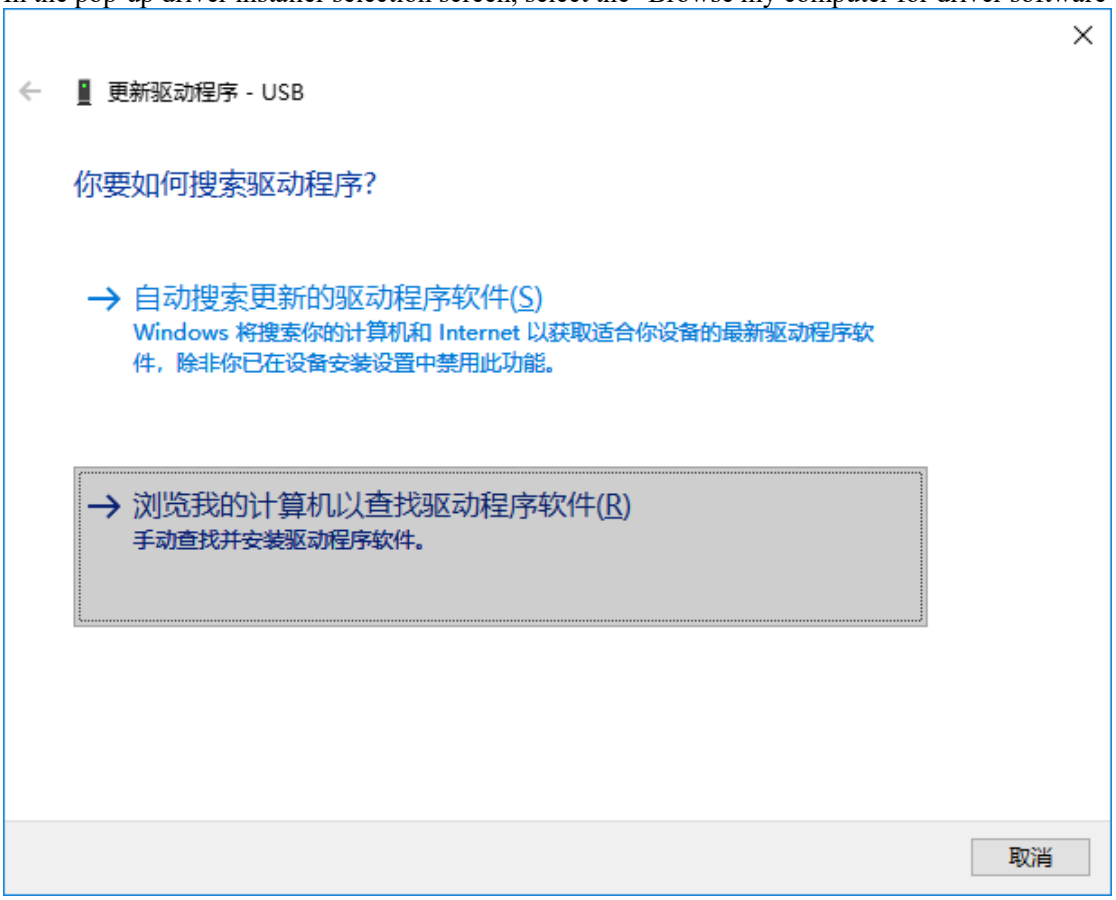
After the computer is started, connect the prepared chip to the computer with a USB cable and open the "Device Manager". Now, the driver has not yet been installed, so it will appear as an unknown device with an exclamation point in the Device Manager.



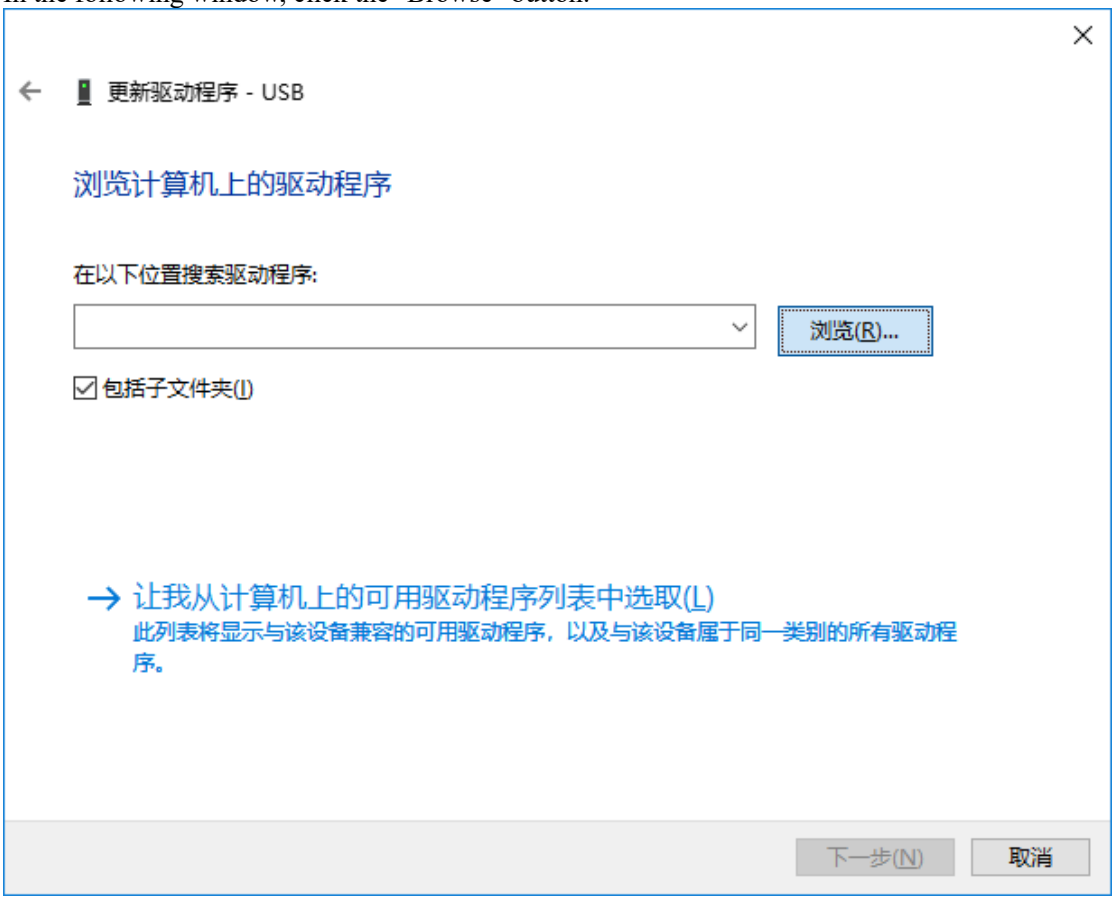
Right-click the unknown device and select "Update Driver" from the right-click menu.



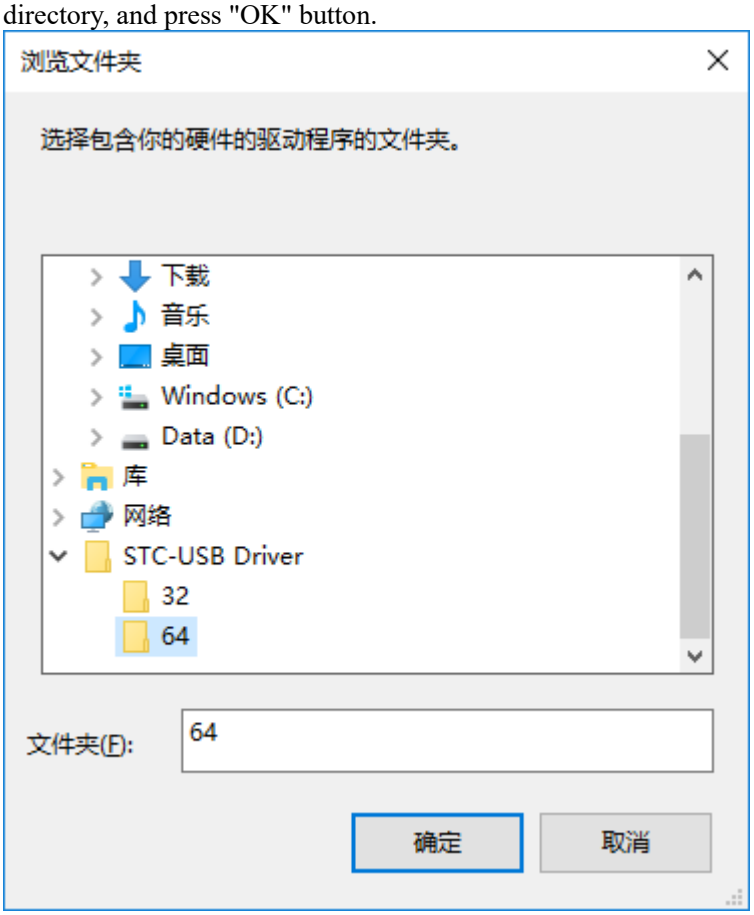
In the pop-up driver installer selection screen, select the "Browse my computer for driver software" item.



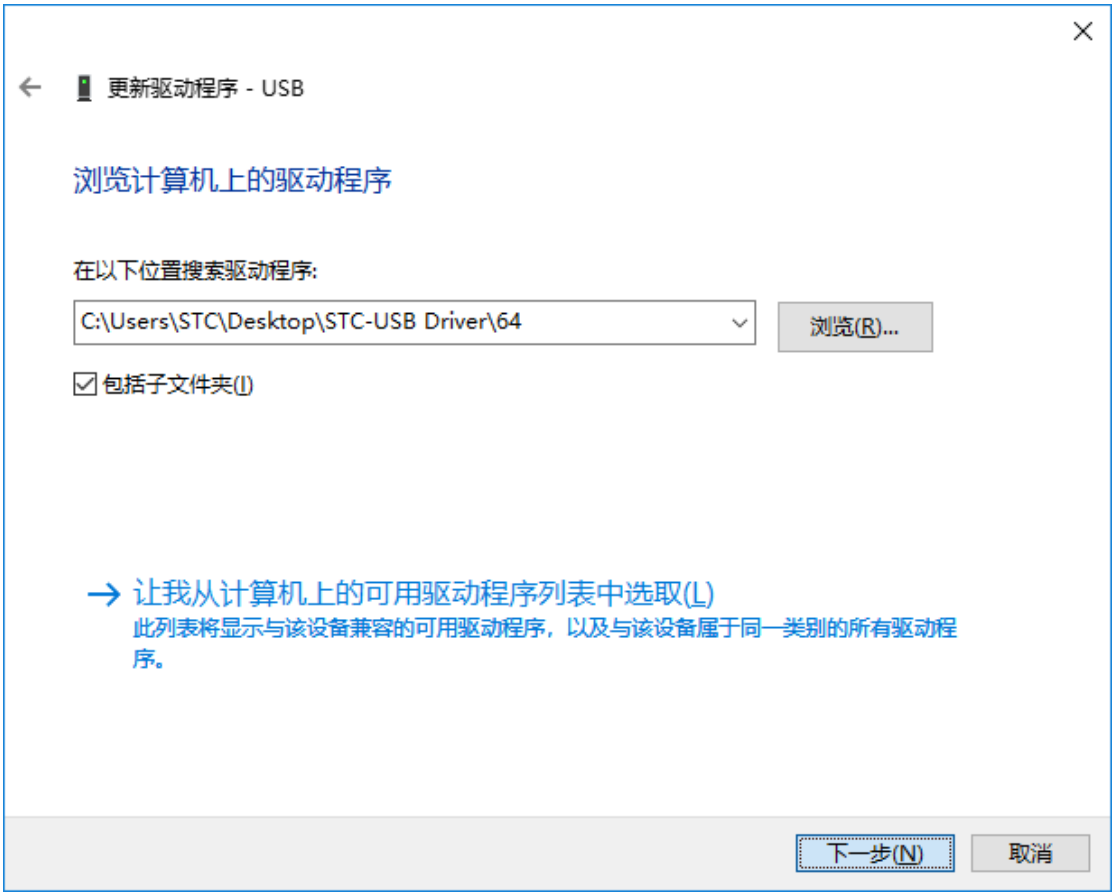
In the following window, click the "Browse" button.



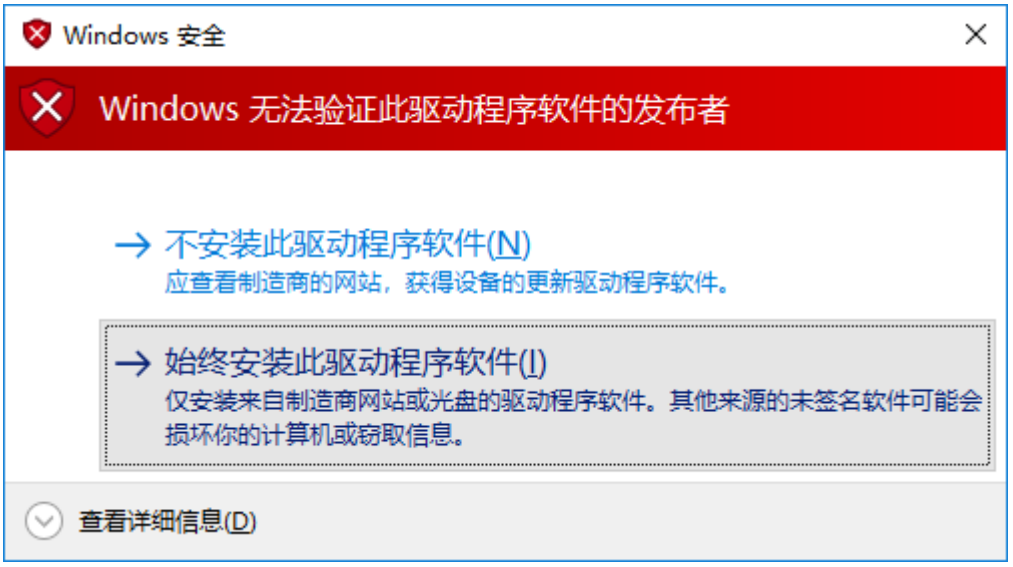
Find the "STC-USB Driver" directory that was previously extracted to your hard disk, select the "64" directory in the directory, and press "OK" button.



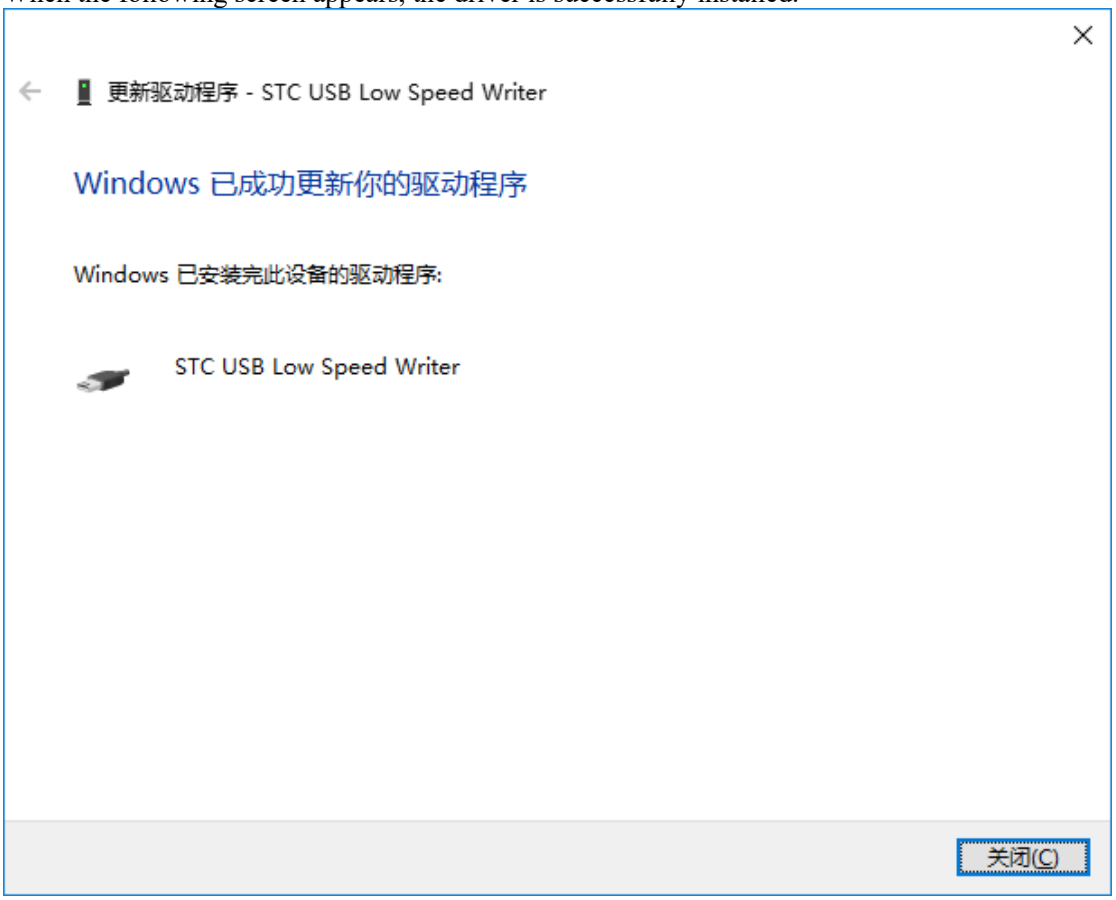
Click "Next" to start the driver installation.



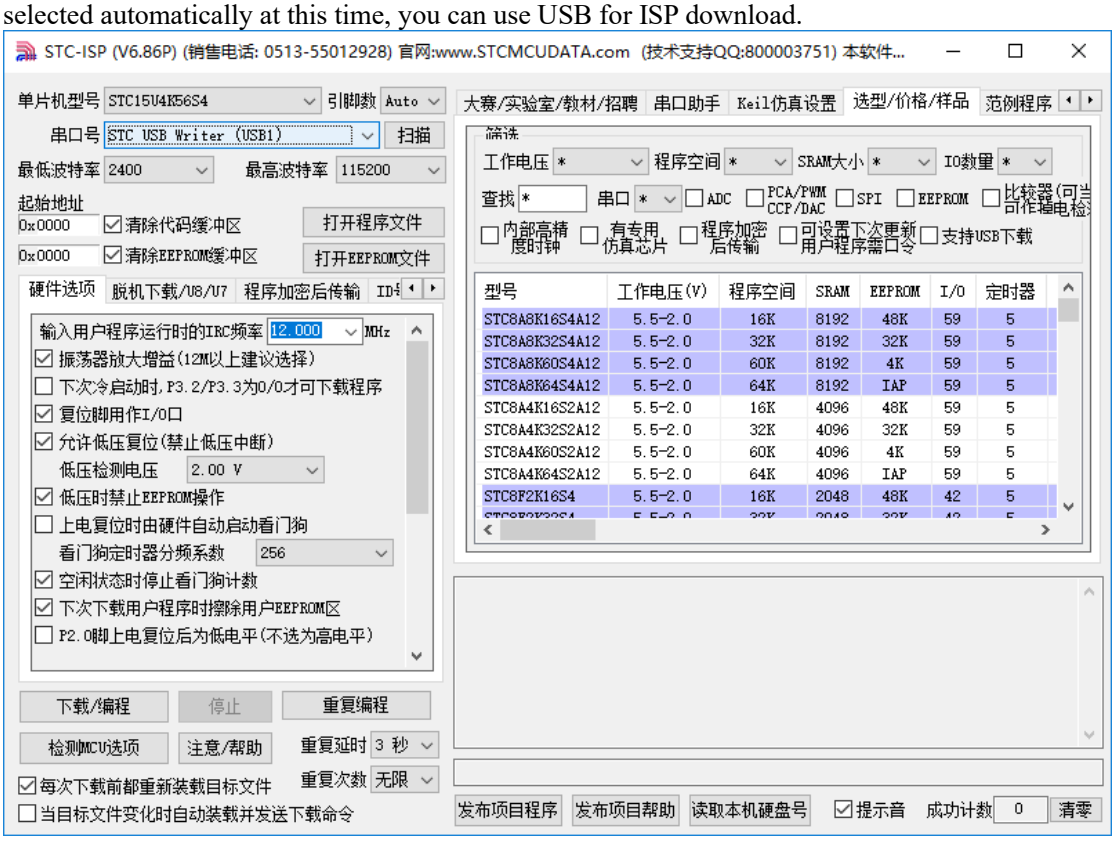
During the driver installation process, the following warning screen will pop up, select "Always install this driver software".



When the following screen appears, the driver is successfully installed.

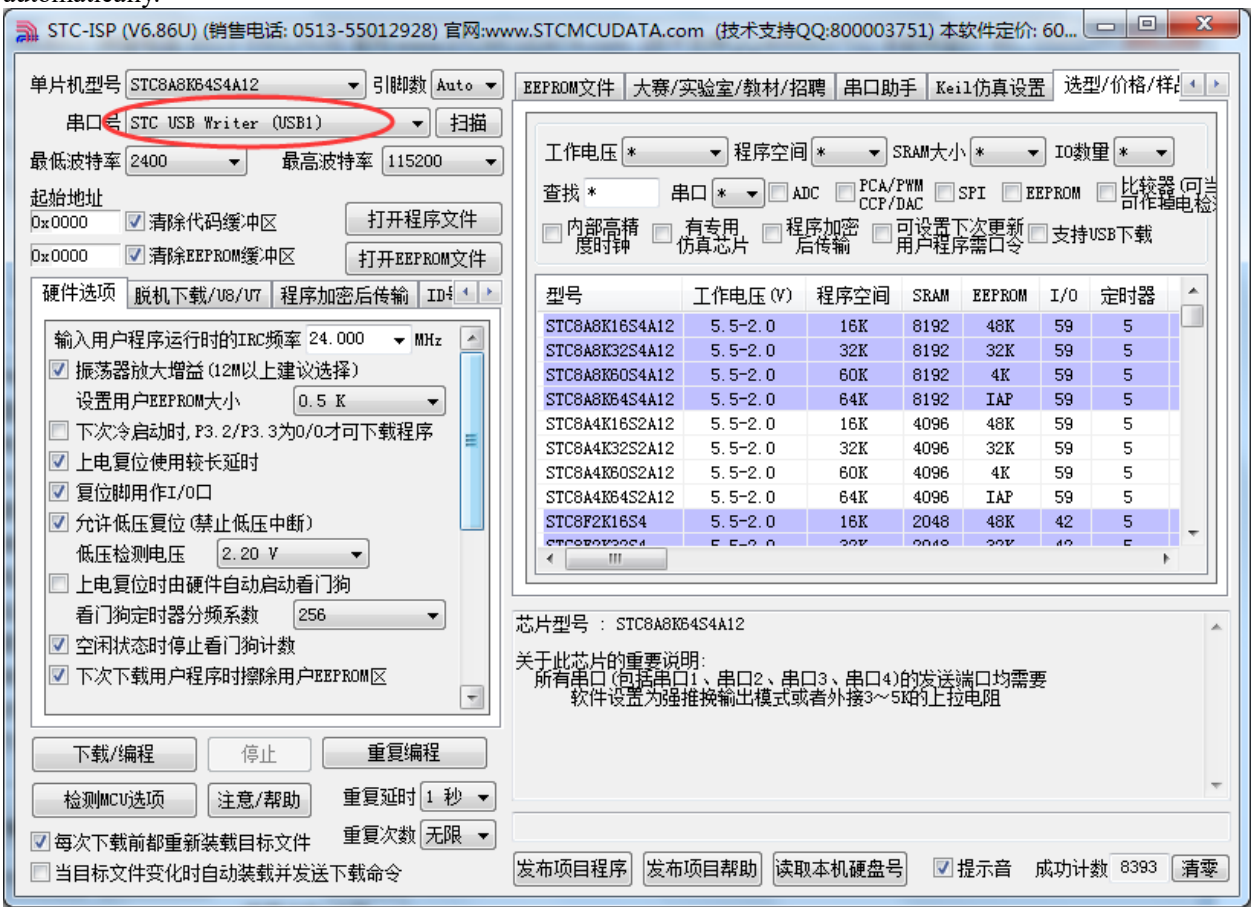


Back to the download software of STC-ISP, "STC USB Writer (USB1)" in the "Serial Port Number" drop-down list is selected automatically at this time, you can use USB for ISP download.

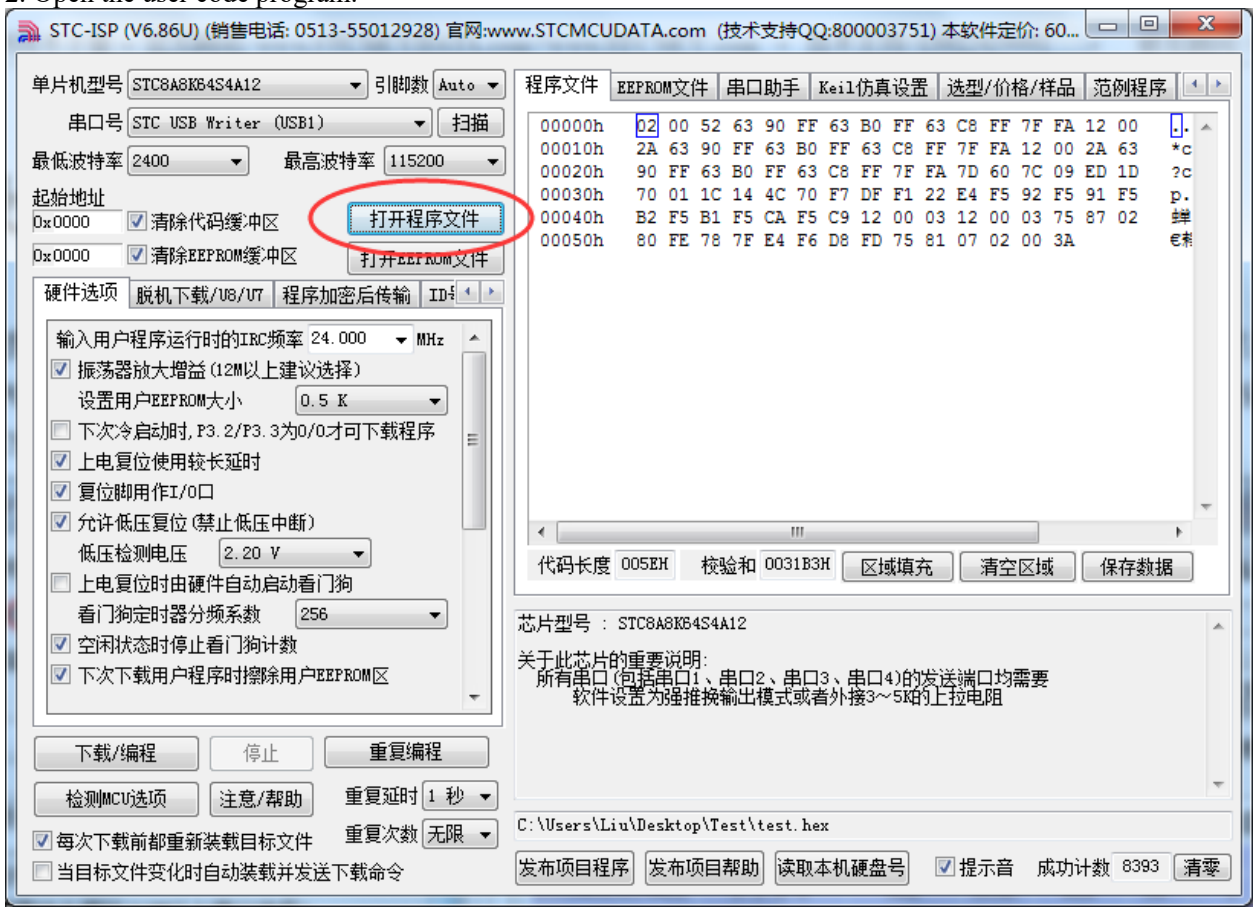


Appendix E Download Step Demo using USB

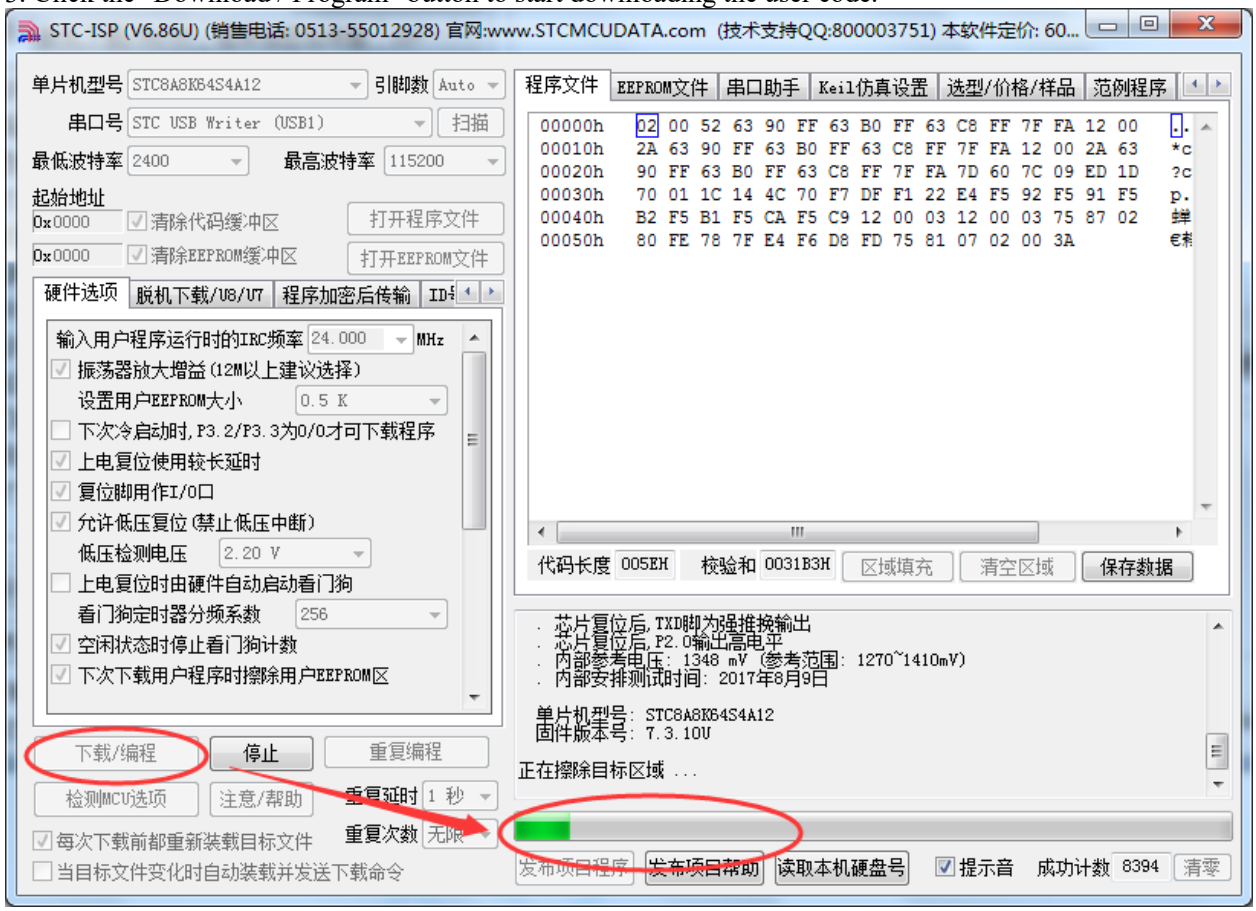
1. Refer to the application circuit diagram in P5.1.5 to connect the microcontroller firstly, and connect the P3.2 port of the target chip to GND, and then connect the system to the USB port on the PC side. Open the ISP to download the software, and the serial number of the downloaded software will search for the "STC USB Writer (USB1)" USB device automatically.



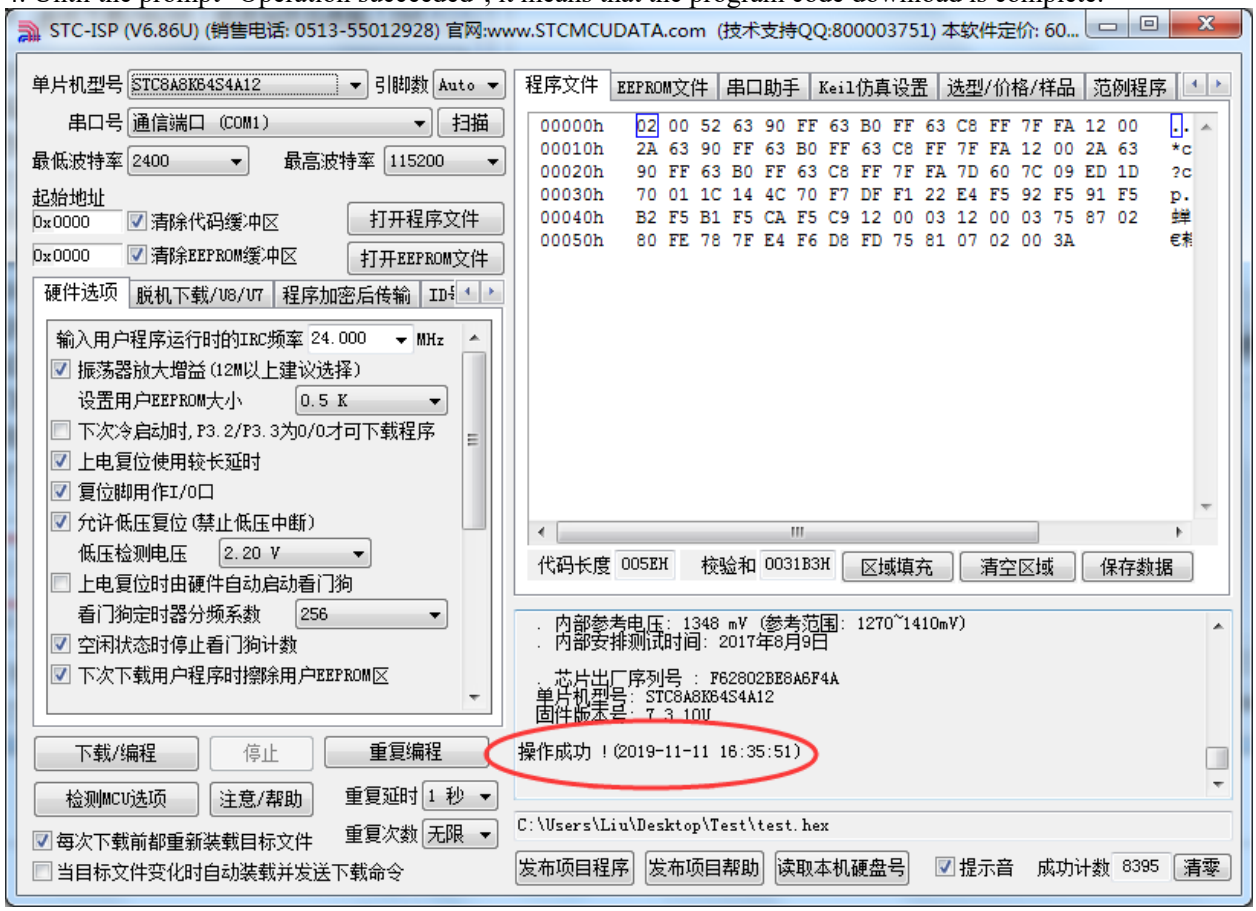
2. Open the user code program.



3. Click the "Download / Program" button to start downloading the user code.

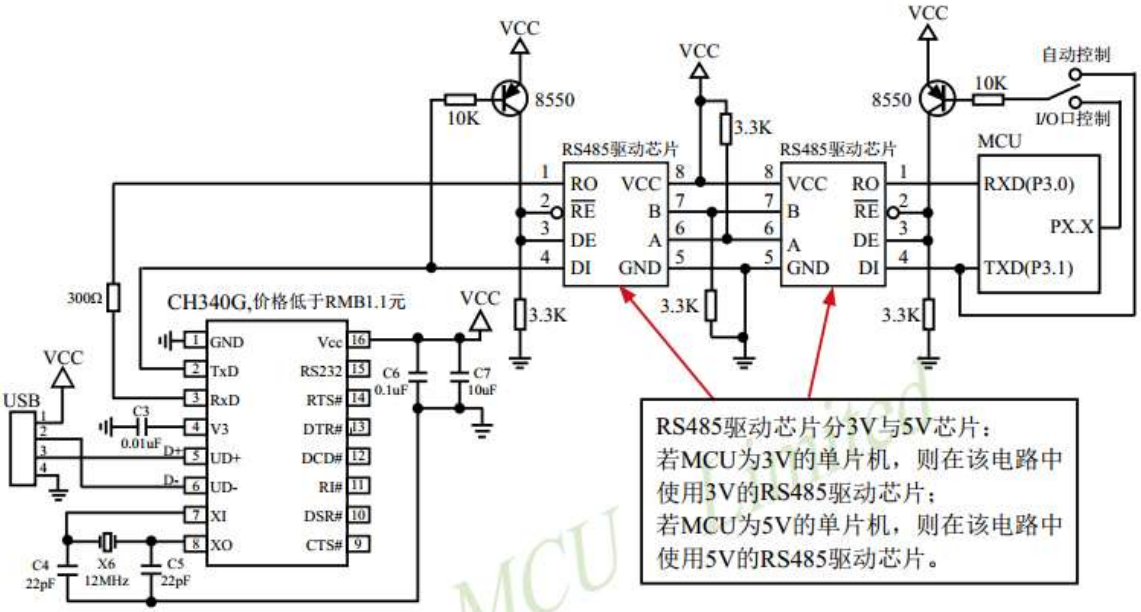


4. Until the prompt "Operation succeeded", it means that the program code download is complete.

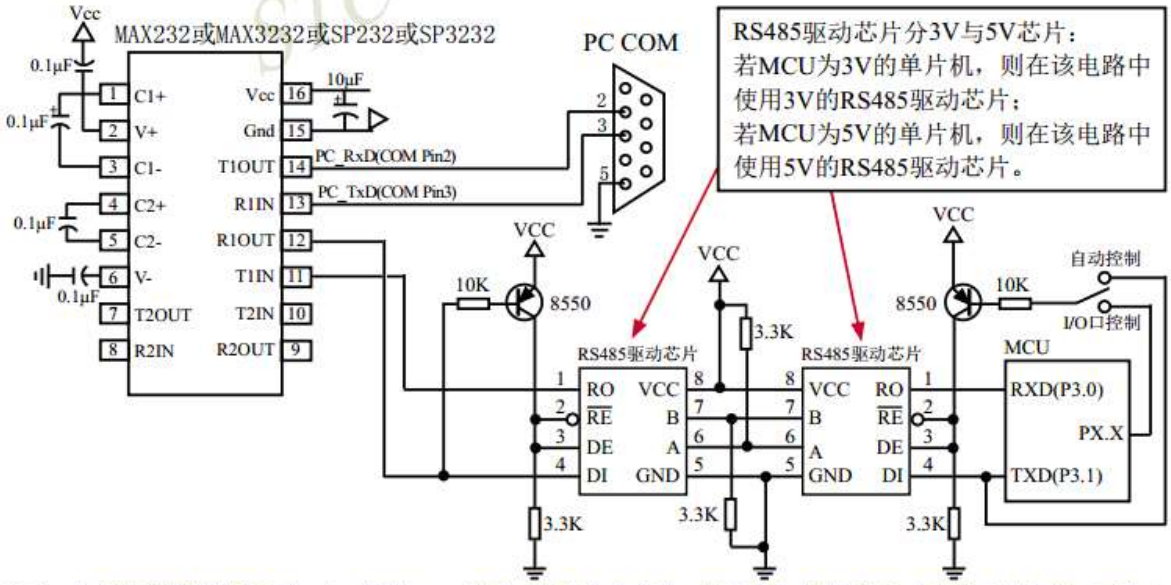


Appendix F RS485 Automatic control or I/O port control circuit diagram

1. Use the USB to serial port to connect the computer's RS485 to control the download circuit diagram (automatic control or I/O port control)



2. Use RS232 to serial port to connect the computer's RS485 to control the download circuit diagram (automatic control or I/O port control)



注意：如果要设置单片机某个I/O口控制RS485发送或接收命令有效，则必须将单片机焊入电路板之前先用U8下载工具结合电脑ISP软件对该单片机进行“RS485控制”设置并烧录一下（如上节所述），否则将单片机实现不了RS485控制功能。

建议用户将本节所述“RS485控制下载线路图(自动控制或I/O口控制)”设计到您的用户板上

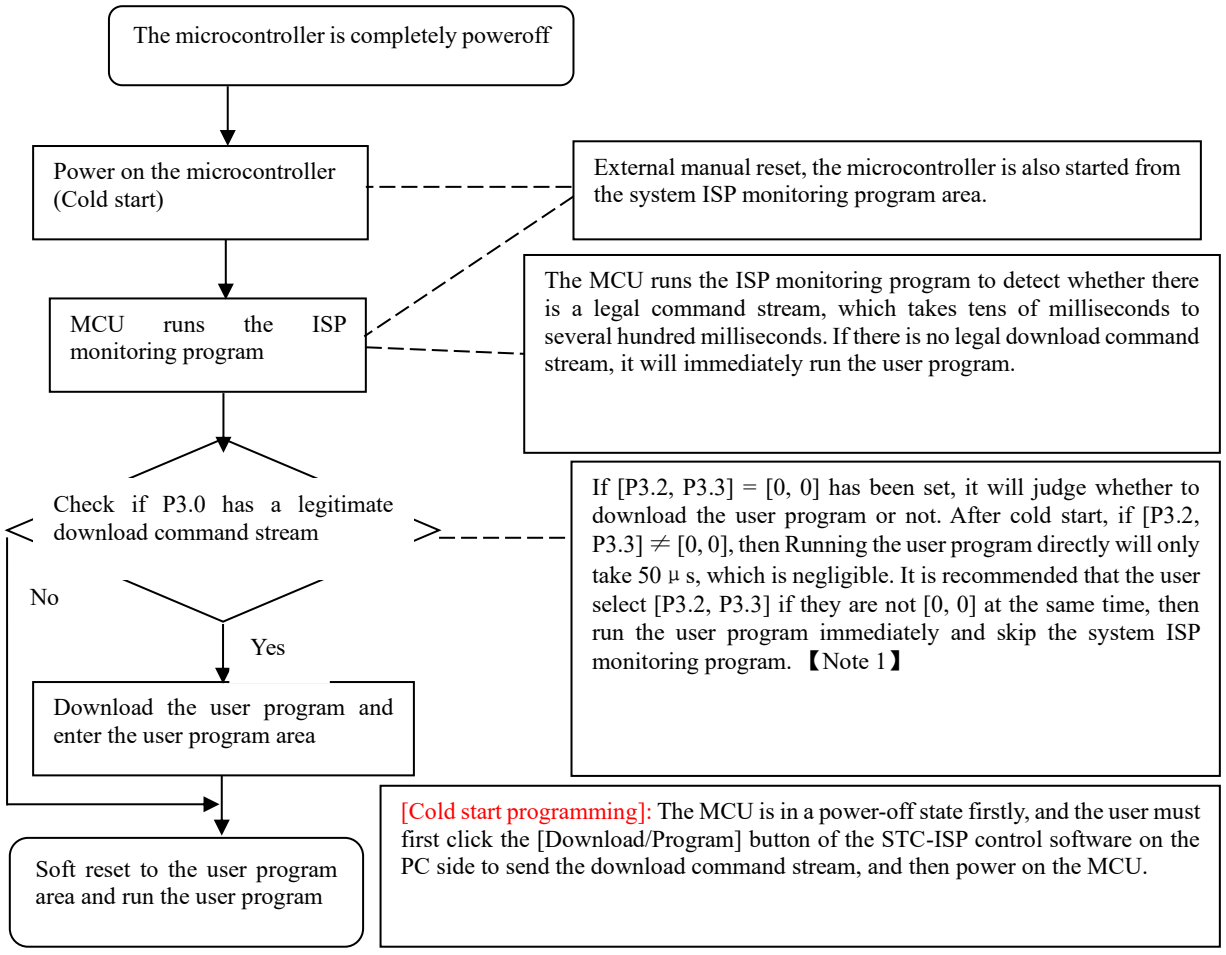
Appendix G STC tool instruction manual

G.1 Overview

U8W/U8W-Mini is a series of programming tools that integrates online download and offline download. STC Universal USB to Serial Port Tool is a programming tool that supports online download and online simulation.

Tool type	Online Download	Offline download	Burner download	Online simulation	Price(CNY)
U8W	support	support	support	Need to set pass-through mode	100 yuan
U8W-Mini	support	support	not support	Need to set pass-through mode	50 yuan
Universal USB to serial port	support	not support	not support	support	30 yuan

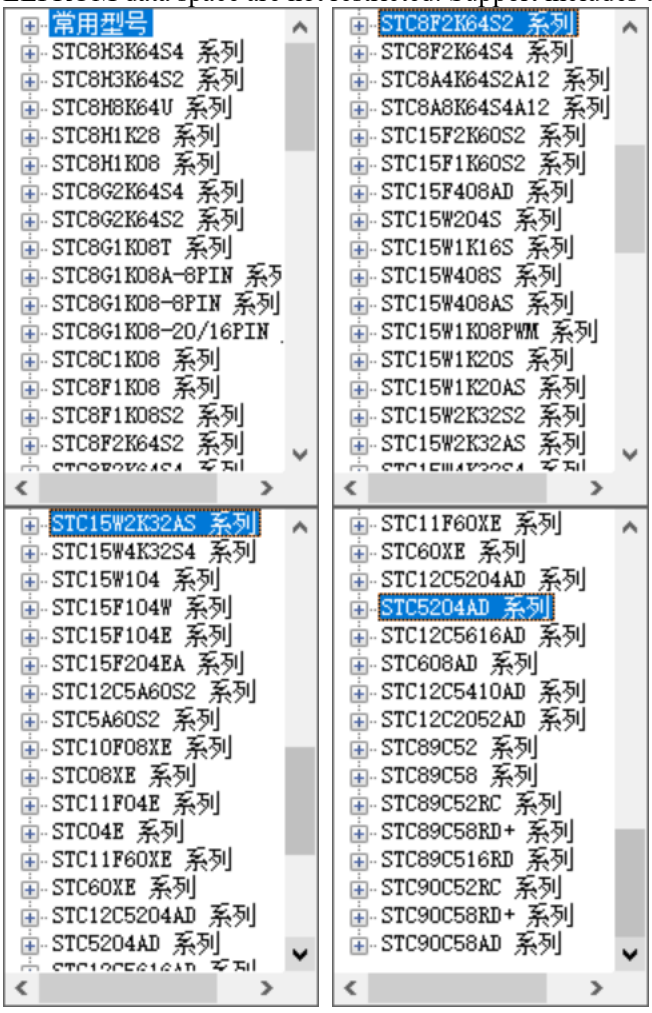
G.2 System Programmable (ISP) Process Description



Note: Because [P3.0, P3.1] is used for download/simulation (download/simulation interface is only available [P3.0, P3.1]), it is recommended that users put serial port 1 on P3.6/P3.7 Or P1.6/P1.7, if the user does not want to switch and insists on using P3.0/P3.1 to work or communicate as the serial port 1, be sure to check the "Next cold start, when downloading the program, The program can be downloaded only when P3.2/P3.3 is 0/0". 【Note 1】
[Note 1]: The programming protection pins of STC15, STC8 and later new chips are P3.2/P3.3, and the programming protection pins of earlier chips are P1.0/P1.1.

G.3 USB type online/offline download tool U8W/U8W-Mini

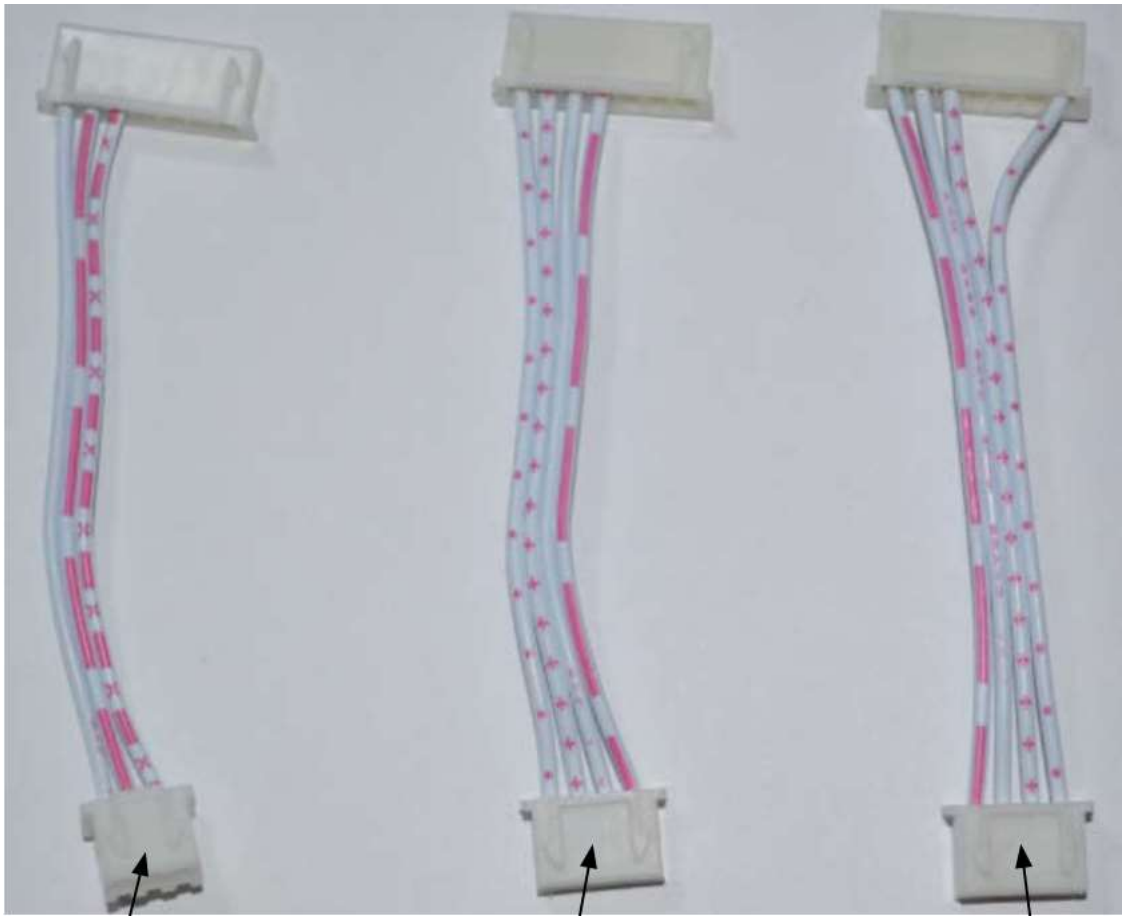
The application range of U8W/U8W-Min can support all current MCU series of STC, and the Flash program space and EEPROM data space are not restricted. Support includes the following and upcoming STC full series chips:



The offline download tool can be used for downloading without the computer, and can be used for mass production and remote upgrades. The offline download board can support multiple functions such as automatic increment, download limit, and encrypted transmission of user programs.
The following picture shows the front and back views of U8W tools and the front and back views of U8W-Mini:

In addition, some wires and tools are used together as follows, such as:
(1) Two-end male USB cable (shown on the left in the figure below) and USB-Micro cable (shown on the right in the figure below):

Note: This USB cable is a USB enhanced cable specially customized by our company, which can ensure that the download can be successful when directly powered by USB. On the market, some relatively low-quality two-end male USB cables have too much internal resistance and cause a large voltage drop (for example, the voltage of the USB when it is empty is about 5.0V. When using a low-quality USB cable to connect U8W/U8W-Mini/U8 /U8-Mini, the voltage to our download board may drop to 4.2V or lower, causing the chip to be in a reset state and fail to download successfully).
(2) The download cable connecting U8W/U8W-Mini to the user system (ie the connecting cable between U8W/U8W-Mini and the target MCU on the user board), such as
As shown below:



U8W/U8W-Mini and User systems are independent Power supply cable

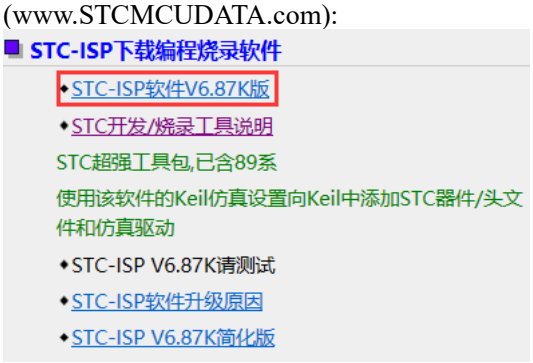
U8W/U8W-Mini to The user's system power supply wiring

User system gives U8W/U8W-Mini Power supply cable

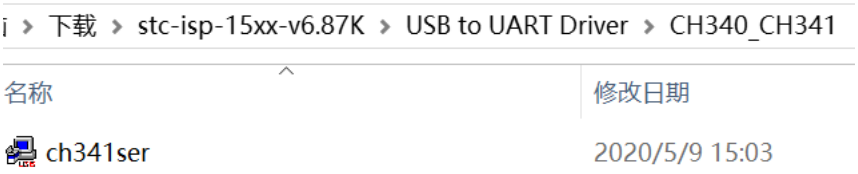
G.3.1 Install U8W/U8W-Mini driver

The U8W/U8W-Mini download board uses a CH340 USB-to-serial universal chip. This saves the trouble that some computers without a serial port must buy a USB to serial port tool to download. But CH340 is the same as other USB-to-serial tools, the driver must be installed before use.

Obtain the driver by downloading the STC-ISP software package
The following is the download location of the STC-ISP software package provided on the STC official website (www.STCMCUDATA.com):



After downloading, decompress, the path of CH340 driver installation package is stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341:

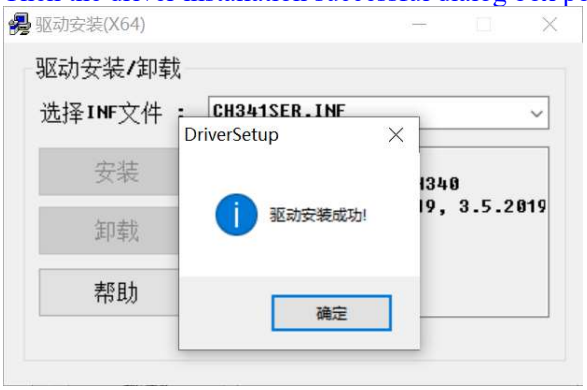


Download the driver manually through STC's official website or in the latest STC-ISP download software
Download the driver manually on the official website of STC or in the latest STC-ISP download software. The download link of the driver is: U8 programmer USB to serial port driver (http://www.stcmcu.com/STCISP/CH341SER.exe). The driver address on the website and STC-ISP download software is shown in the figure below:

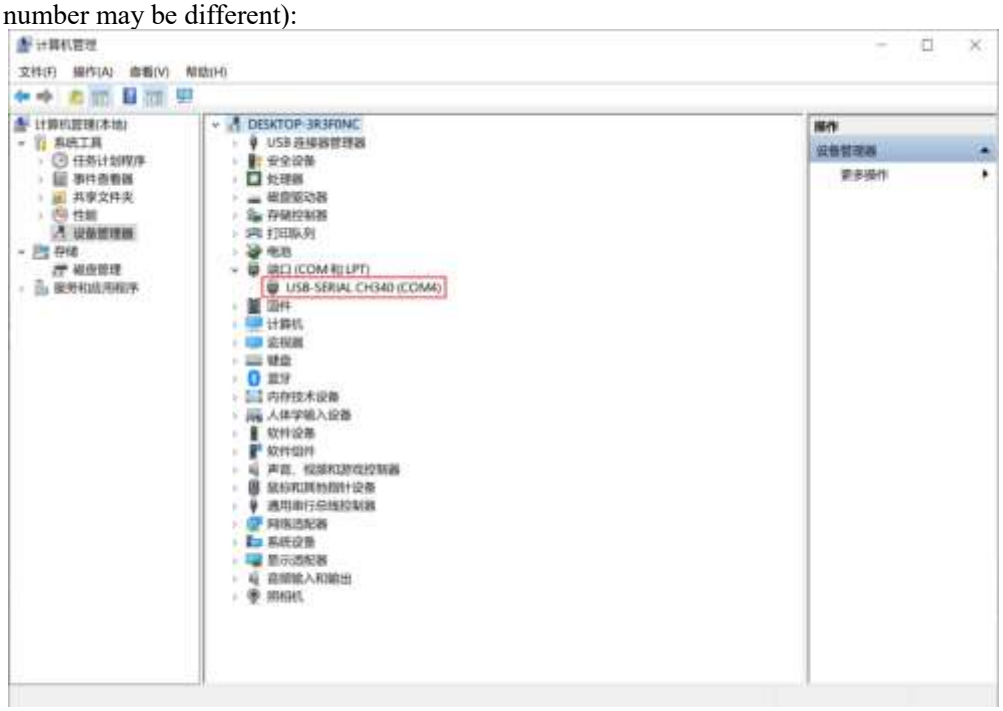
Install U8W/U8W-Mini driver

After the driver is downloaded to the machine, double-click the executable program and run it. The interface shown in the figure below appears, click the "Install" button to start the automatic driver installation:

Then the driver installation successful dialog box pops up, click the "OK" button to complete the installation:



Then use the USB cable provided by STC to connect the U8W/U8W-Mini download board to the computer, open the device manager of the computer, and under the port device category, if there is a device similar to "USB-SERIAL CH340 (COMx)", it means U8W/U8W-Mini can be used normally. As shown in the figure below (different computers, the serial port number may be different):

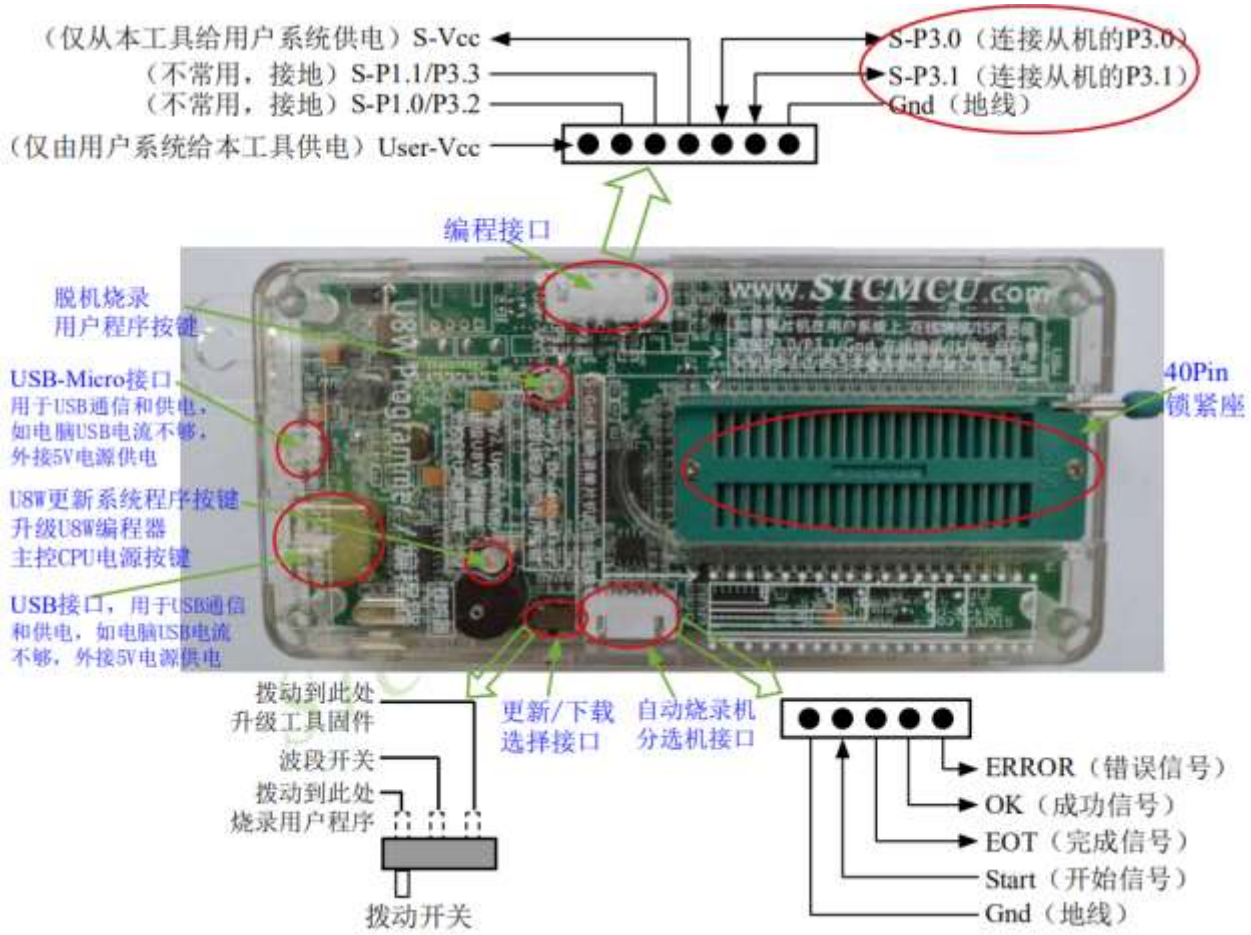


Note: When using STC-ISP to download software later, the selected serial port number must be the corresponding serial port number, as shown in the figure below:

G.3.2 U8W function introduction

The main interfaces and functions of U8W tools are described in detail below:

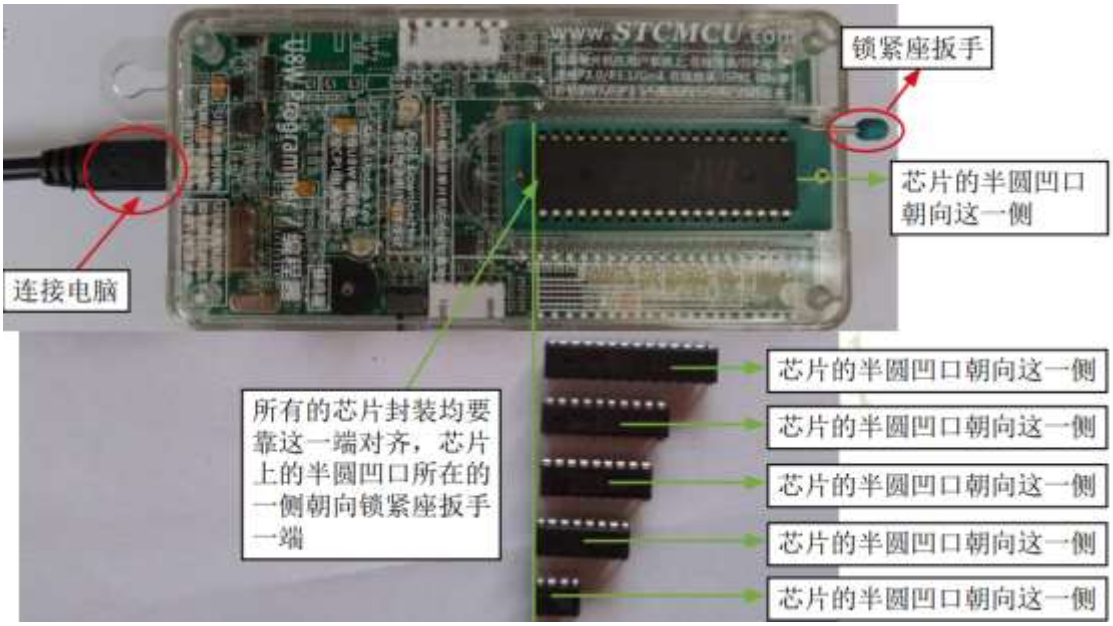
If the MCU is on the user system, P3.0/P3.1/Gnd must be connected during online programming/ISP, and P3.0/P3.1 of the target MCU should not be connected to any other lines during online programming/ISP 去



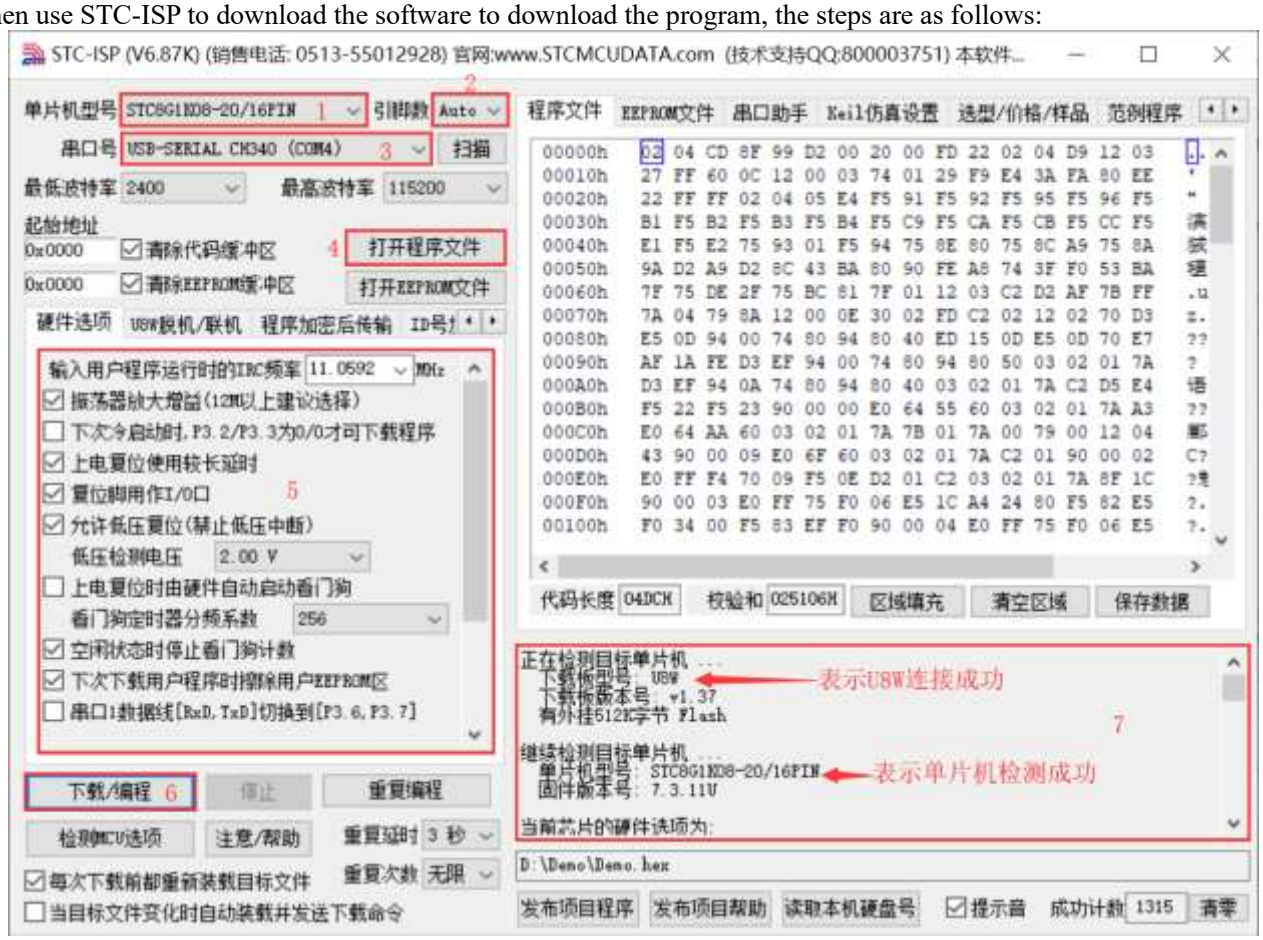
Programming interface: Use different download cables to connect the U8W download board and the user system according to different power supply methods.
U8W update system program button: used to update U8W tools. When there is a new version of U8W firmware, you need to press this button to update the U8W main control chip (note: you must first set the toggle switch on the update/download selection interface Toggle to upgrade tool firmware).
Offline download user program button: Start offline download button. First, the PC downloads the offline code to the U8W board, and then uses the download cable to connect the user system to the U8W, and then press this button to start the offline download (the user code will also start to download every time the power is turned on).
Update/download selection interface: When you need to upgrade the underlying firmware of U8W, you need to switch this toggle switch to the firmware upgrade tool. When you need to program the target chip through U8W, you need to turn the toggle switch to Burn the user program.
(Please refer to the figure above for the connection of the toggle switch)
Automatic burner/sorter interface: It is a control interface used to control the automatic burner/sorter for automatic production.

G.3.3 U8W online download instructions

The target chip is installed on the U8W locking base and connected to the computer by U8W for online download
First use the USB cable provided by STC to connect the U8W to the computer, and then install the target MCU on the U8W in the direction shown in the figure below:

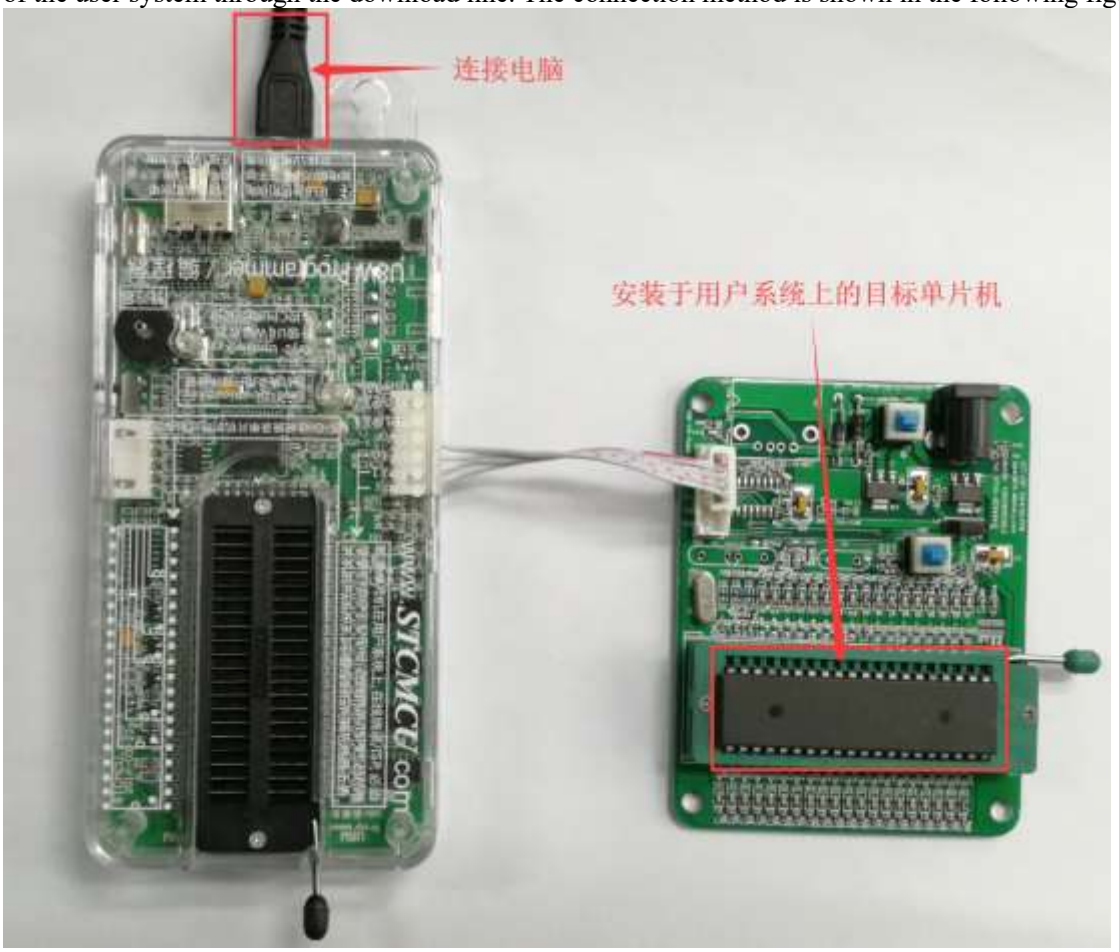


Then use STC-ISP to download the software to download the program, the steps are as follows:



1 Select the MCU model;
2 Select the number of pins. When the chip is directly installed on the U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3 Select the serial port number corresponding to U8W;
4 Open the target file (HEX format or BIN format);
5 Set the hardware options;
6 Click the "Download/Program" button to start burning;
7The step information of the programming process is displayed, and the message "Operation successful!" is displayed when the programming is completed.

When there is the version number information of the output download board and the corresponding information of the external Flash in the information box, it means that the U8W download tool has been correctly detected.
During the downloading process, the 4 LEDs on the U8W download tool will be displayed in marquee mode. After the download is complete, if the download is successful, All 4 LEDs will be on and off at the same time; if the download fails, all 4 LEDs will be off.
It is recommended that users use the latest version of STC-ISP to download the software (please pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com).
The target chip is connected to U8W through the user system leads and U8W is connected to the computer for online download
Firstly, use the USB cable provided by STC to connect U8W to the computer, and then connect U8W to the target MCU of the user system through the download line. The connection method is shown in the following figure:



Then use STC-ISP to download the software to download the program, the steps are as follows:
1. Select the MCU model;
2. Select the serial port number corresponding to U8W;
3. Open the target file (HEX format or BIN format);
4. Set hardware options;
5. Click the "Download/Program" button to start burning;
6. The step information of the programming process is displayed, and the message "Operation successful!" is displayed when the programming is completed.

When there is the version number information of the output download board and the corresponding information of the external Flash in the information box, it means that the U8W download tool has been correctly detected.

During the download process, the 4 LEDs on the U8W download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com).

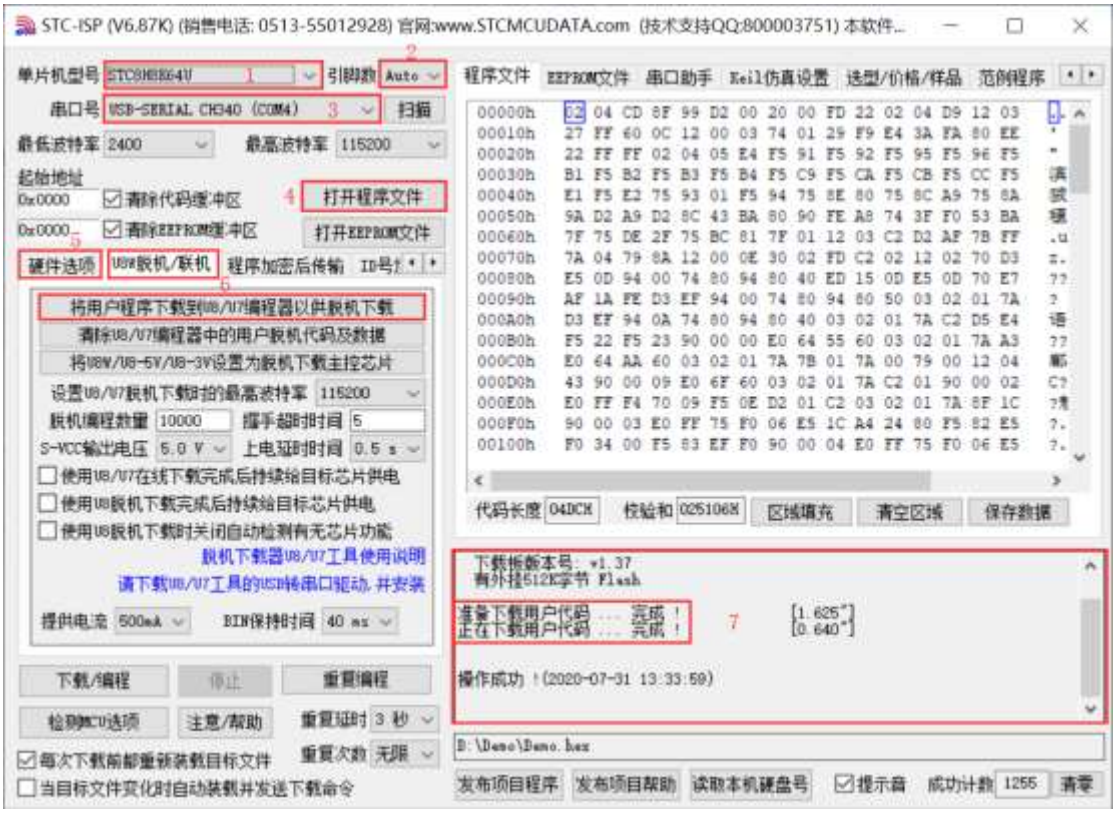
G.3.4 U8W offline download instructions

The target chip is installed on the U8W socket and locked and connected to the computer via USB to power the U8W for offline download

The steps to use USB to power U8W for offline download are as follows:

(1) Use the USB cable provided by STC to connect the U8W download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

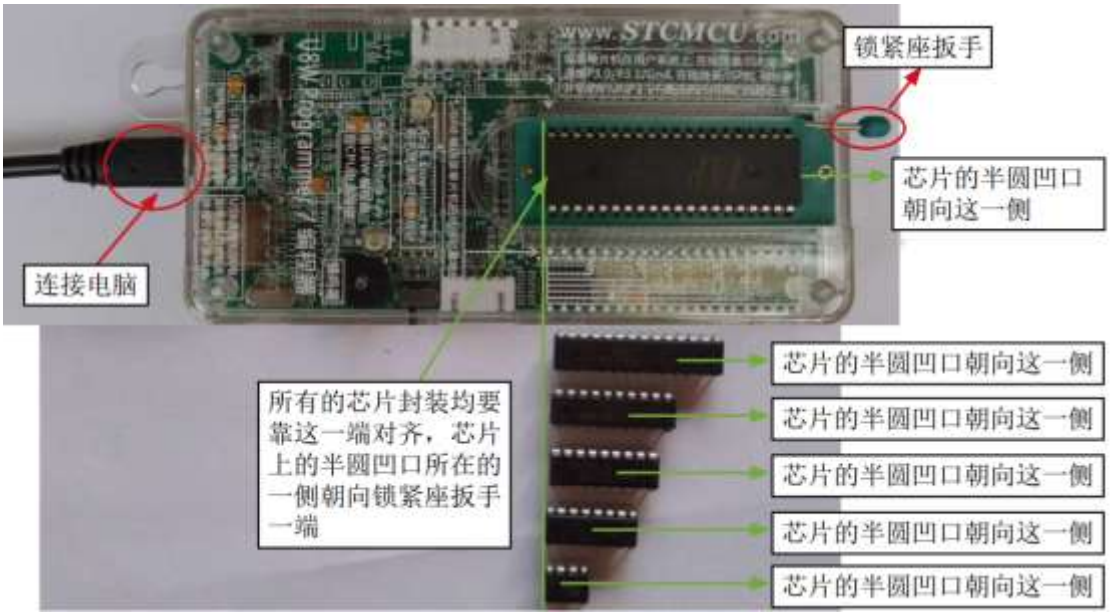


1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, pay attention to the S-VCC output voltage matches the target chip working voltage; click the "Download user program to U8/U7 programmer for offline download" button ;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

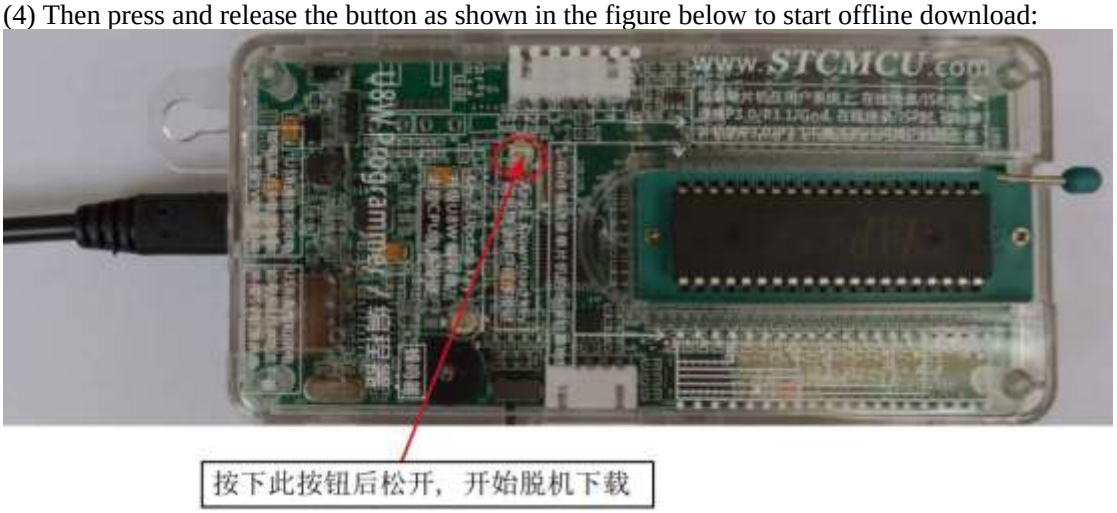
According to the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W download tool.

It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com).

(3) Place the target MCU in the U8W download tool in the direction shown in the figure below, as shown in the figure below:



(4) Then press and release the button as shown in the figure below to start offline download:



During the downloading process, the 4 LEDs on the U8W download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

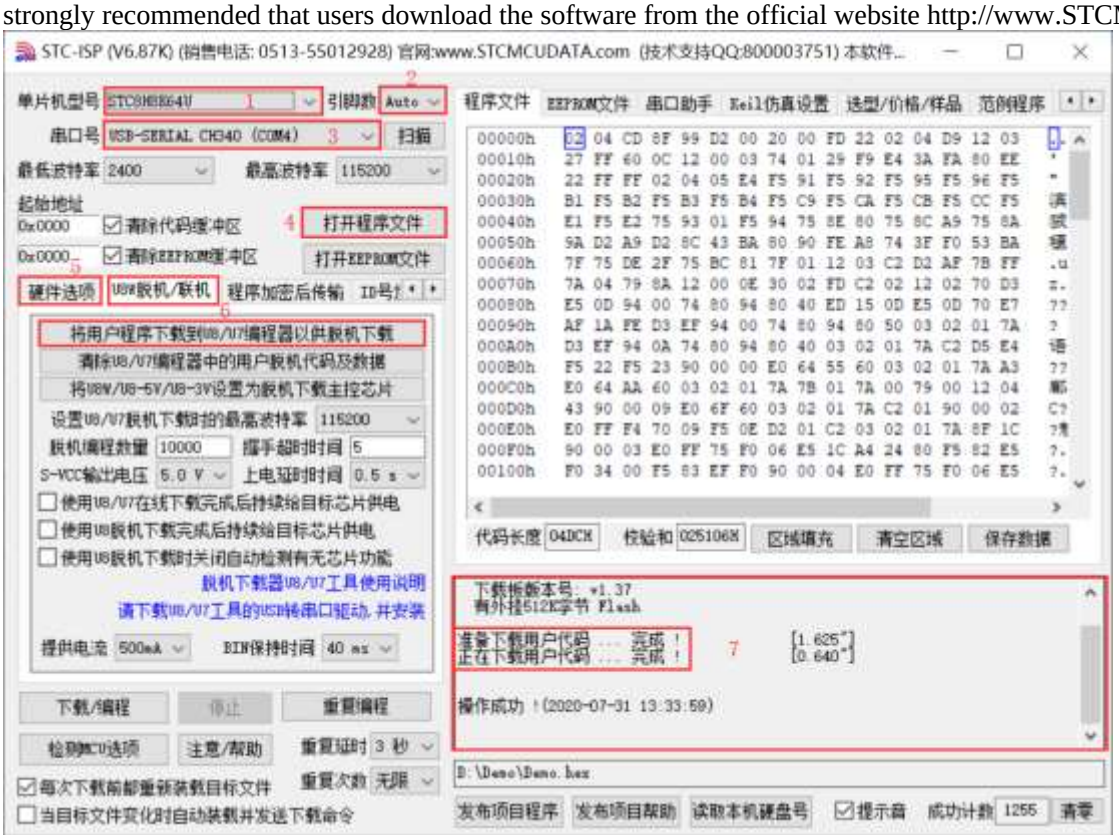
Offline download plug and play burning function introduction:
1. After completing the above steps (1) and (2), U8W is in the plug-and-play programming state by default when it is connected to the computer and powered on;
2. Put the chip into the programming socket according to the instructions in step (3). While tightening the socket wrench, U8W will automatically start programming;
3. Display the burning process and burning result through the indicator light;
4. After programming is completed, loosen the wrench and take out the chip;
5. Repeat steps 2, 3 and 4 for continuous programming, eliminating the need to press the button to trigger the programming action.

The target chip is connected to U8W by the user system lead and connected to the computer via USB to supply power to U8W for offline download.

The steps for offline download using USB to supply power to U8W are as follows:
(1) Use the USB cable provided by STC to connect the U8W download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

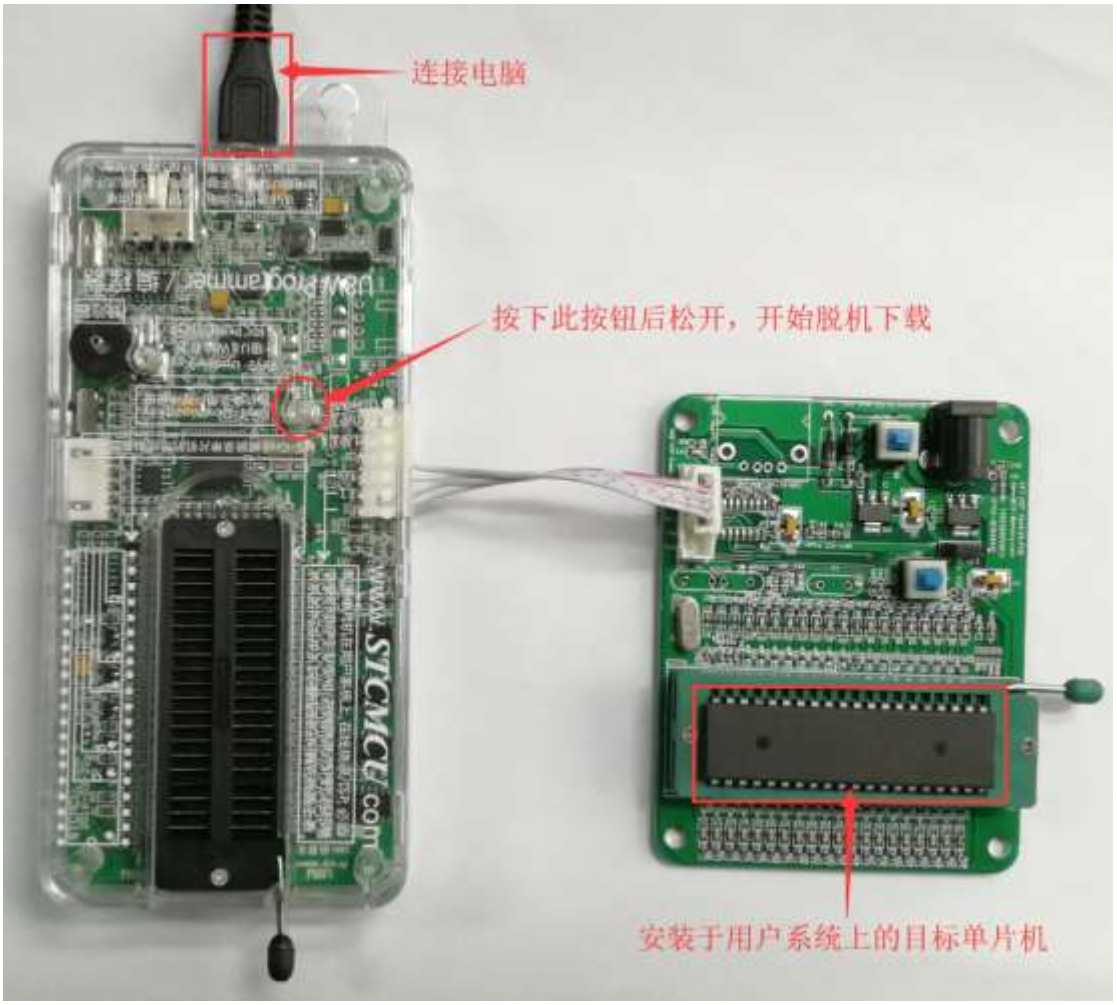
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com).



1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, pay attention to the S-VCC output voltage matches the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

According to the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W download tool.

(3) Then use the cable to connect the computer, connect the U8W download tool and the user system (target MCU) as shown in the figure below, and press the button shown in the figure and release it to start offline downloading:



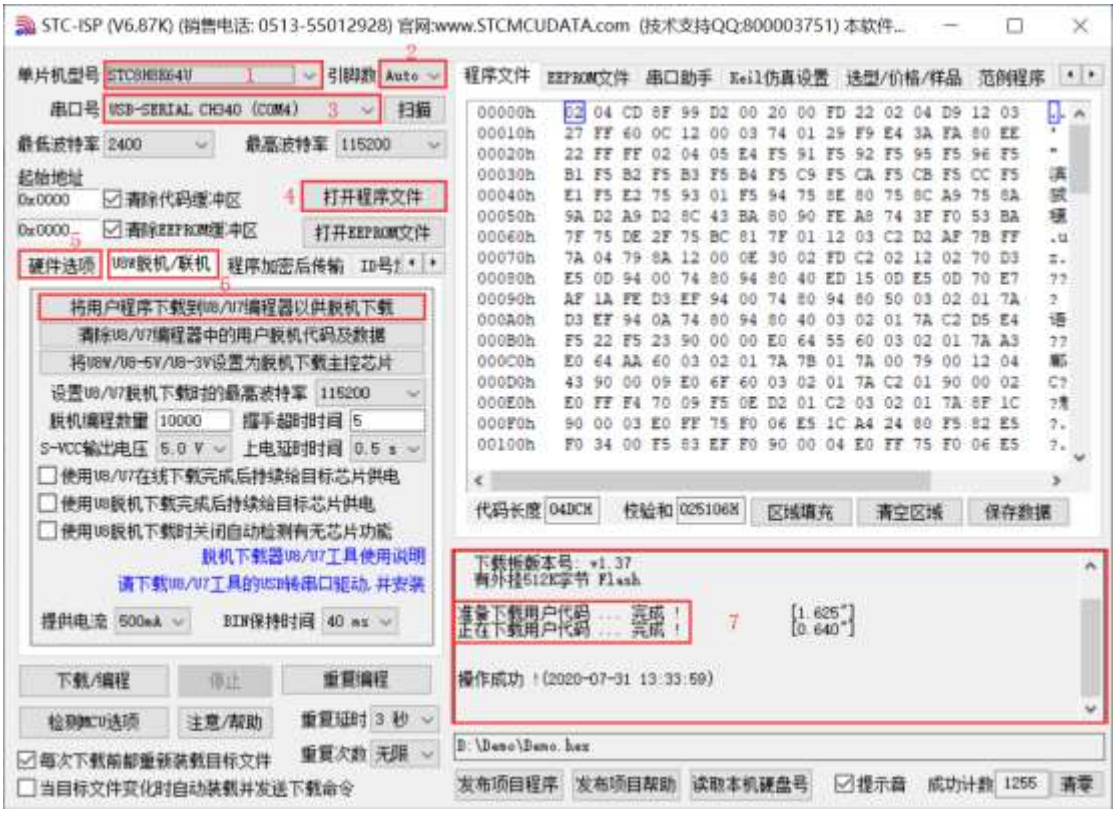
During the downloading process, the 4 LEDs on the U8W download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

The target chip is connected to U8W by the user system lead, and the user system supplies power to U8W for offline download

(1) Firstly, use the USB cable provided by STC to connect the U8W download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

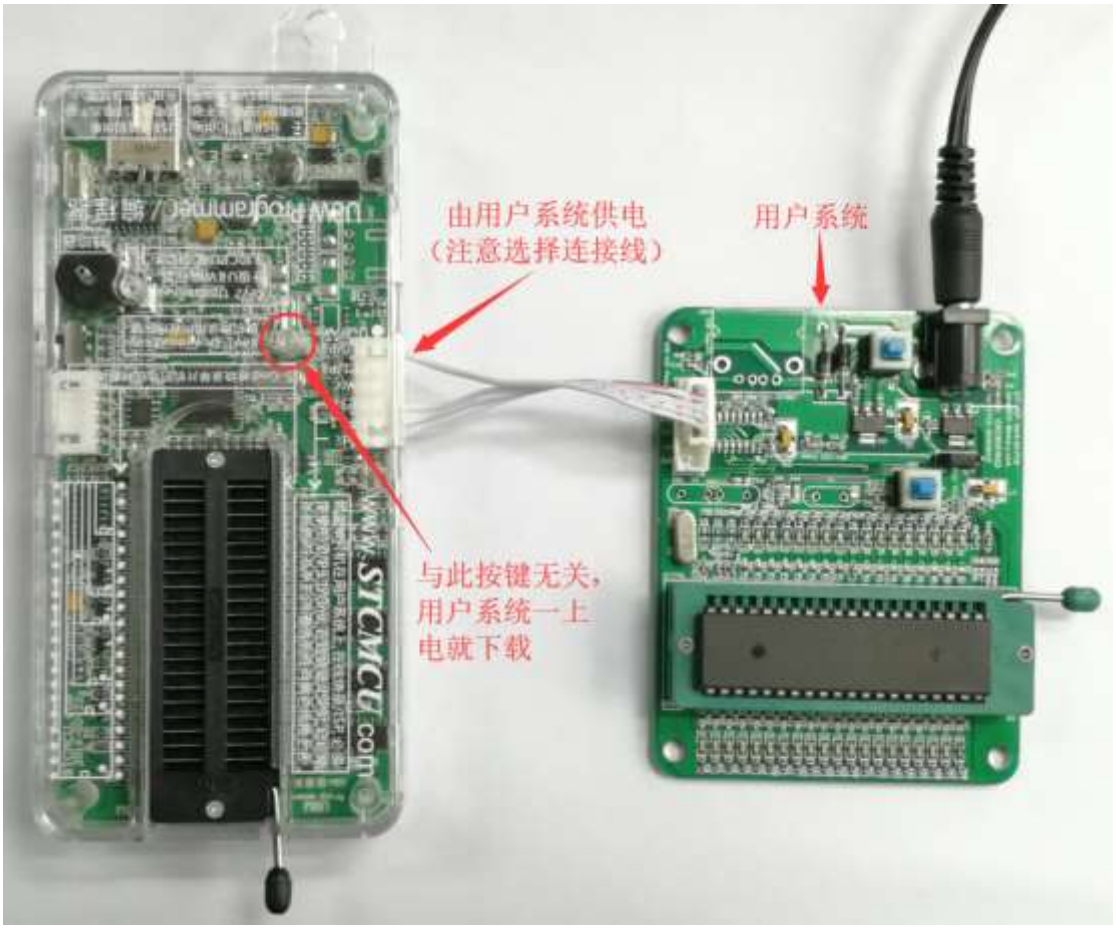
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com).



1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, pay attention to the S-VCC output voltage matches the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

According to the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W download tool.

(3) Then connect U8W to the user system as shown in the figure below, supply power to the user system, and then start offline downloading:



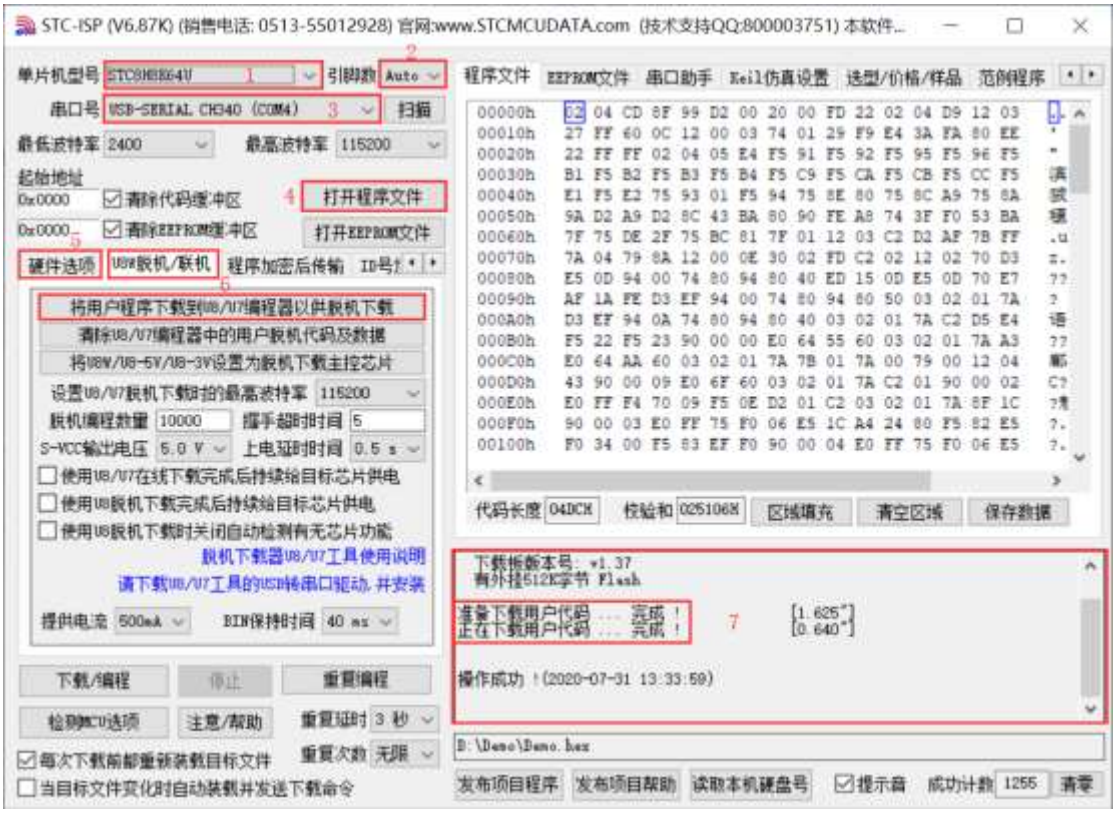
During the downloading process, the 4 LEDs on the U8W download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

The target chip is connected to U8W by the user system lead, and U8W and the user system are independently powered for offline download

(1) Firstly, use the USB cable provided by STC to connect the U8W download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

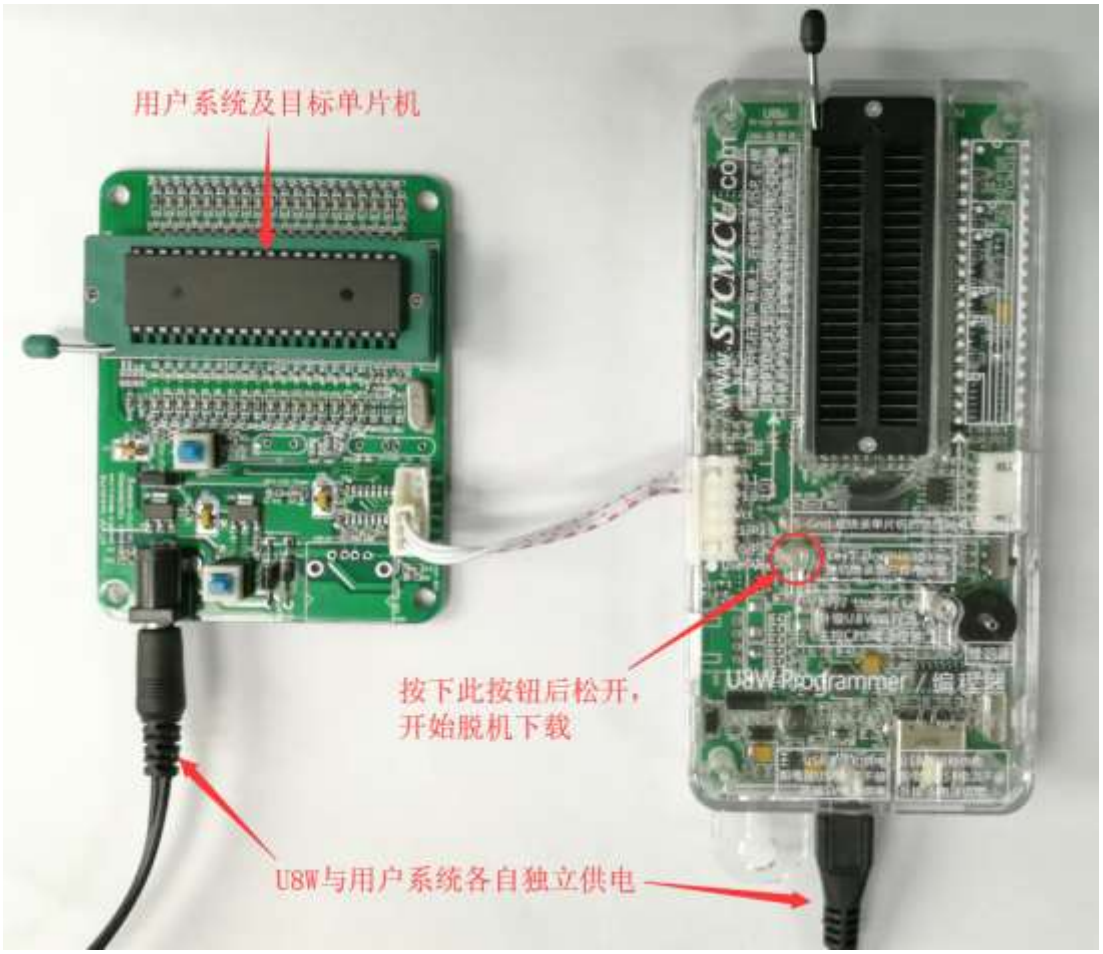
It is recommended that users use the latest version of STC-ISP to download the software (please always pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com).



1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, pay attention to the S-VCC output voltage matches the target chip operating voltage;
Click the "Download user program to U8/U7 programmer for offline download" button;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

According to the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W download tool.

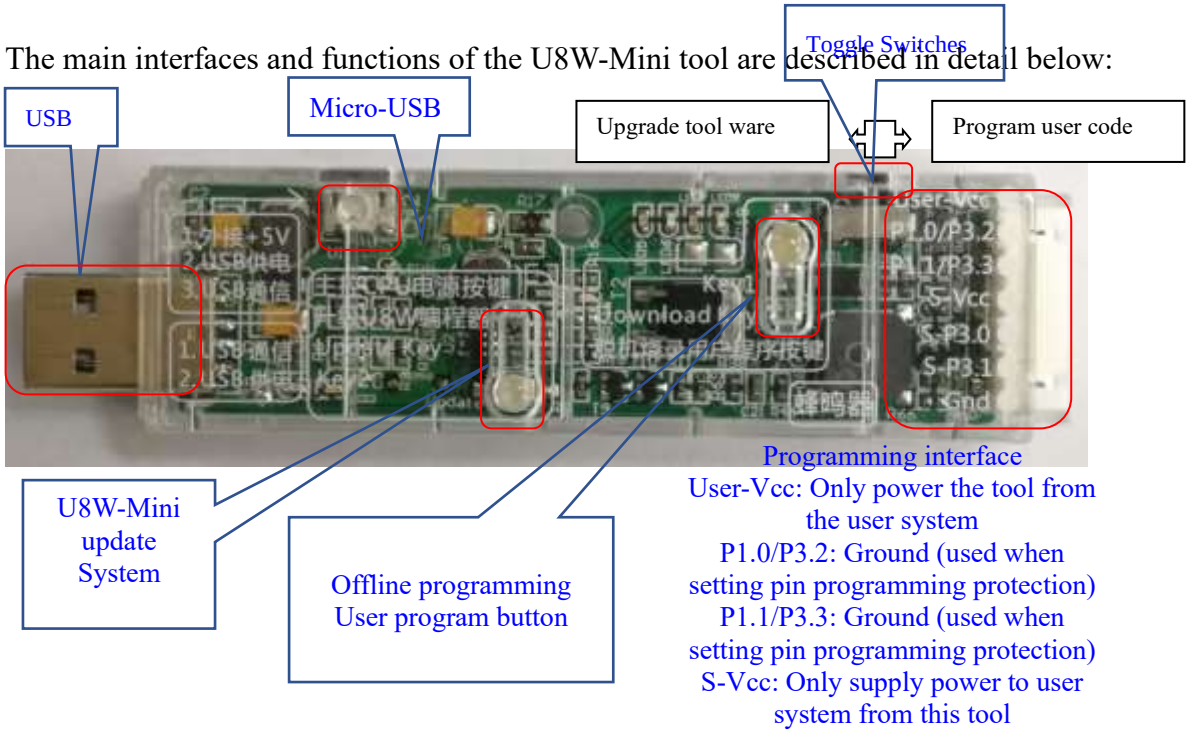
(3) Then connect U8W to the user system as shown in the figure below, and press the button shown in the figure first and then release it, ready to start offline download, and finally power on/on the user system to download the user program Officially begin:



During the downloading process, the 4 LEDs on the U8W download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

G.3.5 U8W-Mini's function introduction

The main interfaces and functions of the U8W-Mini tool are described in detail below:

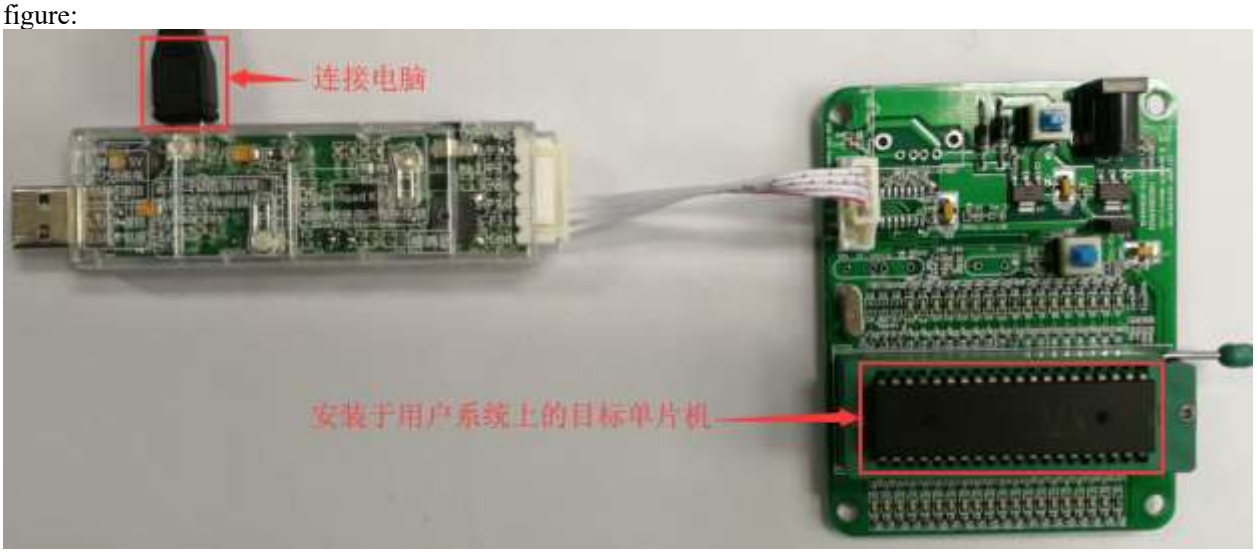


Programming interface: According to different power supply methods, use different download cables to connect the U8W-Mini download board and the user system.
U8W-Mini update system program button: used to update U8W-Mini tools. When there is a new version of U8W firmware, you need to press this button to update the U8W-Mini main control chip (note: you must first select update/download The toggle switch on the interface is moved to the upgrade tool firmware).
Offline download user program button: Start offline download button. First, the PC downloads the offline code to the U8W-Mini, and then uses the download cable to connect the user system to the U8W-Mini, and then press this button to start the offline download (the user will also start downloading immediately every time the power is turned on Code).
Update/download selection interface: When you need to upgrade the underlying firmware of U8W-Mini, you need to move this toggle switch to the firmware of the upgrade tool. When you need to program the target chip through U8W-Mini, you need to Toggle the switch to burn the user program. (Please refer to the figure above for the connection of the toggle switch)
USB interface: The USB interface has the same function as the Micro-USB interface. Users can connect one of them to the computer as needed.

G.3.6 U8W-Mini online download instructions

The target chip is connected to the U8W-Mini through the user system lead, and the U8W-Mini is connected to the computer for online download

Firstly, use the USB cable provided by STC to connect the U8W-Mini to the computer, and then connect the U8W-Mini to the target MCU of the user system through the download cable. The connection method is shown in the following figure:



Then use STC-ISP to download the software to download the program, the steps are as follows:

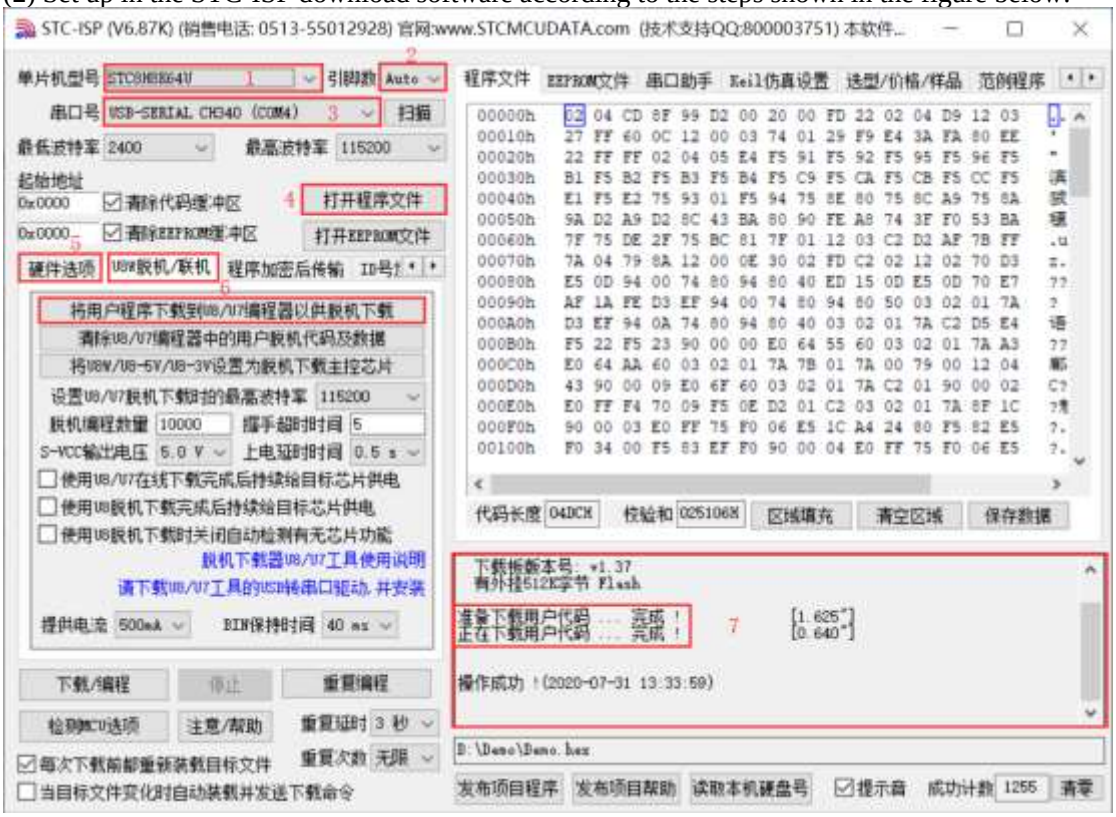
1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W-Mini to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W-Mini;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Click the "Download/Program" button to start burning;
7. The step information of the burning process is displayed, and the message "Operation successful!" is displayed when the burning is completed.

When there is the version number information of the output download board and the corresponding information of the external Flash in the information box, it means that the U8W-Mini download tool has been correctly detected.
During the downloading process, the 4 LEDs on the U8W-Mini download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.
It is recommended that users use the latest version of STC-ISP to download the software (please pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software on the official website http://www.STCMCUDATA.com).

G.3.7 U8W-Mini offline download instructions

The target chip is connected to the U8W-Mini by the user system lead and connected to the computer via USB to power the U8W-Mini for offline download.
The steps for offline download using USB to power the U8W-Mini are as follows:
(1) Use the USB cable provided by STC to connect the U8W-Mini download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:



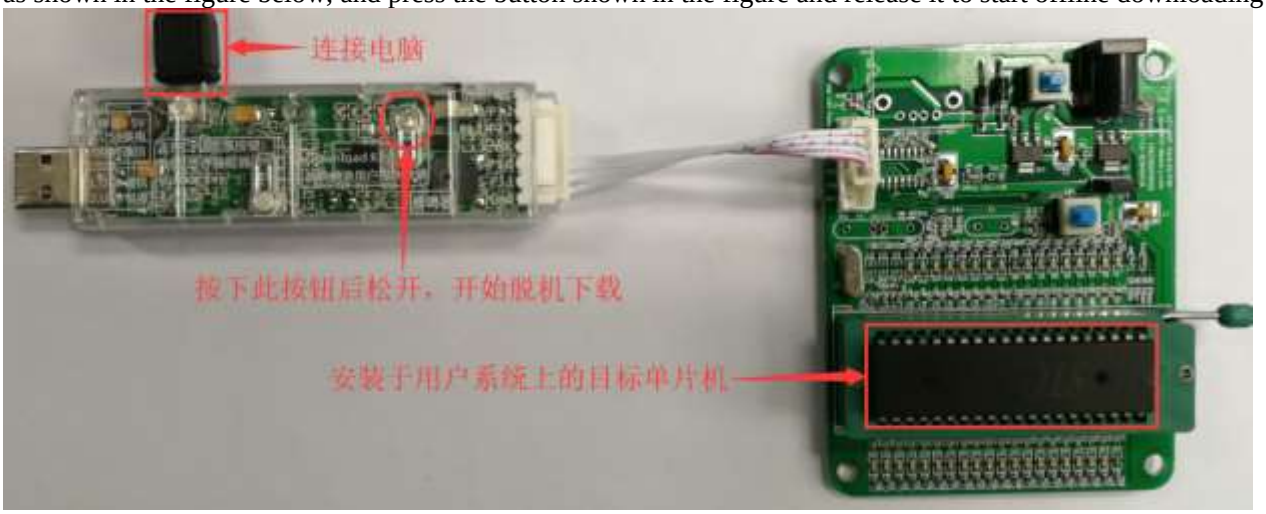
1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W-Mini to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W-Mini;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, pay attention to the S-VCC output voltage matches the target chip working voltage; click the "Download user program to U8/U7 programmer for offline download" button ;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W-Mini download tool.

It is recommended that users use the latest version of STC-ISP to download the software (please pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com).

(3) Then use the cable to connect the computer, connect the U8W-Mini download tool and the user system (target MCU)

as shown in the figure below, and press the button shown in the figure and release it to start offline downloading :

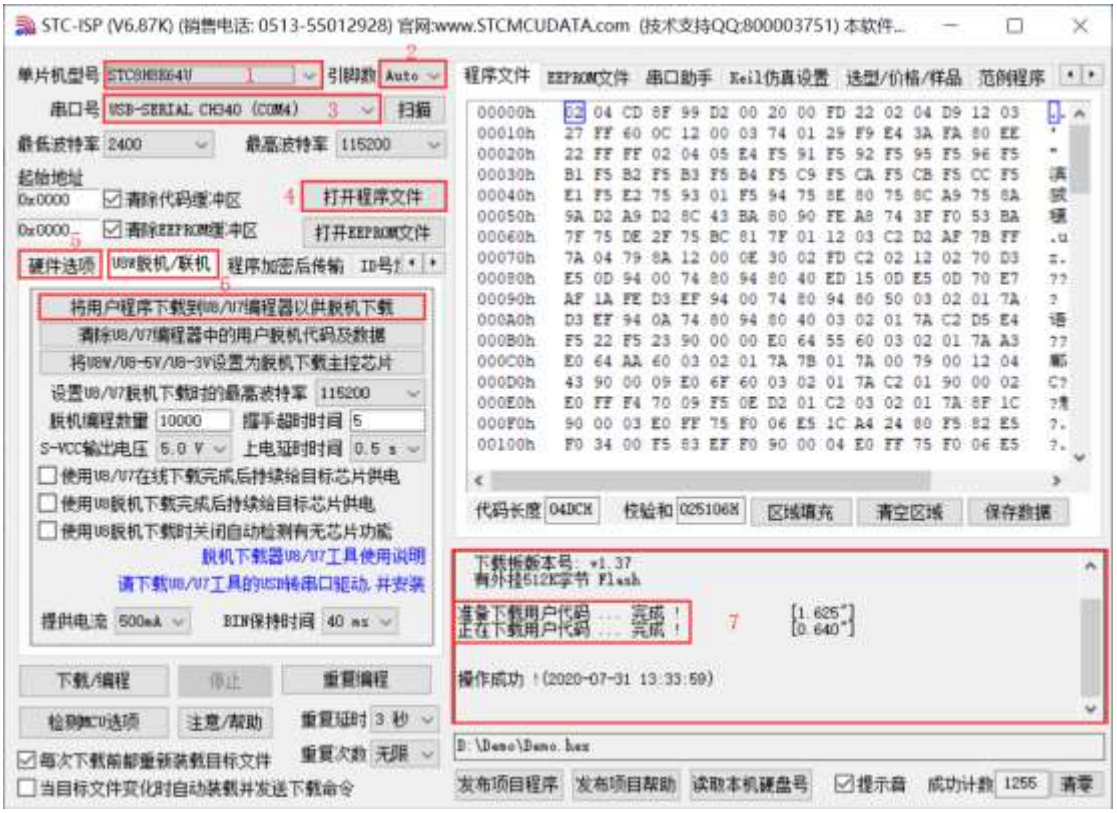


During the downloading process, the 4 LEDs on the U8W-Mini download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

The target chip is connected to the U8W-Mini by the user system lead, and the U8W-Mini is powered by the user system for offline download.

(1) Firstly, use the USB cable provided by STC to connect the U8W-Mini download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

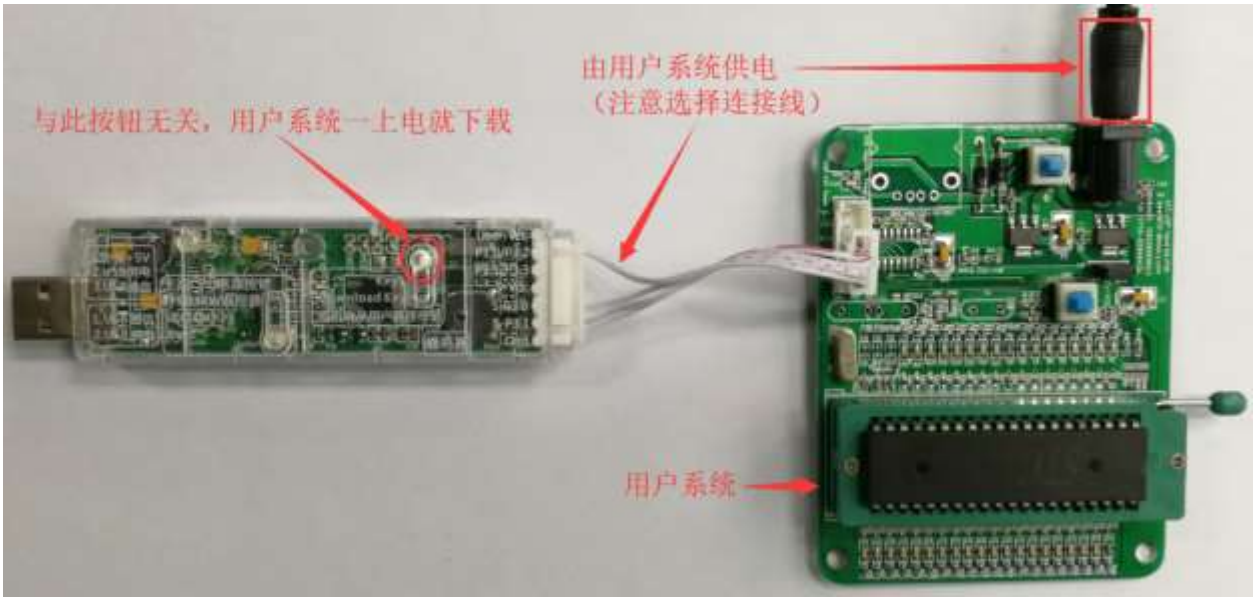


1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W-Mini to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W-Mini;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, pay attention to the S-VCC output voltage matches the target chip working voltage; click the "Download user program to U8/U7 programmer for offline download" button ;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W-Mini download tool.

It is recommended that users use the latest version of STC-ISP to download the software (please pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com)
Software used).

(3) Then connect U8W-Mini to the user system as shown in the figure below. Once the user system is powered on, it will start offline downloading:

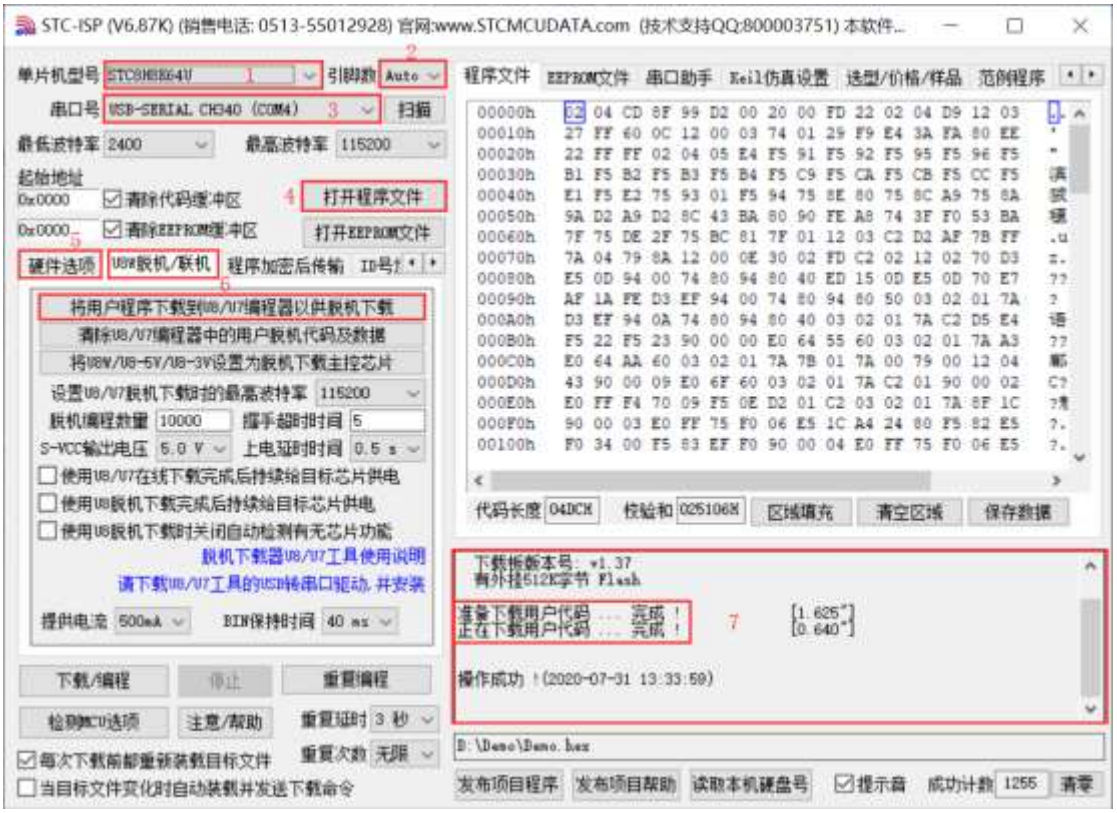


During the downloading process, the 4 LEDs on the U8W-Mini download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

The target chip is connected to U8W-Mini by the user system lead, and U8W-Mini and the user system are independently powered for offline download.

(1) Firstly, use the USB cable provided by STC to connect the U8W-Mini download board to the computer, as shown below:

(2) Set up in the STC-ISP download software according to the steps shown in the figure below:

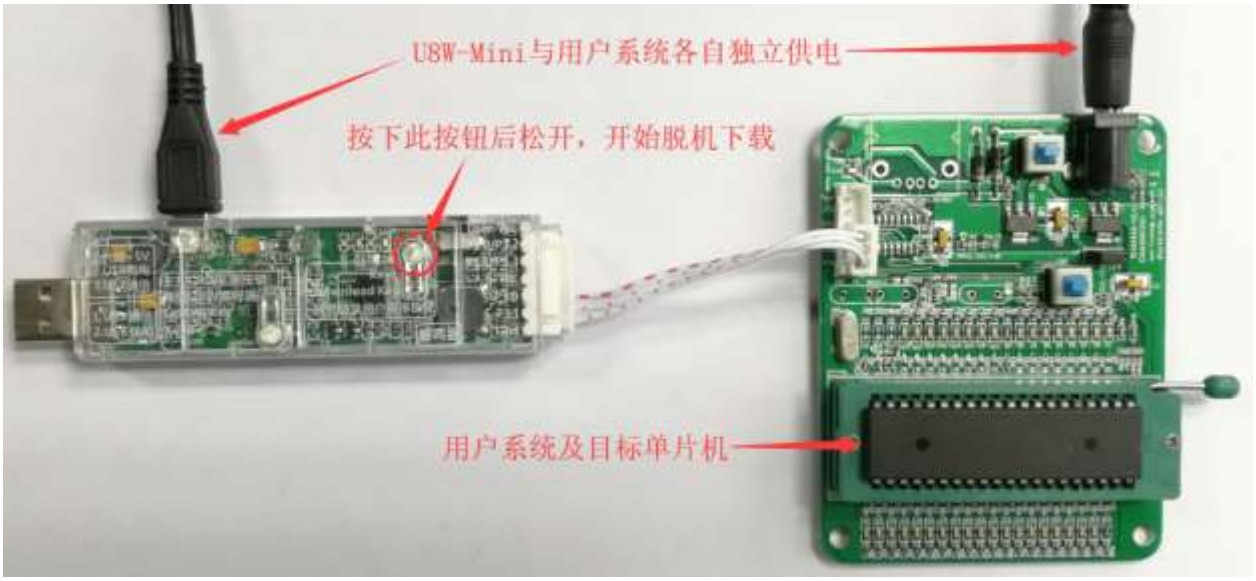


1. Select the MCU model;
2. Select the number of pins. When the chip is directly installed on the U8W-Mini to download, be sure to select the correct number of pins, otherwise the download will fail;
3. Select the serial port number corresponding to U8W-Mini;
4. Open the target file (HEX format or BIN format);
5. Set the hardware options;
6. Select the "U8W Offline/Online" tab, set the offline programming options, pay attention to the S-VCC output voltage matches the target chip working voltage; click the "Download user program to U8/U7 programmer for offline download" button ;
7. The step information of the setting process is displayed, and the prompt "Operation successful!" is displayed after the setting is completed.

Follow the steps in the above figure, after the operation is completed, if the download is successful, it means that the user code and related setting options have been downloaded to the U8W-Mini download tool.

It is recommended that users use the latest version of STC-ISP to download the software (please pay attention to the updates of the STC-ISP download software on the STC official website http://www.STCMCUDATA.com. It is strongly recommended that users download the software from the official website http://www.STCMCUDATA.com).

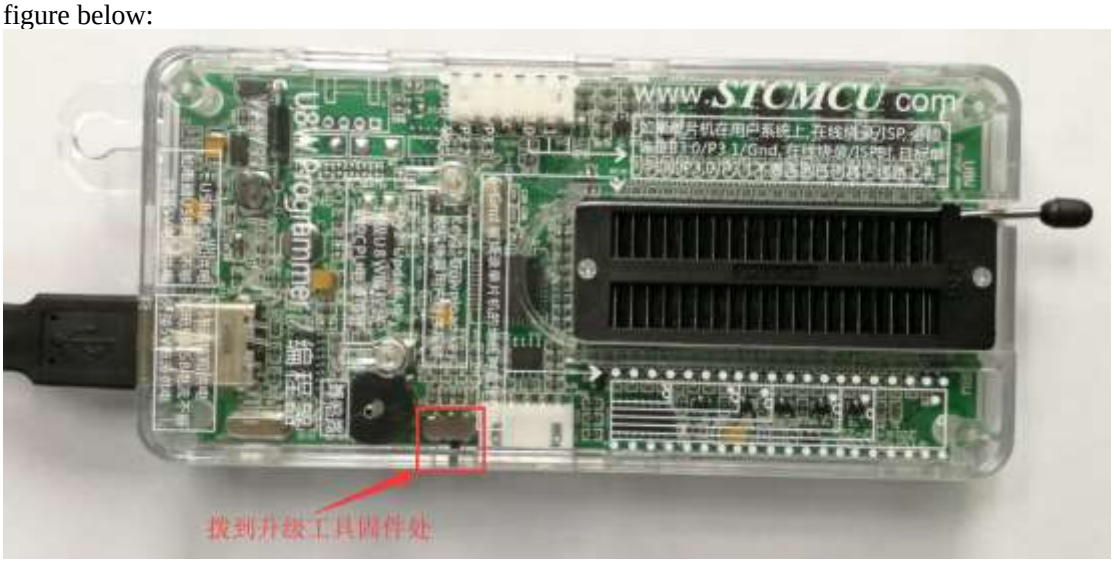
(3) Then connect U8W-Mini to the user system as shown in the figure below, and press the button shown in the figure first and then release it, ready to start offline download, and finally power on/on the user system to download The user program officially starts:



During the downloading process, the 4 LEDs on the U8W-Mini download tool will be displayed in marquee mode. After the download is complete, if the download is successful, the 4 LEDs will be on and off at the same time; if the download fails, all the 4 LEDs will be off.

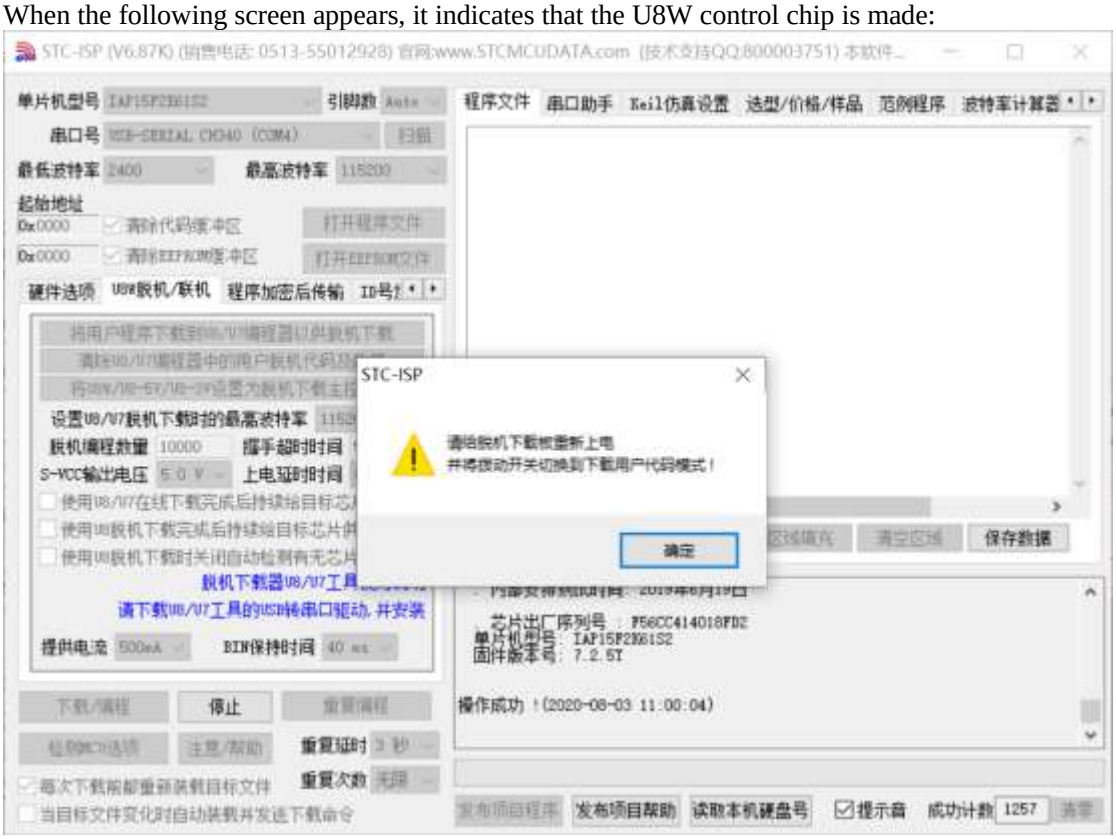
G.3.8 Make/Update U8W/U8W-Mini

The process of making U8W/U8W-Mini download master is similar. To save space, the following uses U8W as an example to describe how to make U8W download master. Before making the U8W download master, you need to dial the "Update/Download Selection Interface" of the U8W download board to "Upgrade Tool Firmware", as shown in the figure below:

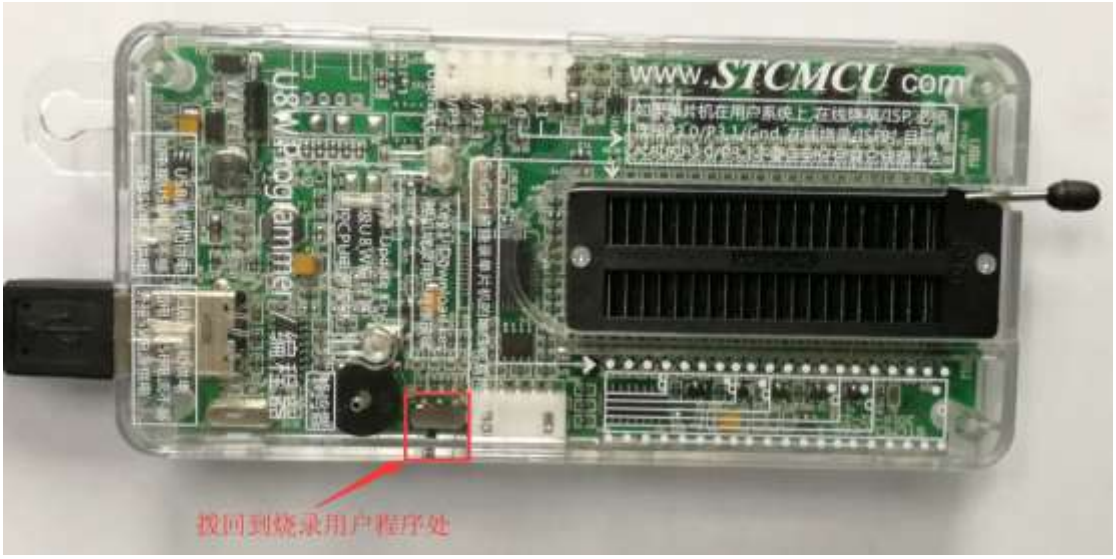


Then click the "Set U8W/U8-5V/U8-3V as the offline download master chip" button on the "U8W Offline/Online" page in the STC-ISP download program, as shown in the figure below: (Note: Be sure to select The serial port corresponding to U8W)

When the following screen appears, it indicates that the U8W control chip is made:



After the production is completed, do not forget to dial the "Update/Download Selection Interface" of U8W back to the "Burn User Program" mode, and power on the U8W download tool again, as shown in the figure below: (Otherwise, programming will not be performed normally)



G.3.9 U8W/U8W-Mini set through mode (can be used for simulation)

To use U8W/U8-Mini for simulation, you must first set U8W/U8-Mini to pass-through mode. The method of U8W/U8W-Mini to realize USB to serial port pass-through mode is as follows:

1. First, the U8W/U8W-Mini firmware must be upgraded to v1.37 and above;
2. After U8W/U8W-Mini is powered on, it is in normal download mode. At this time, press and hold the Key1 (download) button on the tool and do not release it. Press the Key2 (power) button again, and then release the Key2 (power) button. Then release the Key1 (download) button, U8W/U8W-Mini will enter the USB to serial port pass-through mode. (Press Key1 ➡ Press Key2 ➡ Release Key2 ➡ Release Key1);
3. The U8W/U8W-Mini tool that enters the pass-through mode is just a simple USB to serial port and does not have the offline download function. If you need to restore the original functions of U8W/U8W-Mini only need to press the Key2 (power) button separately again.

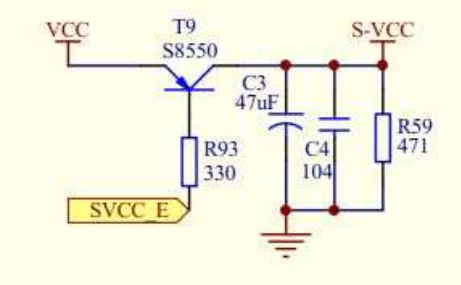
G.3.10 Reference circuit of U8W/U8W-Mini

USB online/offline download board U8W/U8W-Mini provides users with the following common control interfaces:

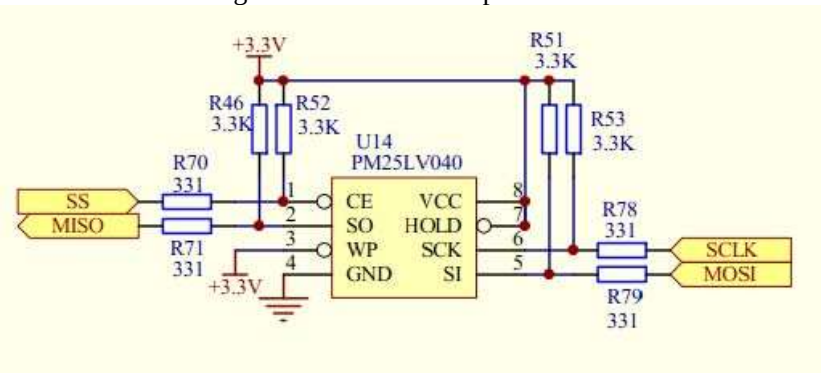
Pin function	Port	Function description
Power control pin	P2.6	Low effective
Download communication pin	P1.0	Serial port RXD, connect to the TXD of the target chip (P3.1)
	P1.1	Serial port TXD, connect to RXD of target chip (P3.0)
Programming button	P3.6	Low effective
display	P3.2	LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
External serial Flash control pin	P2.4	Flash CE Pin
	P2.2	Flash SO Pin
	P2.3	Flash SI Pin
	P2.1	Flash SCLK Pin

Automatic burning tool Sorter signal	P3.6	Start signal
	P1.5	Completion signal
	P5.4	OK signal (good signal)
	P3.7	ERROR signal (defective product signal)
Buzzer (BEEP) control	P2.5	High effective (sound at high level)

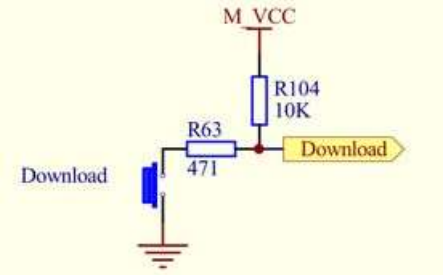
Reference circuit diagram for power control part:



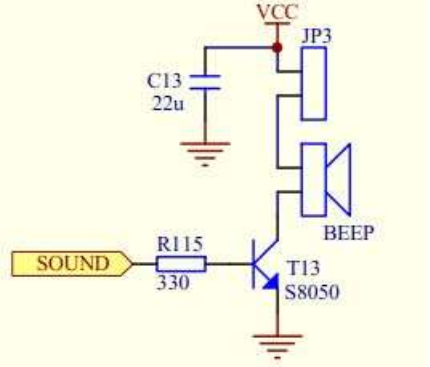
Reference circuit diagram of Flash control part:



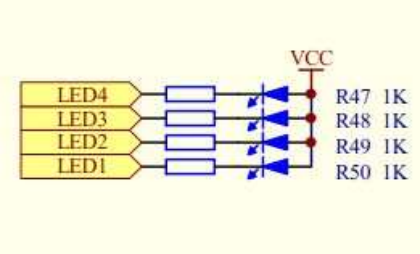
This Flash memory is required when the user program is larger than 41K
The reference circuit diagram of the button part:



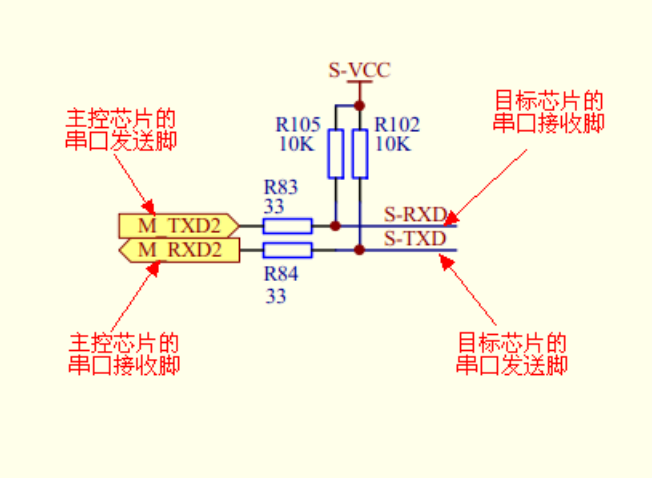
Reference circuit diagram of the buzzer part:



LED display part reference circuit diagram:



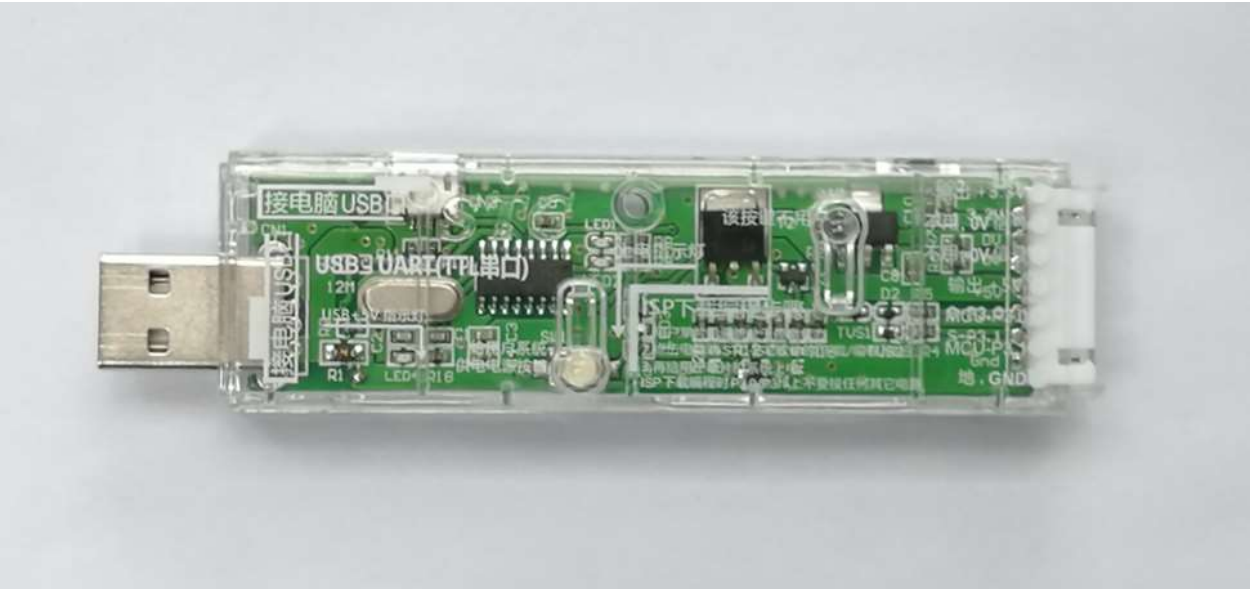
Reference circuit diagram of serial port communication pin connection part:



G.4 STC Universal USB to Serial Tool

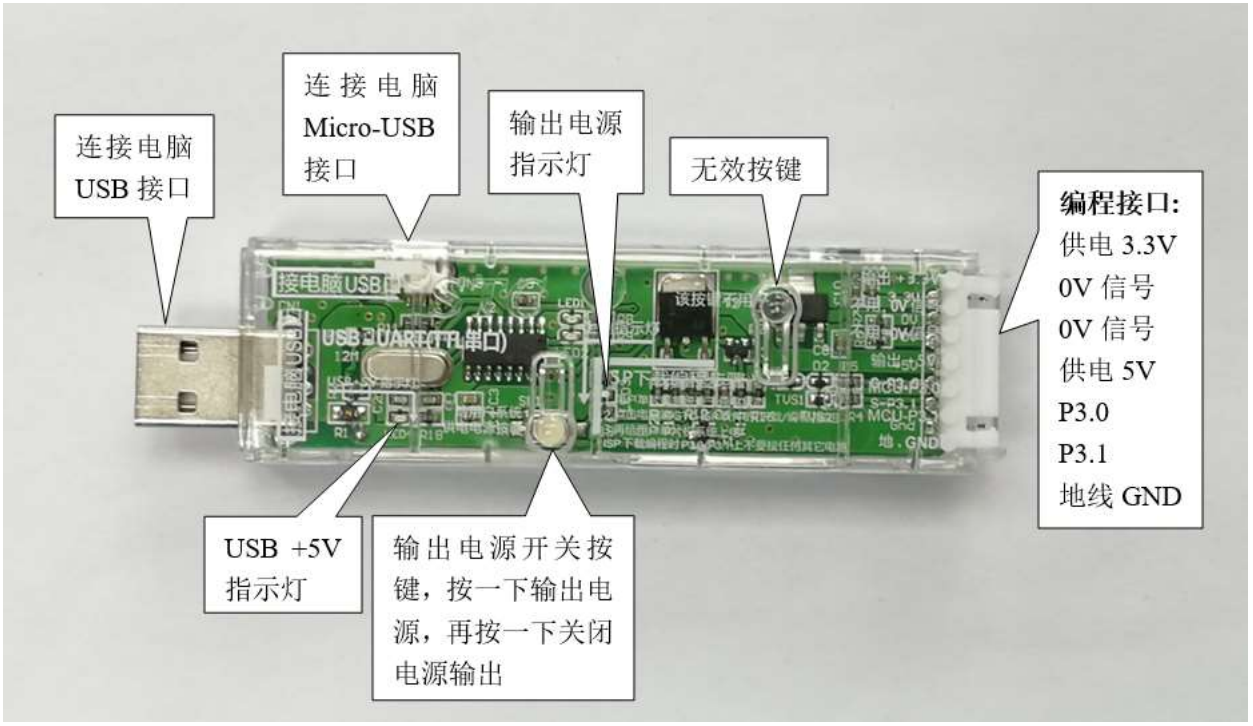
G.4.1 Appearance of STC Universal USB to Serial Tool

Front:



Back:

G.4.2 STC general USB to serial tool layout diagram



Here, the "power switch" needs to be explained:

The function of this button is the same as the self-locking switch. When the switch button is pressed for the first time, the switch is turned on and held, that is, self-locking. When the switch button is pressed for the second time, the switch turns off the power. In view of the characteristics of self-locking switches that are easily damaged during use, we have designed a set of circuits that use light touch switches to replace self-locking switches to increase the service life of the tools.

For STC microcontrollers, if you want to perform ISP download, you must receive the serial port command at power-on reset to start executing the ISP program, so the correct steps to download the program to the MCU using the STC universal USB to serial tool are:

1. Use STC universal USB to serial port tool to connect the MCU to be burned with the computer;
2. Open STC's ISP to download the software;
3. Select the MCU model;
4. Select the serial port corresponding to the STC Universal USB to Serial Tool;
5. Open the target file (HEX format or BIN format);
6. Click the "download/program" button in the ISP download software;
7. Press the "power switch" on the STC Universal USB to Serial Tool to power the MCU, and the download can start.

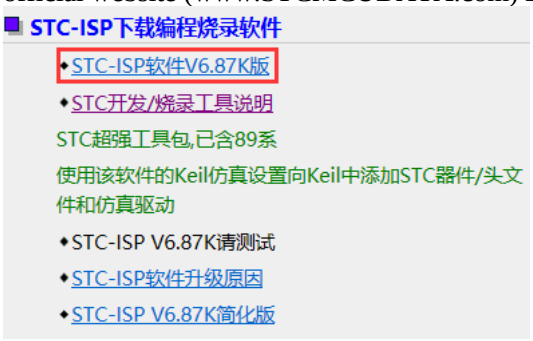
【Cold start burning】

In addition, the USB interface has the same function as the Micro-USB interface, and the user can connect one of the interfaces to the computer as needed.

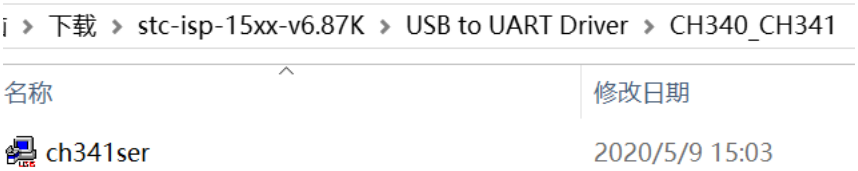
The 0V signal pin of the programming interface has a 470 ohm resistor grounded. It can be downloaded only when P1.0/P1.1=0/0 or P3.2/P3.3=0/0 is set. You can set P1.0, P1 .1 or P3.2, P3.3 are connected to the 0V signal pin.

G.4.3 STC Universal USB to Serial Tool Driver Installation

STC general USB to serial port tool uses CH340 USB to serial chip (can plug in crystal oscillator, more accurate), just download the general CH340 serial driver and install it. The following is the CH341SER serial driver provided by STC official website (www.STCMCUDATA.com) Download location:

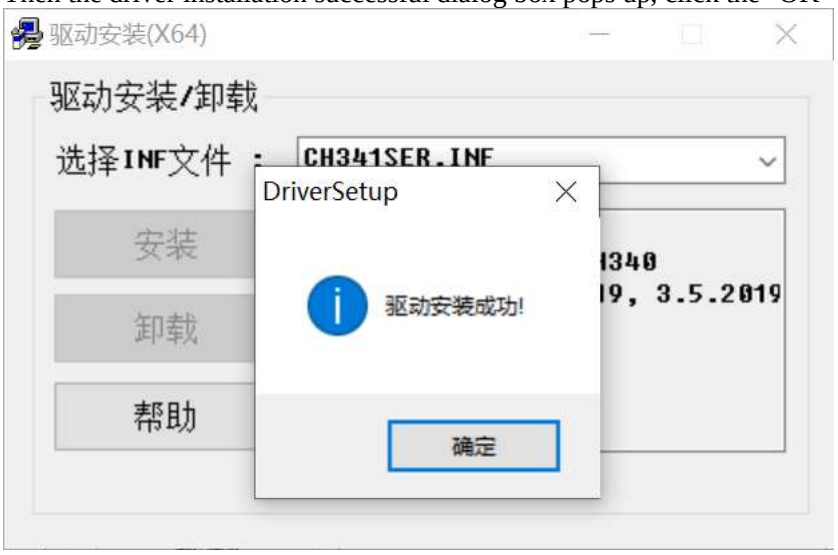


After downloading, decompress, the path of CH340 driver installation package is stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341:



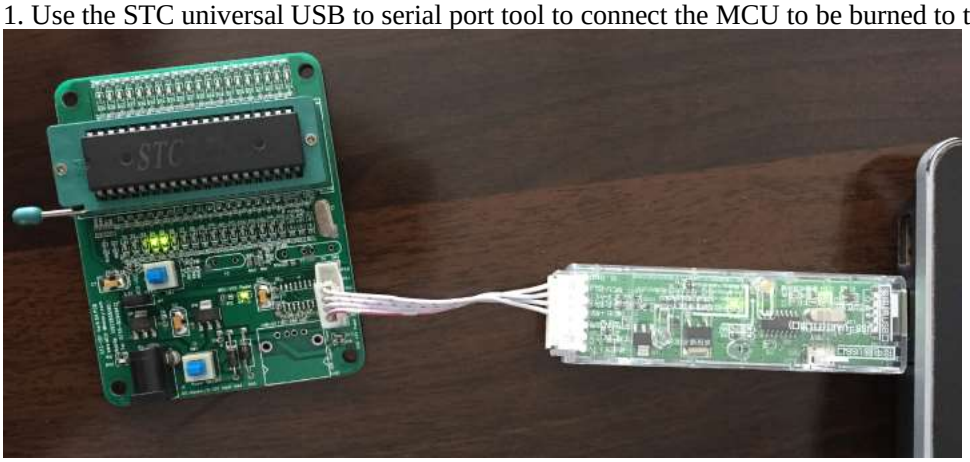
Take the CH341SER serial port driver provided by STC official website as an example, double-click the "CH341SER.exe" installation package, and click the "Install" button on the pop-up main interface to start installing the driver:

Then the driver installation successful dialog box pops up, click the "OK" button to complete the installation:



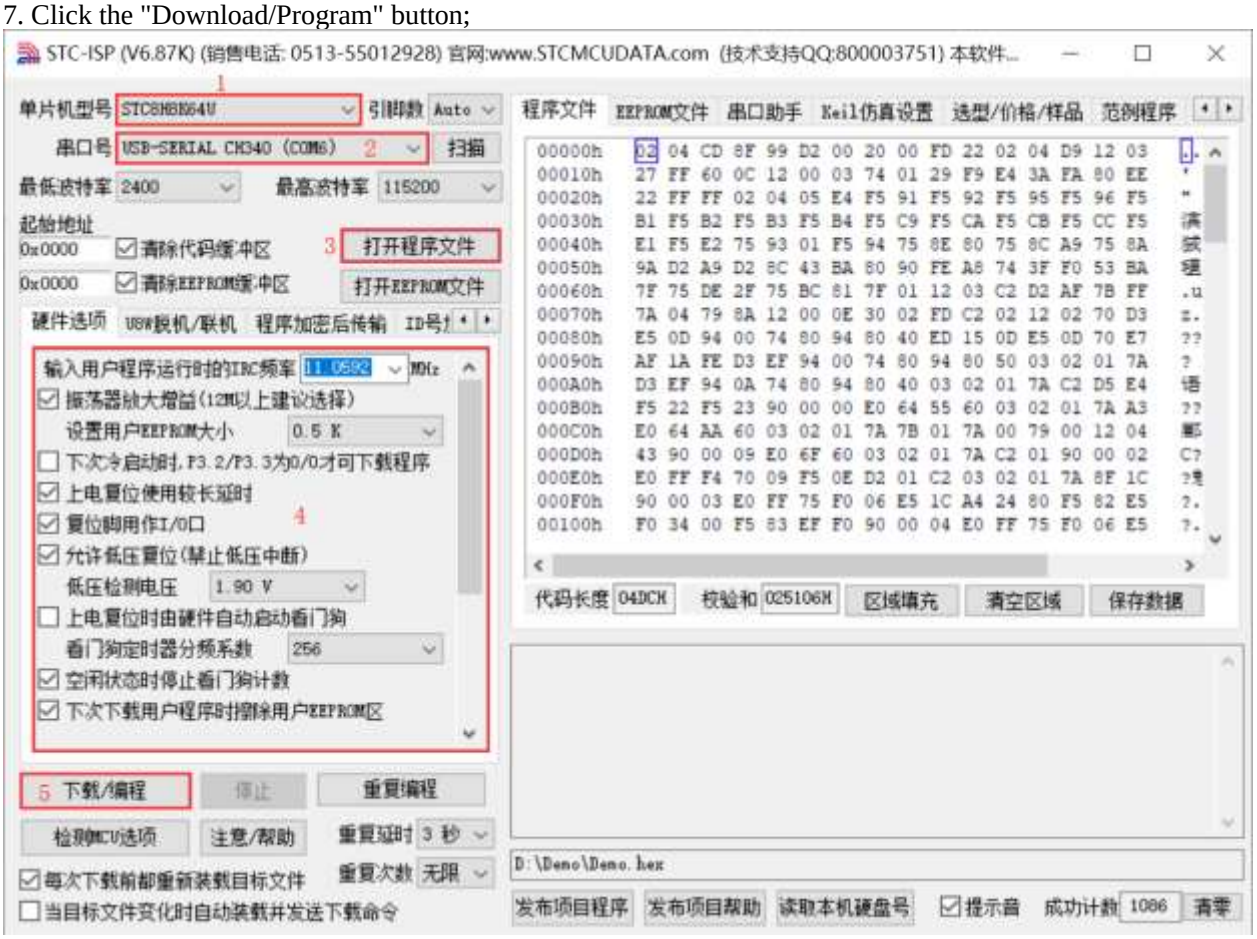
G.4.4 Use STC universal USB to serial port tool to download program to MCU

1. Use the STC universal USB to serial port tool to connect the MCU to be burned to the computer:

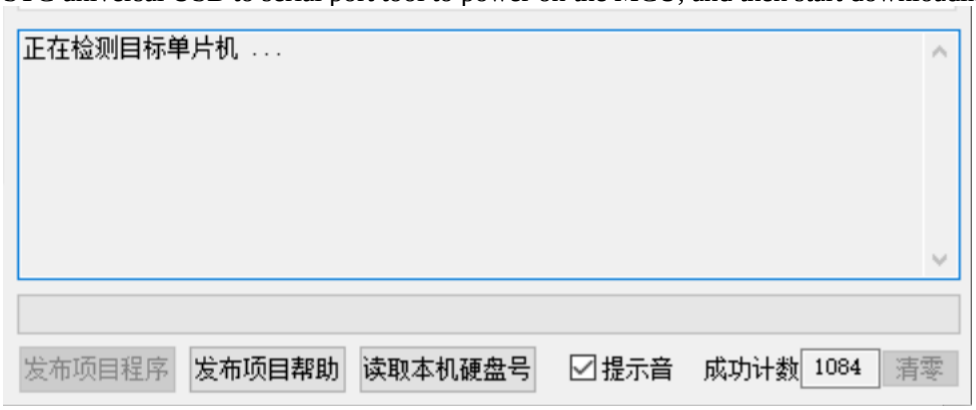


2. Open the STC-ISP software;

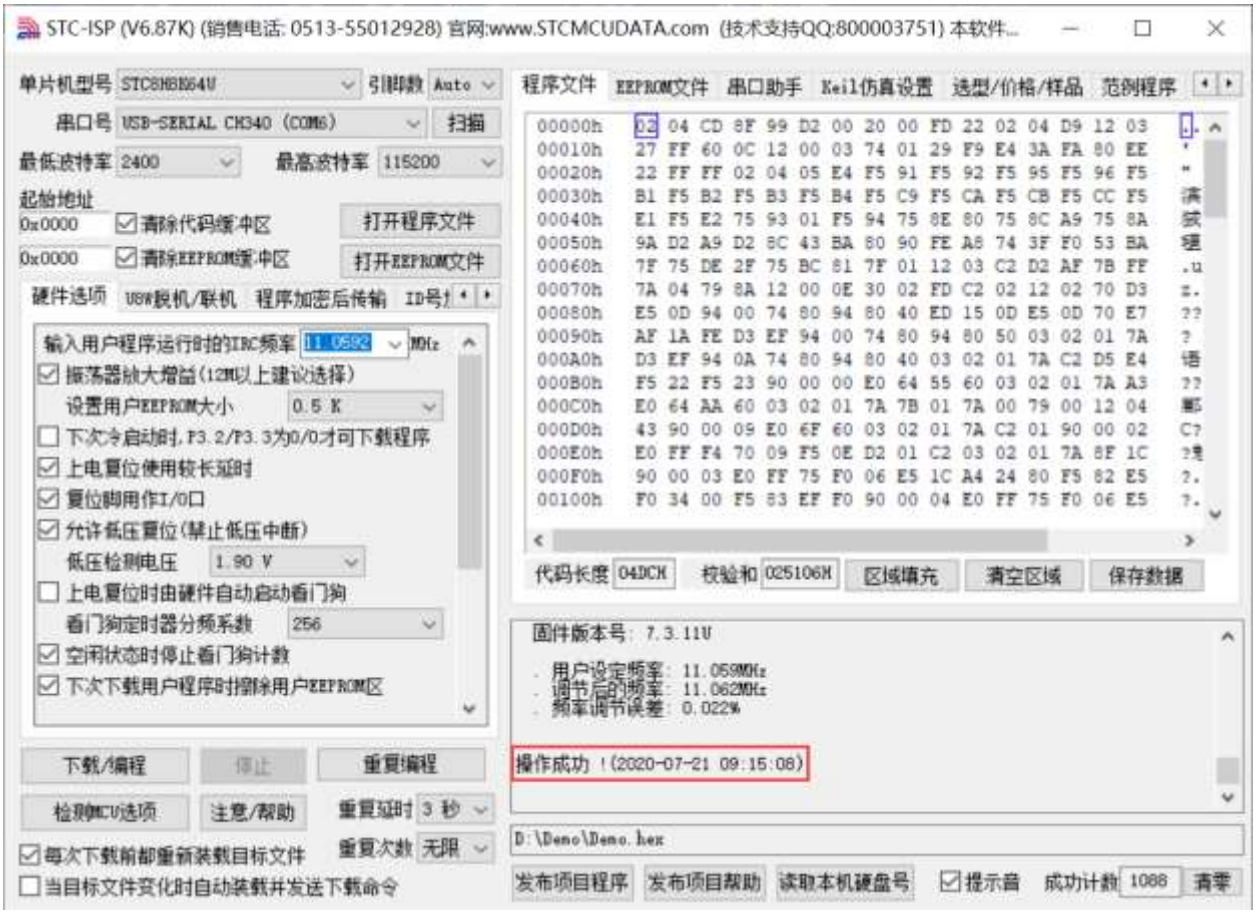
3. Select the model corresponding to the burning chip;
4. Select the serial port number recognized by the STC universal USB to serial tool (when the STC universal USB to serial tool is correctly connected to the computer, the software will automatically scan and identify the serial port named "USB-SERIAL CH340 (COMx)", the specific COM The number will vary from computer to computer). When multiple USB-to-serial cables are connected to the computer, they must be selected manually;
5. Load the burning program;
6. Set burning options;
7. Click the "Download/Program" button;



8. When the prompt box in the lower right corner displays "Detecting target MCU...", press the "power switch" on the STC universal USB to serial port tool to power on the MCU, and then start downloading [Cold Start Programming];



9. Wait for the download to end. If the download is successful, the prompt box in the lower right corner will display "Operation successful!"

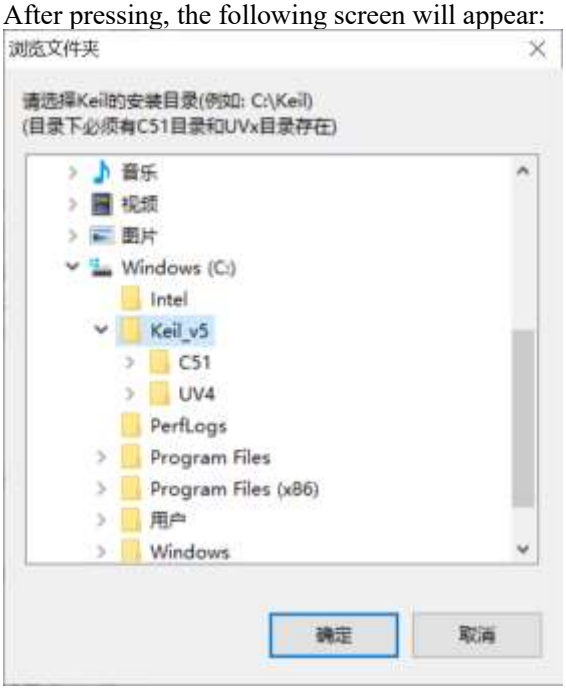


G.4.5 Use STC universal USB to serial port tool to simulate user code

The current STC simulation is based on the Keil environment, so if you need to use the STC universal USB to serial port tool to simulate user code, you must install the Keil software.

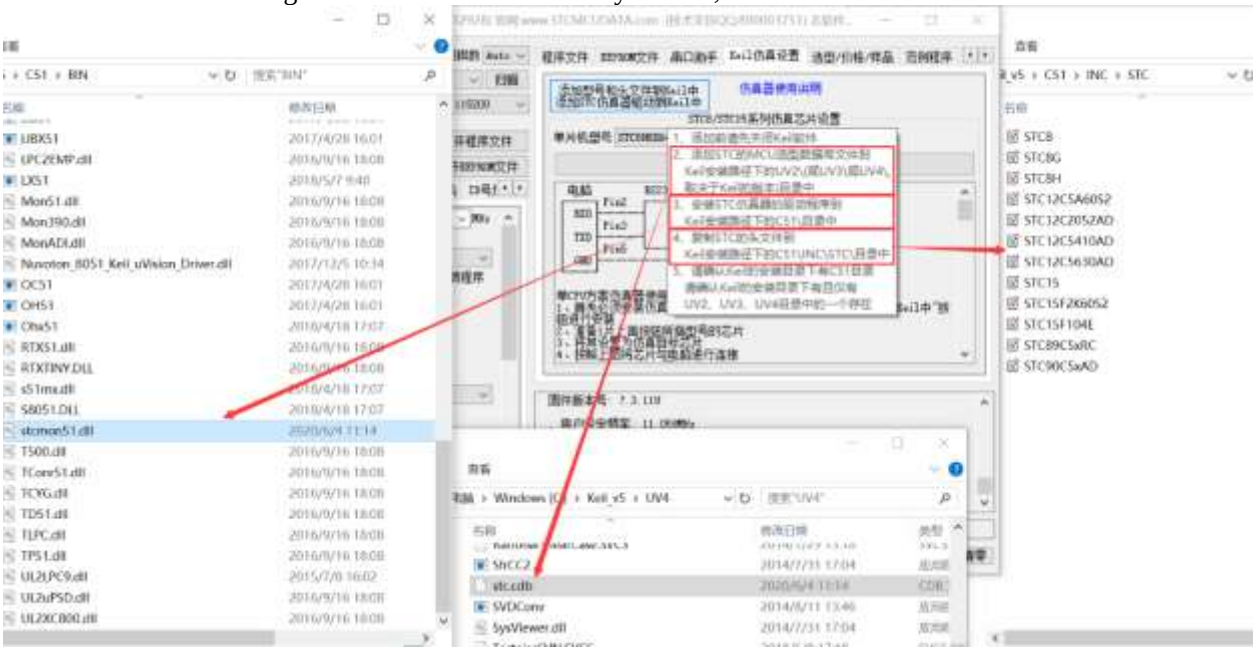
After the Keil software is installed, you also need to install the STC simulation driver. The installation steps of STC's simulation driver are as follows:
Firstly, open STC-ISP to download the software;
Then click the "Add model and header files to Keil and add STC emulator driver to Keil" on the "Keil simulation settings" page in the functional area on the right side of the software:

After pressing, the following screen will appear:



Locate the directory to the installation directory of the Keil software, and then confirm.
After the installation is successful, the following prompt box will pop up:

You can see the following files in the relevant directory of Keil, which means that the driver is installed correctly.



Since in the default state, the main control chip of STC is not an emulation chip and has no emulation function, if simulation is needed, the main control chip of STC needs to be set as an emulation chip.
The steps of making a simulation chip are as follows:

Firstly, use STC universal USB to serial port tool to connect the MCU to the computer;
Then open STC's ISP to download the software, and select the serial port number corresponding to the serial port tool in the serial port number drop-down list;
Select the MCU model;
Select the IRC frequency of the user program to run, and the frequency selected when making the simulation chip is consistent with the frequency set by the simulated user program, in order to achieve the real running effect.

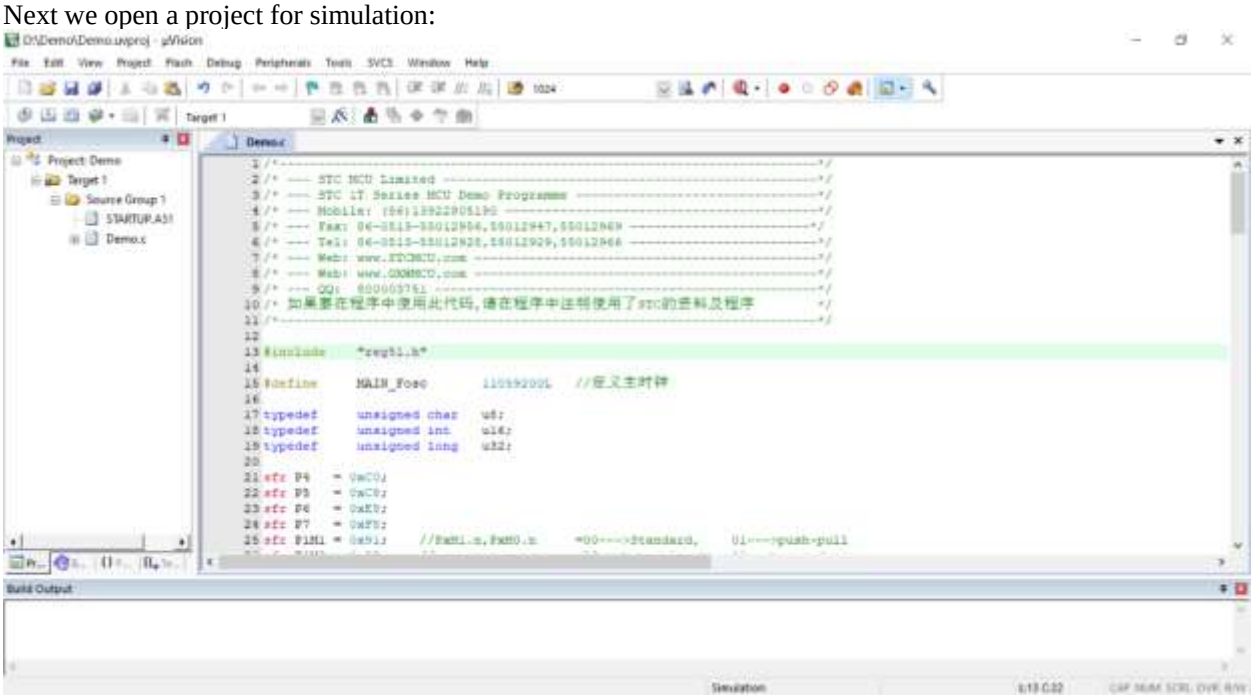
Then click the "Set the selected target MCU as an emulation chip" button on the "Keil Simulation Settings" page in the right functional area of the software,After pressing, the following screen will appear:

Next, you need to press the "power switch" on the STC Universal USB to Serial Tool to supply power to the MCU [cold start], and you can start to make the simulation chip.

If the setting is successful, the following screen will appear:

At this point, the simulation chip has been made successfully.

Next we open a project for simulation:

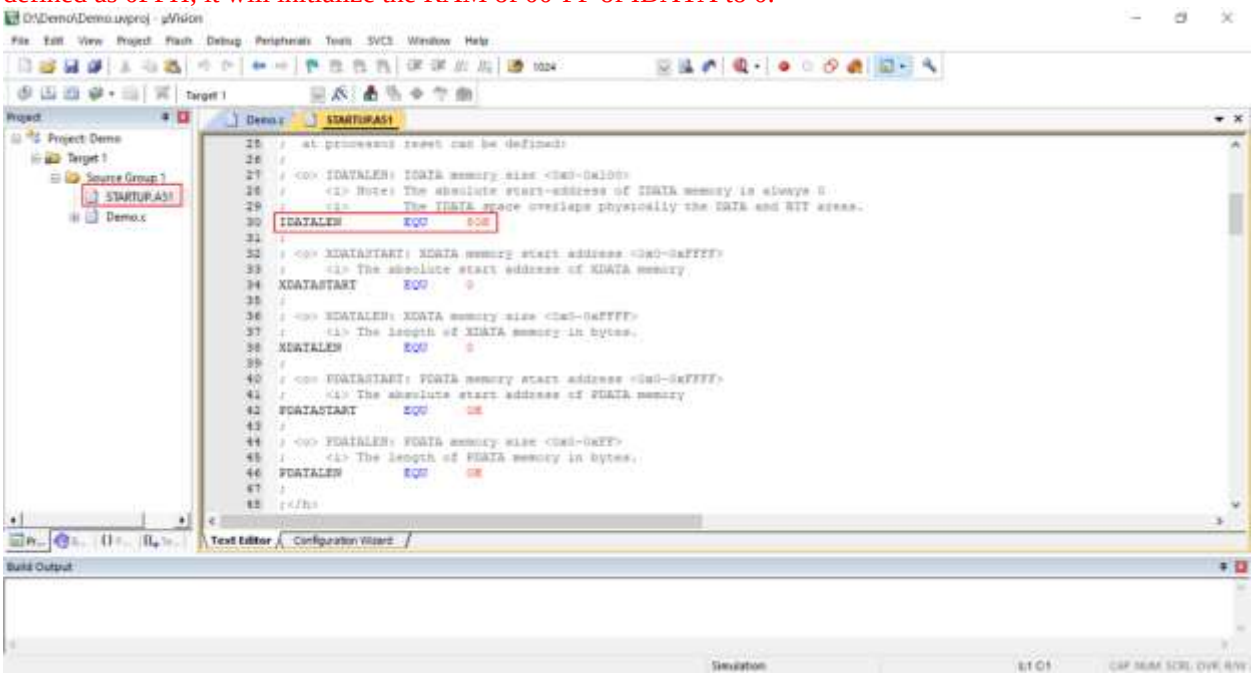


Then make the following project settings:

An additional note:

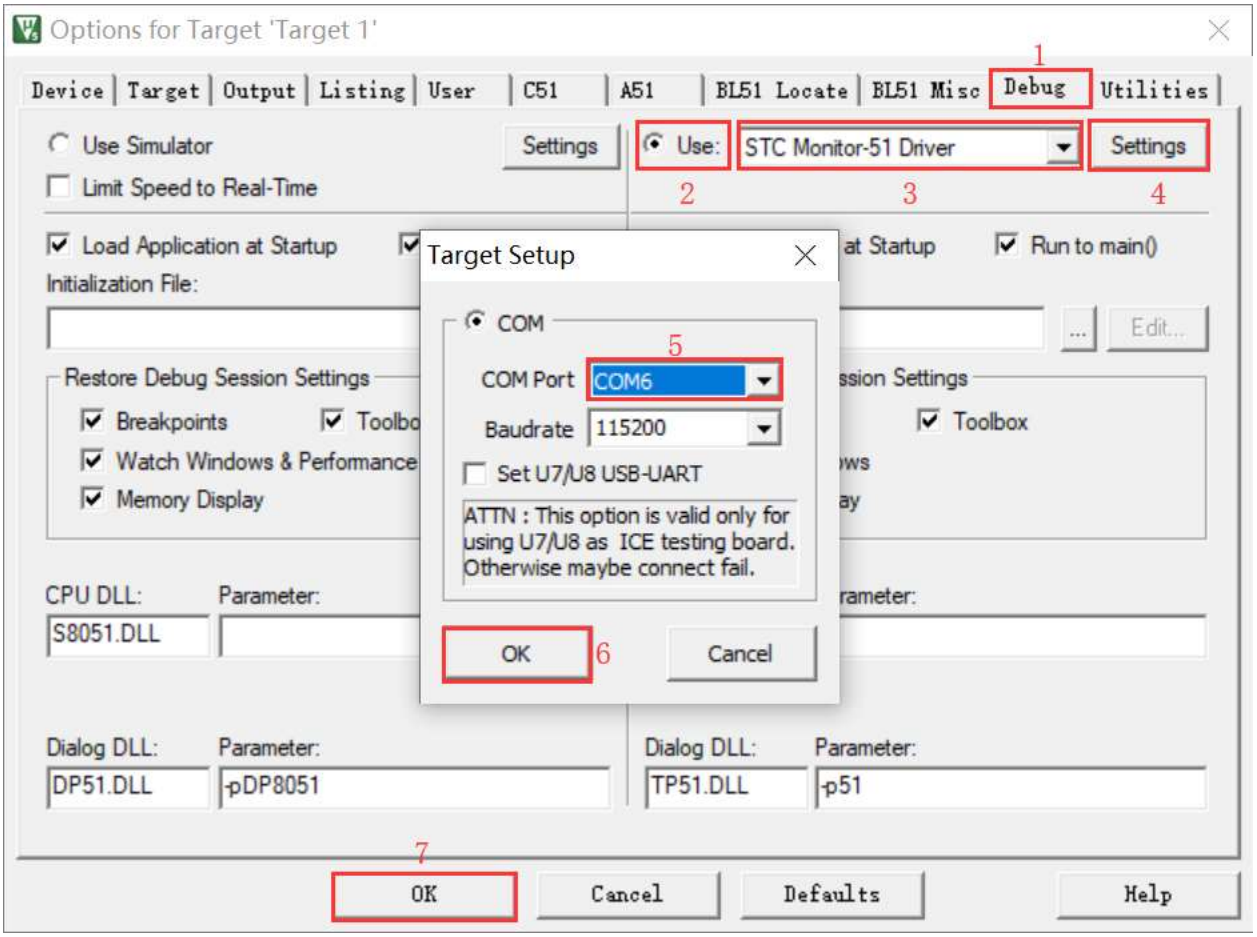
When a C language project is created and the startup file "STARTUP.A51" is added to the project, there is a macro definition named "IDATALEN", which is used to define the size of IDATA. The default value is 128, which is 80H in

hexadecimal, and it is also the size of IDATA that needs to be initialized to 0 in the startup file. So when IDATA is defined as 80H, the code in STARTUP.A51 will initialize the RAM of 00-7F of IDATA to 0; similarly, if IDATA is defined as 0FFH, it will initialize the RAM of 00-FF of IDATA to 0.

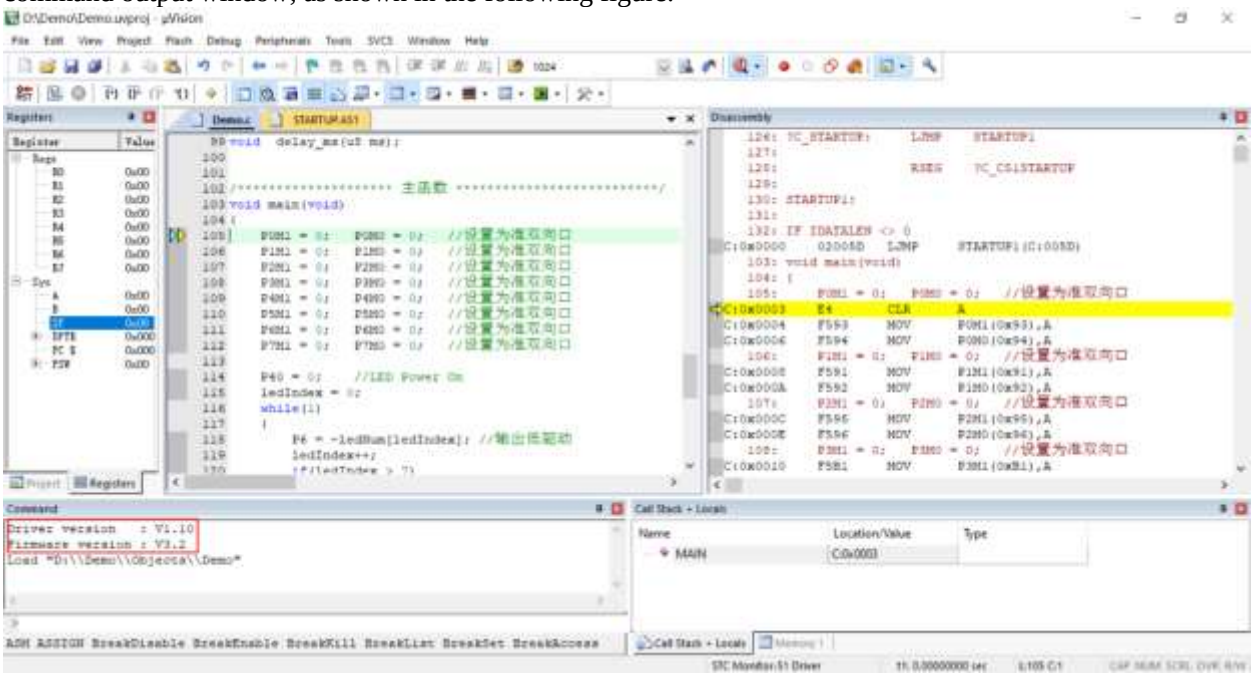


The IDATA size of the STC8 series microcontroller we selected is 256 bytes (00-7F DATA and 80H-FFH IDATA), but because the last 17 bytes of RAM have written ID numbers and related test parameters, If users need to use this part of data in the program, they must not define IDATALEN as 256.

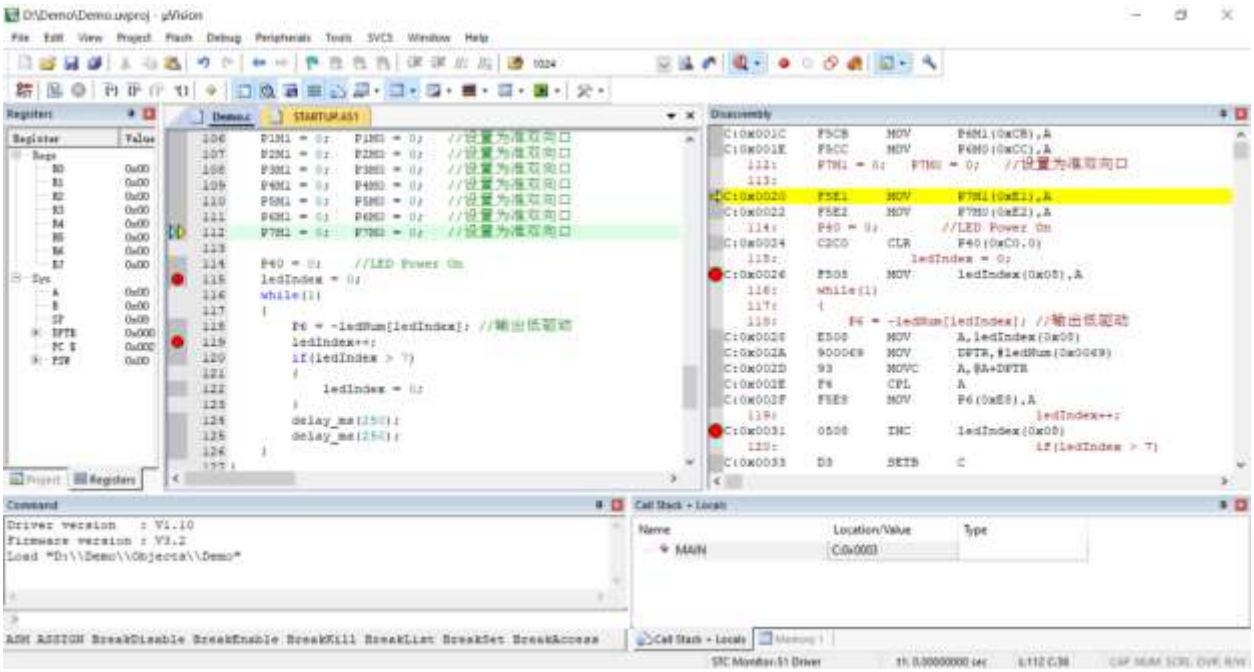
Press the shortcut key "Alt+F7" or select "Option for Target ‘Target1’" in the menu "Project" to configure the project in the "Option for Target ‘Target1’" dialog box:
Step 1. Enter the project setting page and select the "Debug" setting page;
Step 2. Select the hardware emulation "Use …" on the right;
Step 3. Select "STC Monitor-51 Driver" item in the simulation driver drop-down list;
Step 4. Click the "Settings" button to enter the serial port settings screen;
Step 5: Set the port number and baud rate of the serial port. The serial port number should be the serial port corresponding to the STC universal USB to serial port tool. The baud rate is generally 115200 or 57600.
Make sure to complete the simulation settings.
The detailed steps are shown in the figure below:



After finishing all the above work, you can press "Ctrl+F5" in Keil software to start simulation debugging.
If the hardware connection is correct, you will enter a debugging interface similar to the following, and display the current simulation driver version number and the current simulation monitoring code firmware version number in the command output window, as shown in the following figure:



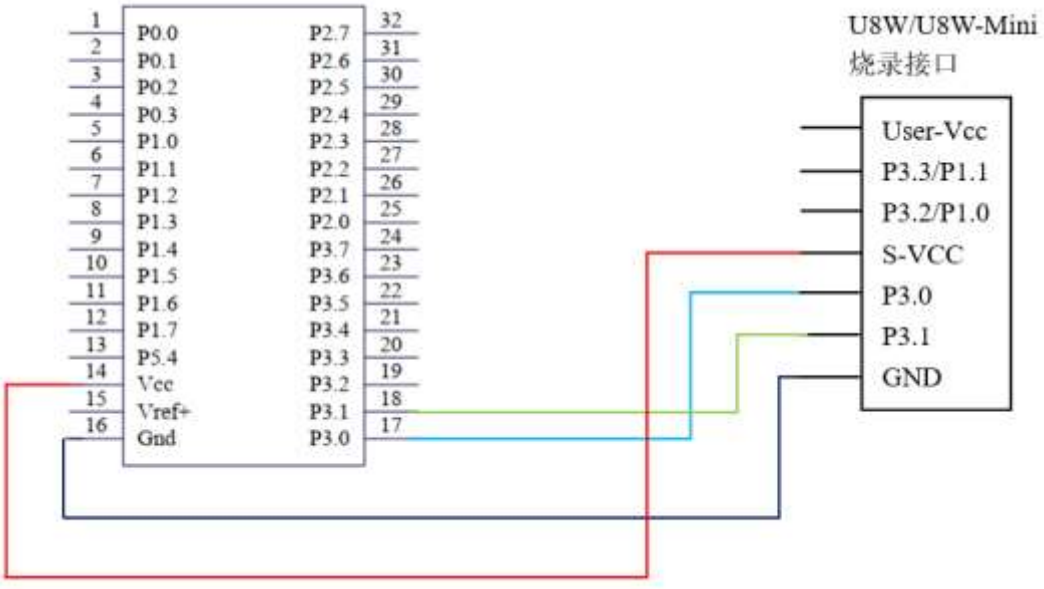
During the simulation debugging process, you can perform multiple operations such as resetting, running at full speed, single stepping, and setting breakpoints.



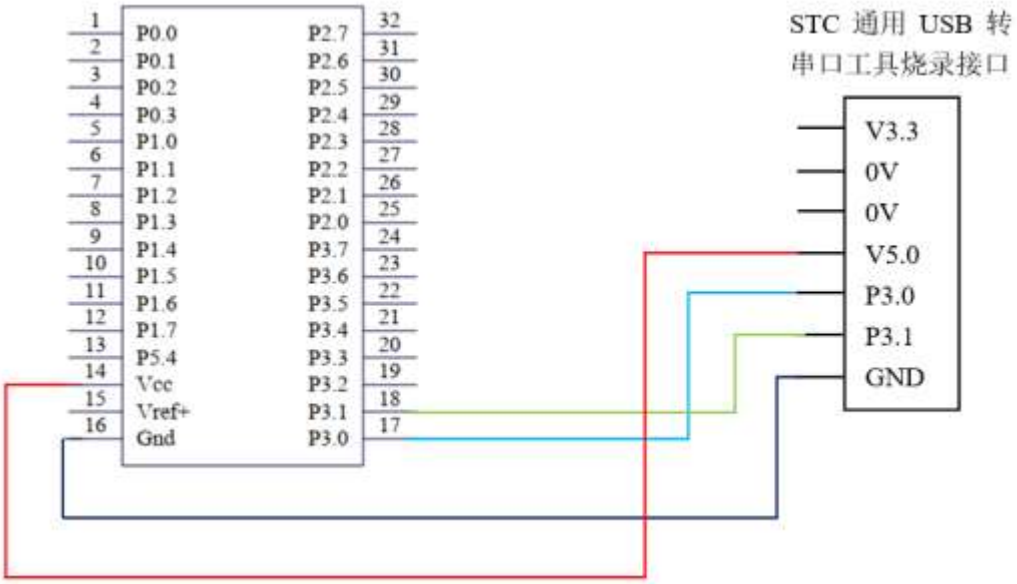
As shown in the figure above, multiple breakpoints can be set in the program, and the maximum number of breakpoint settings currently allowed is 20 (in theory, any number can be set, but setting too many breakpoints will affect the speed of debugging).

G.5 Application circuit diagram

G.5.1 U8W tool application reference circuit diagram



G.5.2 STC Universal USB to Serial Tool Application Reference Circuit Diagram



Appendix H STC Emulation Instruction Manual

H.1 Overview

All STC8G/8H series microcontrollers support online emulation, including downloading user code, chip reset, full-speed operation, single-step operation, setting breakpoints (the number of theoretical breakpoints is unlimited, but in order to improve the emulation efficiency, the current limit is up to 20 breakpoints), viewing variables, etc. The emulation operation is convenient for users to debug the code and find logical errors in the code, thereby shortening the project development cycle.

The emulation interface can be USB or serial port, the microcontroller itself is an emulator, and all emulation functions can be realized without an additional emulator. The corresponding USB port or serial port is originally a dedicated port for emulation, but when the emulation function is turned off, the user can freely use the emulation interface as GPIO, USB or serial port.

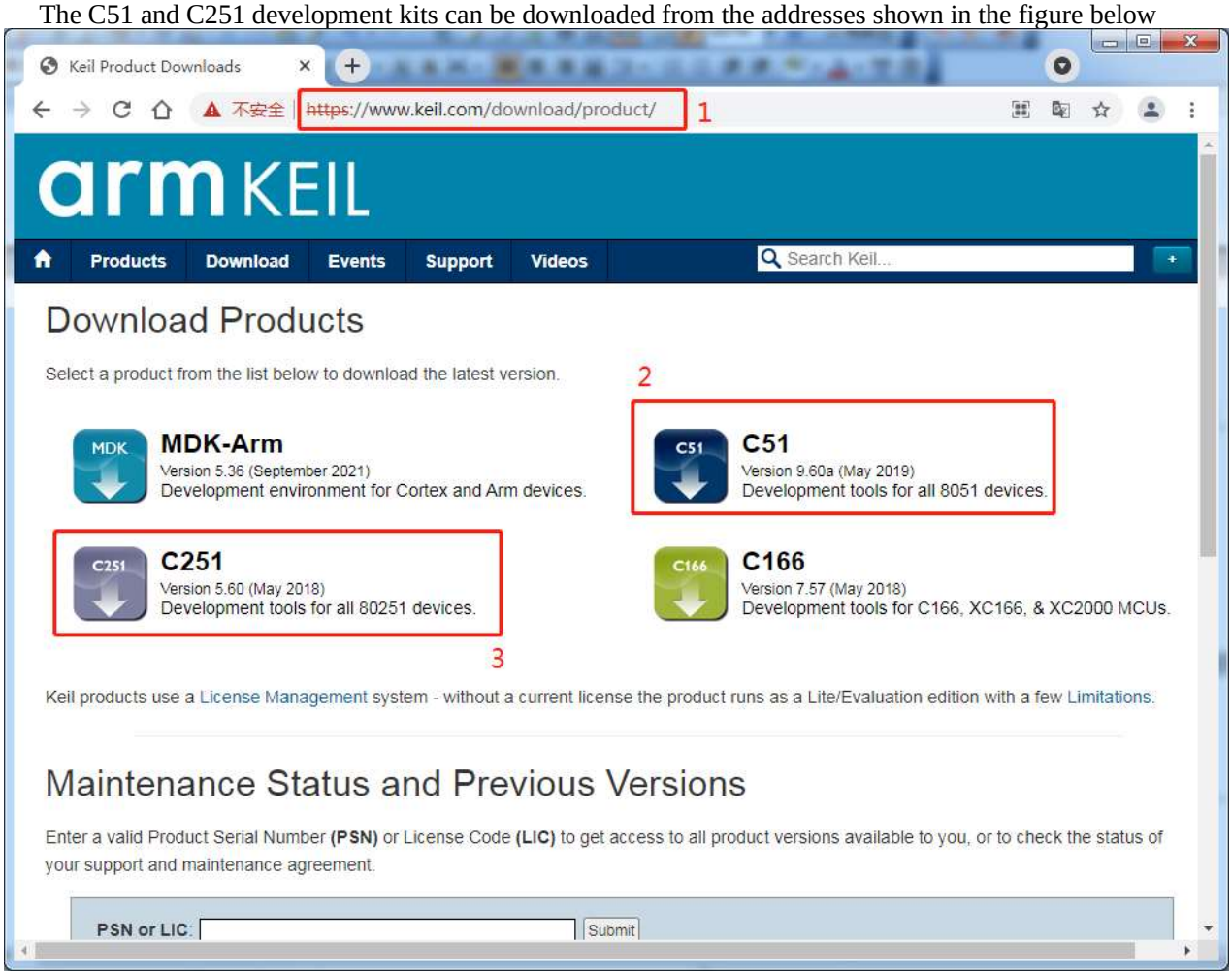
At present, the emulation mode of all MCUs is the software monitoring emulation mode, which will occupy part of the system resources. The resources occupied by each series of MCU simulation are shown in the following table**:**

MCU Series	Emulation Port	Resource occupied	
		Port	Data memory (XDATA)
STC8H8K64U family-B version	USB	D+，D-	768 bytes (1D00H-1FFFH)
	UART	P3.0，P3.1	768 bytes (1D00H-1FFFH)
		P3.6，P3.7	
		P1.6，P1.7	
		P4.3，P4.4	
STC8H8K64U family-A version	UART	P3.0，P3.1	768 bytes (1D00H-1FFFH)
		P3.6，P3.7	
		P1.6，P1.7	
		P4.3，P4.4	
STC8H4K64T family	UART	P3.0，P3.1	768 bytes (0D00H-0FFFH)
STC8H3K64S4 family	UART	P3.0，P3.1	768 bytes (0900H-0BFFH)
STC8H1K16 family	UART	P3.0，P3.1	768 bytes (0100H-03FFH)
STC8H1K08 family	UART	P3.0，P3.1	768 bytes (0100H-03FFH)
STC8G2K64S4 family	UART	P3.0，P3.1	768 bytes (0500H-07FFH)
STC8G1K08 family	UART	P3.0，P3.1	768 bytes (0100H-03FFH)
STC8C2K64S4 family	UART	P3.0，P3.1	768 bytes (0500H-07FFH)
STC8A8K64D4 family	UART	P3.0，P3.1	768 bytes (1D00H-1FFFH)
STC8A8K64S4A12 family	UART	P3.0，P3.1	768 bytes (1D00H-1FFFH)
STC8F2K64S4 family	UART	P3.0，P3.1	768 bytes (0500H-07FFH)
STC8F1K08S2 family	UART	P3.0，P3.1	768 bytes (0100H-03FFH)
IAP15W4K58S4	UART	P3.0，P3.1	768 bytes (0D00H-0FFFH)
IAP15F2K61S2	UART	P3.0，P3.1	768 bytes (0500H-07FFH)

H.2 Install Keil software

The emulation of STC microcontroller is based on the Keil development environment, so before the emulation, the Keil software must be installed.

The C51 and C251 development kits can be downloaded from the addresses shown in the figure below



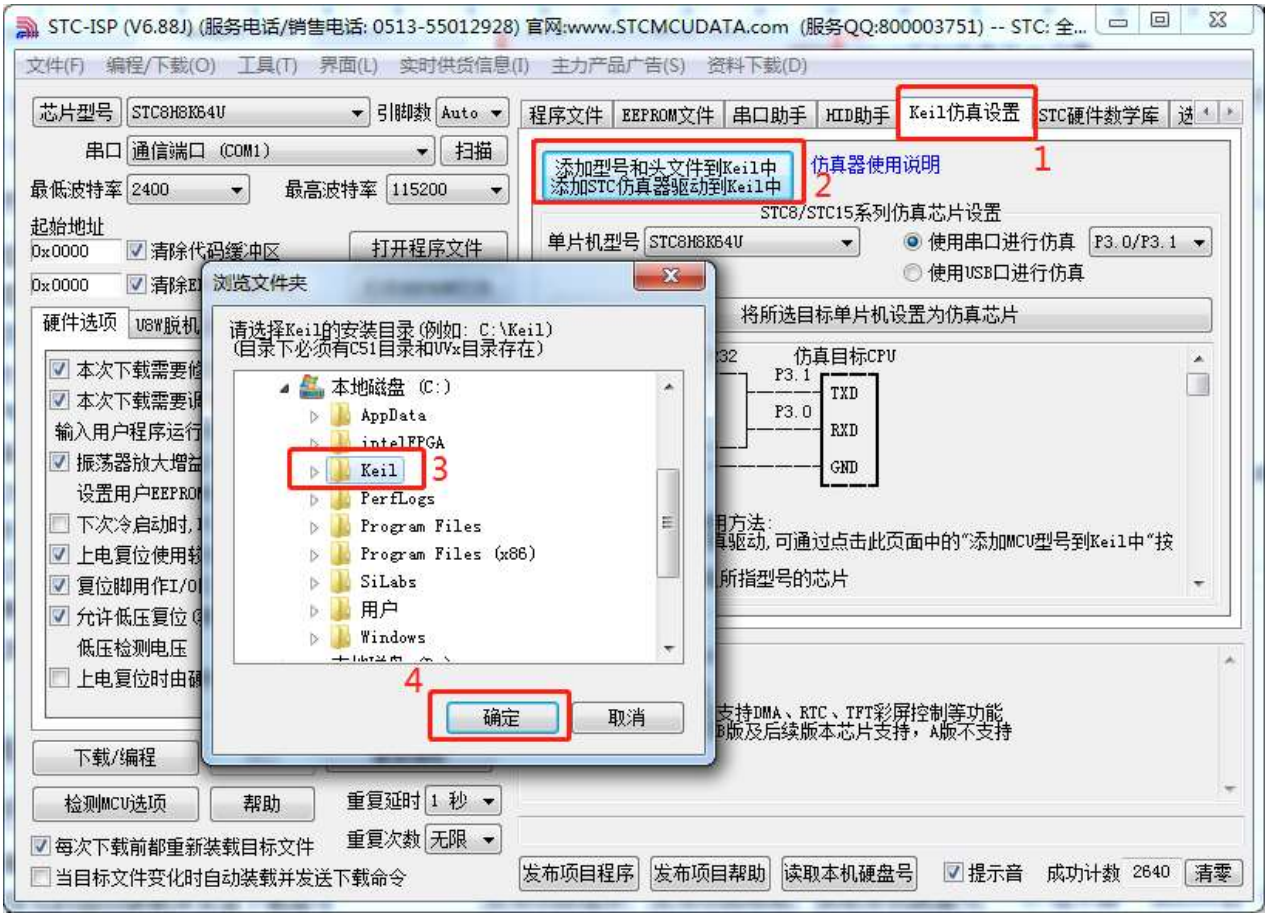
Note: The latest Keil-UV5 software does not include 8051 and 80251 toolkits by default, and must be downloaded and installed manually.

H.3 Install the emulation driver

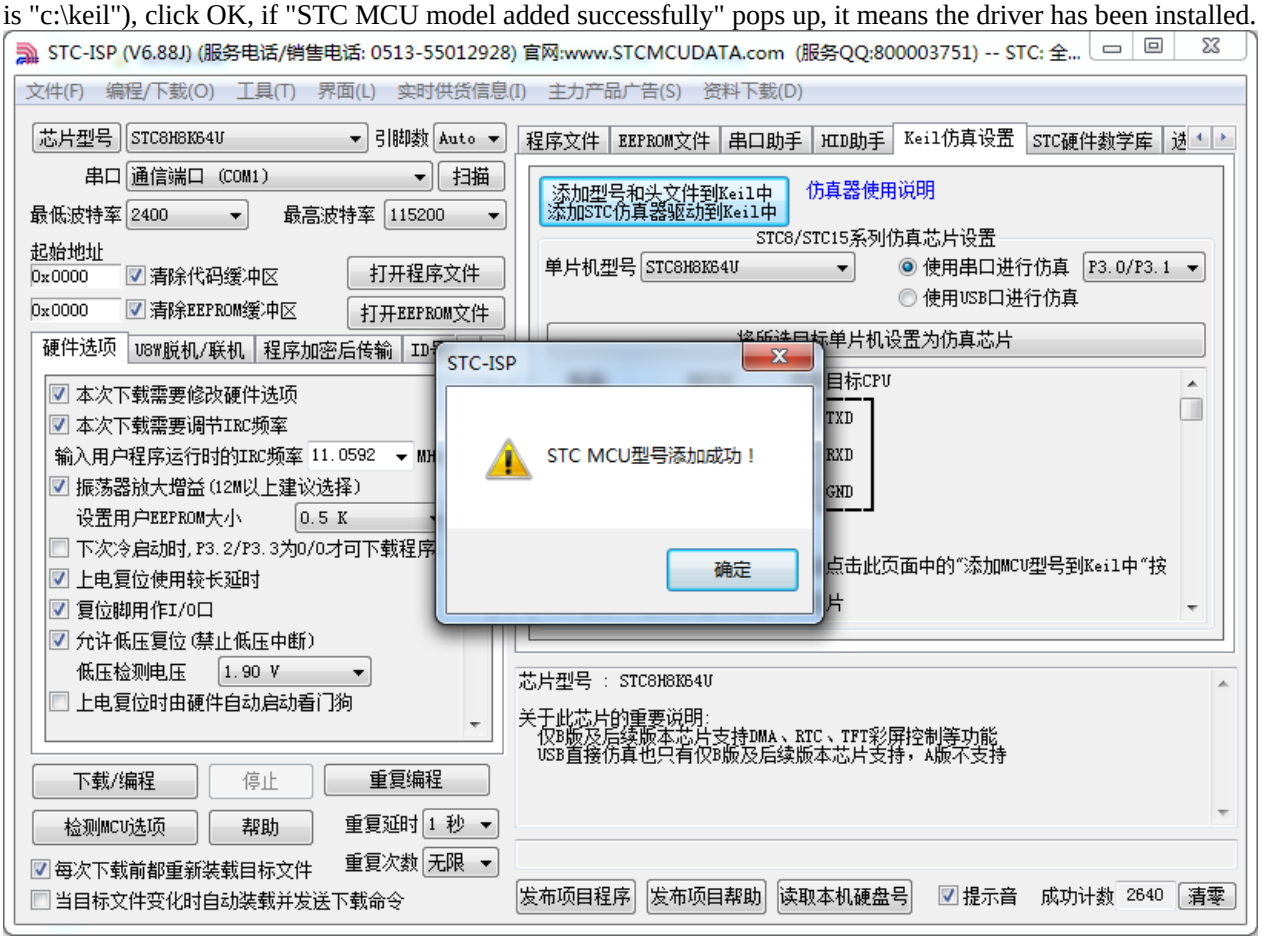
After the Keil development environment is installed, you also need to install the STC-specific emulation driver. Proceed as follows:
Firstly, download the latest STC-ISP download software from the STC official website

After downloading and unzipping, open the "stc-isp-vxx.exe" executable in the package.

Click the "Add Model and Header File..." button in the "Keil Simulation Settings" page of the download software (Figure "2" below)


STC-ISP (V6.88J) (服务电话/销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (服务QQ:800003751) -- STC: 全...
文件(F) 编程/下载(O) 工具(T) 界面(L) 实时供货信息(I) 主力产品广告(S) 资料下载(D)
芯片型号 STC8H8K64U 引脚数 Auto
串口 通信端口 (COM1) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000
硬件选项 U8W脱机
本次下载需要修
本次下载需要调
输入用户程序运行
振荡器放大增益
设置用户EEPROM
下次冷启动时,
上电复位使用较
复位脚用作I/O
允许低压复位
低压检测电压
上电复位时由硬
下载/编程
检测MCU选项 帮助 重复延时 1 秒
每次下载前都重新装载目标文件 重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件 EEPROM文件 串口助手 HID助手 Keil仿真设置 STC硬件数学库
添加型号和头文件到Keil中
添加STC仿真器驱动到Keil中
仿真器使用说明
1
2
STC8/STC15系列仿真芯片设置
单片机型号 STC8H8K64U
使用串口进行仿真 P3.0/P3.1
使用USB口进行仿真
将所选目标单片机设置为仿真芯片
仿真目标CPU
P3.1
P3.0
TXD
RXD
GND
用方法:
具驱动,可通过点击此页面中的"添加MCU型号到Keil中"按
所指型号的芯片
支持DMA、RTC、TFT彩屏控制等功能
B版及后续版本芯片支持,A版不支持
发布项目程序 发布项目帮助 读取本机硬盘号 提示音 成功计数 2640 清零
浏览文件夹
请选择Keil的安装目录(例如: C:\Keil)
(目录下必须有C51目录和UVx目录存在)
本地磁盘 (C:)
AppData
intelFPGA
Keil
3
PerfLogs
Program Files
Program Files (x86)
SiLabs
用户
Windows
4
确定
取消

In the pop-up "Browse Folder" window, select the Keil installation directory (usually Keil's installation directory is "c:\keil"), click OK, if "STC MCU model added successfully" pops up, it means the driver has been installed. .



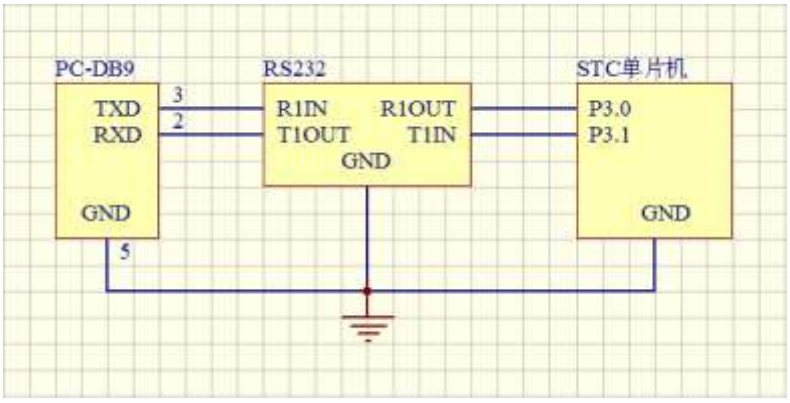
H.4 Serial port emulation directly

H.4.1 Make serial port emulation chip

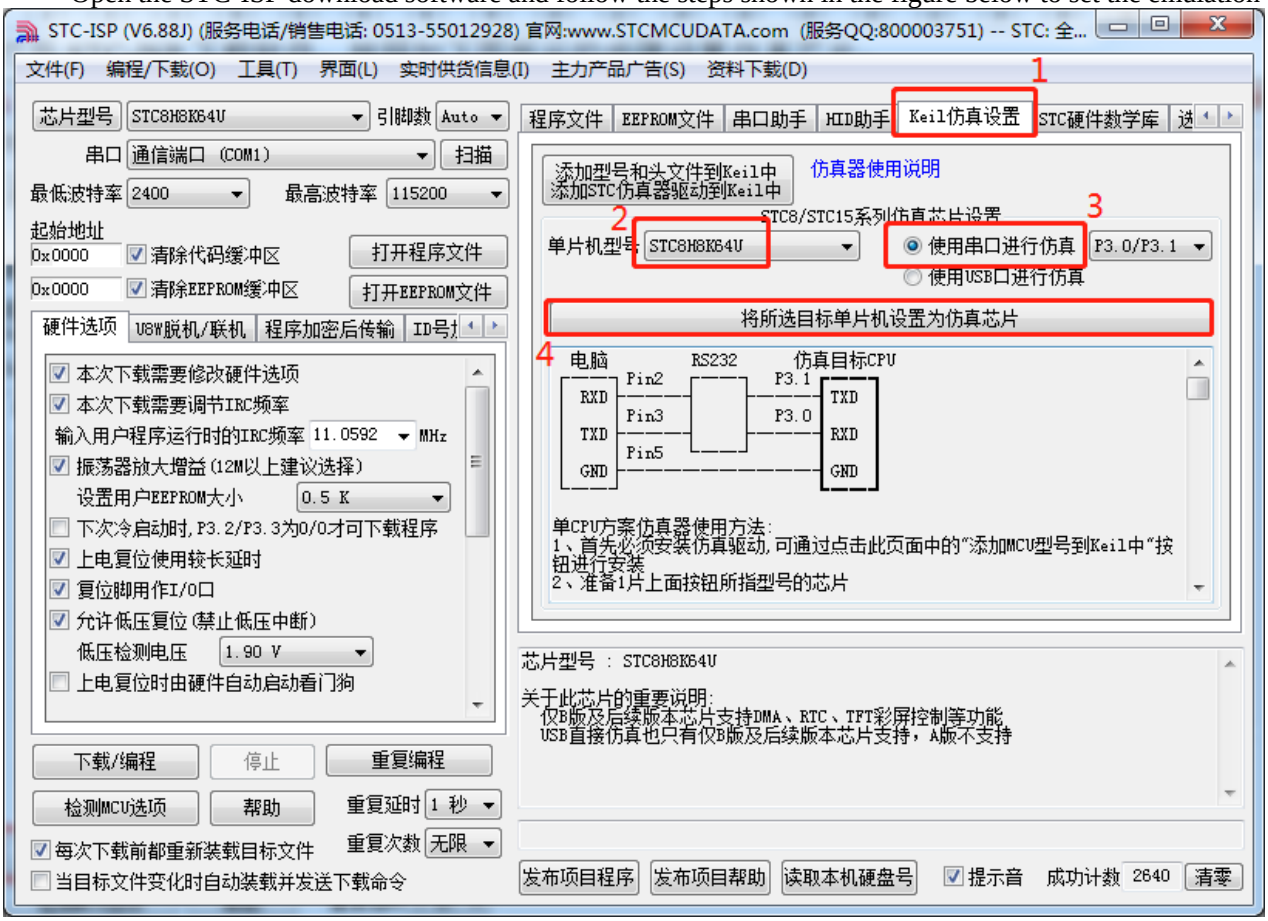
When the STC microcontroller is shipped, the emulation function is disabled by default. If you want to use the emulation function, you need to use the STC-ISP download software to set the target microcontroller as the emulation chip.

The setting steps are as follows:

Firstly, connect the target chip to the serial port of the computer as shown in the figure below, and power off the microcontroller



Open the STC-ISP download software and follow the steps shown in the figure below to set the emulation chip.



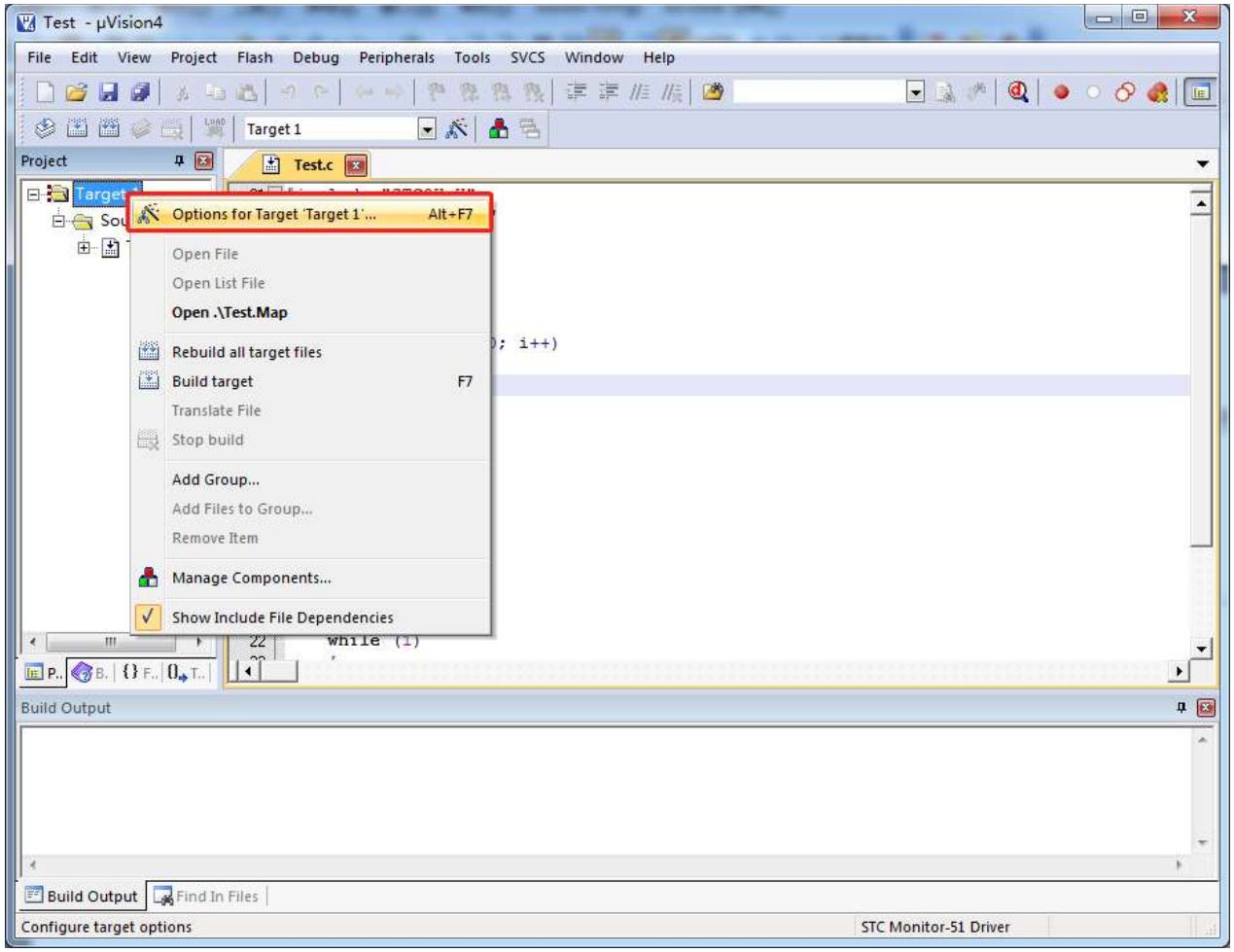
When the following screen appears, power on the microcontroller again.


STC-ISP (V6.88J) (服务电话/销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (服务QQ:800003751) -- STC: 全...
文件(F) 编程/下载(O) 工具(T) 界面(L) 实时供货信息(I) 主力产品广告(S) 资料下载(D)
芯片型号 STC8H8K64U 引脚数 Auto
串口 通信端口 (COM1) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 U8W脱机/联机 程序加密后传输 ID号
本次下载需要修改硬件选项
本次下载需要调节IRC频率
输入用户程序运行时的IRC频率 11.0592 MHz
振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
下次冷启动时, P3.2/P3.3为0/0才可下载程序
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 1.90 V
上电复位时由硬件自动启动看门狗
下载/编程 停止 重复编程
检测MCU选项 帮助 重复延时 1 秒
每次下载前都重新装载目标文件 重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件 EEPROM文件 串口助手 HID助手 Keil仿真设置 STC硬件数学库
添加型号和头文件到Keil中
添加STC仿真器驱动到Keil中
仿真器使用说明
STC8/STC15系列仿真芯片设置
单片机型号 STC8H8K64U
使用串口进行仿真 P3.0/P3.1
使用USB口进行仿真
将所选目标单片机设置为仿真芯片
电脑 RS232 仿真目标CPU
RXD Pin2 P3.1 TXD
TXD Pin3 P3.0 RXD
GND Pin5 GND
单CPU方案仿真器使用方法:
1、首先必须安装仿真驱动, 可通过点击此页面中的"添加MCU型号到Keil中"按钮进行安装
2、准备1片上面按钮所指型号的芯片
正在检测目标单片机 ...
发布项目程序 发布项目帮助 读取本机硬盘号 提示音 成功计数 2641 清零

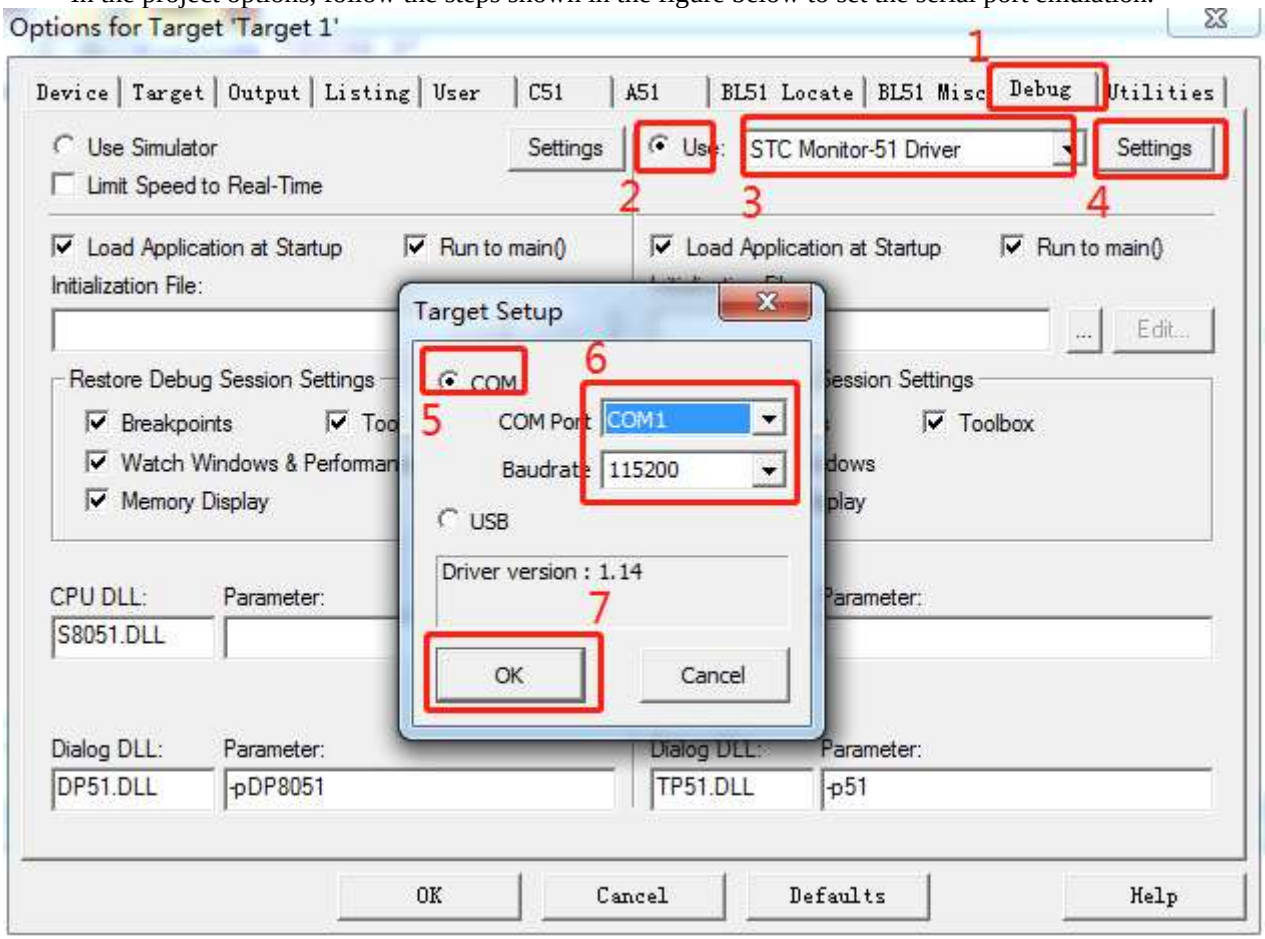
After the download is complete, the emulation chip is completed.

H.4.2 Serial port emulation settings in Keil software

Open the project file in Keil software, and click "Options for ..." in the right-click menu as shown in the figure below.



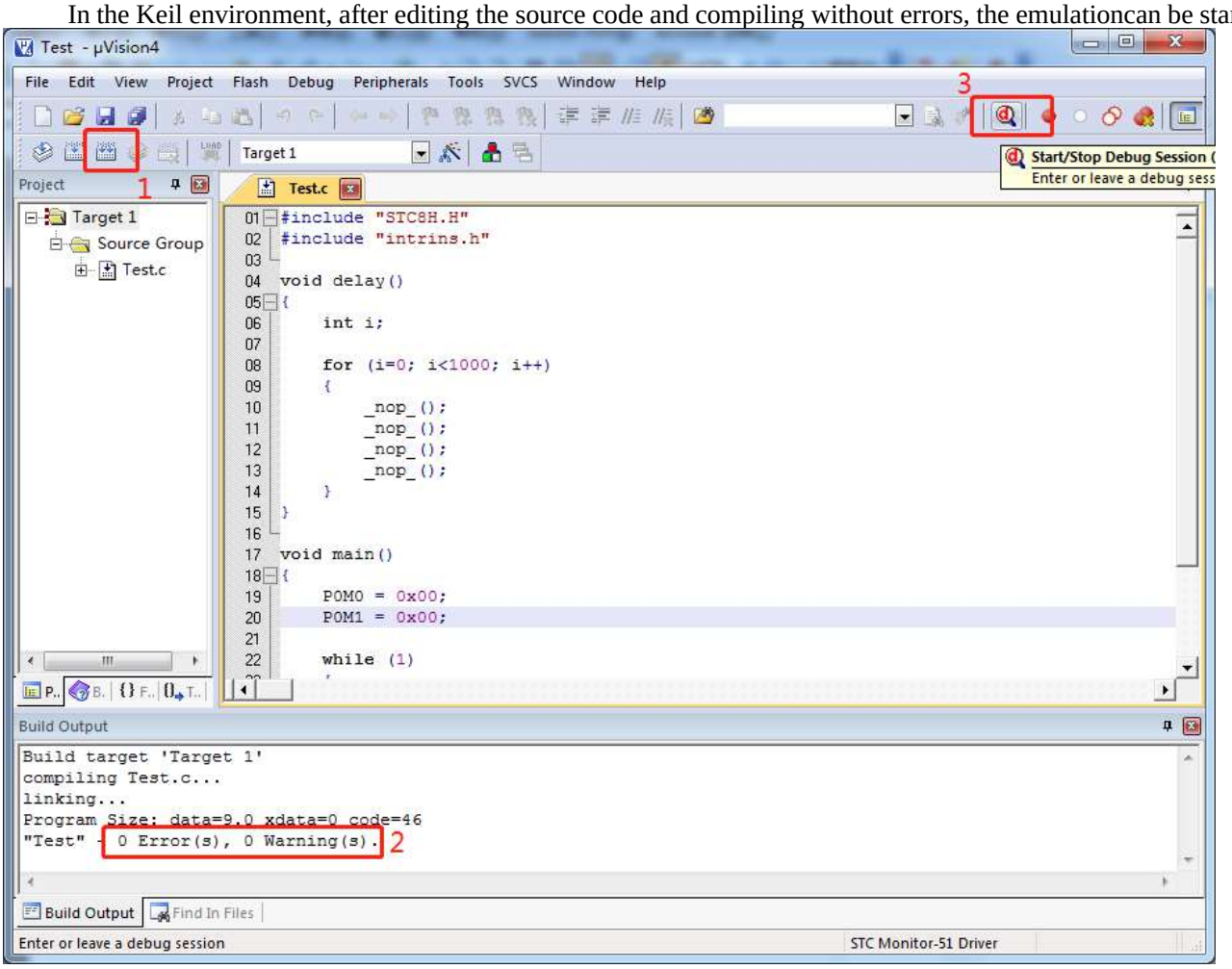
In the project options, follow the steps shown in the figure below to set the serial port emulation.

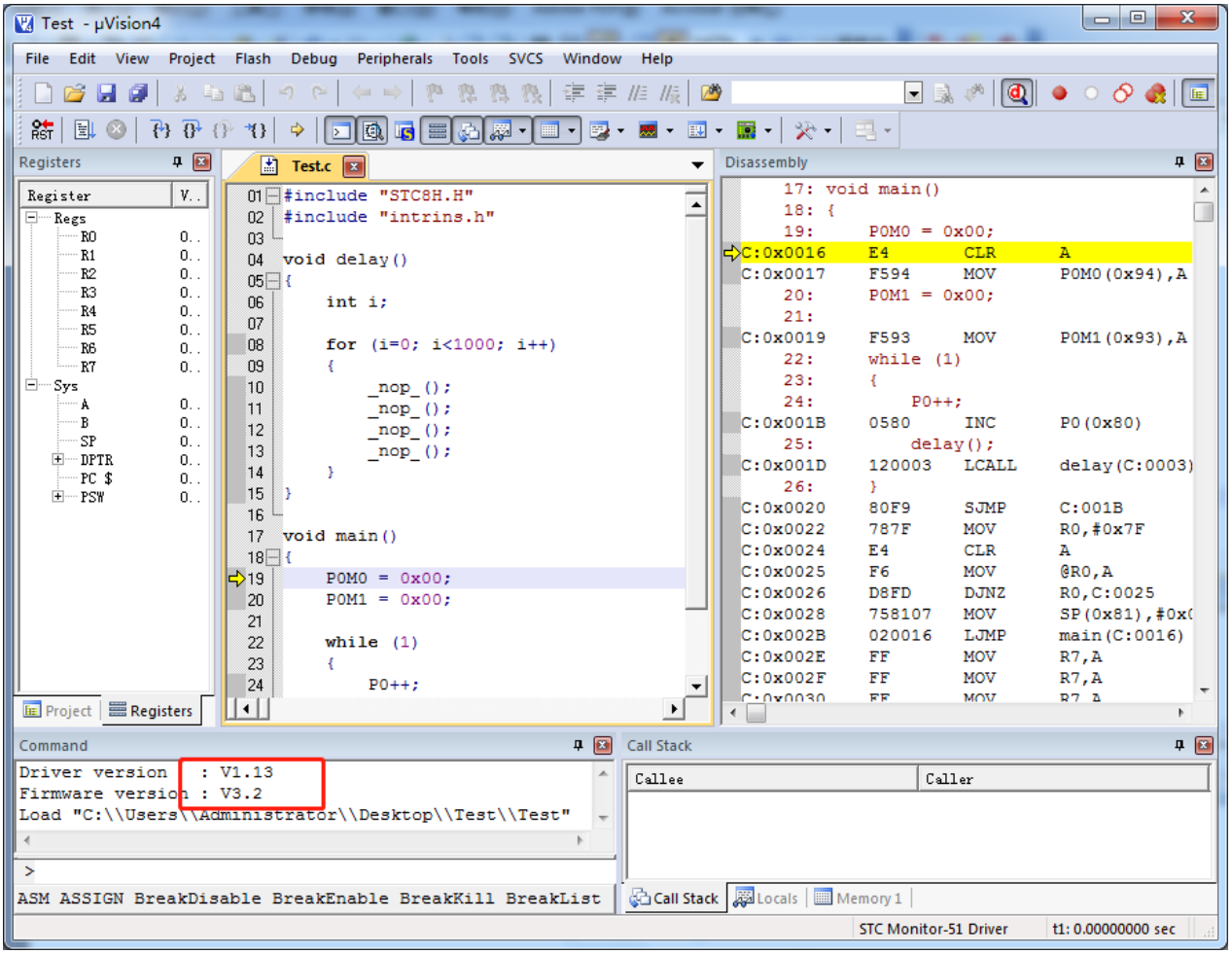


Note: Please select the serial port according to the actual connection, and the baud rate is generally 115200.

H.4.3 Emulation using serial port in Keil software

In the Keil environment, after editing the source code and compiling without errors, the emulationcan be started.





If the chip making and connection are correct, the emulation driver version will be displayed as shown in the figure above, and the user code can be downloaded to the microcontroller correctly, and then debugging functions such as running, single step, and breakpoint can be performed.

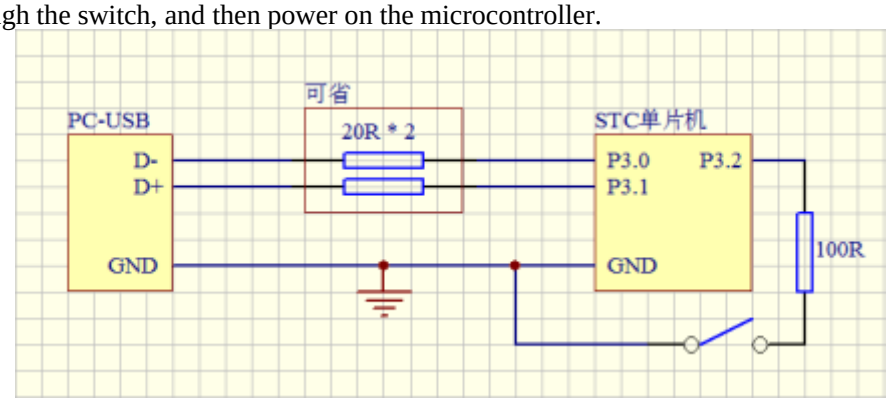
H.5 USB direct emulation (currently only supported by STC8H8K64U-B version chip)

H.5.1 Making a USB emulation chip

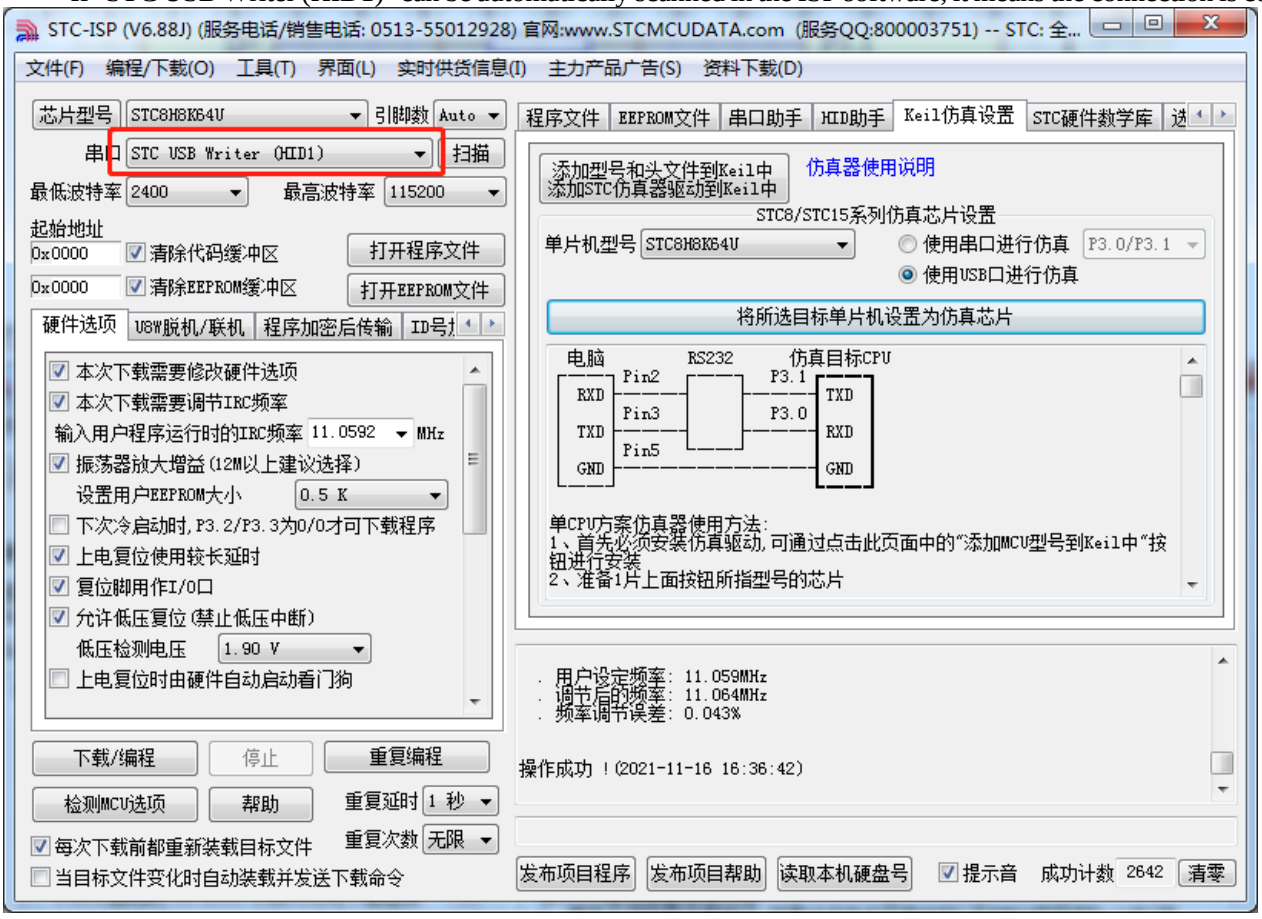
To make a USB emulation chip, you can use the serial ISP to make it according to the steps in Section 4.1, or use the USB-ISP method. This section will introduce how to use the USB-ISP to make it.

The setting steps are as follows:

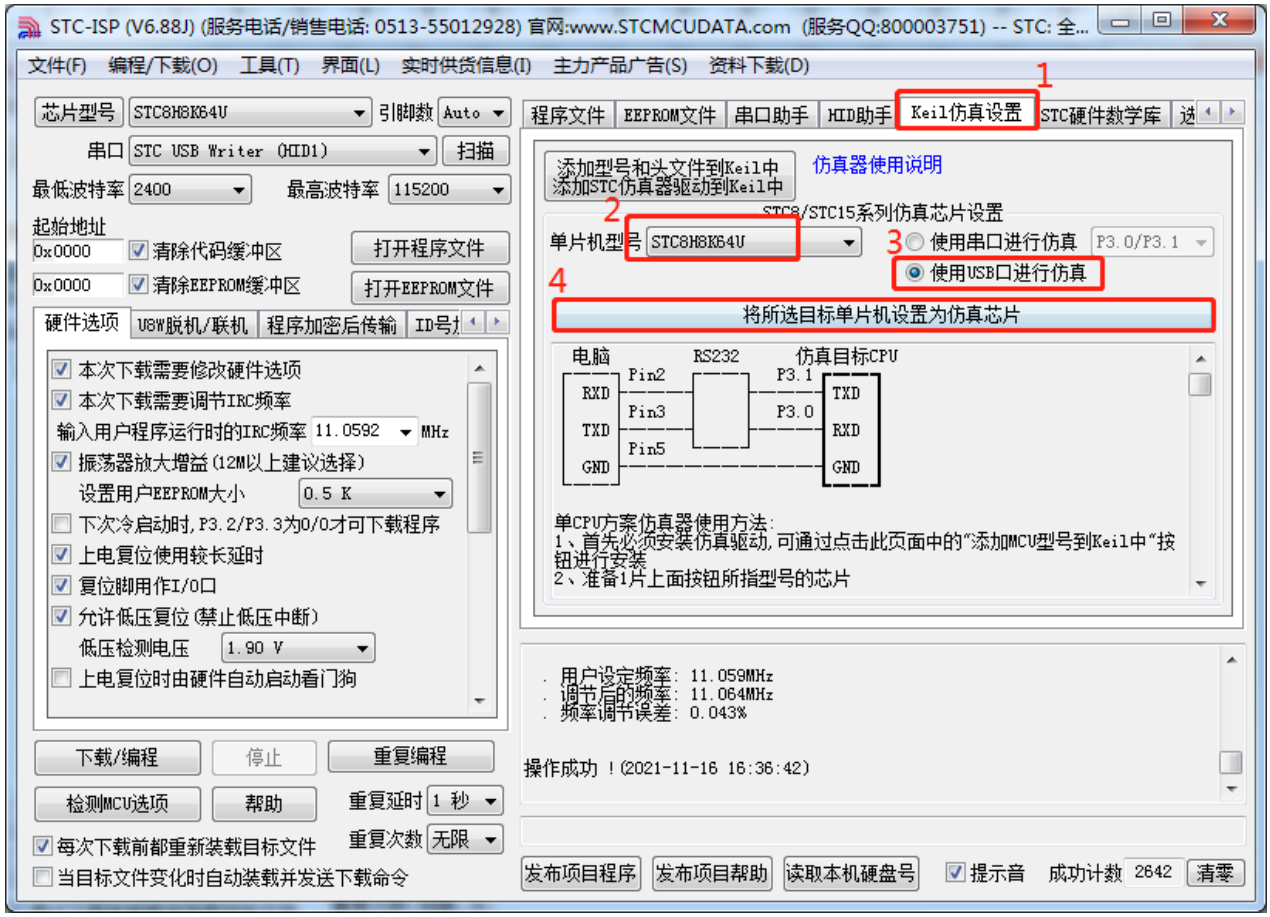
Firstly, connect the target chip to the serial port of the computer as shown in the figure below, and short-circuit P3.2 to GND through the switch, and then power on the microcontroller.



If "STC USB Writer (HID1)" can be automatically scanned in the ISP software, it means the connection is correct.

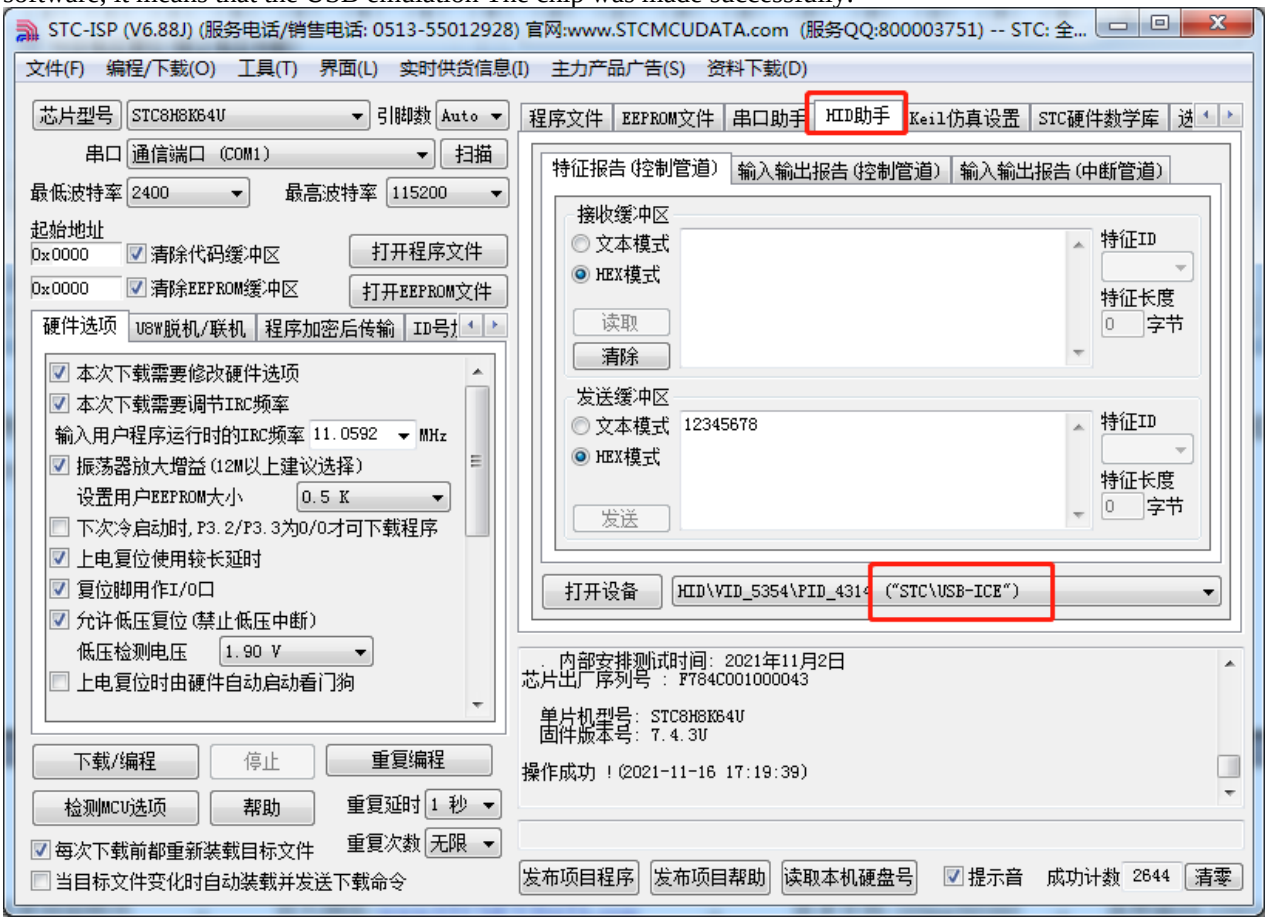


Next, in the STC-ISP download software, follow the steps shown in the figure below to set up the emulation chip.


STC-ISP (V6.88J) (服务电话/销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (服务QQ:800003751) -- STC: 全...
文件(F) 编程/下载(O) 工具(T) 界面(L) 实时供货信息(I) 主力产品广告(S) 资料下载(D)
芯片型号 STC8H8K64U 引脚数 Auto
串口 STC USB Writer (HID1) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 U8W脱机/联机 程序加密后传输 ID号
本次下载需要修改硬件选项
本次下载需要调节IRC频率
输入用户程序运行时的IRC频率 11.0592 MHz
振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
下次冷启动时,P3.2/P3.3为0/0才可下载程序
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 1.90 V
上电复位时由硬件自动启动看门狗
下载/编程 停止 重复编程
检测MCU选项 帮助 重复延时 1 秒
每次下载前都重新装载目标文件 重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件 EEPROM文件 串口助手 HID助手 Keil仿真设置 STC硬件数学库
1
添加型号和头文件到Keil中
添加STC仿真器驱动到Keil中
仿真器使用说明
STC8/STC15系列仿真芯片设置
2
单片机型号 STC8H8K64U
3
使用串口进行仿真 P3.0/P3.1
使用USB口进行仿真
4
将所选目标单片机设置为仿真芯片
电脑 RS232 仿真目标CPU
RXD Pin2 P3.1 TXD
TXD Pin3 P3.0 RXD
GND Pin5 GND
单CPU方案仿真器使用方法:
1、首先必须安装仿真驱动,可通过点击此页面中的"添加MCU型号到Keil中"按钮进行安装
2、准备1片上面按钮所指型号的芯片
用户设定频率: 11.059MHz
调节后的频率: 11.064MHz
频率调节误差: 0.043%
操作成功 !(2021-11-16 16:36:42)
发布项目程序 发布项目帮助 读取本机硬盘号 提示音 成功计数 2642 清零

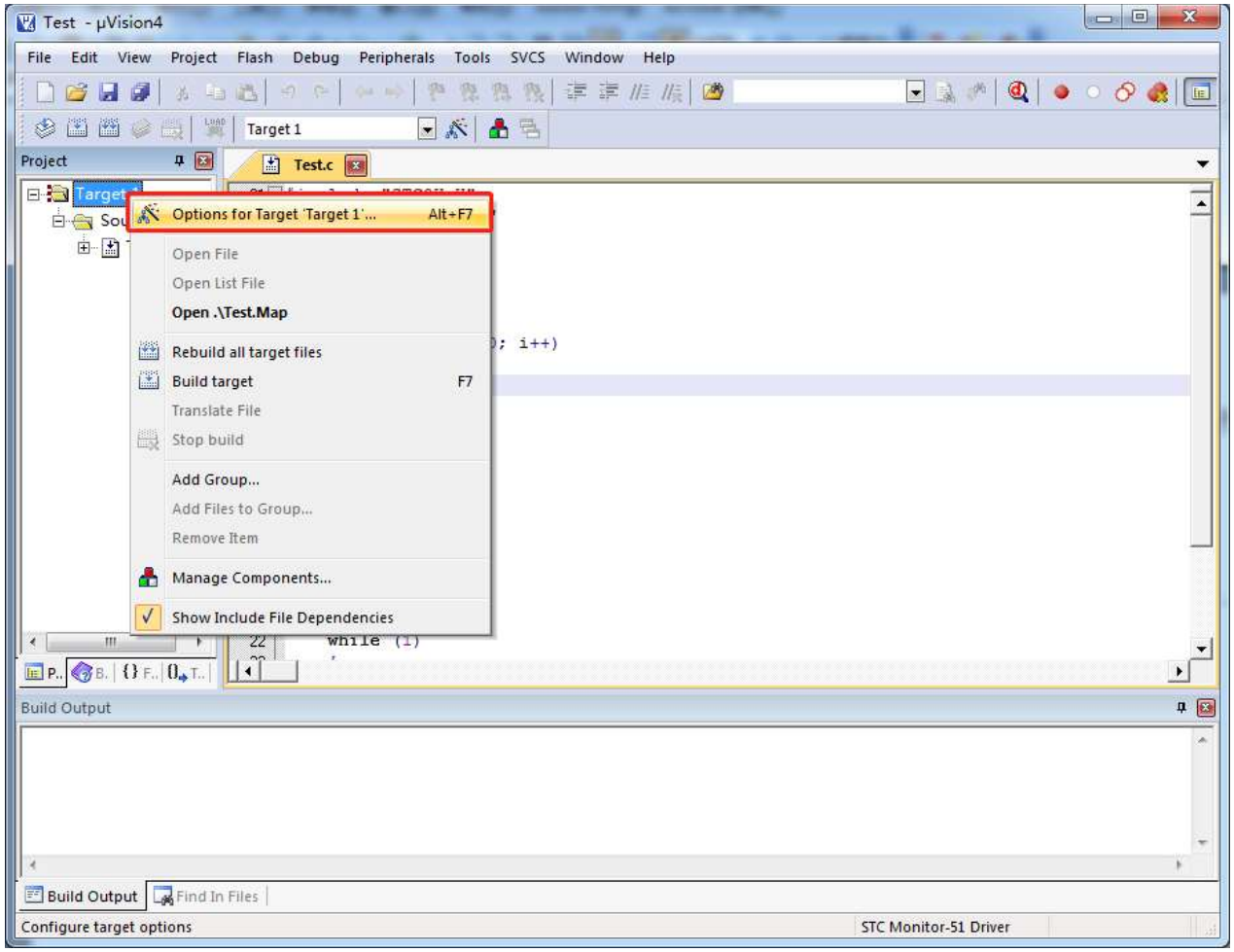
After the download is complete, it will be as shown below.

After the production is completed, the grounding switch of the P3.2 port needs to be disconnected, and the MCU needs to be powered on again. If the "STC\USB-ICE" device can be detected in the "HID Assistant" in the downloaded software, it means that the USB emulation The chip was made successfully.

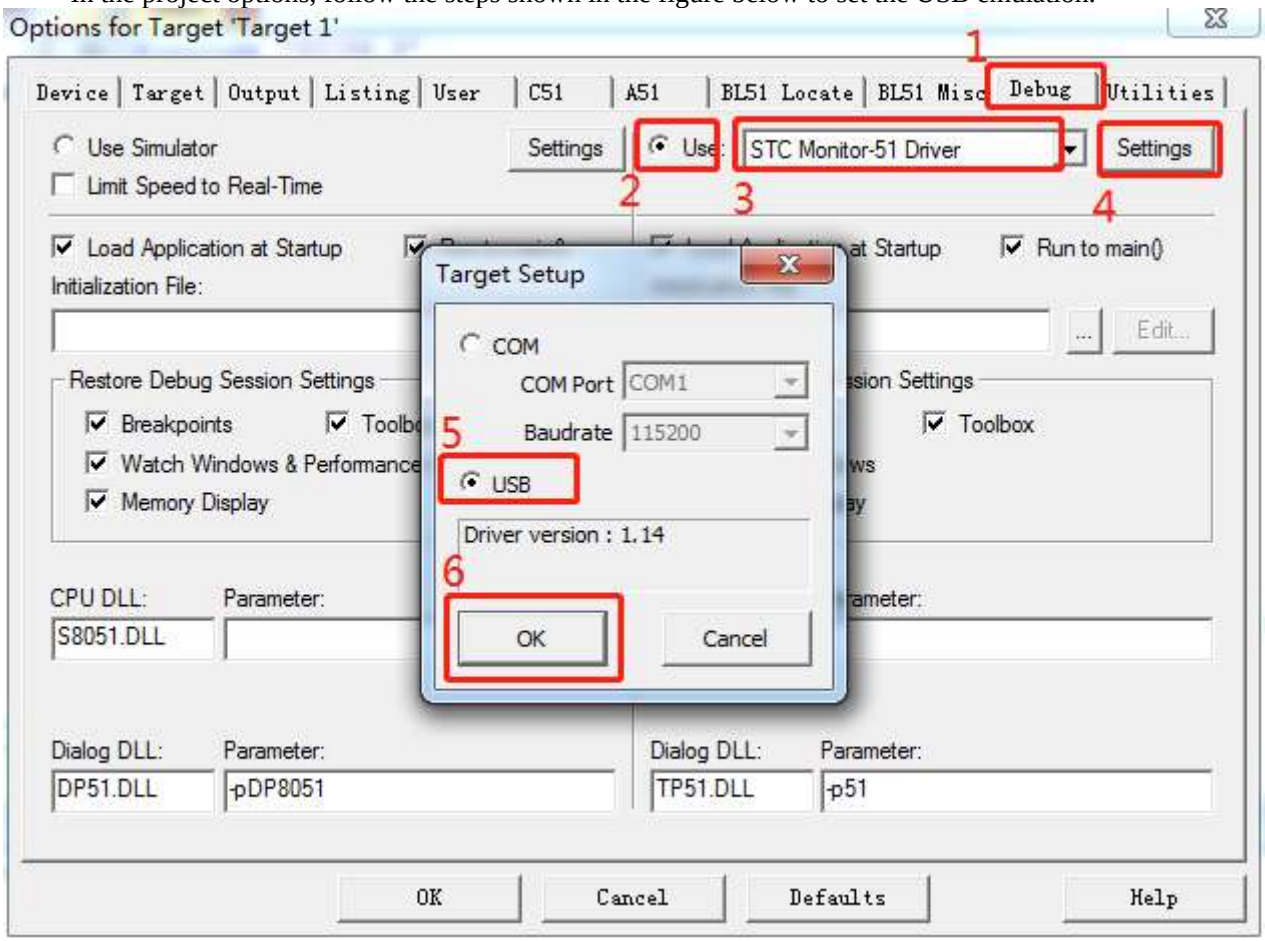


H.5.2 USB emulation settings in Keil software

Open the project file in Keil software, and click "Options for ..." in the right-click menu as shown in the figure below.

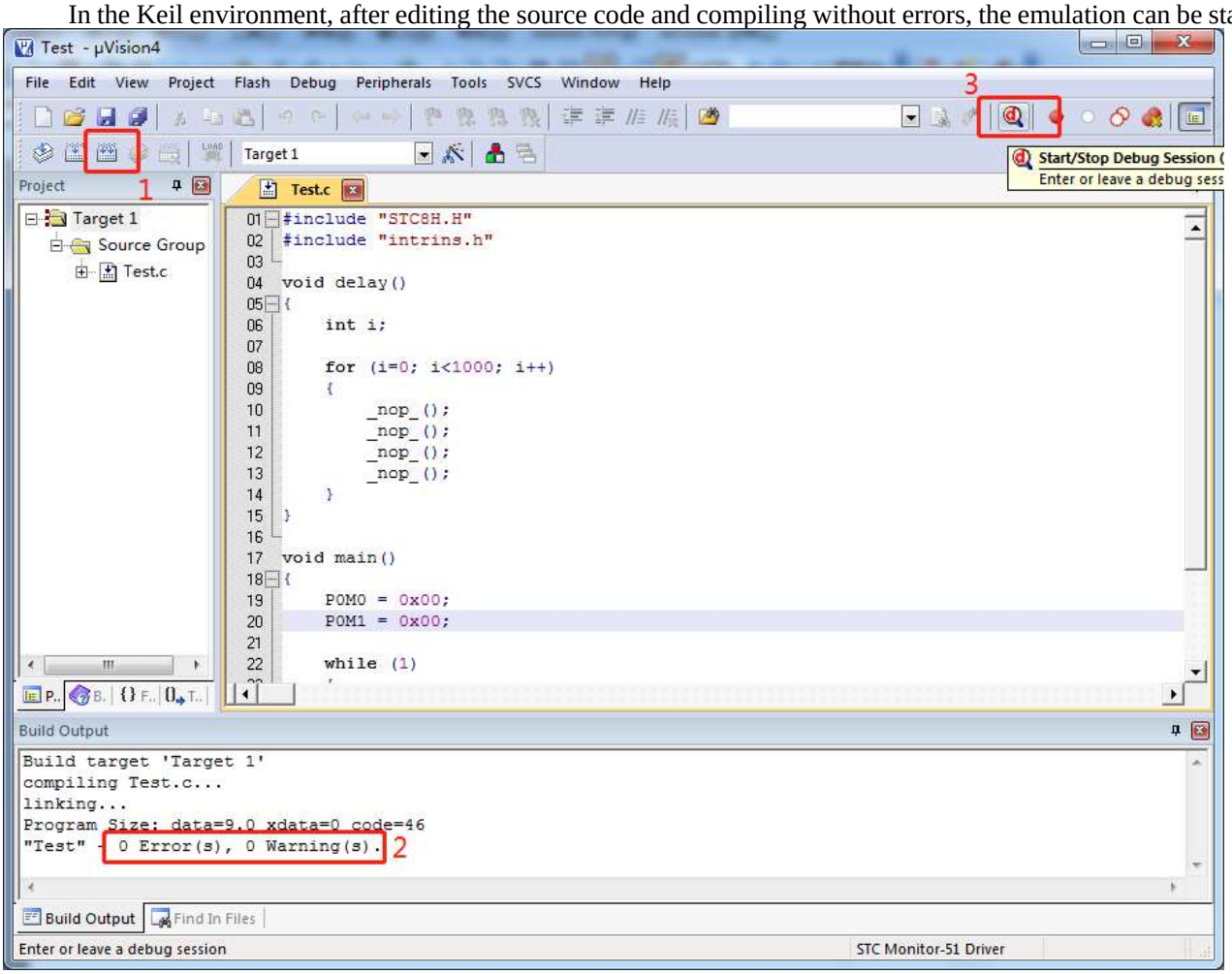


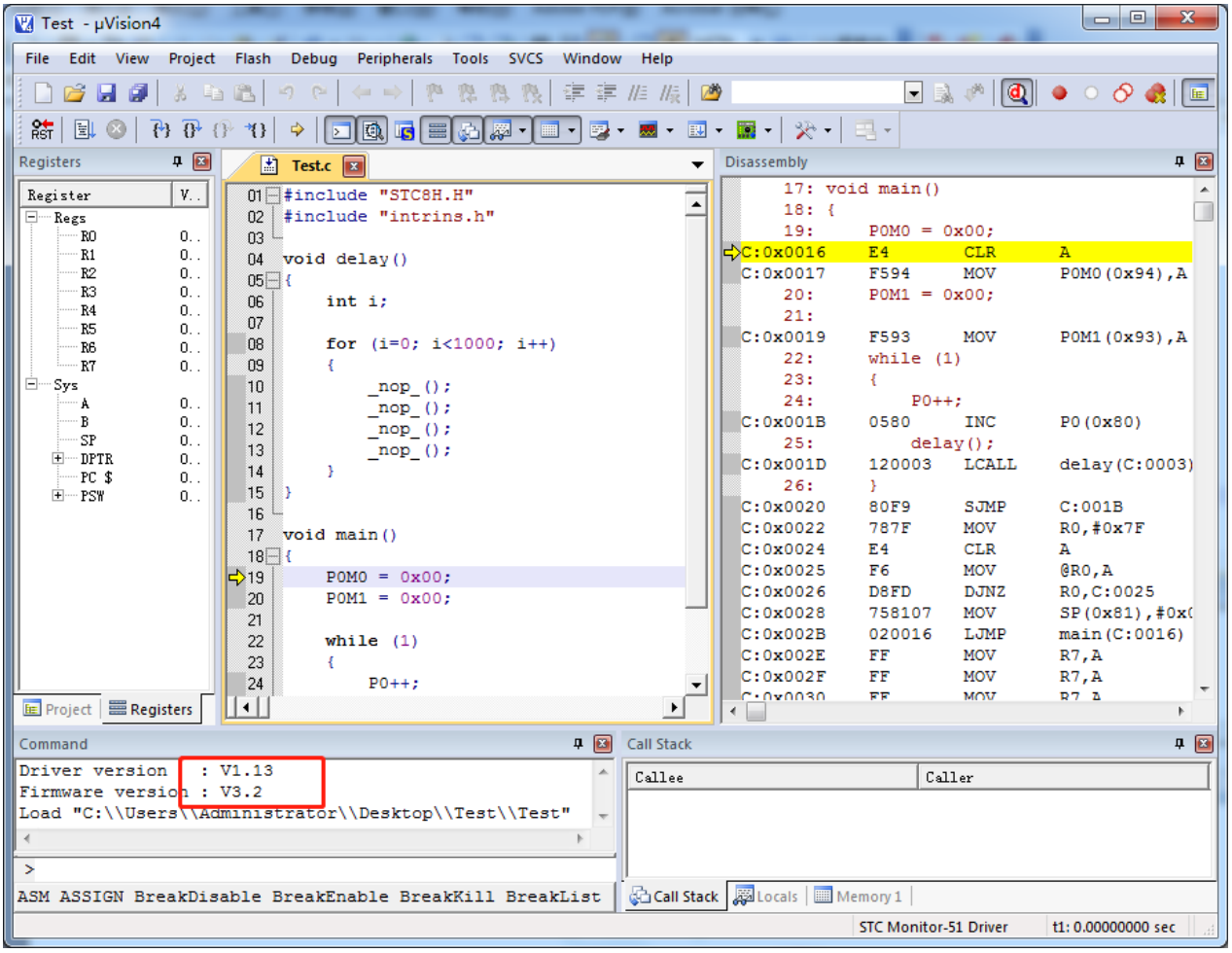
In the project options, follow the steps shown in the figure below to set the USB emulation.



H.5.3 Emulation using USB in Keil software

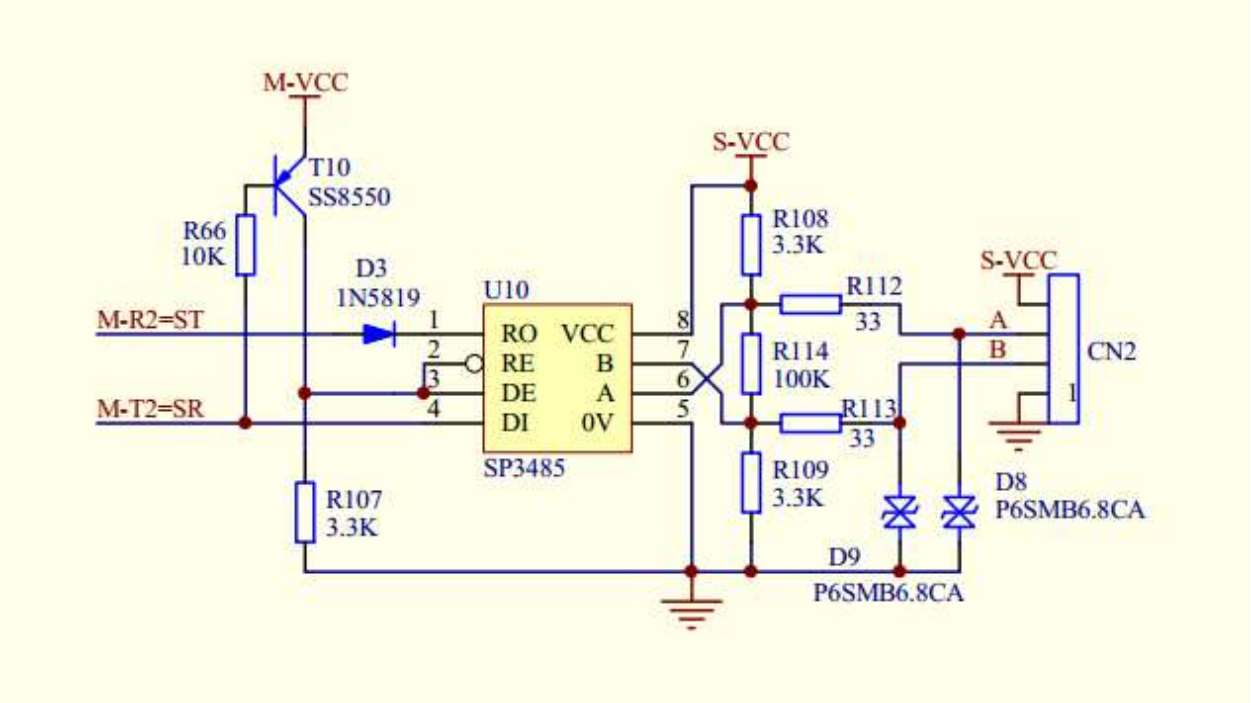
In the Keil environment, after editing the source code and compiling without errors, the emulation can be started.





If the chip making and connection are correct, the emulation driver version will be displayed as shown in the figure above, and the user code can be downloaded to the microcontroller correctly, and then debugging functions such as running, single step, and breakpoint can be performed.

Appendix I Partial Circuit of RS485 in U8W Download Tool



BOM list:

Label	Model	Package	Note
U10	SP3485EN	SOP8	RS485 chip
R66	10K	0603	Resistor
R107	3.3K	0603	Resistor
R108	3.3K	0603	Resistor
R109	3.3K	0603	Resistor
R112	33R	0603	Resistor
R113	33R	0603	Resistor
R114	100K	0603	Resistor
T10	SS8550	SOT-23	PNP Triode
D3	1N5819	0603	Schottky diode
D8	P6SMB6.8CA	DO-214AA	TVS diode
D9	P6SMB6.8CA	DO-214AA	TVS diode
CN2		SIP4	Communication Interface

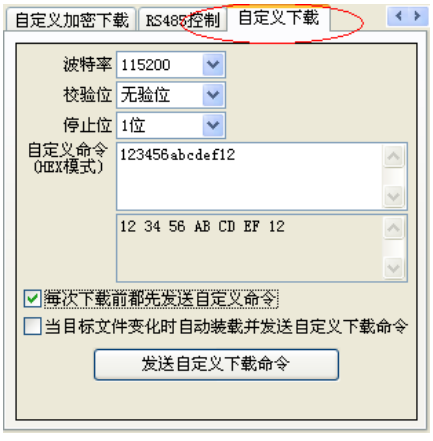
Appendix J ISP Download Starts Automatically After Receiving User Command While Running User Program (no Power-down)

"User-defined download" and "user-defined encrypted download" are two completely different functions. Compared with the function of user-defined encrypted download, the function of user-defined download is simpler.

The specific functions is: Before the computer or offline download board starts to send the real ISP download programming handshake command, it first sends a user-defined string of commands (for this string of serial commands, user can set the baud rate, parity, and stop bits), and then immediately sends the ISP download programming handshake command.

The function of "user-defined download" is mainly used in the early development stage of the project, which can download user code without power-off (without re-power-on to the target chip). The specific implementation method is: User needs to add a piece of code to detect the custom command in user program. When the command is detected, execute the assembly code of "MOV IAP_CONTR, # 60H" or the C language code of "IAP_CONTR = 0x60;" , MCU will reset to ISP area to execute ISP code automatically.

As shown in the figure below, set the custom command sequence with a baud rate of 115200, no parity bit, and one stop bit: 0x12, 0x34, 0x56, 0xAB, 0xCD, 0xEF, 0x12. When the option "Send custom commands before each download" is checked, the user-defined download function can be implemented.



Click "Send user-defined download command" or click the "Download / Program" button in the lower left corner of the window, the application will send the serial data as shown below.


Function
Data (Hex)
IRP_MJ_CREATE Port Opened - stc-isp-15xx.exe
IOCTL_SERIAL_SET_BA... Baud Rate: 115200
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: No, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 12
IRP_MJ_WRITE Length: 0001, Data: 34
IRP_MJ_WRITE Length: 0001, Data: 56
IRP_MJ_WRITE Length: 0001, Data: AB
IRP_MJ_WRITE Length: 0001, Data: CD
IRP_MJ_WRITE Length: 0001, Data: EF
IRP_MJ_WRITE Length: 0001, Data: 12
IOCTL_SERIAL_SET_BA... Baud Rate: 2400
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: Even, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
1. 用户的串口设置
2. 用户的自定义命令
3. isp的串口设置及命令
STC-ISP (V6.38) 官方网站:www.STCMCU.com
单片机型号
引脚数
串口号 通讯端口 (COM1)
扫描
最低波特率 2400
最高波特率 115200
起始地址
0x0000
清除代码缓冲区
打开程序文件
自定义加密下载
RS485控制
自定义下载
波特率 115200
校验位 无校验位
停止位 1位
自定义命令 (HEX模式)
12 34 56 AB CD EF 12
每次下载前都先发送自定义命令
当目标文件变化时自动装载并发送自定义下载命令
发送自定义下载命令

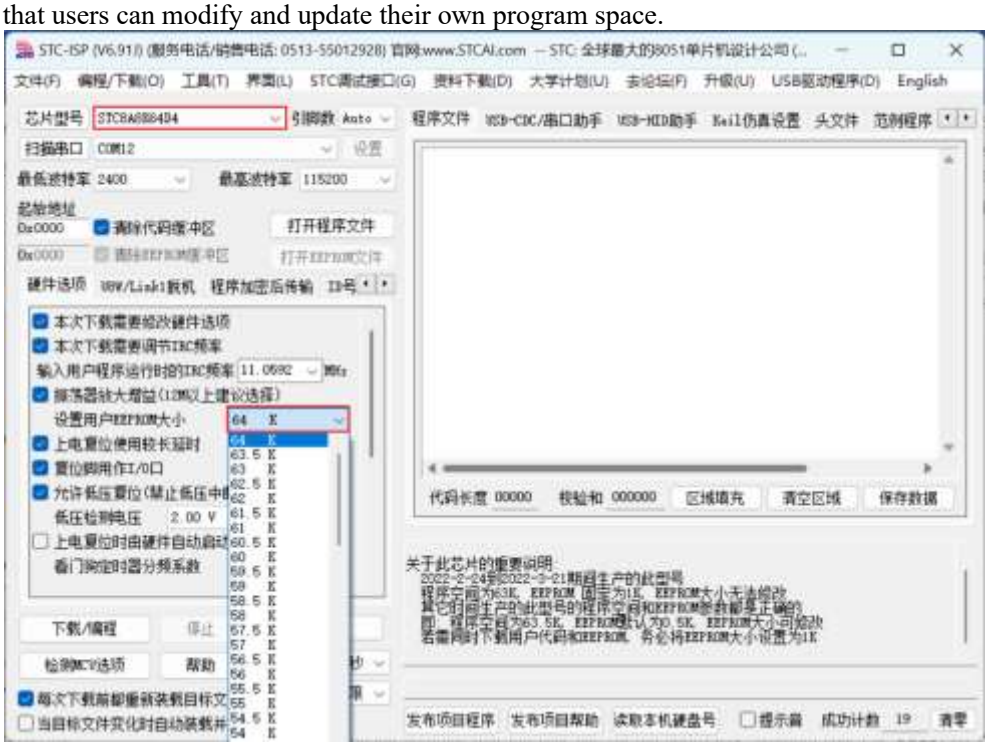
Appendix K Use STC's IAP series MCU to develop your own ISP program

With the continuous development of IAP (In-Application-Programming) technology in the field of single-chip microcomputers, it has brought great convenience to the application system program code upgrade. STC's serial ISP (In-System-Programming) program uses the IAP function to upgrade the user's program online, but for the sake of user code safety, neither the underlying code nor the upper application is open source. For this reason, STC launched With the IAP series single-chip microcomputer, that is, the Flash space of the entire MCU, users can rewrite in their own programs, so that the idea that users need to develop their own ISP programs can be realized.

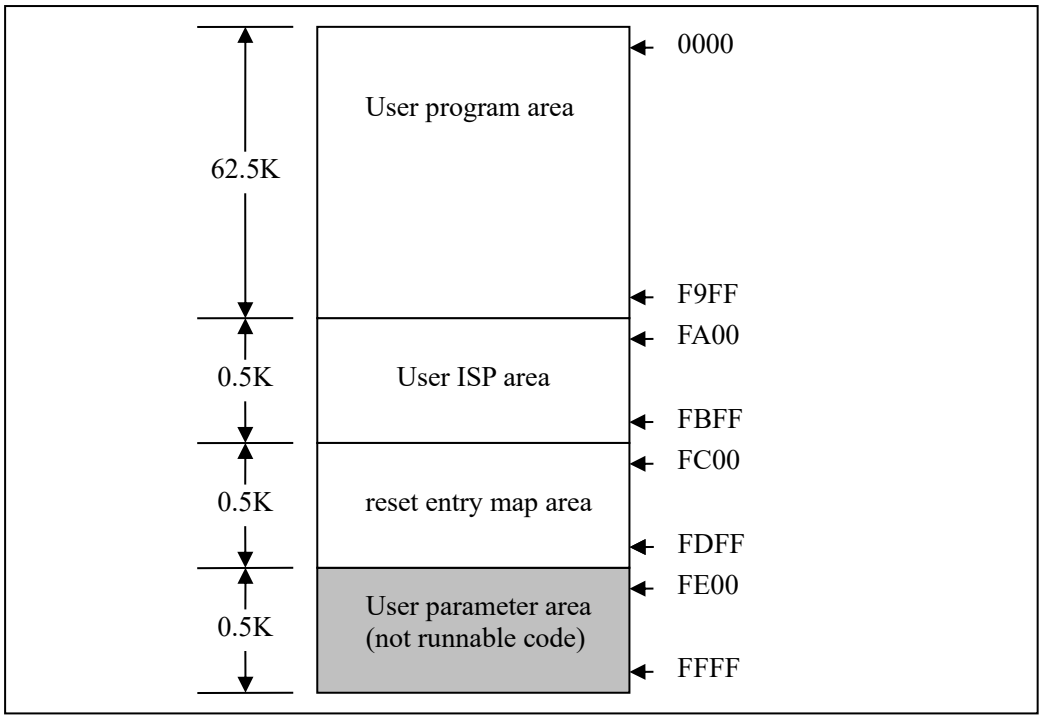
All the models in the STC8A8k64D4 series microcontrollers that can customize the EEPROM size during ISP download are IAP series.Film machine. At present, the STC8A series have the following types of microcontrollers as the IAP series: STC8A8k64D4. This article uses STC8A8k64D4,As an example, the method of using STC's IAP MCU to develop user's own ISP program is explained in detail, and the assembly and C source code based on Keil environment are given.

The first step: internal FLASH planning

Since the EEPROM of the IAP microcontroller of the STC8A8k64D4 series is set by the user during ISP download, if the user needs to implement his own ISP, when downloading the user's own ISP program, the user needs to follow the figure below to set all 64K Set it to EEPROM, so that the user program space and EEPROM space are completely overlapped, so that users can modify and update their own program space.

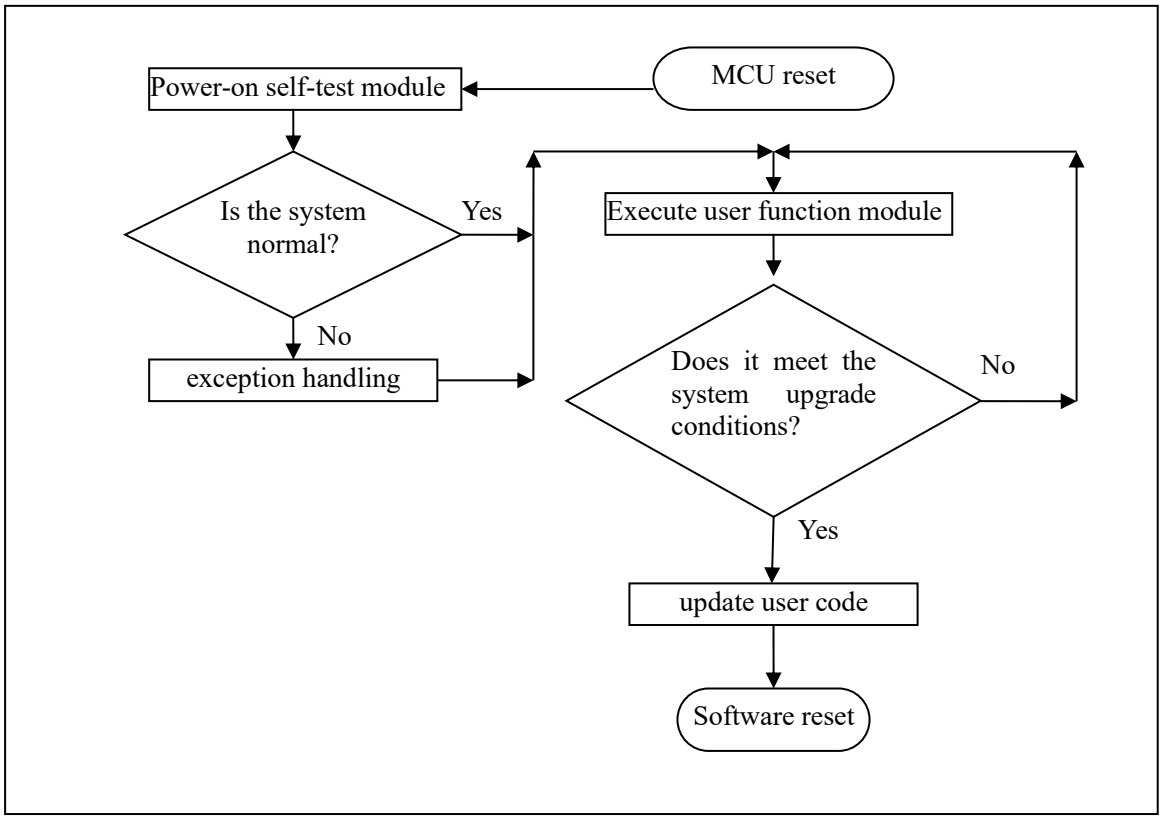


The following assumes that the user has set the entire 64K program space as EEPROM. Now the entire 64K program space is divided as follows:



In the FLASH space, the continuous 62.5K bytes of space starting from address 0000H is the user program area. When the specific download conditions are met, the user is required to jump the PC to the user ISP program area. At this time, the user program area can be erased and rewritten to achieve the purpose of updating the user program

The second step, the basic framework of the program



The third step, the firmware program description of the lower computer

The firmware program of the lower computer includes two parts: ISP (ISP code) and AP (user code)

ISP Code (assembly code)

```
; Operating frequency for test is 11.0592MHz

UARTBAUD    EQU     0FFE8H      ;Define the serial port baud rate   (65536-11059200/4/115200)

AUXR        DATA    08EH        ;Additional Function Control Register
WDT_CONTR   DATA    0C1H        ;Watchdog Control Register
IAP_DATA    DATA    0C2H        ;IAP data register
IAP_ADDRH   DATA    0C3H        ;IAP High Address Register
IAP_ADDRL   DATA    0C4H        ;IAP Low Address Register
IAP_CMD     DATA    0C5H        ;IAP Command Register
IAP_TRIG    DATA    0C6H        ;IAP Command Trigger Register
IAP_CONTR   DATA    0C7H        ;IAP Control Register
IAP_TPS     DATA    0F5H        ;IAP latency control register

ISPCODE     EQU     0FA00H      ;ISP module entry address (1 page), also external interface address
APENTRY     EQU     0FC00H      ; Application entry address data (1 page)

            ORG     0000H

            LJMP    ISP_ENTRY   ;System reset entry

RESET:
            MOV     SCON,#50H   ;Set serial port mode (8 data bits, no parity bit)
            MOV     AUXR,#40H   ;Timer 1 is in 1T mode
            MOV     TMOD,#00H   ;Timer 1 works in mode 0 (16-bit reload)
            MOV     TH1,#HIGH UARTBAUD  ;set reload value
            MOV     TL1,#LOW UARTBAUD
            SETB    TR1         ;start timer 1
NEXT1:
            MOV     R0,#16
NEXT2:
            JNB     RI,$        ;Waiting for serial data
            CLR     RI
            MOV     A,SBUF
            CJNE    A,#7FH,NEXT1    ;Determine whether it is 7F
            DJNZ    R0,NEXT2
            LJMP    ISP_DOWNLOAD    ;Jump to download interface

            ORG     ISPCODE

ISP_DOWNLOAD:
            CLR     A
            MOV     PSW,A       ;ISP module uses group 0 registers
            MOV     IE,A        ;Disable all interrupts
            CLR     RI          ;Clear serial port receive flag
            SETB    TI          ; Set serial port send flag
            CLR     TR0
            MOV     SP,#5FH     ;set stack pointer

            MOV     A,#5AH      ;Return 5A 55 to PC, indicating ISP erase module is ready
            LCALL   ISP_SENDUART
            MOV     A,#055H
            LCALL   ISP_SENDUART
            LCALL   ISP_RECVACK ;Receive response data
```

```
                MOV     IAP_ADDRL,#0            ;First write the “LJMP ISP_ENTRY” instruction at the starting
address of page 2
                MOV     IAP_ADDRH,#02H
                LCALL   ISP_ERASEIAP
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP          ;Programming User Code Reset Vector Code
                MOV     A,#HIGH                 ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;Programming User Code Reset Vector Code
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;Programming User Code Reset Vector Code

                MOV     IAP_ADDRL,#0            ;User code address starts from 0
                MOV     IAP_ADDRH,#0
                LCALL   ISP_ERASEIAP
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP          ;Programming User Code Reset Vector Code
                MOV     A,#HIGH                 ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;Programming User Code Reset Vector Code
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;Programming User Code Reset Vector Code

                MOV     IAP_ADDRL,#0            ;new code buffer address
                MOV     IAP_ADDRH,#02H
                MOV     R7,#124                 ;Erase 62.5K bytes
ISP_ERASEAP:
                LCALL   ISP_ERASEIAP
                INC     IAP_ADDRH               ;target address+512
                INC     IAP_ADDRH
                DJNZ    R7,ISP_ERASEAP          ;Determine if erasing is complete

                MOV     IAP_ADDRL,#LOW APENTRY
                MOV     IAP_ADDRH,#HIGH APENTRY
                LCALL   ISP_ERASEIAP

                MOV     A,#5AH                  ;Return 5A A5 to PC, indicating that the ISP programming module
is ready
                LCALL   ISP_SENDUART
                MOV     A,#0A5H
                LCALL   ISP_SENDUART
                LCALL   ISP_RECVACK             ;Receive response data

                LCALL   ISP_RECVUART            ;Receive length high byte
                MOV     R0,A
                LCALL   ISP_RECVUART            ;Receive length low byte
                MOV     R1,A
                CLR     C                       ; total length -3
                MOV     A,#03H
                SUBB    A,R1
                MOV     DPL,A
                CLR     A
                SUBB    A,R0
                MOV     DPH,A                   ;Total length complement stored in DPTR

                LCALL   ISP_RECVUART            ;Map user code reset entry code to map area
                LCALL   ISP_PROGRAMIAP          ;0000
                LCALL   ISP_RECVUART
                LCALL   ISP_PROGRAMIAP          ;0001
```

```
                LCALL   ISP_RECVUART
                LCALL   ISP_PROGRAMIAP          ;0002

                MOV     IAP_ADDRL,#03H          ;User code start address
                MOV     IAP_ADDRH,#00H
ISP_PROGRAMNEXT:
                LCALL   ISP_RECVUART            ;receive code data
                LCALL   ISP_PROGRAMIAP          ;Program the data into the user code area
                INC     DPTR
                MOV     A,DPL
                ORL     A,DPH
                JNZ     ISP_PROGRAMNEXT         ;length detection

ISP_SOFTRESET:
                MOV     IAP_CONTR,#20H          ;Software reset system
                SJMP    $


ISP_ENTRY:
                MOV     WDT_CONTR,#17H          ;clear watchdog
                MOV     IAP_CONTR,#80H          ;Enable IAP function
                MOV     IAP_TPS,#11             ;Set the IAP wait time parameter
                MOV     IAP_ADDRL,#LOW ISP_DOWNLOAD
                MOV     IAP_ADDRH,#HIGH ISP_DOWNLOAD
                MOV     IAP_DATA,#00H           ;Test data 1
                MOV     IAP_CMD,#1              ;Read command
                MOV     IAP_TRIG,#5AH           ;Triger ISP command
                MOV     IAP_TRIG,#0A5H
                MOV     A,IAP_DATA
                CJNE    A,#0E4H,ISP_ENTRY       ; If the data cannot be read, wait for the voltage to stabilize
                INC     IAP_ADDRL               ;Test address FC01H
                MOV     IAP_DATA,#45H           ;Test data 2
                MOV     IAP_CMD,#1              ;Read command
                MOV     IAP_TRIG,#5AH           ;Triger ISP command
                MOV     IAP_TRIG,#0A5H
                MOV     A,IAP_DATA
                CJNE    A,#0F5H,ISP_ENTRY       ; If the data cannot be read, wait for the voltage to stabilize

                MOV     SCON,#50H               ;Set serial port mode (8 data bits, no parity bit)
                MOV     AUXR,#40H               ;Timer 1 is in 1T mode
                MOV     TMOD,#00H               ;Timer 1 works in mode 0 (16-bit reload)
                MOV     TH1,#HIGH UARTBAUD      ;set reload value
                MOV     TL1,#LOW UARTBAUD
                SETB    TR1                     ;start timer 1
                SETB    TR0

                LCALL   ISP_RECVUART            ; Check if there is serial data
                JC      GOTOAP
                MOV     R0,#16
ISP_CHECKNEXT:
                LCALL   ISP_RECVUART            ; receive sync data
                JC      GOTOAP
                CJNE    A,#7FH,GOTOAP           ;Determine whether it is 7F
                DJNZ    R0,ISP_CHECKNEXT
                MOV     A,#5AH                  ; Return 5A 69 to PC, indicating ISP module is ready
                LCALL   ISP_SENDUART
                MOV     A,#69H
                LCALL   ISP_SENDUART
```

```
                LCALL   ISP_RECVACK             ;Receive response data
                LJMP    ISP_DOWNLOAD            ;Jump to download interface
GOTOAP:
                CLR     A                       ; Reset SFR to reset value
                MOV     TCON,A
                MOV     TMOD,A
                MOV     TL0,A
                MOV     TH0,A
                MOV     TL1,A
                MOV     TH1,A
                MOV     SCON,A
                MOV     AUXR,A
                LJMP    APENTRY                 ; Run the user program normally

ISP_RECVACK:
                LCALL   ISP_RECVUART
                JC      GOTOAP
                XRL     A,#7FH
                JZ      ISP_RECVACK             ; skip sync data
                CJNE    A,#25H,GOTOAP           ; Response data 1 detection
                LCALL   ISP_RECVUART
                JC      GOTOAP
                CJNE    A,#69H,GOTOAP           ; Response data 2 detection
                RET

ISP_RECVUART:
                CLR     A
                MOV     TL0,A                   ; Initialize timeout timer
                MOV     TH0,A
                CLR     TF0
                MOV     WDT_CONTR,#17H          ;clear watchdog
ISP_RECVWAIT:
                JBC     TF0,ISP_RECVTIMEOUT     ; Timeout detection
                JNB     RI,ISP_RECVWAIT         ; wait for receive to complete
                MOV     A,SBUF                  ; Read serial data
                CLR     RI                      ;Clear flag
                CLR     C                       ; Receive serial data correctly
                RET
ISP_RECVTIMEOUT:
                SETB    C                       ; timeout
                RET

ISP_SENDUART:
                MOV     WDT_CONTR,#17H          ;clear watchdog
                JNB     TI,ISP_SENDUART         ; Wait for the previous data transmission to complete
                CLR     TI                      ; Clear flag
                MOV     SBUF,A                  ; send current data
                RET

ISP_ERASEIAP:
                MOV     WDT_CONTR,#17H          ;clear watchdog
                MOV     IAP_CMD,#3              ; Erase command
                MOV     IAP_TRIG,#5AH           ; Trigger ISP command
                MOV     IAP_TRIG,#0A5H
                NOP
                NOP
                NOP
                NOP
```

```
            RET

ISP_PROGRAMIAP:
            MOV     WDT_CONTR,#17H      ;clear watchdog
            MOV     IAP_CMD,#2          ; programming command
            MOV     IAP_DATA,A          ; send the current data to IAP data register
            MOV     IAP_TRIG,#5AH       ; Trigger ISP command
            MOV     IAP_TRIG,#0A5H
            NOP
            NOP
            NOP
            NOP
            MOV     A,IAP_ADDRL         ;IAP address +1
            ADD     A,#01H
            MOV     IAP_ADDRL,A
            MOV     A,IAP_ADDRH
            ADDC    A,#00H
            MOV     IAP_ADDRH,A
            RET


            ORG     APENTRY
            LJMP    RESET


            END
```

The ISP code includes the following external interface modules:
ISP_DOWNLOAD: program download entry address, absolute address FA00H
ISP_ENTRY: Power-on system self-check program (automatically called by the system)
For the user program, the user only needs to jump the PC value to ISPPROGRAM (ie FA00H, Absolute address), you can update the code.

User codes (C language code)

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"

#define   FOSC              11059200L           // system clock frequency
#define   BAUD              (65536 - FOSC/4/115200)     //Define the serial port baud rate
#define   ISPPROGRAM        0xfa00              //ISP download program entry address

sfr       AUXR      =       0x8e;               // Baud Rate Generator Control Register
sfr       P1M0      =       0x92;
sfr       P1M1      =       0x91;

void (*IspProgram)() = ISPPROGRAM;              // define pointer function
char cnt7f;                                     //Isp_Check Internally used variables

void uart() interrupt 4                         // UART Interrupt Service Routine
{
    if (TI) TI = 0;                             // send complete interrupt
    if (RI)                                     // Receive complete interrupt
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
```

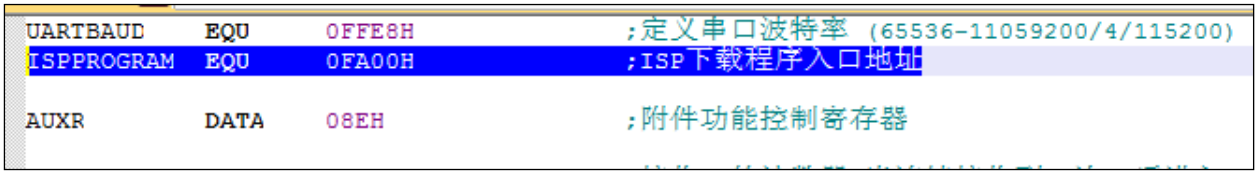
```
                if (cnt7f >= 16)
                {
                    IspProgram();                   // Invoke the download module (****important statement ****)
                }
            }
            else
            {
                cnt7f = 0;
            }
            RI = 0;                                 // Clear the reception completion flag
        }
    }

    void main()
    {
        SCON = 0x50;                                // Define the serial port mode as 8-bit, variable baud rate, no parity
    bit
        AUXR = 0x40;
        TH1  = BAUD >> 8;
        TL1  = BAUD;
        TR1  = 1;
        ES = 1;                                     //Enable UART interrupt
        EA = 1;                                     //Enable CPU interrupt

        P1M0 = 0;
        P1M1 = 0;

        while (1)
        {
            P1++;
        }
    }
```

User codes (assembly code)

```
; Operating frequency for test is 11.0592MHz

UARTBAUD    EQU     0FFE8H      ;Define the serial port baud rate   (65536-11059200/4/115200)
ISPPROGRAM  EQU     0FA00H      ; ISP download program entry address

AUXR        DATA    08EH        ; Additional Function Control Register

CNT7F       DATA    60H         ; Receive 7F counter

            ORG     0000H
            LJMP    START       ;System reset entry

            ORG     0023H
            LJMP    UART_ISR    ; UART interrupt entry

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            JNB     TI,CHECKRI  ; Detect transmission interruption
            CLR     TI          ; clear flag
CHECKRI:
            JNB     RI,UARTISR_EXIT ; Detect receive interruption
            CLR     RI          ; clear flag
```

```
                MOV     A,SBUF
                CJNE    A,#7FH,ISNOT7F
                INC     CNT7F
                MOV     A,CNT7F
                CJNE    A,#16,UARTISR_EXIT
                LJMP    ISPPROGRAM          ; Invoke the download module (****important statement ****)
ISNOT7F:
                MOV     CNT7F,#0
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

START:
                MOV     R0,#7FH             ;Clear RAM
                CLR     A
                MOV     @R0,A
                DJNZ    R0,$-1
                MOV     SP,#7FH             ; Initialize SP

                MOV     SCON,#50H           ; Set UART mode (8-bit data, variable baud rate, no parity bit)
                MOV     AUXR,#15H           ; BRT works in 1T mode, start BRT
                MOV     TMOD,#00H           ;Timer 1 works in mode 0 (16-bit reload)
                MOV     TH1,#HIGH UARTBAUD  ;set reload value
                MOV     TL1,#LOW UARTBAUD
                SETB    TR1                 ;start timer 1
                SETB    ES                  ; Enable UART interrupt
                SETB    EA                  ; Enable CPU interrupt

MAIN:
                INC     P0
                SJMP    MAIN

                END
```

User code can be written in C or assembly language, but one thing to note about assembly code: the instruction at the reset entry address of 0000H must be a long jump statement (similar to LJMP START). In the user code, the serial port needs to be set up, and when the download conditions are met, the PC value is jumped to ISPPROGRAM (that is, the absolute address of FA00H) to achieve code update. For assembly code, we can use the "LJMP 0FA00H" instruction to call, as shown below



```
18          CLR     TI                          ;清除标志
19  CHECKRI:
20          JNB     RI,UARTISR_EXIT             ;检测接收中断
21          CLR     RI                          ;清除标志
22          MOV     A,SBUF
23          CJNE    A,#7FH,ISNOT7F
24          INC     CNT7F
25          MOV     A,CNT7F
26          CJNE    A,#16,UARTISR_EXIT
27          LJMP    ISPPROGRAM                  ;调用下载模块(****重要语句****)
28  ISNOT7F:
29          MOV     CNT7F,#0
30  UARTISR_EXIT:
31          POP     PSW
32          POP     ACC
33          RETI
34
35  START:
```

In the C code, you must define a function pointer variable, and assign this variable to 0xFA00, and then call, as shown below:

```
#include "reg51.h"

#define FOSC            11059200L                   //系统时钟频率
#define BAUD            (65536 - FOSC/4/115200)     //定义串口波特率
#define ISPPROGRAM      0xfa00                      //ISP下载程序入口地址

sfr AUXR    =   0x8e;                               //波特率发生器控制寄存器
sfr P1M0    =   0x92;
sfr P1M1    =   0x91;

void (*IspProgram)() = ISPPROGRAM;          //定义指针函数
char cnt7f;                                 //Isp_Check内部使用的变量

void uart() interrupt 4                     //串口中断服务程序
{
    if (TI) TI = 0;                         //发送完成中断
    if (RI)                                 //接收完成中断
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();               //调用下载模块(****重要语句****)
            }
        }
        else
        {
            cnt7f = 0;
        }
        RI = 0;                         //清接收完成标志
    }
```

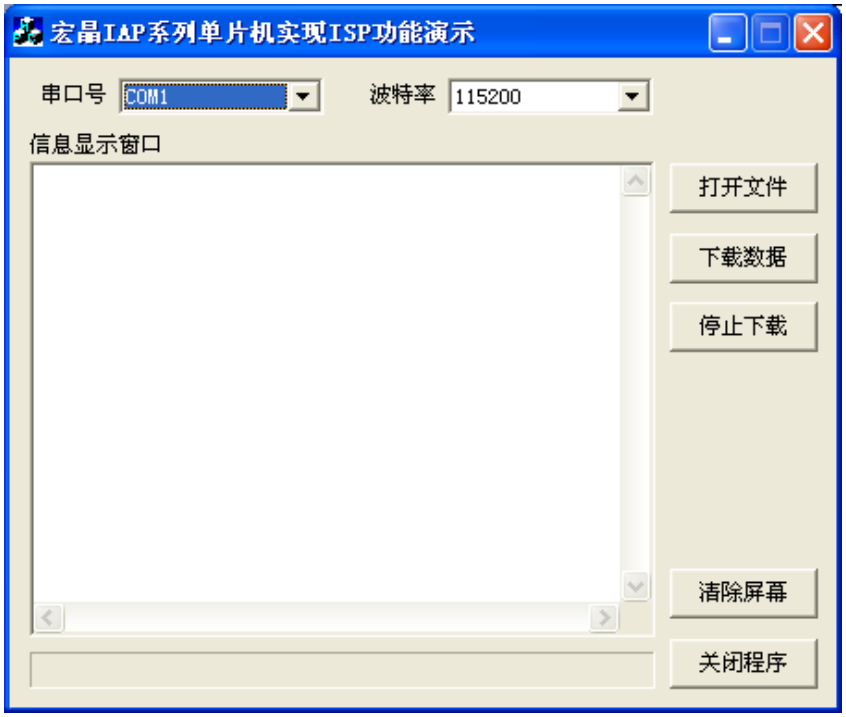
Step 4, the host computer application program description

The program of the upper computer is a dialog box project based on MFC. The access to the serial port is to directly call the API function of Windows without using the serial port control, which saves the registration of the control and many problems of system version incompatibility. The interface is relatively simple, but it provides a framework for the realization of this function. Other functions and requirements can be added in the past.

The core module of the host computer program is a friend function "UINT Download(LPVOID pParam);" based on CISPDlg,This function is responsible for communicating with the lower computer and sending various communication commands to complete the update of the user program. Users can add commands according to their different needs.

Step 5: How to use the host computer application

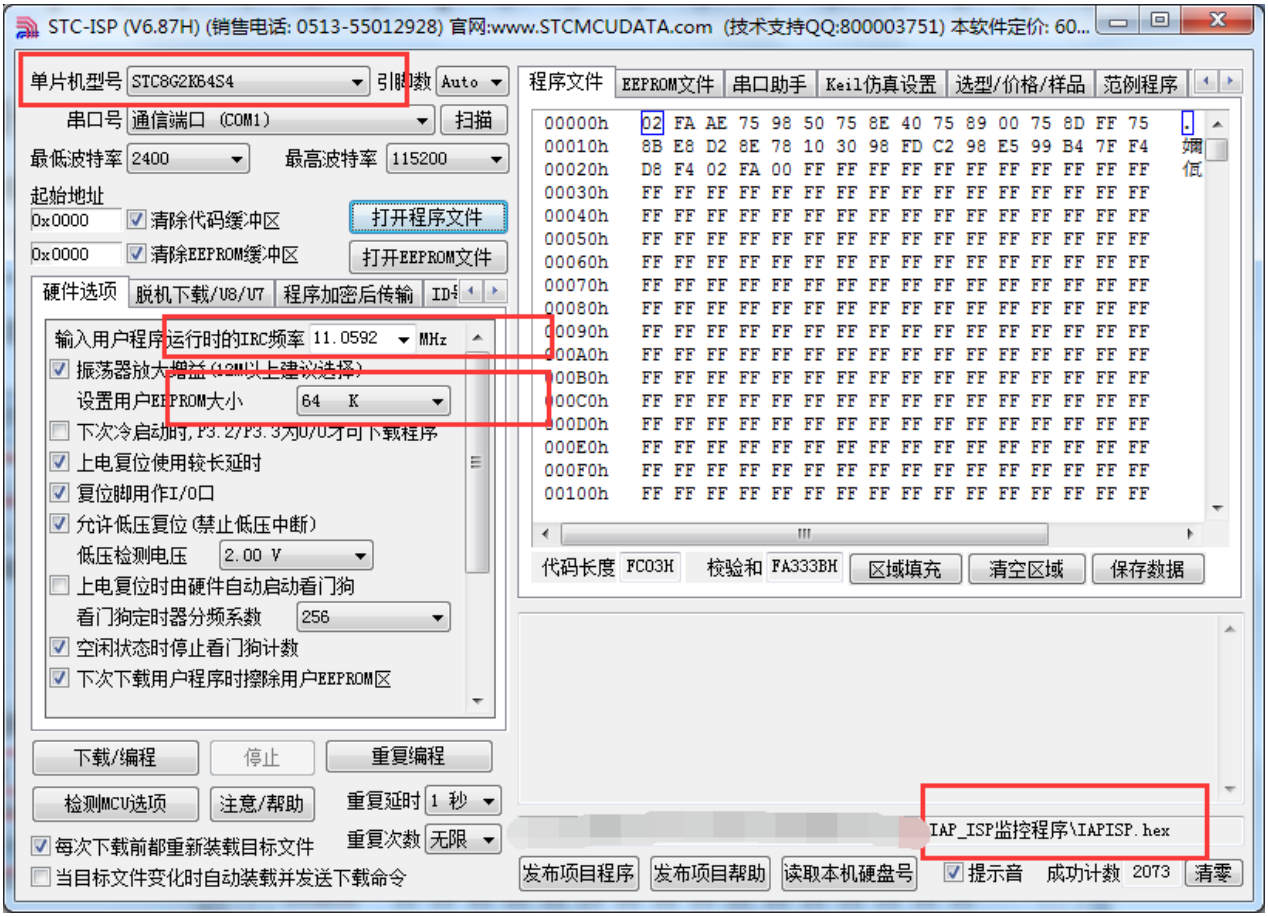
1. Open the host computer interface, as shown below



2. Select the serial port number, set the same serial port baud rate as the lower computer
3. Open the source data file to be downloaded, either in Bin or Intel hex format
4. Click the "download data" button to start downloading data

The sixth step, how to use the firmware of the lower computer

The target file of the lower computer has two "IAPISP.hex" and "AP.hex". For a new single-chip computer, for the first time, you must use the ISP download tool of Acer Technology to write "IAPISP.hex" into the chip. As shown below. No need to write after updating "IAPISP.hex" is the file. The "AP.hex" in the attachment is just a template for the user program. When the download conditions are met, the user only needs to jump the PC value to the address of FA00H to update the code.


STC-ISP (V6.87H) (销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (技术支持QQ:800003751) 本软件定价: 60...
单片机型号 STC8G2K64S4
引脚数 Auto
串口号 通信端口 (COM1)
扫描
最低波特率 2400
最高波特率 115200
起始地址
0x0000
清除代码缓冲区
打开程序文件
0x0000
清除EEPROM缓冲区
打开EEPROM文件
硬件选项
脱机下载/U8/U7
程序加密后传输
输入用户程序运行时的IRC频率 11.0592 MHz
振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 64 K
下次冷启动时, P3.2/P3.3为0/0才可下载程序
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 2.00 V
上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
空闲状态时停止看门狗计数
下次下载用户程序时擦除用户EEPROM区
下载/编程
停止
重复编程
检测MCU选项
注意/帮助
重复延时 1 秒
每次下载前都重新装载目标文件
重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件
EEPROM文件
串口助手
Keil仿真设置
选型/价格/样品
范例程序
00000h 02 FA AE 75 98 50 75 8E 40 75 89 00 75 8D FF 75
00010h 8B E8 D2 8E 78 10 30 98 FD C2 98 E5 99 B4 7F F4
00020h D8 F4 02 FA 00 FF FF FF FF FF FF FF FF FF FF FF
00030h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00040h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00050h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00060h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00070h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00080h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00090h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000A0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000B0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000C0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000D0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000E0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000F0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00100h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
代码长度 FC03H
校验和 FA333BH
区域填充
清空区域
保存数据
IAP_ISP监控程序\IAPISP.hex
发布项目程序
发布项目帮助
读取本机硬盘号
提示音
成功计数 2073
清零

Appendix L The method of resetting the user program to the system area for ISP download (without power off)

When the project is in the development stage, it is necessary to repeatedly download the user code to the target chip for code verification, and the STC microcontroller

For normal ISP downloads, the target chip needs to be re-powered, which will make the project development phase more cumbersome. For this reason, STC MCU has added a special function register IAP_CONTR. When the user writes 0x60 to this register, the software can be reset to the system area, and then ISP download can be performed without power failure.

But how do users judge whether ISP download is in progress? When to write 0x60 to register IAP_CONTR to trigger a soft reset? Regarding these two issues, four methods of judgment are introduced below:

Use P3.0 port to detect serial port start signal

The serial port ISP of the STC microcontroller uses P3.0 and P3.1. When the ISP download software starts to download, it will send a handshake command to the P3.0 port of the microcontroller. If the user's P3.0 and P3.1 are only used for ISP download, you can use the P3.0 port to detect the start signal of the serial port to judge the ISP download.

C Language Code

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     IAP_CONTR   =   0xc7;
sfr     P3M0        =   0xb2;
sfr     P3M1        =   0xb1;
sbit    P30         =   P3^0;

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
    P30 = 1;

    while (1)
    {
        if (!P30) IAP_CONTR = 0x60;          // The low level of P3.0 is the serial port start signal
                                             // Software reset to system area

        ...                                  //User codes
    }
}
```

Use the falling edge interrupt of P3.0/INT4 to detect the serial port start signal

Method B is similar to method A, except that method A uses query mode, and method B uses interrupt mode. Because the P3.0 port of the STC single-chip computer is the interrupt port of INT4.

C Language Code

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         IAP_CONTR   =   0xc7;
sfr         INTCLKO     =   0x8f;
sfr         P3M0        =   0xb2;
sfr         P3M1        =   0xb1;

void Int4Isr() interrupt 16                     //INT4 interrupt service routine
{
    IAP_CONTR = 0x60;                           // UART start signal triggers INT4 interrupt
                                                // Software reset to system area
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    INTCLKO |= 0x40;                            // Enable INT4 interrupt
    EA = 1;

    while (1)
    {
        ...                                     //User codes
    }
}
```

Use P3.0/RxD to receive and detect the 7F sent by the ISP download software

Method A and Method B are very simple, but easy to be interfered. If there is any interference signal on P3.0 port, it will trigger the software Reset, so method C is to verify the serial port data.

When STC's ISP download software performs ISP download, it will first use the lowest baud rate (usually 2400) + even parity 9+1 stop bit to continuously send the handshake command 7F, so the user can set the serial port to 9 in the program Bit data bit + 2400 baud rate, and then continue to detect 7F. For example, if 8 7Fs are continuously detected, it means that ISP download is required, and then a software reset is triggered.

C Language Code

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define     FOSC            11059200UL
#define     BR2400          (65536 - FOSC / 4 / 2400)

sfr         IAP_CONTR   =   0xc7;
sfr         AUXR        =   0x8e;
sfr         P3M0        =   0xb2;
```

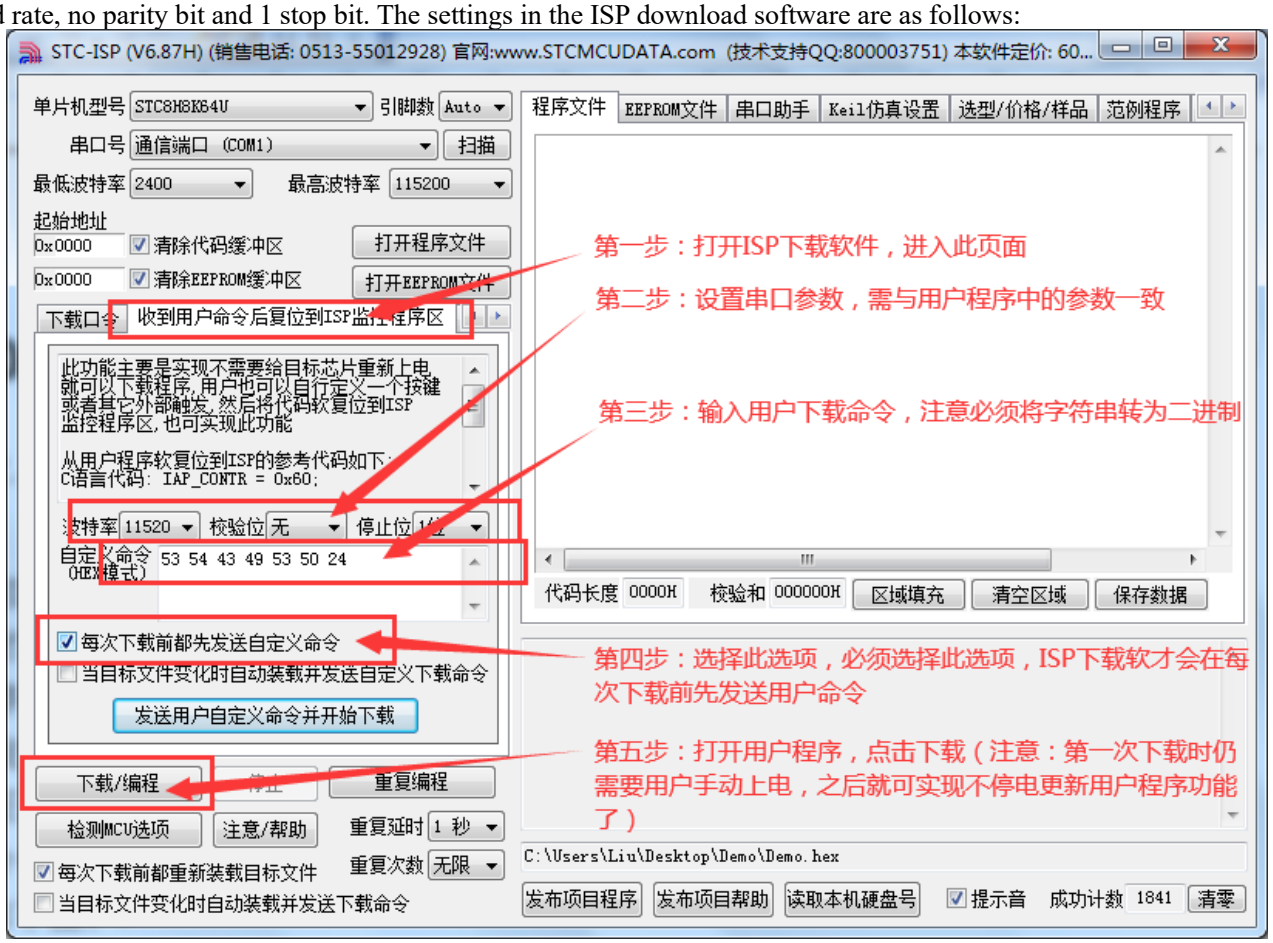
```
sfr         P3M1            =     0xb1;

char cnt7f;

void UartIsr() interrupt 4                          // UART Interrupt Service Routine
{
    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        if ((SBUF == 0x7f) && (RB8 == 1))           // Handshake command 7F sent by ISP download software
                                                    //The even parity bit of 7F is 1
        {
            if (++cnt7f == 8)                       // When 8 consecutive 7Fs are detected
                IAP_CONTR = 0x60;                   // reset to system area
        }
        else
        {
            cnt7f = 0;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0xd0;                                    // Set the UART to 9 data bits
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                              // Set the UART baud rate to 2400
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

    cnt7f = 0;

    while (1)
    {
        ...                                         //User codes
    }
}
```

Use P3.0/RxD to receive and detect user download commands sent by ISP download software

If the user code needs to use the serial port for communication, the above 3 methods may not be applicable. At this time, you can use the interface provided by the STC ISP download software to customize a set of dedicated user download commands (you can specify the baud rate, Check bit and stop bit). If this function is enabled, the ISP download software will use the user-specified baud rate, check bit and stop bit to send the user download command before ISP download, and then send the handshake command. The user only needs to monitor the serial port command sequence in his own code. When the correct user download command is detected, the software is reset to the system

area to realize the ISP function without power failure.

The following assumes that the user download command is the string "STCISP$", the serial port is set to 115200 baud rate, no parity bit and 1 stop bit. The settings in the ISP download software are as follows:



User sample code is as follows:

C Language Code

```
// Operating frequency for test is 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define    FOSC         11059200UL
#define    BR115200     (65536 - FOSC / 4 / 115200)

sfr        IAP_CONTR    =    0xc7;
sfr        AUXR         =    0x8e;
sfr        P3M0         =    0xb2;
sfr        P3M1         =    0xb1;

char stage;

void UartIsr() interrupt 4                          // UART Interrupt Service Routine
{
    char dat;

    if (TI)
    {
        TI = 0;
```

```
    }

    if (RI)
    {
        RI = 0;

        dat = SBUF;
        switch (stage)
        {
        case 0:
        default:
L_Check1st:
            if (dat == 'S') stage = 1;
            else stage = 0;
            break;
        case 1:
            if (dat == 'T') stage = 2;
            else goto L_Check1st;
            break;
        case 2:
            if (dat == 'C') stage = 3;
            else goto L_Check1st;
            break;
        case 3:
            if (dat == 'I') stage = 4;
            else goto L_Check1st;
            break;
        case 4:
            if (dat == 'S') stage = 5;
            else goto L_Check1st;
            break;
        case 5:
            if (dat == 'P') stage = 6;
            else goto L_Check1st;
            break;
        case 6:
            if (dat == '$')                         // When the correct user download command is detected
                IAP_CONTR = 0x60;                   // reset to system area
            else goto L_Check1st;
        break;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0x50;                                    // Set the UART to 8 data bits
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                              // Set the UART baud rate to 115200
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;
```

```
    stage = 0;

    while (1)
    {
        ...                                         // User codes
    }
}
```

Appendix M Example Routine of ISP download for STC8A8K64D4 series MCUs using third-party MCU

C language code

```
/*Note: When using this code to download the STC8A8K64D4 series of microcontrollers, you must execute the Download code before powering on the target chip, otherwise the target chip will not download correctly.*/

#include "reg51.h"

typedef    bit              BOOL;
typedef    unsigned char    BYTE;
typedef    unsigned short   WORD;

//Macro and constant definition
#define    FALSE            0
#define    TRUE             1
#define    LOBYTE(w)        ((BYTE)(WORD)(w))
#define    HIBYTE(w)        ((BYTE)((WORD)(w) >> 8))

#define    MINBAUD          2400L
#define    MAXBAUD          115200L

#define    FOSC             11059200L          //Main chip working frequency
#define    BR(n)            (65536 - FOSC/4/(n))   // Calculation formula of serial port baud rate of main chip
#define    T1MS             (65536 - FOSC/1000)    // 1ms timing initial value of main chip

#define    FUSER            24000000L          //STC8 Series target chip operating frequency
#define    RL(n)            (65536 - FUSER/4/(n))  //STC8 Serial target chip baud rate calculation formula

sfr        AUXR = 0x8e;
sfr        P3M1 = 0xB1;
sfr        P3M0 = 0xB2;

// Variable definitions
BOOL f1ms;                    //1ms flag
BOOL UartBusy;                // Serial transmit busy flag
BOOL UartReceived;            // Serial data receiving completion flag
BYTE UartRecvStep;            // Serial data receiving control
BYTE TimeOut;                 // Serial communication timeout counter
BYTE xdata TxBuffer[256];     // Serial data transmission buffer
BYTE xdata RxBuffer[256];     // Serial data receiving buffer
char code DEMO[256];          // Demo code data

// Functions declarations
```

```
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
BOOL Download(BYTE *pdat, long size);

// Main function entry
void main(void)
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    Initial();
    if (Download(DEMO, 256))
    {
        // download successfully
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
    }
    else
    {
        // download failed
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
    }

    while (1);
}

//1ms Timer interrupt service routine
void tm0(void) interrupt 1
{
    static BYTE Counter100;

    f1ms = TRUE;
```

```
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}

// Serial port interrupt service routine
void uart(void) interrupt 4
{
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;
        switch (UartRecvStep)
        {
        case 1:
            if (dat != 0xb9) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 2:
            if (dat != 0x68) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 3:
            if (dat != 0x00) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 4:
            RecvSum = 0x68 + dat;
            RecvCount = dat - 6;
            RecvIndex = 0;
            UartRecvStep++;
            break;
        case 5:
            RecvSum += dat;
            RxBuffer[RecvIndex++] = dat;
            if (RecvIndex == RecvCount)     UartRecvStep++;
            break;
        case 6:
            if (dat != HIBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 7:
            if (dat != LOBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
```

```
            case 8:
                if (dat != 0x16) goto L_CheckFirst;
                UartReceived = TRUE;
                UartRecvStep++;
                break;
L_CheckFirst:
            case 0:
            default:
                CommInit();
                UartRecvStep = (dat == 0x46       ? 1 : 0);
                break;
            }
        }
}

// system initialization
void Initial(void)
{
        UartBusy = FALSE;

        SCON = 0xd0;                            // Serial data format must be 8-bit data + 1-bit even check
        AUXR = 0xc0;
        TMOD = 0x00;
        TH0 = HIBYTE(T1MS);
        TL0 = LOBYTE(T1MS);
        TR0 = 1;
        TH1 = HIBYTE(BR(MINBAUD));
        TL1 = LOBYTE(BR(MINBAUD));
        TR1 = 1;
        ET0 = 1;
        ES = 1;
        EA = 1;
}

//Xms Delay program
void DelayXms(WORD x)
{
        do
        {
            f1ms = FALSE;
            while (!f1ms);
        } while (x--);
}

// Serial data sending program
BYTE UartSend(BYTE dat)
{
        while (UartBusy);

        UartBusy = TRUE;
        ACC = dat;
        TB8 = P;
        SBUF = ACC;

        return dat;
}

// Serial communication initialization
```

```
void CommInit(void)
{
	UartRecvStep = 0;
	TimeOut = 20;
	UartReceived = FALSE;
}

// Send serial communication packets
void CommSend(BYTE size)
{
	WORD sum;
	BYTE i;

	UartSend(0x46);
	UartSend(0xb9);
	UartSend(0x6a);
	UartSend(0x00);
	sum = size + 6 + 0x6a;
	UartSend(size + 6);
	for (i=0; i<size; i++)
	{
		sum  += UartSend(TxBuffer[i]);
	}
	UartSend(HIBYTE(sum));
	UartSend(LOBYTE(sum));
	UartSend(0x16);
	while (UartBusy);

	CommInit();
}

//对 STC15H Series of chips for ISP download
BOOL Download(BYTE *pdat, long size)
{
	BYTE arg;
	BYTE offset;
	BYTE cnt;
	WORD addr;

	// Shake hands
	CommInit();
	while (1)
	{
		if (UartRecvStep == 0)
		{
			UartSend(0x7f);
			DelayXms(10);
		}
		if (UartReceived)
		{
			arg = RxBuffer[4];
			if (RxBuffer[0] == 0x50) break;
			return FALSE;
		}
	}

	// Set parameters (set the parameters such as the highest baud rate used by the slave chip and erase wait time)
	TxBuffer[0] = 0x01;
```

```
    TxBuffer[1] = arg;
    TxBuffer[2] = 0x40;
    TxBuffer[3] = HIBYTE(RL(MAXBAUD));
    TxBuffer[4] = LOBYTE(RL(MAXBAUD));
    TxBuffer[5] = 0x00;
    TxBuffer[6] = 0x00;
    TxBuffer[7] = 0x97;
    CommSend(8);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x01) break;
            return FALSE;
        }
    }

    //prepare
    TH1 = HIBYTE(BR(MAXBAUD));
    TL1 = LOBYTE(BR(MAXBAUD));
    DelayXms(10);
    TxBuffer[0] = 0x05;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    CommSend(5);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x05) break;
            return FALSE;
        }
    }

    // Erase
    DelayXms(10);
    TxBuffer[0] = 0x03;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    CommSend(5);
    TimeOut = 100;
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x03) break;
            return FALSE;
        }
    }

    // Write user code
```

```
    DelayXms(10);
    addr = 0;
    TxBuffer[0] = 0x22;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    offset = 5;
    while (addr < size)
    {
        TxBuffer[1] = HIBYTE(addr);
        TxBuffer[2] = LOBYTE(addr);
        cnt    = 0;
        while (addr < size)
        {
            TxBuffer[cnt+offset] = pdat[addr];
            addr++;
            cnt++;
            if (cnt >= 128) break;
        }
        CommSend(cnt + offset);
        while (1)
        {
            if (TimeOut == 0) return FALSE;
            if (UartReceived)
            {
                if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
                return FALSE;
            }
        }
        TxBuffer[0] = 0x02;
    }

    //// Write hardware options
    //// If you do not need to modify the hardware options, this step can be skipped directly.At this time, all the hardware options
    //// remain unchanged, the frequency of the MCU is the last adjusted frequency
    //// If you write the hardware option, the MCU's internal IRC frequency will be fixed to 24MHz,
   //// and other options will be restored to the factory settings.
    ////Suggestion: Set the hardware options of the slave chip when you use STC-ISP download software the first time.

    //// Do not write hardware options when downloading programs from the master chip to the slave chip.
    //DelayXms(10);
    //for (cnt=0; cnt<128; cnt++)
    //{
    //      TxBuffer[cnt] = 0xff;
    //}
    //TxBuffer[0] = 0x04;
    //TxBuffer[1] = 0x00;
    //TxBuffer[2] = 0x00;
    //TxBuffer[3] = 0x5a;
    //TxBuffer[4] = 0xa5;
    //TxBuffer[33] = arg;
    //TxBuffer[34] = 0x00;
    //TxBuffer[35] = 0x01;
    //TxBuffer[41] = 0xbf;
    //TxBuffer[42] = 0xbd;                  //P5.4 is I/O port
    ////TxBuffer[42] = 0xad;                //P5.4 is reset pin
    //TxBuffer[43] = 0xf7;
    //TxBuffer[44] = 0xff;
    //CommSend(45);
```

```
    //while (1)
    //{
    //    if (TimeOut == 0) return FALSE;
    //    if (UartReceived)
    //    {
    //        if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
    //        return FALSE;
    //    }
    //}

    // Download completed
    return TRUE;
}

char code DEMO[256] =
{
    0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
    0xD9,0xFC,0x22,
};
```

Note: If user needs to set different working frequencies, please refer to the example codes in chapters 7.3.7 and 7.3.8.

Appendix N Use a third-party application program to call the STC release project program to download the ISP of the MCU

The release project program generated by STC's ISP download software is an executable EXE format file. The user can directly double-click the released project program to run it for ISP download, or call the release project program in a third-party application for ISP download. Two methods of calling are introduced below.

Simple call

In the third-party application, it is only a simple process of creating and publishing the project program. All other download operations are carried out in the publishing project program. The third-party application only needs to wait for the completion of the publishing project program and clean the scene.

VC code

```
BOOL IspProcess()
{
	// define related variables
	STARTUPINFO si;
	PROCESS_INFORMATION pi;
	CString path;

	// Full path of publishing project program
	path = _T("D:\\Work\\Upgrade.exe");

	// variable initialization
	memset(&si, 0, sizeof(STARTUPINFO));
	memset(&pi, 0, sizeof(PROCESS_INFORMATION));

	// Set startup variables
	si.cb = sizeof(STARTUPINFO);
	GetStartupInfo(&si);
	si.wShowWindow = SW_SHOWNORMAL;
	si.dwFlags = STARTF_USESHOWWINDOW;

	// Create Publish Project Program Process
	if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
	{
		// Wait for the publish project program operation to complete
		// Since the main process will be blocked here, it is recommended to create a new working process and wait in the worker process
		WaitForSingleObject(pi.hProcess,INFINITE);
```

```
        // clean up
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);

        return TRUE;
    }
    else
    {
        AfxMessageBox(_T("创建进程失败!"));

        return FALSE;
    }
}
```

Advanced call

The process of creating and publishing project programs in third-party applications, including selecting serial ports and starting ISP in third-party applications

All ISP download operations such as programming, ISP programming with vibration stopped, and closing the release project program, do not require interface interaction in the release project program.

VC 代码

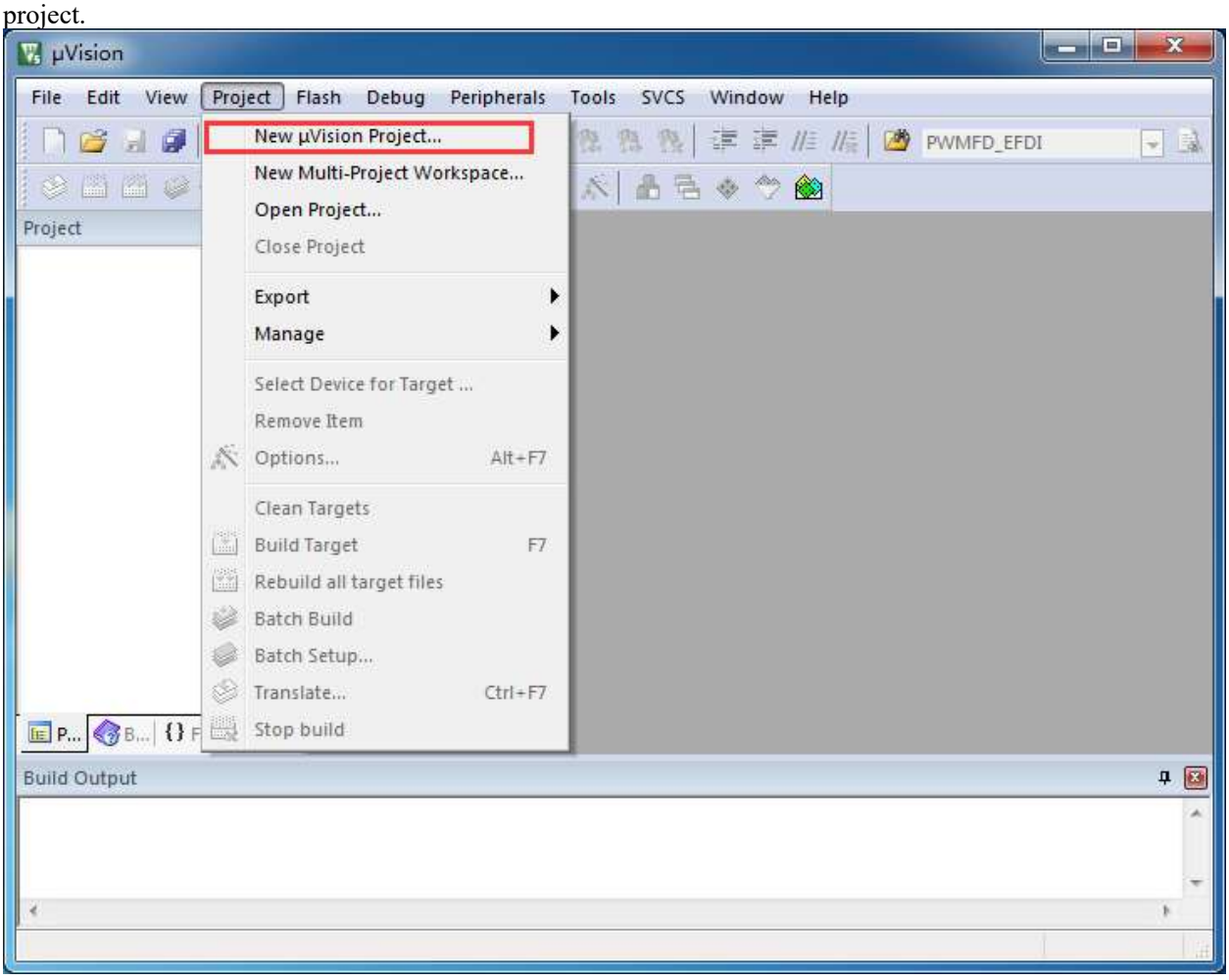
```
// A data structure that defines the parameters of the callback function
struct CALLBACK_PARAM
{
    DWORD dwProcessId;                  // main process id
    HWND hMainWnd;                      // main window handle
};

// Callback function for enumerating windows to get the main window handle
BOOL CALLBACK EnumWindowCallBack(HWND hWnd, LPARAM lParam)
{
    CALLBACK_PARAM *pcp = (CALLBACK_PARAM *)lParam;
    DWORD id;

    GetWindowThreadProcessId(hWnd, &id);
    if ((pcp->dwProcessId == id) && (GetParent(hWnd) == NULL))
    {
        pcp->hMainWnd = hWnd;

        return FALSE;
    }

    return TRUE;
}

BOOL IspProcess()
{
    // define related variables
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CALLBACK_PARAM cp;
    CString path;

    // the IDs of some controls in the publishing project program
    const UINT ID_PROGRAM       = 1046;
    const UINT ID_STOP          = 1044;
    const UINT ID_COMPORT       = 1009;
```

```
const UINT ID_PROGRESS       = 1044;

// Full path of publishing project program
path = _T("D:\\Work\\Upgrade.exe");

// variable initialization
memset(&si, 0, sizeof(STARTUPINFO));
memset(&pi, 0, sizeof(PROCESS_INFORMATION));
memset(&cp, 0, sizeof(CALLBACK_PARAM));

// Set startup variables
si.cb = sizeof(STARTUPINFO);
GetStartupInfo(&si);
si.wShowWindow = SW_SHOWNORMAL;                    // If it is set to SW_HIDE here, the operation interface for publishing the project program will not be displayed, and all ISP operations can be performed in the background

si.dwFlags = STARTF_USESHOWWINDOW;

// Create a process for publishing a project program
if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
{
    // Wait for the release project program process initialization to complete
    WaitForInputIdle(pi.hProcess, 5000);

    // Get the main window handle of the publishing project program
    cp.dwProcessId = pi.dwProcessId;
    cp.hMainWnd = NULL;
    EnumWindows(EnumWindowCallBack, (LPARAM)&cp);

    if (cp.hMainWnd != NULL)
    {
        HWND hProgram;
        HWND hStop;
        HWND hPort;

        // Get the handle of some controls in the main window of the publishing project program
        hProgram = ::GetDlgItem(cp.hMainWnd, ID_PROGRAM);
        hStop = ::GetDlgItem(cp.hMainWnd, ID_STOP);
        hPort = ::GetDlgItem(cp.hMainWnd, ID_COMPORT);

        // Set the serial port number in the release project program, the third parameter is 0:COM1, 1:COM2, 2:COM3, ...
        ::SendMessage(hPort, CB_SETCURSEL, 0, 0);

        // Trigger the programming button to start ISP programming
        ::SendMessage(hProgram, BM_CLICK, 0, 0);

        // wait for programming to complete,
        // Since the main process will be blocked here, it is recommended to create a new working process and wait in the worker process
        while (!::IsWindowEnabled(hProgram));

        // Close the release project program after programming is complete
        ::SendMessage(cp.hMainWnd, WM_CLOSE, 0, 0);
    }

    // wait for the process to end
    WaitForSingleObject(pi.hProcess,INFINITE);
```

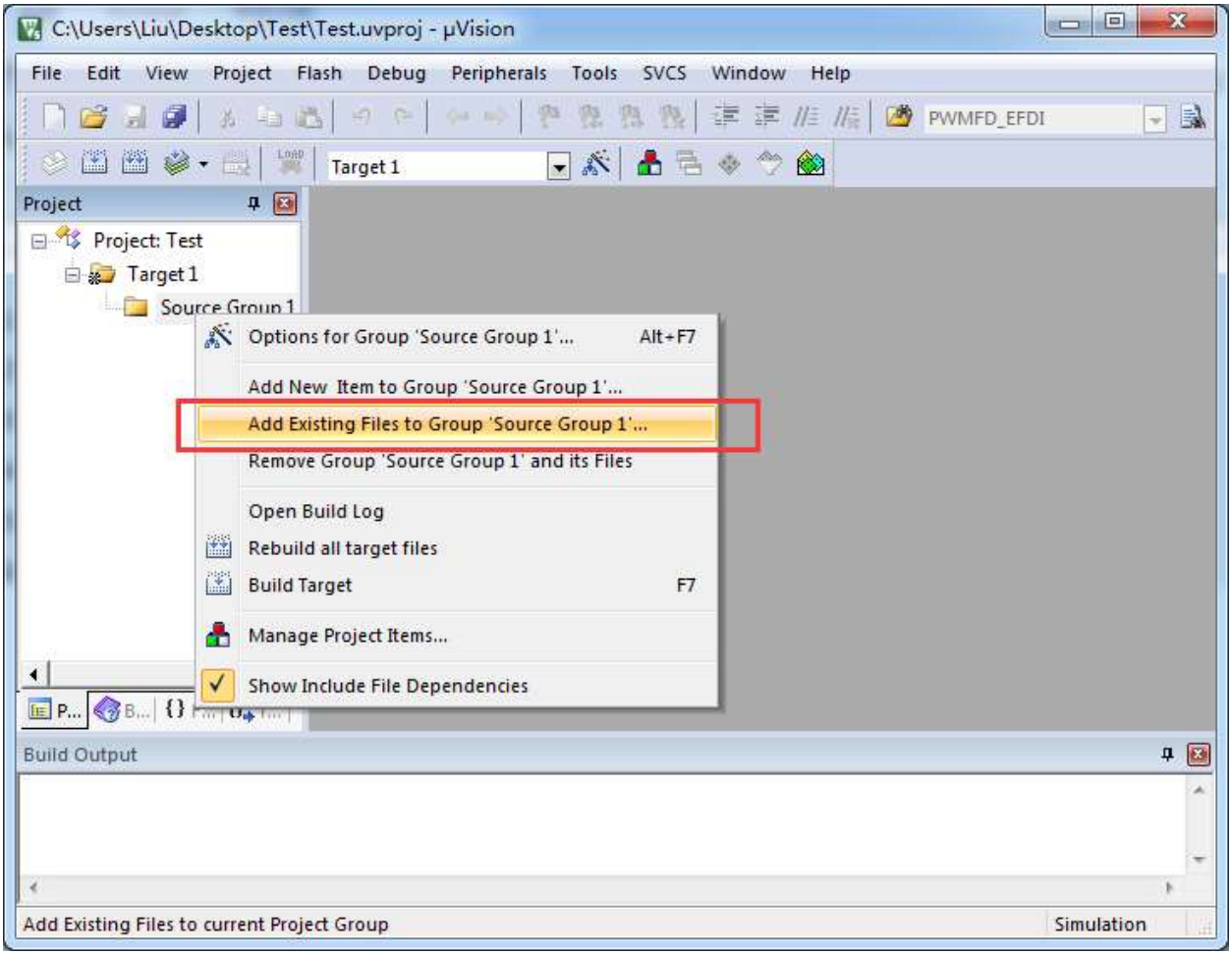
```
        // clean up
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);

        return TRUE;
    }
    else
    {
        AfxMessageBox(_T("创建进程失败!"));

        return FALSE;
    }
}
```

Appendix O Method for Creating Multi-file Projects in Keil

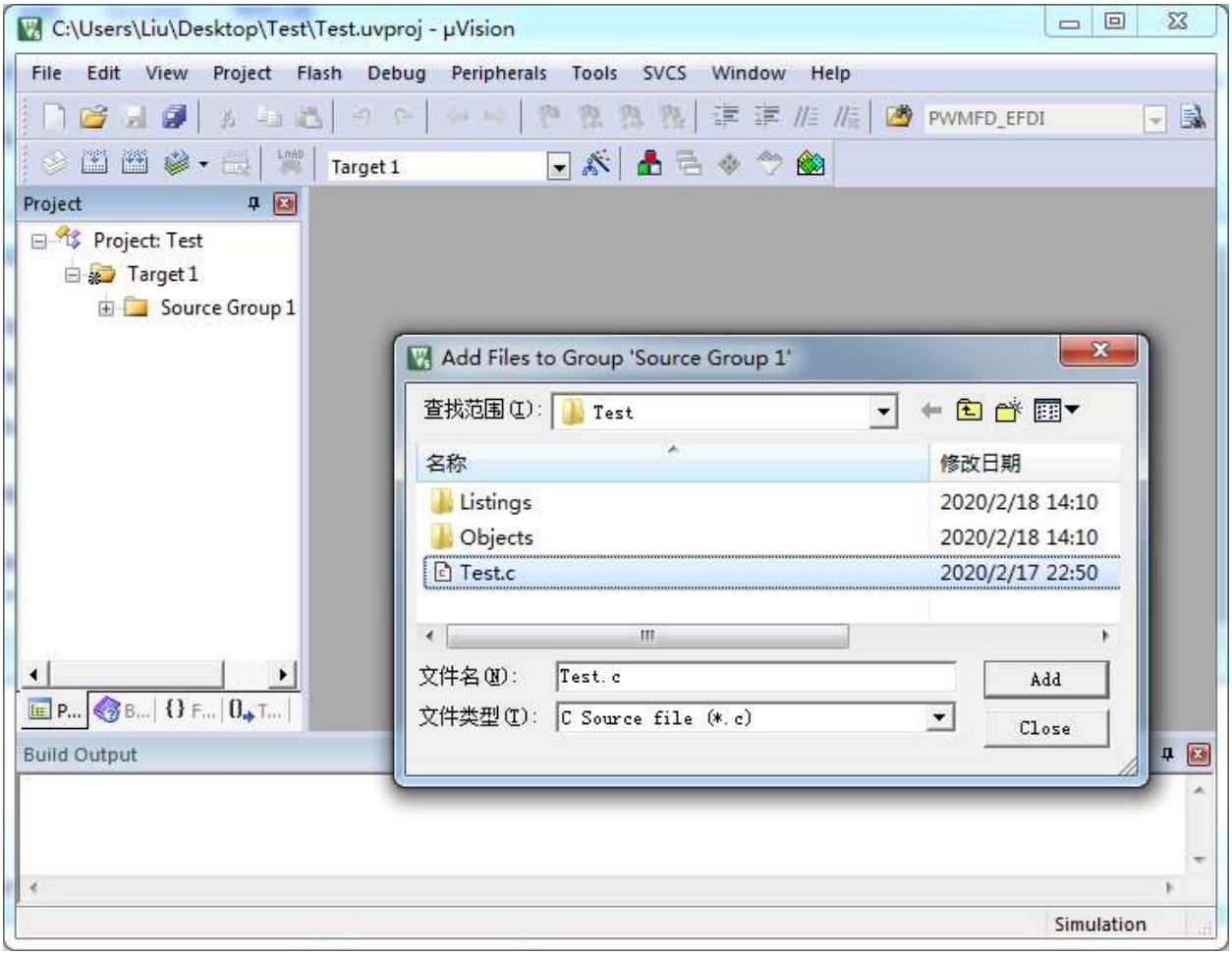
In Keil, relatively small projects generally have only one source file, but for some slightly more complex projects, multiple source files are often required. Here's how to set up a multi-file project:
1. Open Keil firstly and select "New uVision Project ..." from the "Project" menu to complete the creation of an empty project.



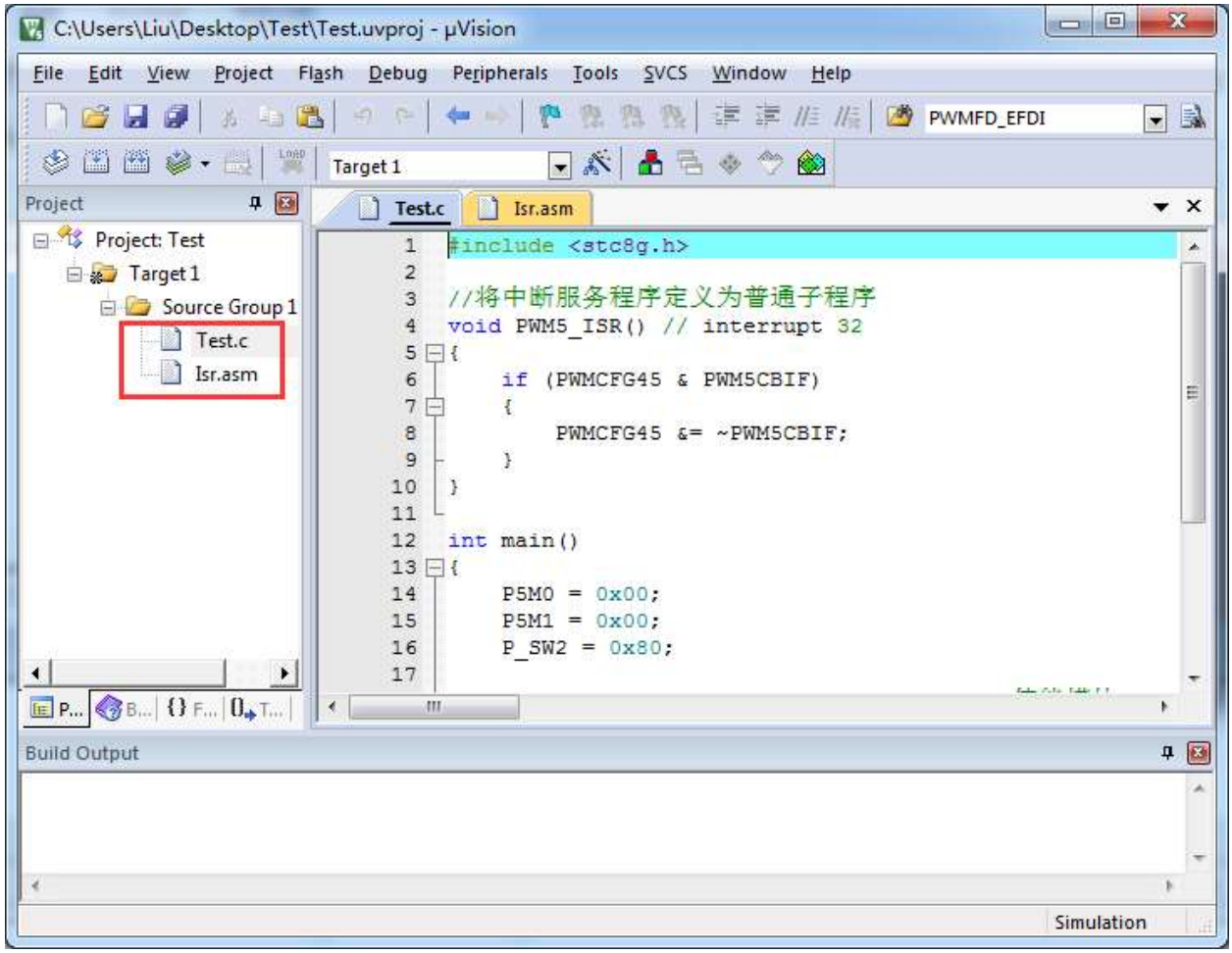
2. In the project tree of the empty project, right-click "Source Group 1" and select "Add Existing Files to Group" Source Group 1 "..." from the right-click menu.



3. In the file dialog that pops up, add the source file multiple times.

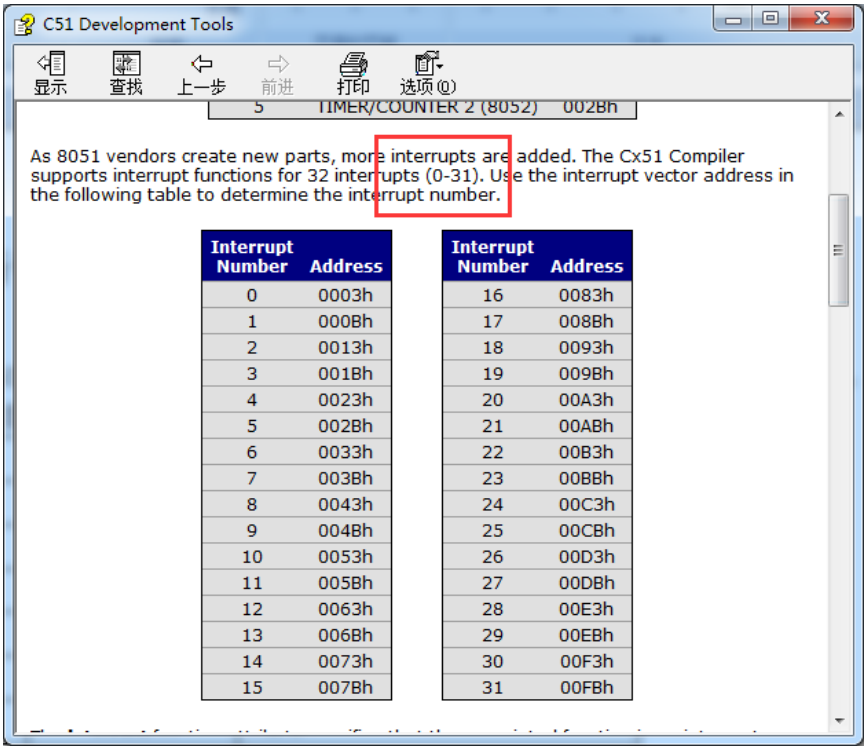


Complete the creation of the multi-file project as shown in the figure below.


C:\Users\Liu\Desktop\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
PWMFD_EFDI
Target 1
Project
Project: Test
Target 1
Source Group 1
Test.c
Isr.asm
Test.c
Isr.asm
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

Appendix P Handling of Compilation Error in Keil with Interrupt Numbers Greater Than 31

In Keil's C51 compilation environment, only 0 ~ 31 of the interrupt number are supported, that is, the interrupt vector must be less than 0100H.

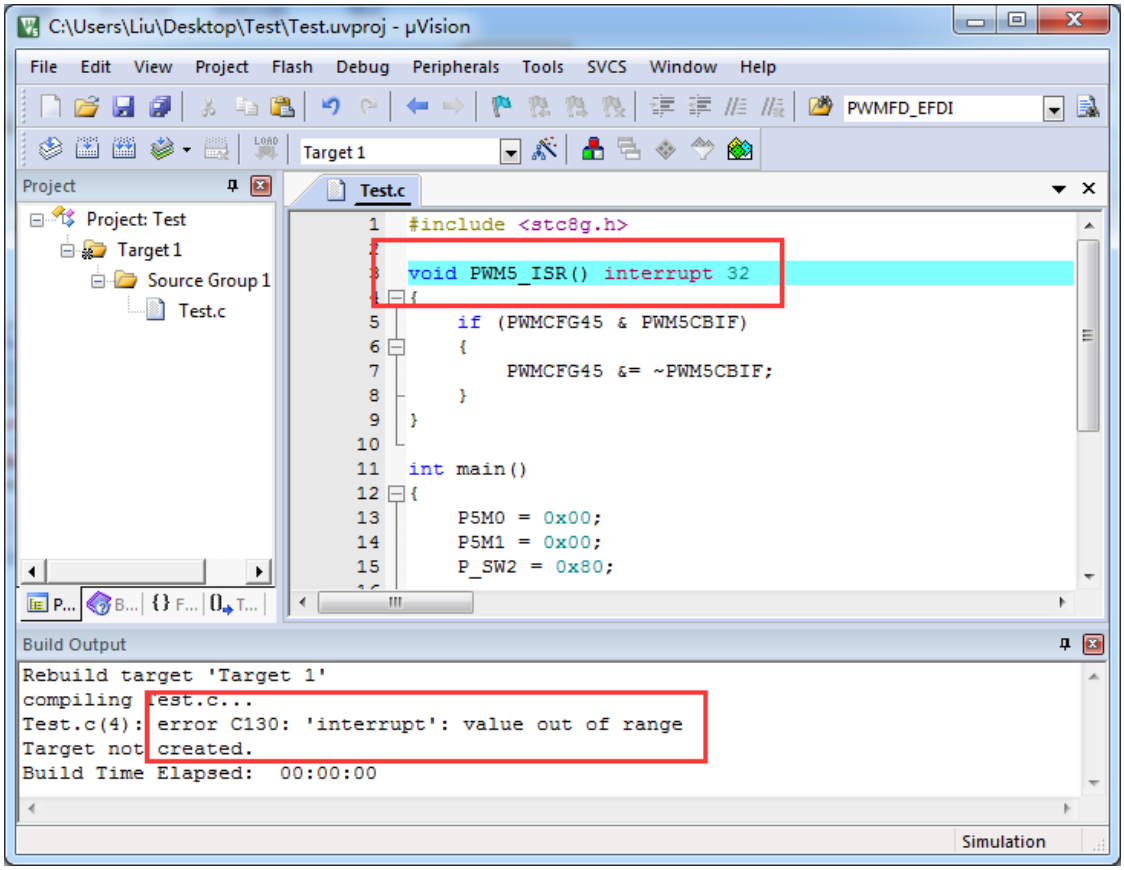


The following table is a list of interrupts for all current STC series:

Interrupt number	Interrupt vector	Interrupt type
0	0003 H	INT0
1	000B H	Timer 0
2	0013 H	INT1
3	001B H	Timer 1
4	0023 H	UART 1
5	002B H	ADC
6	0033 H	LVD
7	003B H	PCA
8	0043 H	UART 2
9	004B H	SPI
10	0053 H	INT2
11	005B H	INT3
12	0063 H	Timer 2

13	**006B H**	
14	0073 H	System internal interrupt
15	007B H	System internal interrupt
16	0083 H	INT4
17	008B H	UART 3
18	0093 H	UART 4
19	009B H	Timer 3
20	00A3 H	Timer 4
21	00AB H	Comparator
22	00B3 H	Waveform generator 0
23	00BB H	Waveform generator fault 0
24	00C3 H	I2C
25	00CB H	USB
26	00D3 H	PWMA
27	00DB H	PWMB
28	00E3 H	Waveform generator 1
29	00EB H	Waveform generator 2
30	00F3 H	Waveform generator 3
31	00FB H	Waveform generator 4
32	**0103 H**	**Waveform generator 5**
33	**010B H**	**Waveform generator fault 2**
34	**0113 H**	**Waveform generator fault 4**
35	**011B H**	**Touch Key**
36	**0123 H**	**RTC**
37	**012B H**	**P0 interrupt**
38	**0133 H**	**P1 interrupt**
39	**013B H**	**P2 interrupt**
40	**0143 H**	**P3 interrupt**
41	**014B H**	**P4 interrupt**
42	**0153 H**	**P5 interrupt**
43	**015B H**	**P6 interrupt**
44	**0163 H**	**P7 interrupt**
45	**016B H**	**P8 interrupt**
46	**0173 H**	**P9 interrupt**

It is not difficult to find that starting from the interrupt of the waveform generator 5, there will be errors when all subsequent interrupt service routines be compiled in keil, as shown in the following figure:

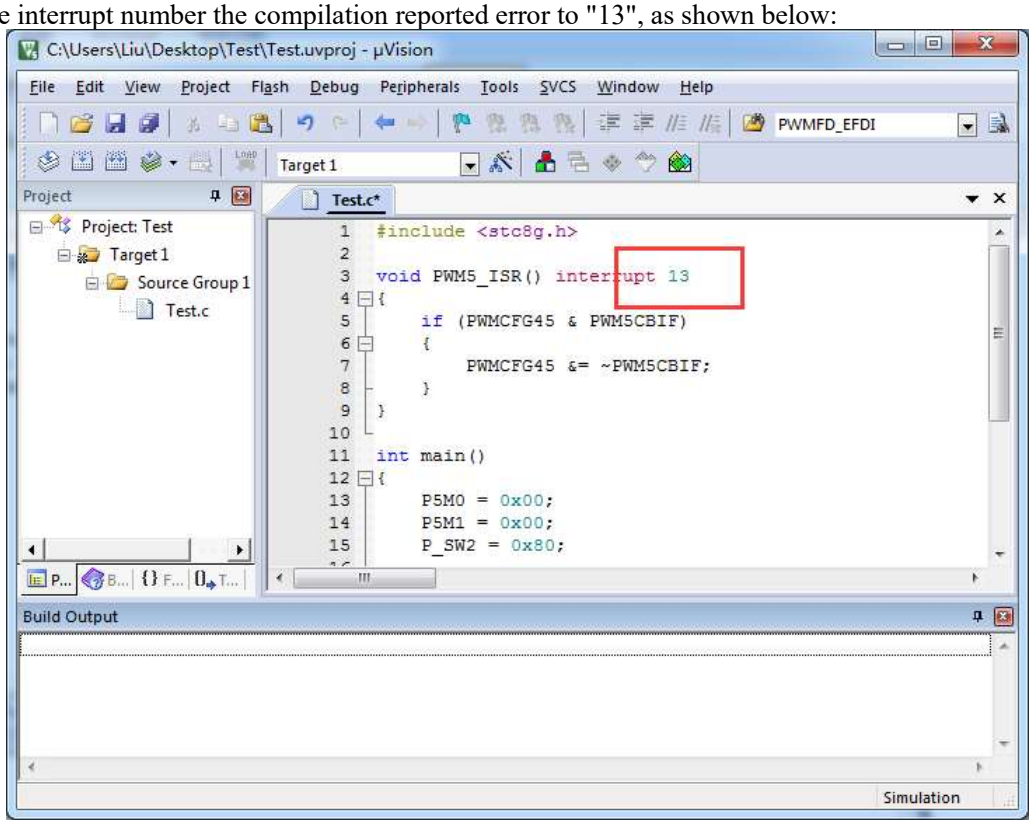


There are three ways to deal with this kind of error: (All of them need the help of assembly code, the first method is recommended)

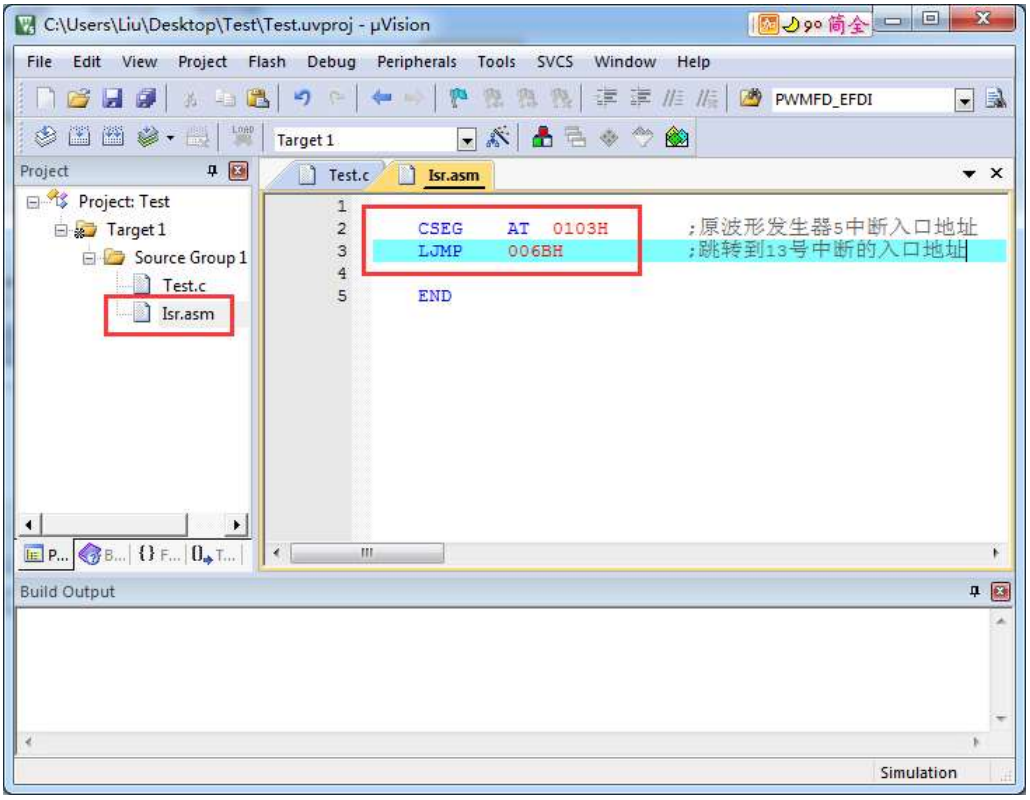
Method 1: Borrow Interrupt Vector 13

Among interrupts 0 ~ 31, the 13th is a reserved interrupt number, we can borrow this interrupt number.
The steps are as follows:

1. Change the interrupt number the compilation reported error to "13", as shown below:



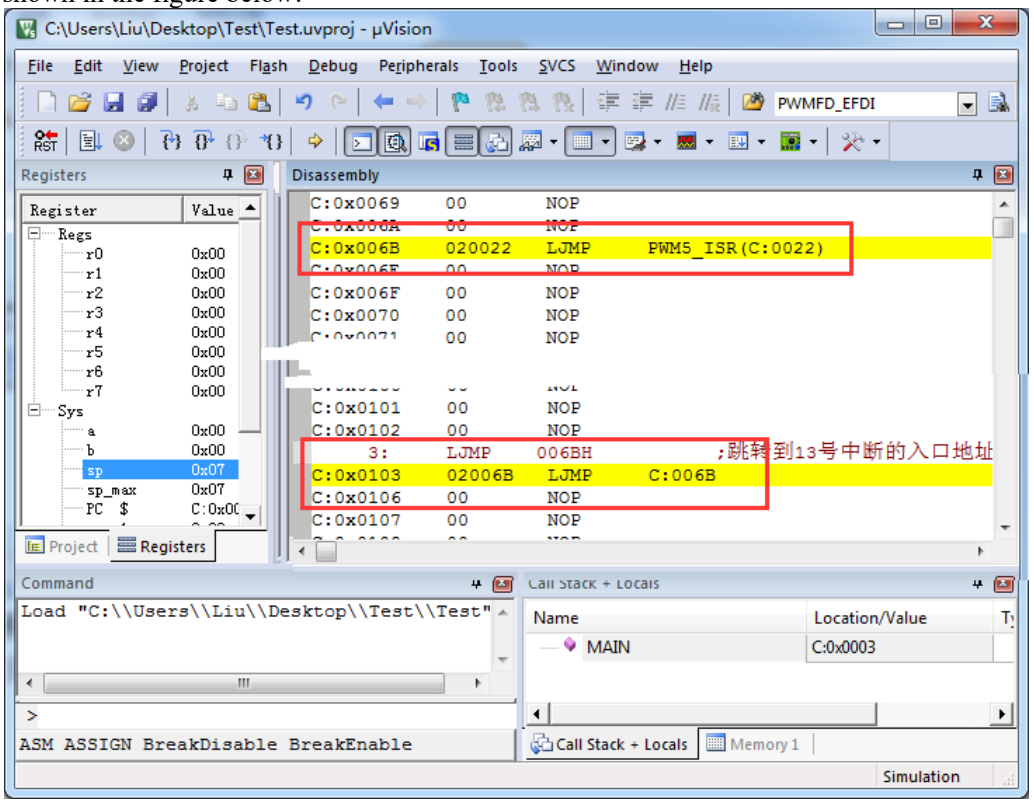
2. Create a new assembly language file, such as "isr.asm", add it to the project, and add "LJMP 006BH" at the address "0103H", as shown below:



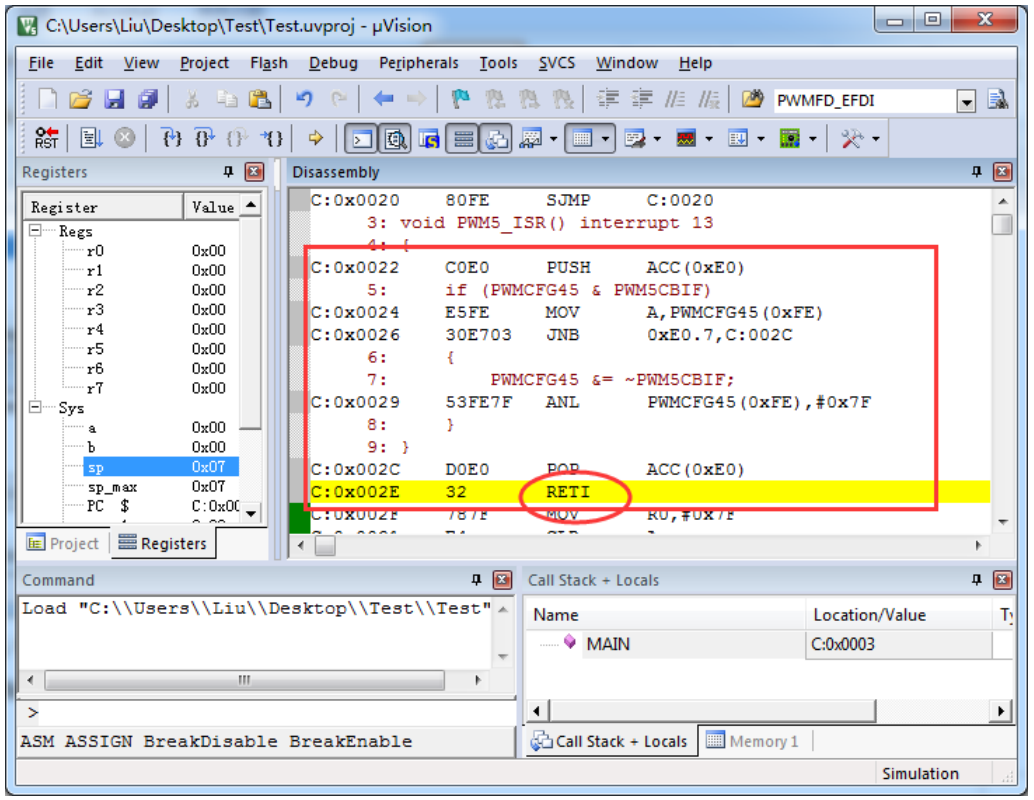
3. Compile successfully.

Now, after being compiled by Keil's C51 compiler, there is an "LJMP PWM5_ISR" at 006BH and an "LJMP 006BH"

at 0103H, as shown in the figure below:



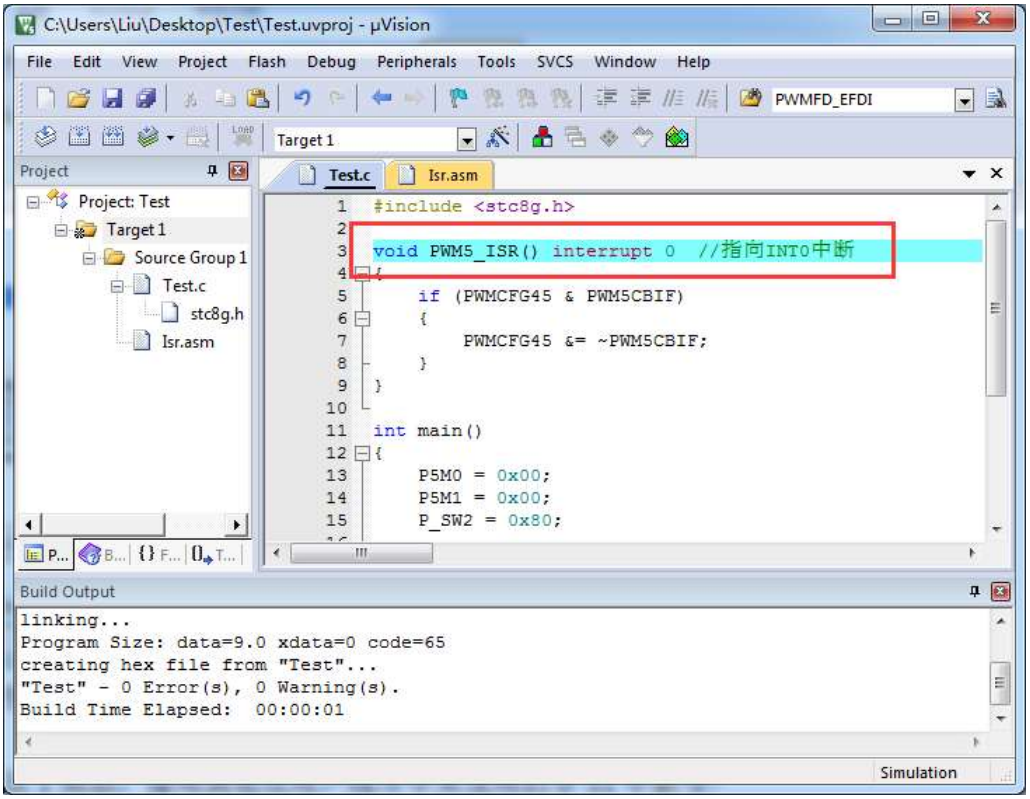
When the PWM5 interrupt occurs, the hardware will jump to the 0103H address automatically to execute "LJMP 006BH", and then execute "LJMP PWM5_ISR" at 006BH to jump to the real interrupt service routine, as shown in the figure below:

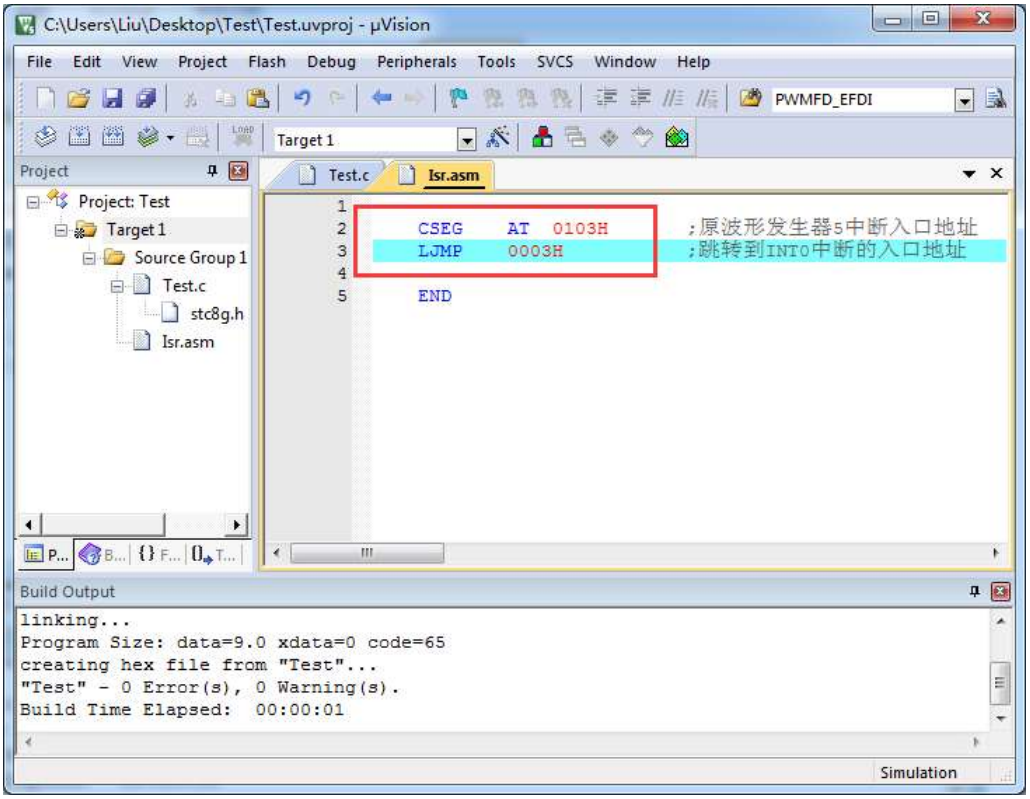


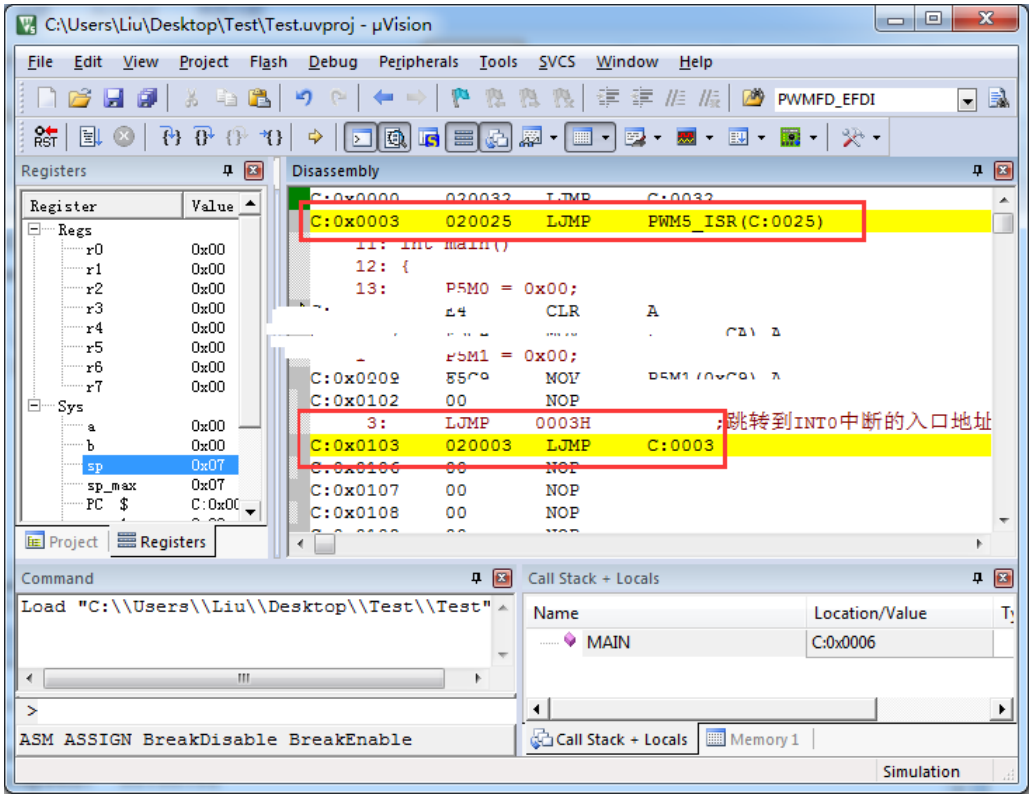
After the execution of the interrupt service routine is completed, it returns through the RETI instruction. The entire interrupt response process just executed an additional LJMP statement.

Method 2: Similar to method 1, borrow unused interrupt numbers from 0 to 31 in user program.

For example, in the user's code, if the INT0 interrupt is not used, the above code can be modified similarly to method 1:





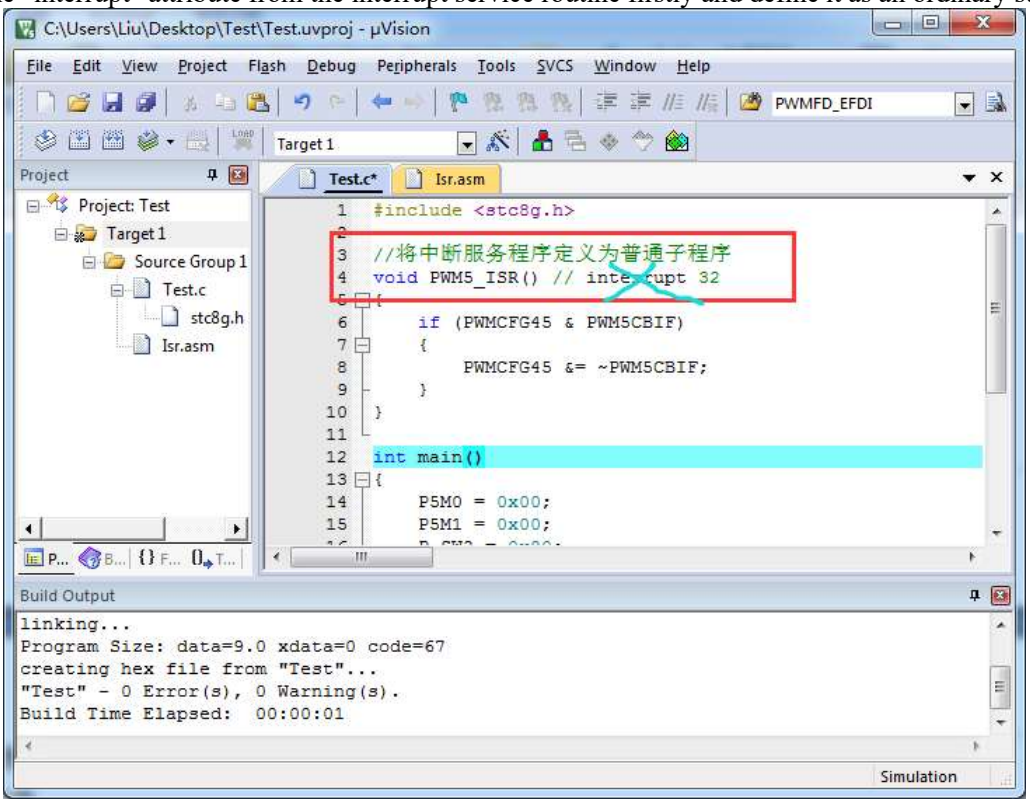


The execution effect is the same as Method 1. This method is applicable to the situation where multiple interrupt numbers greater than 31 need to be remapped.

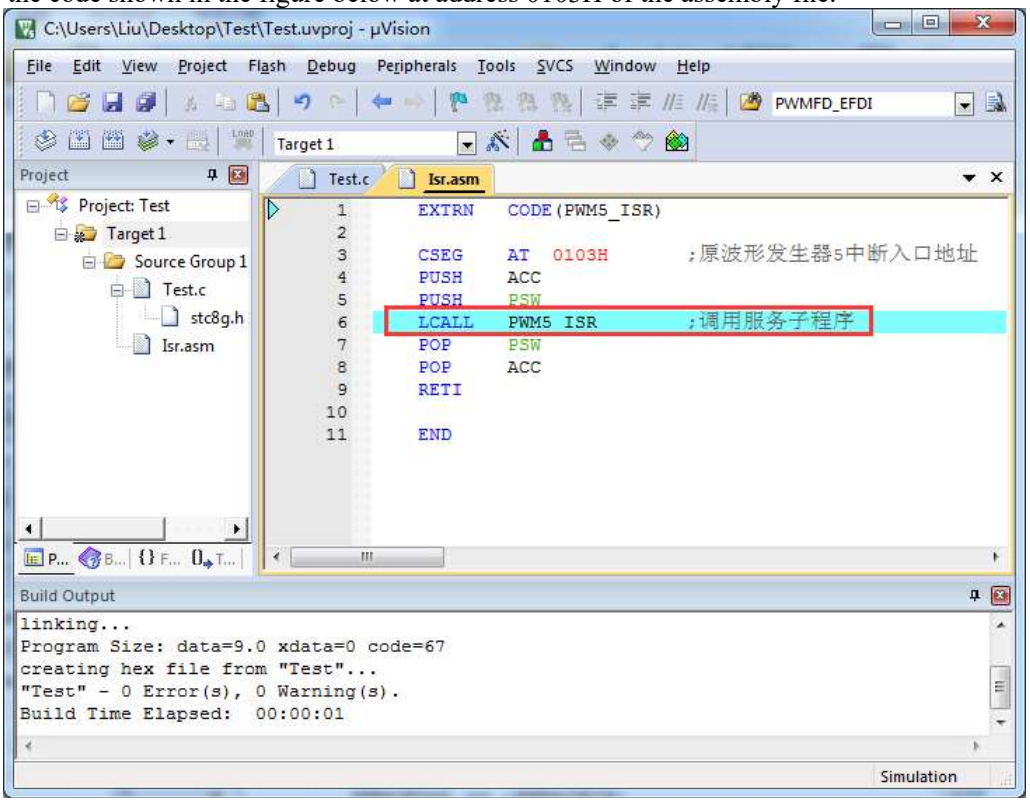
Method 3: Define the interrupt service routine as a subroutine, and then use the LCALL instruction in the interrupt entry address in the assembly code to execute the service routine.

The steps are as follows:

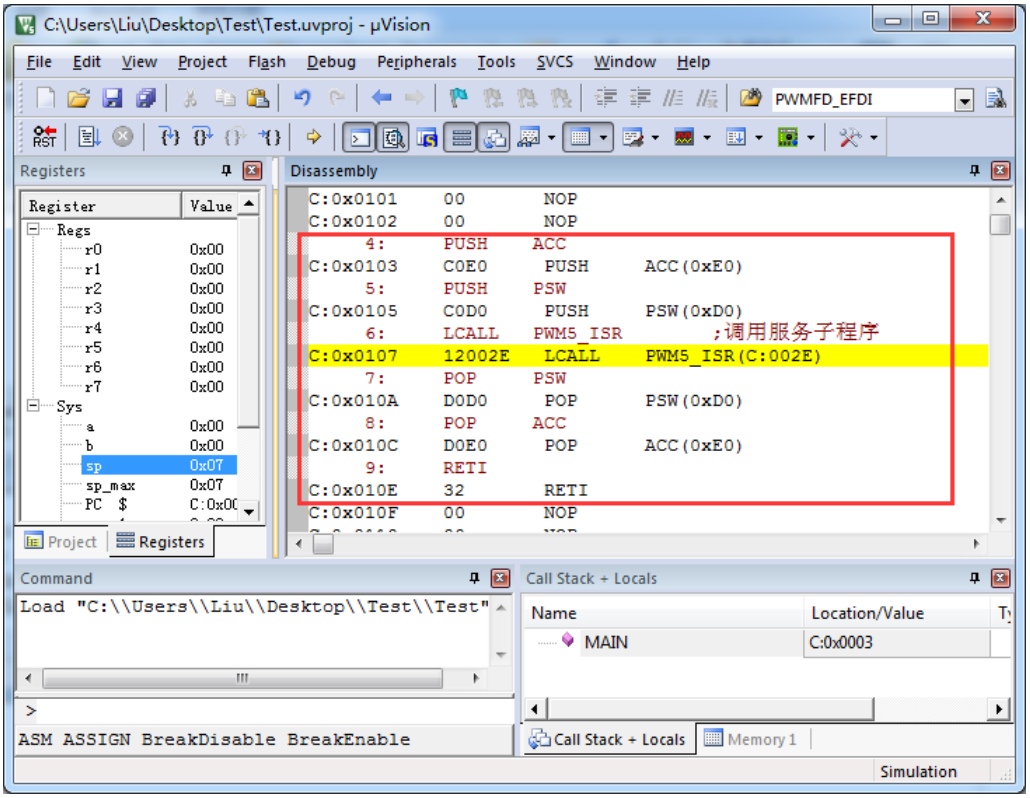
1. Remove the "interrupt" attribute from the interrupt service routine firstly and define it as an ordinary subroutine.



2. Then enter the code shown in the figure below at address 0103H of the assembly file.



3. After compiling, you can find the interrupt service routine at the address of 0103H.



This method does not need to remap interrupt entries. But there is a problem with this method. It requires the user to check the disassembly code of the C program to determine which registers need to be pushed onto the stack in the assembly file. PSW, ACC, B, DPL, DPH and R0 ~ R7 are included generally. In addition to the PSW must be pushed onto the stack, registers which are used in the user subroutine must be pushed onto the stack.

Appendix Q Electrical Characteristics

Q.1 Absolute Maximum Rating

Parameters	Minimum	Maximum	UNIT	Description
Storage temperature	-55	+125	℃	
Operating temperature	-40	+85	℃	If the operating temperature is higher than 85°C (such as around 125°C), due to the large temperature drift of the internal IRC clock frequency at high temperature, it is recommended to use an external high temperature clock or crystal oscillator. In addition, when the temperature is high, the frequency does not run fast, and it is recommended to use a working frequency below 24M; if the system must run at a higher temperature, please use an external high-reliability low-frequency active clock. If the working temperature is around -55°C, the working voltage should not be too low. It is strongly recommended that the MCU-VCC voltage should not be lower than 3.0V. In addition, the rising speed of the power supply must also be as fast as possible, preferably at the millisecond level.
Operating Voltage	1.9	5.5	V	
VDD to ground voltage	-0.3	+5.5	V	
I/O port to ground voltage	-0.3	VDD+0.3	V	

Q.2 DC ELECTRICAL CHARACTERISTICS (3.3V)

(VSS=0V, VDD=3.3V, test temperature =25℃)

Symbol	Parameter	Limits				Test Conditions
		MIN	TYP	MAX	UNIT	
I_{PD}	Power-down mode current	-	0.4	-	uA	
I_{WKT}	Power-down Wake-up timer	-	1.5	-	uA	
I_{LVD}	Low-voltage detection module	-	10	-	uA	
ICMP	Comparator power consumption	-	90	-	uA	
I_{IDL}	Idle mode current (internal 32KHz)	-	0.48	-	mA	Equivalent to 0.5M of traditional 8051
	Idle mode current (6MHz)	-	0.88	-	mA	Equivalent to 79M of traditional 8051
	Idle mode current (12MHz)	-	1.0	-	mA	Equivalent to 158M of traditional 8051
	Idle mode current (24MHz)	-	1.16	-	mA	Equivalent to 317M of traditional 8051

I_{NOR}	Normal mode current（internal 32KHz）	-	0.48	-	mA	Equivalent to 0.5M of traditional 8051
	Normal mode current（500KHz）	-	0.88	-	mA	Equivalent to 7M of traditional 8051
	Normal mode current（600KHz）	-	0.88	-	mA	Equivalent to 8M of traditional 8051
	Normal mode current（700KHz）	-	0.90	-	mA	Equivalent to 9M of traditional 8051
	Normal mode current（800KHz）	-	0.91	-	mA	Equivalent to 11M of traditional 8051
	Normal mode current（900KHz）	-	0.91	-	mA	Equivalent to 12M of traditional 8051
	Normal mode current（1MHz）	-	0.94	-	mA	Equivalent to 13M of traditional 8051
	Normal mode current（2MHz）	-	1.05	-	mA	Equivalent to 26M of traditional 8051
	Normal mode current（3MHz）	-	1.17	-	mA	Equivalent to 40M of traditional 8051
	Normal mode current（4MHz）	-	1.26	-	mA	Equivalent to 53M of traditional 8051
	Normal mode current（5MHz）	-	1.40	-	mA	Equivalent to 66M of traditional 8051
	Normal mode current（6MHz）	-	1.49	-	mA	Equivalent to 79M of traditional 8051
	Normal mode current（12MHz）	-	2.09	-	mA	Equivalent to 158M of traditional 8051
	Normal mode current（24MHz）	-	3.16	-	mA	Equivalent to 317M of traditional 8051
V_{IL1}	Input low voltage	-	-	0.99	V	Turn on Schmitt trigger
		-	-	1.07	V	Turn off Schmitt trigger
V_{IH1}	Input high voltage1(general I/O)	1.18	-	-	V	Turn on Schmitt trigger
		1.09	-	-	V	Turn off Schmitt trigger
V_{IH2}	Input high voltage2(RST pin)	1.18	-	0.99	V	
I_{OL1}	Output low-level sink current	-	20	-	mA	Port voltage 0.45V
I_{OH1}	Output high level current (bi-direction mode)	200	270	-	uA	
I_{OH2}	Output high level current (Push-pull mode)	-	20	-	mA	Port voltage 2.4V
I_{IL}	Logic 0 input current	-	-	50	uA	Port voltage 0V
I_{TL}	Logical 1 to 0 transition current	100	270	600	uA	Port voltage 2.0V
R_{PU}	I/O port pull-up resistor	5.8	5.9	6.0	KΩ	
I/O speed	I/O high current drive, I/O fast conversion		25		MHz	PxDR=0, PxSR=0
	I/O high current drive, I/O fast conversion		22		MHz	PxDR=1, PxSR=0
	I/O low current drive, I/O slow conversion		16		MHz	PxDR=0, PxSR=1
	I/O low current drive, I/O slow conversion		12		MHz	PxDR=1, PxSR=1
Comparators	Fastest speed		10		MHz	Turn off all analog and digital filtering
	Analog filter time		0.1		us	
	Digital filter time		0		system clock	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
IPD2	Power-down mode power consumption when the comparator is enabled	-	400	-	uA	
IPD3	Power-down mode power consumption when LVD is enabled	-	470	-	uA	

Q.3 DC ELECTRICAL CHARACTERISTICS (5V)

(VSS=0V, VDD=5.0V, test temperature =25℃)

Symbol	Parameter	Limits				Test Conditions
		MIN	TYP	MAX	UNIT	
I_{PD}	Power-down mode current	-	0.6	-	uA	
I_{WKT}	Power-down Wake-up timer	-	4.4	-	uA	
I_{LVD}	Low-voltage detection module	-	30	-	uA	
ICMP	Comparator power consumption	-	90	-	uA	
I_{IDL}	Idle mode current (internal 32KHz)	-	1.25	-	mA	Equivalent to 0.5M of traditional
	Idle mode current (6MHz)	-	0.58	-	mA	Equivalent to 79M of traditional 8051
	Idle mode current (12MHz)	-	0.98	-	mA	Equivalent to 158M of traditional 8051
	Idle mode current (24MHz)	-	1.10	-	mA	Equivalent to 317M of traditional 8051
I_{NOR}	Normal mode current（internal 32KHz）	-	0.58	-	mA	Equivalent to 0.5M of traditional 8051
	Normal mode current（500KHz）		0.97		mA	Equivalent to 7M of traditional 8051
	Normal mode current（600KHz）		0.97		mA	Equivalent to 8M of traditional 8051
	Normal mode current（700KHz）		1.00		mA	Equivalent to 9M of traditional 8051
	Normal mode current（800KHz）		1.01		mA	Equivalent to 11M of traditional 8051
	Normal mode current（900KHz）		1.01		mA	Equivalent to 12M of traditional 8051
	Normal mode current（1MHz）		1.03		mA	Equivalent to 13M of traditional 8051
	Normal mode current（2MHz）		1.15		mA	Equivalent to 26M of traditional 8051
	Normal mode current（3MHz）		1.27		mA	Equivalent to 40M of traditional 8051
	Normal mode current（4MHz）		1.35		mA	Equivalent to 53M of traditional 8051
	Normal mode current（5MHz）		1.49		mA	Equivalent to 66M of traditional 8051
	Normal mode current（6MHz）	-	1.59	-	mA	Equivalent to 79M of traditional 8051
	Normal mode current（12MHz）	-	2.19	-	mA	Equivalent to 158M of traditional 8051
	Normal mode current（24MHz）	-	3.27	-	mA	Equivalent to 317M of traditional 8051
V_{IL1}	Input low voltage	-	-	1.32	V	Turn on Schmitt trigger
		-	-	1.48	V	Turn off Schmitt trigger
V_{IH1}	Input high voltage1(general I/O)	1.60	-	-	V	Turn on Schmitt trigger
		1.54	-	-	V	Turn off Schmitt trigger
V_{IH2}	Input high voltage2(RST pin)	1.60	-	1.32	V	
I_{OL1}	Output low-level sink current	-	20	-	mA	Port voltage 0.45V
I_{OH1}	Output high level current (bi-direction mode)	200	270	-	uA	
I_{OH2}	Output high level current (Push-pull mode)	-	20	-	mA	Port voltage 2.4V
I_{IL}	Logic 0 input current	-	-	50	uA	Port voltage 0V
I_{TL}	Logical 1 to 0 transition current	100	270	600	uA	Port voltage 2.0V
R_{PU}	I/O port pull-up resistor	4.1	4.2	4.4	KΩ	
I/O	I/O high current drive, I/O fast conversion		36		MHz	PxDR=0, PxSR=0

speed	I/O high current drive, I/O fast conversion		32		MHz	PxDR=1, PxSR=0
	I/O low current drive, I/O slow conversion		26		MHz	PxDR=0, PxSR=1
	I/O low current drive, I/O slow conversion		22		MHz	PxDR=1, PxSR=1
Comparators	Fastest speed		10		MHz	Turn off all analog and digital filtering
	Analog filter time		0.1		us	
	Digital filter time		0		system clock	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
IPD2	Power-down mode power consumption when the comparator is enabled	-	460	-	uA	
IPD3	Power-down mode power consumption when LVD is enabled	-	520	-	uA	

Q.4 Internal IRC temperature drift characteristic (reference temperature 25℃)

Temperature	Range		
	MIN	TYP	MAX
-40℃～85℃		−1.38%～+1.42%	
-20℃～65℃		−0.88%～+1.05%	

Q.5 Low voltage reset threshold voltage (test temperature 25℃)

Level	Voltage		
	MIN	TYP	MAX
POR		（1.69V~1.82V）	
LVR0		2.0V（1.88V~1.99V）	
LVR1		2.4V（2.28V~2.45V）	
LVR2		2.7V（2.58V~2.76V）	
LVR3		3.0V（2.86V~3.06V）	

Appendix R Application Considerations

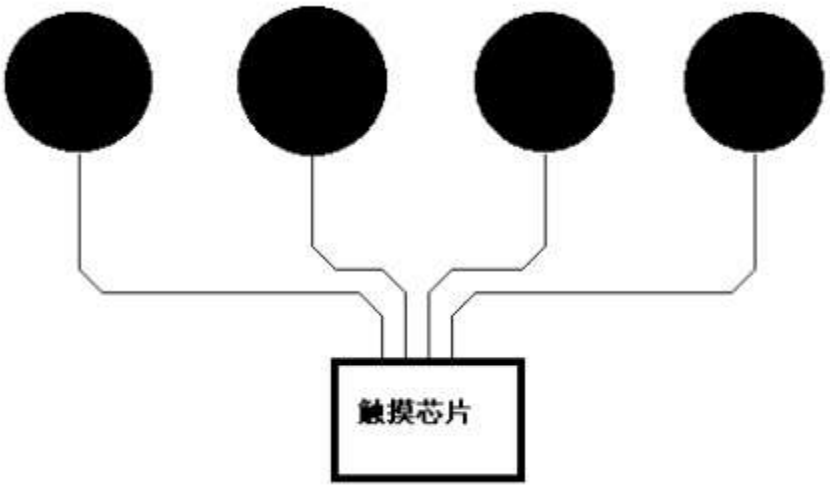
R.1 STC8A8K64D4-64Pin/48Pin series

1. A version of the chip is in example.

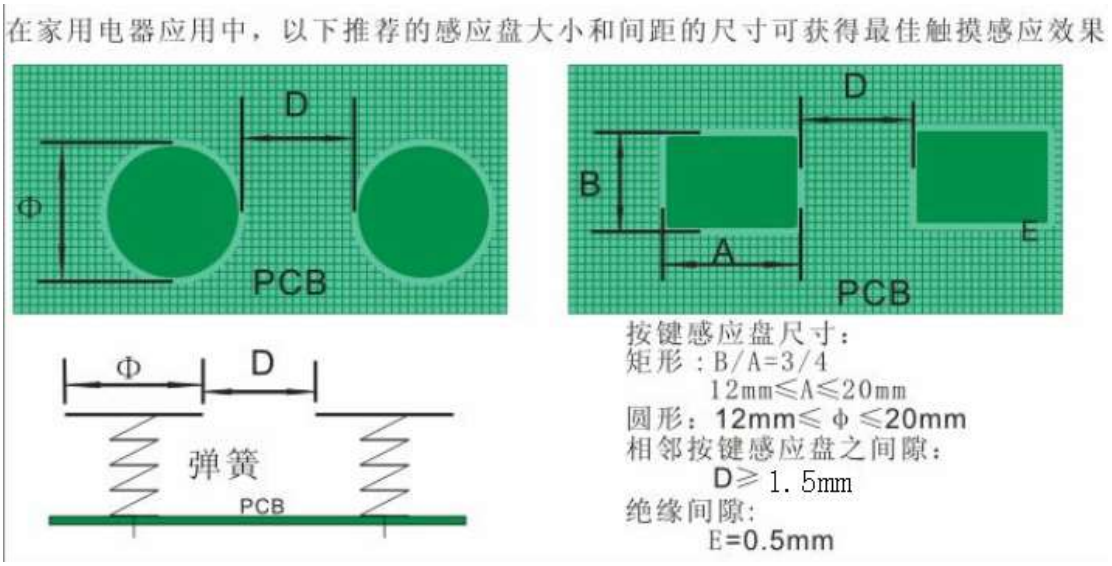
Appendix S PCB design guidance for touch keys

The touch key has strict requirements on PCB design, otherwise its effect will be greatly reduced or even fail.It is recommended that users follow the following principles when designing PCB:

1. **Follow the basic principles of common digital-analog hybrid circuit design.**
   The capacitive touch button module integrates an analog circuit for precise capacitance measurement, so it should be treated as an independent analog circuit when designing the PCB. Follow the basic principles of common digital-analog hybrid circuit design.
2. **Use star grounding**
   The ground wire of the touch chip should not be shared with other circuits. It should be connected to the ground point of the board's power input separately, which is usually called "star grounding".
3. **The impact of noise generated on the power supply on the touch chip**
   The power ripple and noise should be as small as possible. It is best to use an independent trace to take power from the power supply point of the board and add filtering measures. Do not share the power circuit with other circuits.
4. **The connection between the IC and the sensing board should be as long as possible, so that it has an approximate distributed capacitance, as shown in the figure below.**



5. **The size and gap of the key induction plate (capacitive sensor)**
   In the case of meeting the aesthetic design requirements of the panel, the optimal touch sensing effect must be obtained through a reasonable arrangement of the size of the sensing plate and the interval size. The induction plate is placed on the bottom layer, and the IC is also placed on the bottom layer. There should be no vias between the induction plate and the IC. The distance between the edges of adjacent sensing plates is preferably at least 1.5mm (dimension D in the figure below). If the PCB area allows, try to use a larger distance. The distance between the copper paving and the induction plate is 0.5mm (dimension E in the figure below).



6. Copper plating

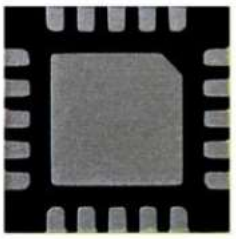
The bottom layer can be covered with grid copper or solid copper. Note that the distance between the copper and the induction plate is 0.5mm. The silk screen information of the top layer is printed on the button. The frame shape of the silk screen is the same as the bottom sensor plate. The top layer corresponding to the bottom sensor plate should not be covered with copper, otherwise the touch action will be shielded. The copper on the top layer is the same as the copper on the bottom layer.

7. Wiring processing

It is better to use a smaller line width for the connection between the sensor plate and the IC, such as between 10 and 15 mils. The connection between the induction plate and the touch chip should not cross the lines with strong interference, high frequency, and high current. Do not run other signal lines within 1.5mm of the connection between the sensor pad and the touch chip, the farther away the better. The top layer corresponds to the bottom sensor plate and connecting line, it is best not to put any line.

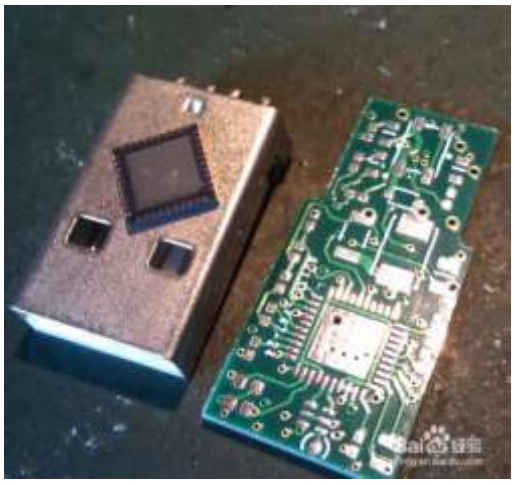
Appendix T QFN/DFN packaged components welding method

In the package form of STC products, the more popular QFN and DFN packages have been added. Since the pins of the chip in this package are at the bottom of the chip, manual soldering is difficult. There are small companies on the market that specialize in welding engineering samples, which can undertake engineering sample proofing. If users need to weld by themselves, please refer to the following welding method.



1. Firstly, you need to prepare the following tools: electric soldering iron, hot air gun, tweezers, fixing frame and other tools

2. The PCB boards and chips that need to be soldered are as follows:

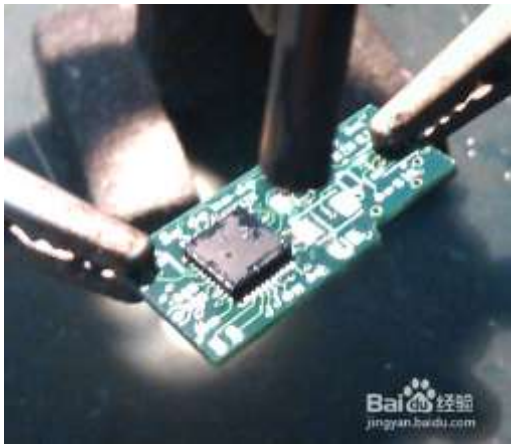


3. Tin the pads of the chip on the board:

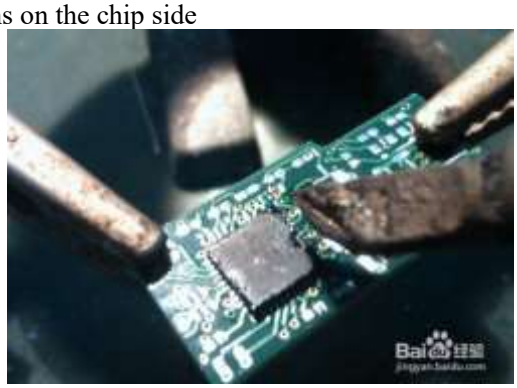
4. Then apply tin to the bottom of the chip. After the tin is applied, it should be flattened to reduce the tin as much as possible, but it cannot be eliminated.

5.Adjust the temperature of the hot air gun, the actual air output is about 240 degrees, because the quality of the air gun is different, adjust it according to the actual situation.

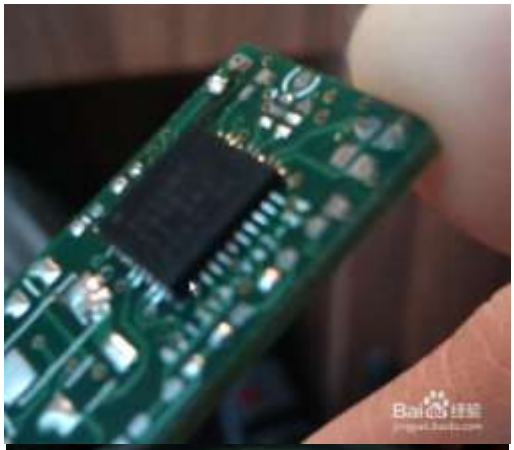
6. Put the chip on the pad, be sure to place it straight, and then blow it with a hot air gun at a uniform speed until the tin melts, usually within 20 seconds.

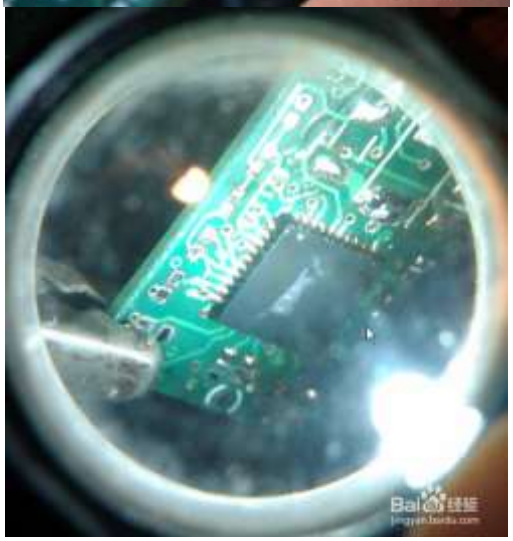


7. Use a soldering iron to tin the pins on the chip side



8. The effect after welding





Appendix U Precautions for STC8A8K64D4 series MCU to replace STC8A8K64S4A12 series

I/O port

After the STC8A8K64D4 series MCU is powered on, the I/O mode is different from that of the STC8A8K64S4A12 series. All I/O ports of STC8A8K64S4A12 series single-chip microcomputers are in 8051 quasi-bidirectional port mode after power-on. In the I/O of STC8A8K64D4 series single-chip microcomputers, except for ISP download pins P3.0/P3.1 which are quasi-bidirectional port modes, all the other I/O ports are in high-impedance input mode after power on. All I/O ports of the traditional 8051 are in quasi-bidirectional port mode and output high level after power-on. Often customers use I/O to drive motors or LED lights in their systems, so there will be moments when the single-chip microcomputer is powered on. Move it once or the LED will flash once. The I/O of the STC8A8K64D4 series is in high-impedance input mode after power-on, which can avoid this kind of malfunction of the motor and LED.

Except ISP download pin P3.0/P3.1 which is quasi-bidirectional port mode, all other I/O ports of STC8A8K64D4 series are in high-impedance input mode after power-on, so the two registers PxM0 and PxM1 must be used to set the I/O working mode before the I/O ports output signals.

Reset foot

The P5.4 port of the STC8A8K64D4 series and STC8A8K64S4A12 series is generally used as a normal I/O port. When the user sets P5.4 as the reset pin function during ISP download, the P5.4 port is the reset of the microcontroller Pin (RESET pin). For the STC8A8K64S4A12 series, when the reset pin is high, the microcontroller is in the reset state, and when the reset pin is low, the microcontroller is released from the reset state. The reset levels of STC8A8K64D4 series and STC8A8K64S4A12 series are reversed, that is, for STC8A8K64D4 series, when the reset pin is low, the microcontroller is in the reset state, and when the reset pin is high, the microcontroller is released from the reset state.

Therefore, when the user enables the reset pin function of port P5.4, it is necessary to pay attention to the reset level.

EEPROM

The waiting time for EEPROM erasing and programming of STC8A8K64S4A12 series is set by Bit2-Bit0 of the register IAP_CONTR. The setting is only an approximate frequency range value. The STC8A8K64D4 series adds a new register IAP_TPS (SFR address: 0F5H), dedicated to setting EEPROM erasing In addition to the waiting time for programming, and the user does not need to calculate, just fill in IAP_TPS directly according to the current CPU working frequency, and the hardware will automatically calculate the waiting time. (For example: the current CPU operating frequency is 24MHz, you only need to fill in 24 to IAP_TPS).

ADC

The ADCs of the STC8A8K64D4 series are fully functionally compatible with the STC8A8K64S4A12 series. Based on the STC8A8K64S4A12 series ADC, the STC8A8K64D4 series adds new functions such as external trigger function and automatic multiple conversion and averaging.

Comparators

The positive input of the STC8A8K64D4 series comparator is 4-way optional, and the negative input is two-way optional. The input selection is set in the register CMPEXCFG. The STC8A8K64S4A12 series are inconsistent.

SPI

The 4 SPI clock frequencies of the STC8A8K64D4 series are: SYSclk/4, SYSclk/8, SYSclk/16 and

SYSclk/2. The 4 SPI clock frequencies of the STC8A8K64S4A12 series are: SYSclk/4, SYSclk/8, SYSclk/16 and SYSclk/32.

■ PCA/CCP/PWM

Among the PCA-related SFRs of the STC8A8K64D4 series, the SFRs of the first three groups of modules are the same as those of the STC8A8K64S4A12 series, and the control registers of the fourth group of modules (PCA3/CCP3/PWM3) are in the XFR area and are not compatible with the STC8A8K64S4A12 series. (specifically CCAPM3, CCAP3L, CCAP3H and PCA_PWM3).

■ 15 位增强型 PWM

The enhanced PWM related SFR addresses of the STC8A8K64D4 series are not compatible with the STC8A8K64S4A12 series.

Appendix V Update Records

- **2022/3/1**
  1. Correct typos in the document

- **2022/2/18**
  1. Correct typos in the document
  2. Fixed the incorrect description of ADC data structure in the DMA chapter

- **2021/12/17**
  1. The name of the reset pin in all pin diagrams is changed to NRST
  2. Corrected the timing calculation formula of timer 2/3/4

- **2021/11/30**
  1. Correct typos in the document
  2. All BMMs are renamed to DMA
  3. Add EEPROM application example program
  4. Update the selection price list
  5. Add the package dimension drawing of LQFP44
  6. Add a detailed description to the read-only special function register (CHIPID)
  7. Added appendix "STC Simulation User Manual" chapter

- **2021/10/16**
  8. Modify some chapter titles
  9. Correct typos found in the document
  10. Update the port switching table for 8-bit data and 16-bit data in the LCM chapter
  11. Added the description of external interrupt to the interrupt source table in the Interrupt System chapter
  12. Take screenshots using the new version of the software in the Emulator chapter in the appendix
  13. Added the chapter "How to Test I/O Ports" in the appendix

- **2021/9/26**
  14. Update some descriptions in the features and price list
  15. Update technical support calls
  16. Add a sample program for serial port DMA timeout processing and data verification
  17. Added notes on using the reset interrupt in the Enhanced PWM chapter

- **2021/8/26**
  1. Correct the comment error in the ADC chapter example program

- **2021/6/29**
  1. Add LQFP44 pin diagram

- **2021/6/26**
  1. Correct the wrong description in the comparator structure diagram
  2. Revise the formula for calculating the enhanced PWM output frequency

- **2021/5/10**

1. Added the description of timer 2/3/4 interrupt flag bits
2. Modify the DMA read trigger control bit of serial port 1/2/3/4

2021/4/28

1. Create STC8A8K64D4 series MCU technical reference manual document