STC32G Series Microcontroller Technical Reference Manual

- 10 32-bit Accumulators
- 16 16-Bit Accumulators
- 16 8-Bit Accumulators
- 32-bit add and subtract instructions
- 16-bit multiply and divide instructions
- 32-bit multiplication and division (MDU32)
- 32-bit Arithmetic Compare Instruction
- All SFRs (80H~FFH) support bit addressing.
- ebdata (20H~7FH) all support bit addressing.
- Single-clock 32/16/8-bit data read/write (edata)
- Single clock port read/write
- Theoretical stack depth up to 64K (actually depends on edata)
- FreeRTOS for STC32G12K128: Official STC port of the efficient and stable version has been released!
- RT-Thread for STC32G12K128: RT-Thread is officially being ported!
- Compiler: KEIL C251 Compiler

Information updated: **2022/11/14**

directory (on computer hard drive)

1 outlined

STC32G series microcontrollers are microcontrollers that do not require external crystal and external reset. They are 32-bit 8051 microcontrollers aiming at super anti-jamming/ultra-low price/high speed/low power consumption, and at the same operating frequency, STC32G series microcontrollers are about 70 times faster than the traditional 8051.

STC32G series microcontrollers are single-clock (1T) microcontrollers produced by STC, which are wide voltage/high speed/high reliability/low power consumption/strong anti-static/stronger anti-jamming new generation 32-bit 8051 microcontrollers with super encryption.

MCU internal integration of high-precision R/C clock (±0.3%, +25℃ at room temperature), -1.38%~+1.42% temperature drift (-40℃~+85℃).
-0.88%~+1.05% temperature fluctuation (-20℃~+65℃). 4MHz~33MHz wide range can be set during ISP programming, which can completely eliminate the need for expensive external crystals and external reset circuits (high reliability reset circuits have been integrated internally, and the 4-stage reset threshold voltage can be selected during ISP programming).

There are 4 optional clock sources inside the MCU: internal high-precision IRC clock (frequency can be adjusted during ISP programming), internal 32KHz low-speed IRC, external 4M~33M crystal or external clock signal, and internal PLL output clock. The clock source can be freely selected by the user code, and after the clock source is selected, it can be divided by an 8-bit frequency divider to provide the clock signal to the CPU and various peripherals (e.g. timer, serial port, SPI, etc.).

The MCU provides two low-power modes: IDLE mode and STOP mode; in IDLE mode, the MCU stops supplying clock to the CPU, the CPU has no clock, and the CPU stops executing instructions, but all the peripherals are still in working state, and the power consumption is about 1.3mA (6MHz working frequency); STOP mode is the main clock stopping mode, i.e., the traditional power-down mode/power-down mode/stop mode, and the power consumption can be reduced to less than 1uA. STOP mode is the main clock stop mode, i.e., the traditional power-down mode/power-off mode/shutdown mode, in which the CPU and all the peripherals stop working, and the power consumption can be reduced to less than 1uA.

The MCU provides a rich set of digital peripherals (4 serial ports, 5 timers, 2 sets of 16-bit advanced PWM timers capable of outputting complementary/symmetrical/deadband control signals for 3-phase motor control, as well as I2C, SPI, USB, CAN, and LIN) and analogue peripherals (ultra-high-speed 12-bit ADCs and comparators) to meet a wide range of users' design needs.

The STC32G series microcontrollers have 268 powerful instructions, including 32-bit addition and subtraction instructions and 16-bit multiplication and division instructions. Hardware Expansion 32-bit hardware multiplication and division unit MDU32 (contains 32-bit division by 32-bit and 32-bit multiplication by 32-bit).

STC32G series microcontrollers have an enhanced dual data pointer integrated inside. Through the programme control, the data pointer can be automatically incremented or decremented and the two data pointers can be automatically switched.

product	ports	asynchronous serial ports	synchronous serial ports	timers	ADC	high-speed PWM	comparator	SPI	I2C	I2S	USB	CAN	LIN	RTC	DMA	Colour screen driver	I/O discrete drive	MDU32	FPM

2 Features, Price and Pinout

2.1 STC32G12K128-LQFP64/LQFP48/LQFP32/PDIP40

2.1.1 Characteristics and price

➢ Option Price (no external crystal, no external reset, 12-bit ADC, 15 channels)

Microcontroller Model	operating voltage (V)	Flash program memory	edata	xdata			EEPROM	I/O																														Price & Packaging lqfp64 <12mm*12mm	lqfp48 <9mm*9mm>	LQFP32 <9mm*9mm>	PDIP40	Supply Information
STC32G12K64 √ √ √ √	1.9-5.5	64K	4K	8K	2	64K	60	Yes	Yes	Yes	Yes	Yes	2			2	2 2 2	3	Yes	3	Yes		58	Yes	12-bit	Yes	Yes	Yes	Level 4	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
STC32G12K128 Yes Yes Yes Yes Yes	1.9-5.5	128K	4K	8K	2	IAP	60	Yes	Yes	Yes	Yes	Yes	2			2	2 2 2	3	Yes	3	Yes		58	Yes	12 bits	Yes	Yes	Yes	4 levels	Yes	Yes	Yes		Yes ¥4.5	Yes ¥4.5	Yes ¥5	Yes	¥4.5				appear commodity

➢ kernel (computer science)

✓ Ultra-fast 32-bit 8051 core (1T), approximately 70 times faster than traditional 8051

✓ 49 interrupt sources, 4 interrupt priority levels

✓ Supports online simulation

➢ **operating voltage**

✓ 1.9V to 5.5V (when the operating temperature is lower than -40°C, the operating voltage shall not be lower than 3.0V)

➢ **operating temperature**

✓ -40°C to 85°C (internal high-speed IRC (36MHz or less) and external crystal available)

✓ -40°C～125°C (When the temperature is higher than 85°C, please use an external high-temperature-resistant crystal, and keep the operating frequency below 24MHz.)

➢ **Flash memory**

✓ Maximum 128K bytes of FLASH programme memory (ROM) for storing user code

✓ Supports user-configurable EEPROM size, 512-byte single page erase, up to 100,000 erasures or more

✓ Supports hardware USB direct downloads and regular serial port downloads.

✓ Support hardware SWD real-time emulation, P3.0/P3.1 (requires STC-USB Link1D tool)

➢ **SRAM, 12K bytes total**

✓ 4K bytes internal SRAM (edata)

✓ 8K bytes of internal extended RAM (internal xdata)

✓ Note on use: (it is strongly recommended not to declare variables using **idata** and **pdata**)

➢ Clock control

✓ Internal high-precision IRC (adjustable up or down during ISP programming)
- ⌖ Error ± 0.3% (25°C at room temperature)
- ⌖ -1.35% to +1.30% temperature drift (full temperature range, -40°C to 85°C)
- ⌖ -0.76% to +0.98% temperature drift (temperature range, -20°C to 65°C)

✓ Internal 32KHz low-speed IRC (large error)

✓ External crystal (4MHz to 33MHz) and external clock with special internal circuitry for external clock interference, software startable

✓ Internal PLL output clock (Note: 96MHz/144MHz PLL output can be used independently as the clock source for high-speed PWM and high-speed SPI) Users are free to choose from the above four clock sources.

➢ reset (a dislocated joint, an electronic device etc)

✓ hardware reset
- ⌖ Power-on reset, the reset voltage value is 1.7V to 1.9V. (Valid when the chip does not enable the low-voltage reset function)
- ⌖ □□□□□ pin reset, factory P5.4 is I/O port by default, ISP download can set P5.4 pin as reset pin (Note: When setting P5.4)

  (The reset level is low when the pin is a reset pin)
- ⌖ Watchdog overflow reset
- ⌖ Low-voltage detection reset, provides 4 levels of low-voltage detection voltage: 2.0V, 2.4V, 2.7V, 3.0V.

✓ software reset
- ⌖ Write Reset Trigger Register in Software Mode

➢ disruptions

✓ Provides 49 interrupt sources: INT0, INT1, INT2, INT3, INT4, Timer0, Timer1, Timer2, Timer3, Timer4, USART1, USART2, UART3, UART4, ADC analogue-to-digital converter, LVD low-voltage detector, SPI, I² C, comparator, PWMA, PWMB, USB, CAN, CAN2, LIN, LCMIF colour screen interface interrupts, RTC real-time clock, all I/O interrupts (8 groups), DMA receive and transmit interrupts for Serial 1, DMA receive and transmit interrupts for Serial 2, DMA receive and transmit interrupts for Serial 3, DMA receive and transmit interrupts for Serial 4, I2C DMA receive and transmit interrupts, DMA interrupts for SPI, DMA interrupts for ADC, DMA interrupts for LCD driver. DMA interrupt for I2C, DMA interrupt for SPI, DMA interrupt for ADC, DMA interrupt for LCD driver, and DMA interrupt for memory to memory.

✓ Provides 4 levels of interrupt priority

➢ digital peripheral (computing)

✓ 5 16-bit timers: Timer 0, Timer 1, Timer 2, Timer 3, Timer 4, with Mode 3 of Timer 0 with NMI (non-maskable interrupt) function, Timer 0 and Timer 1 mode 0 is 16-bit auto-reload mode

✓ 2 high-speed synchronous/asynchronous serial ports: Serial 1 (USART1), Serial 2 (USART2), baud rate clock source up to FOSC/4. Supports synchronous serial mode, asynchronous serial mode, SPI mode, LIN mode, infrared mode (IrDA), smart card mode (ISO7816).

✓ 2 high-speed asynchronous serial ports: Serial 3, Serial 4, baud rate clock source up to FOSC/4

✓ 2 advanced PWMs for 8-channel (4 complementary symmetrical) PWM with deadband control and external anomaly detection support

✓ SPI: 3 hardware SPIs (one stand-alone SPI, two USARTs in SPI mode) with support for master and slave modes and automatic master/slave switching (note: one stand-alone SPI can support

DMA, two USARTs in SPI do not support DMA)

- ✓ I² C: Supports host mode and slave mode
- ✓ ICE: Hardware Supported Emulation
- ✓ RTC: Supports year, month, day, hour, minute, second, sub-second (1/128th of a second), and supports clock interrupt and a set of alarms
- ✓ USB: USB2.0/USB1.1 compliant full-speed USB, 6 bi-directional endpoints, supports 4 endpoint transfer modes (control transfer, interrupt transfer, batch transfer, and synchronous transfer), each endpoint has a 64-byte buffer
- ✓ CAN: two independent CAN 2.0 control units
- ✓ LIN: 3 hardware LINs (one stand-alone LIN, two USARTs in LIN mode) and one stand-alone LIN control unit (supports 1.3 and 2.1)

  version)

- ✓ MDU32: Hardware 32-bit multiply-divider (contains 32-bit division by 32-bit, 32-bit multiplication by 32-bit)
- ✓ I/O port interrupt: all I/Os support interrupt, each group of I/O interrupt has independent interrupt entry address, all I/O interrupts can support 4 interrupt modes: high level interrupt, low level interrupt, rising edge interrupt, falling edge interrupt, I/O port interrupt can be wake-up from power-down, and there are 4 levels of interrupt priority.
- ✓ LCD Driver Module: Supports 8080 and 6800 interfaces and 8-bit and 16-bit data widths.
- ✓ DMA: Supports SPI shifted receive data to memory, SPI shifted send data to memory, I2C send data to memory, I2C receive data to memory, Serial port 1/2/3/4 receive data to memory, Serial port 1/2/3/4 send data to memory, ADC auto-sampling data to memory (while calculating the average value), LCD driver send data to memory, and memory-to-memory data copying. data to memory, LCD driver sending data to memory, and memory-to-memory data copying.
- ✓ Hardware numeric ID: 32 bytes supported

➢ analogue peripherals

- ✓ ADC: Ultra-high-speed ADC, supports 12-bit high-precision analogue-to-digital conversion of 15 channels (channels 0 to 14), channel 15 of the ADC is used to test the internal reference voltage (the internal reference voltage is adjusted to 1.19V when the chip is shipped from the factory, with an error of ±1%).
- ✓ Comparator: a set of comparators

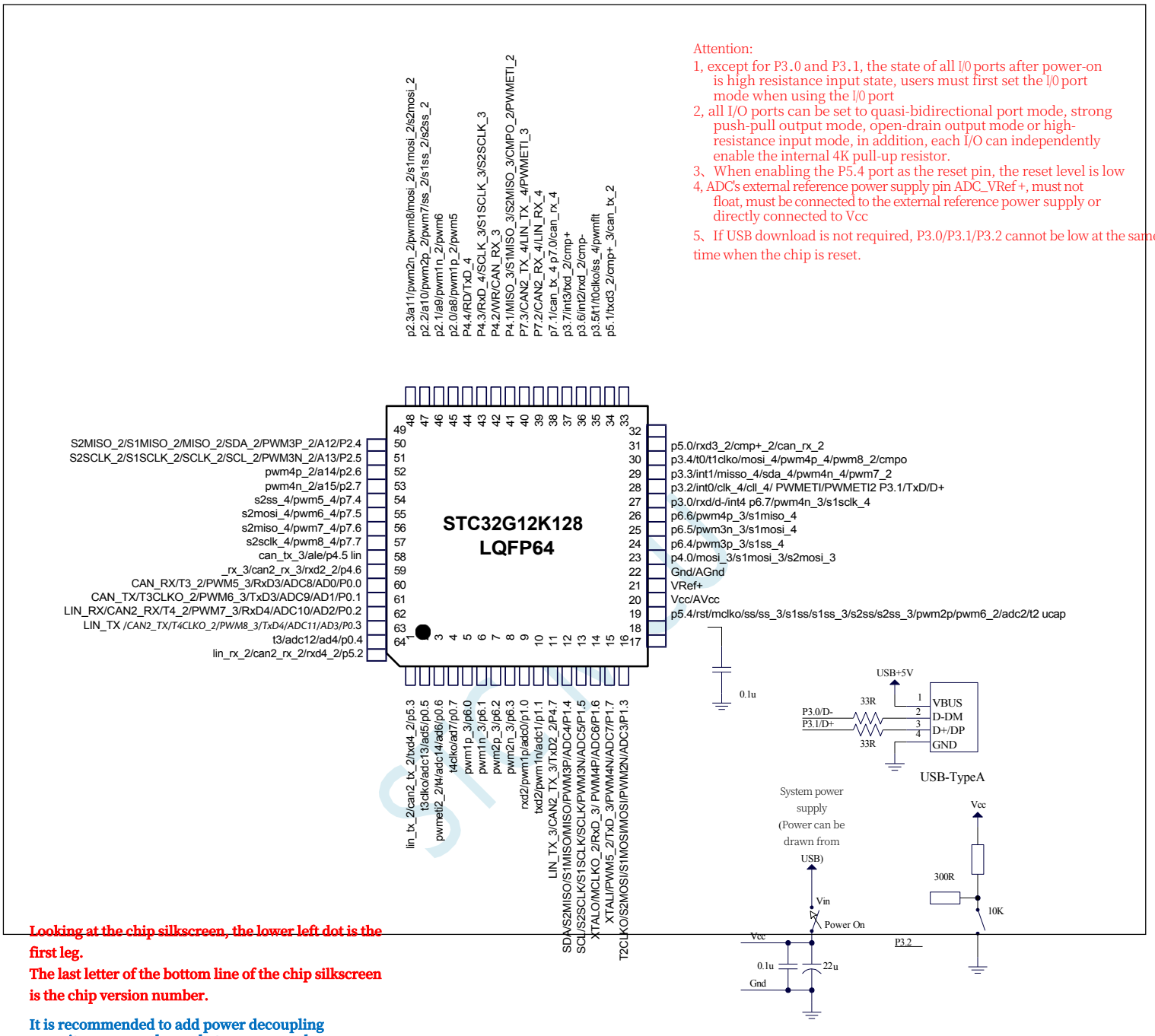
➢ GPIO

- ✓ Up to 60 GPIOs: P0.0~P0.7, P1.0~P1.7 (no P1.2), P2.0~P2.7, P3.0~P3.7, P4.0~P4.7, P5.0~P5.4, P6.0~P6.7, P7.0~P7.7
- ✓ All GPIOs support the following four modes: quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode, and high-resistance input mode.
- ✓ Except for P3.0 and P3.1, all other IO ports are in high resistance input state after power up, users must set the IO port mode first when using IO ports.
- ✓ In addition, each I/O can independently enable the internal 4K pull-up resistor.

➢ **seal inside**

- ✓ lqfp64, lqfp48, lqfp32, pdip40

2.1.2 Pinout, Minimum System



Looking at the chip silkscreen, the lower left dot is the first leg.

The last letter of the bottom line of the chip silkscreen is the chip version number.

It is recommended to add power decoupling capacitors 22uF and 0.1uF between Vcc and Gnd nearby, which can remove the power line noise and improve the anti-interference ability.

Now STC's MCUs with hardware USB support downloading of user programs with the included hardware USB because it uses the USB-HID communication protocol and does not require any driver installation.

ISP Download Steps:

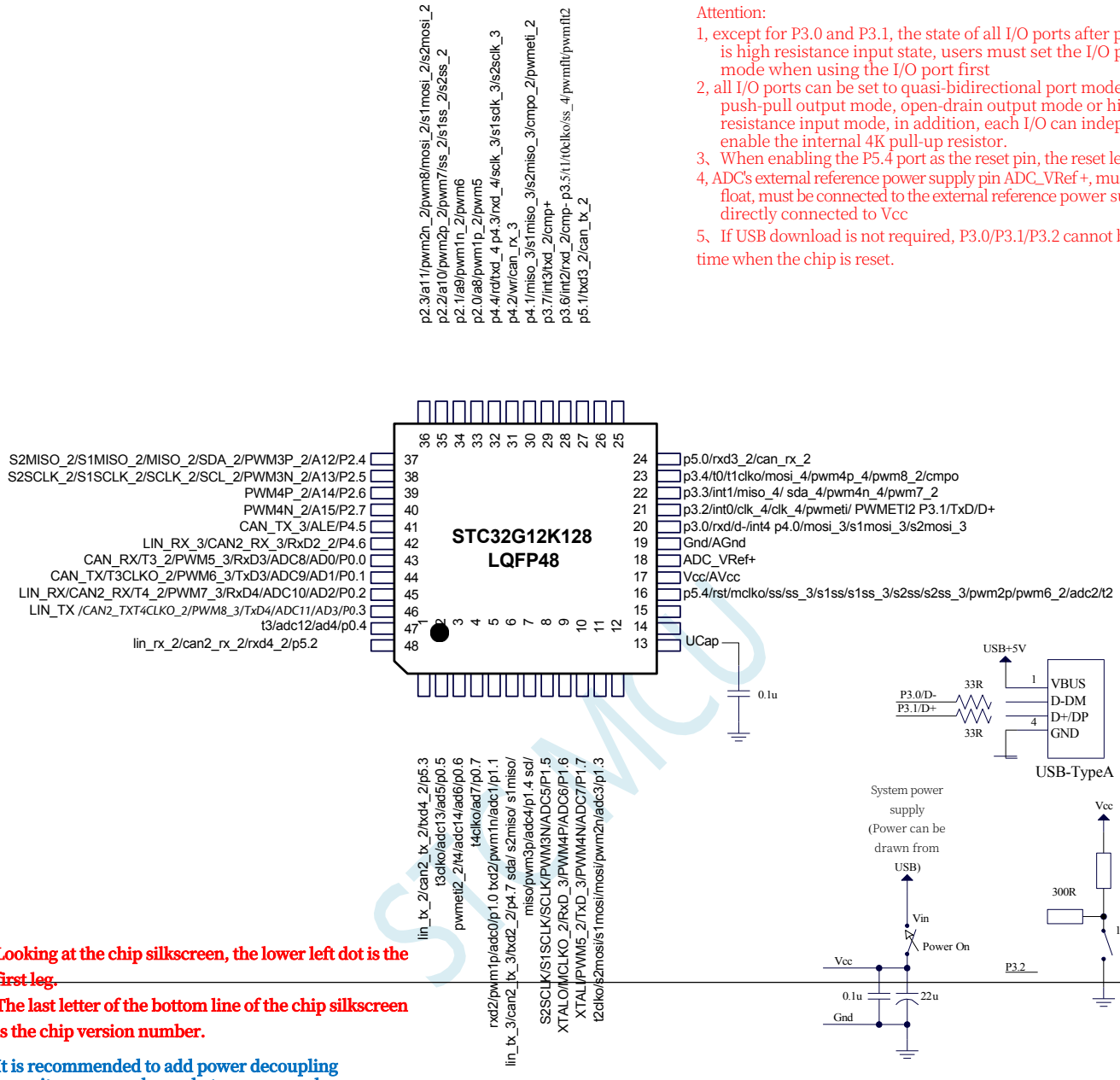
1、D-/P3.0, D+/P3.1 and PC-USB port are connected.

2、Shorten P3.2 and GND, press the P3.2/INT0 button on the board of the laboratory box.

3. Repower the target chip. If the target chip has been powered off, it can be powered on directly; if the target chip is powered on, it needs to be powered off and powered on again (cold

start). Wait for the STC-ISP download software to automatically identify the "STC USB Writer (HID1)" identified, it has nothing to do with the state of P3.2 (at this time there is no need to keep pressing the P3.2 port, keep pressing the handle is tired, it does not matter, will be a problem to break the button).

4, click the download software in the "download / programming" button (Note: USB download and serial port download the order of operation is different, **do not click the download button, be sure to wait until the computer recognises the "STC USB Writer (HID1)" device before starting to download) "(Note: The order of operation of USB download is different from that of serial port download.)**



Attention:

1, except for P3.0 and P3.1, the state of all I/O ports after power-on is high resistance input state, users must set the I/O port mode when using the I/O port first

2, all I/O ports can be set to quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode or high-resistance input mode, in addition, each I/O can independently enable the internal 4K pull-up resistor.

3、When enabling the P5.4 port as the reset pin, the reset level is low

4, ADC's external reference power supply pin ADC_VRef+, must not float, must be connected to the external reference power supply or directly connected to Vcc

5、If USB download is not required, P3.0/P3.1/P3.2 cannot be low at the same time when the chip is reset.

Looking at the chip silkscreen, the lower left dot is the ~~first leg~~.

The last letter of the bottom line of the chip silkscreen is the chip version number.

It is recommended to add power decoupling capacitors 22uF and 0.1uF between Vcc and Gnd nearby, which can remove the power line noise and improve the anti-interference ability.

Now STC's MCUs with hardware USB support downloading of user programs with the included hardware USB because it uses the USB-HID communication protocol and does not require any driver installation.

ISP Download Steps:

1、D-/P3.0, D+/P3.1 and PC-USB port are connected.

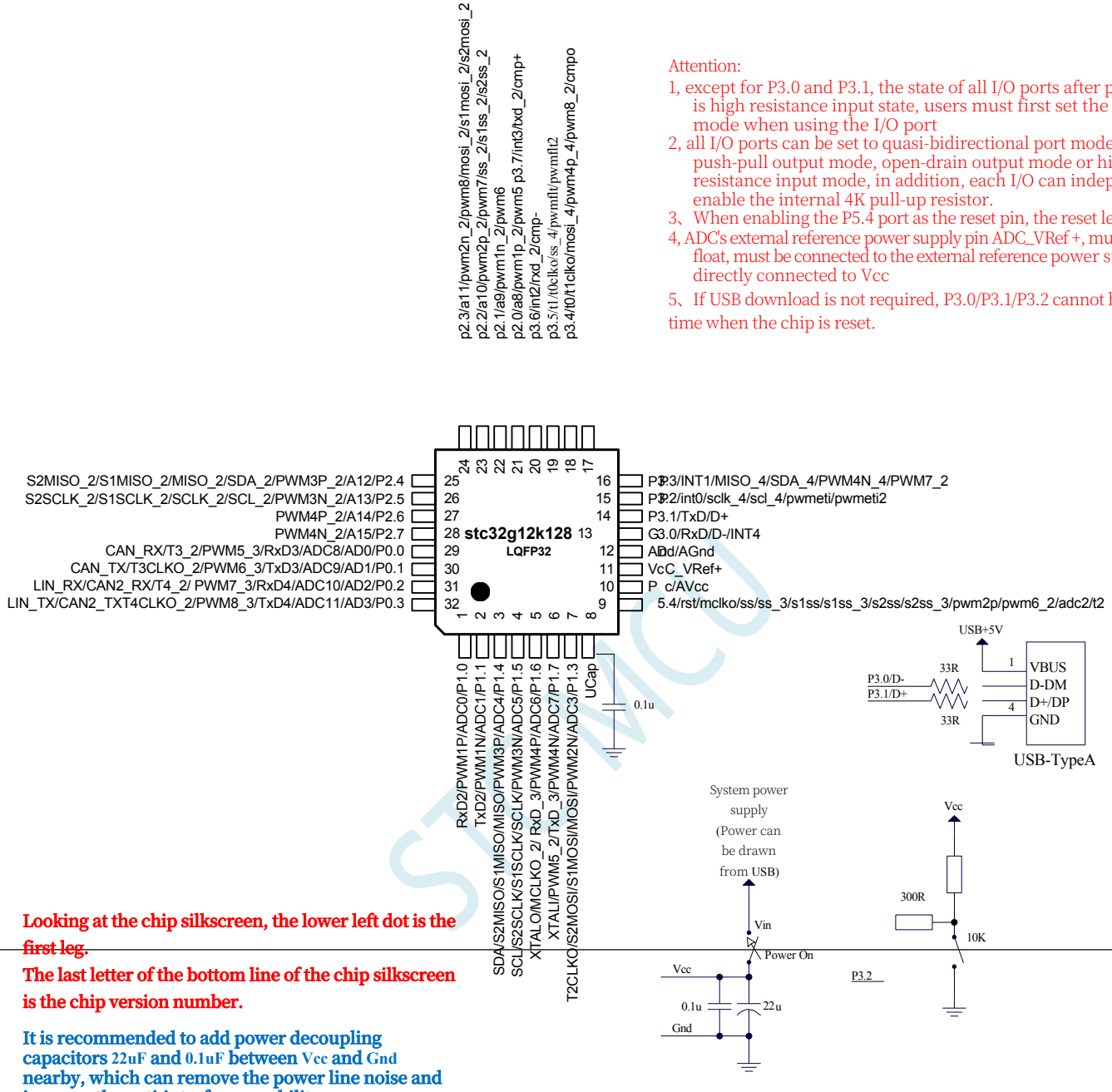
2、Shorten P3.2 and GND, press the P3.2/INT0 button on the board of the laboratory box.

3. Repower the target chip. If the target chip has been powered off, it can be powered on directly; if the target chip is powered on, it needs to be powered off and powered on again (cold start). Wait for the STC-ISP download software to automatically identify the "STC USB Writer (HID1)" identified, it has nothing to do with the state of P3.2 (at this time there is no

need to keep pressing the P3.2 port, keep pressing the handle is tired, it does not matter, will be a problem to break the button).

4, click the download software in the "download / programming" button (Note: USB download and serial port download the order of operation is different, **do not click the download button, be sure to wait until the computer recognises the "STC USB Writer (HID1)" device before starting to download) "(Note: The order of operation of USB download is different from that of serial port download.)**

Hardware USB direct download reference wiring diagram



Now STC's MCUs with hardware USB support downloading of user programs with the included hardware USB because it uses the USB-HID communication protocol and does not require any driver installation.

ISP Download Steps:

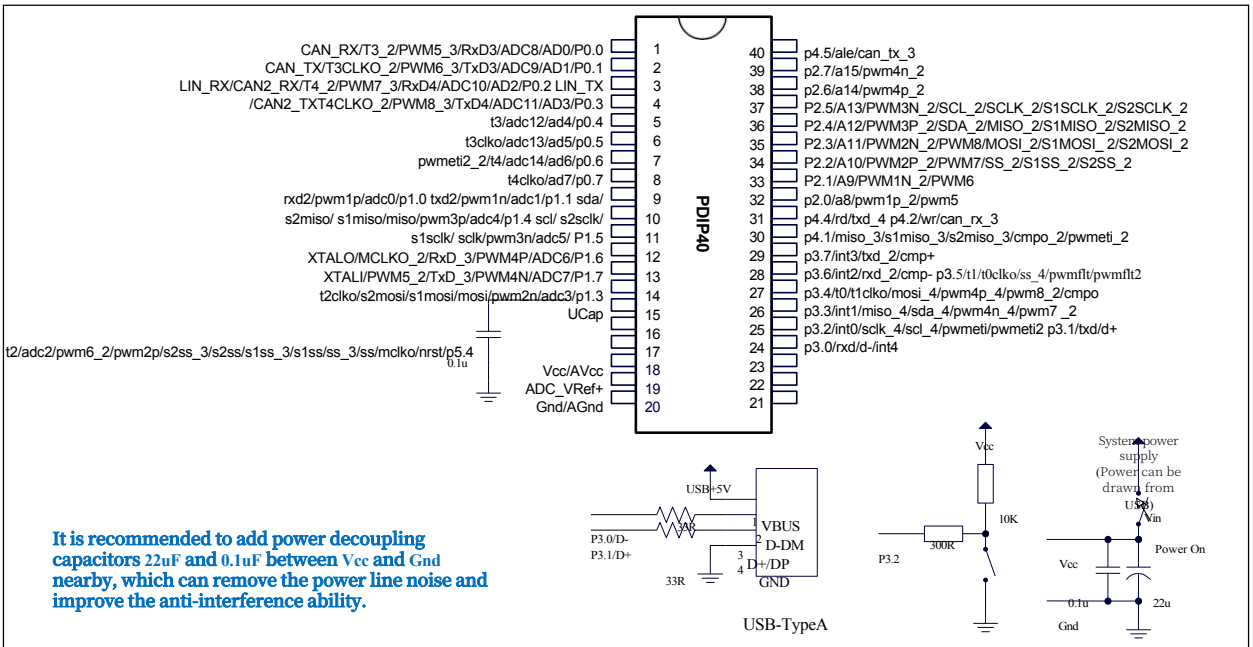
1、D-/P3.0, D+/P3.1 and PC-USB port are connected.

2、Shorten P3.2 and GND, press the P3.2/INT0 button on the board of the laboratory box.

3. Repower the target chip. If the target chip has been powered off, it can be powered on directly; if the target chip is powered on, it needs to be powered off and powered on again (cold

start). Wait for the STC-ISP download software to automatically identify the "STC USB Writer (HID1)" identified, it has nothing to do with the state of P3.2 (at this time there is no need to keep pressing the P3.2 port, keep pressing the handle is tired, it does not matter, will be a problem to break the button).

4, click the download software in the "download / programming" button (Note: USB download and serial port download the order of operation is different, **do not click the download button first, be sure to wait until the computer recognises "STC USB Writer (HID1)" device before starting to download) "(Note: The order of operation of USB download is different from that of serial port download.)**



~~Now STC's MCUs with hardware USB support downloading of user programs with the included~~ hardware USB because it uses the USB-HID communication protocol and does not require any driver installation.

ISP Download Steps:

1、D-/P3.0, D+/P3.1 and PC-USB port are connected.

2、Shorten P3.2 and GND, press the P3.2/INT0 button on the board of the laboratory box.

3. Repower the target chip. If the target chip has been powered off, it can be powered on directly; if the target chip is powered on, it needs to be powered off and powered on again (cold start). Wait for the STC-ISP download software to automatically identify the "STC USB Writer (HID1)" identified, it has nothing to do with the state of P3.2 (at this time there is no need to keep pressing the P3.2 port, keep pressing the handle is tired, it does not matter, will be a problem to break the button).

4, click the download software in the "download / programming" button (Note: USB download and serial port download the order of operation is different, **do not click the download button, be sure to wait until the computer recognises the "STC USB Writer (HID1)" device before starting to download) "(Note: The order of operation of USB download is different from that of serial port download.)**

Hardware USB Direct Download Reference Wiring Diagram

2.1.3 Pin Description

serial number				name (of a thing)	resemble type (e.g. blood type)	instructions
LQFP64	LQFP48	LQFP32	PDIP40			
1	1			P5.3	I/O	Standard IO Port
				TxD4_2	O	Transmit pin of serial port 4
				CAN2_TX_2	O	CAN2 bus transmitter pin
				LIN_TX_2	O	LIN bus transmitter pin
2	2		6	P0.5	I/O	Standard IO Port
				AD5	I	address bus (computing)
				ADC13	I	ADC Analogue Input Channel 13
				T3CLKO	O	Timer 3 clock divider output
3	3		7	P0.6	I/O	Standard IO Port
				AD6	I	address bus (computing)
				ADC14	I	ADC Analogue Input Channel 14
				T4	I	Timer 4 External clock input
				PWMFLT2_2	I	Enhanced PWM external exception detection pin
4	4		8	P0.7	I/O	Standard IO Port
				AD7	I	address bus (computing)
				T4CLKO	O	Timer 4-clock divided output
5				P6.0	I/O	Standard IO Port
				PWM1P_3	I/O	Capture Input and Pulse Output Positive for PWM1
6				P6.1	I/O	Standard IO Port
				PWM1N_3	I/O	Capture Input and Pulse Output Negative for PWM1
7				P6.2	I/O	Standard IO Port
				PWM2P_3	I/O	Capture Input and Pulse Output Positive for PWM2
8				P6.3	I/O	Standard IO Port
				PWM2N_3	I/O	Capture Input and Pulse Output Negative for PWM2
				P1.0	I/O	Standard IO Port
				ADC0	I	ADC Analogue input channel 0

9	5	1	9	PWM1P	I/O	Capture Input and Pulse Output Positive for PWM1
				RxD2	I	Receive pin of serial port 2
10	6	2	10	P1.1	I/O	Standard IO Port
				ADC1	I	ADC analogue input channel 1
				PWM1N	I/O	Capture Input and Pulse Output Negative for PWM1
				TxD2	I	Serial Port 2 Transmit Pin

serial number				name (of a thing)	typology	instructions
LQFP64	LQFP48	LQFP32	PDIP40			
11	7			P4.7	I/O	Standard IO Port
				TxD2_2	I	Serial Port 2 Transmit Pin
				CAN2_TX_3	O	CAN2 bus transmitter pin
				LIN_TX_3	O	LIN bus transmitter pin
12	8	3	11	P1.4	I/O	Standard IO Port
				ADC4	I	ADC analogue input channels 4
				PWM3P	I/O	Capture Input and Pulse Output Positive for PWM3
				MISO	I/O	SPI Host Input Slave Output
				S1MISO	I/O	USART1-SPI Host Input Slave Output
				S2MISO	I/O	USART2-SPI Host Input Slave Output
				SDA	I/O	I2C interface data lines
13	9	4	12	P1.5	I/O	Standard IO Port
				ADC5	I	ADC Analogue Input Channel 5
				PWM3N	I/O	Capture Input and Pulse Output Negative for PWM3
				SCLK	I/O	Clock pin for SPI
				S1SCLK	I/O	Clock pin of USART1-SPI
				S2SCLK	I/O	Clock pin of USART2-SPI
				SCL	I/O	I2C clock line
14	10	5	13	P1.6	I/O	Standard IO Port
				ADC6	I	ADC analogue input channels 6
				RxD_3	I	Receive pin of serial port 1
				PWM4P	I/O	Capture Input and Pulse Output Positive for PWM4
				MCLKO_2	O	Master Clock Split Output
				XTALO	O	Output pin of external crystal
15	11	6	14	P1.7	I/O	Standard IO Port
				ADC7	I	ADC Analogue Input Channel 7
				TxD_3	O	Transmit pin of serial port 1
				PWM4N	I/O	Capture Input and Pulse Output Negative for PWM4
				PWM5_2	I/O	Capture Input and Pulse Output for PWM5
				XTALI	I	Input pin for external crystal/external clock
16	12	7	15	P1.3	I/O	Standard IO Port
				ADC3	I	ADC analogue input channels 3
				MOSI	I/O	SPI Master Output Slave Input
				S1MOSI	I/O	USART1-SPI Host Output Slave Input
				S2MOSI	I/O	USART2-SPI Host Output Slave Input
				PWM2N	I/O	Capture Input and Pulse Output Negative for PWM2
				T2CLKO	O	Timer 2 clock divider output

17	13	8	16	UCAP	I	USB core power regulator pin

serial number				name (of a thing)	typology	instructions
LQFP64	LQFP48	LQFP32	PDIP40			
18	14	9	17	P5.4	I/O	Standard IO Port
				RST	I	Reset pin
				MCLKO	O	Master Clock Split Output
				SS_3	I	Slave select pin for SPI (host is output)
				SS	I	Slave select pin for SPI (host is output)
				S1SS_3	I	Slave select pin for USART1-SPI (host is output)
				S1SS	I	Slave select pin for USART1-SPI (host is output)
				S2SS_3	I	Slave select pin for USART2-SPI (host is output)
				S2SS	I	Slave select pin for USART2-SPI (host is output)
				PWM2P	I/O	Capture Input and Pulse Output Positive for PWM2
				PWM6_2	I/O	Capture Input and Pulse Output for PWM6
				T2	I	Timer 2 External clock input
				ADC2	I	ADC analogue input channels 2
19	15	10	18	Vcc	VCC	power supply pin
				AVcc	VCC	ADC Power Pin
20	16	11	19	Vref+	I	ADC reference voltage pin
21	17	12	20	Gnd	GND	earth (wire)
				Agnd	GND	ADC Ground
				Vref-	I	Reference Voltage Ground for ADC
22	18			P4.0	I/O	Standard IO Port
				MOSI_3	I/O	SPI Master Output Slave Input
				S1MOSI_3	I/O	USART1-SPI Host Output Slave Input
				S2MOSI_3	I/O	USART2-SPI Host Output Slave Input
23				P6.4	I/O	Standard IO Port
				PWM3P_3	I/O	Capture Input and Pulse Output Positive for PWM3
				S1SS_4	I	Slave select pin for USART1-SPI (host is output)
24				P6.5	I/O	Standard IO Port
				PWM3N_3	I/O	Capture Input and Pulse Output Negative for PWM3
				S1MOSI_4	I/O	USART1-SPI Host Output Slave Input
25				P6.6	I/O	Standard IO Port
				PWM4P_3	I/O	Capture Input and Pulse Output Positive for PWM4
				S1MISO_4	I/O	USART1-SPI Host Input Slave Output

26				P6.7	I/O	Standard IO Port
				PWM4N_3	I/O	Capture Input and Pulse Output Negative for PWM4
				S1SCLK_4	I/O	Clock pin of USART1-SPI
27	19	13	21	P3.0	I/O	Standard IO Port
				D-	I/O	USB Data Port
				RxD	I	Receive pin of serial port 1
				INT4	I	External interrupt 4

serial number				name (of a thing)	typology	instructions
LQFP64	LQFP48	LQFP32	PDIP40			
28	20	14	22	P3.1	I/O	Standard IO Port
				D+	I/O	USB Data Port
				TxD	O	Transmit pin of serial port 1
29	21	15	23	P3.2	I/O	Standard IO Port
				INT0	I	External Interrupt 0
				SCLK_4	I/O	Clock pin for SPI
				SCL_4	I/O	I2C clock line
				PWMETI	I	PWM External Trigger Input Pin
				PWMETI2	I	PWM external trigger input pin 2
30	22	16	24	P3.3	I/O	Standard IO Port
				INT1	I	External Interrupt 1
				MISO_4	I/O	SPI Host Input Slave Output
				SDA_4	I/O	I2C interface data lines
				PWM4N_4	I/O	Capture Input and Pulse Output Negative for PWM4
				PWM7_2	I/O	Capture Input and Pulse Output of PWM7
31	23	17	25	P3.4	I/O	Standard IO Port
				T0	I	Timer 0 External clock input
				T1CLKO	O	Timer 1 clock divider output
				MOSI_4	I/O	SPI Master Output Slave Input
				PWM4P_4	I/O	Capture Input and Pulse Output Positive for PWM4
				PWM8_2	I/O	Capture Input and Pulse Output for PWM8
				CMPO	O	Comparator Output
32	24			P5.0	I/O	Standard IO Port
				RxD3_2	I	Receive pin of serial port 3
				CMP+_2	I	Comparator positive input
				CAN_RX_2	I	CAN bus receive pin
33	25			P5.1	I/O	Standard IO Port
				TxD3_2	O	Serial port 3 transmitter pin
				CMP+_3	I	Comparator positive input
				CAN_TX_2	O	CAN bus transmitter pin
34	26	18	26	P3.5	I/O	Standard IO Port
				T1	I	Timer 1 External clock input
				T0CLKO	O	Timer 0 Clock divider output
				SS_4	I	Slave select pin for SPI (host is output)
				PWMFLT	I	Enhanced PWM external exception detection pin
35	27	19	27	P3.6	I/O	Standard IO Port
				INT2	I	External Interrupt 2
				RxD_2	I	Receive pin of serial port 1

				CMP-	I	Comparator negative input

serial number				name (of a thing)	typology	clarification
LQFP64	LQFP48	LQFP32	PDIP40			
36	28	20	28	P3.7	I/O	Standard IO Port
				INT3	I	External interrupt 3
				TxD_2	O	Transmit pin of serial port 1
				CMP+	I	Comparator positive input
37				P7.0	I/O	Standard IO Port
				CAN_RX_4	I	CAN bus receive pin
38				P7.1	I/O	Standard IO Port
				CAN_TX_4	O	CAN bus transmitter pin
39				P7.2	I/O	Standard IO Port
				CAN2_RX_4	I	CAN2 bus receive pin
				LIN_RX_4	I	LIN bus receive pin
40				P7.3	I/O	Standard IO Port
				CAN2_TX_4	O	CAN2 bus transmitter pin
				LIN_TX_4	O	LIN bus transmitter pin
				PWMETI_3	I	PWM External Trigger Input Pin
41	29		29	P4.1	I/O	Standard IO Port
				MISO_3	I/O	SPI Host Input Slave Output
				S1MISO_3	I/O	USART1-SPI Host Input Slave Output
				S2MISO_3	I/O	USART2-SPI Host Input Slave Output
				CMPO_2	O	Comparator Output
				PWMETI_3	I	PWM External Trigger Input Pin
42	30		30	P4.2	I/O	Standard IO Port
				WR	O	Write signal line for external bus
				CAN_RX_3	I	CAN bus receive pin
43	31			P4.3	I/O	Standard IO Port
				RxD_4	I	Receive pin of serial port 1
				SCLK_3	I/O	Clock pin for SPI
				S1SCLK_3	I/O	Clock pin of USART1-SPI
				S2SCLK_3	I/O	Clock pin of USART2-SPI
44	32		31	P4.4	I/O	Standard IO Port
				RD	O	Read signal line of the external bus
				TxD_4	O	Transmit pin of serial port 1
45	33	21	32	P2.0	I/O	Standard IO Port
				A8	I	address bus
				PWM1P_2	I/O	Capture Input and Pulse Output Positive for PWM1
				PWM5	I/O	Capture Input and Pulse Output for PWM5

serial number				name (of a thing)	typology	instructions
LQFP64	LQFP48	LQFP32	PDIP40			
46	34	22	33	P2.1	I/O	Standard IO Port
				A9	I	address bus (computing)
				PWM1N_2	I/O	Capture Input and Pulse Output Negative for PWM1
				PWM6	I/O	Capture Input and Pulse Output for PWM6
47	35	23	34	P2.2	I/O	Standard IO Port
				A10	I	address bus (computing)
				SS_2	I	Slave select pin for SPI (host is output)
				S1SS_2	I	Slave select pin for USART1-SPI (host is output)
				S2SS_2	I	Slave select pin for USART2-SPI (host is output)
				PWM2P_2	I/O	Capture Input and Pulse Output Positive for PWM2
				PWM7	I/O	Capture Input and Pulse Output of PWM7
48	36	24	356	P2.3	I/O	Standard IO Port
				A11	I	address bus (computing)
				MOSI_2	I/O	SPI Master Output Slave Input
				S1MOSI_2	I/O	USART1-SPI Host Output Slave Input
				S2MOSI_2	I/O	USART2-SPI Host Output Slave Input
				PWM2N_2	I/O	Capture Input and Pulse Output Negative for PWM2
				PWM8	I/O	Capture Input and Pulse Output for PWM8
49	37	25	36	P2.4	I/O	Standard IO Port
				A12	I	address bus (computing)
				MISO_2	I/O	SPI Host Input Slave Output
				S1MISO_2	I/O	USART1-SPI Host Input Slave Output
				S2MISO_2	I/O	USART2-SPI Host Input Slave Output
				SDA_2	I/O	I2C interface data lines
				PWM3P_2	I/O	Capture Input and Pulse Output Positive for PWM3
50	38	26	37	P2.5	I/O	Standard IO Port
				A13	I	address bus (computing)
				SCLK_2	I/O	Clock pin for SPI
				S1SCLK_2	I/O	Clock pin of USART1-SPI
				S2SCLK_2	I/O	Clock pin of USART2-SPI
				SCL_2	I/O	I2C clock line
				PWM3N_2	I/O	Capture Input and Pulse Output Negative for PWM3
51	39	27	38	P2.6	I/O	Standard IO Port
				A14	I	address bus (computing)
				PWM4P_2	I/O	Capture Input and Pulse Output Positive

						for PWM4

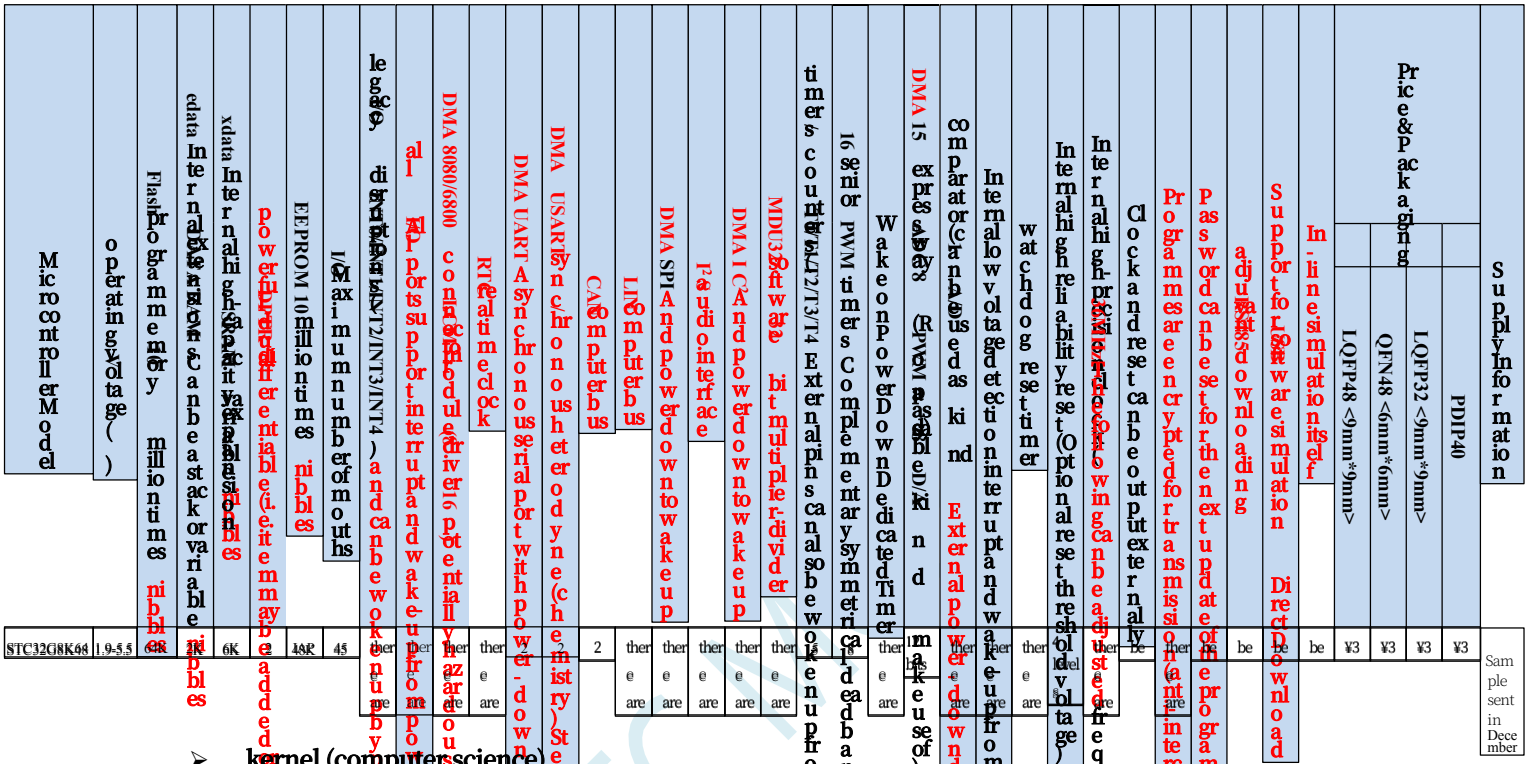
serial number				name (of a thing)	typology	clarification
LQFP64	LQFP48	LQFP32	PDIP40			
52	40	28	39	P2.7	I/O	Standard IO Port
				A15	I	address bus (computing)
				PWM4N_2	I/O	Capture Input and Pulse Output Negative for PWM4
53				P7.4	I/O	Standard IO Port
				PWM5_4	I/O	Capture Input and Pulse Output for PWM5
				S2SS_4	I	Slave select pin for USART2-SPI (host is output)
54				P7.5	I/O	Standard IO Port
				PWM6_4	I/O	Capture Input and Pulse Output for PWM6
				S2MOSI_4	I/O	USART2-SPI Slave Input Host Output
55				P7.6	I/O	Standard IO Port
				PWM7_4	I/O	Capture Input and Pulse Output of PWM7
				S2MISO_4	I/O	USART2-SPI Host Input Slave Output
56				P7.7	I/O	Standard IO Port
				PWM8_4	I/O	Capture Input and Pulse Output for PWM8
				S2SCLK_4	I/O	Clock pin of USART2-SPI
57	41		40	P4.5	I/O	Standard IO Port
				ALE	O	address latch signal
				CAN_TX_3	O	CAN bus transmitter pin
58	42			P4.6	I/O	Standard IO Port
				RxD2_2	I	Receive pin of serial port 2
				CAN2_RX_3	I	CAN2 bus receive pin
				LIN_RX_3	I	LIN bus receive pin
59	43	29	1	P0.0	I/O	Standard IO Port
				AD0	I	address bus (computing)
				ADC8	I	ADC analogue input channels 8
				RxD3	I	Receive pin of serial port 3
				PWM5_3	I/O	Capture Input and Pulse Output for PWM5
				CAN_RX	I	CAN bus receive pin

serial number				name (of a thing)	typology	instructions
LQFP64	LQFP48	LQFP32	PDIP40			
60	44	30	2	P0.1	I/O	Standard IO Port
				AD1	I	address bus (computing)
				ADC9	I	ADC analogue input channels 9
				TxD3	O	Serial port 3 transmitter pin
				PWM6_3	I/O	Capture Input and Pulse Output for PWM6
				CAN_TX	O	CAN bus transmitter pin
61	45	31	3	P0.2	I/O	Standard IO Port
				AD2	I	address bus (computing)
				ADC10	I	ADC Analogue input channels 10
				RxD4	I	Receive pin of serial port 4
				PWM7_3	I/O	Capture Input and Pulse Output of PWM7
				CAN2_RX	I	CAN2 bus receive pin
				LIN_RX	I	LIN bus receive pin
62	46	32	4	P0.3	I/O	Standard IO Port
				AD3	I	address bus (computing)
				ADC11	I	ADC Analogue Input Channel 11
				TxD4	O	Transmit pin of serial port 4
				PWM8_3	I/O	Capture Input and Pulse Output for PWM8
				CAN2_TX	O	CAN2 bus transmitter pin
				LIN_TX	O	LIN bus transmitter pin
63	47		5	P0.4	I/O	Standard IO Port
				AD4	I	address bus (computing)
				ADC12	I	ADC Analogue Input Channel 12
				T3	I	Timer 3 External clock input
64	48			P5.2	I/O	Standard IO Port
				RxD4_2	I	Receive pin of serial port 4
				CAN2_RX_2	I	CAN2 bus receive pin
				LIN_RX_2	I	LIN bus receive pin

2.2 stc32g8k64-lqfp48/lqfp32/pdip40

2.2.1 Characteristics and price

➢ **Option Price (no external crystal, no external reset, 12-bit ADC, 15 channels)**



➢ **kernel (computer science)**

✓ Ultra-fast 32-bit 8051 core (1T), approximately 70 times faster than traditional 8051

✓ 48 interrupt sources, 4 interrupt priority levels

✓ Supports online simulation

➢ **operating voltage**

✓ 1.9V to 5.5V (when the operating temperature is lower than -40°C, the operating voltage shall not be lower than 3.0V)

➢ **operating temperature**

✓ -40°C to 85°C (internal high-speed IRC (38MHz or less) and external crystal available)

✓ -40°C～125°C (When the temperature is higher than 85°C, please use an external high-temperature-resistant crystal)

➢ **Flash memory**

✓ Maximum 64K bytes of FLASH programme memory (ROM) for storing user code

✓ Supports user-configurable EEPROM size, 512-byte single page erase, up to 100,000 erasures or more

✓ Supports hardware USB direct downloads and regular serial port downloads.

✓ Support hardware SWD real-time emulation, P3.0/P3.1 (requires STC-USB Link1D tool)

➢ **SRAM, 6K bytes total**

✓ 2K bytes internal SRAM (edata)

✓ 6K bytes of internal extended RAM (internal xdata)

✓ Note on use: (it is strongly recommended not to declare variables using **idata** and **pdata**)

- Clock control

✓ Internal high-precision IRC (4MHz to 38MHz, adjustable up and down during ISP programming)
- ⌖ Error ± 0.3% (25°C at room temperature)
- ⌖ -1.35% to +1.30% temperature drift (full temperature range, -40°C to 85°C)
- ⌖ -0.76% to +0.98% temperature drift (temperature range, -20°C to 65°C)

✓ Internal 32KHz low-speed IRC (large error)

✓ External crystal (4MHz to 38MHz) and external clock

✓ Internal PLL output clock (Note: 96MHz/144MHz PLL output can be used independently as the clock source for high-speed PWM and high-speed SPI) Users are free to choose from the above four clock sources.

➢ **reset (a dislocated joint, an electronic device etc)**

✓ hardware reset
- ⌖ Power-on reset, the reset voltage value is 1.7V to 1.9V. (Valid when the chip does not enable the low-voltage reset function)
- ⌖ □□□□□ pin reset, factory P5.4 is I/O port by default, ISP download can set P5.4 pin as reset pin (Note: When setting P5.4)

  (The reset level is low when the pin is a reset pin)
- ⌖ Watchdog overflow reset
- ⌖ Low-voltage detection reset, provides 4 levels of low-voltage detection voltage: 2.0V, 2.4V, 2.7V, 3.0V.

✓ software reset
- ⌖ Write Reset Trigger Register in Software Mode

➢ **disruptions**

✓ Provides 48 interrupt sources: INT0, INT1, INT2, INT3, INT4, Timer0, Timer1, Timer2, Timer3, Timer4, USART1, USART2, UART3, UART4, ADC analogue-to-digital converter, LVD low-voltage detector, SPI, I² C, Comparator, PWMA, PWMB, CAN, CAN2 LIN, LCMIF colour screen interface interrupts, RTC real-time clock, all I/O interrupts (8 groups), DMA receive and send interrupts for serial port 1, DMA receive and send interrupts for serial port 2, DMA receive and send interrupts for serial port 3, DMA receive and send interrupts for serial port 4, DMA receive and send interrupts for I2C, DMA interrupts for SPI, DMA interrupts for ADC. DMA interrupt for I2C, DMA interrupt for SPI, DMA interrupt for ADC, DMA interrupt for LCD driver, and DMA interrupt for memory to memory.

✓ Provides 4 levels of interrupt priority

➢ **digital peripheral (computing)**

✓ 5 16-bit timers: Timer 0, Timer 1, Timer 2, Timer 3, Timer 4, with Mode 3 of Timer 0 with NMI (non-maskable interrupt) function, Timer 0 and Timer 1 mode 0 is 16-bit auto-reload mode

✓ 2 high-speed synchronous/asynchronous serial ports: Serial 1 (USART1), Serial 2 (USART2), baud rate clock source up to FOSC/4. Supports synchronous serial mode, asynchronous serial mode, SPI mode, LIN mode, infrared mode (IrDA), smart card mode (ISO7816).

✓ 2 high-speed asynchronous serial ports: Serial 3, Serial 4, baud rate clock source up to FOSC/4

✓ 2 advanced PWMs for 8-channel (4 complementary symmetrical) PWM with deadband control and external anomaly detection support

✓ SPI: Supports master and slave modes and automatic master/slave switching

✓ I² C: Supports host and slave modes

✓ ICE: Hardware Supported Emulation

✓ RTC: Supports year, month, day, hour, minute, second, sub-second (1/128th of a second), and supports clock interrupt and a set of alarms

✓ CAN: two independent CAN 2.0 control units

✓ LIN: One standalone LIN control unit (supports versions 1.3 and 2.1), USART1 and USART2 support two LIN

groups

- ✓ MDU32: Hardware 32-bit multiply-divider (contains 32-bit division by 32-bit, 32-bit multiplication by 32-bit)
- ✓ I/O port interrupt: all I/Os support interrupt, each group of I/O interrupt has independent interrupt entry address, and all I/O interrupts can support 4 interrupt modes: high level interrupt, low level interrupt, rising edge interrupt and falling edge interrupt.
- ✓ LCD Driver Module: Supports 8080 and 6800 interfaces and 8-bit and 16-bit data widths.
- ✓ DMA: Supports SPI shift receive data to memory, SPI shift send data to memory, I2C send data to memory, and I2C receive data.

Data to memory, memory to which serial port 1/2/3/4 receives data, data to be sent from memory by serial port 1/2/3/4, data to be automatically sampled to memory by ADC (while calculating the average value), data to be sent from memory by the LCD driver, and data copying from memory to memory.

✓ Hardware numeric ID: 32 bytes supported

➢ analogue peripherals

✓ ADC: Ultra-high-speed ADC, supporting 12-bit high-precision analogue-to-digital conversion of 15 channels (channels 0 to 14), channel 15 of the ADC is used to test the internal reference voltage (the internal reference voltage is adjusted to 1.19V when the chip is shipped from the factory, with an error of ±1%).

✓ Comparator: a set of comparators

➢ GPIO

✓ Up to 45 GPIOs: P0.0~P0.7, P1.0~P1.7, P2.0~P2.7, P3.0~P3.7, P4.0~P4.7, P5.0~P5.4

✓ All GPIOs support the following four modes: quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode, and high-resistance input mode.

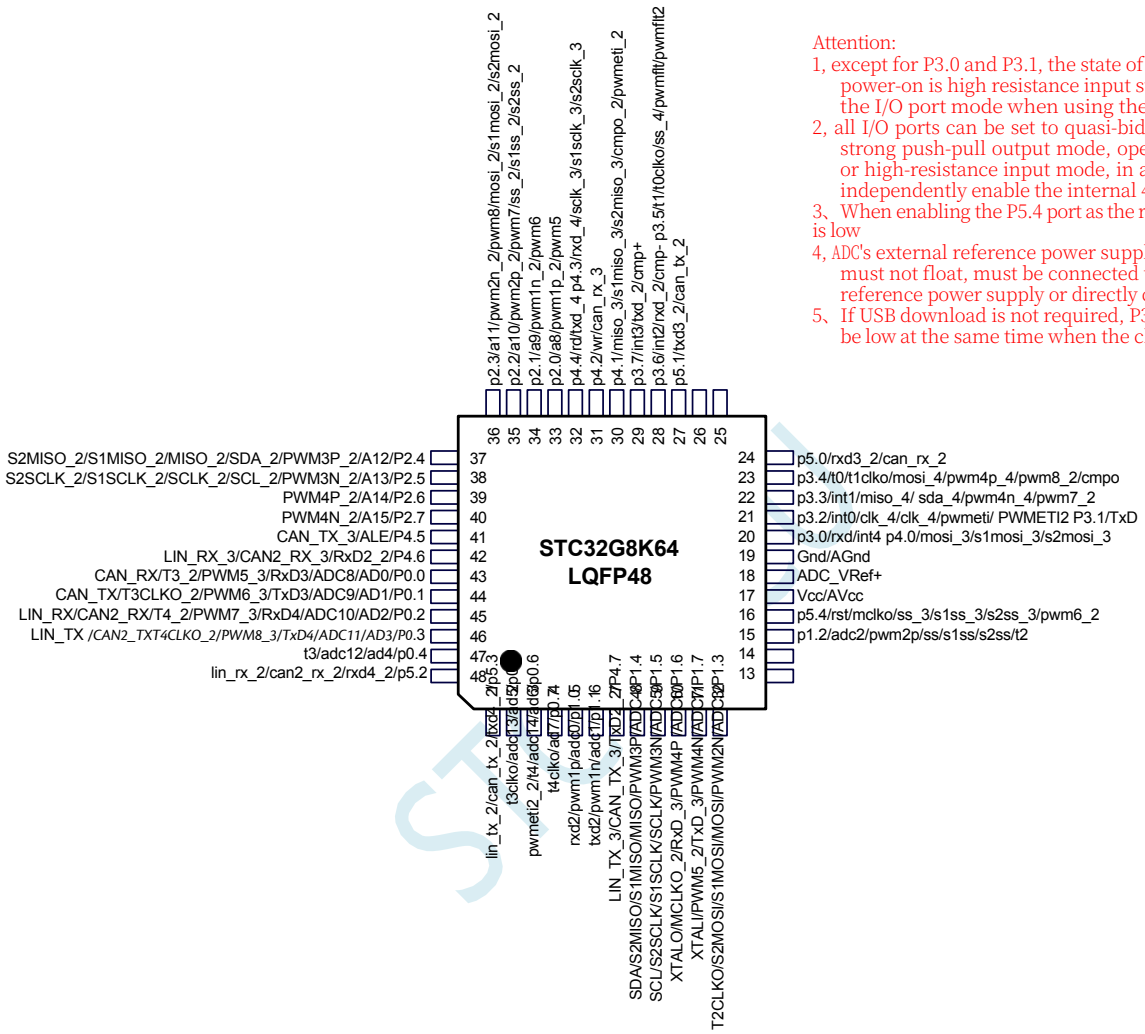
✓ Except for P3.0 and P3.1, all other IO ports are in high resistance input state after power on, users must set the IO port mode first when using IO ports.

✓ In addition, each I/O can independently enable the internal 4K pull-up resistor.

➢ **seal inside**

✓ LQFP48, LQFP32, PDIP40

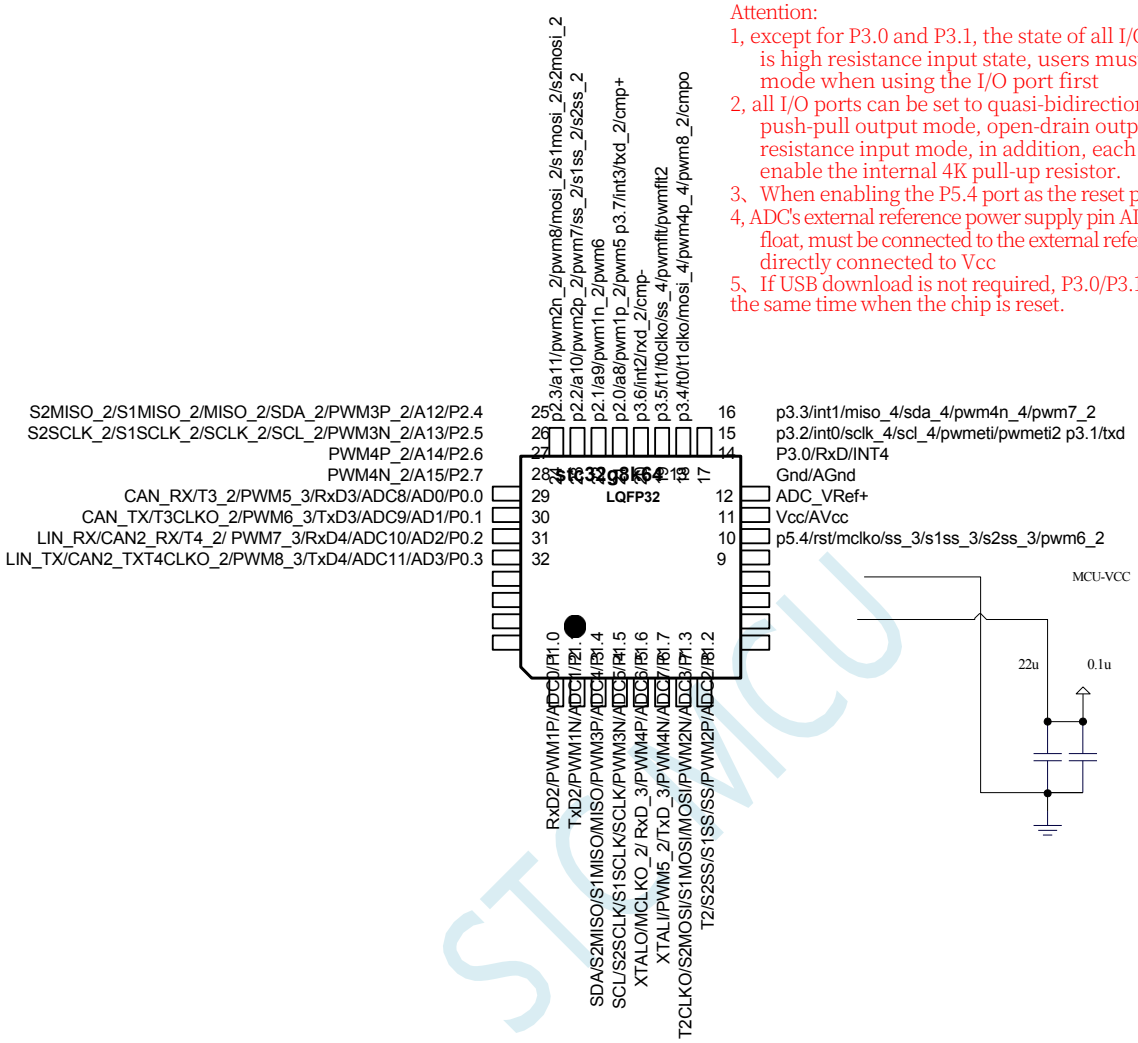
2.2.2 Pinout, Minimum System



Attention:
1, except for P3.0 and P3.1, the state of all I/O ports after power-on is high resistance input state, users must set the I/O port mode when using the I/O port first
2, all I/O ports can be set to quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode or high-resistance input mode, in addition, each I/O can independently enable the internal 4K pull-up resistor.
3、When enabling the P5.4 port as the reset pin, the reset level is low
4, ADC's external reference power supply pin ADC_VRef+, must not float, must be connected to the external reference power supply or directly connected to Vcc
5、If USB download is not required, P3.0/P3.1/P3.2 cannot be low at the same time when the chip is reset.

Looking at the chip silkscreen, the lower left dot is the first leg.
The last letter of the bottom line of the chip silkscreen is the chip version number.

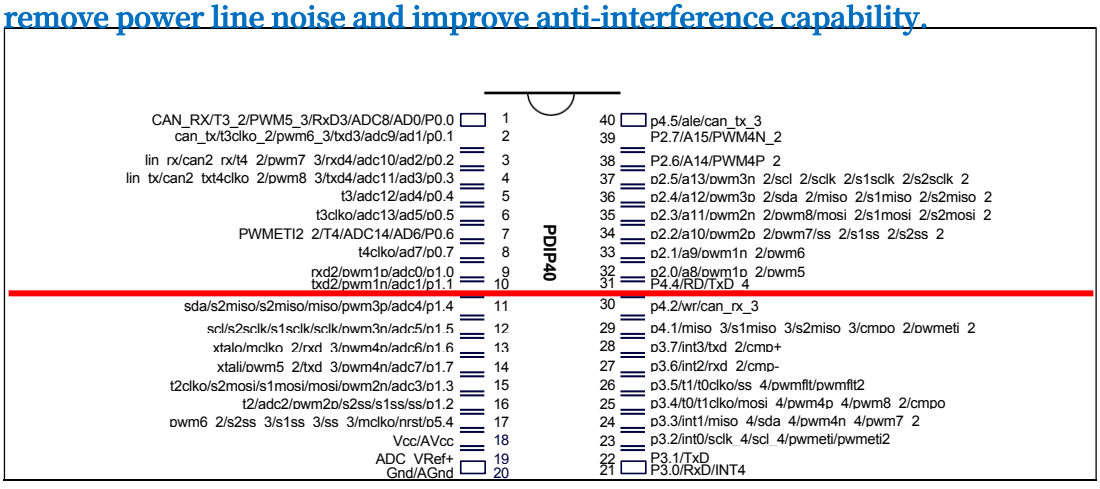
It is recommended to add power supply decoupling capacitors 22uF and 0.1uF between Vcc and Gnd to remove power line noise and improve anti-interference capability.



Looking at the chip silkscreen, the lower left dot is the first leg.

The last letter of the bottom line of the chip silkscreen is the chip version number.

It is recommended to add power supply decoupling capacitors 22uF and 0.1uF between Vcc and Gnd to remove power line noise and improve anti-interference capability.



2.2.3 Pin Description

serial number			name (of a thing)	typology	clarification
LQFP48	LQFP32	PDIP40			
1			P5.3	I/O	Standard IO Port
			TxD4_2	O	Transmit pin of serial port 4
			CAN2_TX_2	O	CAN2 bus transmitter pin
			LIN_TX_2	O	LIN bus transmitter pin
2		6	P0.5	I/O	Standard IO Port
			AD5	I	address bus
			ADC13	I	ADC Analogue Input Channel 13
			T3CLKO	O	Timer 3 clock divider output
3		7	P0.6	I/O	Standard IO Port
			AD6	I	address bus (computing)
			ADC14	I	ADC Analogue Input Channel 14
			T4	I	Timer 4 External clock input
			PWMFLT2_2	I	Enhanced PWM external exception detection pin
4		8	P0.7	I/O	Standard IO Port
			AD7	I	address bus
			T4CLKO	O	Timer 4-clock divided output
5	1	9	P1.0	I/O	Standard IO Port
			ADC0	I	ADC Analogue input channel 0
			PWM1P	I/O	Capture Input and Pulse Output Positive for PWM1
			RxD2	I	Receive pin of serial port 2
6	2	10	P1.1	I/O	Standard IO Port
			ADC1	I	ADC analogue input channel 1
			PWM1N	I/O	Capture Input and Pulse Output Negative for PWM1
			TxD2	I	Serial Port 2 Transmit Pin
7			P4.7	I/O	Standard IO Port
			TxD2_2	I	Serial Port 2 Transmit Pin
			CAN2_TX_3	O	CAN2 bus transmitter pin
			LIN_TX_3	O	LIN bus transmitter pin
8	3	11	P1.4	I/O	Standard IO Port
			ADC4	I	ADC analogue input channels 4
			PWM3P	I/O	Capture Input and Pulse Output Positive for PWM3
			MISO	I/O	SPI Host Input Slave Output
			S1MISO	I/O	USART1-SPI Host Input Slave Output
			S2MISO	I/O	USART2-SPI Host Input Slave Output
			SDA	I/O	I2C interface data lines

serial number			name (of a thing)	typology	clarification
LQFP48	LQFP32	PDIP40			
9	4	12	P1.5	I/O	Standard IO Port
			ADC5	I	ADC Analogue Input Channel 5
			PWM3N	I/O	Capture Input and Pulse Output Negative for PWM3
			SCLK	I/O	Clock pin for SPI
			S1SCLK	I/O	Clock pin of USART1-SPI
			S2SCLK	I/O	Clock pin of USART2-SPI
			SCL	I/O	I2C clock line
10	5	13	P1.6	I/O	Standard IO Port
			ADC6	I	ADC analogue input channels 6
			RxD_3	I	Receive pin of serial port 1
			PWM4P	I/O	Capture Input and Pulse Output Positive for PWM4
			MCLKO_2	O	Master Clock Split Output
			XTALO	O	Output pin of external crystal
11	6	14	P1.7	I/O	Standard IO Port
			ADC7	I	ADC Analogue Input Channel 7
			TxD_3	O	Transmit pin of serial port 1
			PWM4N	I/O	Capture Input and Pulse Output Negative for PWM4
			PWM5_2	I/O	Capture Input and Pulse Output for PWM5
			XTALI	I	Input pin for external crystal/external clock
12	7	15	P1.3	I/O	Standard IO Port
			ADC3	I	ADC analogue input channels 3
			MOSI	I/O	SPI Master Output Slave Input
			S1MOSI	I/O	USART1-SPI Host Output Slave Input
			S2MOSI	I/O	USART2-SPI Host Output Slave Input
			PWM2N	I/O	Capture Input and Pulse Output Negative for PWM2
			T2CLKO	O	Timer 2 clock divider output
13	8	16	P1.2	I/O	Standard IO Port
			ADC2	I	ADC analogue input channels 2
			PWM2P	I/O	Capture Input and Pulse Output Positive for PWM2
			SS	I	Slave select pin for SPI (host is output)
			S1SS	I	Slave select pin for USART1-SPI (host is output)
			S2SS	I	Slave select pin for USART2-SPI (host is output)
			T2	I	Timer 2 External clock input

serial number			name (of a thing)	typology	instructions
LQFP48	LQFP32	PDIP40			
14	9	17	P5.4	I/O	Standard IO Port
			RST	I	Reset pin
			MCLKO	O	Master Clock Split Output
			SS_3	I	Slave select pin for SPI (host is output)
			S1SS_3	I	Slave select pin for USART1-SPI (host is output)
			S2SS_3	I	Slave select pin for USART2-SPI (host is output)
			PWM6_2	I/O	Capture Input and Pulse Output for PWM6
15	10	18	Vcc	VCC	power supply pin
			AVcc	VCC	ADC Power Pin
16	11	19	Vref+	I	ADC reference voltage pin
17	12	20	Gnd	GND	earth (wire)
			Agnd	GND	ADC Ground
			Vref-	I	Reference Voltage Ground for ADC
18			P4.0	I/O	Standard IO Port
			MOSI_3	I/O	SPI Master Output Slave Input
			S1MOSI_3	I/O	USART1-SPI Host Output Slave Input
			S2MOSI_3	I/O	USART2-SPI Host Output Slave Input
19	13	21	P3.0	I/O	Standard IO Port
			RxD	I	Receive pin of serial port 1
			INT4	I	External interrupt 4
20	14	22	P3.1	I/O	Standard IO Port
			TxD	O	Transmit pin of serial port 1
21	15	23	P3.2	I/O	Standard IO Port
			INT0	I	External Interrupt 0
			SCLK_4	I/O	Clock pin for SPI
			SCL_4	I/O	I2C clock line
			PWMETI	I	PWM External Trigger Input Pin
			PWMETI2	I	PWM external trigger input pin 2

serial number			name (of a thing)	typology	clarification
LQFP48	LQFP32	PDIP40			
22	16	24	P3.3	I/O	Standard IO Port
			INT1	I	External Interrupt 1
			MISO_4	I/O	SPI Host Input Slave Output
			SDA_4	I/O	I2C interface data lines
			PWM4N_4	I/O	Capture Input and Pulse Output Negative for PWM4
			PWM7_2	I/O	Capture Input and Pulse Output of PWM7
23	17	25	P3.4	I/O	Standard IO Port
			T0	I	Timer 0 External clock input
			T1CLKO	O	Timer 1 clock divider output
			MOSI_4	I/O	SPI Master Output Slave Input
			PWM4P_4	I/O	Capture Input and Pulse Output Positive for PWM4
			PWM8_2	I/O	Capture Input and Pulse Output for PWM8
			CMPO	O	Comparator Output
24			P5.0	I/O	Standard IO Port
			RxD3_2	I	Receive pin of serial port 3
			CMP+_2	I	Comparator positive input
			CAN_RX_2	I	CAN bus receive pin
25			P5.1	I/O	Standard IO Port
			TxD3_2	O	Serial port 3 transmitter pin
			CMP+_3	I	Comparator positive input
			CAN_TX_2	O	CAN bus transmitter pin
26	18	26	P3.5	I/O	Standard IO Port
			T1	I	Timer 1 External clock input
			T0CLKO	O	Timer 0 Clock divider output
			SS_4	I	Slave select pin for SPI (host is output)
			PWMFLT	I	Enhanced PWM external exception detection pin
27	19	27	P3.6	I/O	Standard IO Port
			INT2	I	External Interrupt 2
			RxD_2	I	Receive pin of serial port 1
			CMP-	I	Comparator negative input

serial number			name (of a thing)	typology	clarification
LQFP48	LQFP32	PDIP40			
28	20	28	P3.7	I/O	Standard IO Port
			INT3	I	External interrupt 3
			TxD_2	O	Transmit pin of serial port 1
			CMP+	I	Comparator positive input
29		29	P4.1	I/O	Standard IO Port
			MISO_3	I/O	SPI Host Input Slave Output
			S1MISO_3	I/O	USART1-SPI Host Input Slave Output
			S2MISO_3	I/O	USART2-SPI Host Input Slave Output
			CMPO_2	O	Comparator Output
			PWMETI_3	I	PWM External Trigger Input Pin
30		30	P4.2	I/O	Standard IO Port
			WR	O	Write signal line for external bus
			CAN_RX_3	I	CAN bus receive pin
31			P4.3	I/O	Standard IO Port
			RxD_4	I	Receive pin of serial port 1
			SCLK_3	I/O	Clock pin for SPI
			S1SCLK_3	I/O	Clock pin of USART1-SPI
			S2SCLK_3	I/O	Clock pin of USART2-SPI
32		31	P4.4	I/O	Standard IO Port
			RD	O	Read signal line of the external bus
			TxD_4	O	Transmit pin of serial port 1
33	21	32	P2.0	I/O	Standard IO Port
			A8	I	address bus (computing)
			PWM1P_2	I/O	Capture Input and Pulse Output Positive of PWM1
			PWM5	I/O	Capture Input and Pulse Output for PWM5
34	22	33	P2.1	I/O	Standard IO Port
			A9	I	address bus (computing)
			PWM1N_2	I/O	Capture Input and Pulse Output Negative for PWM1
			PWM6	I/O	Capture Input and Pulse Output for PWM6
35	23	34	P2.2	I/O	Standard IO Port
			A10	I	address bus (computing)
			SS_2	I	Slave select pin for SPI (host is output)
			S1SS_2	I	Slave select pin for USART1-SPI (host is output)
			S2SS_2	I	Slave select pin for USART2-SPI (host is output)
			PWM2P_2	I/O	Capture Input and Pulse Output Positive for PWM2
			PWM7	I/O	Capture Input and Pulse Output of PWM7

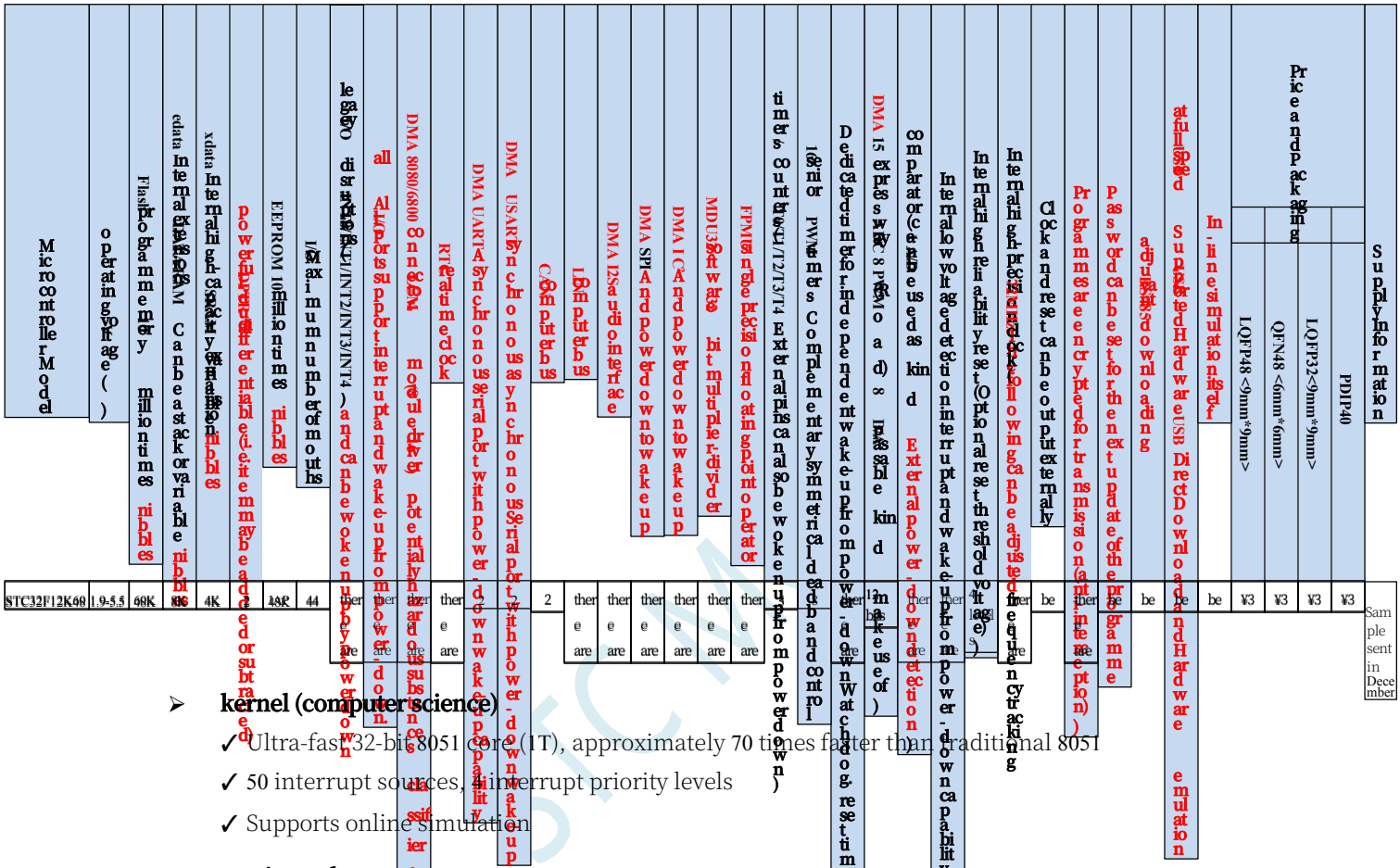
serial number			name (of a thing)	typology	clarification
LQFP48	LQFP32	PDIP40			
36	24	35	P2.3	I/O	Standard IO Port
			A11	I	address bus (computing)
			MOSI_2	I/O	SPI Master Output Slave Input
			S1MOSI_2	I/O	USART1-SPI Host Output Slave Input
			S2MOSI_2	I/O	USART2-SPI Host Output Slave Input
			PWM2N_2	I/O	Capture Input and Pulse Output Negative for PWM2
			PWM8	I/O	Capture Input and Pulse Output for PWM8
37	25	36	P2.4	I/O	Standard IO Port
			A12	I	address bus (computing)
			MISO_2	I/O	SPI Host Input Slave Output
			S1MISO_2	I/O	USART1-SPI Host Input Slave Output
			S2MISO_2	I/O	USART2-SPI Host Input Slave Output
			SDA_2	I/O	I2C interface data lines
			PWM3P_2	I/O	Capture Input and Pulse Output Positive for PWM3
38	26	37	P2.5	I/O	Standard IO Port
			A13	I	address bus (computing)
			SCLK_2	I/O	Clock pin for SPI
			S1SCLK_2	I/O	Clock pin of USART1-SPI
			S2SCLK_2	I/O	Clock pin of USART2-SPI
			SCL_2	I/O	I2C clock line
			PWM3N_2	I/O	Capture Input and Pulse Output Negative for PWM3
39	27	38	P2.6	I/O	Standard IO Port
			A14	I	address bus (computing)
			PWM4P_2	I/O	Capture Input and Pulse Output Positive for PWM4
40	28	39	P2.7	I/O	Standard IO Port
			A15	I	address bus
			PWM4N_2	I/O	Capture Input and Pulse Output Negative for PWM4
41		40	P4.5	I/O	Standard IO Port
			ALE	O	address latch signal
			CAN_TX_3	O	CAN bus transmitter pin

serial number			name (of a thing)	typology	clarification
LQFP48	LQFP32	PDIP40			
42			P4.6	I/O	Standard IO Port
			RxD2_2	I	Receive pin of serial port 2
			CAN2_RX_3	I	CAN2 bus receive pin
			LIN_RX_3	I	LIN bus receive pin
43	29	1	P0.0	I/O	Standard IO Port
			AD0	I	address bus
			ADC8	I	ADC analogue input channels 8
			RxD3	I	Receive pin of serial port 3
			PWM5_3	I/O	Capture Input and Pulse Output for PWM5
			CAN_RX	I	CAN bus receive pin
44	30	2	P0.1	I/O	Standard IO Port
			AD1	I	address bus (computing)
			ADC9	I	ADC analogue input channels 9
			TxD3	O	Serial port 3 transmitter pin
			PWM6_3	I/O	Capture Input and Pulse Output for PWM6
			CAN_TX	O	CAN bus transmitter pin
45	31	3	P0.2	I/O	Standard IO Port
			AD2	I	address bus (computing)
			ADC10	I	ADC Analogue Input Channel 10
			RxD4	I	Receive pin of serial port 4
			PWM7_3	I/O	Capture Input and Pulse Output of PWM7
			CAN2_RX	I	CAN2 bus receive pin
			LIN_RX	I	LIN bus receive pin
46	32	4	P0.3	I/O	Standard IO Port
			AD3	I	address bus (computing)
			ADC11	I	ADC Analogue Input Channel 11
			TxD4	O	Transmit pin of serial port 4
			PWM8_3	I/O	Capture Input and Pulse Output for PWM8
			CAN2_TX	O	CAN2 bus transmitter pin
			LIN_TX	O	LIN bus transmitter pin
47		5	P0.4	I/O	Standard IO Port
			AD4	I	address bus (computing)
			ADC12	I	ADC Analogue input channels 12
			T3	I	Timer 3 External clock input
48			P5.2	I/O	Standard IO Port
			RxD4_2	I	Receive pin of serial port 4
			CAN2_RX_2	I	CAN2 bus receive pin
			LIN_RX_2	I	LIN bus receive pin

2.3 stc32f12k60-lqfp48/lqfp32/pdip40

2.3.1 Characteristics and price

➢ **Option Price (no external crystal, no external reset, 12-bit ADC, 15 channels)**



➢ **kernel (computer science)**

✓ Ultra-fast 32-bit 8051 core (1T), approximately 70 times faster than traditional 8051

✓ 50 interrupt sources, 4 interrupt priority levels

✓ Supports online simulation

➢ **operating voltage**

✓ 1.9V to 5.5V (when the operating temperature is lower than -40℃, the operating voltage shall not be lower than 3.0V)

➢ **operating temperature**

✓ -40°C to 85°C (internal high-speed IRC (64MHz or less) and external crystal available)

✓ -40°C～125°C (When the temperature is higher than 85°C, please use an external high-temperature-resistant crystal)

➢ **Flash memory**

✓ Maximum 60K bytes of FLASH programme memory (ROM) for storing user code

✓ Supports user-configurable EEPROM size, 512-byte single page erase, up to 100,000 erasures or more

✓ Supports hardware USB direct downloads and regular serial port downloads.

✓ Support hardware SWD real-time emulation, P3.0/P3.1 (requires STC-USB Link1D tool)

➢ **SRAM, 12K bytes total**

✓ 8K bytes internal SRAM (edata)

✓ 4K bytes of internal extended RAM (internal xdata)

✓ Note on use: (it is strongly recommended not to declare variables using **idata** and **pdata**)

➢ Clock control

✓ Internal high-precision IRC (4MHz to 64MHz, adjustable up and down during ISP programming)
- ⌖ Error ± 0.3% (25°C at room temperature)
- ⌖ -1.35% to +1.30% temperature drift (full temperature range, -40°C to 85°C)
- ⌖ -0.76% to +0.98% temperature drift (temperature range, -20°C to 65°C)

✓ Internal 32KHz low-speed IRC (large error)
✓ External crystal (4MHz to 64MHz) and external clock
✓ Internal PLL output clock (Note: 96MHz/144MHz PLL output can be used independently as the clock source for high-speed PWM and high-speed SPI) Users are free to choose from the above four clock sources.

➢ **reset (a dislocated joint, an electronic device etc)**

✓ hardware reset
- ⌖ Power-on reset, the reset voltage value is 1.7V to 1.9V. (Valid when the chip does not enable the low-voltage reset function)
- ⌖ □□□□□ pin reset, factory P5.4 is I/O port by default, ISP download can set P5.4 pin as reset pin (Note: When setting P5.4)

  (The reset level is low when the pin is a reset pin)
- ⌖ Watchdog overflow reset
- ⌖ Low-voltage detection reset, provides 4 levels of low-voltage detection voltage: 2.0V, 2.4V, 2.7V, 3.0V.

✓ software reset
- ⌖ Write Reset Trigger Register in Software Mode

➢ **disruptions**

✓ Provides 50 interrupt sources: INT0, INT1, INT2, INT3, INT4, Timer0, Timer1, Timer2, Timer3, Timer4, USART1, USART2, UART3, UART4, ADC analogue-to-digital converter, LVD low-voltage detector, SPI, I² C, comparator, PWMA, PWMB, USB, CAN, CAN2, LIN, LCMIF colour screen interface interrupt, RTC real-time clock, all I/O interrupts (6 groups), I2S audio interface, DMA receive and send interrupt for I2S audio interface, DMA receive and send interrupt for serial port 1, DMA receive and send interrupt for serial port 2, DMA receive and send interrupt for serial port 3, DMA receive and send interrupt for serial port 4, DMA receive and transmit interrupt for I2C, DMA interrupt for SPI, DMA interrupt for ADC, DMA interrupt for LCD driver, and memory-to-memory DMA interrupt.

✓ Provides 4 levels of interrupt priority

➢ **digital peripheral (computing)**

✓ 5 16-bit timers: Timer 0, Timer 1, Timer 2, Timer 3, Timer 4, with Mode 3 of Timer 0 with NMI (non-maskable interrupt) function, Timer 0 and Timer 1 mode 0 is 16-bit auto-reload mode
✓ 2 high-speed synchronous/asynchronous serial ports: Serial 1 (USART1), Serial 2 (USART2), baud rate clock source up to FOSC/4. Supports synchronous serial mode, asynchronous serial mode, SPI mode, LIN mode, infrared mode (IrDA), smart card mode (ISO7816).
✓ 2 high-speed asynchronous serial ports: Serial 3, Serial 4, baud rate clock source up to FOSC/4
✓ 2 advanced PWMs for 8-channel (4 complementary symmetrical) PWM with deadband control and external anomaly detection support
✓ SPI: Supports master and slave modes and automatic master/slave switching
✓ I² C: Supports host mode and slave mode
✓ ICE: Hardware Supported Emulation
✓ RTC: Supports year, month, day, hour, minute, second, sub-second (1/128th of a second), and supports clock interrupt and a set of alarms
✓ USB: USB2.0/USB1.1 compliant full-speed USB, 6 bi-directional endpoints, supports 4 endpoint

transfer modes (control transfer, interrupt transfer, batch transfer, and synchronous transfer), each endpoint has a 64-byte buffer

- ✓ I2S: Audio Interface
- ✓ CAN: two independent CAN 2.0 control units
- ✓ LIN: One standalone LIN control unit (supports versions 1.3 and 2.1), USART1 and USART2 support two LIN groups
- ✓ MDU32: Hardware 32-bit multiply-divider (contains 32-bit division by 32-bit, 32-bit multiplication by 32-bit)
- ✓ I/O port interrupts: all I/Os support interrupts, each group of I/O interrupts has an independent interrupt entry address, and all I/O interrupts can support 4 kinds of interrupts.

Mode: High-level interrupt, low-level interrupt, rising-edge interrupt, falling-edge interrupt.I/O port interrupt can be wake-up from power-down and has 4 levels of interrupt priority.

- ✓ LCD Driver Module: Supports 8080 and 6800 interfaces and 8-bit and 16-bit data widths.
- ✓ DMA: Supports SPI shifted receive data to memory, SPI shifted send data to memory, I2C send data to memory, I2C receive data to memory, Serial port 1/2/3/4 receive data to memory, Serial port 1/2/3/4 send data to memory, ADC auto-sampling data to memory (while calculating the average value), LCD driver send data to memory, and memory-to-memory data copying. data to memory, LCD driver sending data to memory, and memory-to-memory data copying
- ✓ Hardware numeric ID: 32 bytes supported

➢ analogue peripherals

- ✓ ADC: Ultra-high-speed ADC, supporting 12-bit high-precision analogue-to-digital conversion of 15 channels (channels 0 to 14), channel 15 of the ADC is used to test the internal reference voltage (the internal reference voltage is adjusted to 1.19V when the chip is shipped from the factory, with an error of ±1%).
- ✓ Comparator: a set of comparators

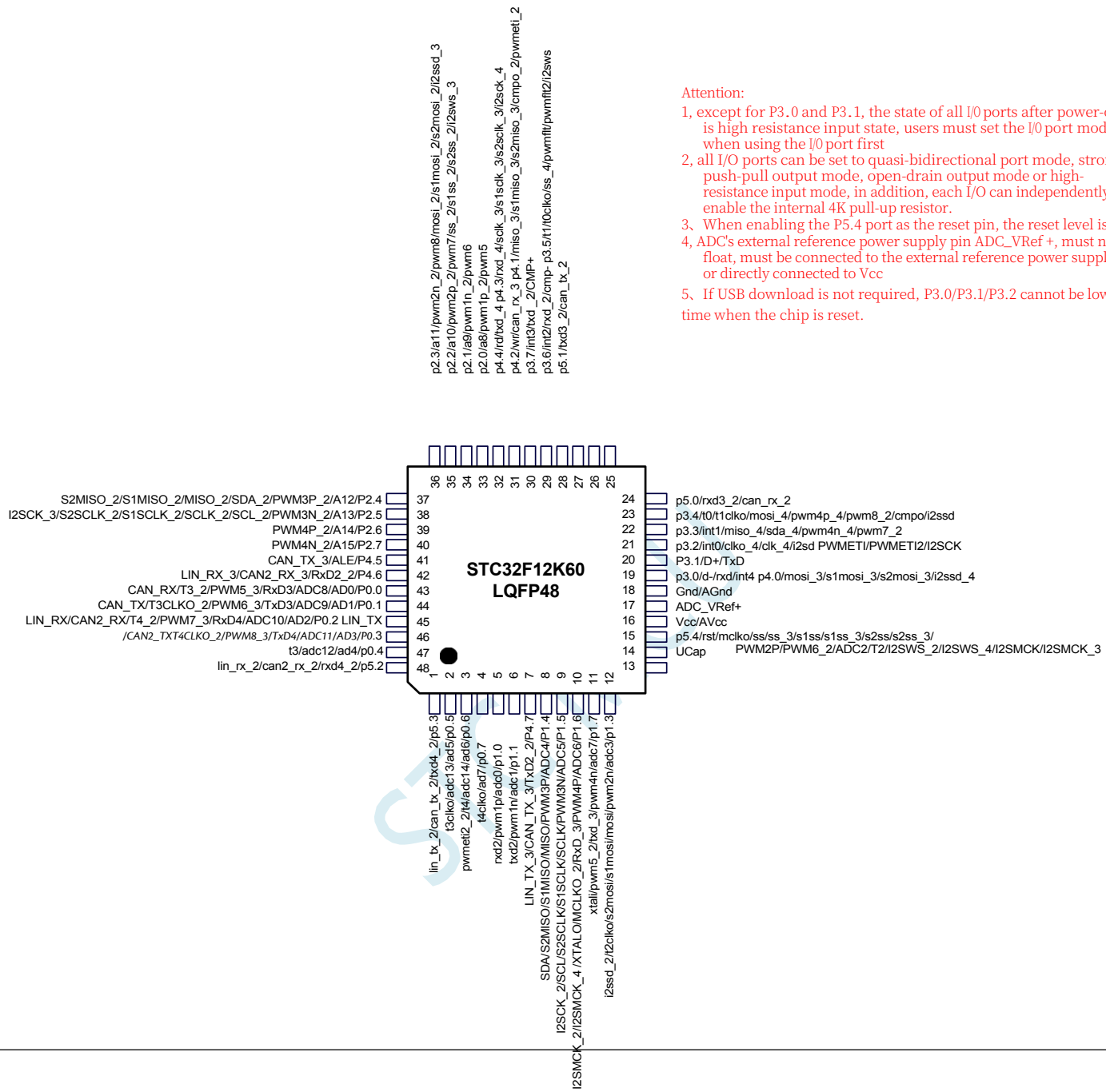
➢ GPIO

- ✓ Up to 44 GPIOs: P0.0~P0.7, P1.0~P1.7 (no P1.2), P2.0~P2.7, P3.0~P3.7, P4.0~P4.7, P5.0~P5.4
- ✓ All GPIOs support the following four modes: quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode, and high-resistance input mode.
- ✓ Except for P3.0 and P3.1, all other IO ports are in high resistance input state after power on, users must set the IO port mode first when using IO ports.
- ✓ In addition, each I/O can independently enable the internal 4K pull-up resistor.

➢ **seal inside**

- ✓ LQFP48, LQFP32, PDIP40

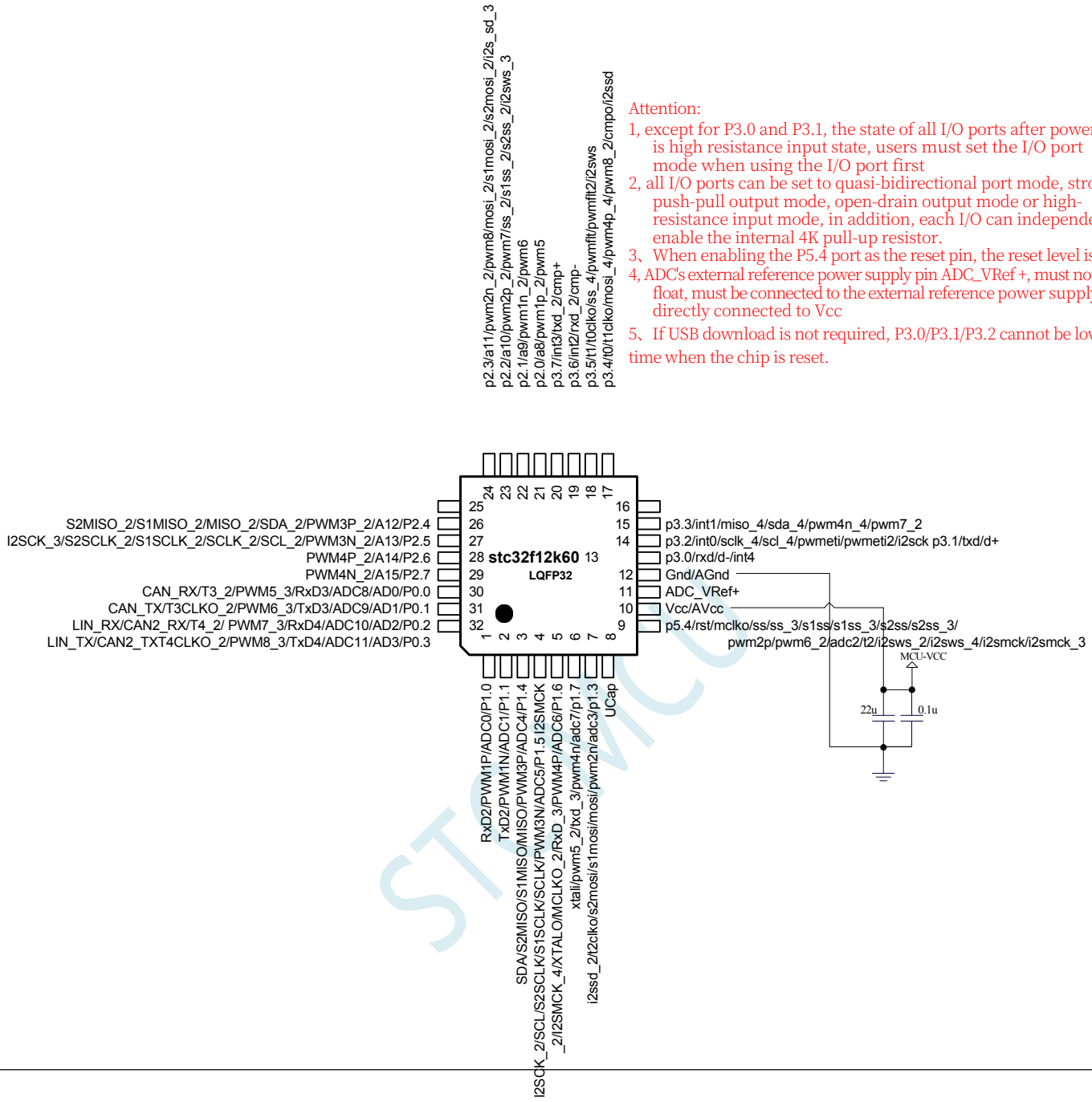
2.3.2 Pinout, Minimum System



Looking at the chip silkscreen, the lower left dot is the first leg.

The last letter of the bottom line of the chip silkscreen is the chip version number.

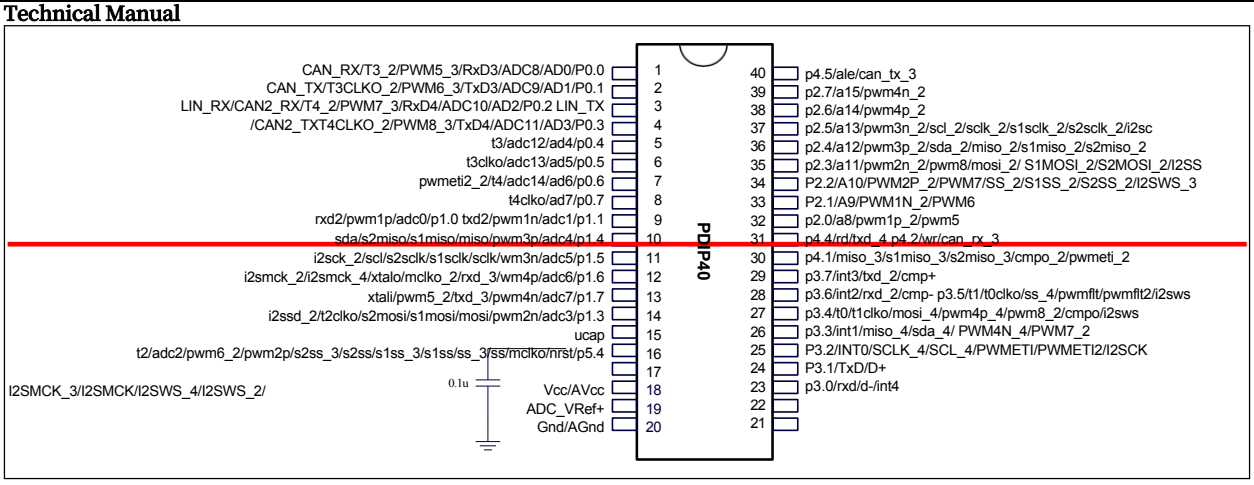
It is recommended to add power supply decoupling capacitors 22uF and 0.1uF between Vcc and Gnd to remove power line noise and improve anti-interference capability.



Looking at the chip silkscreen, the lower left dot is the first leg.

The last letter of the bottom line of the chip silkscreen is the chip version number.

It is recommended to add power supply decoupling capacitors 22uF and 0.1uF between Vcc and Gnd to remove power line noise and improve anti-interference capability.


CAN_RX/T3_2/PWM5_3/RxD3/ADC8/AD0/P0.0
CAN_TX/T3CLKO_2/PWM6_3/TxD3/ADC9/AD1/P0.1
LIN_RX/CAN2_RX/T4_2/PWM7_3/RxD4/ADC10/AD2/P0.2 LIN_TX
/CAN2_TXT4CLKO_2/PWM8_3/TxD4/ADC11/AD3/P0.3
t3/adc12/ad4/p0.4
t3clko/adc13/ad5/p0.5
pwmeti2_2/t4/adc14/ad6/p0.6
t4clko/ad7/p0.7
rxd2/pwm1p/adc0/p1.0 txd2/pwm1n/adc1/p1.1
sda/s2miso/s1miso/miso/pwm3p/adc4/p1.4
i2sck_2/scl/s2sclk/s1sclk/sclk/wm3n/adc5/p1.5
i2smck_2/i2smck_4/xtalo/mclko_2/rxd_3/wm4p/adc6/p1.6
xtali/pwm5_2/txd_3/pwm4n/adc7/p1.7
i2ssd_2/t2clko/s2mosi/s1mosi/mosi/pwm2n/adc3/p1.3
ucap
t2/adc2/pwm6_2/pwm2p/s2ss_3/s2ss/s1ss_3/s1ss/ss_3/ss/mclko/nrst/p5.4
I2SMCK_3/I2SMCK/I2SWS_4/I2SWS_2/
0.1u
Vcc/AVcc
ADC_VRef+
Gnd/AGnd
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
PDIP40
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
p4.5/ale/can_tx_3
p2.7/a15/pwm4n_2
p2.6/a14/pwm4p_2
p2.5/a13/pwm3n_2/scl_2/sclk_2/s1sclk_2/s2sclk_2/i2sc
p2.4/a12/pwm3p_2/sda_2/miso_2/s1miso_2/s2miso_2
p2.3/a11/pwm2n_2/pwm8/mosi_2/ S1MOSI_2/S2MOSI_2/I2SS
P2.2/A10/PWM2P_2/PWM7/SS_2/S1SS_2/S2SS_2/I2SWS_3
P2.1/A9/PWM1N_2/PWM6
p2.0/a8/pwm1p_2/pwm5
p4.4/rd/txd_4 p4.2/wr/can_rx_3
p4.1/miso_3/s1miso_3/s2miso_3/cmpo_2/pwmeti_2
p3.7/int3/txd_2/cmp+
p3.6/int2/rxd_2/cmp- p3.5/t1/t0clko/ss_4/pwmflt/pwmflt2/i2sws
p3.4/t0/t1clko/mosi_4/pwm4p_4/pwm8_2/cmpo/i2sws
p3.3/int1/miso_4/sda_4/ PWM4N_4/PWM7_2
P3.2/INT0/SCLK_4/SCL_4/PWMETI/PWMETI2/I2SCK
P3.1/TxD/D+
p3.0/rxd/d-/int4

2.3.3 Pin Description

serial number			name (of a thing)	typology	clarification
LQFP48	LQFP32	PDIP40			
1			P5.3	I/O	Standard IO Port
			TxD4_2	O	Transmit pin of serial port 4
			CAN2_TX_2	O	CAN2 bus transmitter pin
			LIN_TX_2	O	LIN bus transmitter pin
2		6	P0.5	I/O	Standard IO Port
			AD5	I	address bus
			ADC13	I	ADC Analogue Input Channel 13
			T3CLKO	O	Timer 3 clock divider output
3		7	P0.6	I/O	Standard IO Port
			AD6	I	address bus (computing)
			ADC14	I	ADC Analogue Input Channel 14
			T4	I	Timer 4 External clock input
			PWMFLT2_2	I	Enhanced PWM external exception detection pin
4		8	P0.7	I/O	Standard IO Port
			AD7	I	address bus (computing)
			T4CLKO	O	Timer 4-clock divided output
5	1	9	P1.0	I/O	Standard IO Port
			ADC0	I	ADC Analogue input channel 0
			PWM1P	I/O	Capture Input and Pulse Output Positive for PWM1
			RxD2	I	Receive pin of serial port 2
6	2	10	P1.1	I/O	Standard IO Port
			ADC1	I	ADC analogue input channel 1
			PWM1N	I/O	Capture Input and Pulse Output Negative for PWM1
			TxD2	I	Serial Port 2 Transmit Pin
7			P4.7	I/O	Standard IO Port
			TxD2_2	I	Serial Port 2 Transmit Pin
			CAN2_TX_3	O	CAN2 bus transmitter pin
			LIN_TX_3	O	LIN bus transmitter pin
8	3	11	P1.4	I/O	Standard IO Port
			ADC4	I	ADC analogue input channels 4
			PWM3P	I/O	Capture Input and Pulse Output Positive for PWM3
			MISO	I/O	SPI Host Input Slave Output
			S1MISO	I/O	USART1-SPI Host Input Slave Output
			S2MISO	I/O	USART2-SPI Host Input Slave Output
			SDA	I/O	I2C interface data lines

serial number			name (of a thing)	typology	instructions
LQFP48	LQFP32	PDIP40			
9	4	12	P1.5	I/O	Standard IO Port
			ADC5	I	ADC Analogue Input Channel 5
			PWM3N	I/O	Capture Input and Pulse Output Negative for PWM3
			SCLK	I/O	Clock pin for SPI
			S1SCLK	I/O	Clock pin of USART1-SPI
			S2SCLK	I/O	Clock pin of USART2-SPI
			SCL	I/O	I2C clock line
			I2SCK_2	I/O	I2S clock lines
10	5	13	P1.6	I/O	Standard IO Port
			ADC6	I	ADC analogue input channels 6
			RxD_3	I	Receive pin of serial port 1
			PWM4P	I/O	Capture Input and Pulse Output Positive for PWM4
			MCLKO_2	O	Master Clock Split Output
			XTALO	O	Output pin of external crystal
			I2SMCK_2	O	Master clock line for I2S
			I2SMCK_4	O	Master clock line for I2S
11	6	14	P1.7	I/O	Standard IO Port
			ADC7	I	ADC Analogue Input Channel 7
			TxD_3	O	Transmit pin of serial port 1
			PWM4N	I/O	Capture Input and Pulse Output Negative for PWM4
			PWM5_2	I/O	Capture Input and Pulse Output for PWM5
			XTALI	I	Input pin for external crystal/external clock
12	7	15	P1.3	I/O	Standard IO Port
			ADC3	I	ADC analogue input channels 3
			MOSI	I/O	SPI Master Output Slave Input
			S1MOSI	I/O	USART1-SPI Host Output Slave Input
			S2MOSI	I/O	USART2-SPI Host Output Slave Input
			PWM2N	I/O	Capture Input and Pulse Output Negative for PWM2
			T2CLKO	O	Timer 2 clock divider output
			I2SSD_2	I/O	I2S data lines
13	8	16	UCAP	I	USB core power regulator pin

serial number			name (of a thing)	typology	instructions
LQFP48	LQFP32	PDIP40			
14	9	17	P5.4	I/O	Standard IO Port
			RST	I	Reset pin
			MCLKO	O	Master Clock Split Output
			SS_3	I	Slave select pin for SPI (host is output)
			SS	I	Slave select pin for SPI (host is output)
			S1SS_3	I	Slave select pin for USART1-SPI (host is output)
			S1SS	I	Slave select pin for USART1-SPI (host is output)
			S2SS_3	I	Slave select pin for USART2-SPI (host is output)
			S2SS	I	Slave select pin for USART2-SPI (host is output)
			PWM2P	I/O	Capture Input and Pulse Output Positive for PWM2
			PWM6_2	I/O	Capture Input and Pulse Output for PWM6
			T2	I	Timer 2 External clock input
			ADC2	I	ADC analogue input channels 2
			I2SWS_2	I/O	Channel selector line for I2S
			I2SWS_4	I/O	Channel selector line for I2S
			I2SMCK	O	I2S master clock line
			I2SMCK_3	O	I2S master clock line
15	10	18	Vcc	VCC	power supply pin
			AVcc	VCC	ADC Power Pin
16	11	19	Vref+	I	ADC reference voltage pin
17	12	20	Gnd	GND	earth (wire)
			Agnd	GND	ADC Ground
			Vref-	I	Reference Voltage Ground for ADC
18			P4.0	I/O	Standard IO Port
			MOSI_3	I/O	SPI Master Output Slave Input
			S1MOSI_3	I/O	USART1-SPI Host Output Slave Input
			S2MOSI_3	I/O	USART2-SPI Host Output Slave Input
			I2SSD_4	I/O	I2S data lines
19	13	21	P3.0	I/O	Standard IO Port
			D-	I/O	USB Data Port
			RxD	I	Receive pin of serial port 1
			INT4	I	External interrupt 4

serial number			name (of a thing)	typology	clarification
LQFP48	LQFP32	PDIP40			
20	14	22	P3.1	I/O	Standard IO Port
			D+	I/O	USB Data Port
			TxD	O	Transmit pin of serial port 1
21	15	23	P3.2	I/O	Standard IO Port
			INT0	I	External Interrupt 0
			SCLK_4	I/O	Clock pin for SPI
			SCL_4	I/O	I2C clock line
			PWMETI	I	PWM External Trigger Input Pin
			PWMETI2	I	PWM external trigger input pin 2
			I2SCK	I/O	I2S clock lines
22	16	24	P3.3	I/O	Standard IO Port
			INT1	I	External Interrupt 1
			MISO_4	I/O	SPI Host Input Slave Output
			SDA_4	I/O	I2C interface data lines
			PWM4N_4	I/O	Capture Input and Pulse Output Negative for PWM4
			PWM7_2	I/O	Capture Input and Pulse Output of PWM7
23	17	25	P3.4	I/O	Standard IO Port
			T0	I	Timer 0 External clock input
			T1CLKO	O	Timer 1 clock divider output
			MOSI_4	I/O	SPI Master Output Slave Input
			PWM4P_4	I/O	Capture Input and Pulse Output Positive for PWM4
			PWM8_2	I/O	Capture Input and Pulse Output for PWM8
			CMPO	O	Comparator Output
			I2SSD	I/O	I2S data lines
24			P5.0	I/O	Standard IO Port
			RxD3_2	I	Receive pin of serial port 3
			CMP+_2	I	Comparator positive input
			CAN_RX_2	I	CAN bus receive pin
25			P5.1	I/O	Standard IO Port
			TxD3_2	O	Serial port 3 transmitter pin
			CMP+_3	I	Comparator positive input
			CAN_TX_2	O	CAN bus transmitter pin

serial number			name (of a thing)	typology	instructions
LQFP48	LQFP32	PDIP40			
26	18	26	P3.5	I/O	Standard IO Port
			T1	I	Timer 1 External clock input
			T0CLKO	O	Timer 0 Clock divider output
			SS_4	I	Slave select pin for SPI (host is output)
			PWMFLT	I	Enhanced PWM external exception detection pin
			I2SWS	I/O	Channel selector line for I2S
27	19	27	P3.6	I/O	Standard IO Port
			INT2	I	External Interrupt 2
			RxD_2	I	Receive pin of serial port 1
			CMP-	I	Comparator negative input
28	20	28	P3.7	I/O	Standard IO Port
			INT3	I	External interrupt 3
			TxD_2	O	Transmit pin of serial port 1
			CMP+	I	Comparator positive input
29		29	P4.1	I/O	Standard IO Port
			MISO_3	I/O	SPI Host Input Slave Output
			S1MISO_3	I/O	USART1-SPI Host Input Slave Output
			S2MISO_3	I/O	USART2-SPI Host Input Slave Output
			CMPO_2	O	Comparator Output
			PWMETI_3	I	PWM External Trigger Input Pin
30		30	P4.2	I/O	Standard IO Port
			WR	O	Write signal line for external bus
			CAN_RX_3	I	CAN bus receive pin
31			P4.3	I/O	Standard IO Port
			RxD_4	I	Receive pin of serial port 1
			SCLK_3	I/O	Clock pin for SPI
			S1SCLK_3	I/O	Clock pin of USART1-SPI
			S2SCLK_3	I/O	Clock pin of USART2-SPI
			I2SCK_4	I/O	I2S clock lines
32		31	P4.4	I/O	Standard IO Port
			RD	O	Read signal line of the external bus
			TxD_4	O	Transmit pin of serial port 1
33	21	32	P2.0	I/O	Standard IO Port
			A8	I	address bus (computing)
			PWM1P_2	I/O	Capture Input and Pulse Output Positive of PWM1
			PWM5	I/O	Capture Input and Pulse Output for PWM5

serial number			name (of a thing)	typology	instructions
LQFP48	LQFP32	PDIP40			
34	22	33	P2.1	I/O	Standard IO Port
			A9	I	address bus
			PWM1N_2	I/O	Capture Input and Pulse Output Negative for PWM1
			PWM6	I/O	Capture Input and Pulse Output for PWM6
35	23	34	P2.2	I/O	Standard IO Port
			A10	I	address bus
			SS_2	I	Slave select pin for SPI (host is output)
			S1SS_2	I	Slave select pin for USART1-SPI (host is output)
			S2SS_2	I	Slave select pin for USART2-SPI (host is output)
			PWM2P_2	I/O	Capture Input and Pulse Output Positive for PWM2
			PWM7	I/O	Capture Input and Pulse Output of PWM7
			I2SWS_3	I/O	Channel selector line for I2S
36	24	35	P2.3	I/O	Standard IO Port
			A11	I	address bus
			MOSI_2	I/O	SPI Master Output Slave Input
			S1MOSI_2	I/O	USART1-SPI Host Output Slave Input
			S2MOSI_2	I/O	USART2-SPI Host Output Slave Input
			PWM2N_2	I/O	Capture Input and Pulse Output Negative for PWM2
			PWM8	I/O	Capture Input and Pulse Output for PWM8
			I2SSD_3	I/O	I2S data lines
37	25	36	P2.4	I/O	Standard IO Port
			A12	I	address bus
			MISO_2	I/O	SPI Host Input Slave Output
			S1MISO_2	I/O	USART1-SPI Host Input Slave Output
			S2MISO_2	I/O	USART2-SPI Host Input Slave Output
			SDA_2	I/O	I2C interface data lines
			PWM3P_2	I/O	Capture Input and Pulse Output Positive for PWM3
38	26	37	P2.5	I/O	Standard IO Port
			A13	I	address bus
			SCLK_2	I/O	Clock pin for SPI
			S1SCLK_2	I/O	Clock pin of USART1-SPI
			S2SCLK_2	I/O	Clock pin of USART2-SPI
			SCL_2	I/O	I2C clock line
			PWM3N_2	I/O	Capture Input and Pulse Output Negative for PWM3
			I2SCK_3	I/O	I2S clock lines

39	27	38	P2.6	I/O	Standard IO Port
			A14	I	address bus
			PWM4P_2	I/O	Capture Input and Pulse Output Positive for PWM4

serial number			name (of a thing)	typology	instructions
LQFP48	LQFP32	PDIP40			
40	28	39	P2.7	I/O	Standard IO Port
			A15	I	address bus
			PWM4N_2	I/O	Capture Input and Pulse Output Negative for PWM4
41		40	P4.5	I/O	Standard IO Port
			ALE	O	address latch signal
			CAN_TX_3	O	CAN bus transmitter pin
42			P4.6	I/O	Standard IO Port
			RxD2_2	I	Receive pin of serial port 2
			CAN2_RX_3	I	CAN2 bus receive pin
			LIN_RX_3	I	LIN bus receive pin
43	29	1	P0.0	I/O	Standard IO Port
			AD0	I	address bus
			ADC8	I	ADC analogue input channels 8
			RxD3	I	Receive pin of serial port 3
			PWM5_3	I/O	Capture Input and Pulse Output for PWM5
			CAN_RX	I	CAN bus receive pin
44	30	2	P0.1	I/O	Standard IO Port
			AD1	I	address bus
			ADC9	I	ADC analogue input channels 9
			TxD3	O	Serial port 3 transmitter pin
			PWM6_3	I/O	Capture Input and Pulse Output for PWM6
			CAN_TX	O	CAN bus transmitter pin
45	31	3	P0.2	I/O	Standard IO Port
			AD2	I	address bus
			ADC10	I	ADC Analogue input channels 10
			RxD4	I	Receive pin of serial port 4
			PWM7_3	I/O	Capture Input and Pulse Output of PWM7
			CAN2_RX	I	CAN2 bus receive pin
			LIN_RX	I	LIN bus receive pin

serial number			name (of a thing)	typology	instructions
LQFP48	LQFP32	PDIP40			
46	32	4	P0.3	I/O	Standard IO Port
			AD3	I	address bus
			ADC11	I	ADC Analogue Input Channel 11
			TxD4	O	Transmit pin of serial port 4
			PWM8_3	I/O	Capture Input and Pulse Output for PWM8
			CAN2_TX	O	CAN2 bus transmitter pin
			LIN_TX	O	LIN bus transmitter pin
47		5	P0.4	I/O	Standard IO Port
			AD4	I	address bus (computing)
			ADC12	I	ADC Analogue Input Channel 12
			T3	I	Timer 3 External clock input
48			P5.2	I/O	Standard IO Port
			RxD4_2	I	Receive pin of serial port 4
			CAN2_RX_2	I	CAN2 bus receive pin
			LIN_RX_2	I	LIN bus receive pin

3 Function foot switching

The special peripheral serial, SPI, PWM, I² C, CAN, LIN and bus control pins of the STC32G series microcontrollers can be used in multiple I/Os.

Direct switching is performed to enable time-sharing multiplexing of a peripheral as multiple devices.

3.1 Function pin switching related registers

notation	descriptions	addr-ess	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW1	Peripheral Port Switching Register 1	A2H	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]		nn00,0000
P_SW2	Peripheral Port Switching Register 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000
P_SW3	Peripheral Port Switching Register 2	BBH	I2S_S[1:0]		S2SPI_S[1:0]		S1SPI_S[1:0]		CAN2_S[1:0]		0000,0000

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
MCLKOCR	Master Clock Output Control Register	7EFE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
PWMA_PS	PWMA Switching Register	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMB_PS	PWMB Switching Register	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMA_ETRPS	ERT Toggle Register for PWMA	7EFEB0H						BRKAPS	ETRAPS[1:0]		xxx,x000
PWMB_ETRPS	ERT Toggle Register for PWMB	7EFEB4H						BRKBPS	ETRBPS[1:0]		xxx,x000
T3T4PIN	T3/T4 Selection Register	7EFEACH	-	-	-	-	-	-	-	T3T4SEL	xxxx,xxx0

3.1.1 Peripheral Port Switching Control Register 1 (P_SW1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]	

S1_S[1:0]: Serial port 1 function pin select 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

CAN_S[1:0]: CAN function pin select bits

CAN_S[1:0]	CAN_RX	CAN_TX
00	P0.0	P0.1
01	P5.0	P5.1
10	P4.2	P4.5
11	P7.0	P7.1

LIN_S[1:0]: LIN function pin select bits

LIN_S[1:0]	LIN_RX	LIN_TX
00	P0.2	P0.3
01	P5.2	P5.3

10	P4.6	P4.7
11	P7.2	P7.3

SPI_S[1:0]: SPI function pin select bits

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2/P5.4[1]	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P3.5	P3.4	P3.3	P3.2

[1] : 有 P1.2 port, this function is on the P1.2 port, and for models without the P1.2 port, this function is on the P5.4 口上

3.1.2 Peripheral Port Switching Control Register 2 (P_SW2)

notation	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 disabled

1: Enables access to the XFR.

When you need to access the XFR, you must set EAXFR to 1 before you can read or write to the XFR normally. It is recommended to set EAXFR to 1 during initialisation and not to change it later.

I2C_S[1:0]: I² C Function Pin Select Bits

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

CMPO_S: Comparator output pin select bit

CMPO_S	CMPO
0	P3.4
1	P4.1

S4_S: Serial port 4 function pin select bit

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

S3_S: Serial port 3 function pin select bit

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

S2_S: Serial Port 2 Function Pin Select Bit

S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P4.6	P4.7

3.1.3 Peripheral Port Switching Control Register 3 (P_SW3)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW3	BBH	I2S_S		S2SPI_S[1:0]		S1SPI_S[1:0]		CAN2_S[1:0]	

S2SPI_S[1:0]: SPI function pin select bits for USART2

S2SPI_S[1:0]	S2SS	S2MOSI	S2MISO	S2SCLK
00	P1.2/P5.4[1]	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P7.4	P7.5	P7.6	P7.7

[1] ：有 P1.2 port, this function is on the P1.2 port, and for models without the P1.2 port, this function is on the P5.4 口上

S1SPI_S[1:0]: SPI function pin select bit for USART1

S1SPI_S[1:0]	S1SS	S1MOSI	S1MISO	S1SCLK
00	P1.2/P5.4[1]	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P6.4	P6.5	P6.6	P6.7

[1] ：有 P1.2 port, this function is on the P1.2 port, and for models without the P1.2 port, this function is on the P5.4 口上

CAN2_S[1:0]: CAN2 function pin select bits

CAN2_S[1:0]	CAN2_RX	CAN2_TX
00	P0.2	P0.3
01	P5.2	P5.3
10	P4.6	P4.7
11	P7.2	P7.3

I2S_S[1:0]: I2S function pin select bits

I2S_S[1:0]	I2SCK	I2SSD	I2SMCK	I2SWS
00	P3.2	P3.4	P5.4	P3.5
01	P1.5	P1.3	P1.6	P5.4
10	P2.5	P2.3	P5.4	P2.2
11	P4.3	P4.0	P1.6	P5.4

3.1.4 Clock Select Register (MCLKOCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	7EFE07H	MCLKO_S	MCLKODIV[6:0]						

MCLKO_S: Master clock output pin select bit

MCLKO_S	MCLKO
0	P5.4
1	P1.6

3.1.5 T3/T4 Select Register (T3T4PIN)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
T3T4PIN	7EFEACH	-	-	-	-	-	-	-	T3T4SEL

T3T4SEL: T3/T3CLKO/T4/T4CLKO pin select bit

T3T4SEL	T3	T3CLKO	T4	T4CLKO
0	P0.4	P0.5	P0.6	P0.7
1	P0.0	P0.1	P0.2	P0.3

3.1.6 Advanced PWM Select Register (PWMn_PS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWMB_PS	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	

C1PS[1:0]: Advanced PWM channel 1 output pin select bits

C1PS[1:0]	PWM1P	PWM1N
00	P1.0	P1.1
01	P2.0	P2.1
10	P6.0	P6.1
11	-	-

C2PS[1:0]: Advanced PWM channel 2 output pin select bits

C2PS[1:0]	PWM2P	PWM2N
00	P1.2/P5.4[1]	P1.3
01	P2.2	P2.3
10	P6.2	P6.3
11	-	-

[1] ：有 P1.2 port, this function is on the P1.2 port, and for models without the P1.2 port, this function is on the P5.4 口上

C3PS[1:0]: Advanced PWM channel 3 output pin select bits

C3PS[1:0]	PWM3P	PWM3N
00	P1.4	P1.5
01	P2.4	P2.5
10	P6.4	P6.5
11	-	-

C4PS[1:0]: Advanced PWM channel 4 output pin select bits

C4PS[1:0]	PWM4P	PWM4N
00	P1.6	P1.7
01	P2.6	P2.7
10	P6.6	P6.7
11	P3.4	P3.3

C5PS[1:0]: Advanced PWM channel 5 output pin select bits

C5PS[1:0]	PWM5
00	P2.0
01	P1.7

10	P0.0
11	P7.4

C6PS[1:0]: Advanced PWM channel 6 output pin select bits

C6PS[1:0]	PWM6
00	P2.1
01	P5.4
10	P0.1
11	P7.5

C7PS[1:0]: Advanced PWM channel 7 output pin select bits

C7PS[1:0]	PWM7
00	P2.2
01	P3.3
10	P0.2
11	P7.6

C8PS[1:0]: Advanced PWM channel 8 output pin select bits

C8PS[1:0]	PWM8
00	P2.3
01	P3.4
10	P0.3
11	P7.7

3.1.7 Advanced PWM Function Pin Select Register (PWMx_ETRPS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETRPS	7EFEB0H						BRKAPS	ETRAPS[1:0]	
PWMB_ETRPS	7EFEB4H						BRKBPS	ETRBPS[1:0]	

ETRAPS[1:0]: External trigger pin ERI selection bit for advanced PWMA

ETRAPS [1:0]	PWMETI
00	P3.2
01	P4.1
10	P7.3
11	-

ETRBPS[1:0]: External trigger pin ERIB selection bits for advanced PWMBs

ETRBPS [1:0]	PWMETI2
00	P3.2
01	P0.6
10	-
11	-

BRKAPS: Brake Foot PWMFLT Selection Bits for Advanced PWMAs

BRKAPS	PWMFLT
0	P3.5
1	Comparator output

BRKBPS: Brake pin PWMFLT2 select bit for advanced PWMBs

BRKBPS	PWMFLT2

0	P3.5
1	Comparator output

3.2 sample procedure

3.2.1 Serial port 1 switching

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    s1_s1 = 0; s1_s0 = 0.               //RXD/P3.0, TXD/P3.1
    //S1_S1 = 0; S1_S0 = 1;             //RXD_2/P3.6, TXD_2/P3.7
    //S1_S1 = 1; S1_S0 = 0;             //RXD_3/P1.6, TXD_3/P1.7
    //S1_S1 = 1; S1_S0 = 1;             //RXD_4/P4.3, TXD_4/P4.4

    while (1);
}
```

3.2.2 Serial Port 2 Switching

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.
```

```
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    S2_S = 0;                                   //RXD2/P1.0, TXD2/P1.1
//  S2_S = 1;                                   //RXD2_2/P4.6, TXD2_2/P4.7

    while (1);
}
```

3.2.3 Serial Port 3 Switching

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             // see download software for header files

void main()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    S3_S = 0;                                   //RXD3/P0.0, TXD3/P0.1
    //S3_S = 1;                                 //RXD3_2/P5.0, TXD3_2/P5.1

    while (1);
}
```

3.2.4 Serial 4 Switching

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign 0 to set the CPU to execute the programme as fast as
                                             possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    S4_S = 0;                                //RXD4/P0.2, TXD4/P0.3
    //S4_S = 1;                              //RXD4_2/P5.2, TXD4_2/P5.3

    while (1);
}
```

3.2.5 SPI Switching

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign 0 to set the CPU to execute the programme as fast as
                                             possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
```

```
    p4m1 = 0x00;
    p4m0 = 0x00;
    p5m1 = 0x00;
    p5m0 = 0x00.
```

```
    P5M1 = 0x00.

    spi_s1 = 0; spi_s1 = 0.                    //SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
    //SPI_S1 = 0; SPI_S0 = 1;                  //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
    //SPI_S1 = 1; SPI_S0 = 0;                  //SS_3/P5.4, MOSI_3/P4.0, MISO_3/P4.1, SCLK_3/P4.3
    //SPI_S1 = 1; SPI_S0 = 1;                  //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

    while (1);
}
```

3.2.6 I2C switching

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                            // see download software for header files

void main()
{
    EAXFR = 1;                                 //Enable access to XFR
    CKCON = 0x00;                              //Set the external data bus speed to fastest
    WTST = 0x00;                               //set the program code wait parameter.
                                               //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

  icc_s1 = 0; icc_s0 = 0; //SCL/P1.5, SDA/P1.4; //SCL/P1.5, SDA/P1.4.   //SCL/P1.5, SDA/P1.4
    //I2C_S1 = 0; I2C_S0 = 1;                  //SCL_2/P2.5, SDA_2/P2.4
    //I2C_S1 = 1; I2C_S0 = 0;                  //SCL_3/P7.7, SDA_3/P7.6
    //I2C_S1 = 1; I2C_S0 = 1;                  //SCL_4/P3.2, SDA_4/P3.3

    while (1);
}
```

3.2.7 Comparator output switching

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                            // see download software for header files
```

```
void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    CMPO_S = 0;                         //CMPO/P3.4
    //CMPO_S = 1; //CMPO_2/P4.1         //CMPO_2/P4.1

    while (1);
}
```

3.2.8 Master clock output switching

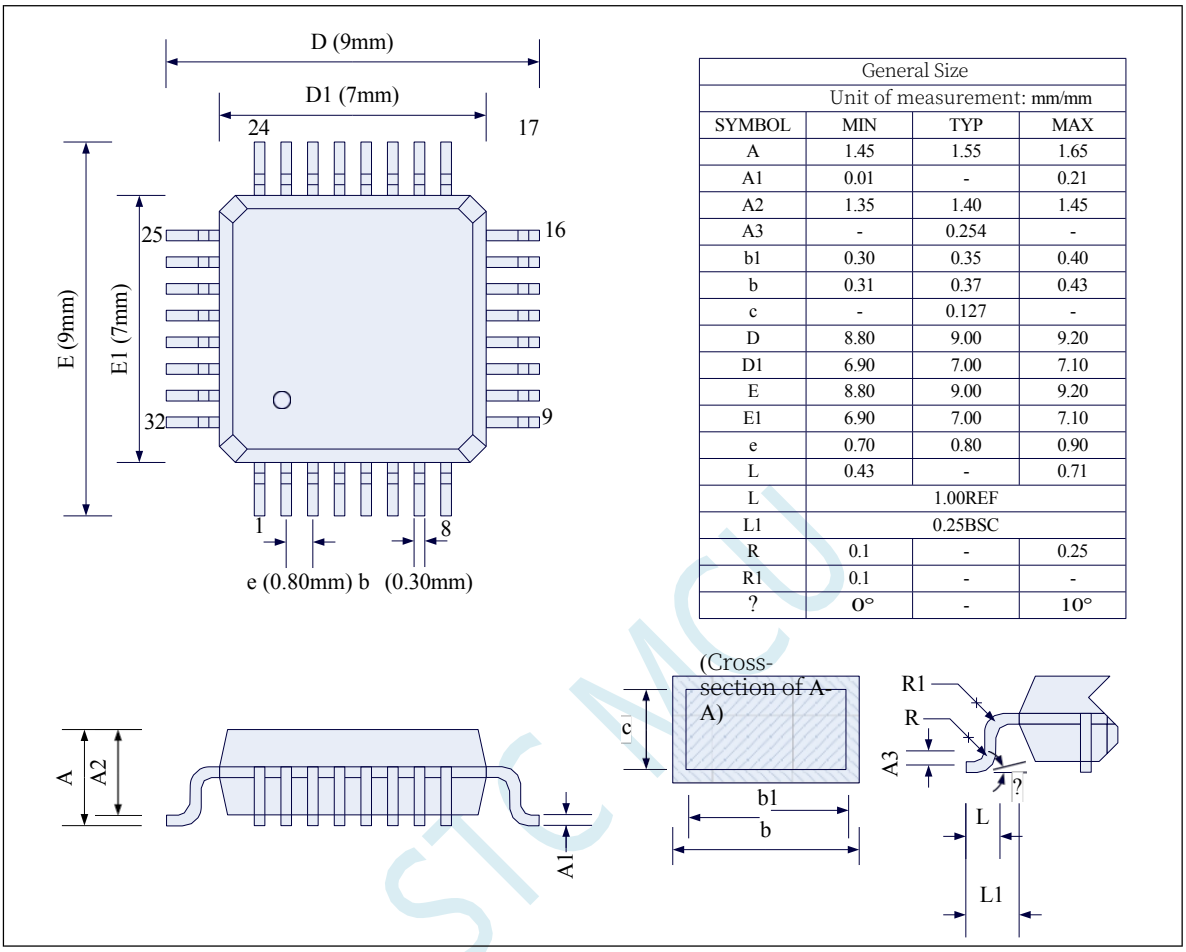
```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    mclkocr = 0x04.                     //IRC/4 output via MCLKO/P5.4
```

```
    MCLKOCR = 0x84;                                 //IRC/4 output via MCLKO_2/P1.6
```

```
    while (1);
}
```

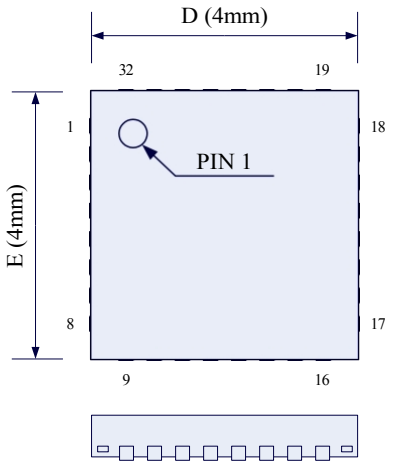
4 Package Dimension Drawing

4.1 LQFP32 Package Dimension Drawing (9mm*9mm)

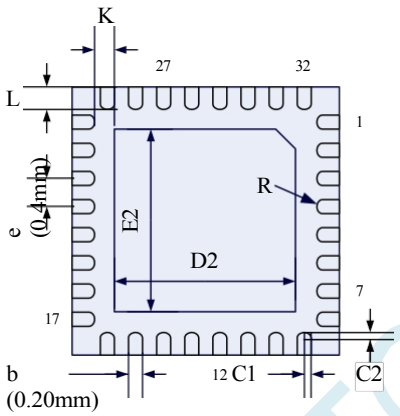


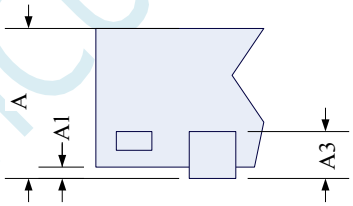
General Size			
Unit of measurement: mm/mm			
SYMBOL	MIN	TYP	MAX
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
b1	0.30	0.35	0.40
b	0.31	0.37	0.43
c	-	0.127	-
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.70	0.80	0.90
L	0.43	-	0.71
L	1.00REF		
L1	0.25BSC		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

4.2 QFN32 Package Dimension Drawing (4mm*4mm)

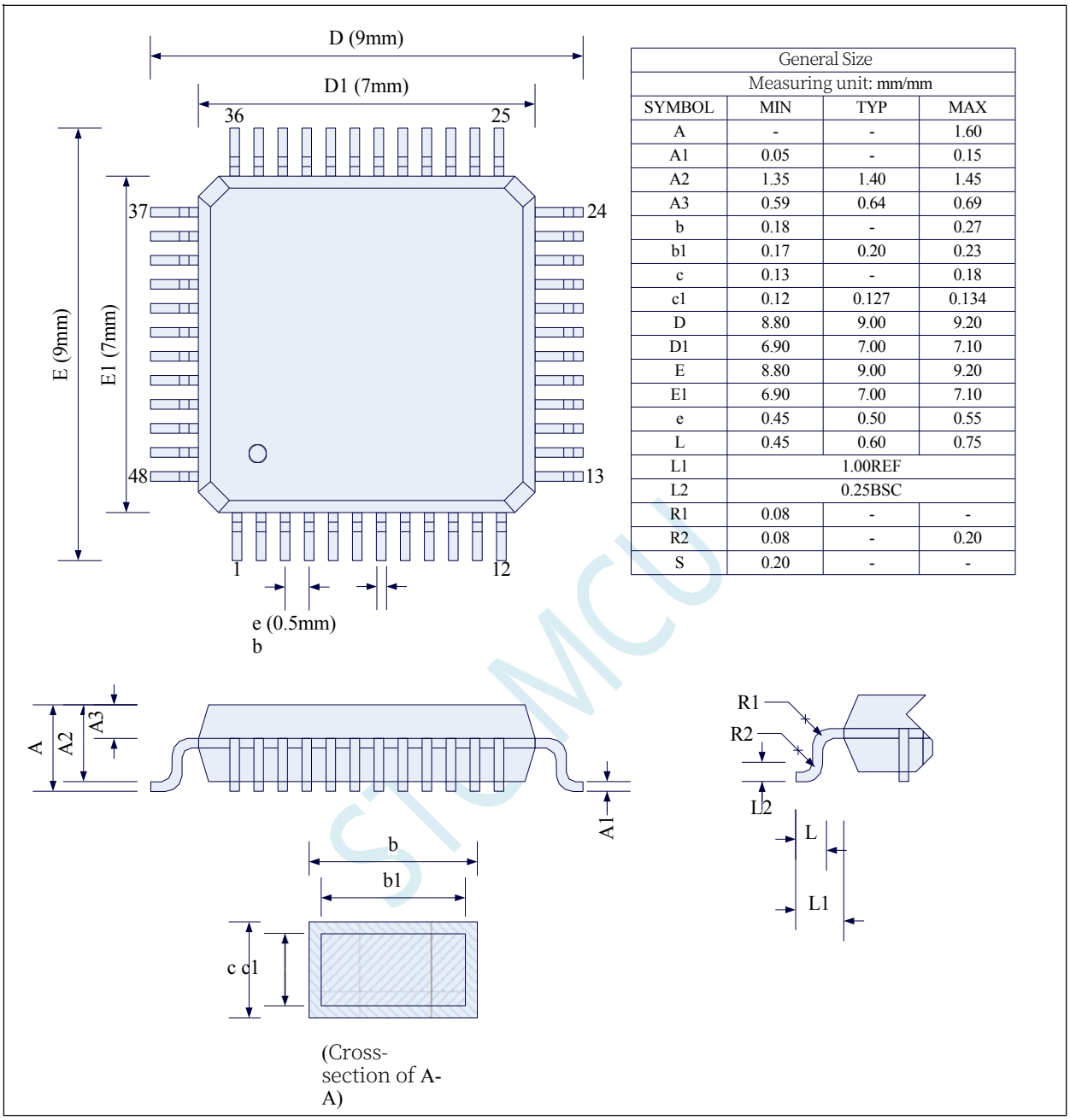


General Size			
Unit of measurement: mm/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.60	2.70	2.80
E2	2.60	2.70	2.80
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K		0.25REF	
R	0.09	-	-
C1	-	0.16	-
C2	-	0.16	-



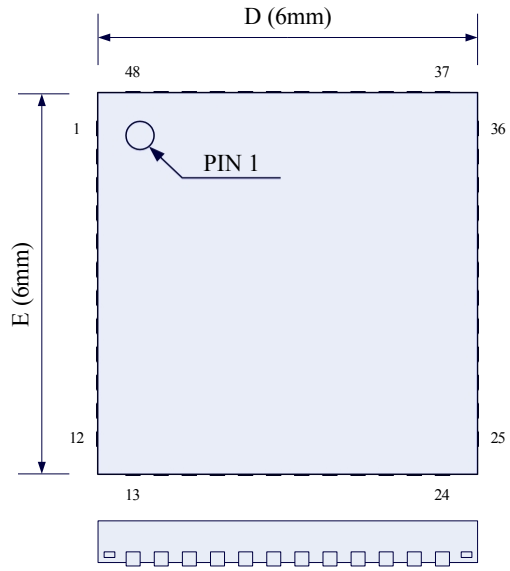


4.3 LQFP48 Package Dimension Drawing (9mm*9mm)

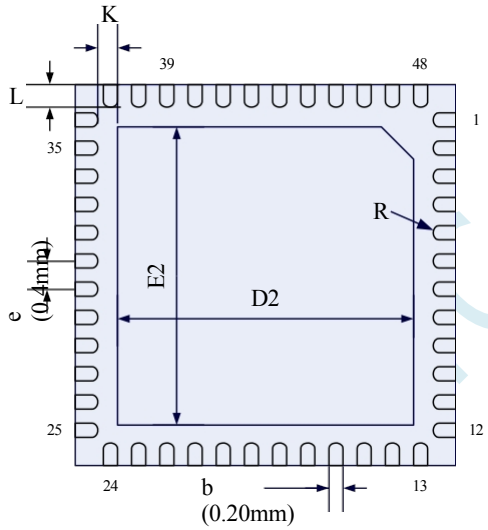


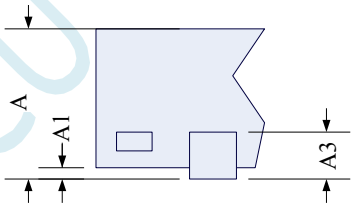
General Size			
Measuring unit: mm/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.45	0.50	0.55
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-

4.4 QFN48 Package Dimension Drawing (6mm*6mm)

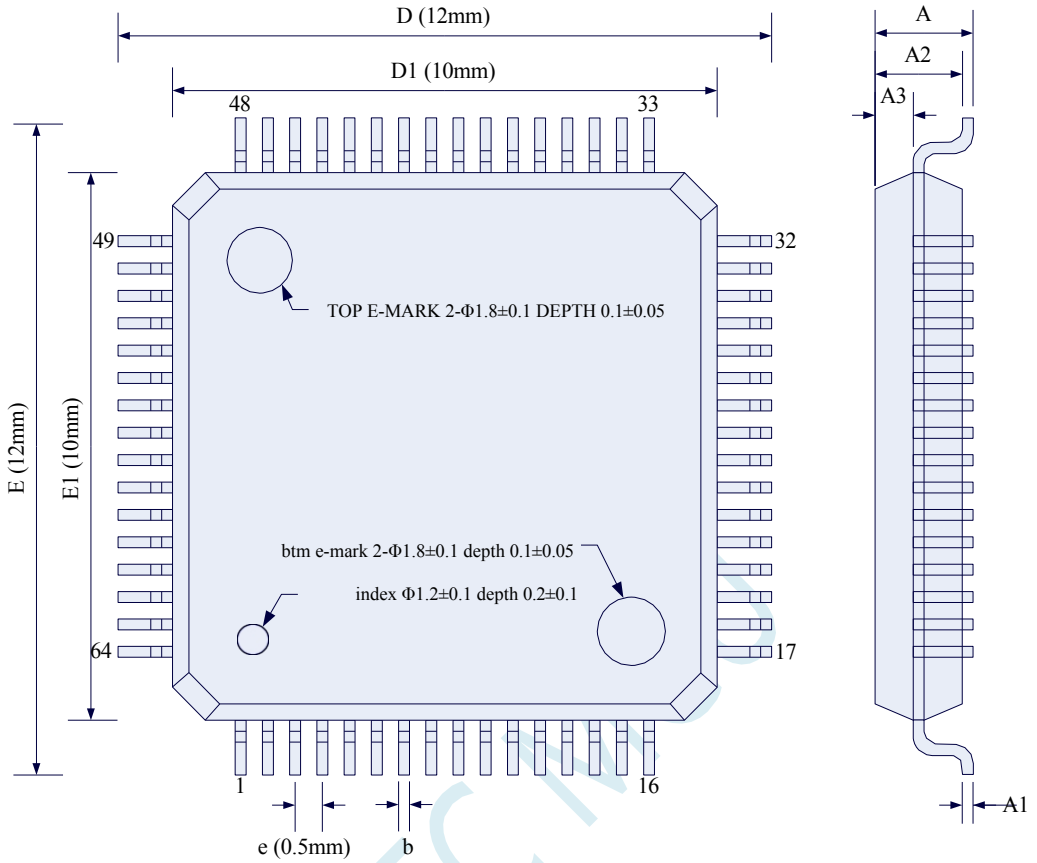


General Size			
Unit of measurement: mm/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3		0.20REF	
b	0.15	0.20	0.25
D	5.90	6.00	6.10
E	5.90	6.00	6.10
D2	3.95	4.05	4.15
E2	3.95	4.05	4.15
e	0.35	0.40	0.45
L	0.35	0.40	0.45
K		0.20	
R	0.09	-	-

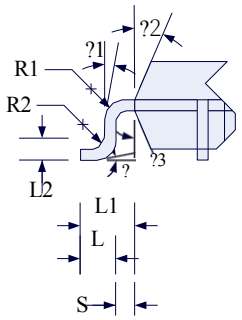


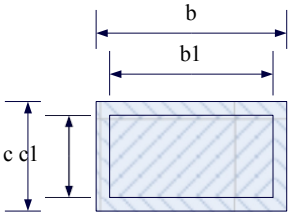


4.5 LQFP64 Package Dimension Drawing (12mm*12mm)



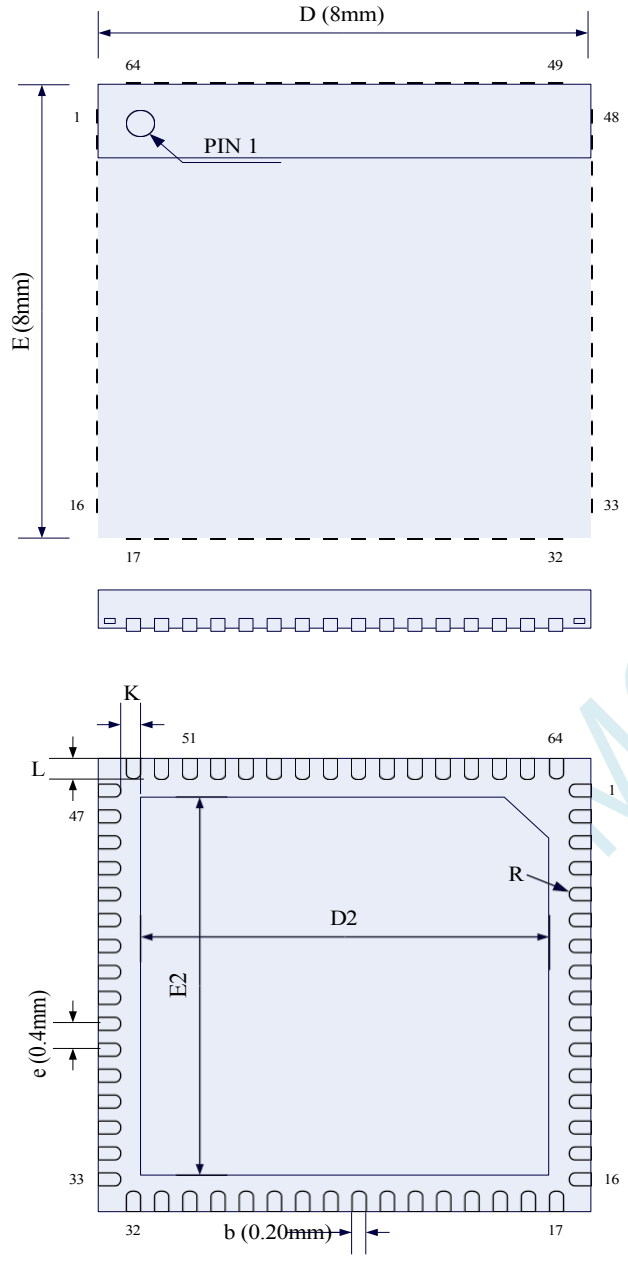
General Size			
Unit of measurement: mm/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.18	-	0.27
b1	0.17	0.20	0.23
c	0.13	-	0.18
c1	0.12	0.127	0.134
D	11.80	12.00	12.20
D1	9.90	10.00	10.10
E	11.80	12.00	12.20
E1	9.90	10.00	10.10
e	0.50BSC		
L	0.45	0.60	0.75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





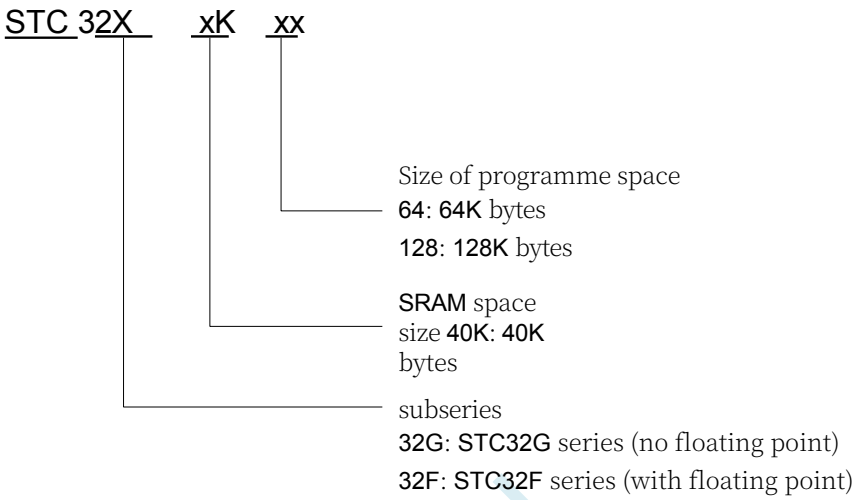
(Cross-section of A-A)

4.6 QFN64 Package Dimension Drawing (8mm*8mm)



General Size			
Unit of measurement: mm/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	-	-

4.7 STC32G Series Microcontroller Naming Rules

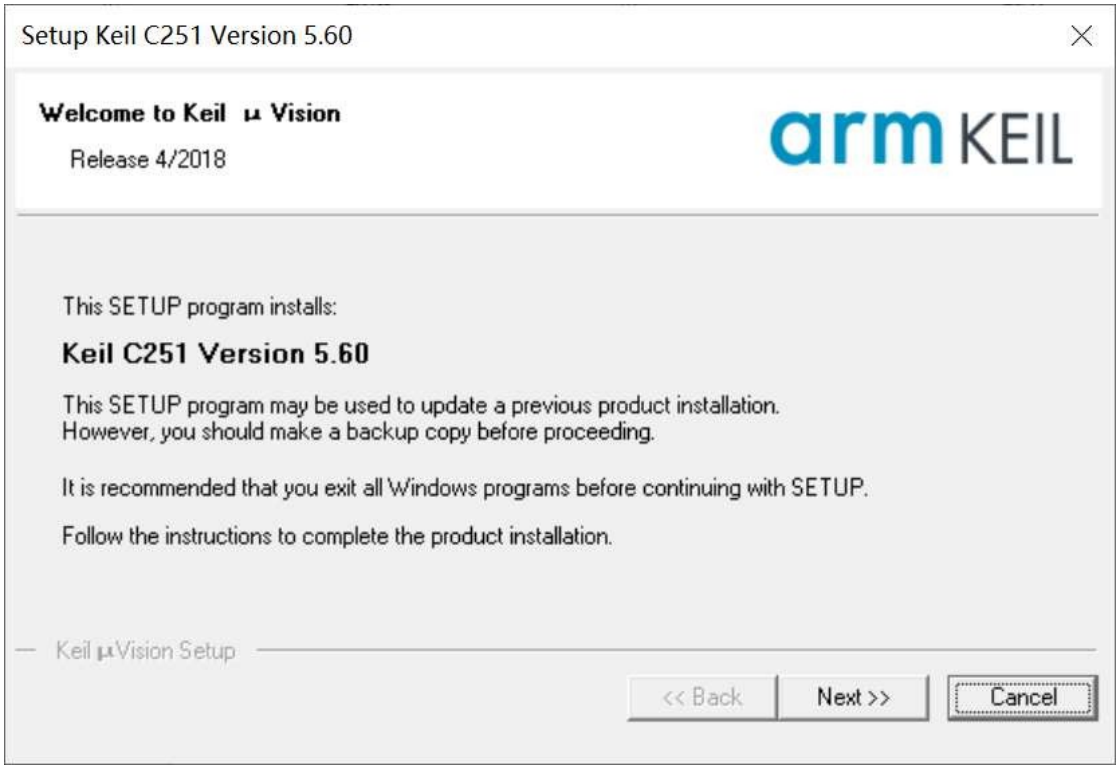


5 Establishment of compilation and simulation development environment and ISP downloads

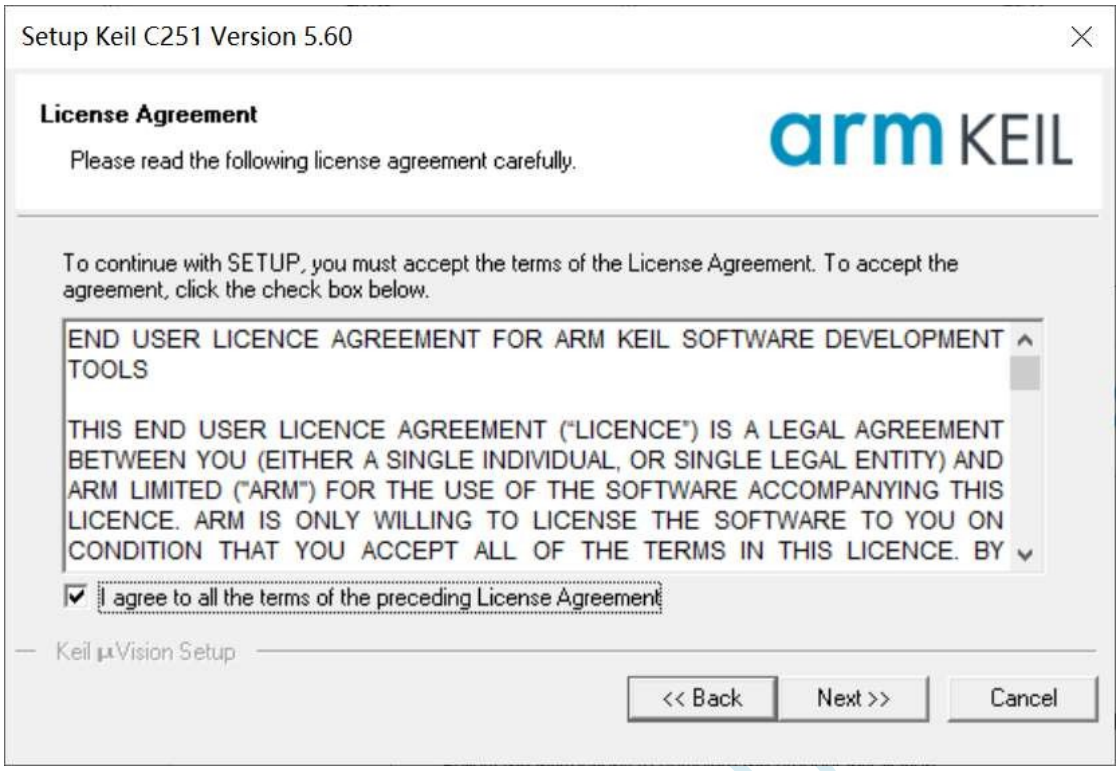
5.1 Installing Keil

5.1.1 Installing the C251 compilation environment

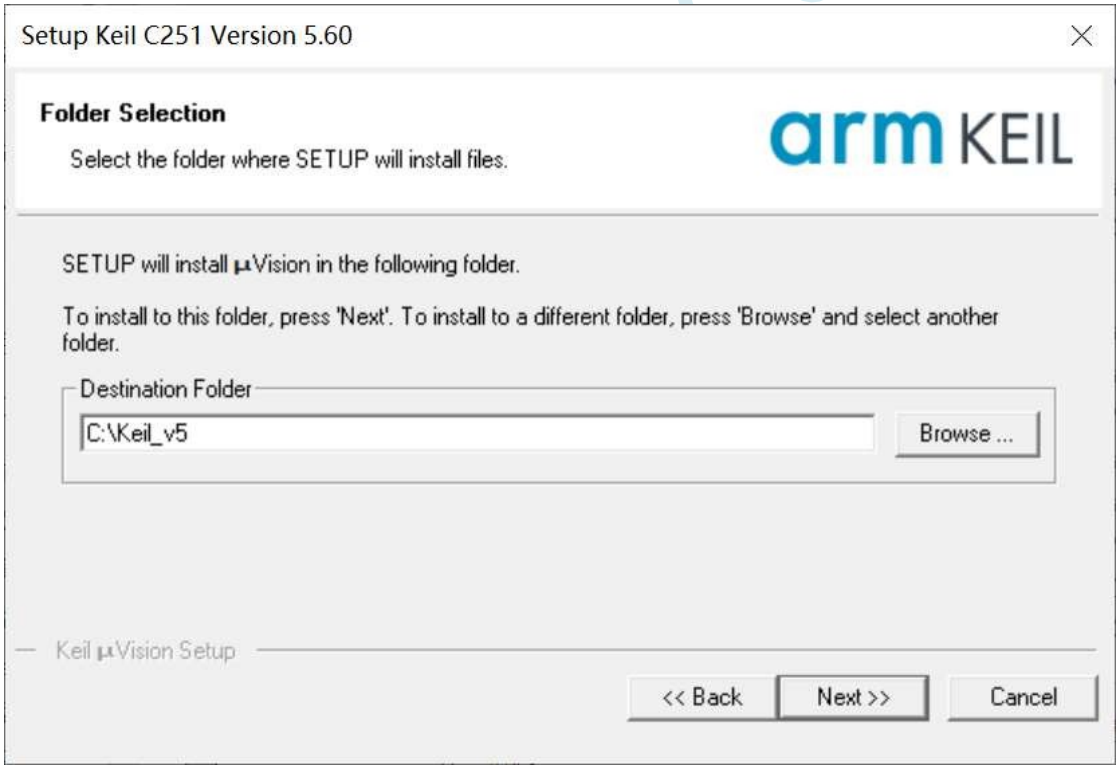
Firstly, go to the Keil website and download the latest version of the C251 installation package, the download link is as follows:
Keil Product Downloads
Fill in the information as you like, click OK and then go to the download page to download. Double click the downloaded package to start installation, click "Next".



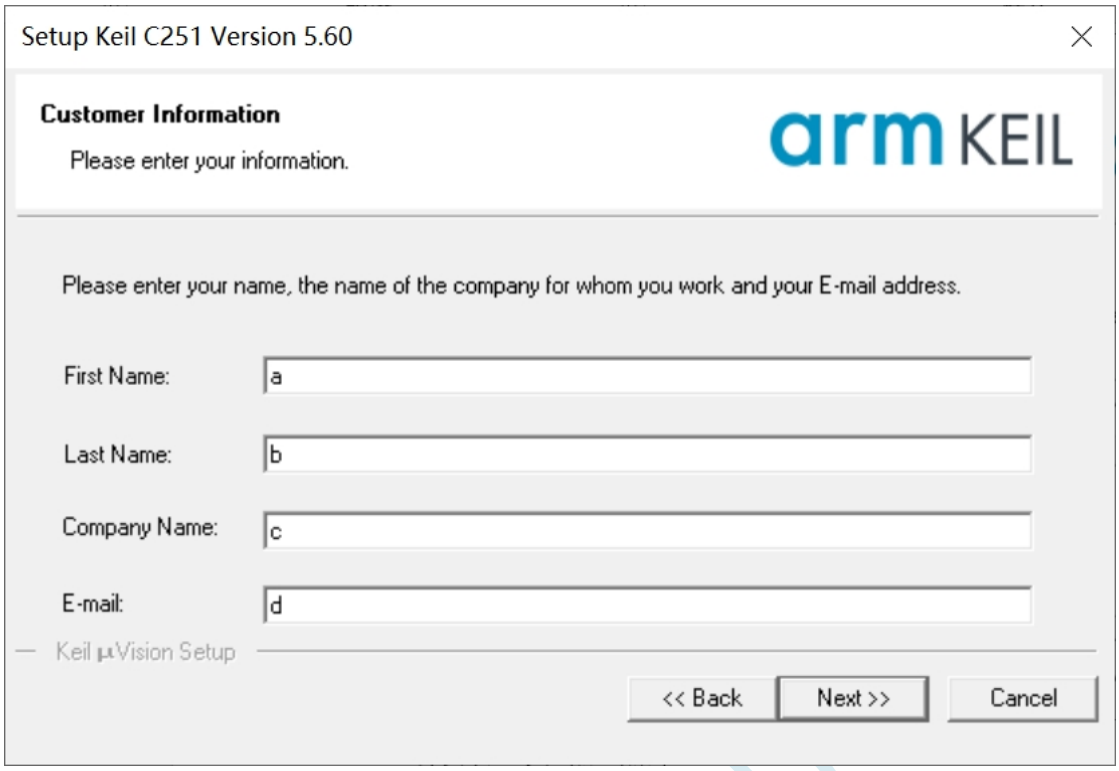
Tick "I agree to all the terms of the preceding License Agreement", then click "Next".



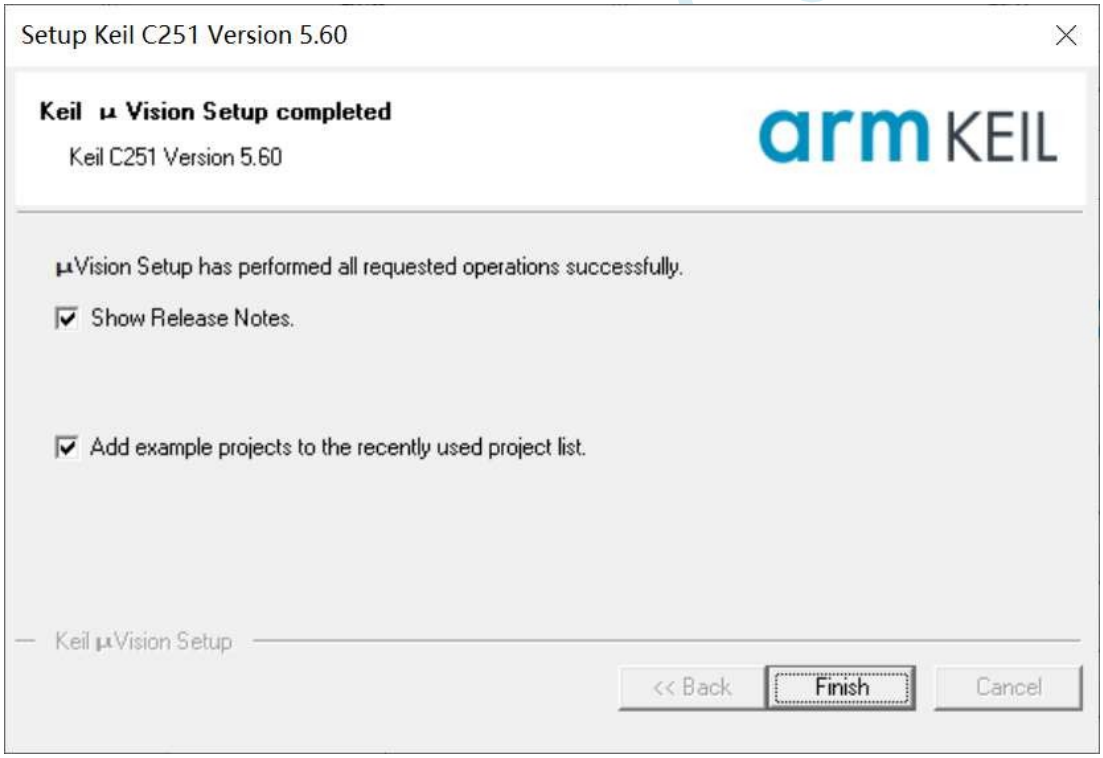
Select the installation directory and click "Next".



Fill in your personal details and click "Next".

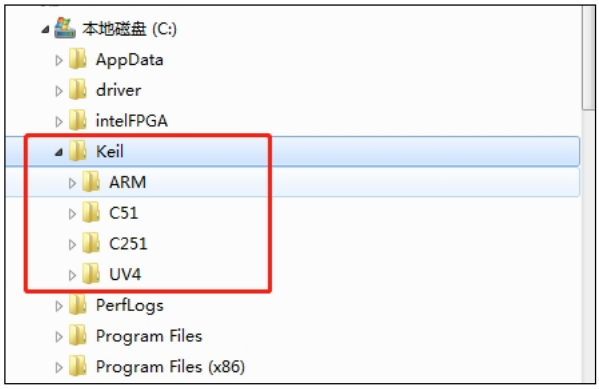


Installation is complete, click "Finish" to end.



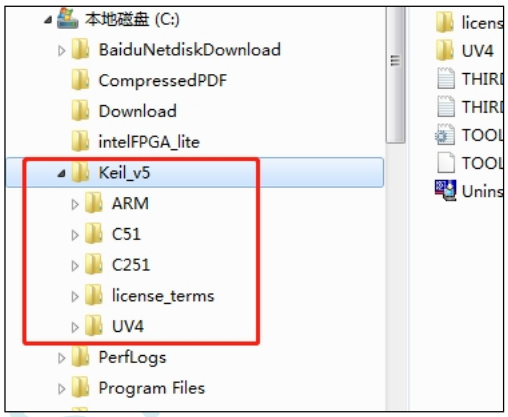
5.1.2 How to install Keil's C51, C251 and MDK at the same time

The default installation directory for older versions of Keil software is C:\Keil. The C51, C251 and MDK will be installed in the C51, C251 and ARM directories under the C:\Keil directory, respectively, as shown below.



The default installation directory for the new Keil software is C:\Keil_v5, and C51, C251 and MDK will be installed in C:\Keil_v5 respectively.

directory in the C51, C251 and ARM directories as shown below.

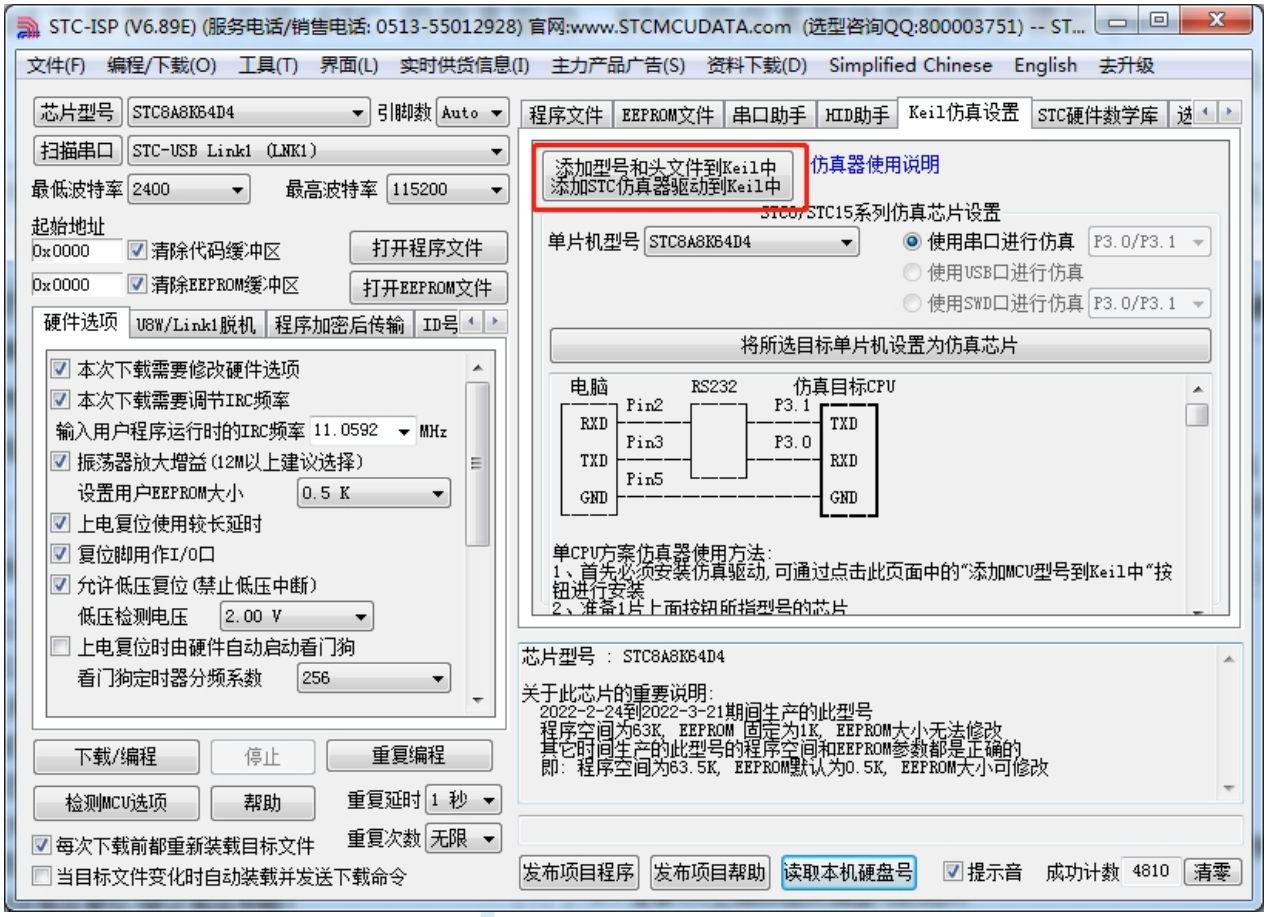


Whether it is a new version or an old version, C51, C251 and MDK are installed in different directories and there is no conflict. Harmony of the software is also carried out by the 3 software separately, the software that has been installed and set up before will not be changed because of the installation of the new software. So you only need to follow the default installation method, Keil software will take care of it automatically.

5.2 Adding Models and Header Files to Keil

Before using Keil, you need to install the STC emulation driver, the installation procedure of STC emulation driver is as follows:

Firstly, open the ISP download software of STC, and then click "Add model and header file to Keil Add STC emulator driver to Keil" button in "Keil Emulation Settings" page of the



function area on the right side of the software:

When pressed, the following screen appears:

Locate the directory to the Keil software installation directory and then OK. After successful installation, the following prompt box will pop up:

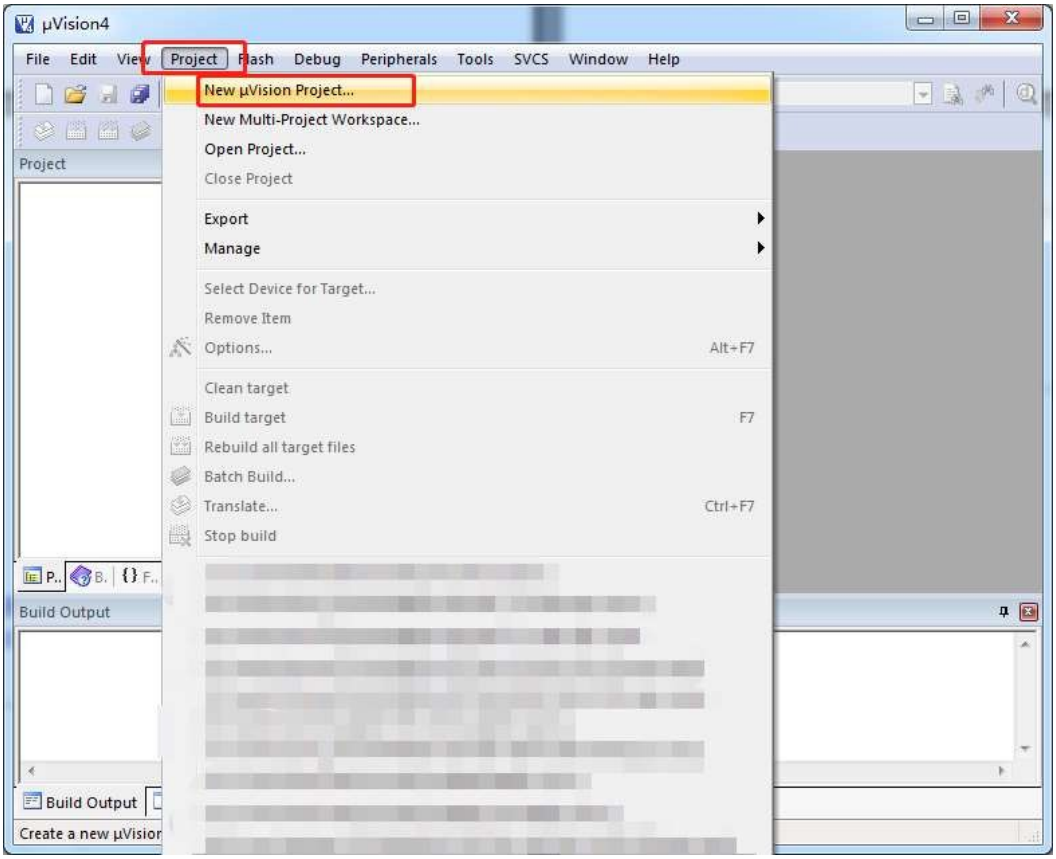
That means the driver is correctly installed

The header files are copied to the "C251\INC\STC" directory in the Keil installation directory by default. Using "#include <STC32G.H>" or "#include "STC32G.H"" for inclusion in C code works correctly!

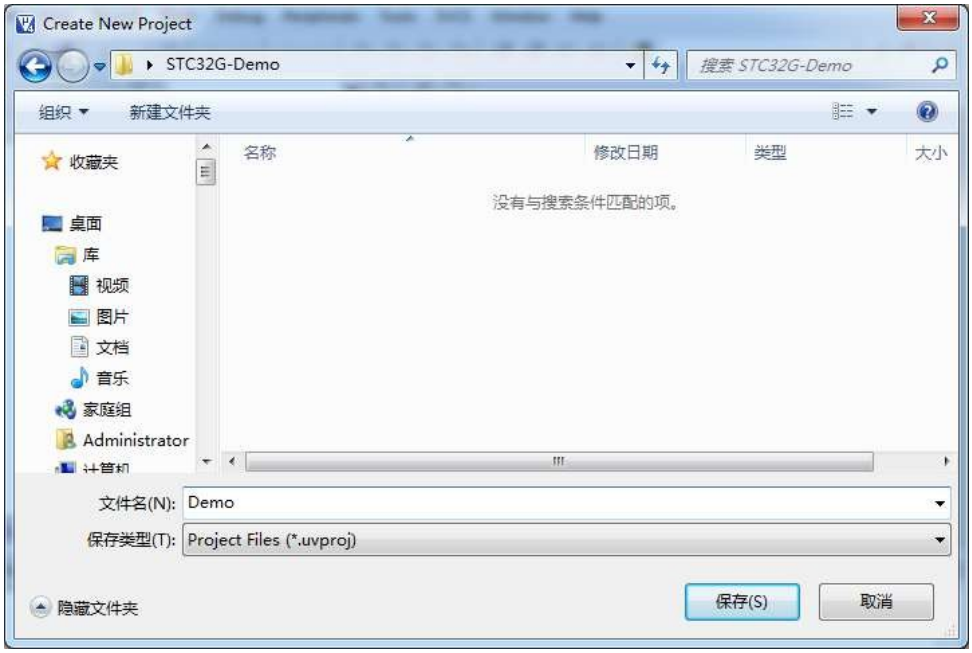
5.3 Creating and setting up a project with more than 64K program code (Source mode)

5.3.1 Setting the project path and project name

Open Keil software and click "New uVision Project ..." in the "Project" menu. in the "Project" menu.

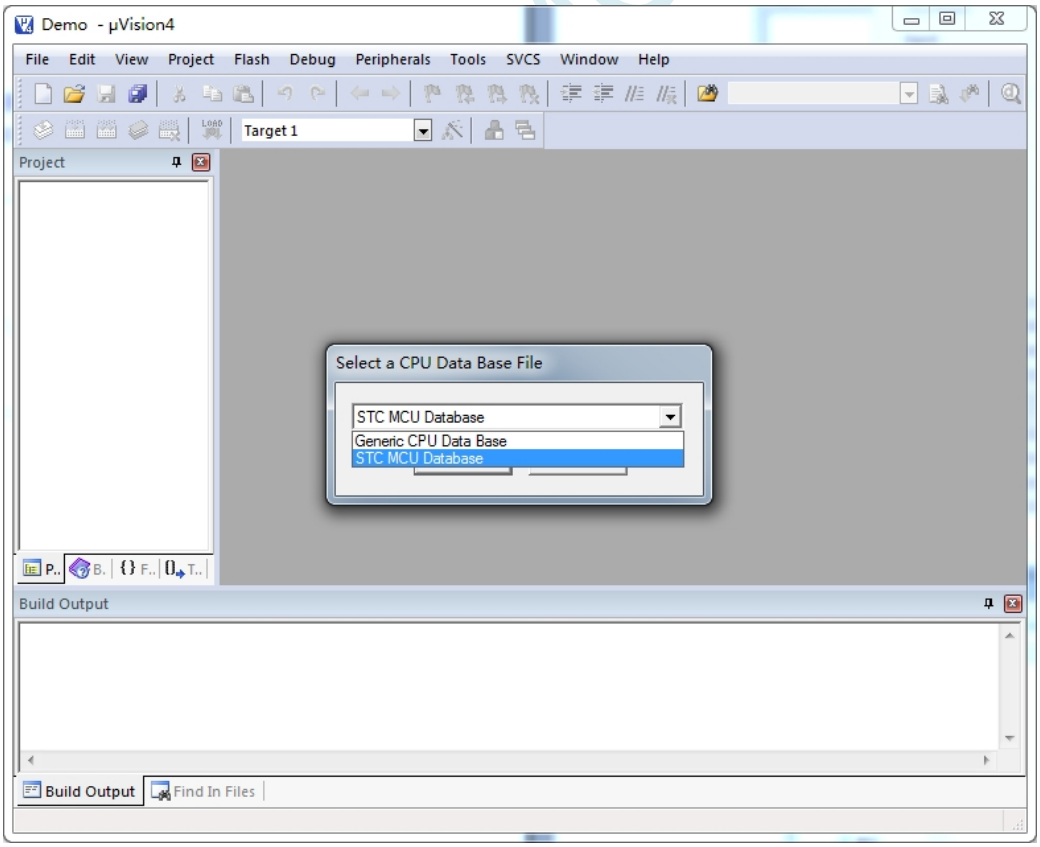


Locate the directory in the prepared project folder and enter the project name (e.g. Demo)

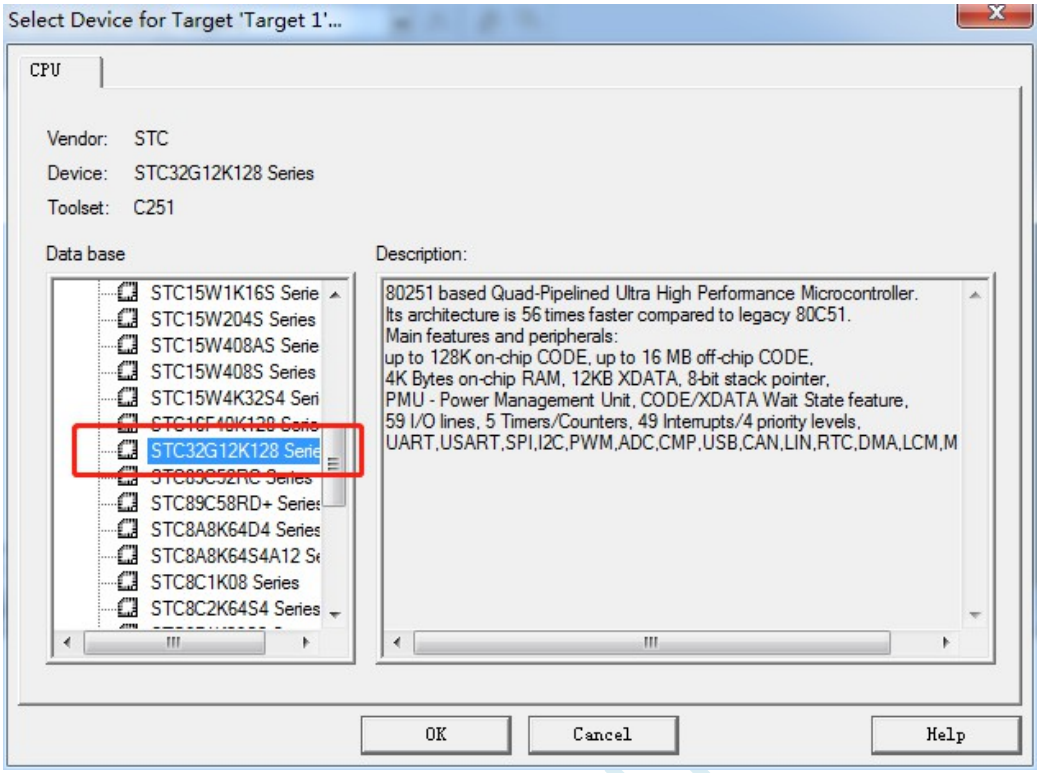


5.3.2 Select target microcontroller model (STC32G12K128)

Select "STC MCU Database" in the "Select a CPU Data Base File" pop-up window.



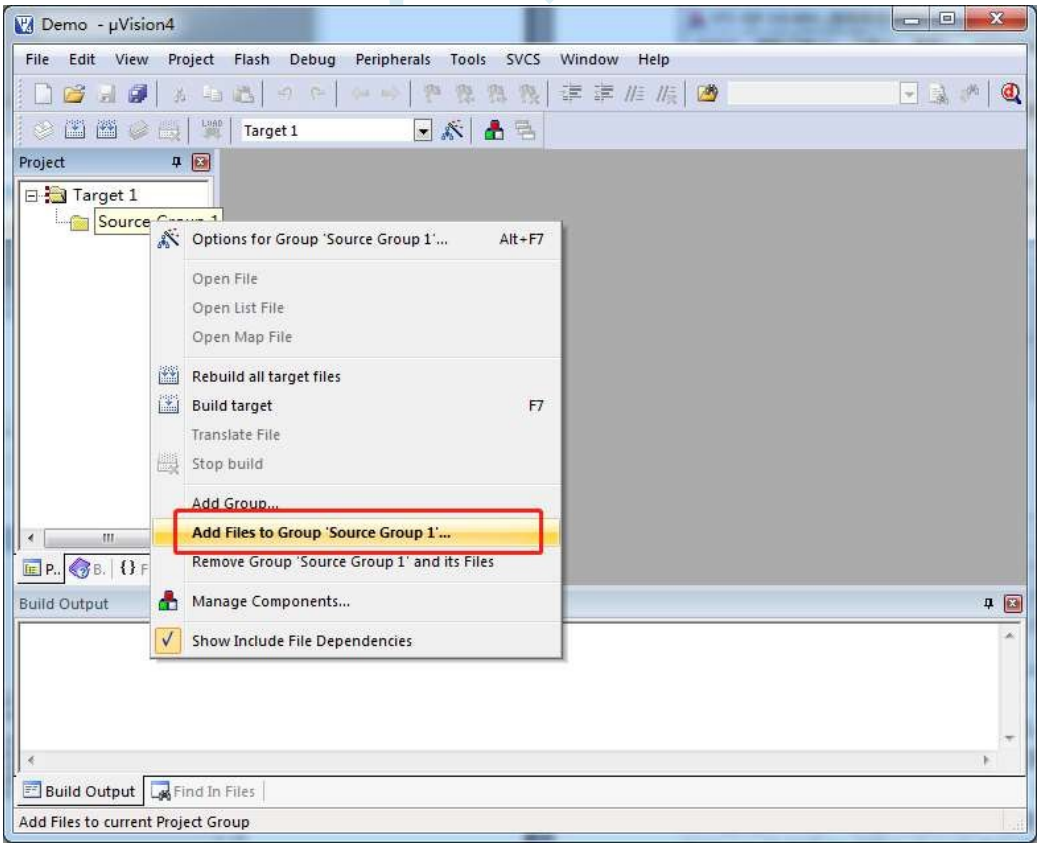
Select the correct target microcontroller model (e.g. STC32G12K128) in the "Select Device for Target ..."



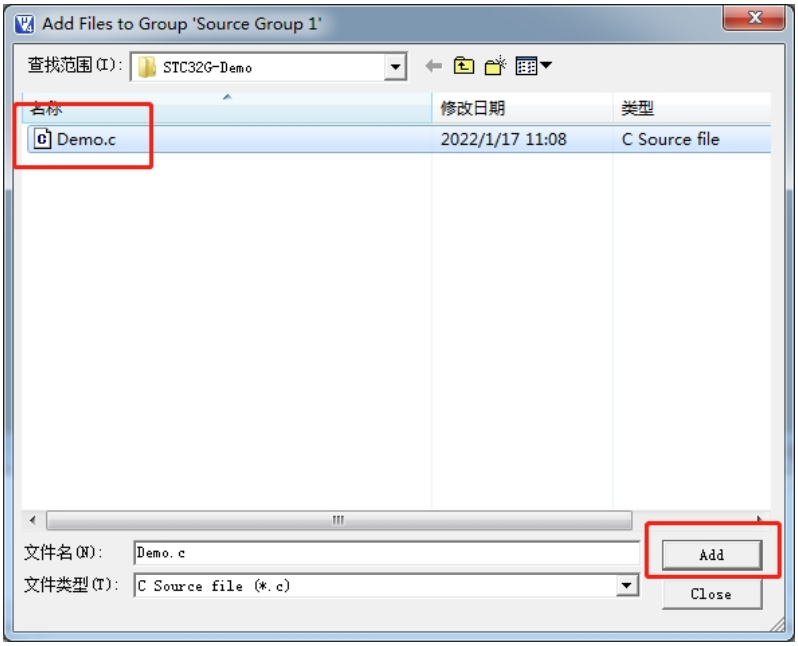
window. window, select the correct target microcontroller model (e.g. STC32G12K128).

5.3.3 Adding source code files to a project

As shown in the figure below, right-click on the icon where "Source Group 1" is located and select "Add Files to Group 'Source Group 1'..." in the context menu.

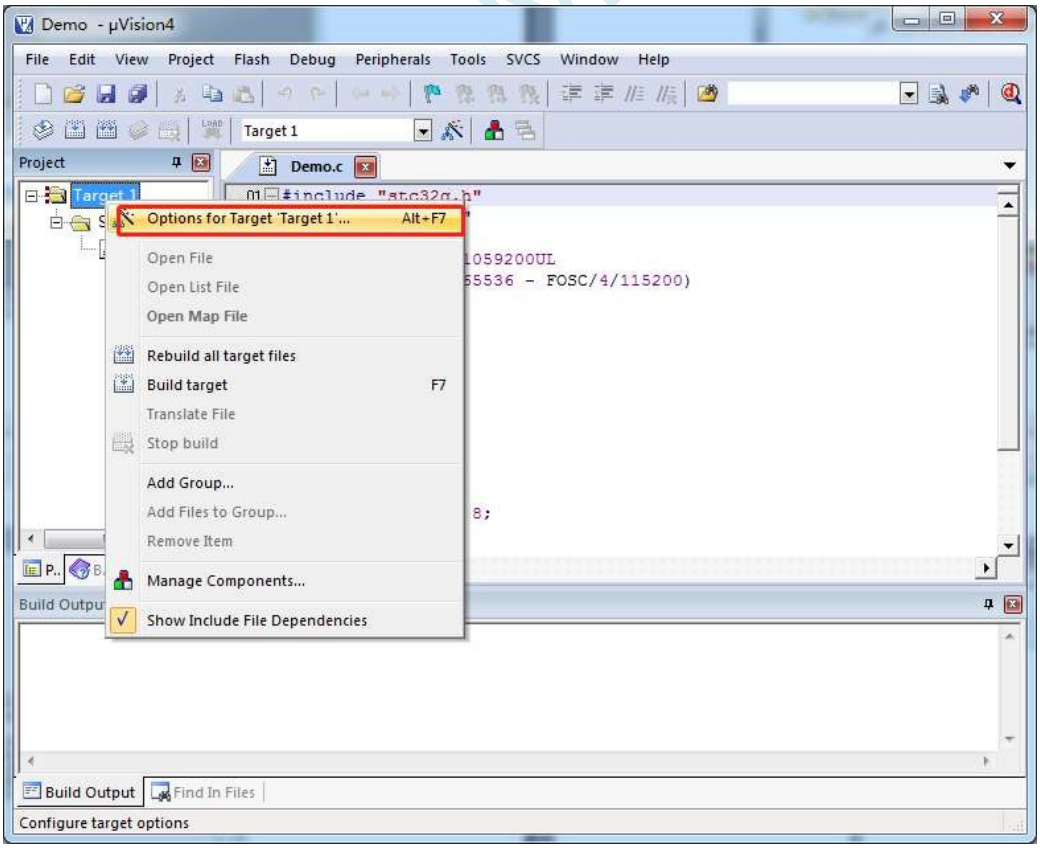


Select the edited code file to add to the project

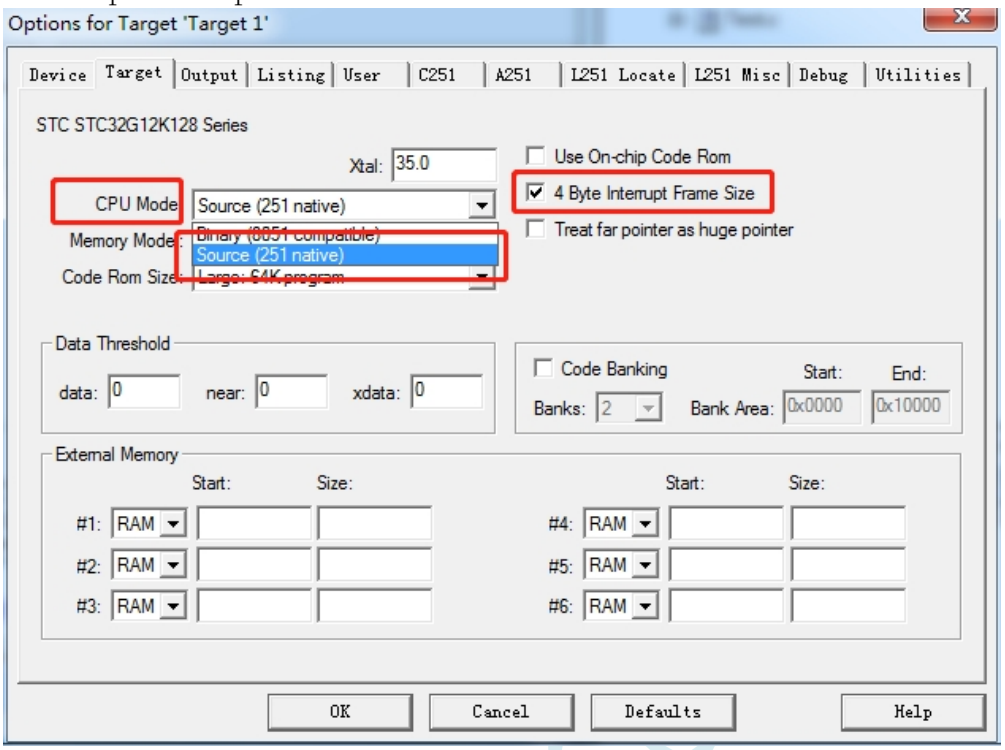


5.3.4 Setting item 1 ("CPU Mode" selects Source mode)

As shown in the figure below, right click on the icon where "Target1" is located and select "Options for Target ' Target 1'...".



In the pop-up window "Options for Target 'Target 1'", select "Target" option page, and in the drop-down option of "CPU Mode", select "Source (251 Native)". Select "Source (251 Native)" in the "CPU Mode" drop-down option.

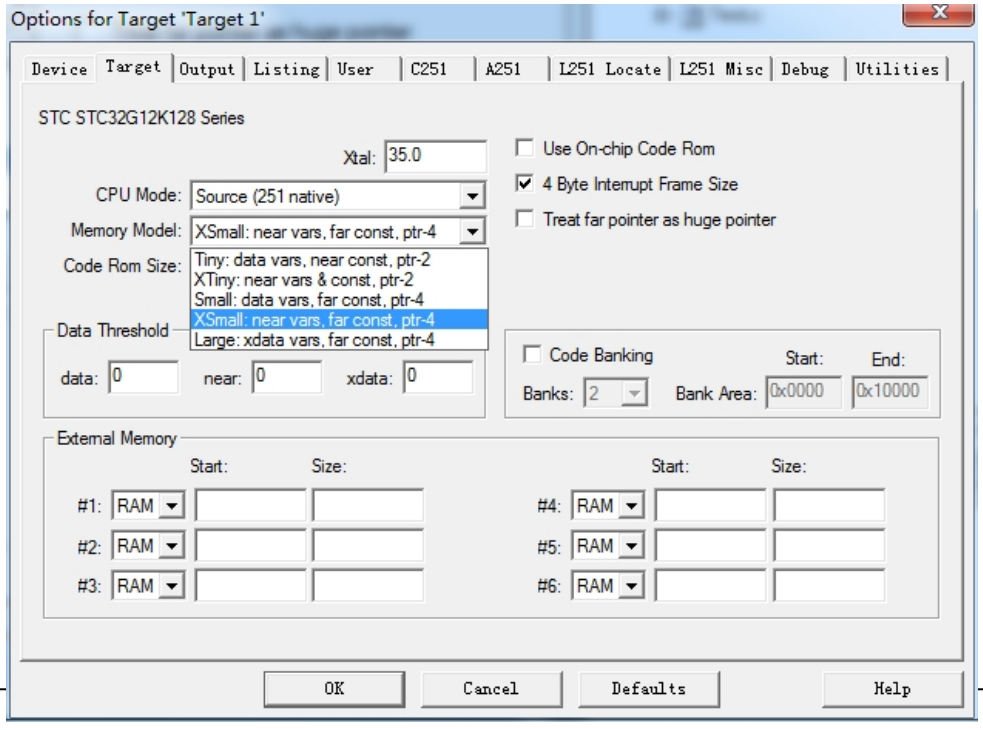


The 80251 instruction modes are "Binary" and "Source", and the STC32G series only supports the "Source" mode.

Since the STC32G series microcontrollers have 4 byte mode for both stacking and stacking out in interrupt, it is recommended that the "4 Byte Interrupt Frame Size" option is also ticked.

5.3.5 Setting item 2 (select XSmall mode for "Memory Model")

Select "XSmall: ..." from the drop-down list of "Memory Model". The 80251 has five memory modes as shown below in the Keil environment:



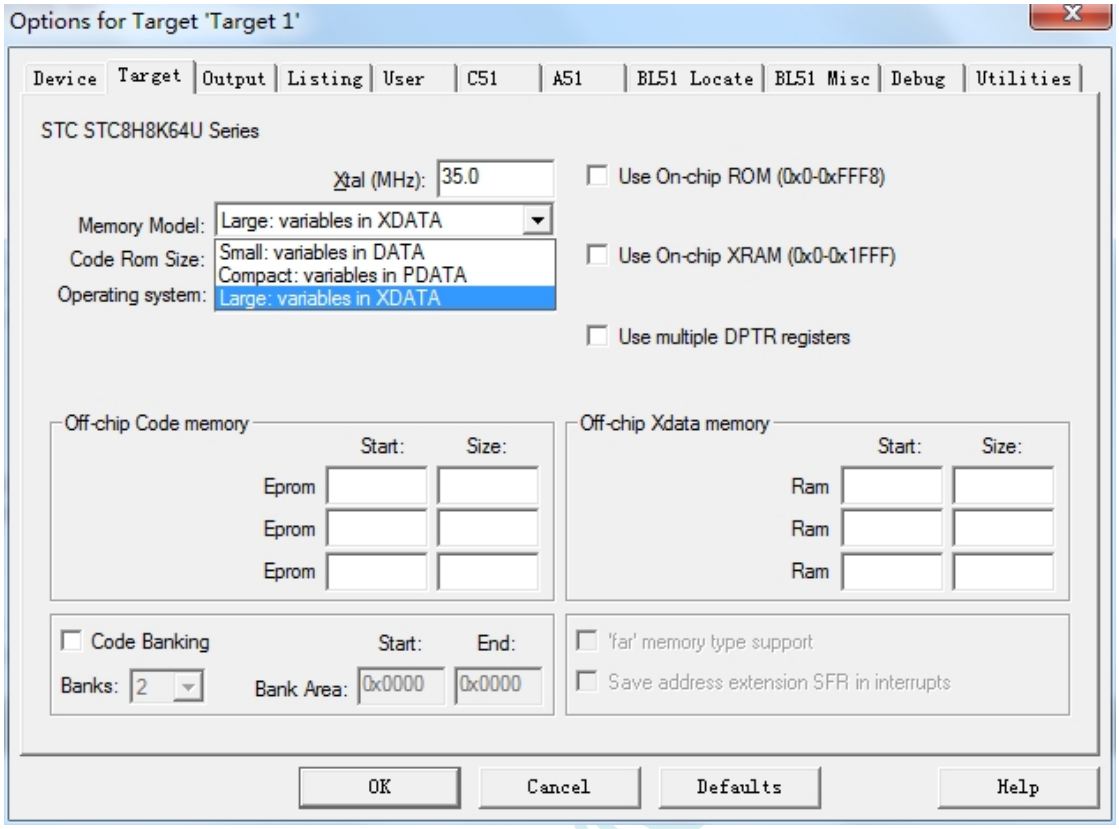
The various models are compared in the table below:

Memory Model	Default variable type (data memory)	Default Constant Type (programme memory)	Default Pointer Variable	Pointer access range
Tiny mode	data	near	2 bytes	00:0000 ~ 00:FFFF
XTiny Mode	edata	near	2 bytes	00:0000 ~ 00:FFFF
Small mode	data	far	4 bytes	00:0000 ~ FF:FFFF
XSmall Mode	**edata**	**far**	**4 bytes**	**00:0000 ~ FF:FFFF**
Large mode	xdata	far	4 bytes	00:0000 ~ FF:FFFF

Since the program logical address of STC32G is from FE:0000H to FF:FFFFH, it needs to use 24-bit address line to access it correctly, the default constant type (programme memory type) must be "far" type, and the default pointer variable must be 4 bytes.

Therefore, it is not recommended to use "Small", "Tiny" and "XTiny" modes, and it is recommended to use "XSmall" mode, which defines variables in internal RAM (edata) by default, with single clock access, fast access speed, and the STC32G12K128 series chips have 12K edata. It is recommended to use "XSmall" mode, which defines the variables in internal RAM (edata) by default, with single clock access, fast access speed, and 12K edata available for STC32G12K128 series chips; when using "Small" mode, it defines the variables in internal RAM (data) by default, with single clock access, fast access speed, and 12K edata available for STC32G12K128 series chips; when using "Small" mode, it defines the variables in internal RAM (data) by default. When using the "Small" mode, the variables are defined in the internal RAM (data) by default, single clock access, fast access speed (data is only 128 bytes by default, when the user's demand for RAM exceeds 128 bytes, the Keil compiler will report an error, and then the user needs to switch the storage mode to XSmall) is limited in the number, and it is easy to report an error, so it is not recommended to use it. "Although this mode can also correctly access all 16M addressing space of STC32G, "Large" mode defines variables in internal extended RAM (xdata) by default, which requires 2~3 clocks to access, and the access speed is slow.

For the corresponding STC8H8K64U series, there are three choices for "Memory Model" in Keil software as follows



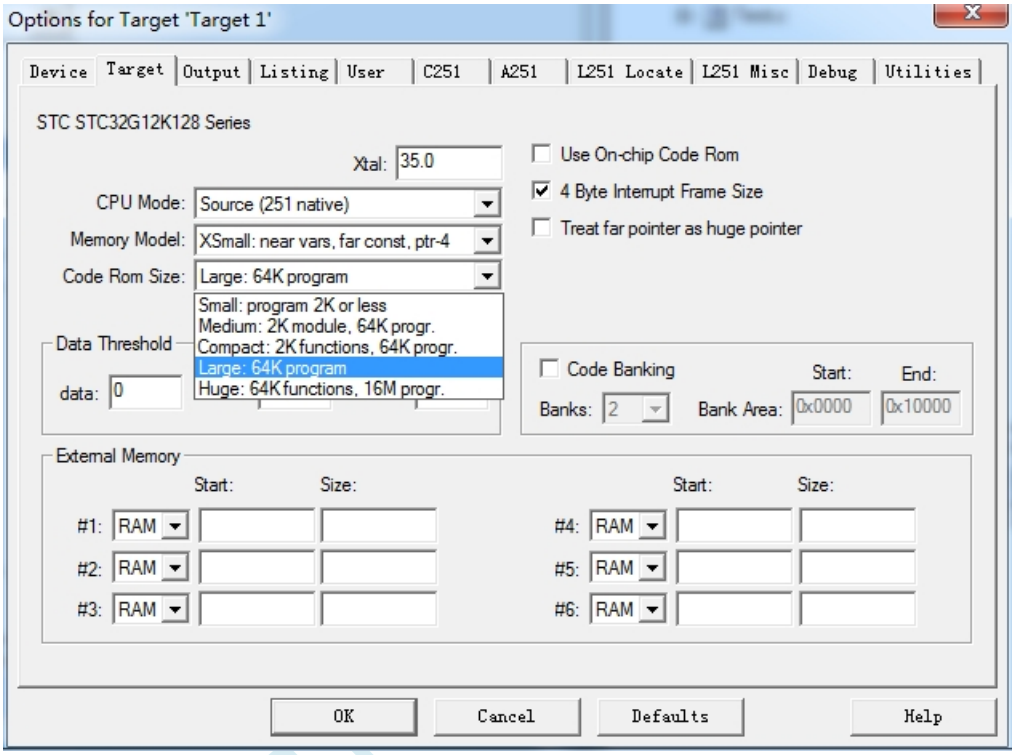
The various models are compared in the table below:

Memory Model	Default variable type (Data memory)	memory size	address range
Small mode	data	128 bytes	D:00 - D:7F
Compact mode	pdata	256 bytes	X:0000 ~ X:00FF
Large mode	**xdata**	**64K bytes (theoretical)**	**X:0000 ～ X:FFFF**

In order to achieve higher efficiency, it is generally recommended to choose "Small" mode, when the compiler shows the error "error C249: 'DATA': SEGMENT TOO LARGE", you need to manually assign some of the larger arrays to the XDATA area by using "xdata" (e.g. char xdata buffer[256];). When the compiler gets "error C249: 'DATA': SEGMENT TOO LARGE", then you need to manually assign some of the larger arrays to the XDATA area via "xdata" (e.g. char xdata buffer[256];).

5.3.6 Setting item 3 (select Large or Huge mode for "Code Rom Size")

Select "Large: ..." in the "Code Rom Size" dropdown. or "Huge: ...".
There are 5 modes of code size for 80251 under Keil environment as



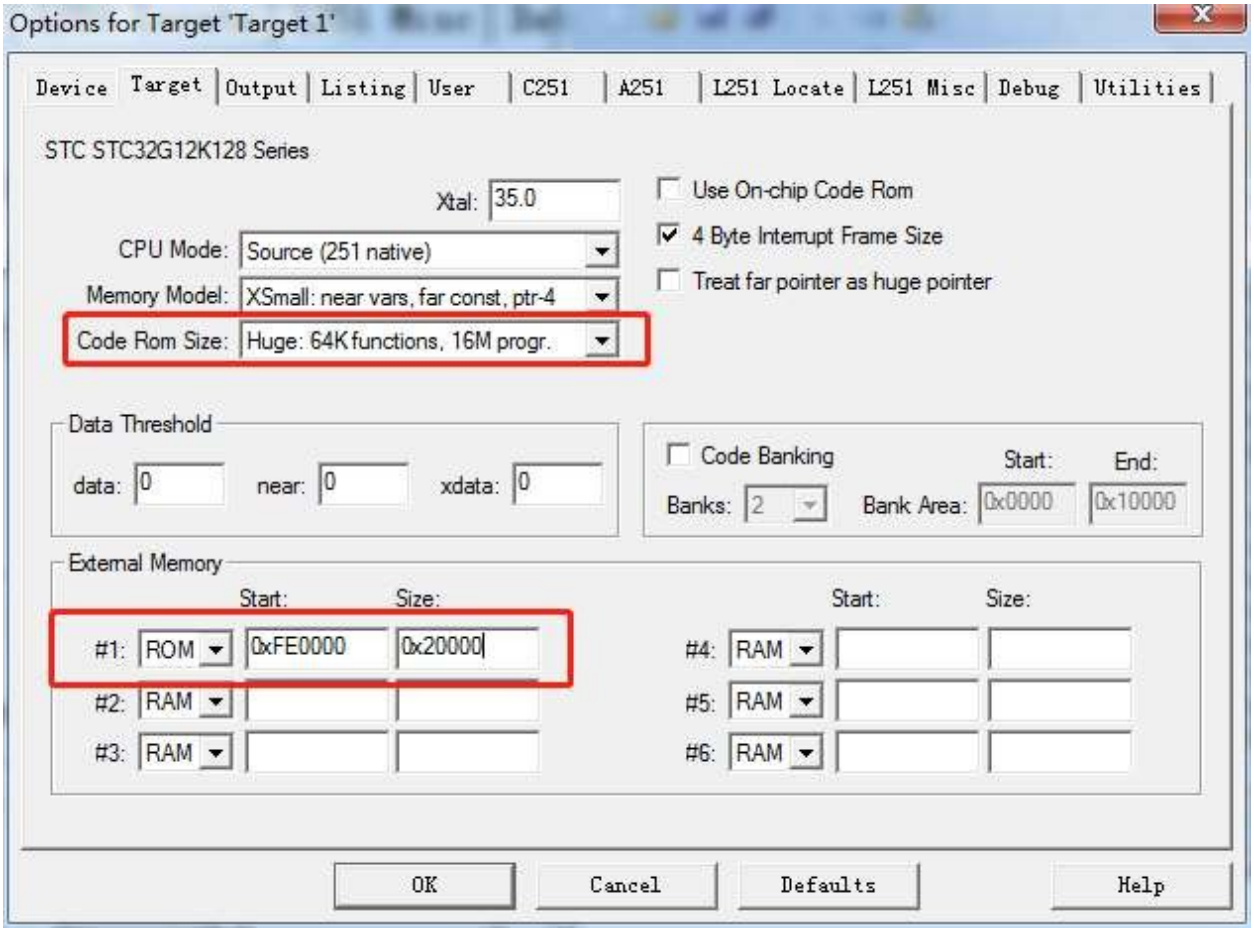
shown in the figure below:

The various models are compared in the table below:

Code Rom Size	Jump/call commands	Code Size Limit	
		Individual functions/modules/ documents The code size of the	Total Code Size
Small mode	AJMP/ACALL	2K	2K
Medium mode	Internal module code using AJMP/ACALL External module code using LJMP/LCALL	2K	64K
Compact mode	LCALL/AJMP	2K	64K
Large mode	LCALL/LJMP	64K	64K
Huge mode	Internal module code using LJMP/ECALL External module code using EJMP/ECALL (In a multi-file project, the code inside the file is an internal module) (block code, code in other files is external module code)	64K	16M

5.3.7 Setting item 4 (Settings related to over 64K codes)

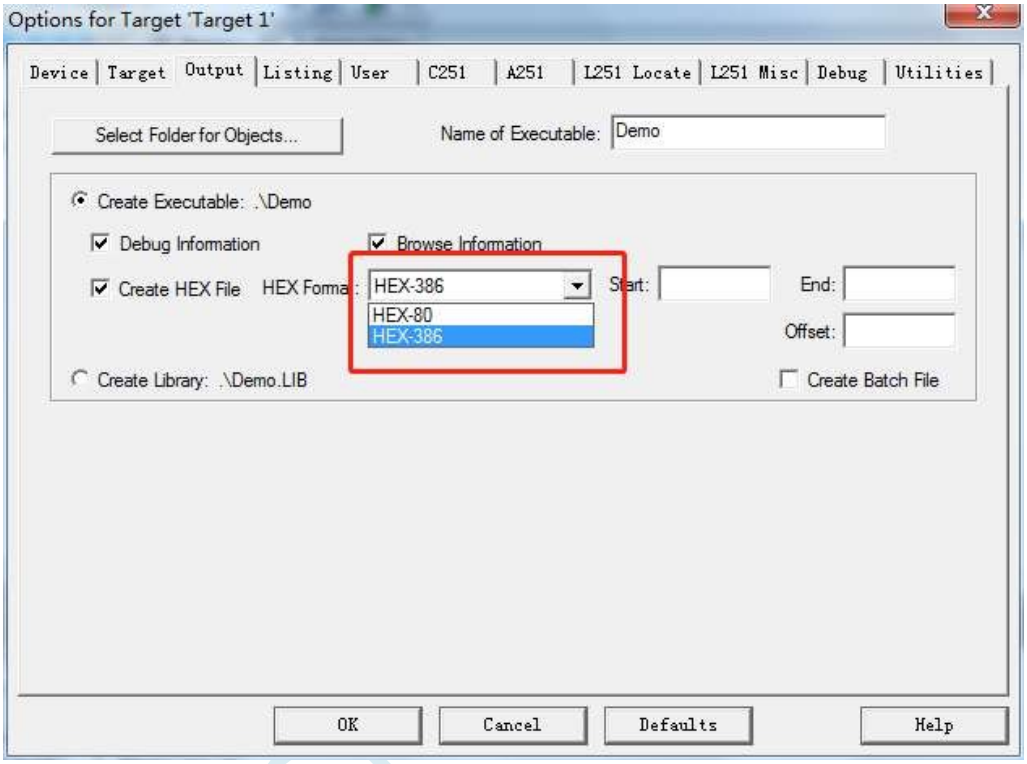
If the code size is within 64K, select "Large" mode. If the code size is more than 64K, then you need to choose "Huge" mode, and you need to ensure that the code size of a single function and a single file must be within 64K bytes, and the amount of data in a single form must also be within 64K bytes. You also need to make the settings as shown in the figure below:



Be sure to note that the setting is for the ROM area!

5.3.8 Setting item 5 (HEX file format setting)

In the ''Options for Target ' Target 1'" window, select the "Output" option page and tick the "Create HEX File" option. Check the "Create HEX File" option in the "Output" tab. If the programme space is more than 64K, ''HEX format'' must select ''HEX-386'' mode, only if the programme space is within 64K, ''HEX format" can select "HEX-386" mode. HEX format" can only select "HEX-80" mode if the programme space is within 64K;



After completing the above settings, mouse click the Compile button shown below, if the code has no errors, you can generate HEX files

5.4 Writing Assembly Code in Keil Based on STC32G Series

5.4.1 Assembler methodology for code sizes up to 64K

```
P0      DATA    080H
P0M1    DATA    093H
P0M0    P0M0    094H
WTST    WTST    0E9H

        ORG     0000H                   Reset Entry Address
        JMP     JMP                     1. 64K program size code, can directly use 0000H to define the
address.

                                        ;   The compiler will automatically concatenate the code to
start at FF:0000
                                        The JMP statement can be used at interrupt vectors, and the
compiler will automatically
                                        ;   Intelligent replacement to AJMP/LJMP/EJMP according to the
actual compilation situation

        ORG     0003H                   Interrupt
        entry address JMP               INT0_ISR
        ORG     000BH
        JMP     TIMER0_ISR
        ORG     0013H
        JMP     INT1_ISR
        ORG     001BH
        JMP     TIMER1_ISR

        ORG
        0200H NOP

INT0_ISR.                               ;Interrupt function
        NOP
        NOP
        RETI                            ;64K program size interrupts use RETI return

TIMER0_ISR
        NOP
        NOP
        RETI

INT1_ISR
        NOP
        NOP
        RETI

TIMER1_ISR
        NOP
        NOP
        RETI

RESET.
```

```
        MOV     spx,#0100h wtst,#0      ;Set the initial value
        MOV                             of the stack pointer
        MOV     P0M0,#0                 ;System
                                        initialisation
        MOV     P0M1,#0
```

```
MAINLOOP.
        INC         P0
        LCALL       DELAY                 Functions with a program size of 64K are
                                          called using LCALL or ACALL.
        JMP         MAINLOOP

DELAY:
        MOV         WR0,#5
DELAY1.
        DEC         WR0,#1
        JNE         DELAY1
        RET                               ;64K program size functions use RET return

        END
```

5.4.2 Assembler methodology for code sizes over 64K

```
p0      DATA    080H
p0m1    DATA    093H
p0m0    DATA    094H
wtst    DATA    0E9H
        DATA

        ORG     0FF:0000H                 ;Reset entry address
        JMP     RESET                     Code exceeding 64K programme size.
                                          ORG  ORG addressing must be done using
                                          0xx:xxxx
                                          The JMP statement can be used at interrupt
                                          vectors, and the compiler will automatically
                                          ;    Intelligent replacement to AJMP/LJMP/EJMP
                                          according to the actual compilation situation
        ORG     0FF:0003H                 ;Interrupt entry address
        JMP     INT0_ISR
        ORG     0FF:000BH
        JMP     TIMER0_ISR
        ORG     0FF:0013H
        JMP     INT1_ISR
        ORG     0FF:001BH
        JMP     TIMER1_ISR

        ORG     0FF:0200H

INT0_ISR.                                 ;Interrupt function
        NOP
        NOP
        RETI                              ; Interrupt return

TIMER0_ISR.
        NOP
        NOP
        RETI

INT1_ISR.
        NOP
        NOP
```

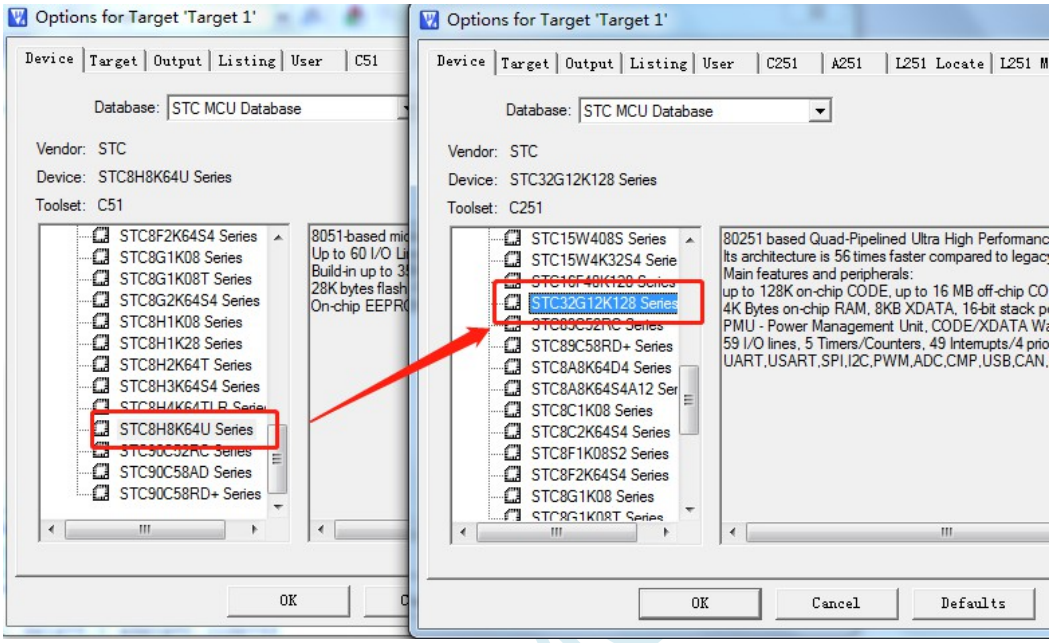
R ETI

TIMER1_ISR.

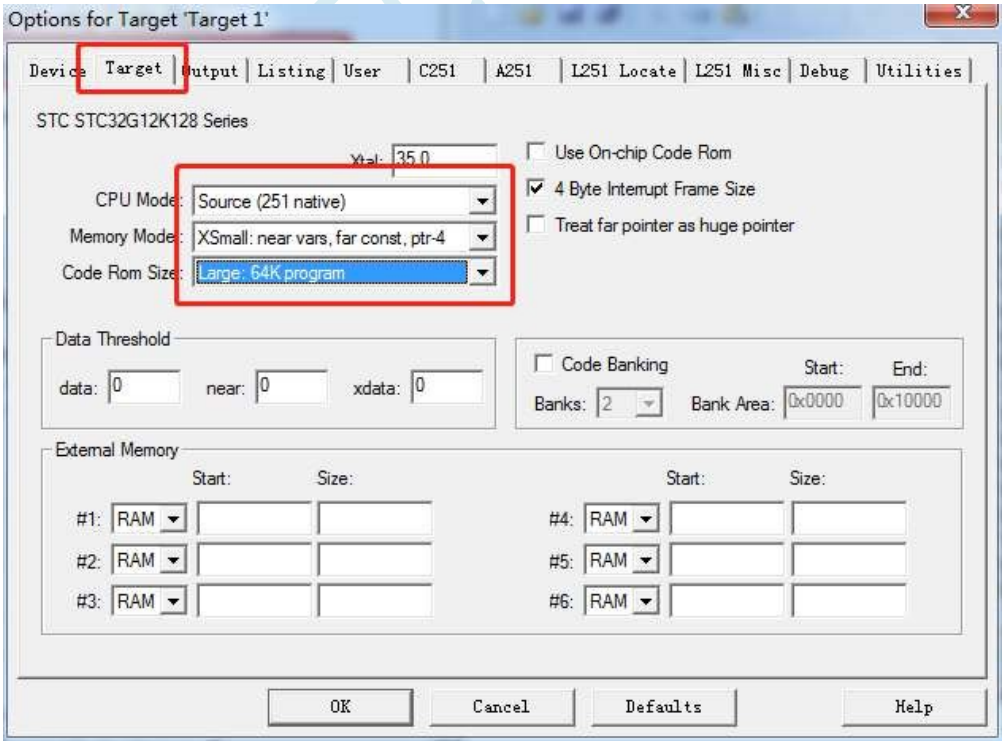
```
        NOP
        NOP
        RETI

RESET.
        MOV     SPX,#0100H      ;Set the initial value of the stack pointer
        MOV     WTST,#0

        MOV     P0M0,#0         ;System initialisation
        MOV     P0M1,#0
MAINLOOP.
        INC     P0
        ECALL   DELAY           Functions larger than 64K program size are called using ECALL.
        JMP     MAINLOOP

        ORG     0FE:0000H

DELAY.
        MOV     WR0,#1000
DELAY1.
        DEC     WR0,#1
        JNE     DELAY1
        ERET                    ;Functions exceeding 64K program size are returned using ERET.

        END
```

5.5 STC8H series project to STC32G series

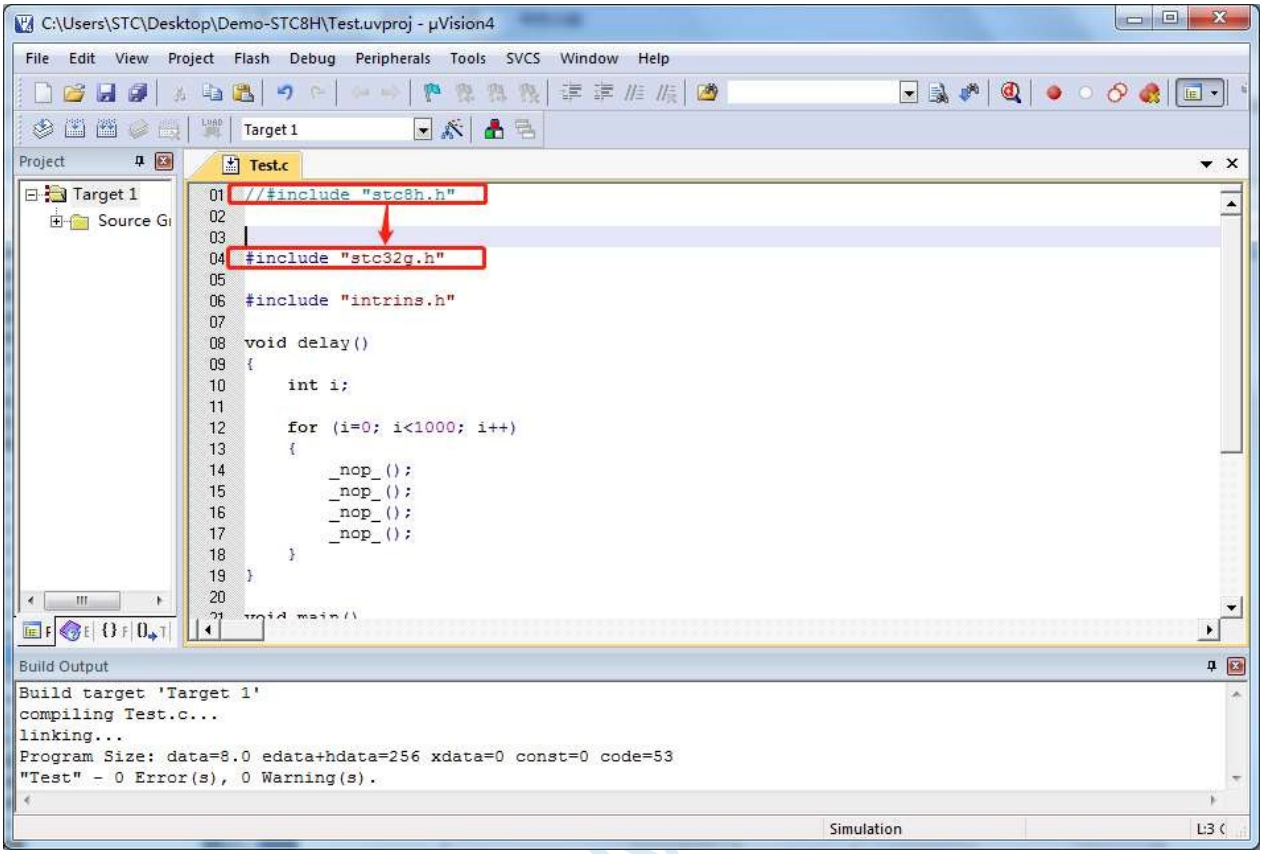
1. Change the target microcontroller model. As shown in the figure below, change the original microcontroller model "STC8H8K64U" to "STC32G12K128" for STC8H project.



2, in the "Options for Target " Target1"" in the "Target" page in accordance with the following figure for the settings



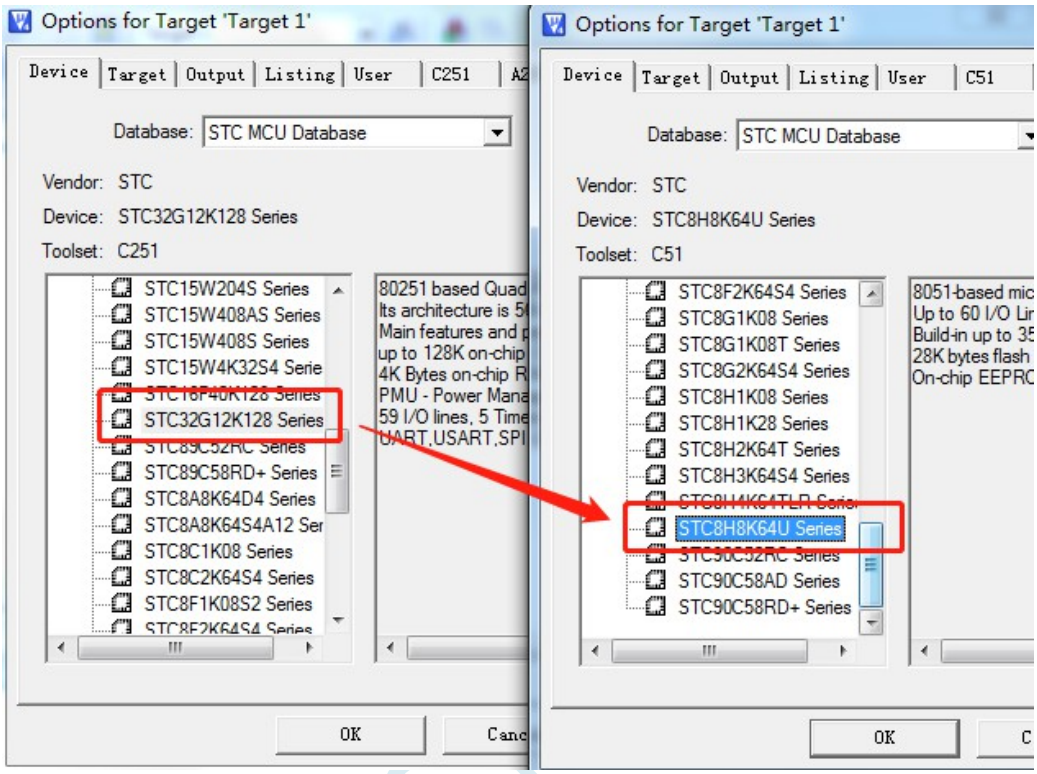
3, replace the header file references. Replace the original "#include "stc8h.h"" with "#include "stc32g.h" "



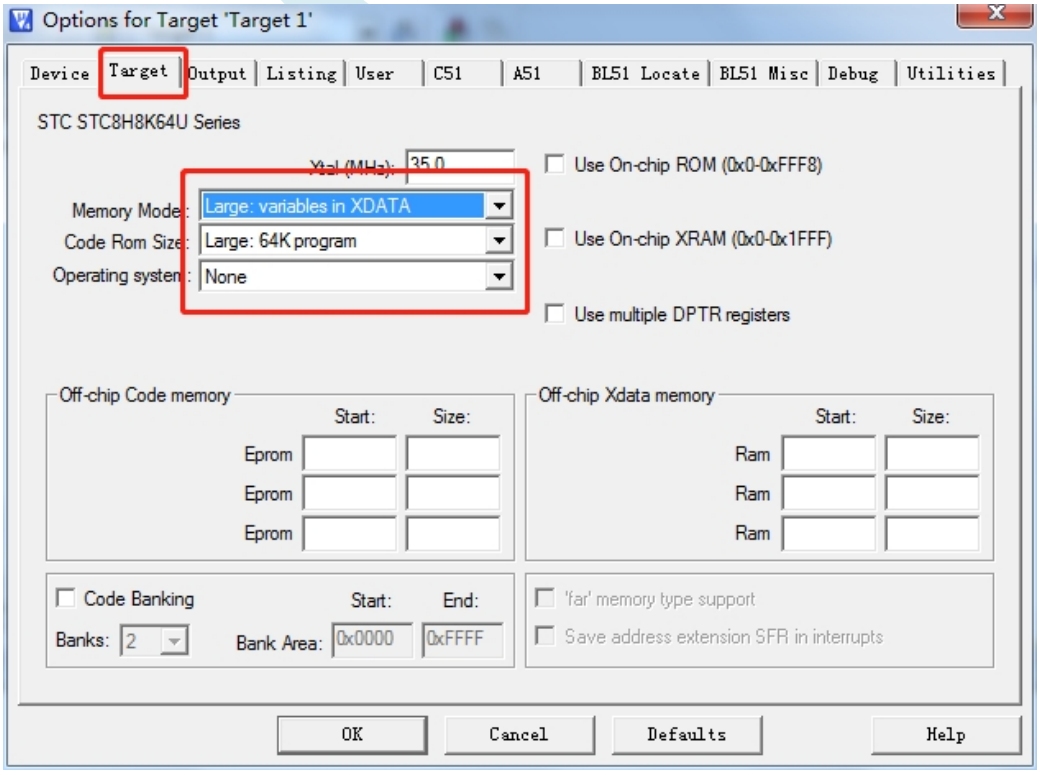
Since most of the SFRs and XFRs of the STC32G and STC8H series are fully compatible, the compilation will basically pass after the above settings. If there are a few incompatibilities, you can just modify them a little bit.

5.6 STC32G series project to STC8H series

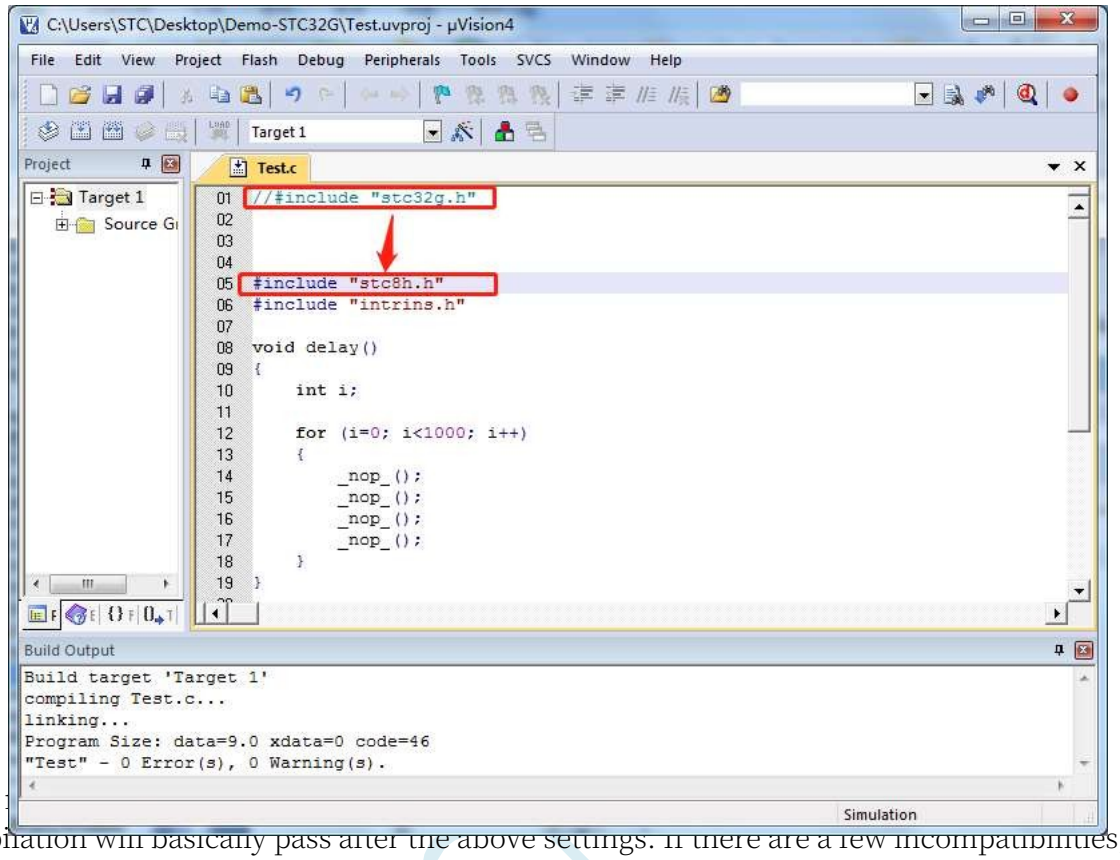
1. Change the target microcontroller model. As shown in the figure below, change the original microcontroller model "STC32G12K128" to "STC8H8K64U" for the STC32G project.



2, in the "Options for Target " Target1"" in the "Target" page in accordance with the following figure for the settings



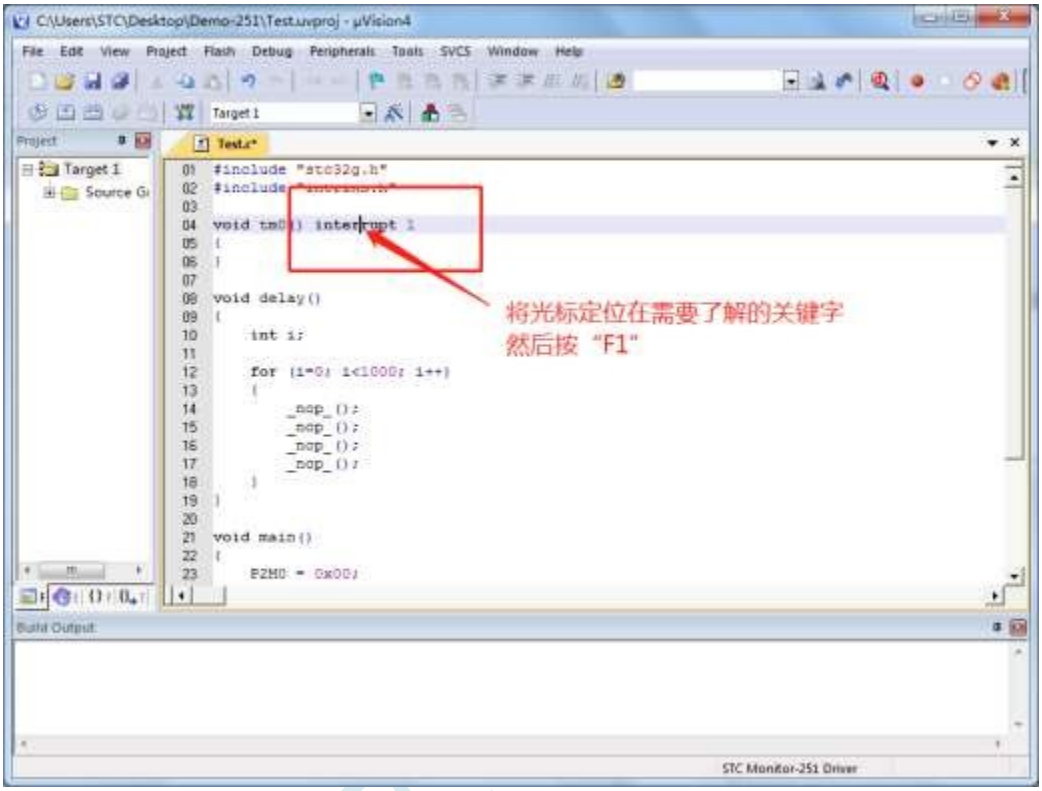
3. Replace the header file references. Replace the original "#include "stc32g.h"" with "#include "stc8h.h" "



Since i
compilation will basically pass after the above settings. If there are a few incompatibilities, you can just modify them a little bit.

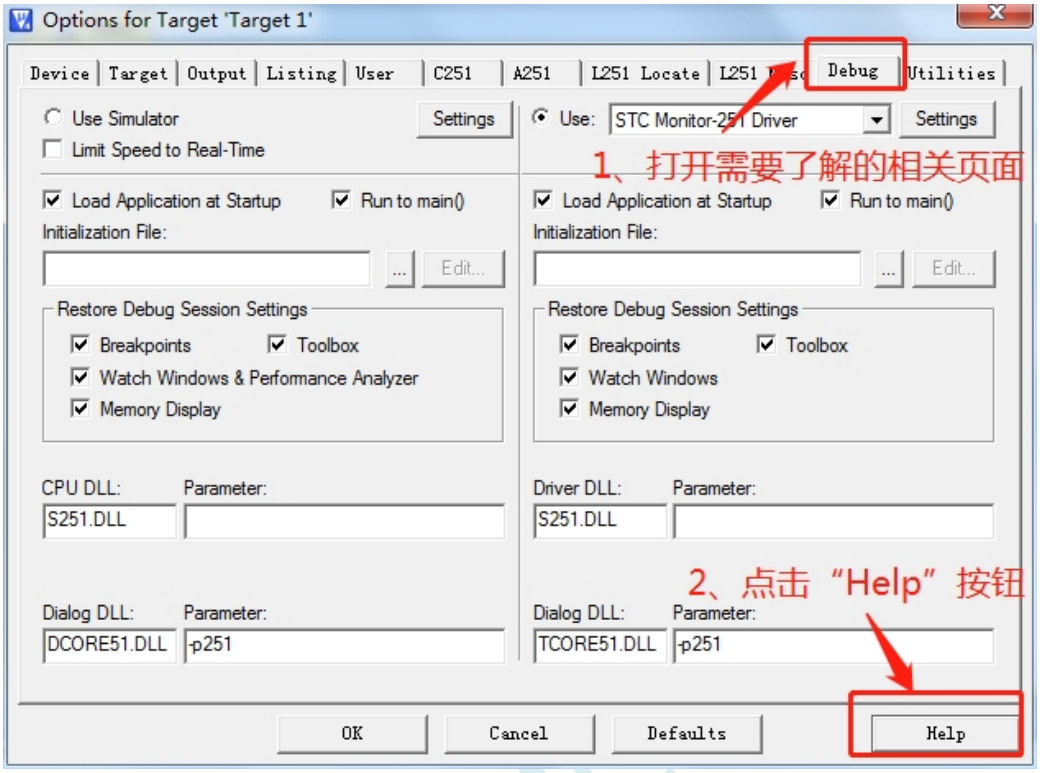
5.7 An easy way to get help in Keil software

Keil software provides a very complete set of help files, for general software use and programming problems, direct use of the Keil software help group can basically be solved. The



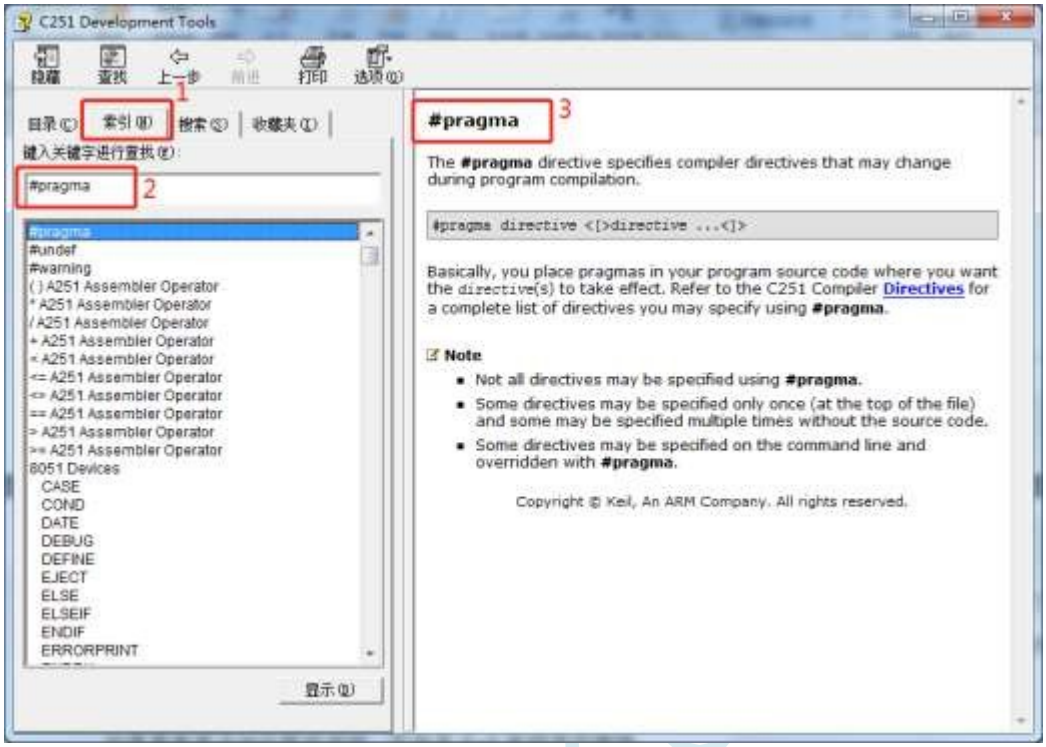
following figure:

If you need to know the relevant settings in the project settings, you can get the help group as shown in



the figure below.

Alternatively, you can enter the information you want to know directly into the help window. For example, if you want to know how to set up special compilation instructions in a



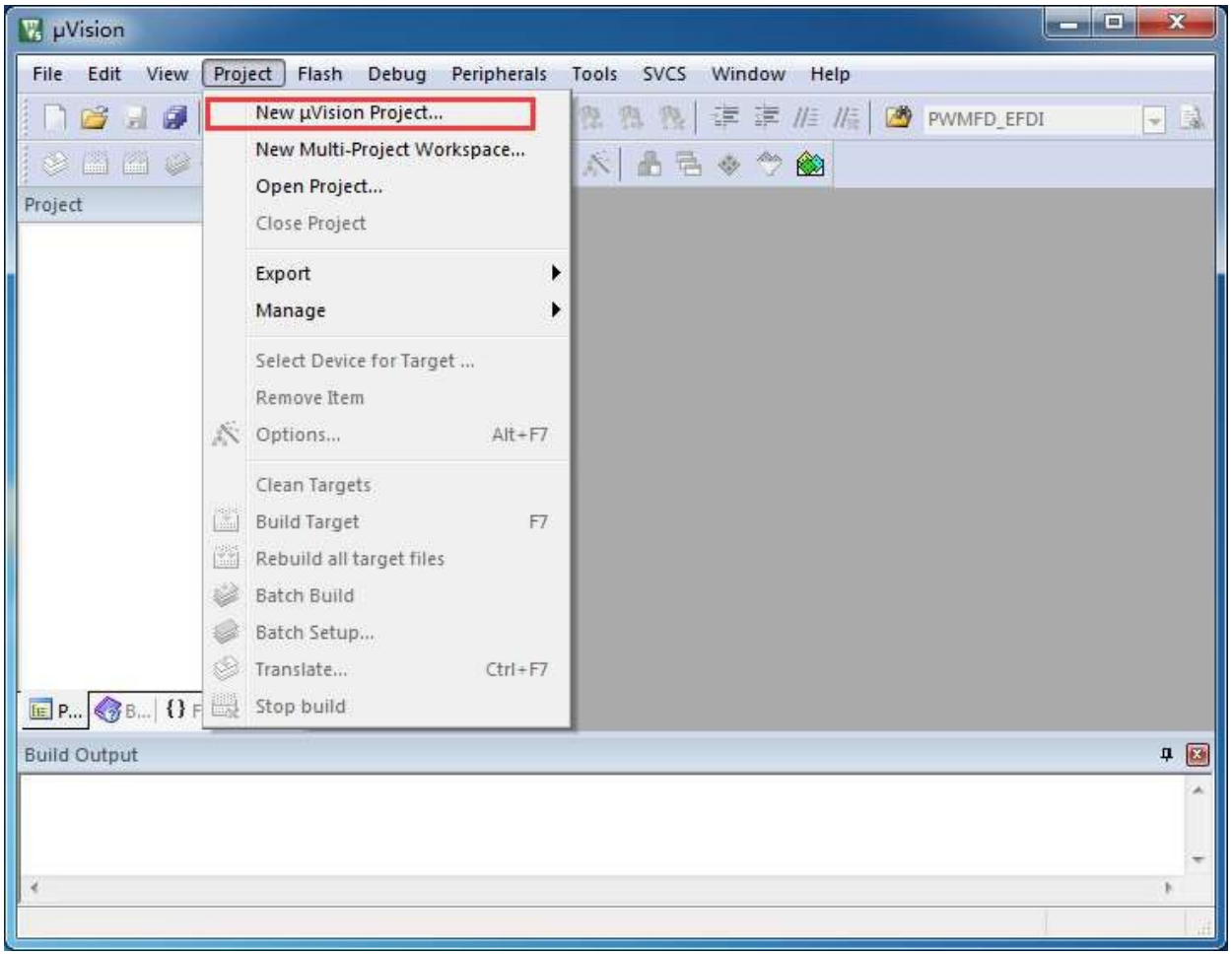
programme, you can type "#pragma" in the search box as shown below.

If you need more detailed help group details, you can visit the Keil official website for

5.8 Creating a Multi-File Project in Keil

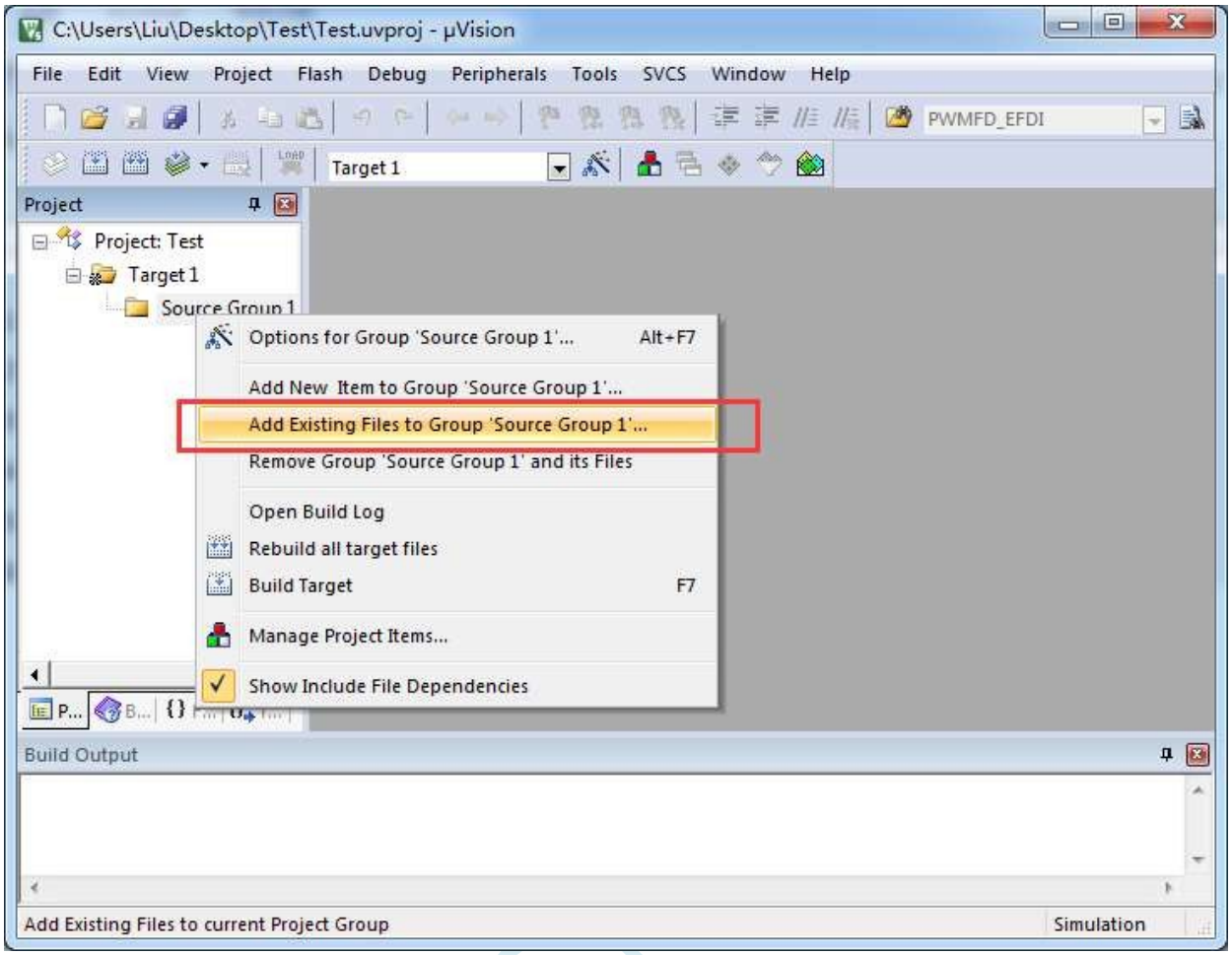
In Keil, generally smaller projects have only one source file, but for some slightly more complex projects often need more than one source file to create a multi-file project as follows:

1, first of all, open Keil, in the menu "Project", select "New uVision Project ...".

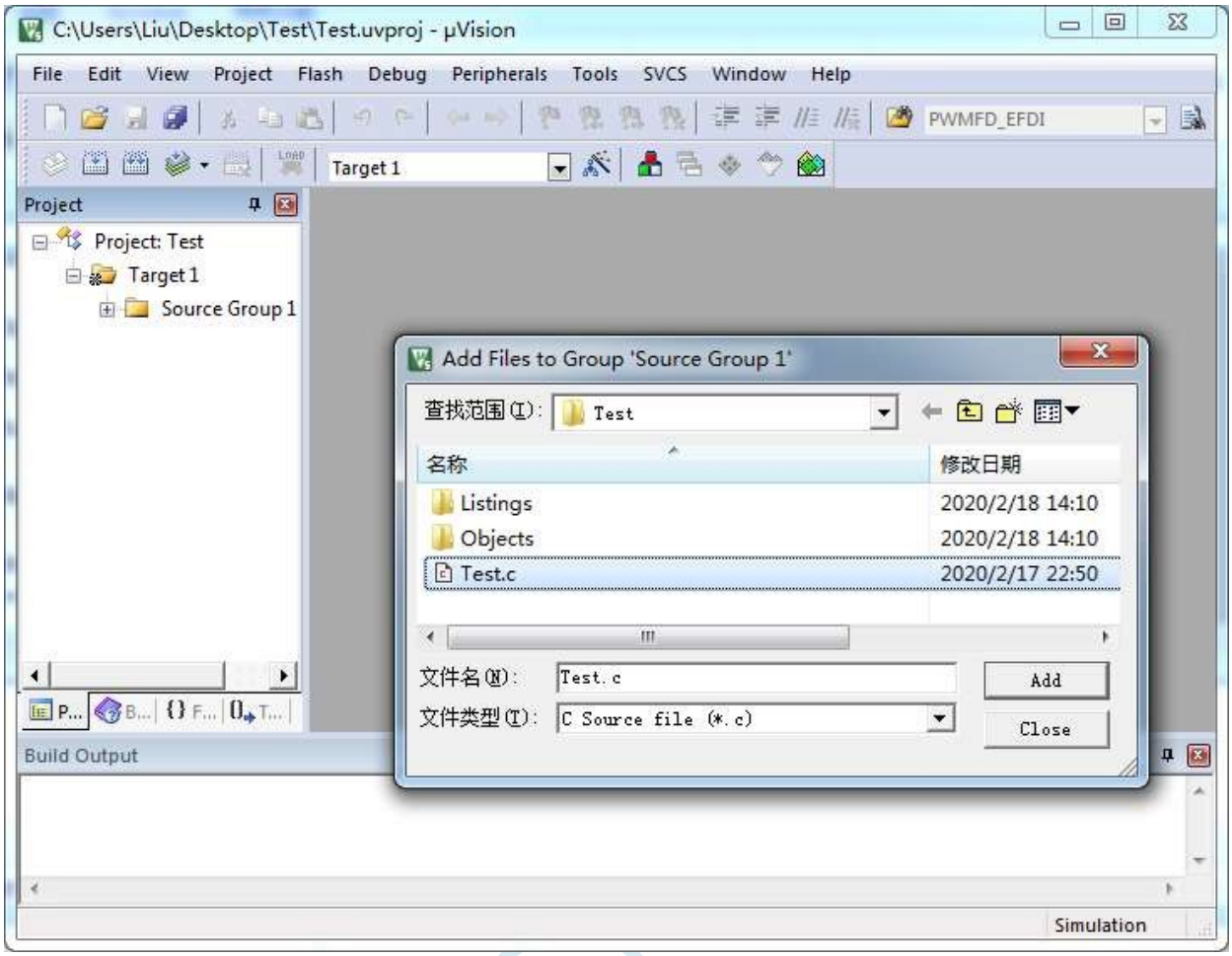


You can create an empty project

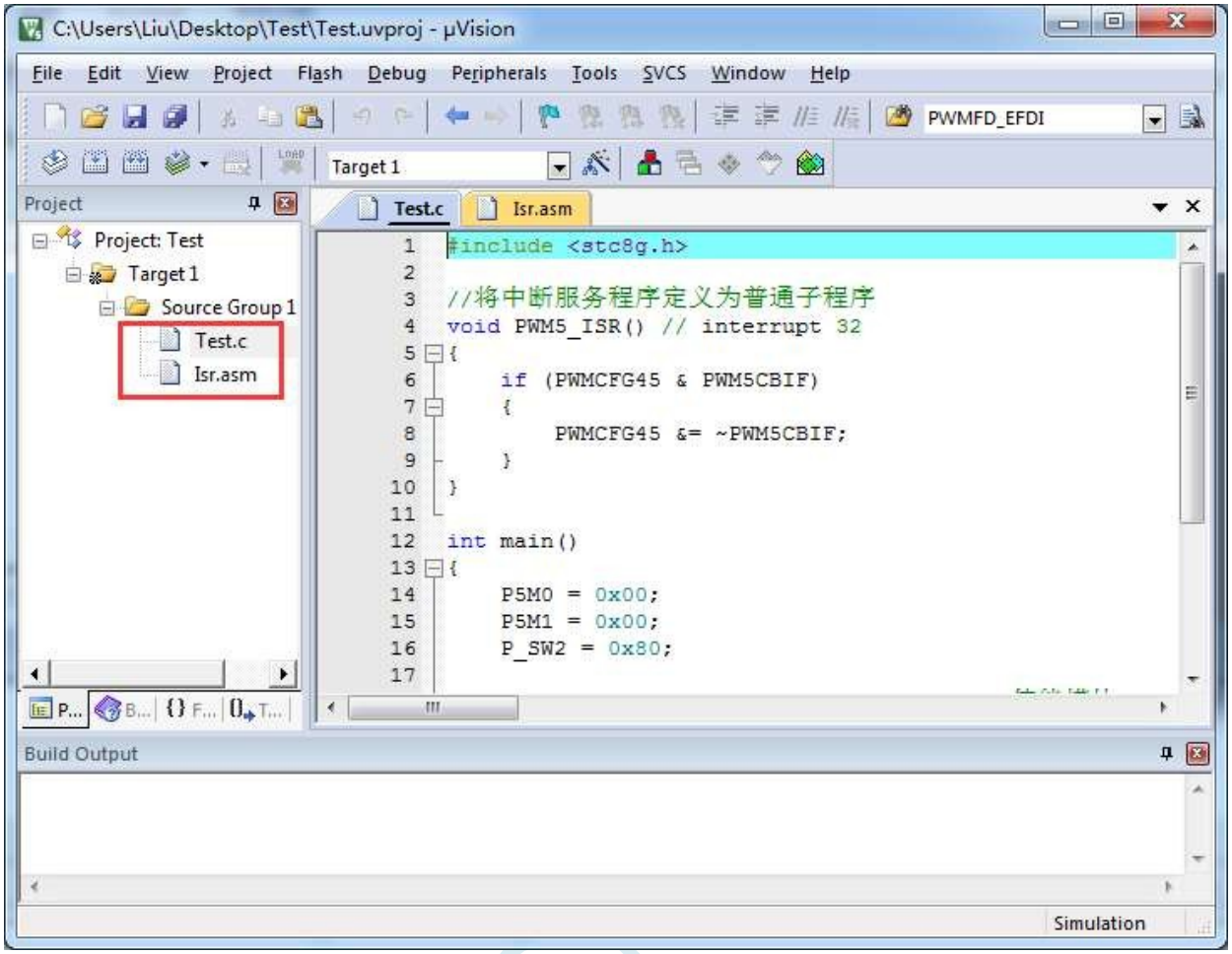
2, in the empty project tree, right-click "Source Group 1", and select the right-click menu in the "Add Existing Files to Group "Source Group 1" ... "



3, in the pop-up file dialogue box, multiple times to add the source file



As shown in the figure below, you can complete the creation of a multi-file project.


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

5.9 Handling of compilation error in Keil with interrupt number greater than 31

Note: At present, all versions of Keil C51 and C251 compilers only support 32 interrupt numbers (0-31). After much negotiation and discussion between our company and Keil, Keil promised to increase our demand for more than 32 interrupt numbers in one of the subsequent versions. However, for the current Keil version, we can only use the methods in this chapter to solve the problem temporarily.

5.9.1 Use the popular online break number expansion tool

The enthusiasts have a simple extension tool to extend the interrupt number to 254. the tool interface is as

follows:

Click the "Open" button to locate the Keil installation directory and click "OK".

Since Keil's version is constantly being updated, and there are too many early versions to collect them all, here is a list of tested C51.EXEs.

version and C251.EXE version

Tested version of C51.EXE:

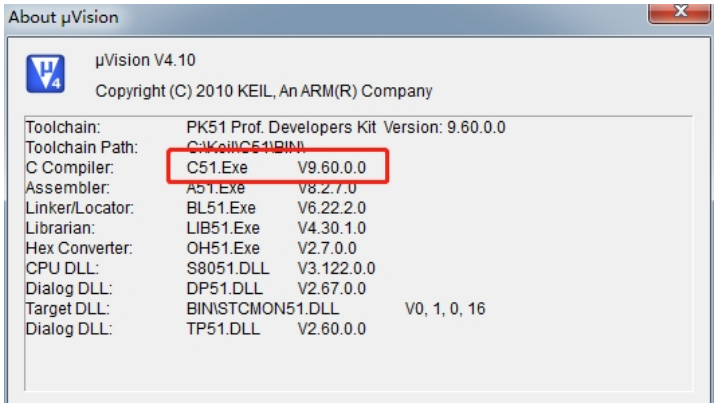
V6.12.0.1
V8.8.0.1
V9.0.0.1
V9.1.0.1
V9.53.0.0
V9.54.0.0
V9.57.0.0
V9.59.0.0
V9.60.0.0

Tested version of C251.EXE:

V5.57.0.0
V5.60.0.0

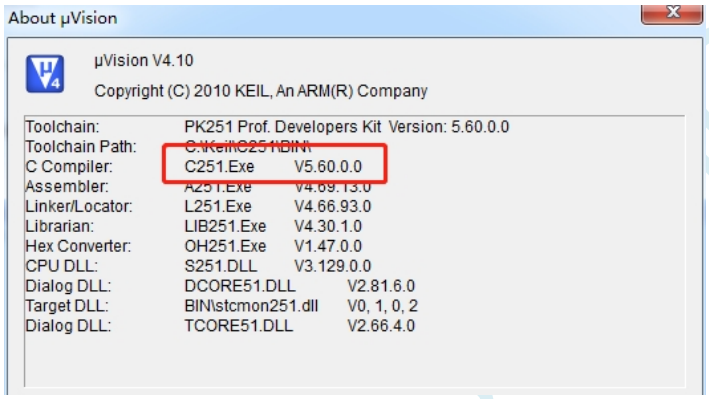
A method to view the version of C51.EXE:

Open a project based on STC8 series or STC15 series microcontroller in keil, and open "About uVision..." in Keil software menu item "Help".



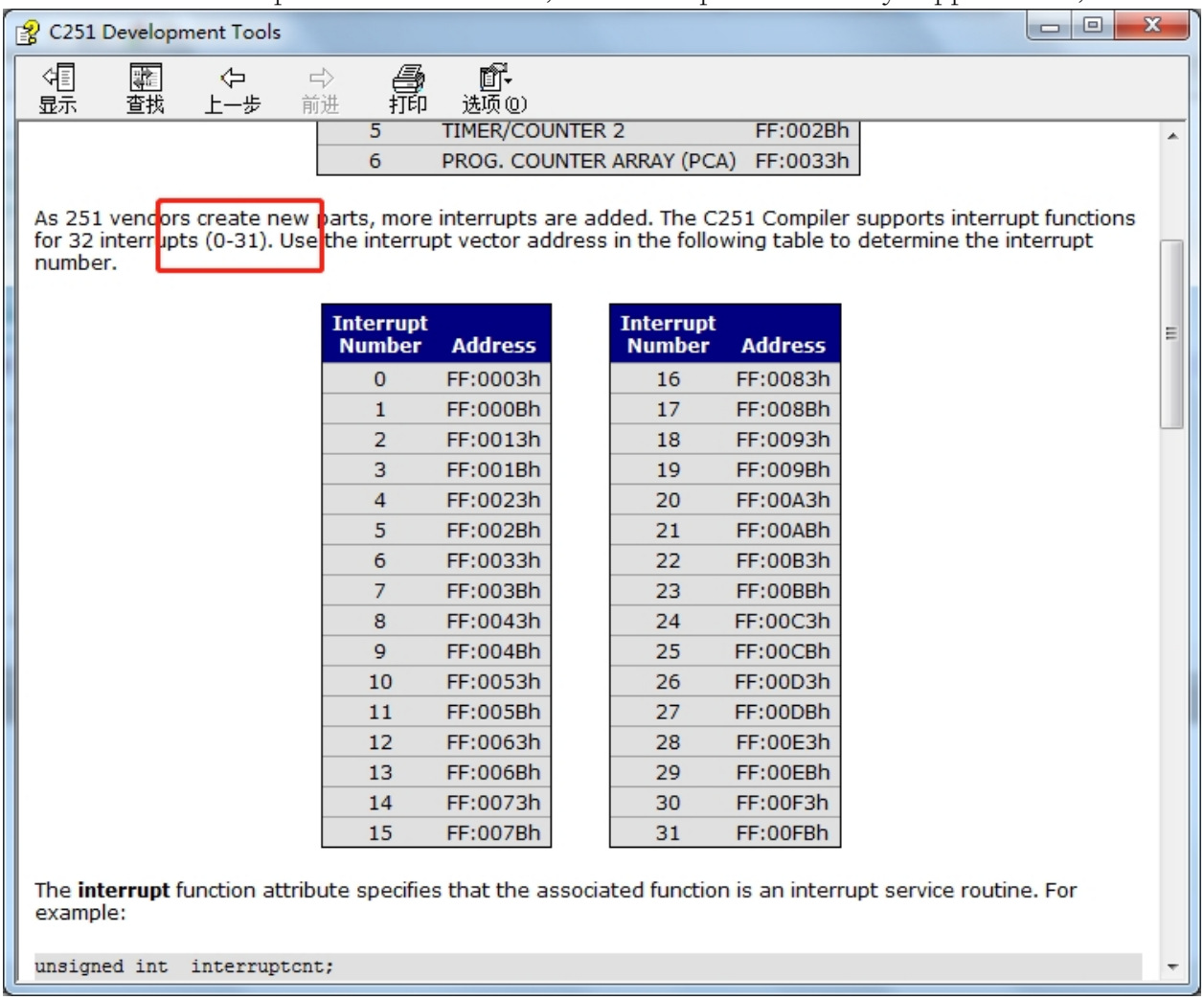
A way to view the version of C251.EXE:

Open a project based on STC32G series microcontroller in keil, open "About uVision..." in Keil software menu item "Help".



5.9.2 Transit using reserved interrupt numbers

Under Keil's C251 compilation environment, the interrupt number only supports 0~31, i.e. the interrupt



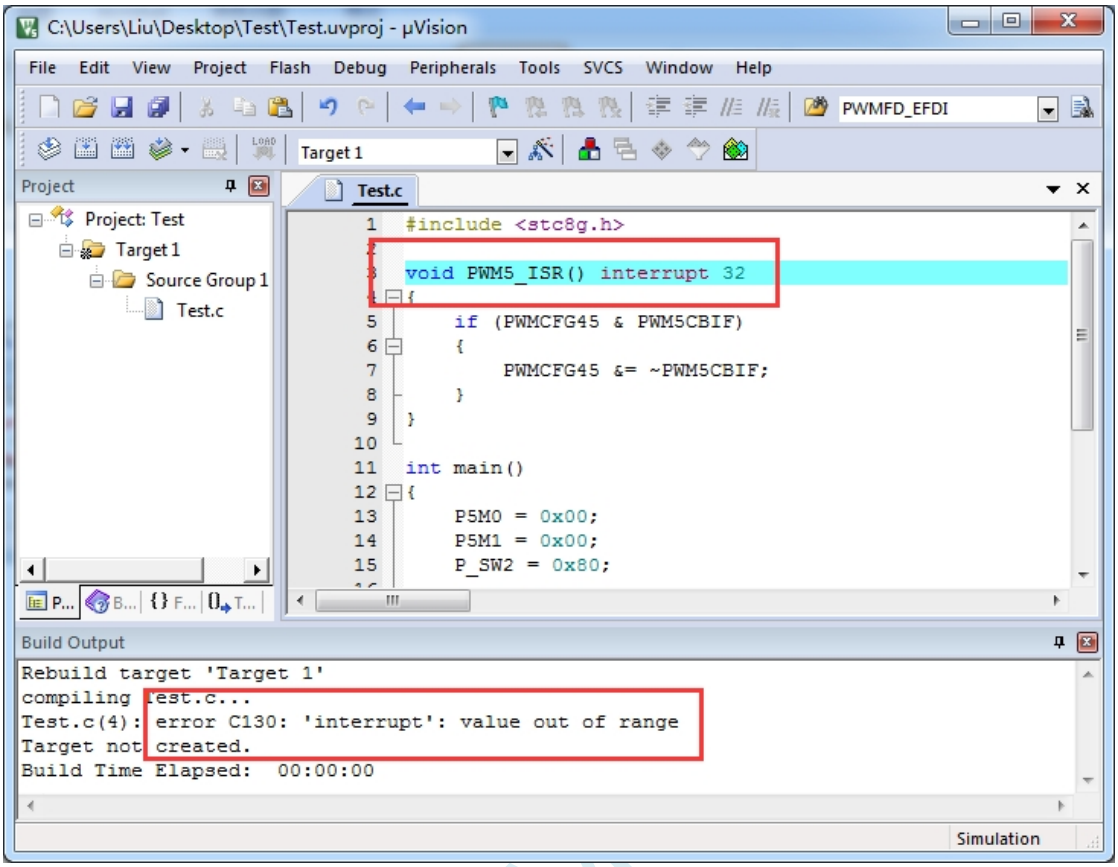
vector must be less than 0100H.

The following table shows the list of interrupts for all current families of STCs:

cut-off number (i.e. punctuation mark)	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	Serial Port 1
5	002B H	ADC
6	0033 H	LVD
8	0043 H	Serial Port 2
9	004B H	SPI
10	0053 H	INT2
11	005B H	INT3
12	0063 H	Timer 2
13	**006B H**	
14	0073 H	Intra-system interruptions

15	007B H	Internal system interrupt
16	0083 H	INT4
17	008B H	Serial port 3
18	0093 H	Serial port 4
19	009B H	Timer 3
20	00A3 H	Timer 4
21	00AB H	comparator
24	00C3 H	I2C
25	00CB H	USB
26	00D3 H	PWMA
27	00DB H	PWMB
28	00E3 H	CAN1
29	00EB H	CAN2
30	00F3 H	LIN
36	**0123 H**	**RTC**
37	**012B H**	**P0 port interrupt**
38	**0133 H**	**P1 port interrupt**
39	**013B H**	**P2 port interrupt**
40	**0143 H**	**P3 port interrupt**
41	**014B H**	**P4 port interrupt**
42	**0153 H**	**P5 port interrupt**
43	**015B H**	**P6 port interrupt**
44	**0163 H**	**P7 port interrupt**
45	**016B H**	**P8 port interrupt**
46	**0173 H**	**P9 port interrupt**
47	**017BH**	**M2M DMA Interrupt**
48	**0183H**	**ADC DMA Interrupt**
49	**018BH**	**SPI DMA Interrupt**
50	**0193H**	**UR1T DMA Interrupt**
51	**019BH**	**UR1R DMA Interrupt**
52	**01A3H**	**UR2T DMA Interrupt**
53	**01ABH**	**UR2R DMA Interrupt**
54	**01B3H**	**UR3T DMA Interrupt**
55	**01BBH**	**UR3R DMA Interrupt**
56	**01C3H**	**UR4T DMA Interrupt**
57	**01CBH**	**UR4R DMA Interrupt**
58	**01D3H**	**LCM DMA Interrupt**
59	**01DBH**	**LCM Interrupt**
60	**01E3H**	**I2CT DMA Interrupt**
61	**01EBH**	**I2CR DMA Interrupt**
62	**01F3H**	**I2S Interrupt**
63	**01FBH**	**I2ST DMA Interrupt**
64	**0203H**	**I2SR DMA Interrupt**

It is not difficult to find that the RTC interrupt starts, and all the interrupt service programmes that follow will compile with errors in keil, as shown in the following figure:

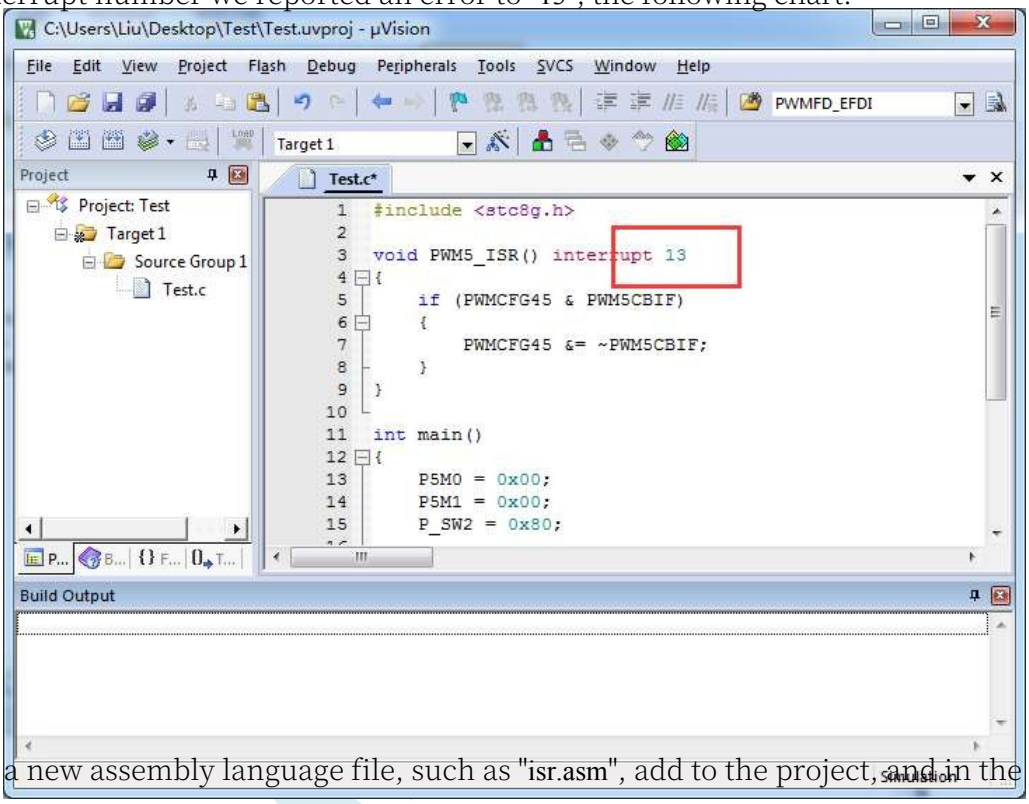


There are three ways to deal with this error: (all require the use of assembly code; method 1 is preferred)

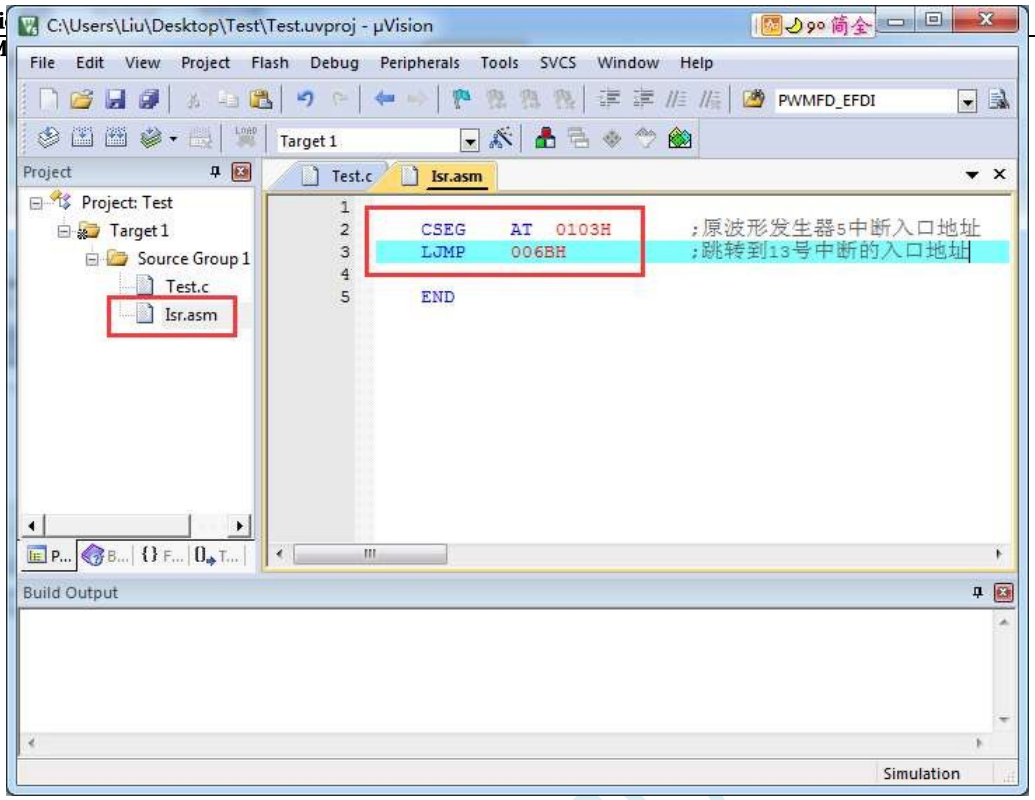
Method 1: Borrow Interrupt Vector 13

Among the interrupts 0~31, No.13 is a reserved interrupt number, we can borrow this interrupt number to operate as follows:

1, the interrupt number we reported an error to "13", the following chart:

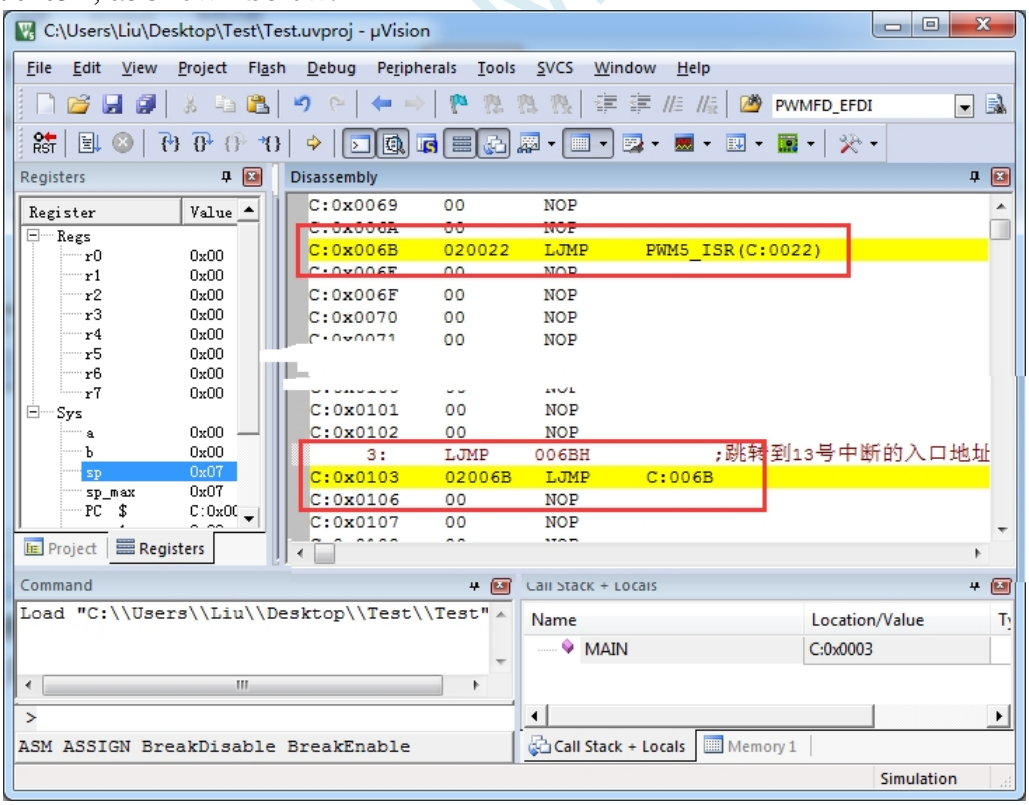


2, create a new assembly language file, such as "isr.asm", add to the project, and in the address "0103H" place to add a "LJMP 006BH", as shown below:

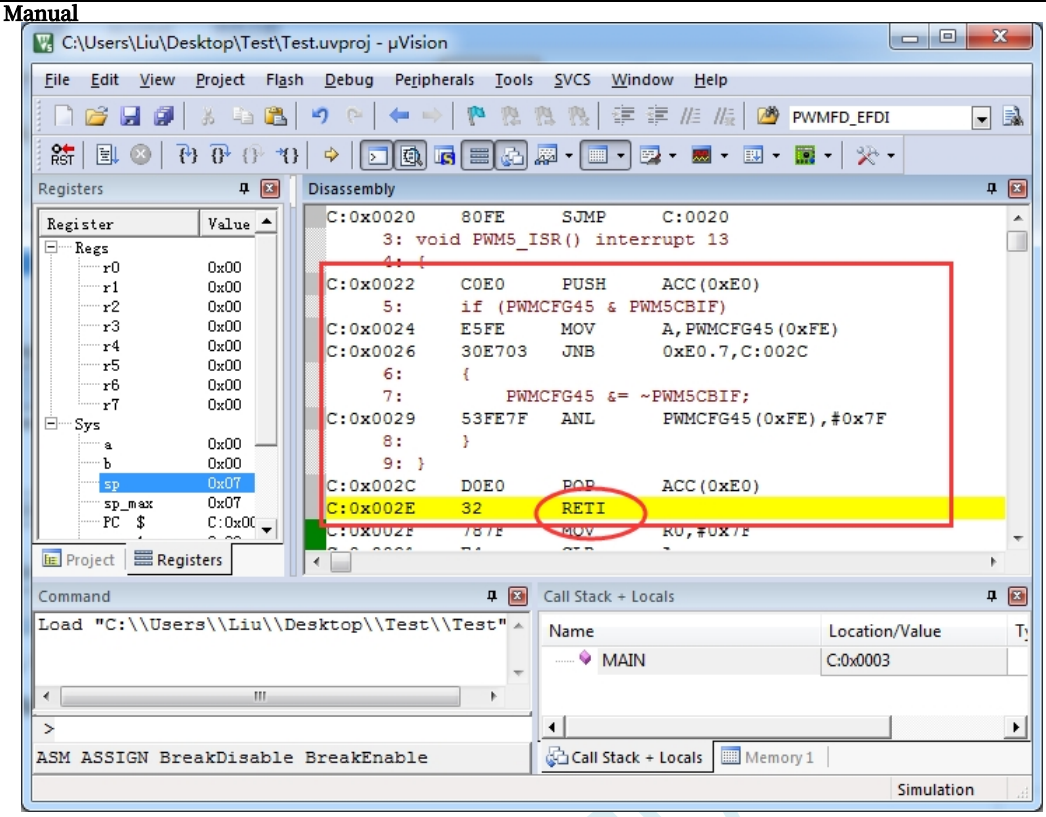


3. Compile to pass.

After Keil's C51 compiler compiles the code, there is a line at 006BH "LJMP PWM5_ISR" at 006BH and "LJMP 006BH" at 006BH and "LJMP 006BH" at 0103H, as shown below:



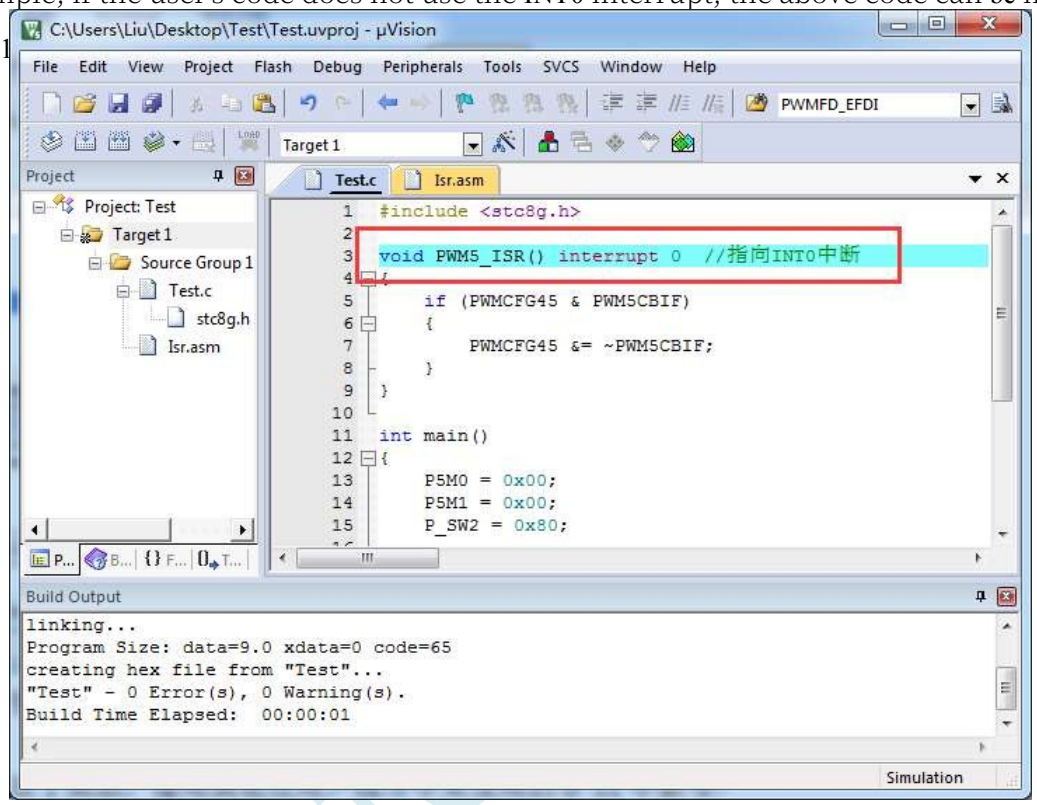
When PWM5 interrupt occurs, the hardware will automatically jump to address 0103H to execute "LJMP 006BH", and then execute "LJMP Then execute "LJMP 006BH" at 006BH and then "PWM5_ISR" at 006BH to jump to the real interrupt service programme, as shown below:

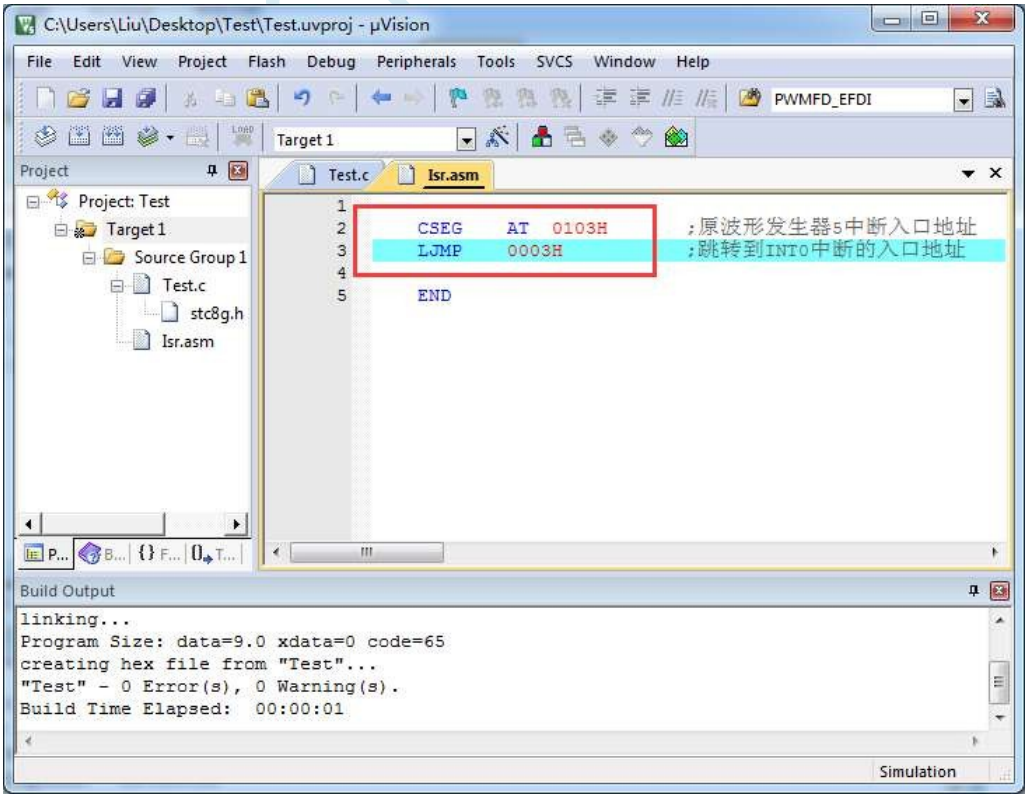


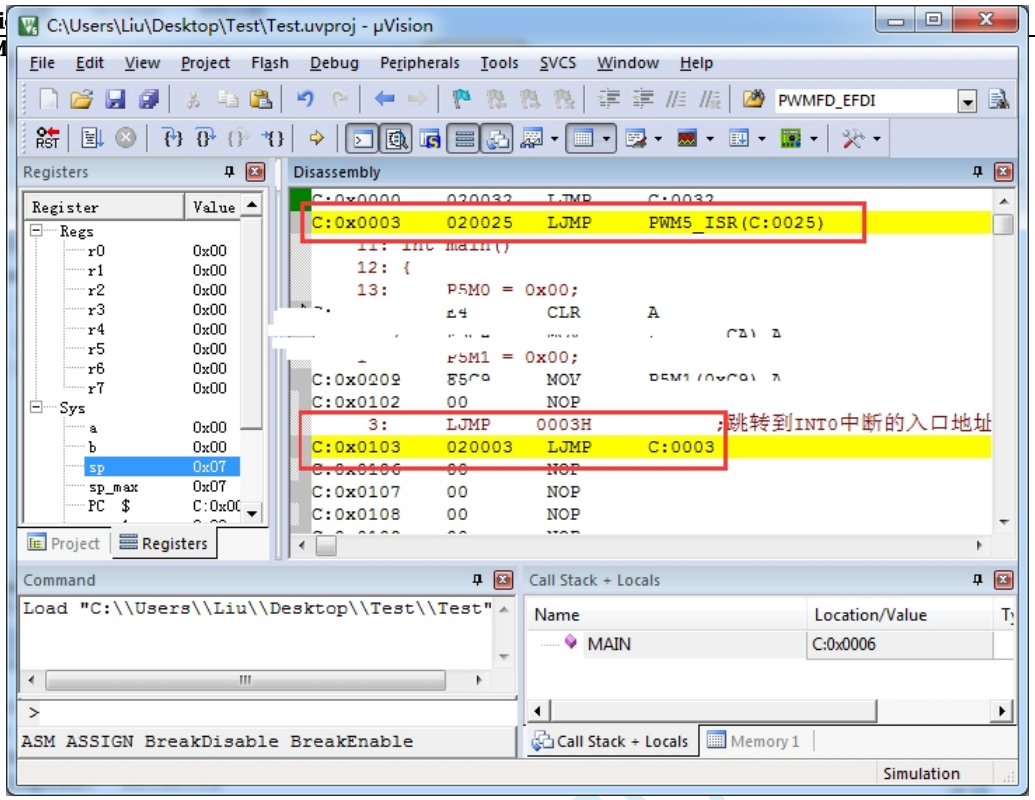
After the interrupt service routine is executed, it is then returned by the RETI instruction. The entire interrupt response process is just one more LJMP statement executed.

Method 2: Similar to method 1, borrow the unused interrupt numbers 0~31 from the user program.

For example, if the user's code does not use the INT0 interrupt, the above code can be modified similarly to method 1





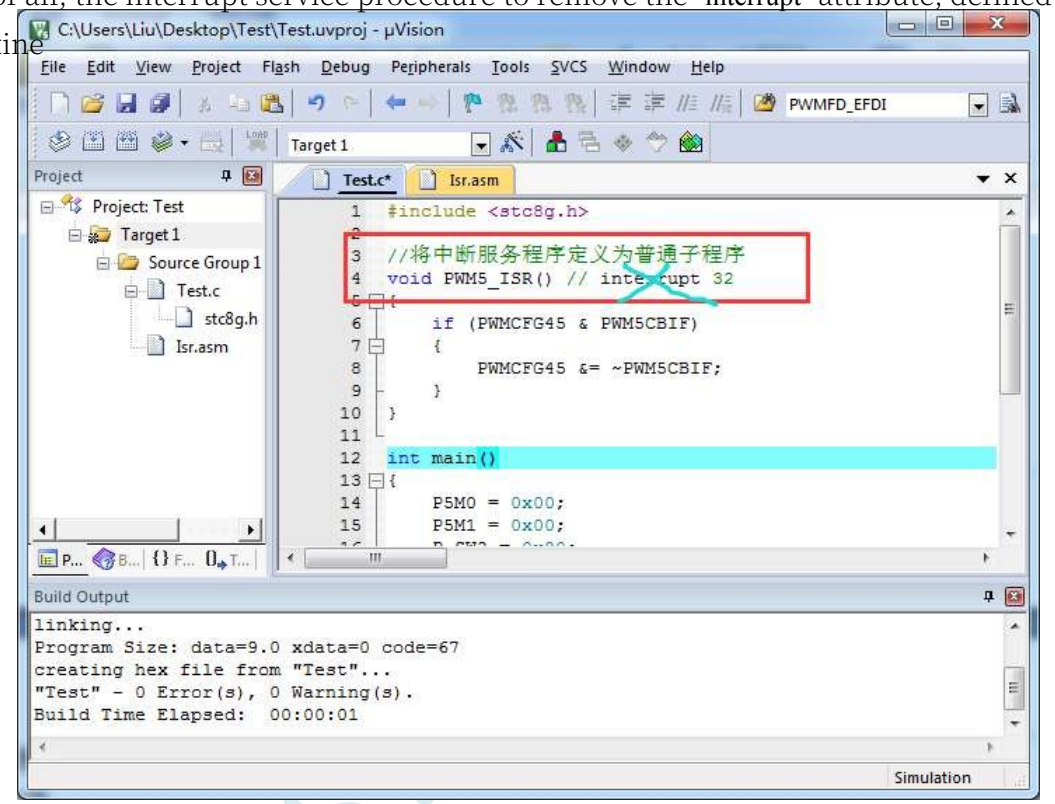


The implementation effect is the same as method 1. This method is applicable to the case where you need to remap more than one interrupt number greater than 31.

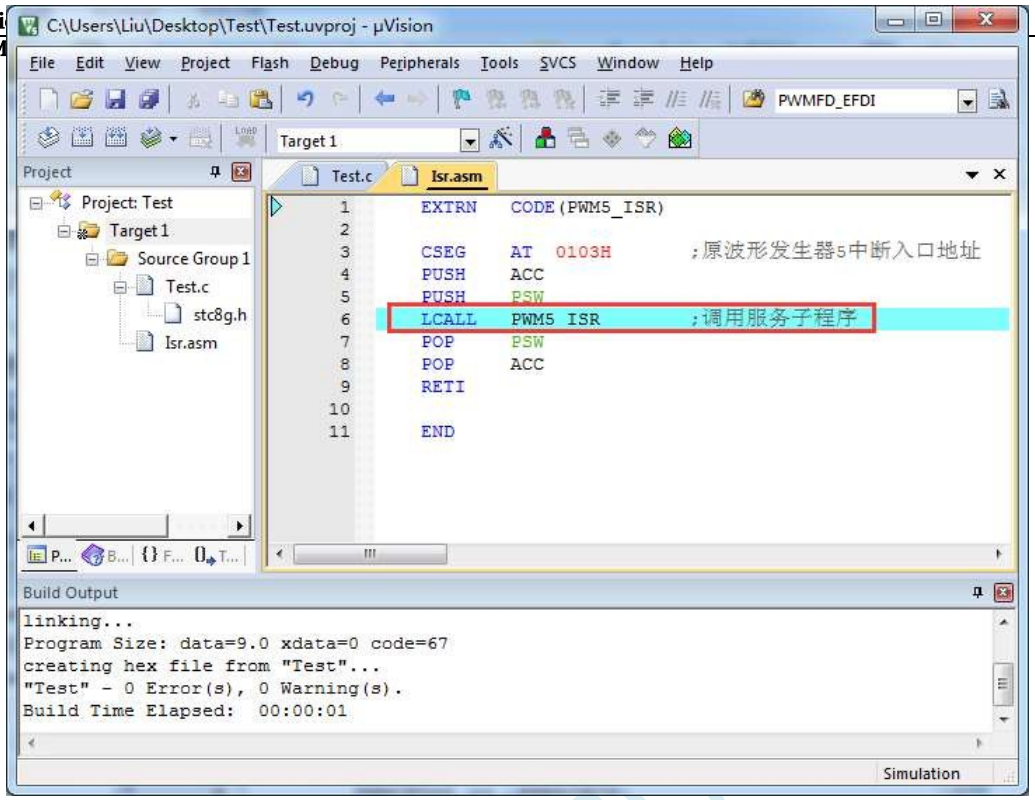
Method 3: Define the interrupt service procedure as a subroutine, and then use the LCALL instruction to execute the service procedure at the interrupt entry address in the assembly code.

The procedure is as follows:

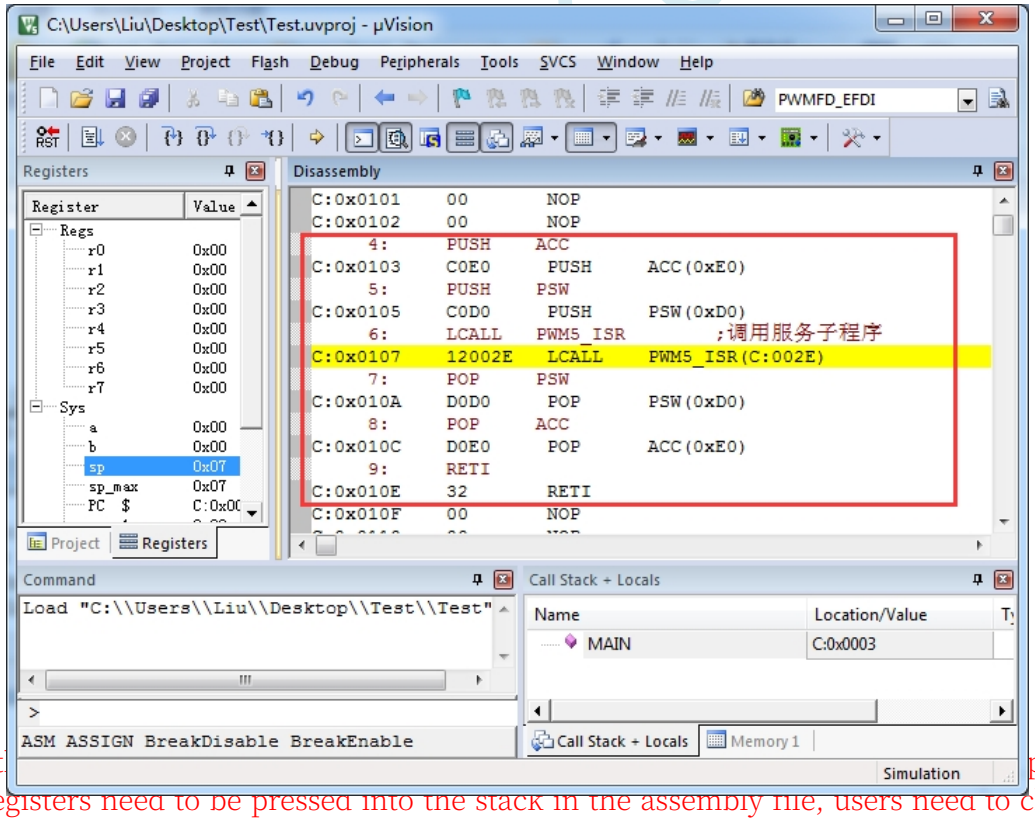
1, first of all, the interrupt service procedure to remove the "interrupt" attribute, defined as a common subroutine



2, and then in the assembly file of the 0103H address to enter the code shown in the following chart



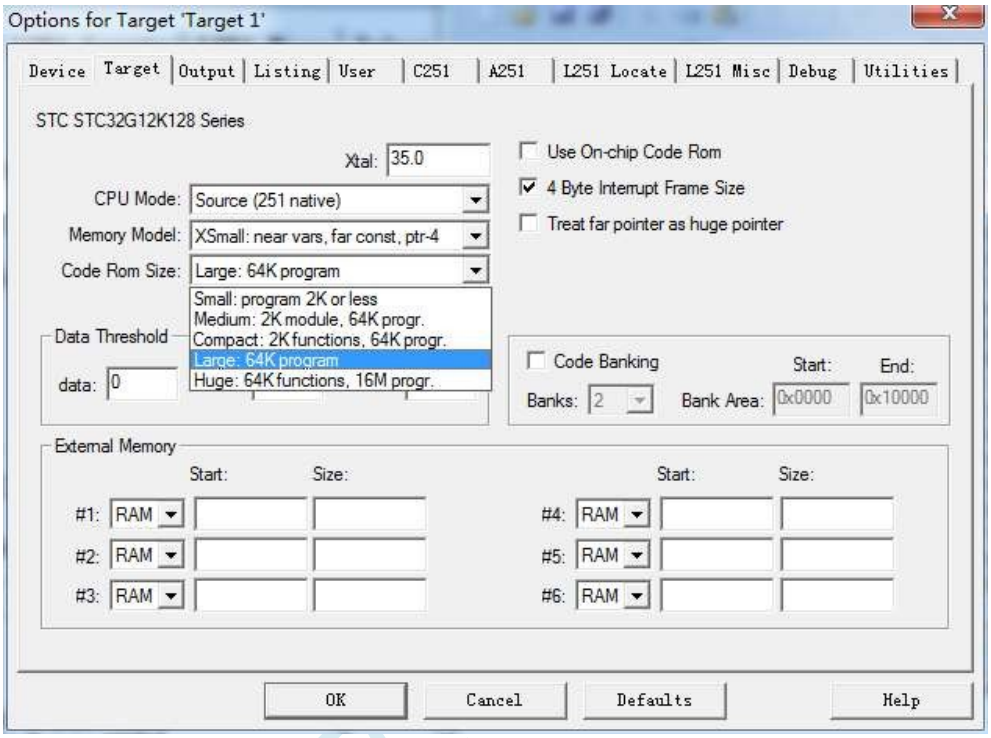
3, compiled through, can be found in the **0103H** address of the place that is the interrupt service procedure



This met problem, which registers need to be pressed into the stack in the assembly file, users need to check the disassembly code of the C program to determine. Generally it includes **PSW**, **ACC**, **B**, **DPL**, **DPH** and **R0~R7**. Except for **PSW** which must be pressed into the stack, other registers which are used in the user subroutine must be pressed into the stack.

5.10 How to set the reserved EEPROM space when the program exceeds 64K

If the user code size is within 64K, you can set "Code Rom Size" to "Large" mode, Keil compiler will automatically link all the code in the address range of FF:0000~FF:FFFF when linking the code block. The 64K of FE:0000 to FE:FFFF can be set according to the user's

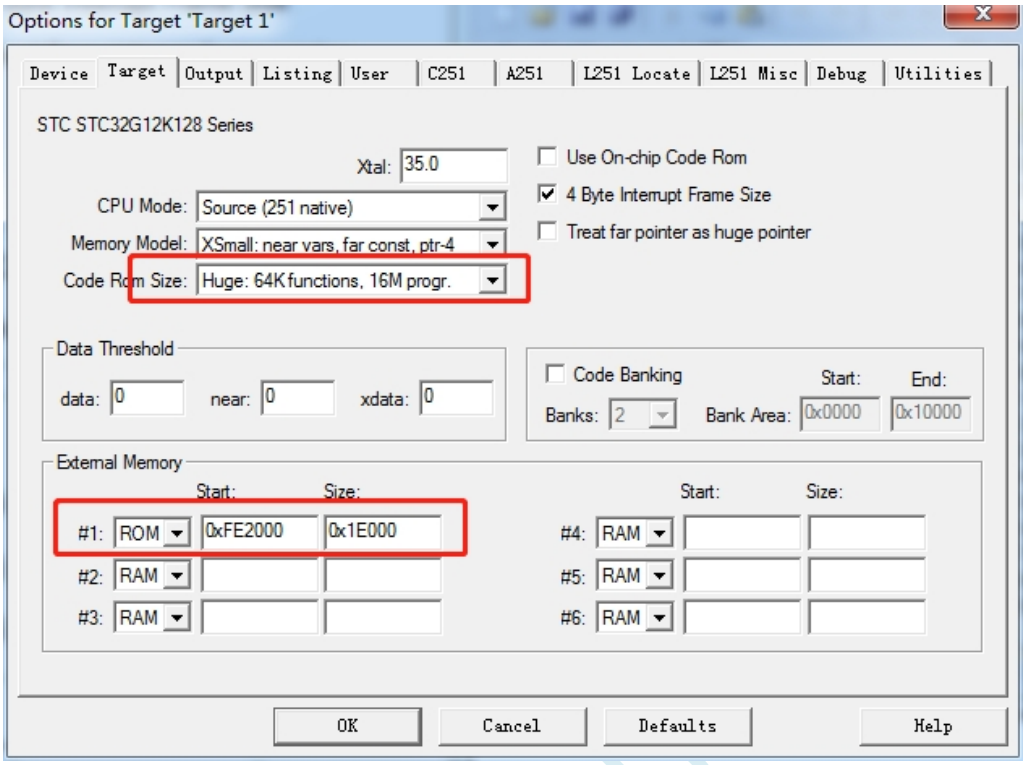


EEPROM size and can be used arbitrarily.

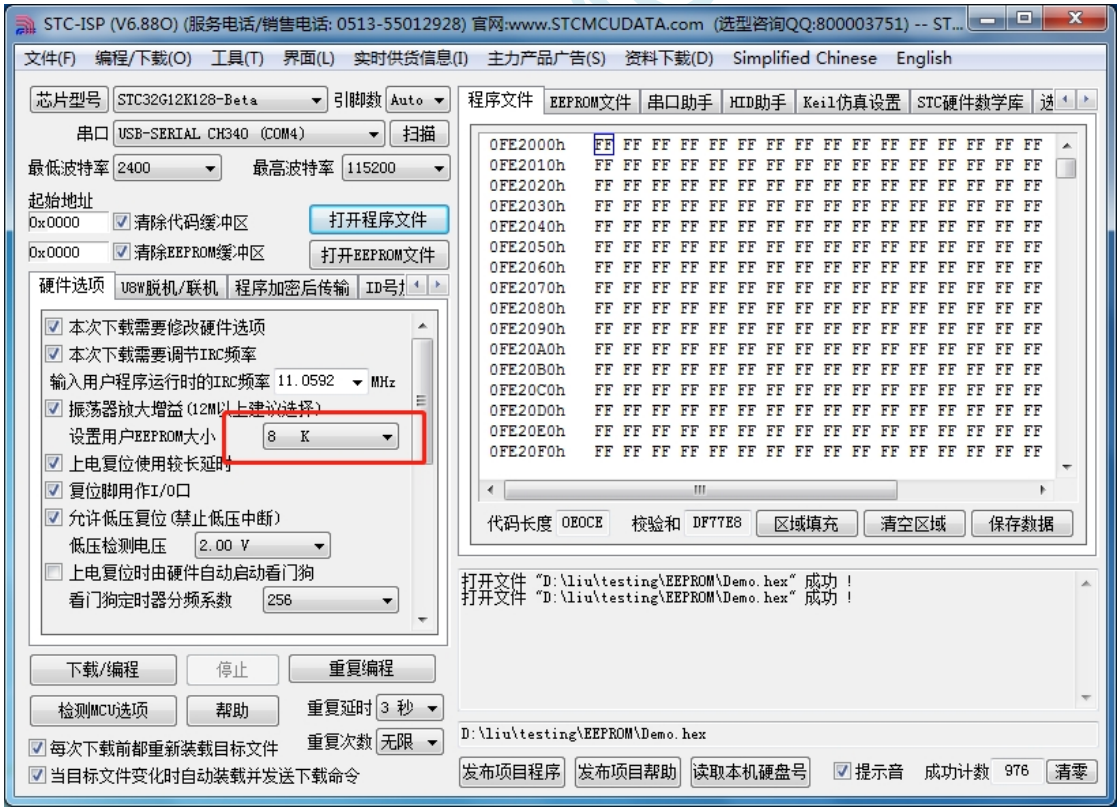
If the user code size exceeds 64K, the "Code Rom Size" must be set to "Huge" mode, in which case the Keil compiler will first link the reset code and interrupt vector code at the address starting from FF: 0000 when linking the code blocks, and the other code blocks will be automatically stored from the address examples set by the user. 0000, and other code blocks will be stored automatically from the user-set address example. Since the EEPROM of STC32G12K128 is fixed at FE:0000 in FLASH, the following settings must be made in order for the compiler not to place the user code in the EEPROM area:

For example, if you need the EEPROM size to be 8K, that is, the FE:0000 to FE:1FFF area of the FLASH address is the EEPROM area.

The FE:2000 to FF:FFFF area of the FLASH address is the user code area. The following settings are required in Keil:



The following settings are required in the ISP download software:



5.11 Simulation of STC32G series microcontrollers using the SWD interface of STC-USB Link1D

5.11.1 Getting to know the STC-USB Link1D tool

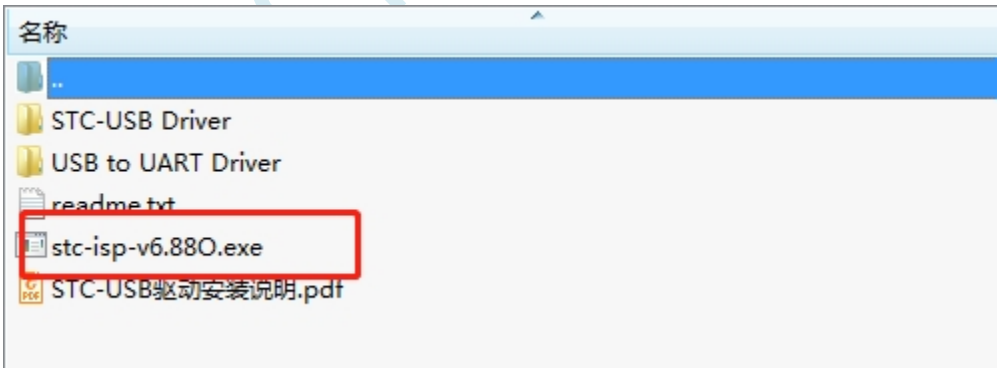
Note: The tool has a large USB connector and a small USB connector, either one can be connected to the computer.

5.11.2 Hardware connection method

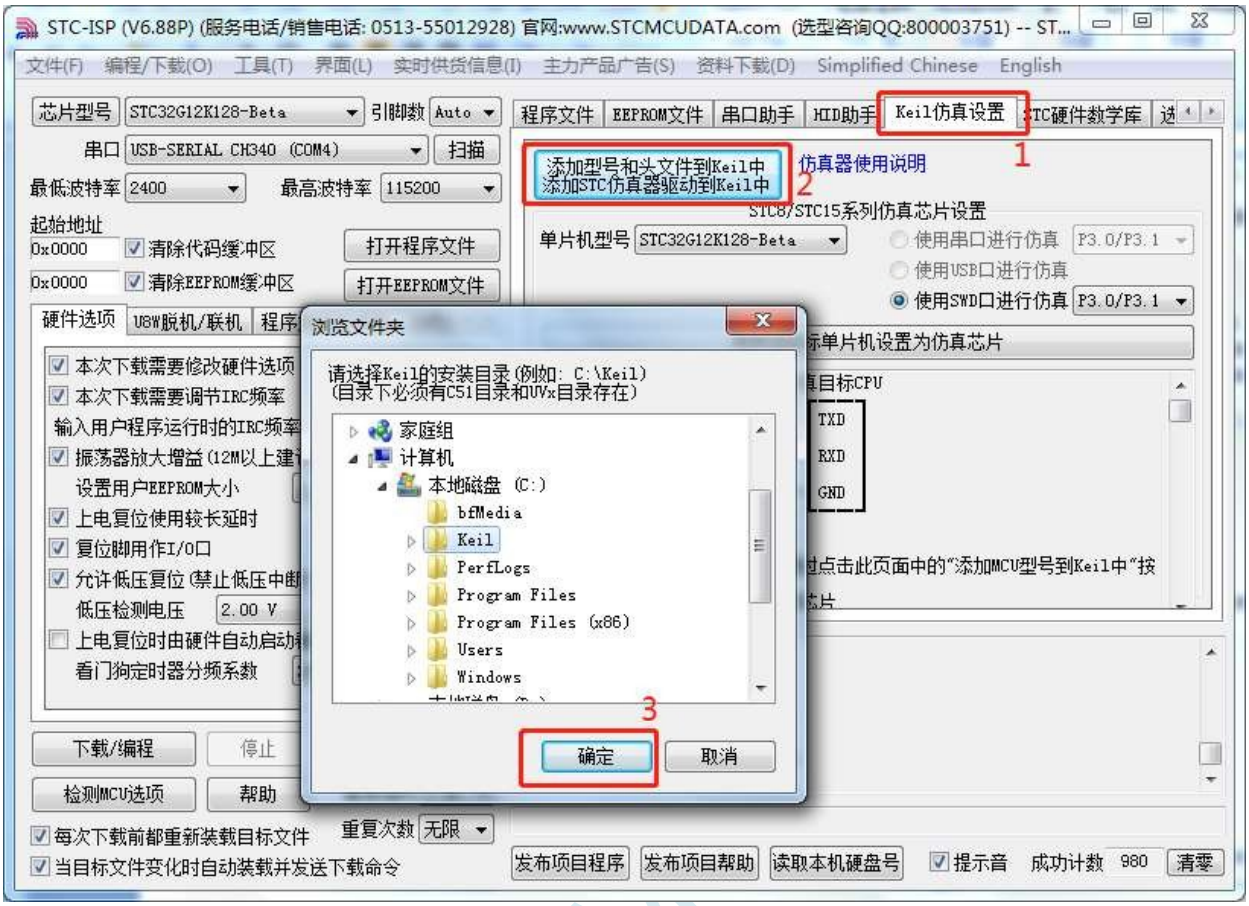
5.11.3 Installation of the emulation driver

Firstly, download the latest STC-ISP download software from the STC website.

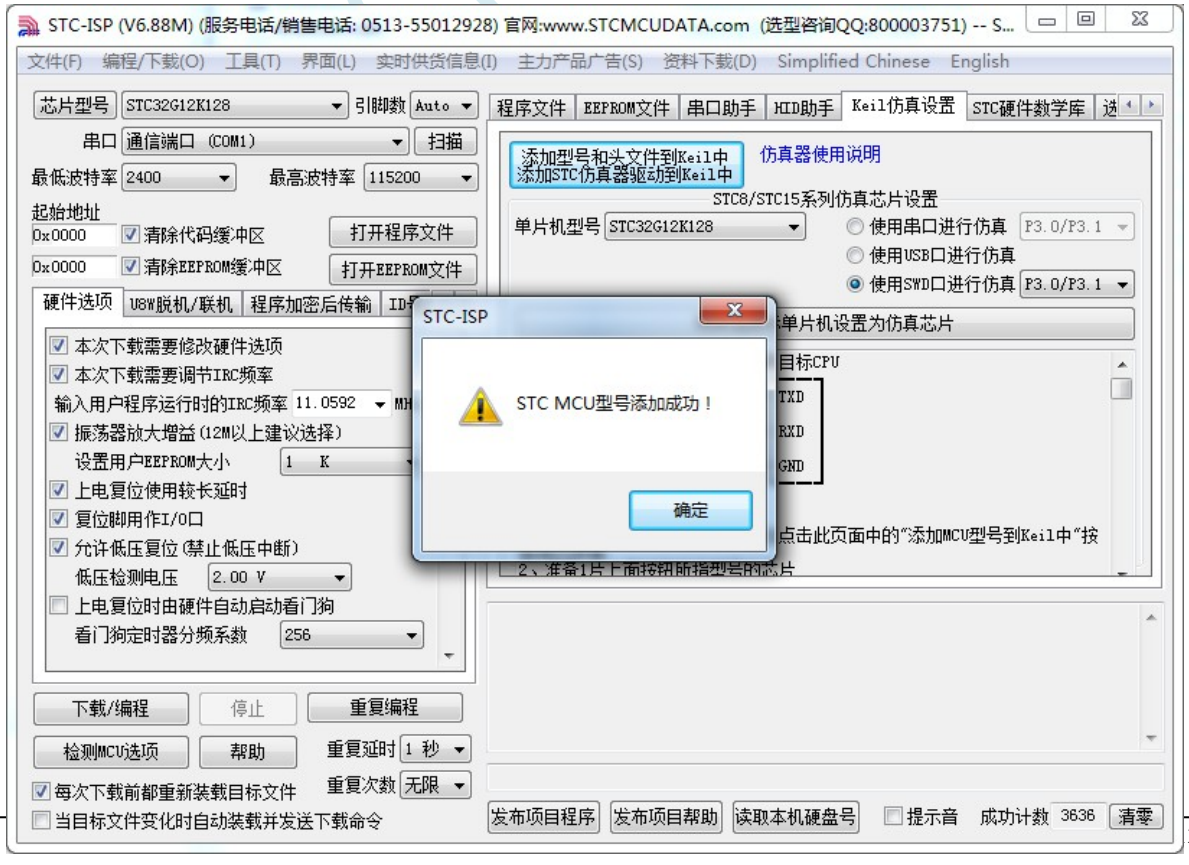
After downloading and unzipping the package, open the "stc-isp-vxx.exe" executable file.



Click "Add model and header file..." in the "Keil simulation settings" page of the download software. button in the "Keil Simulation Settings" page of the download software (Figure 2 below).

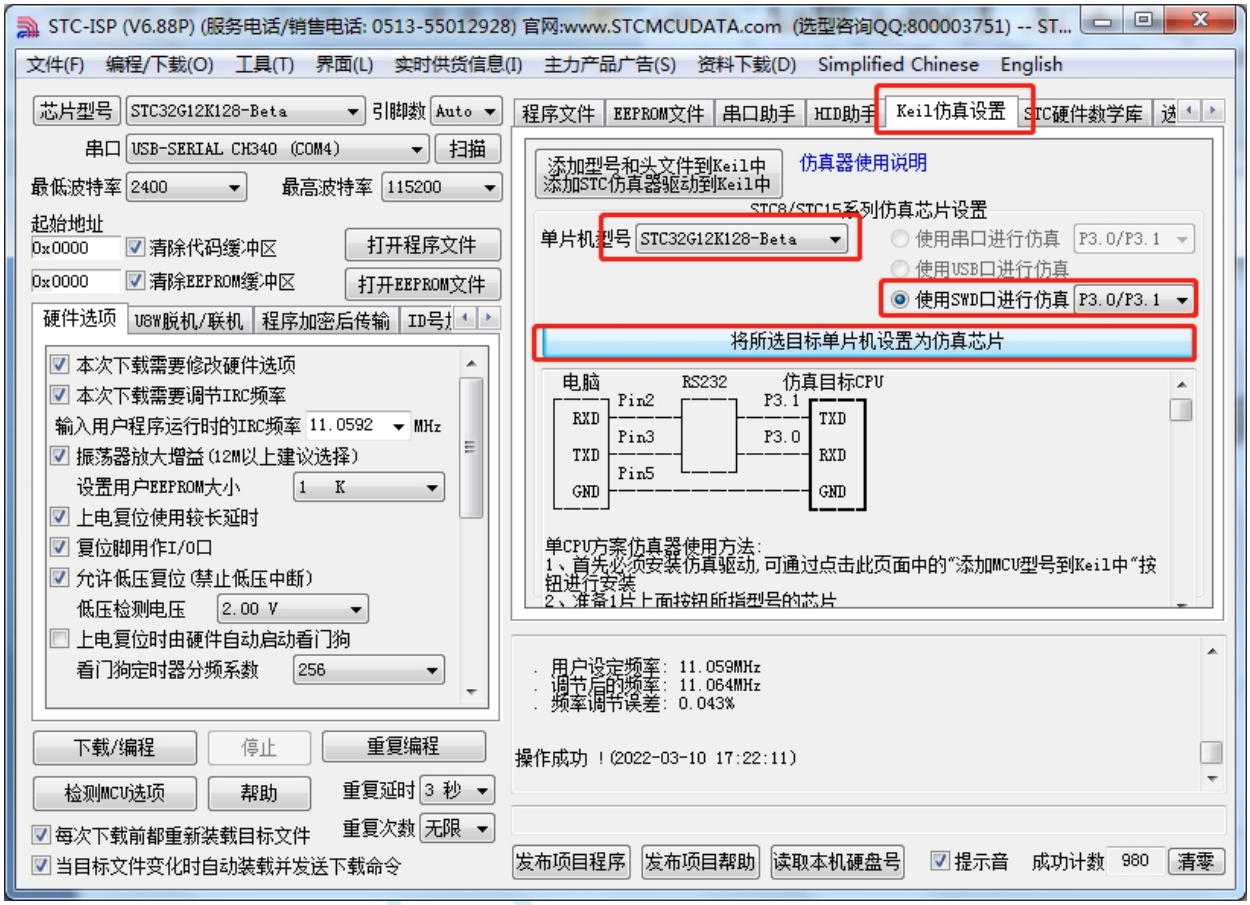


In the pop-up "Browse Folder" window, select the installation directory of Keil (generally the installation directory of Keil is "c:\keil"), click OK, if the pop-up message "STC MCU model added successfully", it means the driver has been installed. After clicking OK, if "STC MCU model added successfully" pops up, it means the driver has been installed.



5.11.4 Making emulation chips

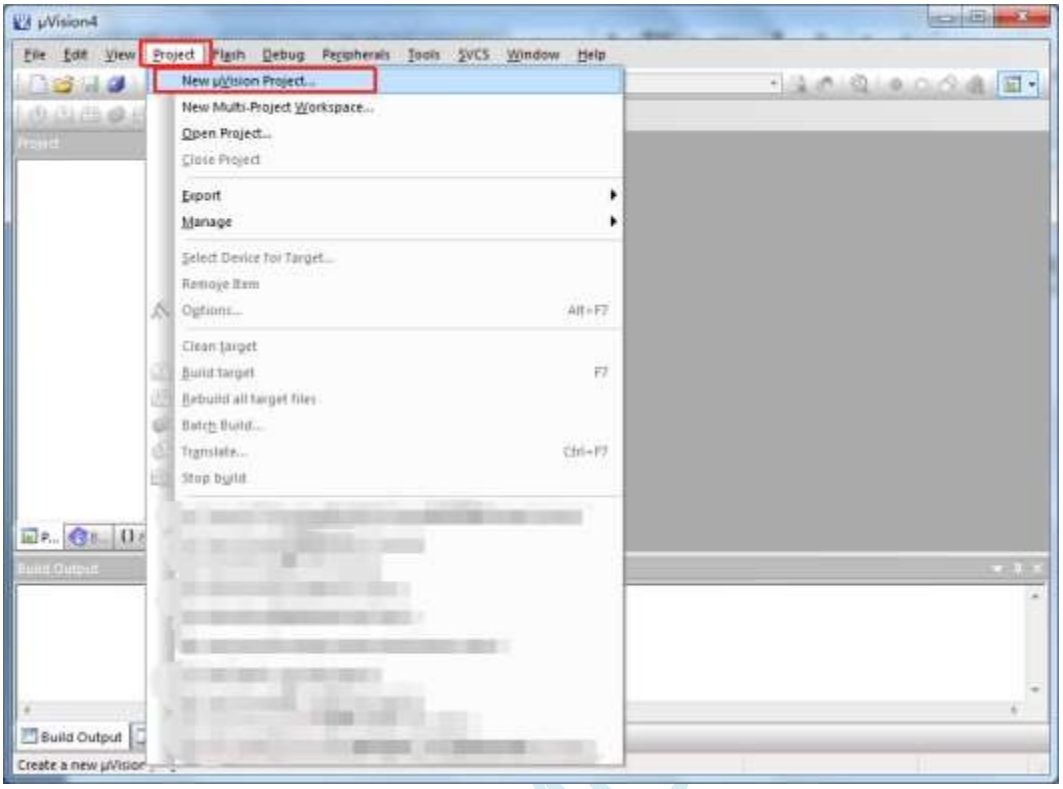
The hardware emulation function is not enabled by default when the chip is shipped from the factory. To enable the hardware emulation function, you need to use the ISP download



software to set it.

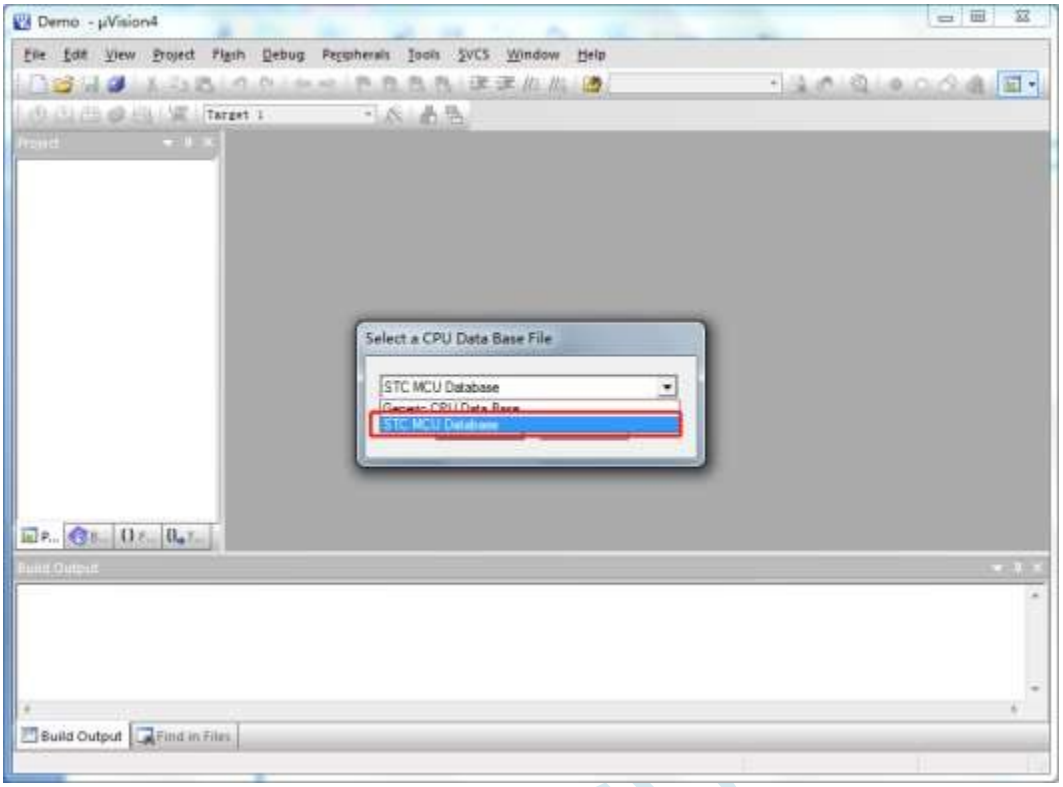
Select "Use SWD port for emulation", and the chip will have the emulation function after the download is completed.

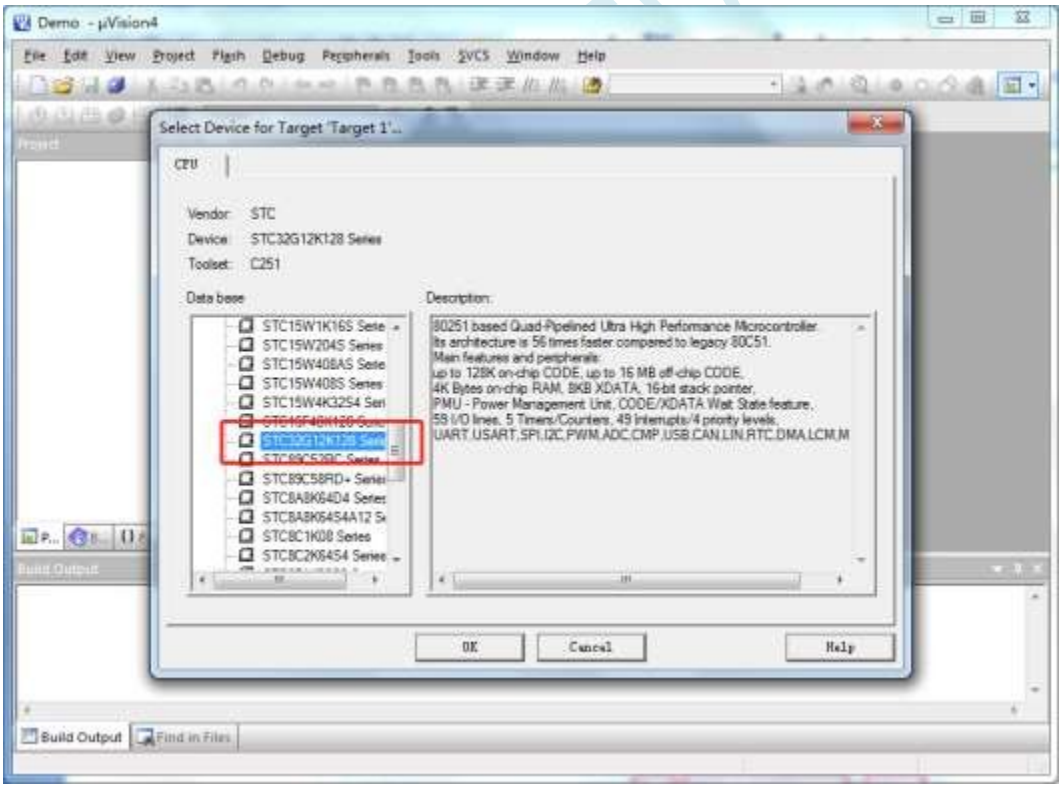
5.11.5 Creating and setting up a project in Keil software



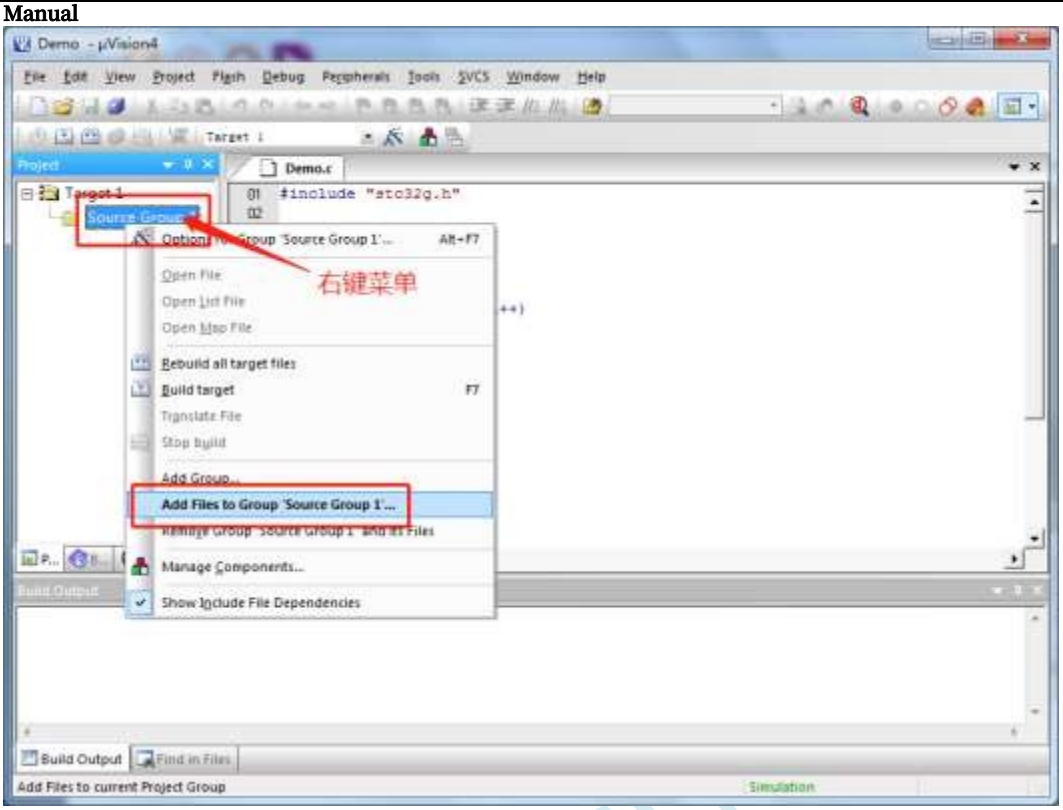
Specify the project path and enter the project name

Select target chip model: STC32G12K128

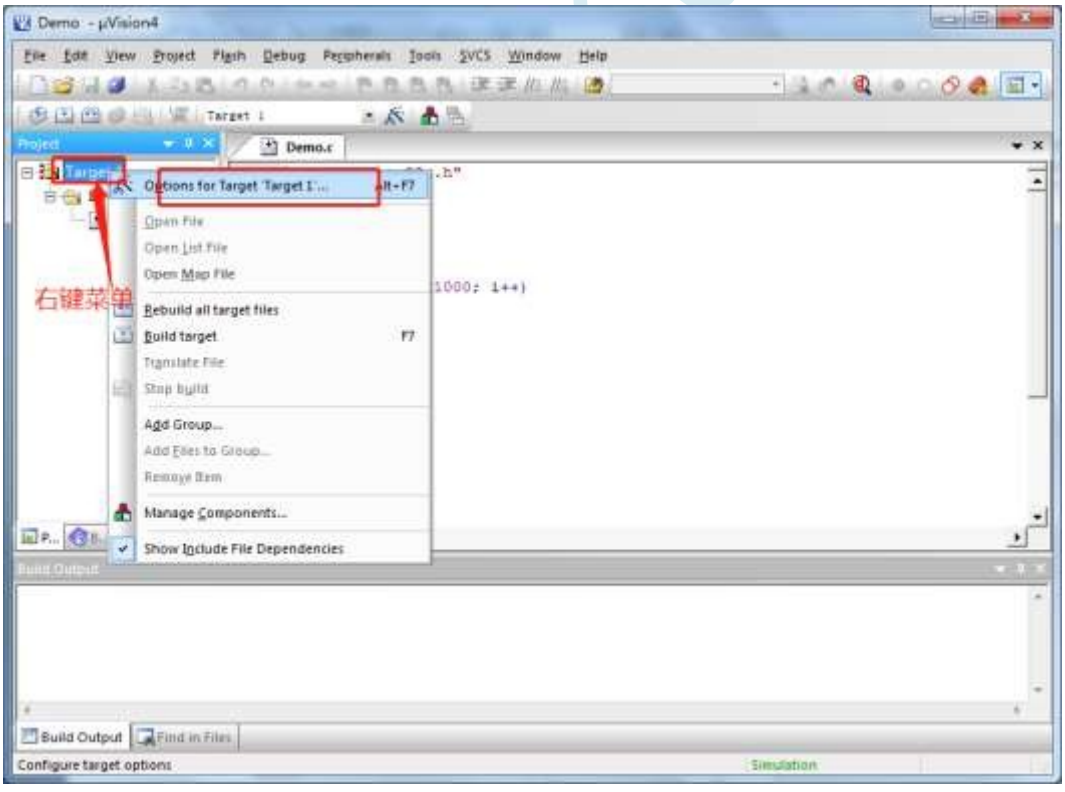


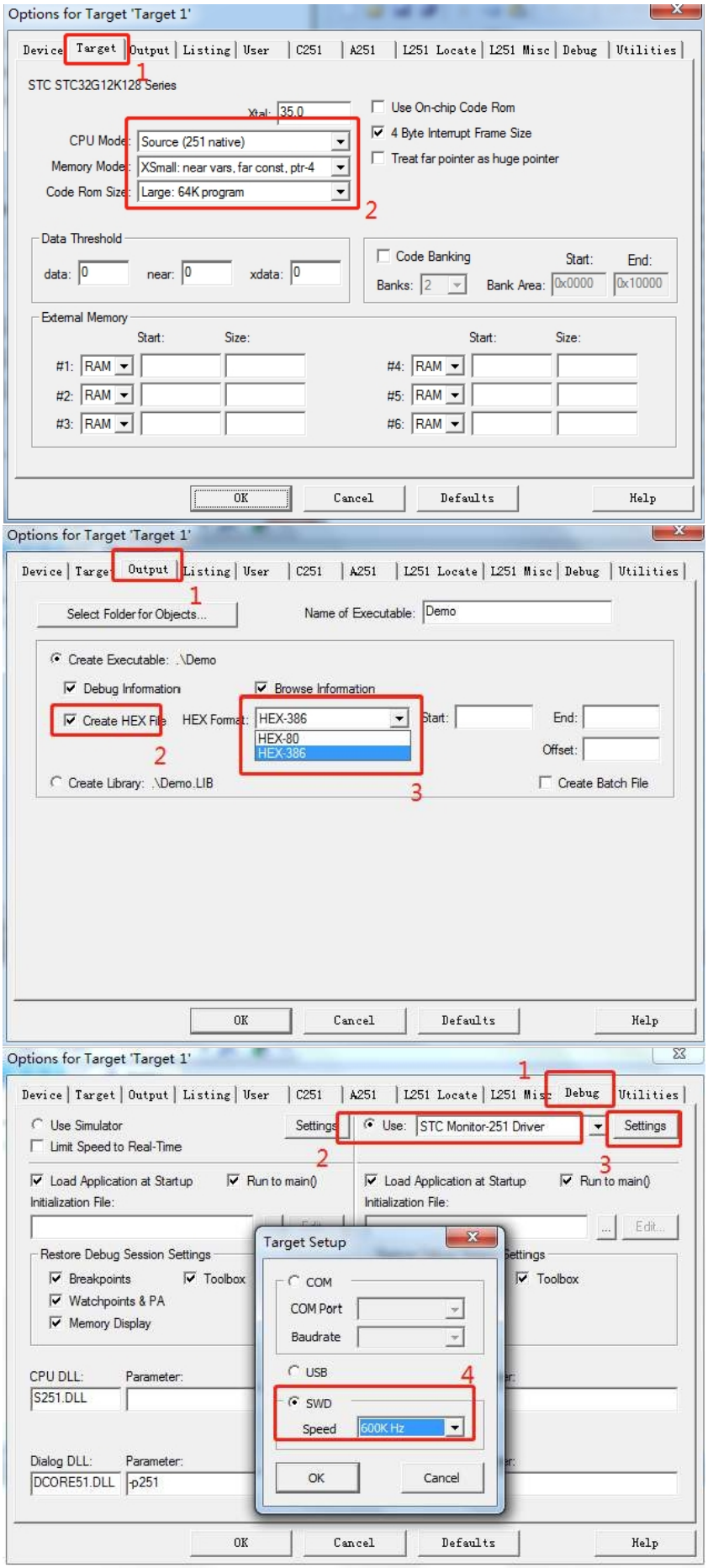


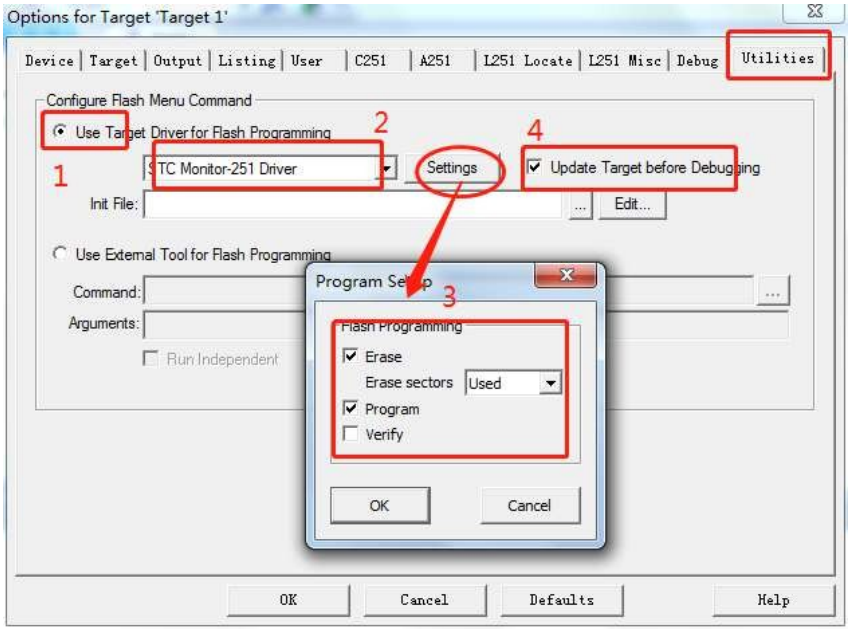
Create a code file and add it to the project



Project Settings

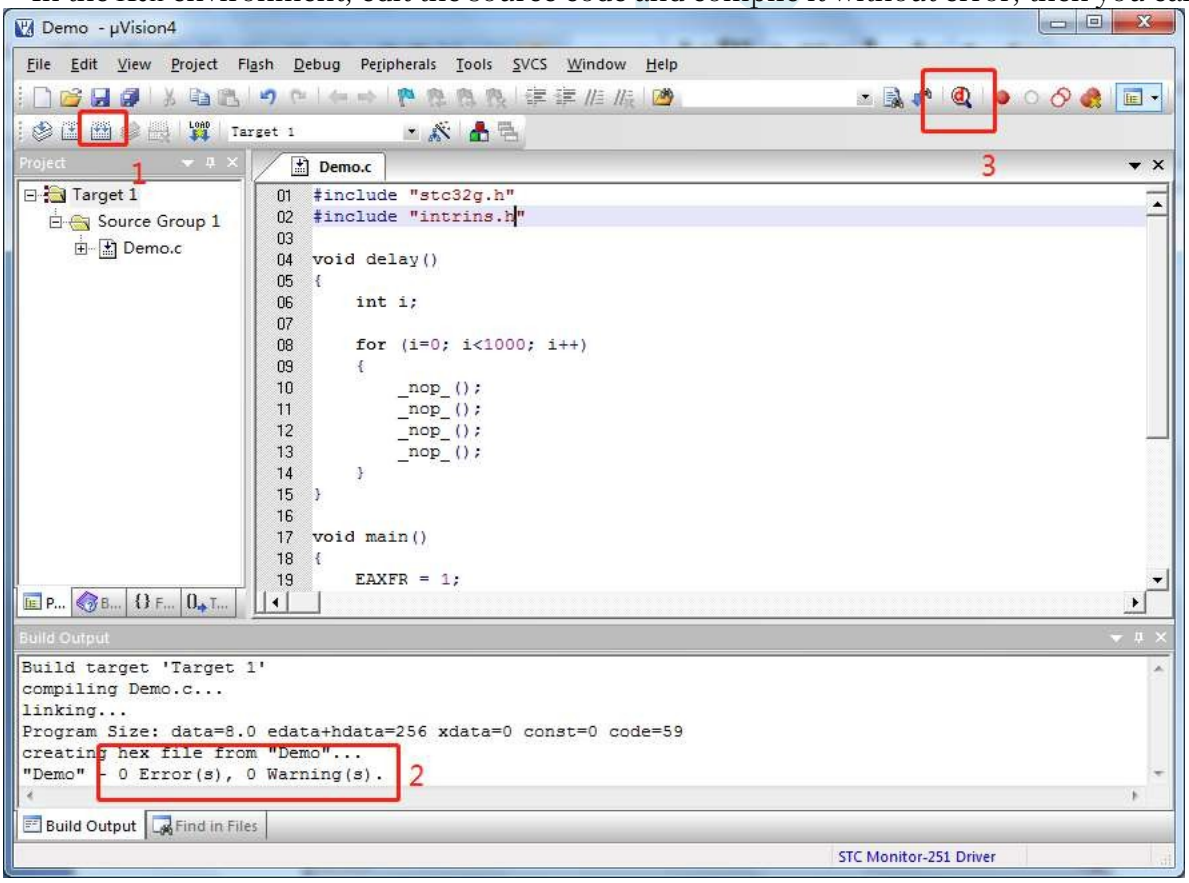


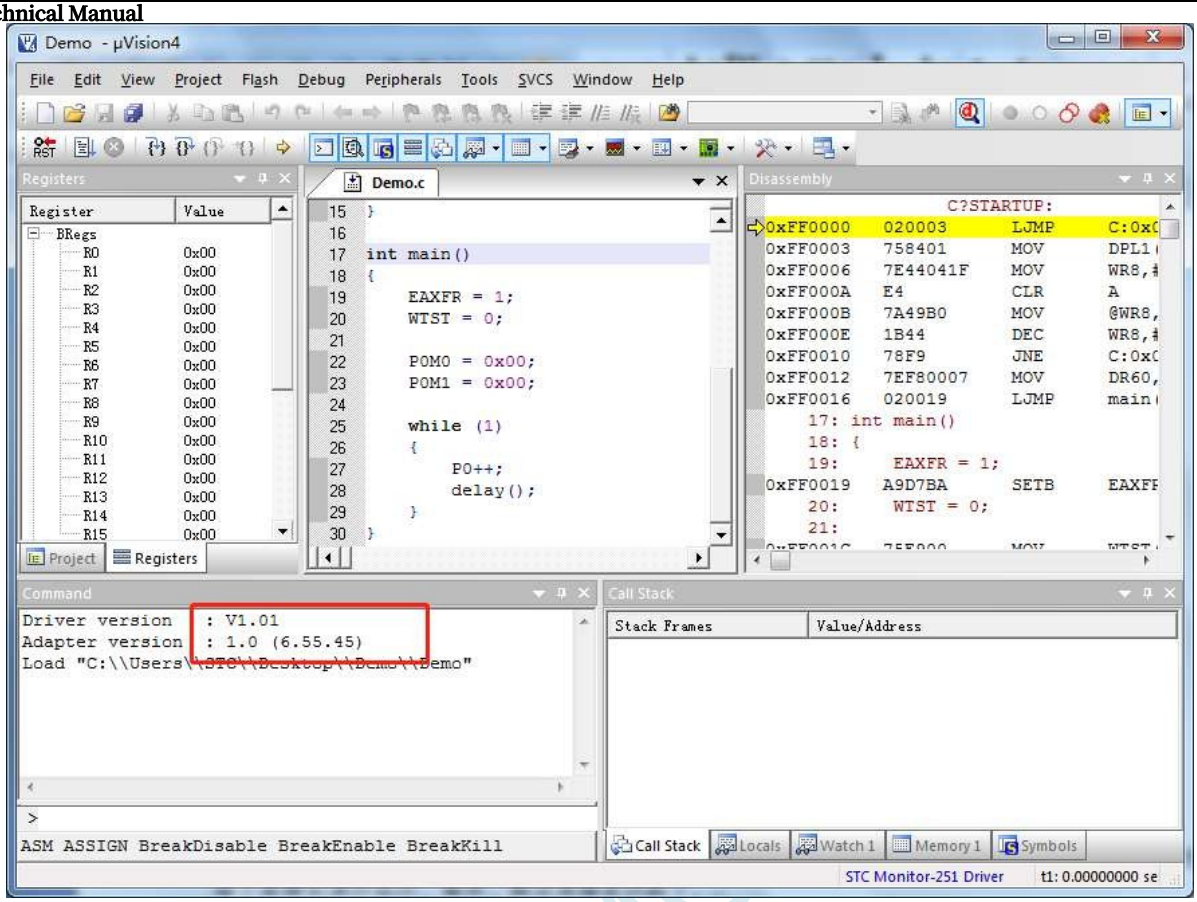

Options for Target 'Target 1'
Device
Target
Output
Listing
User
C251
A251
L251 Locate
L251 Misc
Debug
Utilities
1
STC STC32G12K128 Series
Xtal: 35.0
Use On-chip Code Rom
4 Byte Interrupt Frame Size
Treat far pointer as huge pointer
CPU Mode: Source (251 native)
Memory Model: XSmall: near vars, far const, ptr-4
Code Rom Size: Large: 64K program
2
Data Threshold
data: 0
near: 0
xdata: 0
Code Banking
Start:
End:
Banks: 2
Bank Area: 0x0000
0x10000
External Memory
Start:
Size:
#1: RAM
#2: RAM
#3: RAM
#4: RAM
#5: RAM
#6: RAM
OK
Cancel
Defaults
Help
Options for Target 'Target 1'
Device
Target
Output
Listing
User
C251
A251
L251 Locate
L251 Misc
Debug
Utilities
1
Select Folder for Objects...
Name of Executable: Demo
Create Executable: .\Demo
Debug Information
Browse Information
Create HEX File
HEX Format: HEX-386
HEX-80
HEX-386
Start:
End:
Offset:
2
3
Create Library: .\Demo.LIB
Create Batch File
OK
Cancel
Defaults
Help
Options for Target 'Target 1'
1
Device
Target
Output
Listing
User
C251
A251
L251 Locate
L251 Misc
Debug
Utilities
Use Simulator
Limit Speed to Real-Time
Settings
Use: STC Monitor-251 Driver
Settings
2
3
Load Application at Startup
Run to main()
Initialization File:
Load Application at Startup
Run to main()
Initialization File:
Edit...
Restore Debug Session Settings
Breakpoints
Toolbox
Watchpoints & PA
Memory Display
Toolbox
CPU DLL:
Parameter:
S251.DLL
Dialog DLL:
Parameter:
DCORE51.DLL
-p251
Target Setup
COM
COM Port
Baudrate
USB
4
SWD
Speed
600K Hz
OK
Cancel
OK
Cancel
Defaults
Help



5.11.6 Compile, download and simulate

In the Keil environment, edit the source code and compile it without error, then you can start the





If the chip is made and connected correctly, the emulation driver version will be displayed as shown in the figure above, and the user code can be downloaded correctly to the microcontroller, and then debugging functions such as running, single-stepping and breakpoints can be performed.

5.12 STC-USB Link1D Tool Usage Notes

5.12.1 Tool interface description

The STC-USB Link1D tool is an updated version of the STC-USB Link1, with two additional functions based on the STC-USB Link1.

STC-CDC serial port, can be used as a universal USB to serial port.

Please refer to the appendix section for notes on the use of the STC-USB Link1 tool.

Front view of the tool

Reverse view of the tool

User-Vcc, external power supply to the tool

S-RxD, Group 2 STC-CDC serial transmit pin, connects to the receive pin of the user microcontroller serial port

S-TxD, Group 2 STC-CDC serial port receive pin, connected to the transmit pin of the user microcontroller serial port

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

S-P3.0, Link1 hardware emulation data line, Link1 serial port transmitter pin for ISP download, Group 1 STC-CDC serial port transmitter pin

S-P3.1, Link1 hardware emulation clock line, Link1 serial port receive pin for ISP download, Group 1 STC-CDC serial port receive pin

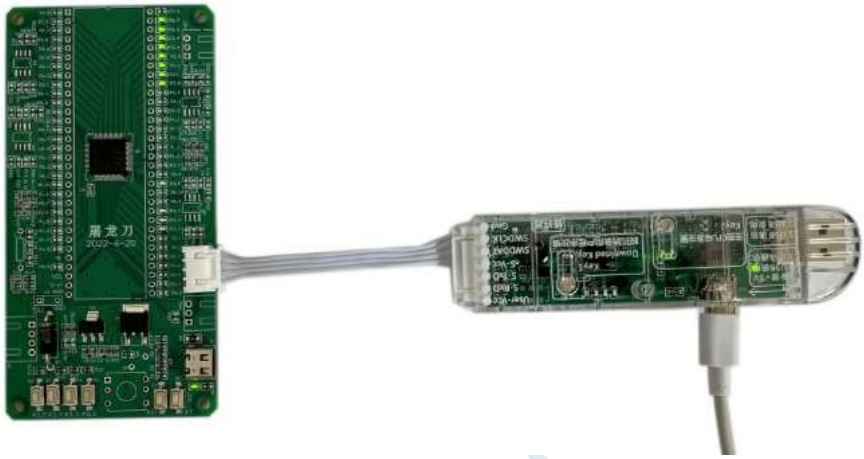
Gnd, Ground

Pin Number	interface name	interface function
1	User-Vcc	Powering the tool from the user's system only
2	S-RxD	Group 2 Transmit pin of the STC-CDC serial port to connect to the receive pin of the serial port of the user microcontroller
3	S-TxD	Group 2 STC-CDC serial port receive pin, connect to user microcontroller serial port transmit pin
4	S-Vcc	Power the user system only from this tool
5	S-P3.0	Serial transmit pin for ISP download using Link1D, connect to P3.0 of the target microcontroller.
		Data pin for SWD hardware emulation using Link1D to connect to the target microcontroller's SWDDAT

		Group 1 Transmit pin of the STC-CDC serial port to connect to the receive pin of the serial port of the user microcontroller
6	S-P3.1	Serial receive pin for ISP download using Link1D, connect to P3.1 of the target microcontroller.
		Clock pin for SWD hardware emulation using Link1D, connect to SWDCLK of the target microcontroller
		Group 1 Receive pin of the STC-CDC serial port, connect to the transmit pin of the user microcontroller serial port
7	Gnd	earth (wire)

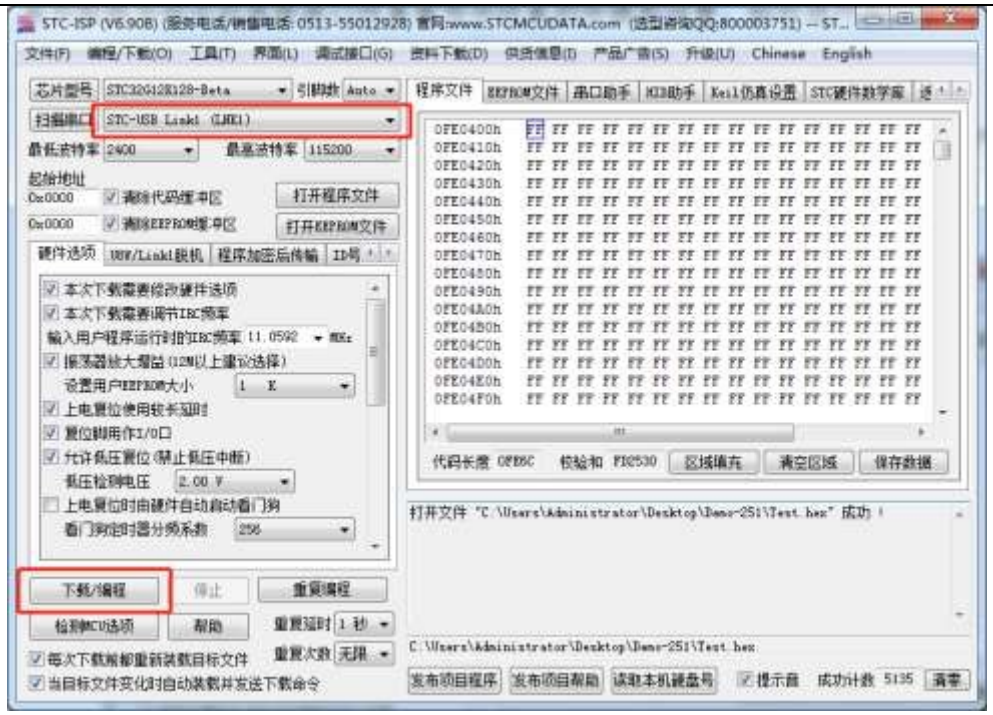
5.12.2 STC-USB Link1D Practical Application

1. SWD hardware simulation of STC32G series microcontrollers using STC-USB Link1D tool.
Connect the S-Vcc, S-P3.0, S-P3.1, and GND of the tool to the M-Vcc, P3.0 (SWDDAT), P3.1 (SWDCLK), and GND of the target microcontroller, respectively, as shown in the following figure, and then refer to the steps and settings of the hardware simulation in the previous chapters to carry out the hardware simulation of SWD.



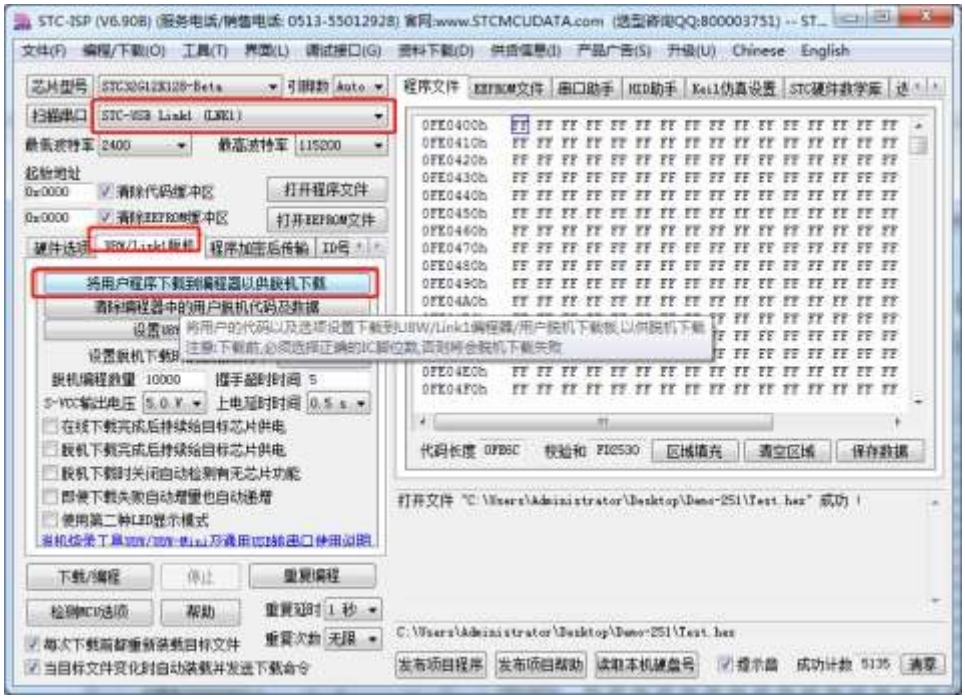
2、Serial port emulation for STC15 and STC8 series using STC-USB Link1D tool
The S-Vcc, S-P3.0, S-P3.1, and GND of the tool are connected to the M-Vcc, P3.0 (RxD), P3.1 (TxD), and GND of the target microcontroller, and then the serial port number corresponding to STC-CDC1 is selected in the Keil emulation settings, and then the steps and settings of the emulation can be carried out by referring to the emulation steps and settings of the chapter of the Direct Serial Emulation of the STC15/STC8 series datasheet. Then refer to the emulation steps and settings in the Direct Serial Emulation chapter in the STC15/STC8 series datasheet to carry out serial port emulation.

3、Use STC-USB Link1D tool to download ISP online for all STC series microcontrollers.
The S-Vcc, S-P3.0, S-P3.1 and GND of the tool are connected to the M-Vcc, P3.0 (RxD), P3.1 (TxD) and GND of the target microcontroller, and then the serial port number of the STC-ISP download software is "STC-USB Link1 (LNK1)". Select "STC-USB Link1 (LNK1)" for the serial port number in the STC-ISP download software, open the program file and set the relevant hardware options, and then click the "Download/Programming" button to carry out the ISP online download.


STC-ISP (V6.90B) (服务电话/销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (选型咨询QQ:800003751) -- ST...
文件(F) 编程/下载(O) 工具(T) 界面(L) 调试接口(G) 资料下载(D) 供货信息(I) 产品广告(S) 升级(U) Chinese English
芯片型号 STC32G12K128-Beta
引脚数 Auto
扫描串口 STC-USB Link1 (LPT1)
最低波特率 2400
最高波特率 115200
起始地址
0x0000
清除代码缓冲区
打开程序文件
0x0000
清除EEPROM缓冲区
打开EEPROM文件
硬件选项
USB/Link1脱机
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 1 K
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
下载/编程
停止
重复编程
检测MCU选项
帮助
重复延时 1 秒
重复次数 无限
每次下载前都重新装载目标文件
当目标文件变化时自动装载并发送下载命令
程序文件
EEPROM文件
串口助手
HID助手
Keil仿真设置
STC硬件数学库
0FE0400h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0410h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0420h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0430h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0440h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0450h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0460h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0470h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0480h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE0490h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE04A0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE04B0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE04C0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE04D0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE04E0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0FE04F0h FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
代码长度 0FB8C
校验和 FB2530
区域填充
清空区域
保存数据
打开文件 "C:\Users\Administrator\Desktop\Demo-251\Test.hex" 成功！
C:\Users\Administrator\Desktop\Demo-251\Test.hex
发布项目程序
发布项目帮助
读取本机硬盘号
提示音
成功计数 5135
清零

4、Use STC-USB Link1D tool to download ISP offline for all series of STC microcontrollers.

Select "STC-USB Link1 (LNK1)" for the serial port number in the STC-ISP download software, open the programme file and set the relevant hardware options, and then click the "Download user program to programmer for offline download" button in the "U8W/Link1 Offline" page. Click the "Download User Program to Programmer for Offline Download" button on the "U8W/Link1 Offline" screen to download the user code and related settings to the memory on the STC-USB Link1 utility.



Connect the S-Vcc, S-P3.0, S-P3.1, and GND of the tool to the M-Vcc, P3.0 (RxD), and P3.1 of the target microcontroller, respectively.

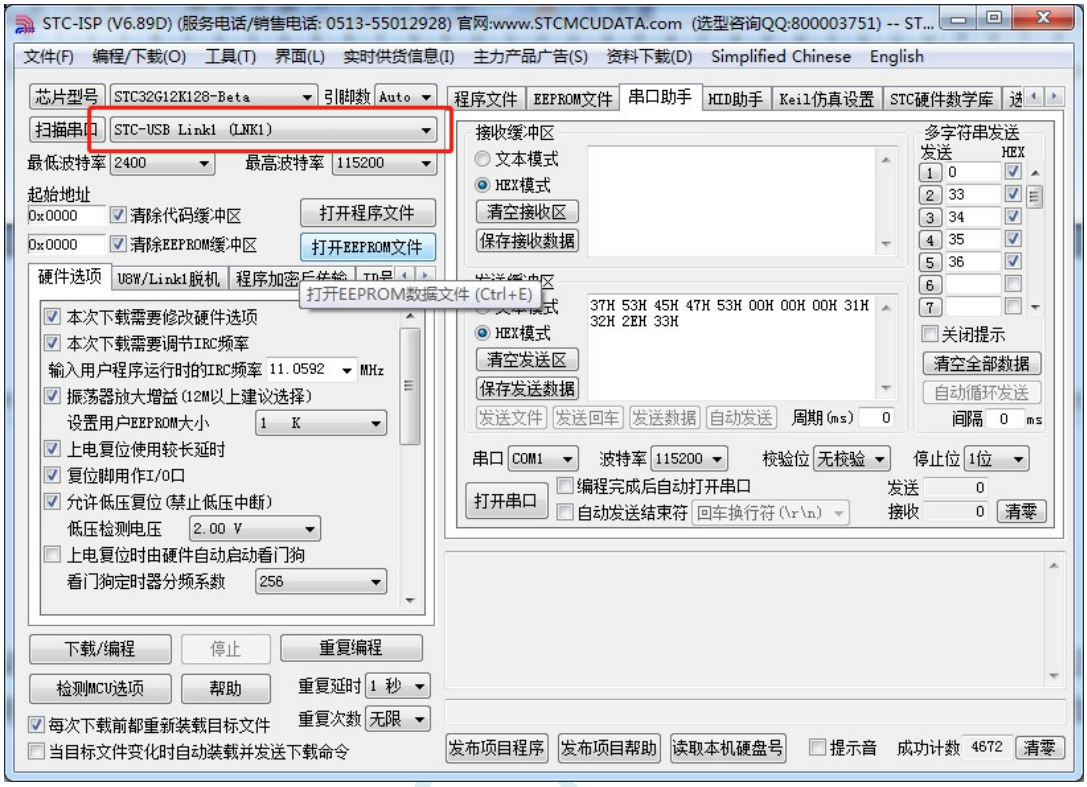
(TxD), GND connection, then press the "Key1" button on the tool to download the target chip offline (i.e., without PC control, independent ISP download).

5, STC-USB Link1D tool as a general-purpose USB dedicated serial port tool use

The STC-USB Link1D tool provides two STC-CDC serial ports, which can be used as a general-purpose USB dedicated serial port tool. Since the first serial port CDC1 shares S-P3.0 and S-P3.1 ports with hardware emulation and ISP downloads, and the second serial port CDC2 is an independent serial port, it is recommended that S-P3.0 and S-P3.1 be used as emulation and ISP downloads, and when you need to use the general-purpose USB dedicated serial port tool, use CDC2 corresponding to S-TxD and S-RxD. (Note: Both CDC1 and CDC2 can be used without usage conflicts. Therefore, it is recommended to use S-P3.0 and S-P3.1 for emulation and ISP download, and use CDC2 corresponding to S-TxD and S-RxD when you need to use the general-purpose USB dedicated serial port tool (Note: CDC1 and CDC2 can be used independently as the general-purpose USB dedicated serial port tool in the absence of usage conflict).

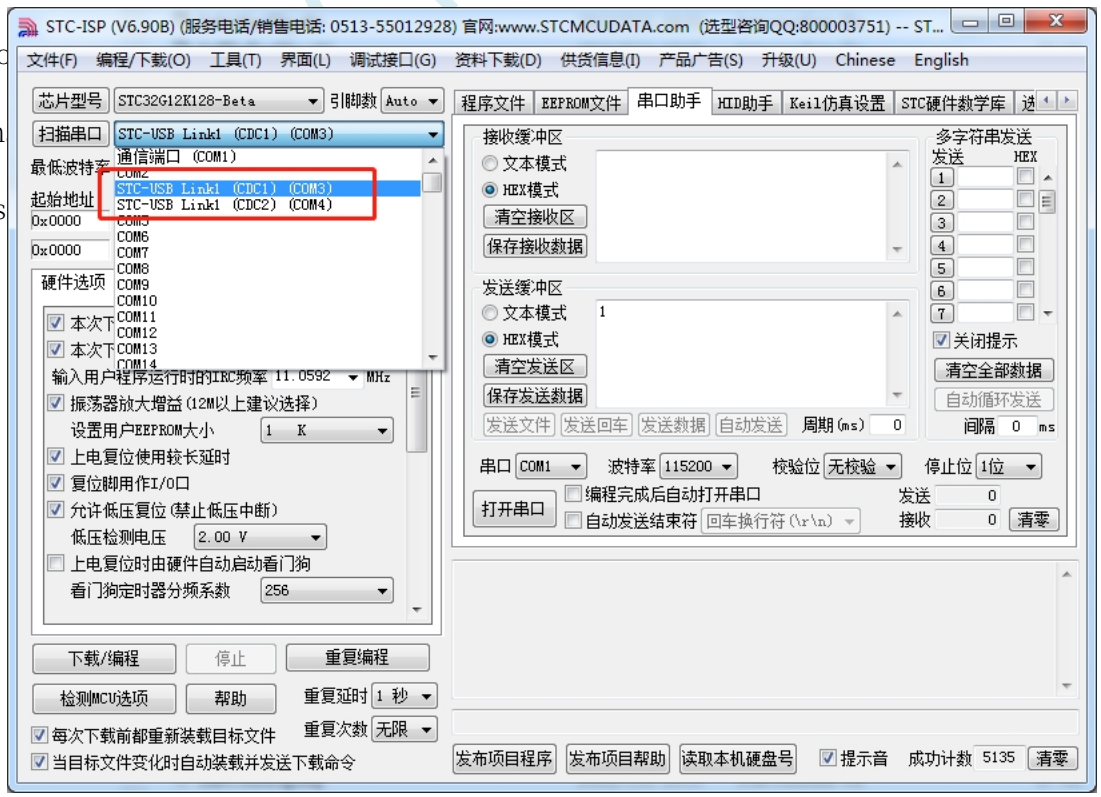
5.12.3 Correct identification of tools

The STC-USB Link1D tool is shipped from the factory with the STC-USB Link1D control programme burned into the main control chip. Under normal circumstances, when the tool is



connected to the computer, "STC-USB Link1 (LNK1)" will be recognised in the STC-ISP download software immediately, as shown in the figure below.

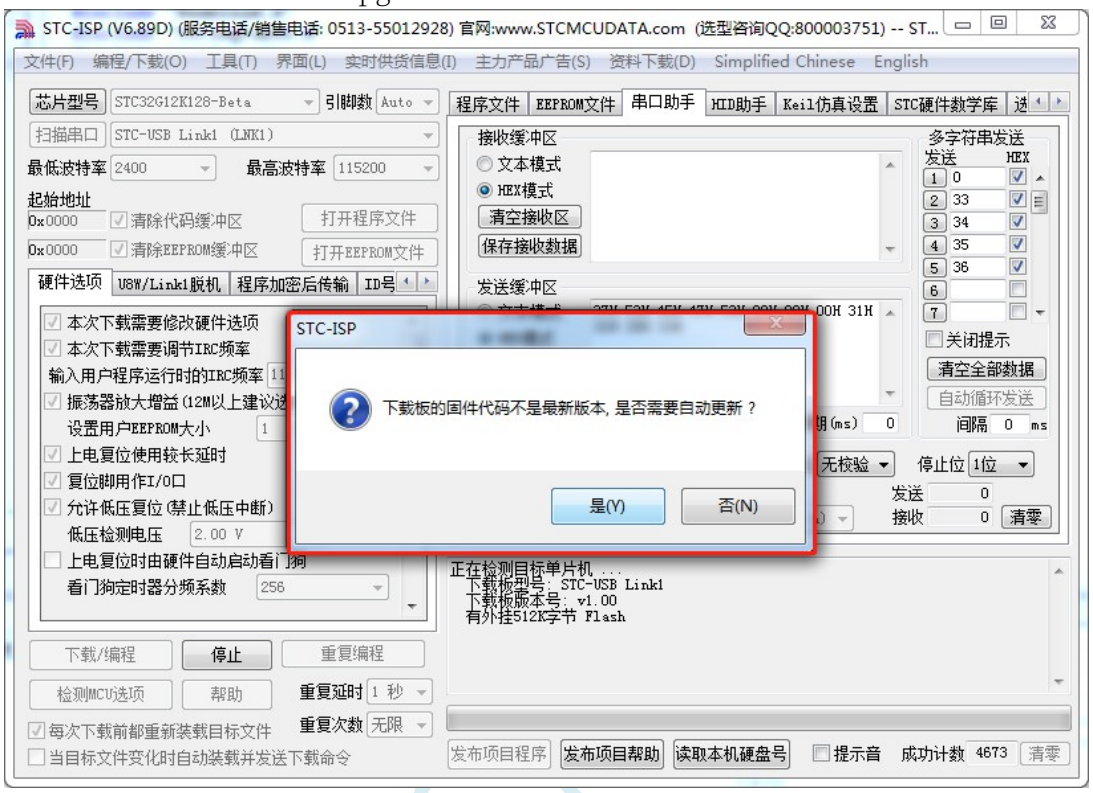
Once correctly recognised, you can use the STC-USB Link1D for online ISP



Can be used as a general purpose USB to Serial tool.

5.12.4 Automatic upgrade of tool firmware

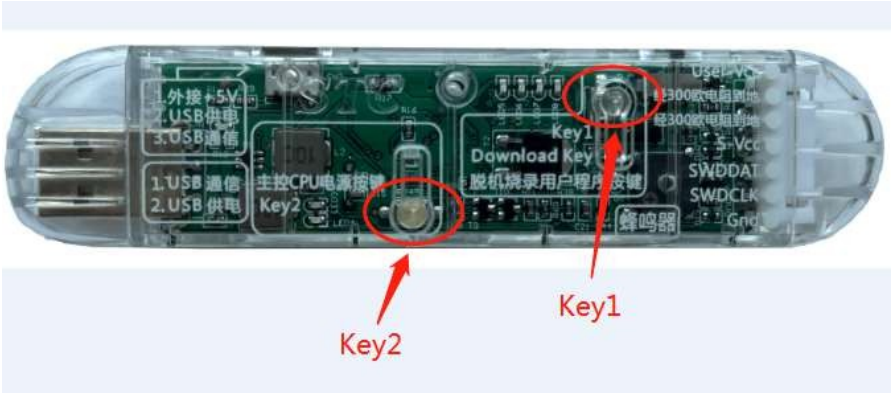
When using the tool for ISP download, the software pops up the following screen, indicating that the tool's firmware needs to be upgraded



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

5.12.5 Access to the updated firmware

Connect the tool to your computer using the USB cable, then press and hold Key1 on the tool, then press Key2 and wait for the STC-ISP download software to recognise "STC USB Writer (HID1)" before releasing Key1.

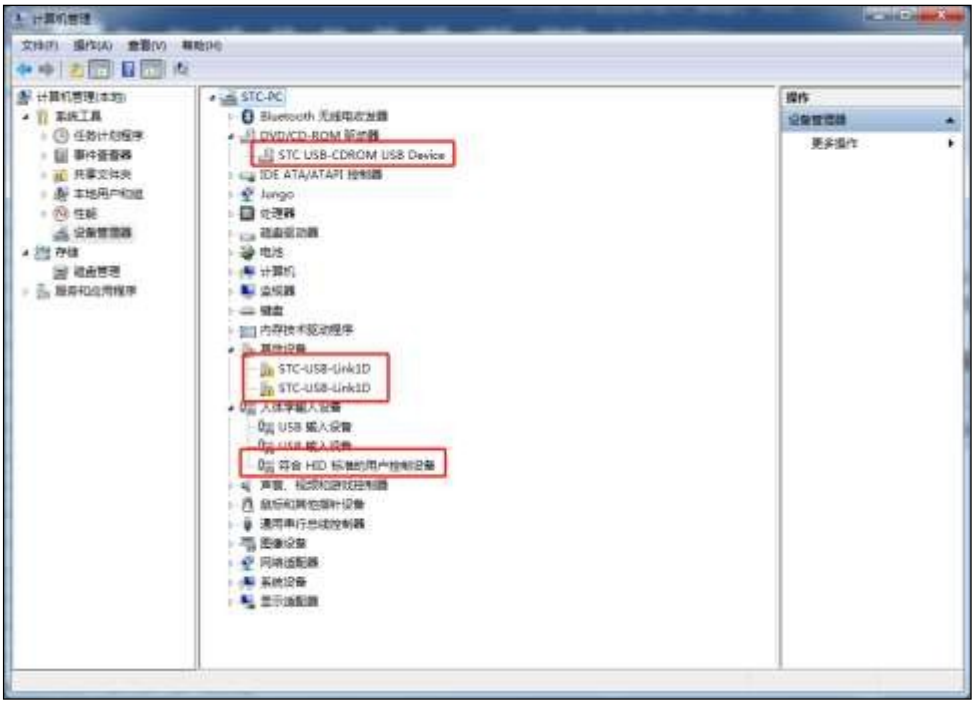


5.12.6 STC-USB Link1D Driver Installation Procedure

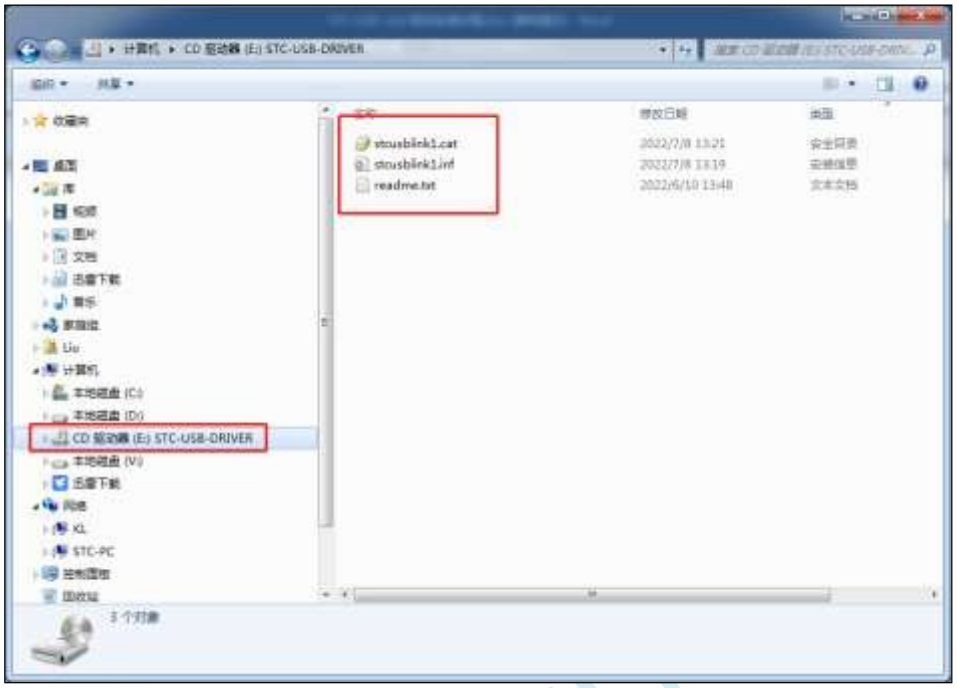
1. Insert the STC-USB Link1D tool into the USB port of the computer, and the computer will display the

following screen

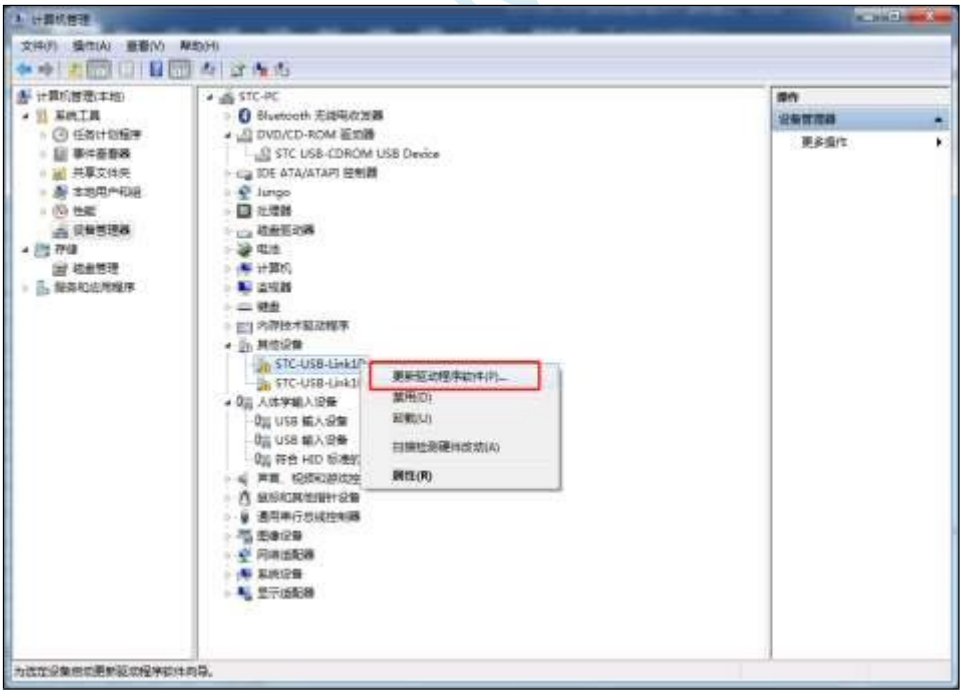
2, the driver is automatically installed, in the computer's device manager, it shows that the STC-USB Link1D device has been automatically identified in the HID interface and USB optical drive interface, but the two CDC virtual serial interface will have a yellow exclamation mark, indicating that the virtual serial port driver is not installed successfully, you need to manually install. As shown in the figure below



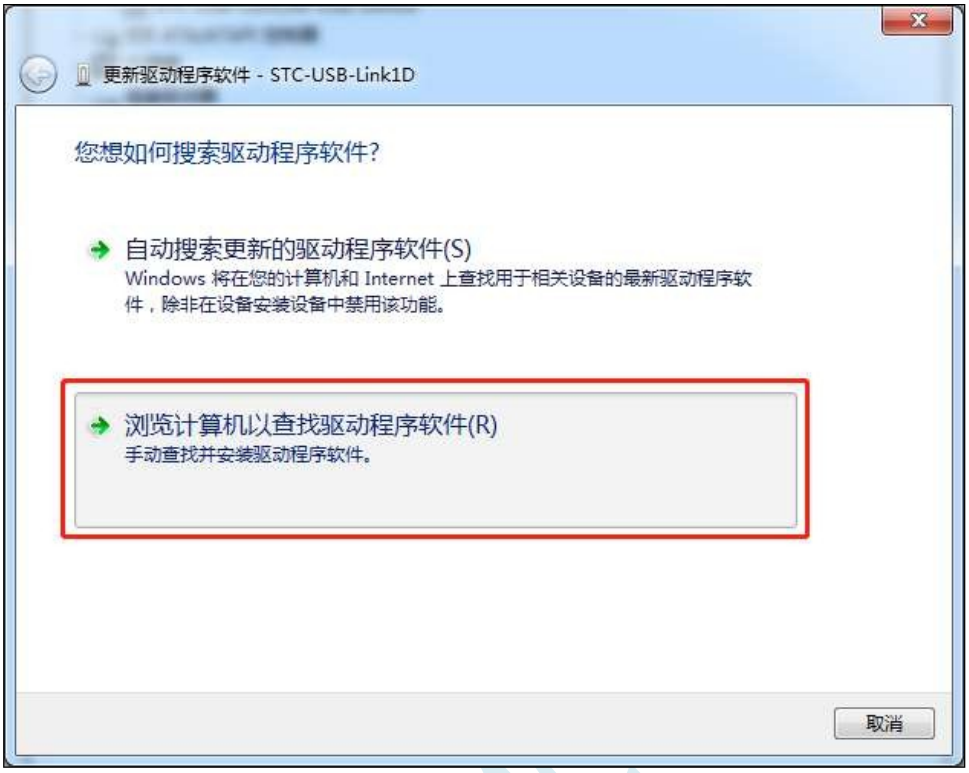
In Windows Explorer, open the automatically recognised USB drive, which contains the driver for the virtual serial port.



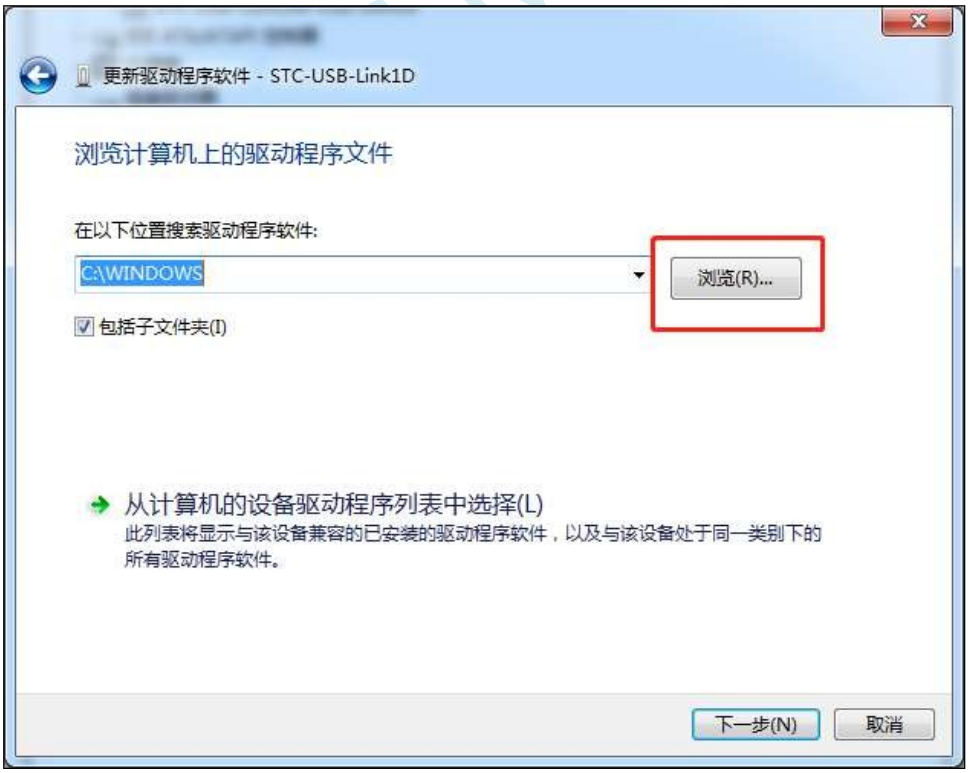
3, manually install the virtual serial port driver steps are as follows: First of all, in the device manager to find the first with a yellow exclamation mark "STC-USB Link1D", and click the right mouse button, select the right-click menu in the "Update Driver Software (P) ... "



4, in the pop-up "Update Driver Software" window, click "Browse Computers for Driver Software".

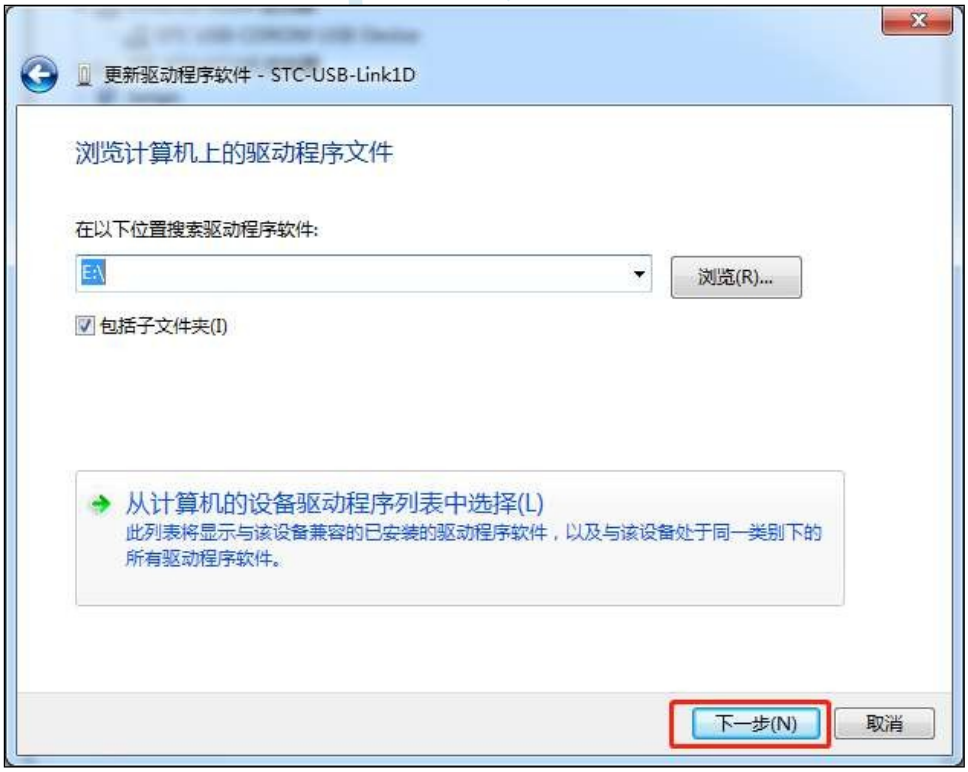


5. Click the "Browse" button in the following screen.



6, in the browse folder window, select the "STC-USB-DRIVER" optical drive, and confirm!

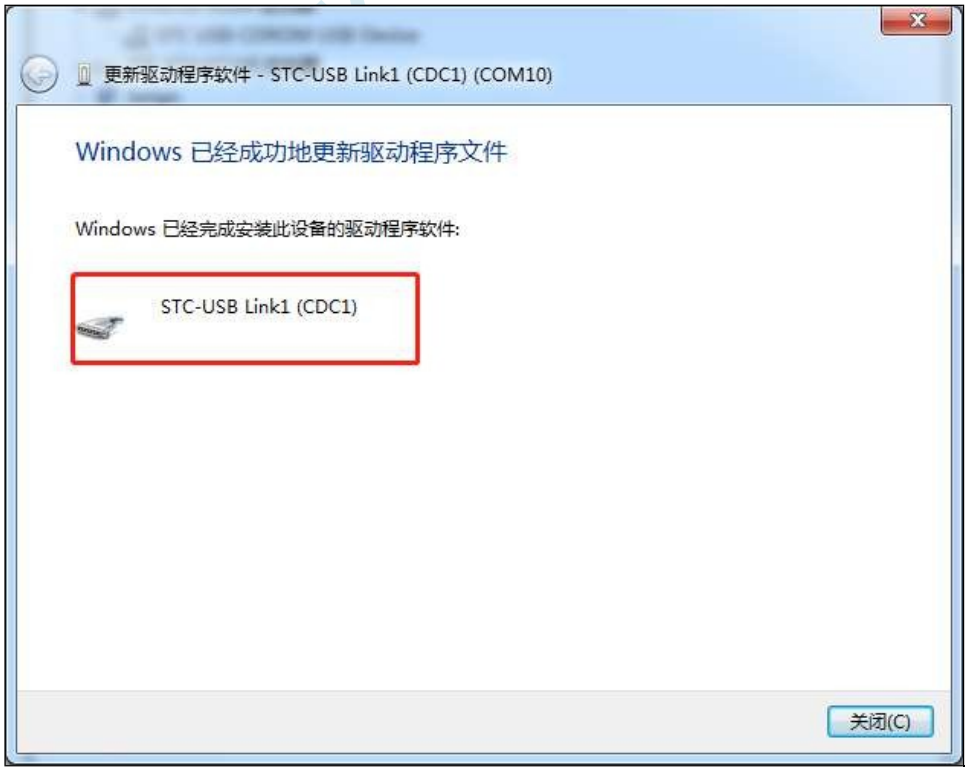
7, the following chart, click "Next" button to start installing the driver



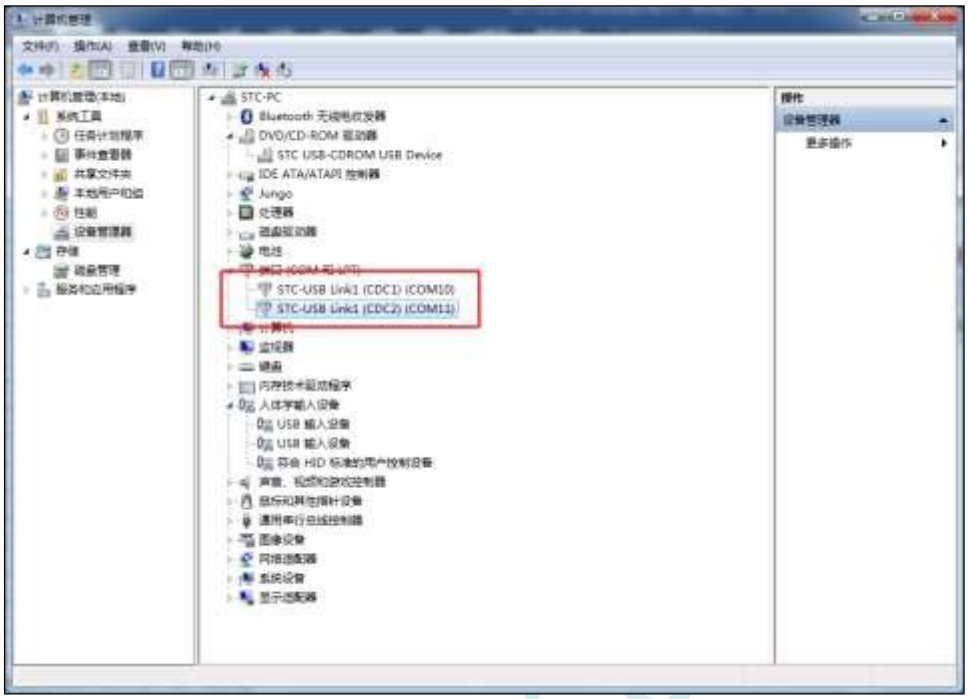
8, the installation process will pop up "Windows Security" pop-up window, click "Always install this

driver software".

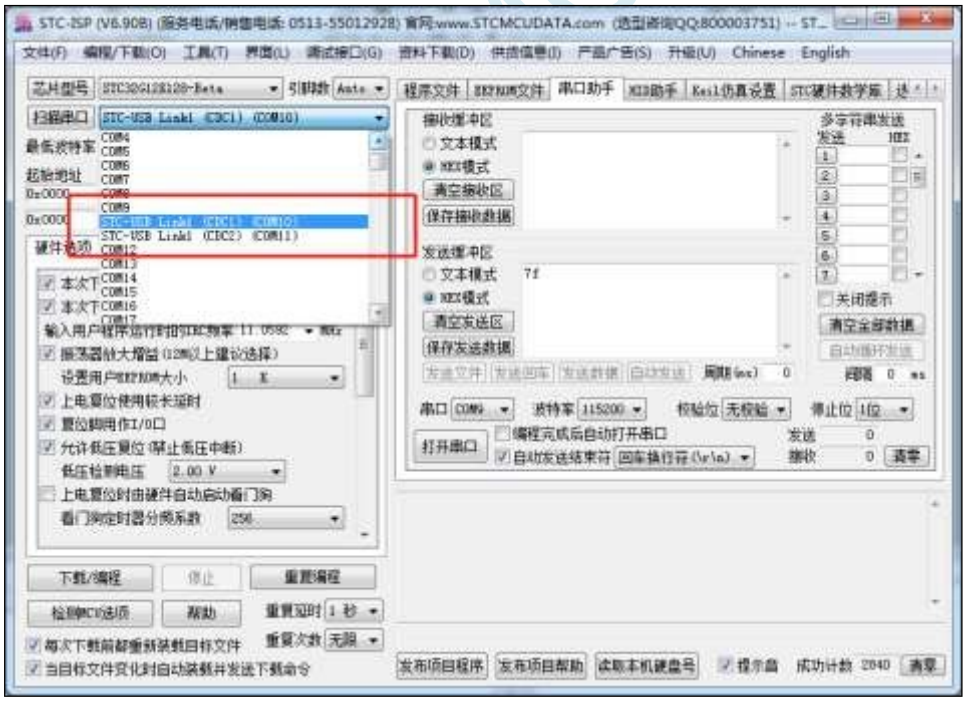
9、After the driver is successfully installed, the following screen will be displayed



10. The installation method of the second CDC virtual serial port driver is similar to the first one. When the driver for both virtual serial ports are installed, you can find the STC-USB LInk1D virtual serial port with the driver installed in both Device Manager and STC-ISP



software. (STC-ISP download software may need to click the "Scan for Serial Ports" button to



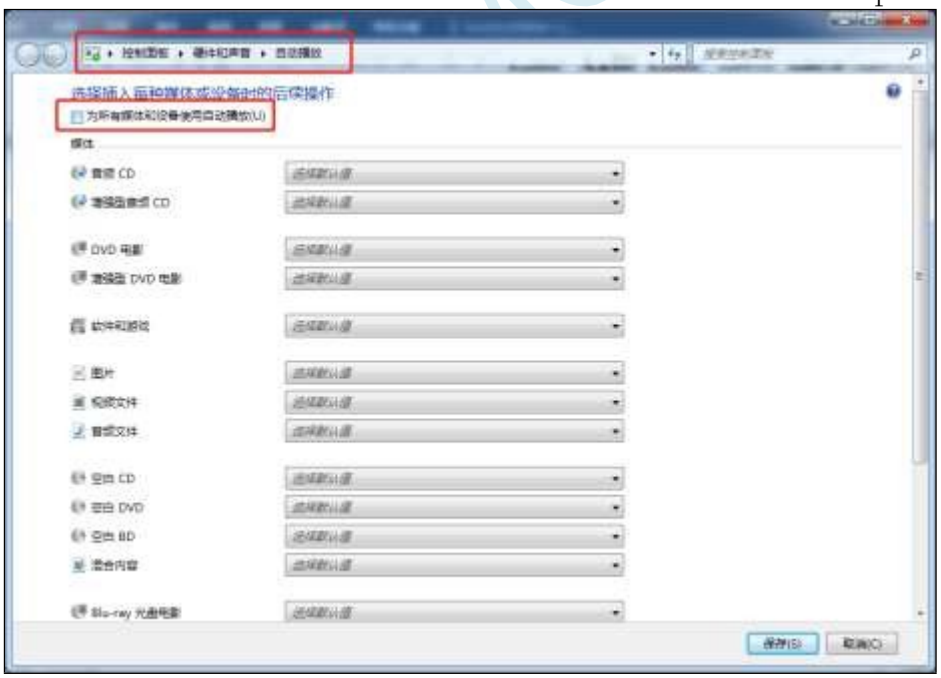
rescan to find the serial port).

5.12.7 How to turn off VirtualDrive autoplay pop-ups

The STC-USB Link1D is a multi-USB interface combination device, which includes a virtual CD-ROM drive containing the CDC virtual serial port driver, which is convenient for customers to install the driver when using the tool for the first time. After the driver is installed, the virtual CDC will no longer be needed during subsequent use, but there will still be a virtual CDC autoplay pop-up window every time the STC-USB Link1D tool is inserted into a USB port on the computer, as shown in the following figure:

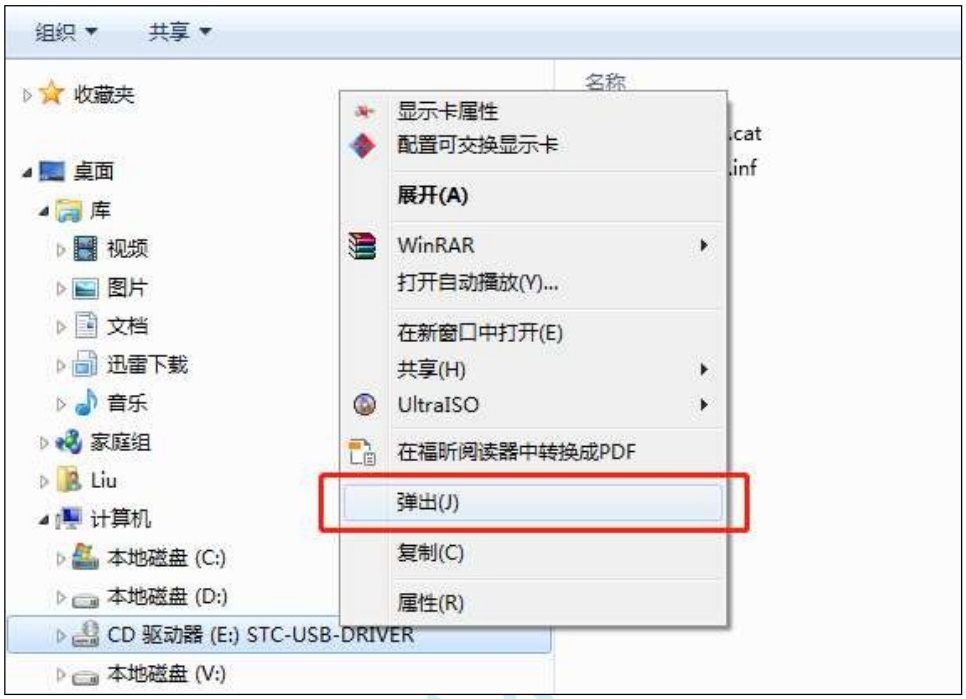
Method 1: Turn off autoplay pop-ups through system settings in Windows

"Control Panel" → "Hardware and Sound" → "Autoplay", remove the tick in front of "Use Autoplay for all media and devices" and then save and exit. Remove the tick in front of "Use AutoPlay for all media and devices" and then save and exit. As shown in the picture below



Method 2: Eject the media device in the virtual optical drive.

Right-click on the virtual CD-ROM icon and click "Eject" in the right-click menu (Note: Ejecting the media means erasing the media data, i.e., permanently deleting the drivers in the media, which cannot be recovered even if the tool is re-powered. The media data in the virtual CD-

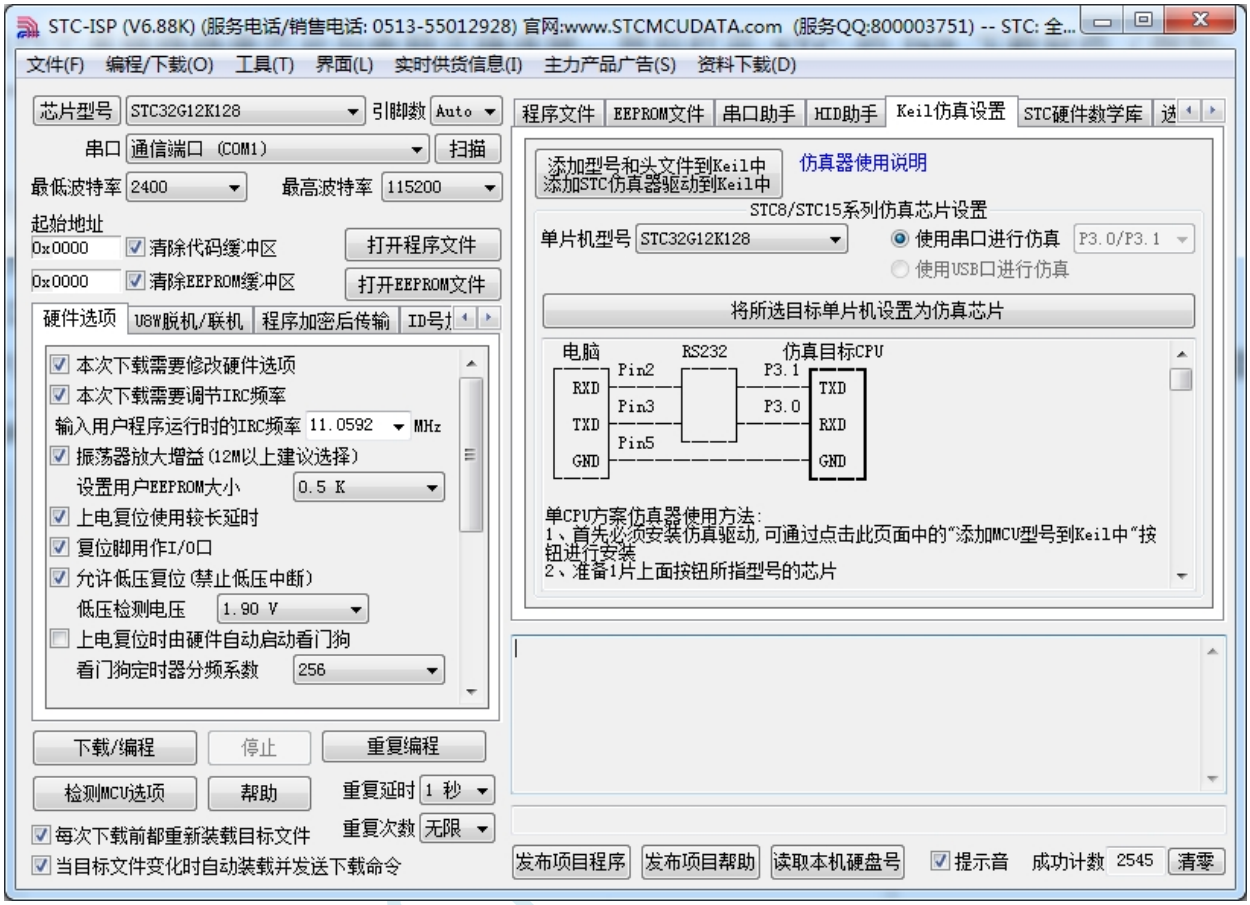


ROM drive will be restored when the Link1 controller chip is rebuilt or when the tool firmware is automatically upgraded.)

Effect after data popup

5.13 Burning with ISP

Firstly, use the serial tool to connect the chip to be burned to the computer correctly, and then open the STC's ISP download software (e.g. "STC-ISP (Ver6.87Q)" and its later versions)



In the above interface, the following points need to be noted:

1, select the microcontroller model to be burned, such as: "STC32G12K128".

2. The serial port must be selected as the serial port number corresponding to the serial port tool.

Click the "Open Program File" button in the interface, and select the file to be downloaded in the dialogue box of opening the program code file: after the file is opened correctly, set the corresponding hardware options, and then click the "Download/Programming" button in the interface to start. Then click "Download/Programming" button in the interface to start the download process:

At this point the ISP software starts trying to handshake with the microcontroller, detecting the target microcontroller

Next, you need to power up the microcontroller, and when a handshake signal is detected during the power-up reset process, it will enter the download process and start downloading. If the download is successful, a prompt screen of successful operation will appear.

5.14 ISP Download instructions for relevant hardware options

Hardware Options	When the option takes effect
☑ 选择使用内部IRC时钟(不选为外部时钟)	Needs to be re-powered to take effect
输入用户程序运行时的IRC频率 11.0592 MHz	Dynamic adjustments with immediate effect
☑ 振荡器放大增益(12M以上建议选择)	Needs to be re-powered to take effect
☑ 使用快速下载模式	Only relevant to this download
设置用户EEPROM大小 0.5 K	Needs to be re-powered to take effect
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序	Valid for next download
☑ 上电复位使用较长延时	Needs to be re-powered to take effect
☑ 复位脚用作I/O口	Needs to be re-powered to take effect
☑ 允许低压复位(禁止低压中断)	Needs to be re-powered to take effect
低压检测电压 2.20 V	Needs to be re-powered to take effect
☑ 低压时禁止EEPROM操作	Needs to be re-powered to take effect
☐ 上电复位时由硬件自动启动看门狗 看门狗定时器分频系数 256 ☑ 空闲状态时停止看门狗计数	Needs to be re-powered to take effect
☑ 下次下载用户程序时擦除用户EEPROM区	Valid for next download
☐ P2.0脚上电复位后为低电平(不选为高电平)	Needs to be re-powered to take effect
☐ 串口1数据线[RxD,TxD]切换到[P3.6,P3.7]	Needs to be re-powered to take effect
☐ TxD脚是否直通输出RxD脚的输入电平	Needs to be re-powered to take effect
☐ P3.7是否为强推挽输出	Needs to be re-powered to take effect
☐ 芯片复位后是否将PWM相关的端口设置为开漏模	Needs to be re-powered to take effect
☐ 在程序区的结束处添加重要测试参数	Write with each download

Needs to be re-powered to take effect: After the option is modified, the target chip needs to be powered down once (blackout) and re-powered again for the new settings to take effect

Dynamic adjustment, immediate effect: valid for this ISP download

Related to this download only: This option is related to this ISP download only and does not affect the next download.

Effective on next download: the option will be effective on the next download after modification, the modification is not effective on this ISP download.

Write with each download: When this option is selected, additional data will be written with this download, independent of the next download.

5.15 User programme reset to system area for USB mode ISP download method (without powering up)

When the project is in the development stage, it is necessary to repeatedly download the user code to the target chip for code verification. To perform normal ISP download to the STC microcontroller in USB mode, it is necessary to short-circuit the P3.2 port to GND first and then re-power up the target chip, which makes the project more tedious to burn the steps in the development stage. For this reason, a special function register IAP_CONTR is added to the STC microcontroller, when the user writes 0x60 to this register, a software reset to the system area can be realised, and the ISP download can be carried out without powering up.

Note: When the user programme is soft reset to the system area, if P3.0/D- and P3.1/D+ are already connected to the USB port of the computer, the system code will automatically enter the USB download mode and wait for the ISP to download, and at this time, it is not necessary to connect P3.2 to the ground.

The following two methods are described below:

1、Using the keys of P3.2 port (non-USB item)

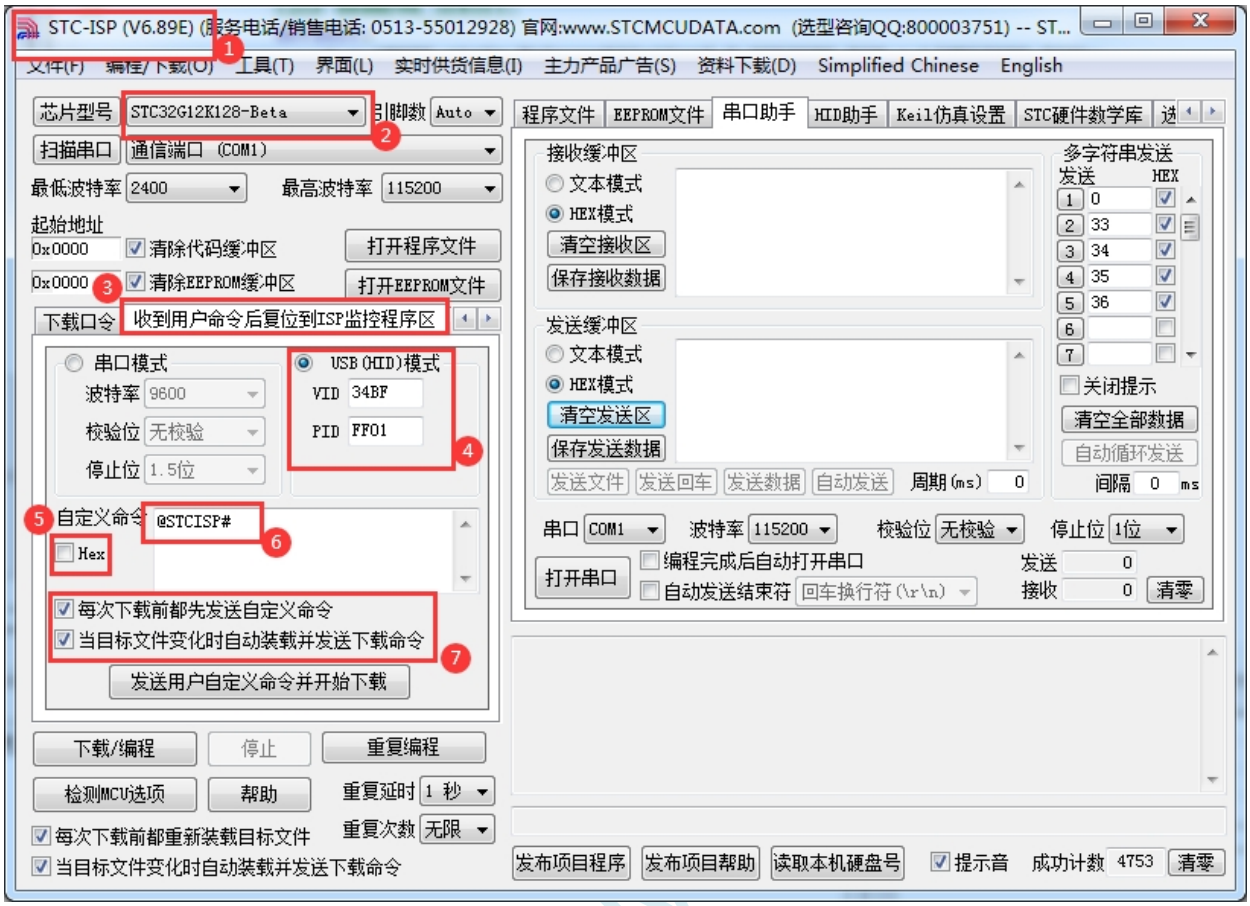
The soft reset triggered by the key of P3.2 port is different from the method of "short-circuiting P3.2 port to GND and then re-powering up the target chip". In the main loop of the user program, the user program judges the level status of the P3.2 port, and when the level of the P3.2 port is detected to be 0, it triggers the software to reset to the system area to carry out the USB ISP download. when the key of the P3.2 port is in the released state, the user program reads the level of 1 from the P3.2 port, and when it needs to be reset to the ISP to carry out the USB download, you only need to press the P3.2 manually.

The following is an example of a program to judge the P3.2 level:

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.

    p3m0 = 0x00;
    p3m1 = 0x00;
    p32 = 1.

    while (1)
    {
        if (!P32) IAP_CONTR = 0x60;     //When the level of P3.2 is detected to be low
```

```
                                        //Software reset to system area
        ...                             //User Code
    }
}
```

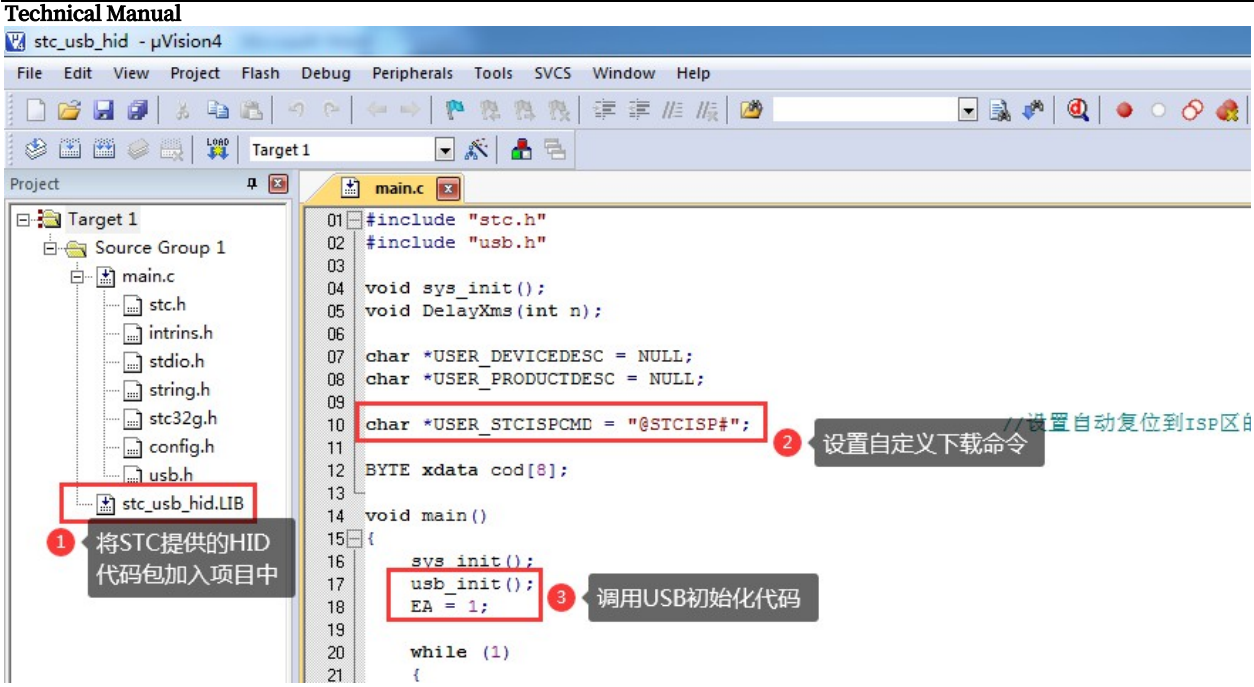
2. User download commands sent using the STC-ISP download software (USB project)



1. Download the latest version of STC-ISP download software.
2、Choose the correct microcontroller model
3. Open the "Reset to the ISP monitor area after receiving a user command" option page.
4, select "USB (HID) mode", and set the VID and PID of the USB device, the VID in the example provided by STC is "34BF", and the PID is "FF01. VID is "34BF" and PID is "FF01" in the example provided by STC.
5. Select HEX mode or Text mode
6, set the custom download command, need to be consistent with the custom command in the code
7. Select these two items, when the target code is recompiled, the STC-ISP download software will automatically send a reset command, and automatically begin to

USB Mode ISP Download

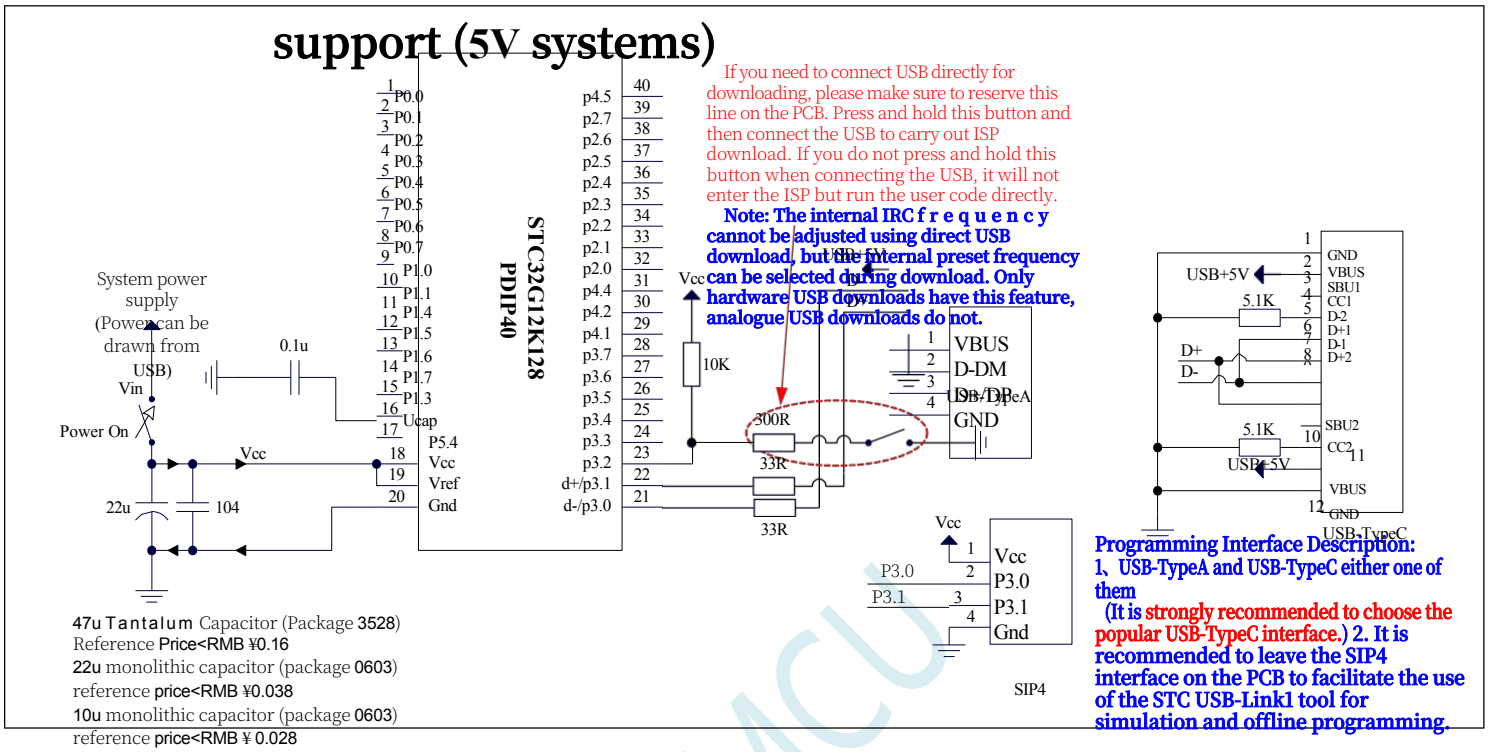
Note: To use this mode, you must add the "stc_usb_hid.lib" codebase provided by STC to the project and set up the custom download command as shown in the following figure.



For detailed code, please refer to "76 - Printing Data Messages via USB HID Protocol - Available for Debugging" in the "STC32G Lab Kit Demo Program" package on the official website.

5.16 ISP Download Typical Application Circuit Diagram

5.16.1 Hardware USB direct ISP download, emulation follow-up support (5V systems)



Now STC's MCUs with hardware USB support downloading of user programs with the included hardware USB because it uses the USB-HID communication protocol and does not require any driver installation.

ISP Download Steps:

1、D-/P3.0, D+/P3.1 and PC-USB port are connected.

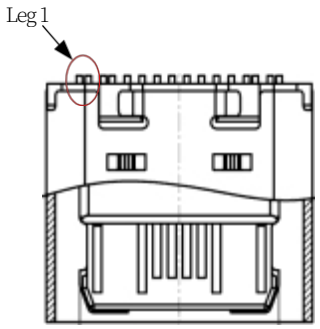
2、Shorten P3.2 and GND, press the P3.2/INT0 button on the board of the laboratory box.

3. Repower the target chip. If the target chip has been powered off, it can be powered on directly; if the target chip is powered on, it needs to be powered off and powered on again (cold start). Wait for the STC-ISP download software to automatically identify the "STC USB Writer (HID1)" identified, it has nothing to do with the state of P3.2 (at this time there is no need to keep pressing the P3.2 port, keep pressing the handle is tired, it does not matter, will be a problem to break the button).

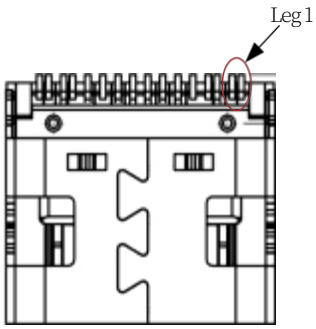
4, click the download software in the "download / programming" button (Note: USB download and serial port download the order of operation is different, **do not click the download button, be sure to wait until the computer recognises the "STC USB Writer (HID1)" device before starting to download) "(Note: The order of operation of USB download is different from that of serial port download.)**

When users use hardware USB to download the ISP for STC32G12K128 series, they can not adjust the frequency of internal IRC, but users can select the 16 frequencies that are preset internally when the chip is shipped from the factory (5.5296M,6M,11.0592M,12M,18.432M,20M,22.1184M, 24M, 27M, 30M, 33.1776M, 35M, 36.864M, 40M, 44.2368M and 48M, which may be different in

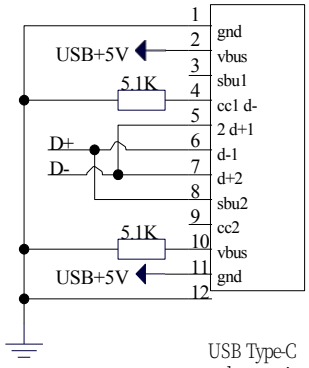
different series according to the list of frequencies in the download software), 33.1776M, 35M, 36.864M, 40M, 44.2368M and 48M, which may vary from series to series, depending on the frequency list of the download software). When downloading, users can only select one of the frequencies from the frequency drop-down list, and cannot manually enter other frequencies. (If you use the serial port to download, you can input any frequency between 4M and 48M).



USB Type-C
Physical
picture
(front)



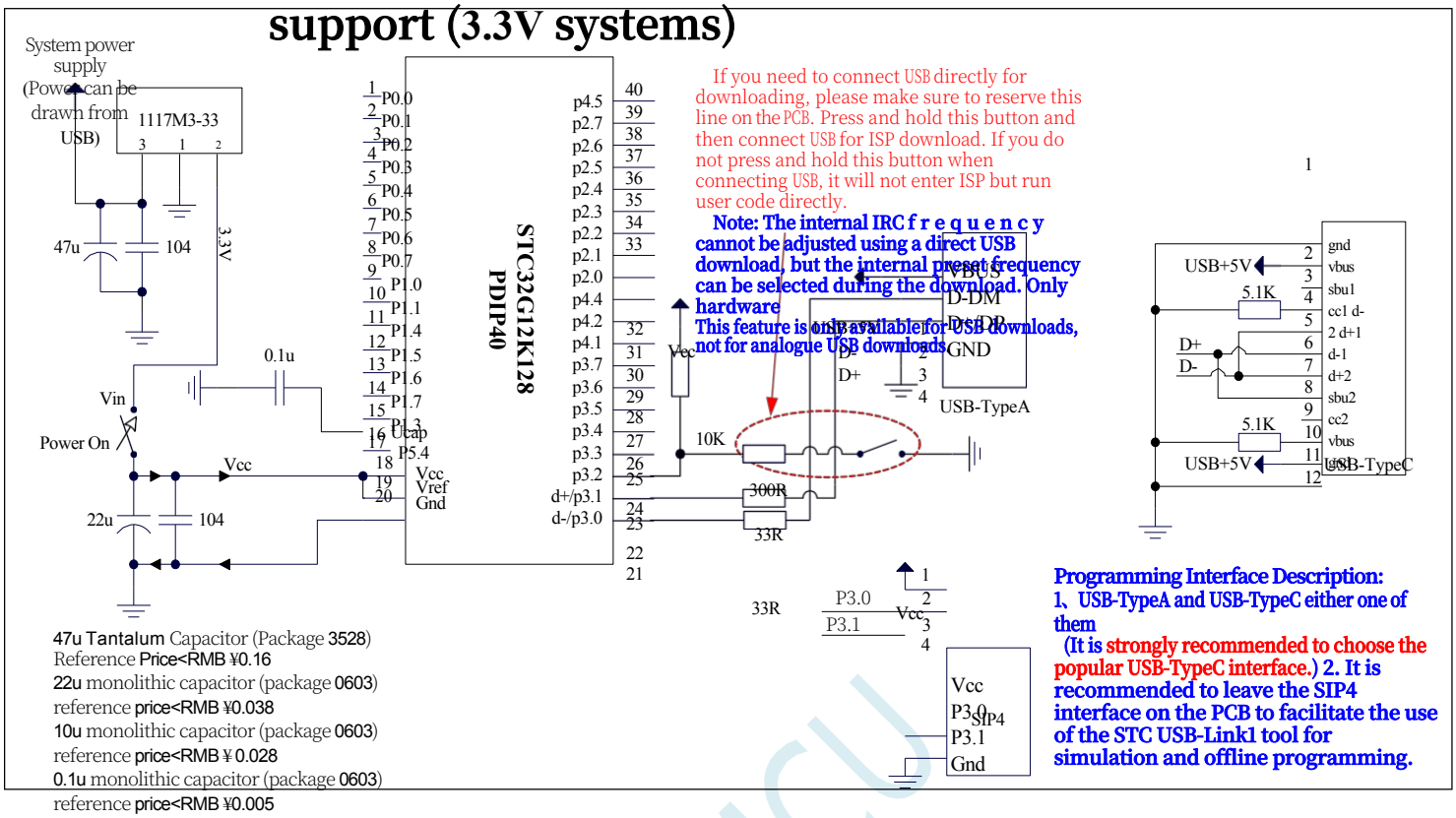
USB Type-C
Physical
drawing
(reverse)



USB Type-C
schematic

	A1	B12	A4	B9	A8	A5	B7	A6	A7	B6	B8	B5	B4	A9	B1	A12
	GND	GND	VBUS	VBUS	SBU1	CC1	D-2	D+1	D-1	D+2	SBU2	CC2	VBUS	VBUS	GND	GND
Type-C 16PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Type-C 12PIN	1		2		3	4	5	6	7	8	9	10	11		12	

5.16.2 Hardware USB direct ISP download, emulation follow-up support (3.3V systems)



47u Tantalum Capacitor (Package 3528)
Reference Price<RMB ¥0.16
22u monolithic capacitor (package 0603)
reference price<RMB ¥0.038
10u monolithic capacitor (package 0603)
reference price<RMB ¥0.028
0.1u monolithic capacitor (package 0603)
reference price<RMB ¥0.005

Now STC's MCUs with hardware USB support downloading of user programs with the included hardware USB because it uses the USB-HID communication protocol and does not require any driver installation.

ISP Download Steps:

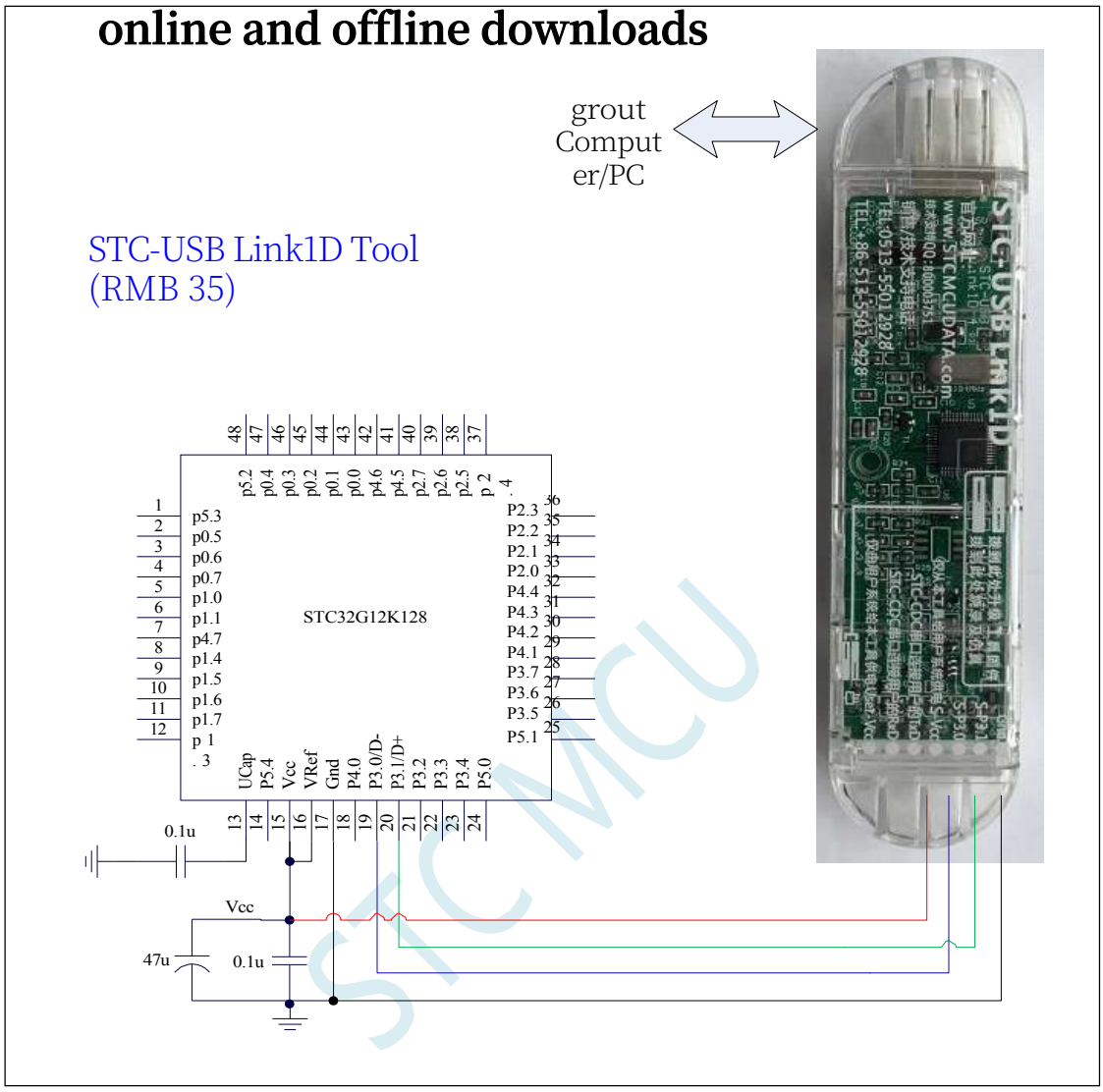
1、D-/P3.0, D+/P3.1 and PC-USB port are connected.

2、Shorten P3.2 and GND, press the P3.2/INT0 button on the board of the laboratory box.

3. Repower the target chip. If the target chip has been powered off, it can be powered on directly; if the target chip is powered on, it needs to be powered off and powered on again (cold start). Wait for the STC-ISP download software to automatically identify the "STC USB Writer (HID1)" identified, it has nothing to do with the state of P3.2 (at this time there is no need to keep pressing the P3.2 port, keep pressing the handle is tired, it does not matter, will be a problem to break the button).

4, click the download software in the "download / programming" button (Note: USB download and serial port download the order of operation is different, **do not click the download button first, be sure to wait until the computer recognises "STC USB Writer (HID1)" device before starting to download) "(Note: The order of operation of USB download is different from that of serial port download.)**

5.16.3 Download using the STC-USB Link1D utility, supports online and offline downloads



ISP Download Steps:

1. Connect the STC-USB Link1D to the target chip as shown in the figure.
2. Click the "Download/Programming" button in the STC-ISP download software.
3. Start ISP download

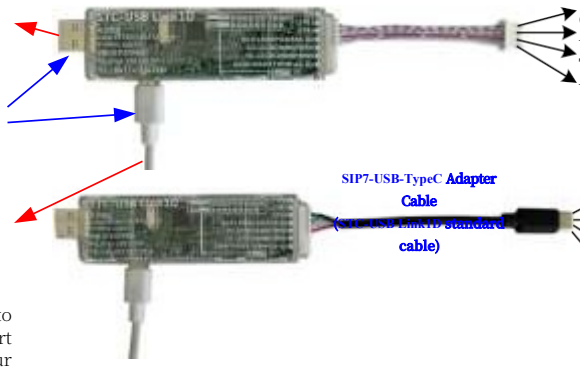
Note: If the STC-USB Link1D is used to supply power to the target system, the total current of the target system must not be greater than 200mA, otherwise the download will fail.

Note: Currently, it has been found that when using USB cable power for ISP download, the voltage drop on the USB cable is too large due to the thin USB cable, resulting in insufficient power supply during ISP download, so please be sure to use a USB booster cable when using USB cable power for ISP download.

Directly connected to the USB port of the computer

The way the tool connects to the computer: either one or the other

Connect to the USB port of your computer via a Type-C to TypeA cable



Connecting the target chip's Gnd
Connecting the target chip's P3.1
Connecting the target chip's P3.0
Tool to power the target system

This connection powers the tool to the target system, and the tool automatically de-energises and reenergises the target system during ISP downloads.

Generally, you can choose these two connections when you use the tool for ISP online download, ISP offline download, or online emulation.

Connecting the Gnd of the target chip Connecting the P3.1 of the target chip Connecting the P3.0 of the target chip Tool to power the target system chip

This type of connection is functionally identical to the previous one, except that it can be connected directly to the Type-C port reserved for the target chip.

Connect the target chip's Gnd Connect the target chip's P3.1 Connect the target chip's P3.0
Separate power supply for tools and target systems

This connection provides independent power to the tool and the target system, and ISP downloads require manually de-energising and re-energising the target system. This connection is required when the power consumption of the target system is greater than 500mA.

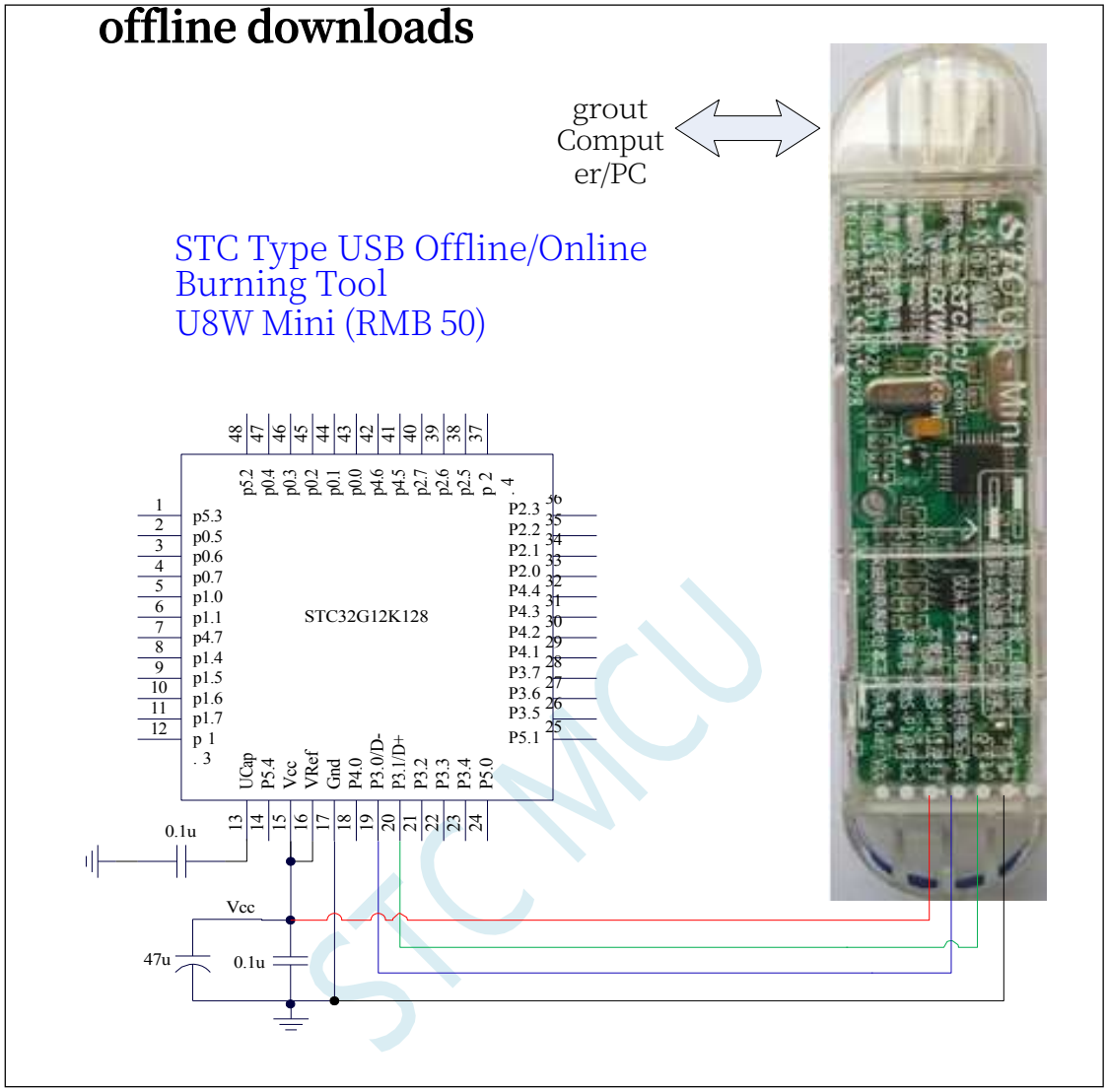
Connecting the target chip's Gnd
Connecting the target chip's P3.1
Connecting the target chip's P3.0
The target system powers up the tool

This connection powers the tool for the target system. This connection is usually chosen when you need to use the tool for ISP offline download (the tool is not connected to the computer).

STC provides this cable to facilitate direct hardware USB downloads or software emulated USB downloads of target chips on user boards with only the SIP4 interface reserved.

This cable with USB Type-A male connector on both ends is a standard cable, and STC will also provide this cable to facilitate hardware USB direct download or software emulation of USB download for target chips with USB Type-A female sockets reserved on the user's system board.

5.16.4 Download with U8-Mini tool, supports ISP online and offline downloads

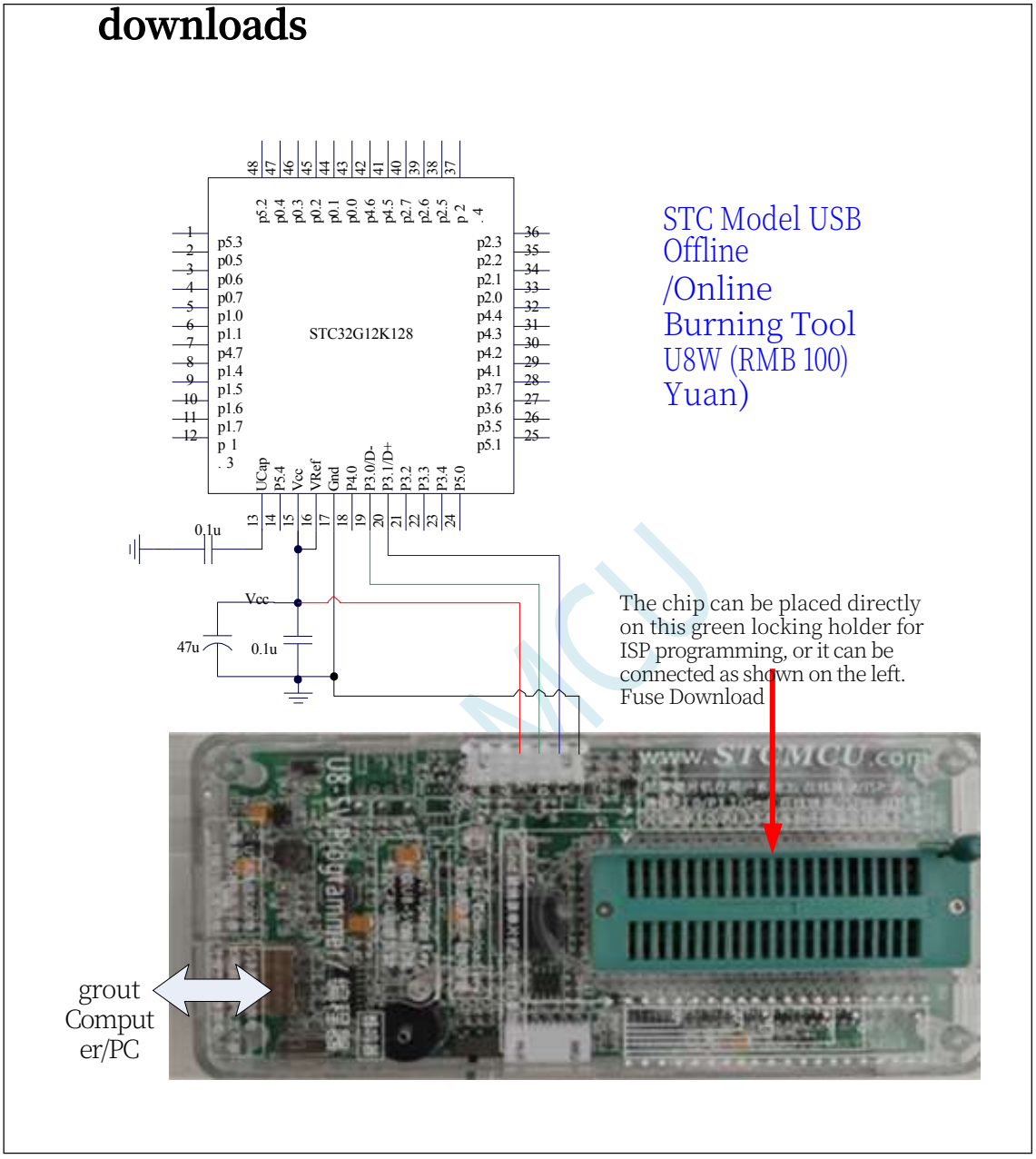


ISP Download Steps:

4. Connect the U8-Mini to the target chip as shown in the figure.
5. Click the "Download/Programming" button in the STC-ISP download software.
6. Start ISP download

Note: If you use U8-Mini to supply power to the target system, the total current of the target system should not be greater than 200mA, otherwise the download will fail. **Note: It has been found that when using the USB cable for ISP download, the voltage drop on the USB cable is too large due to the thin USB cable, resulting in insufficient power supply during ISP download, so please be sure to use a USB booster cable when using the USB cable for ISP download.**

5.16.5 Download with U8W tool, supports ISP online and offline downloads



ISP Download Procedure (Connected mode):

1. Connect the U8W to the target chip as shown in the figure.
2. Click the "Download/Programming" 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 must not be greater than 200mA, otherwise the download will fail.

ISP download procedure (on-board method):

1. Place the target chip in the direction that pin 1 is close to the locking spanner and the pins are aligned downwards.
2. Click the "Download/Programming" button in the STC-ISP download software to start the ISP download.

5.16.6 U8W Straight-through mode for emulation, serial communication

To use U8W for emulation, you must first set U8W to pass-through mode.The U8W/U8W-Mini implements the USB-to-serial port pass-through mode as follows:

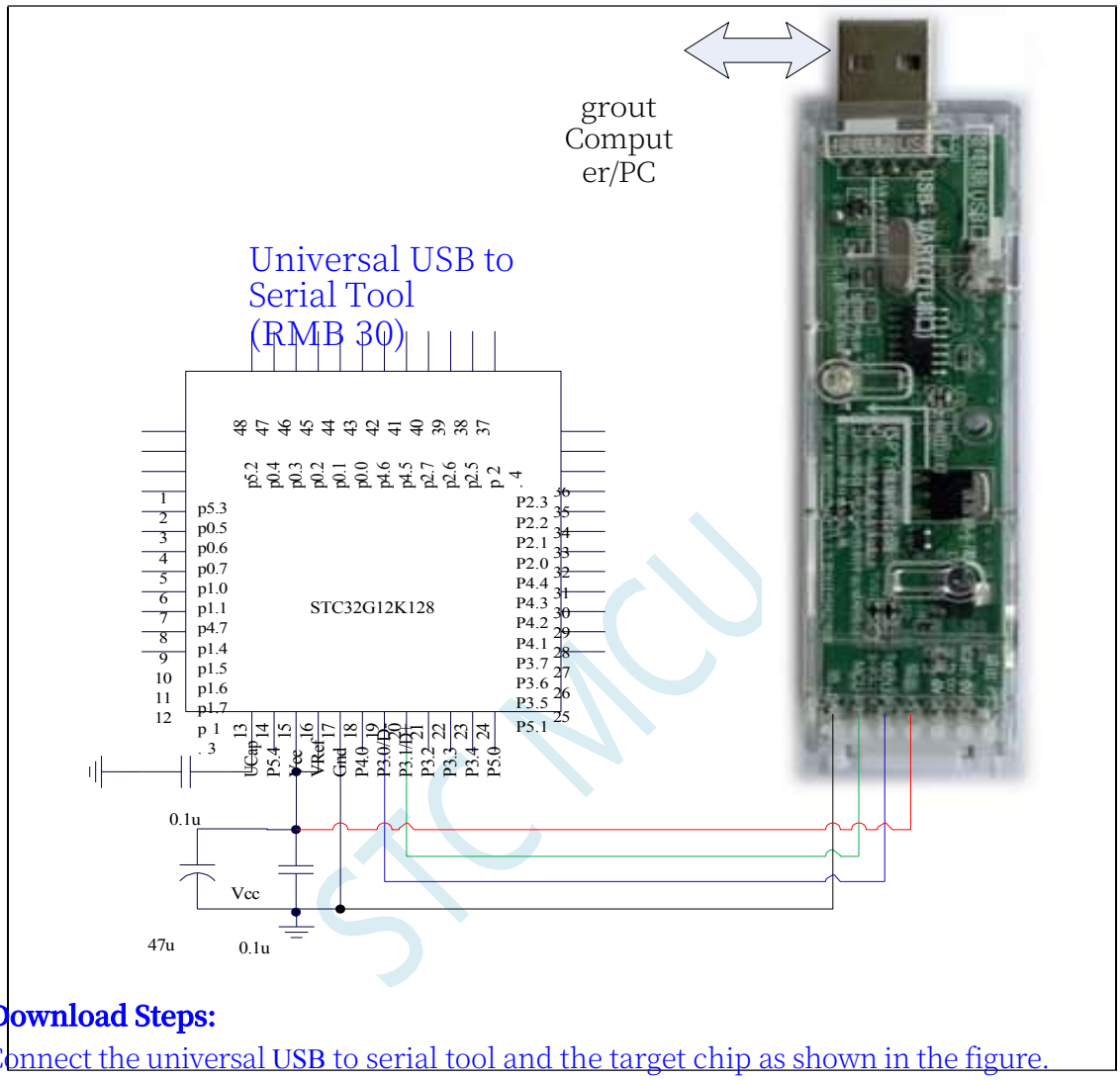
1、 First of all, U8W/U8W-Mini firmware must be upgraded to v1.37 and above.

2、 U8W/U8W-Mini is in normal download mode after powering up, at this time, press and hold the Key1 (download) button on the tool and don't release it, then press the Key2 (power) button, and then release the Key2 (power) button, then release the Key1 (download) button, the U8W/U8W-Mini will be in the USB-to-serial passthrough mode. (**Press Key1 → Press Key2 → Release Key2 → Release Key1**)

3, into the pass-through mode of the U8W/U8W-Mini tool is just a simple USB to serial port does not have offline download function, if you need to restore the

The original function of the U8W/U8W-Mini can be accessed by pressing the Key2 (power) button again separately.

5.16.7 Download Universal USB to Serial Tool



ISP Download Steps:

1、 Connect the universal USB to serial tool and the target chip as shown in the figure.

2. Press the power button to make sure that the target chip is in a **power-off state** (the power-up indicator is off). **Note: The tool is not externally powered the first time it is powered up, so you can skip this step if it is the first time you power up and use the tool.**

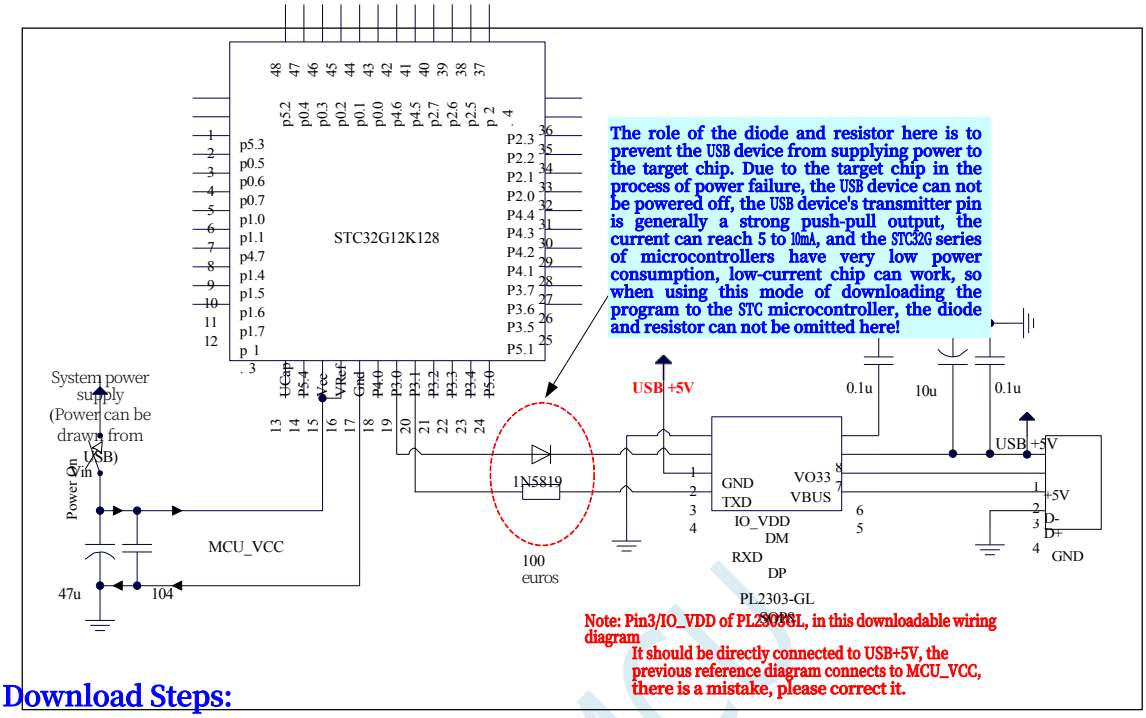
3. Click the "Download/Programming" button in the STC-ISP download software.

4. Press the power button again to power up the target chip (the power-up indicator is on).

5. Start ISP download

Note: Currently, it has been found that when using USB cable power for ISP download, the voltage drop on the USB cable is too large due to the thin USB cable, resulting in insufficient power supply during ISP download, so please be sure to use a USB booster cable when using USB cable power for ISP download.

5.16.8 Downloading with PL2303-GL



ISP Download Steps:

1. Power down the target chip, pay attention to not power down the USB-to-serial chip (e.g. CH340, PL2303-GL, etc.).
2. because the USB to serial chip's transmitter pin is generally a strong push-pull output, you must connect a diode in series between the target chip's P3.0 port and the USB to serial chip's transmitter pin, otherwise the target chip can not be completely disconnected from the power supply, and can not achieve the goal of the target chip to stop the power.
3. Click the "Download/Programming" button in the STC-ISP download software.
4. Power up the target chip
5. Start ISP download

Note: Currently, it has been found that when using USB cable power for ISP download, the voltage drop on the USB cable is too large due to the thin USB cable, resulting in insufficient power supply during ISP download, so please be sure to use a USB booster cable when using USB cable power for ISP download.

5.17 STC-ISP Download Software Advanced Application

5.17.1 Publishing project procedures

Publishing the project programme function is mainly to package the user's programme code and related option settings into a super-simple user-interface executable file that can be downloaded and programmed directly to the target chip.

Regarding the interface, the user can customize it by himself (the user can modify the title, button name and help information of the released project), and at the same time, the user can also specify the hard disc number of the target computer and the ID number of the target chip. After specifying the hard disc number of the target computer, the user can control the released application to be run only in the specified computer (to prevent the program from being stolen from the computer easily by the burners, such as sending it away through the network, such as baking it away through the USB stick, to prevent it from being stolen from your computer. Send away, such as through the U disk baked away, can not be prevented, of course, steal your computer that would not be able to do that, so STC's offline download tool than the computer burning security, can limit the number of chips that can be burned, so that the front desk clerk Miss burn, so that the boss's wife to burn can be), copied to other computers, the application can not be run. Similarly, when the ID number of the target chip is specified, then the user code can only be downloaded to the corresponding ID number of the target chip (for a device to sell tens of millions of dollars of products are particularly useful - tanks, can be sent to the customer's own upgrades, do not need to risk their lives to run to the war in Iraq to upgrade the software), for the ID number is not consistent with the other chip, can not be downloaded for programming.

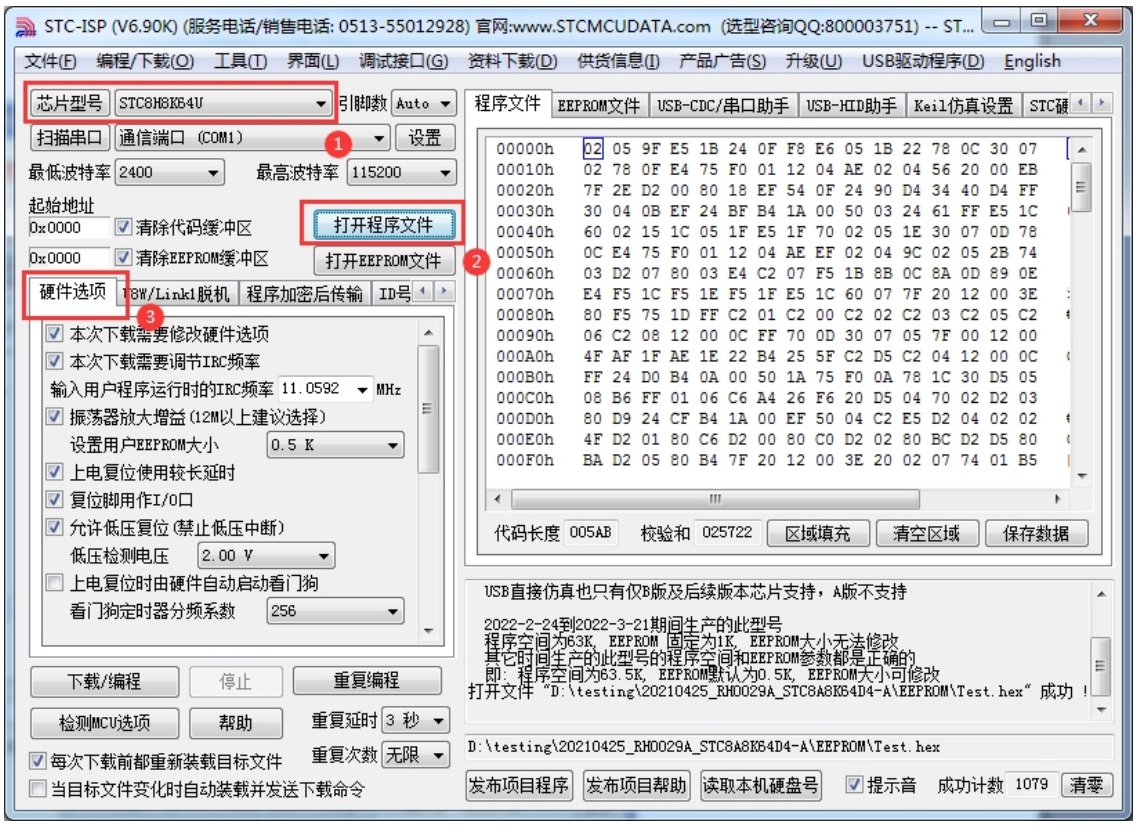
The detailed procedure for posting a project

procedure is as follows:

1, first select the target chip model

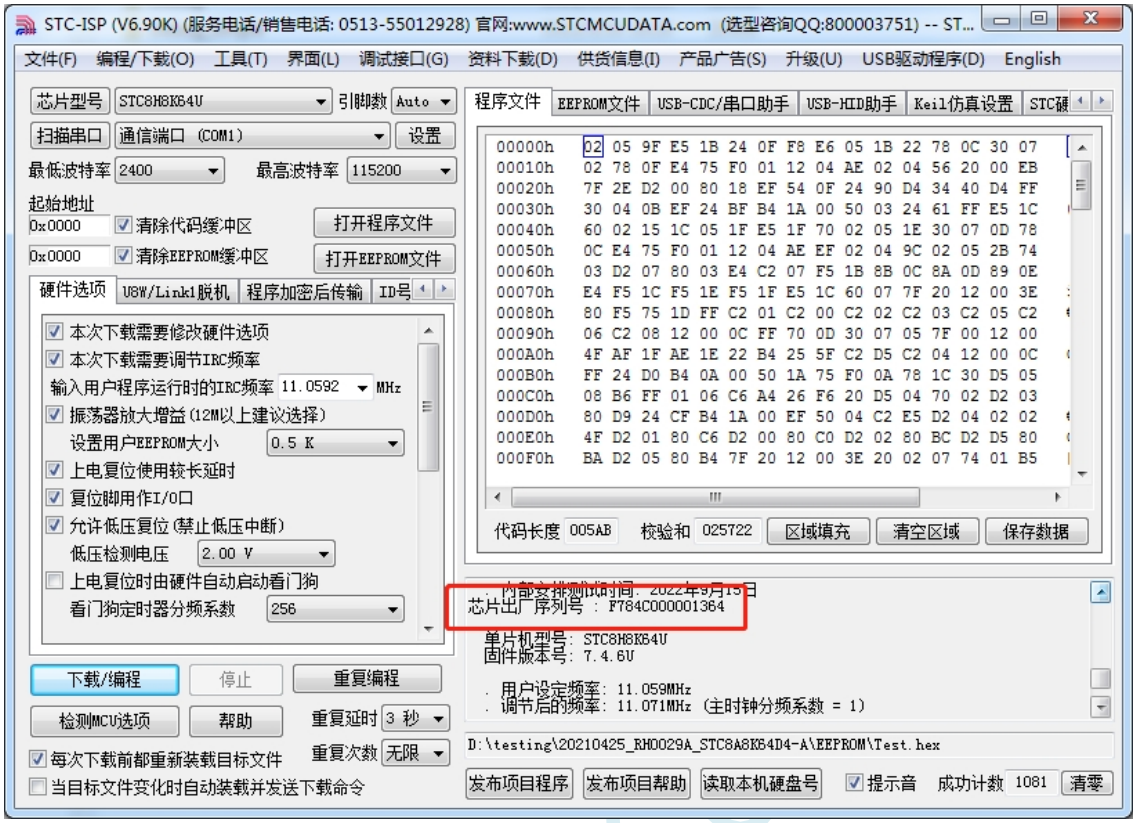
2, open the programme code file

3. Set the appropriate hardware options

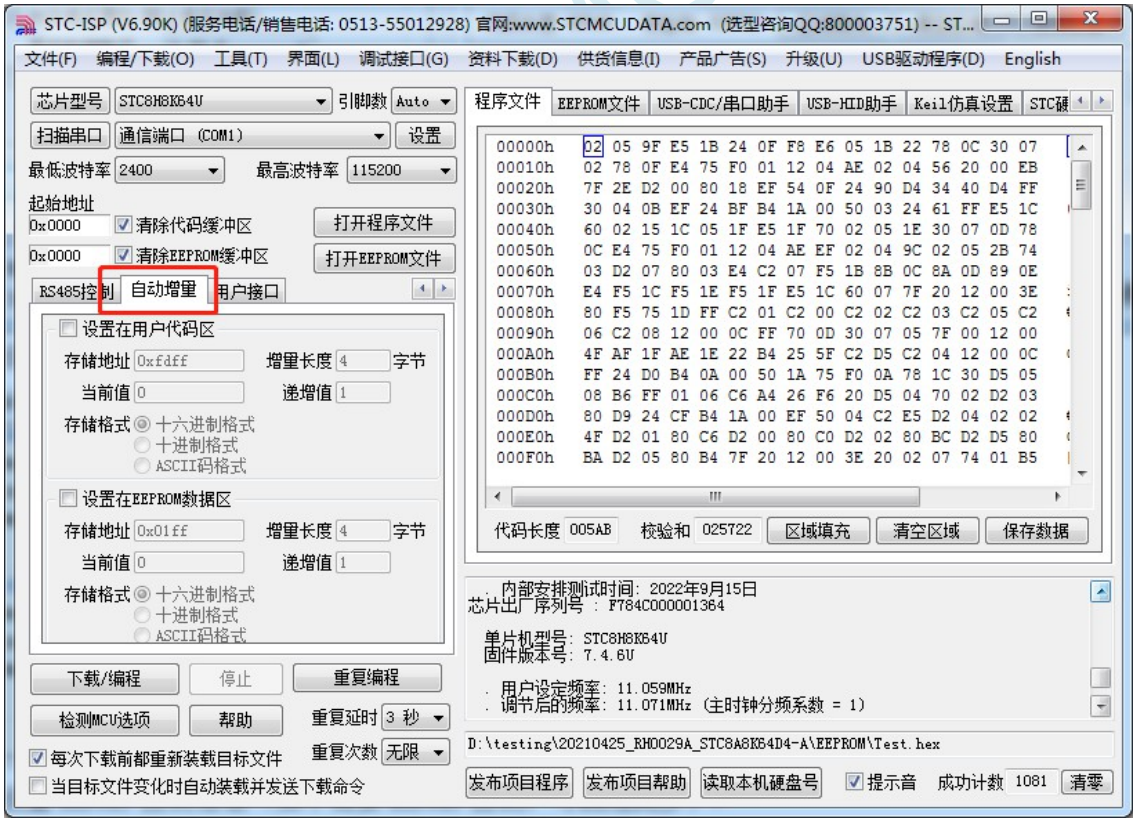


4, try to burn the chip, and note down the ID number of the target chip, as shown in the figure below, the chip's ID number is "F784C000001364".

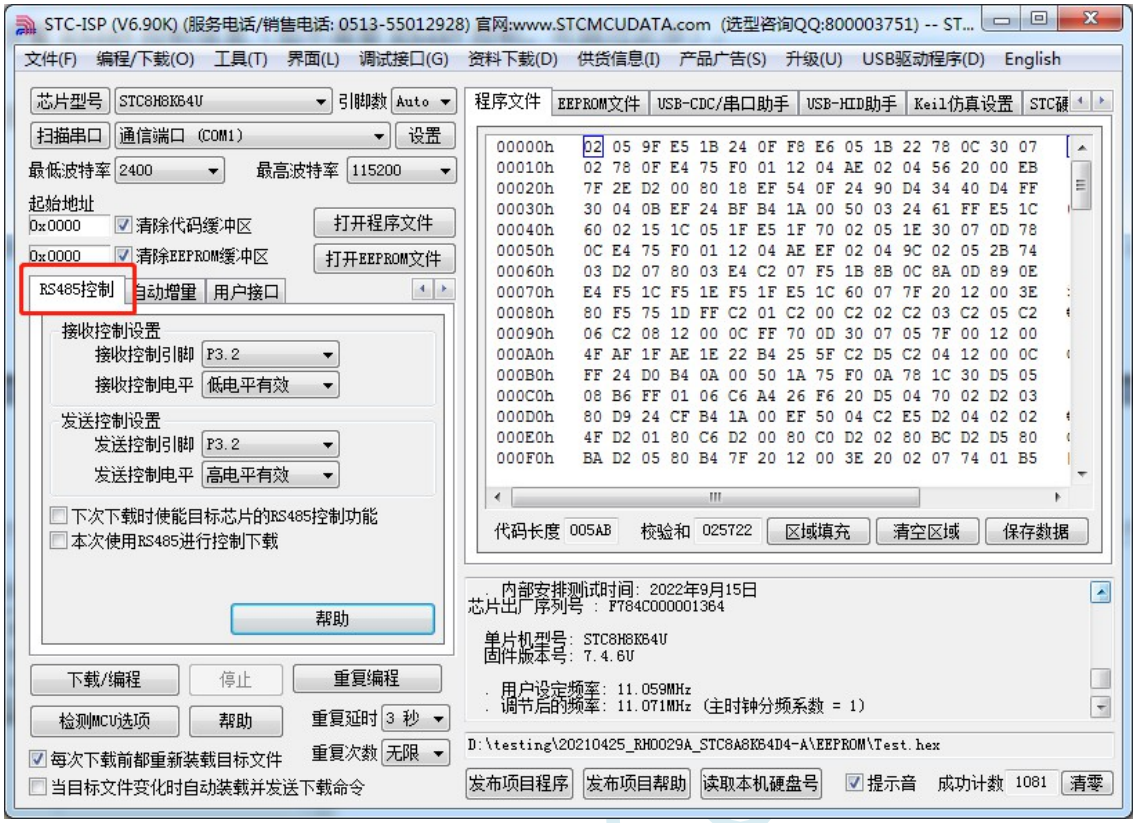
(This step can be skipped if there is no need to verify the ID number of the target chip)



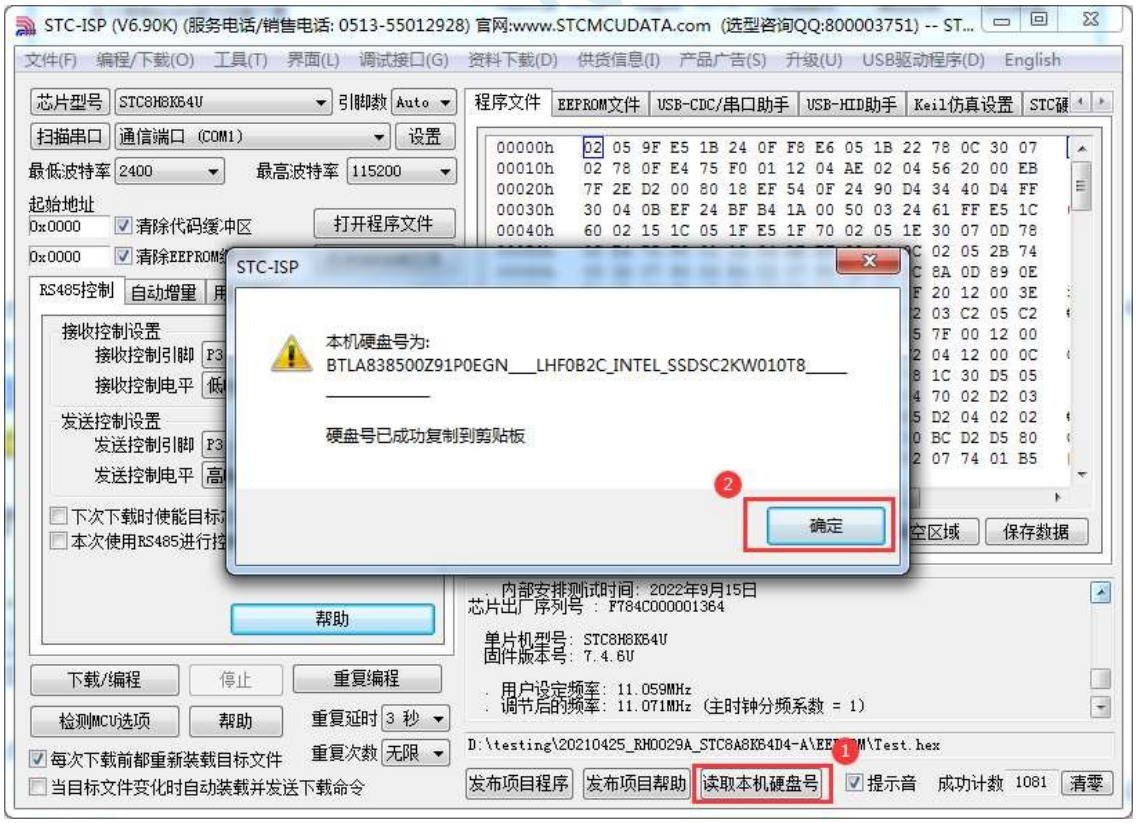
5、Set up automatic increment (if you do not need automatic increment, you can skip this step)



6、Set RS485 control information (if you do not need RS485 control, you can skip this step)



7, click on the interface of the "read the local hard disc number" button, and note down the target computer's hard disc number (such as the target computer does not need to verify the hard disc number, you can skip this step)



8. Click the "Publish Project Procedures" button to enter the publishing application settings interface.

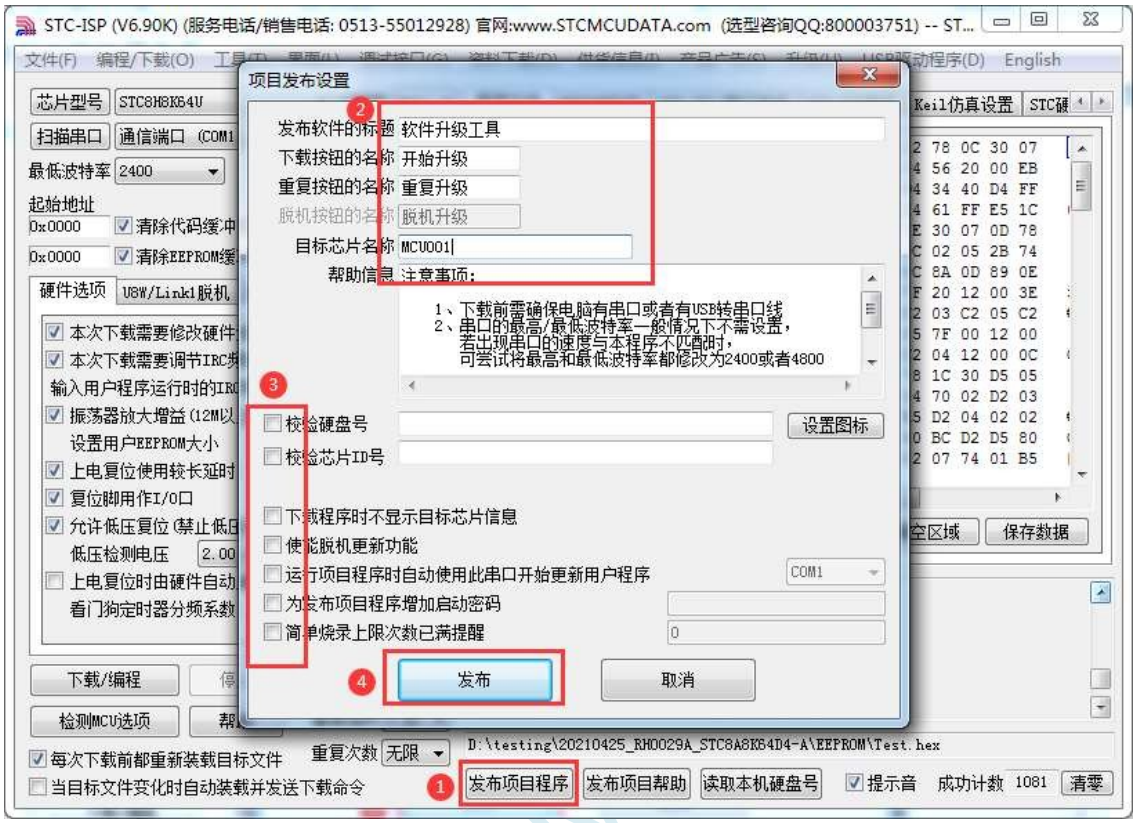
9, according to their respective needs, modify the title of the release software, the name of the download button, the name of the repeat download button, the name of the

automatic increment and help information

10, if you need to verify the target computer's hard disk number, you need to check the "check the hard disk number", and in the back of the text box, enter the previous

The hard disc number of the target computer

11, if you need to verify the ID number of the target chip, you need to check the "verify chip ID number", and enter the ID number of the target chip in the text box behind.



12, and finally click the release button, the project will be released to save the procedure, you can get the corresponding executable file. Published project programme to play the interface as follows

5.17.2 Program encrypted for transmission (to prevent the serial port from analysing the program when burning)

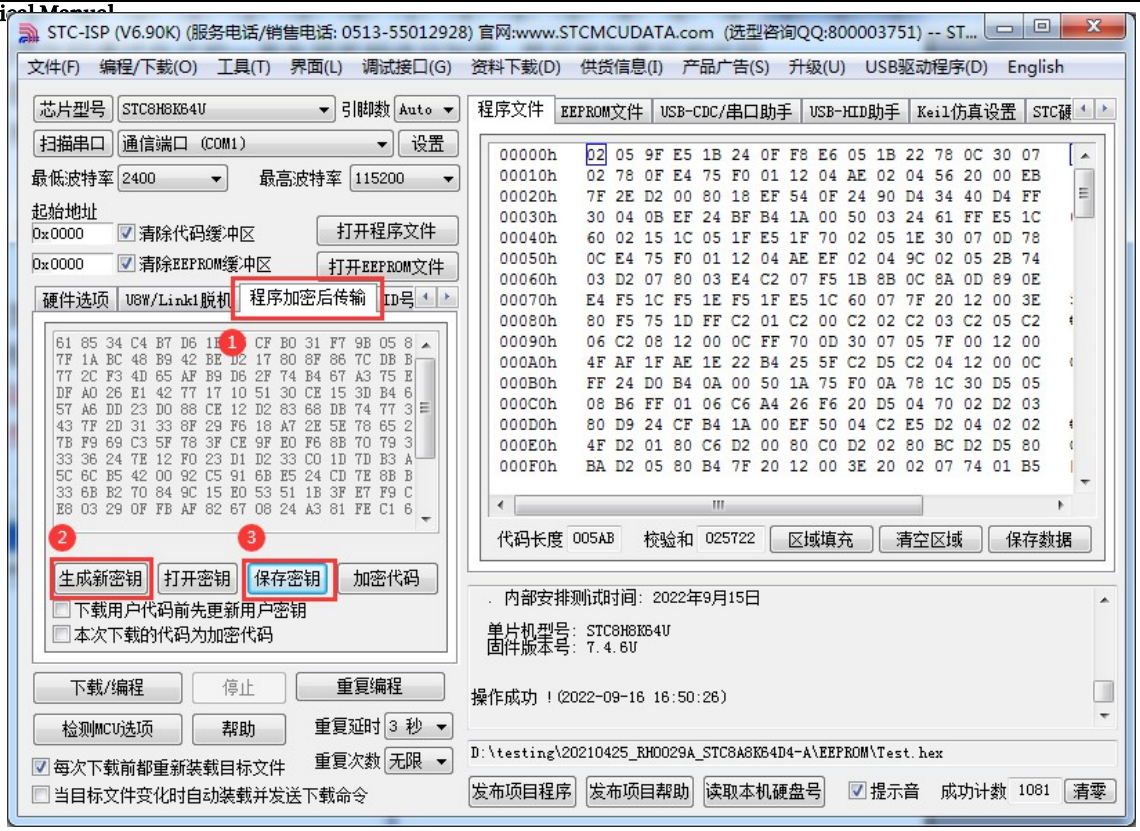
At present, all common serial port download and burn programming are using explicit communication (when the computer and the target chip communicate, or offline download board and the target chip communication), the problem: if the burner by analysing the data of serial port communication when downloading and burning programming, the masters are able to draw 2 wires out of the serial port when burning, and analyse the data of serial port communication to analyse the actual user program code. Of course, using STC's offline download board burn program is always better than using a computer to burn the program (to prevent burning personnel will easily steal the program from the computer, such as through the network to send away, such as through the USB flash drive baked away, can not be prevented, of course, stealing your computer that would not be able to do that, so the STC's offline download tool is safer than the computer to burn the tool to the receptionist clerk to burn the lady to burn the boss's wife to burn it all can be). Even the world's first STC offline download tool, to prevent the genius of the lawless elements in the process of offline download tool burning by analysing the serial port communication data to analyse the actual user program code, there is no way to meet the requirements, which requires the use of the latest STC microcontroller to provide program encryption after the transmission function.

Program encryption after transmission and download is the user first program code through their own set of special keys for encryption, and then the encrypted code and then download through the serial port, at this time the download and transmission of encrypted files, analysed through the serial port is encrypted messy code, such as not by sending someone to sneak into your company to steal the encryption key inside your computer, it is not of any value, it will be able to play a role in the prevention of the program in the burning burner when it is burning the personnel! The purpose of analysing the code by monitoring the serial port.

The following steps are required to use the Transfer after Program Encryption function:

1. Generate and save a new key

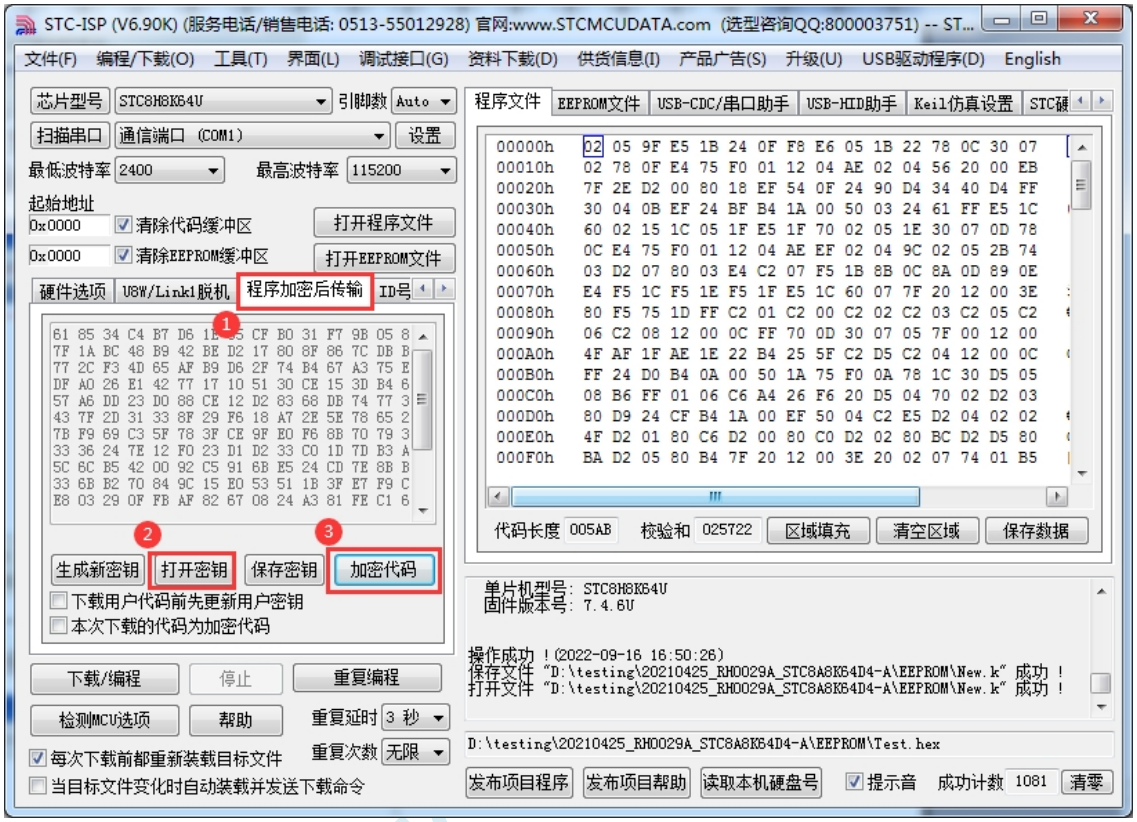
   As shown in the figure below, go to the "Transmission after encryption" page, click "Generate new key" button to display the newly generated 256-byte key in the buffer. Then click "Save Key" button, you can save the new key as a key file with ".K" extension (**Note: this key file must be saved, all future code files need to be encrypted with this key, and the key generation is non-repeating). K" extension (note: this key file must be saved, as it will be used to encrypt all future code releases, and the key is non-repeating, i.e., it is impossible to generate two identical keys at any one time, so it will be impossible to regain the key file if it is lost**). For example, we save the key as "New.k".



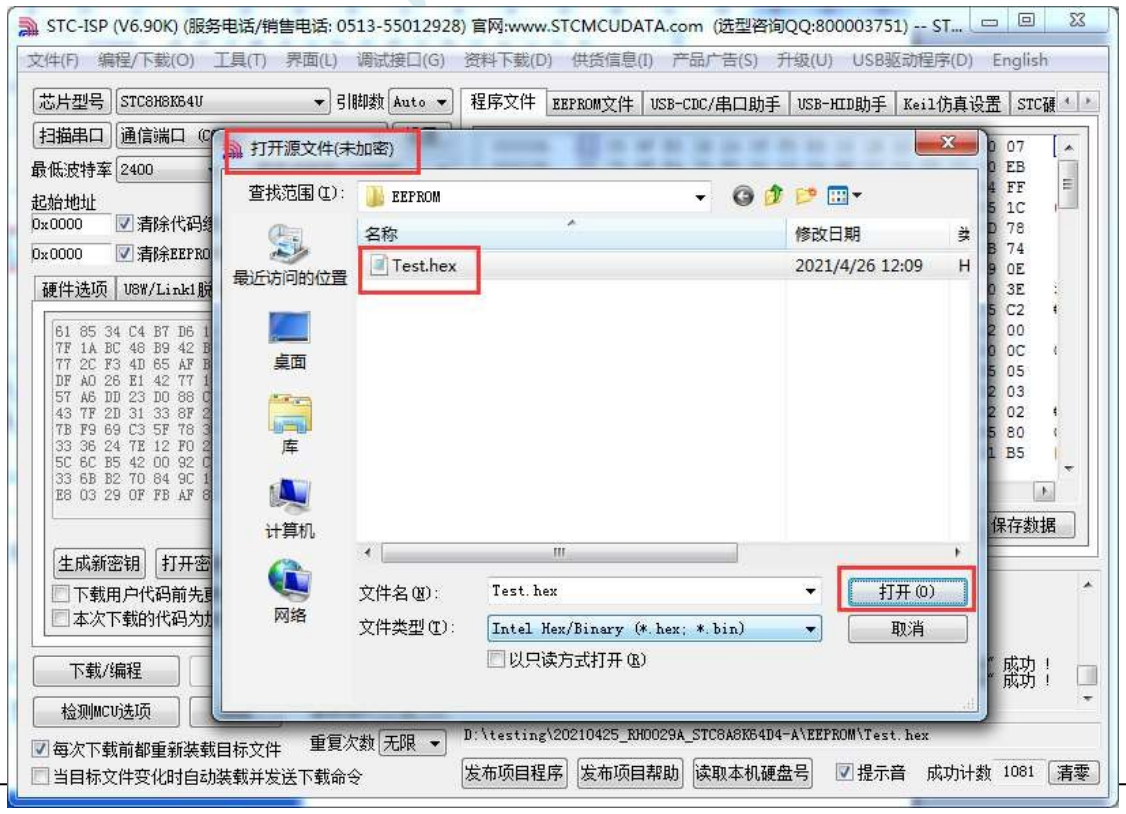
2. Encryption of code files

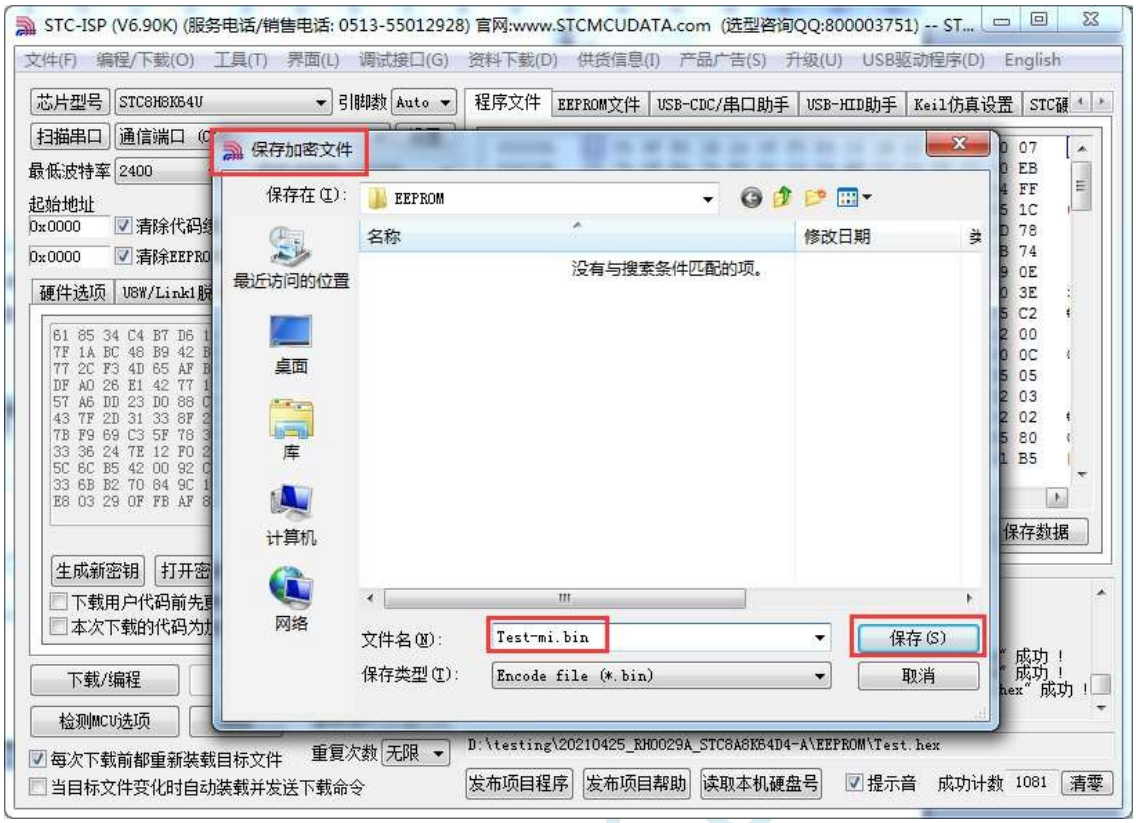
Before encrypting a file, you need to open our own key. If our key is already stored in the buffer, do not open it again. As shown in the figure below, click the "Open Key" button on the "Transfer after encryption" page to open the key file we saved before.

For example, "New.k", and then return to the "Transmission after encryption" page, click the "Encrypt code" button, as shown in the figure below, first of all, it will pop up the dialogue box of "Open source file (unencrypted)". The "Open source file (unencrypted)" dialogue box will pop up, and the original unencrypted code file will be selected.



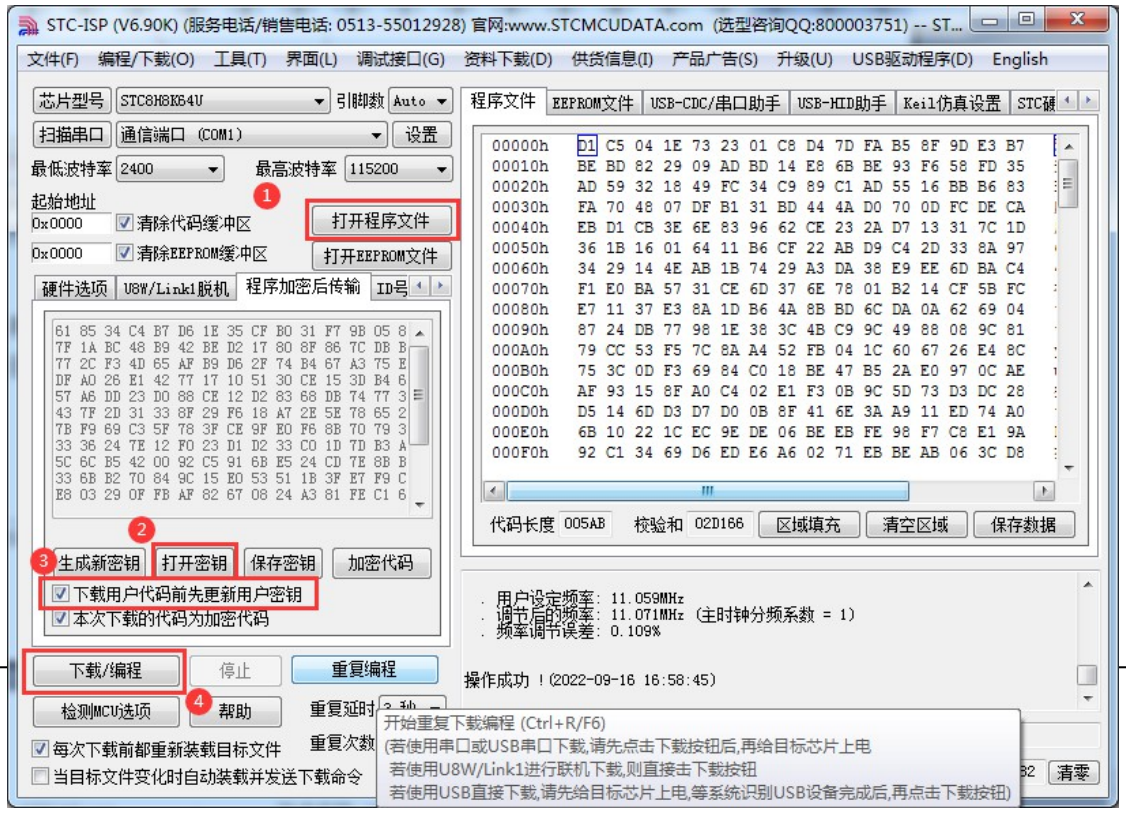
After clicking the Open button, a similar dialogue box will pop up immediately, but this time it is a dialogue box for saving the encrypted file. As shown in the figure below, click the Save button to save the encrypted file.





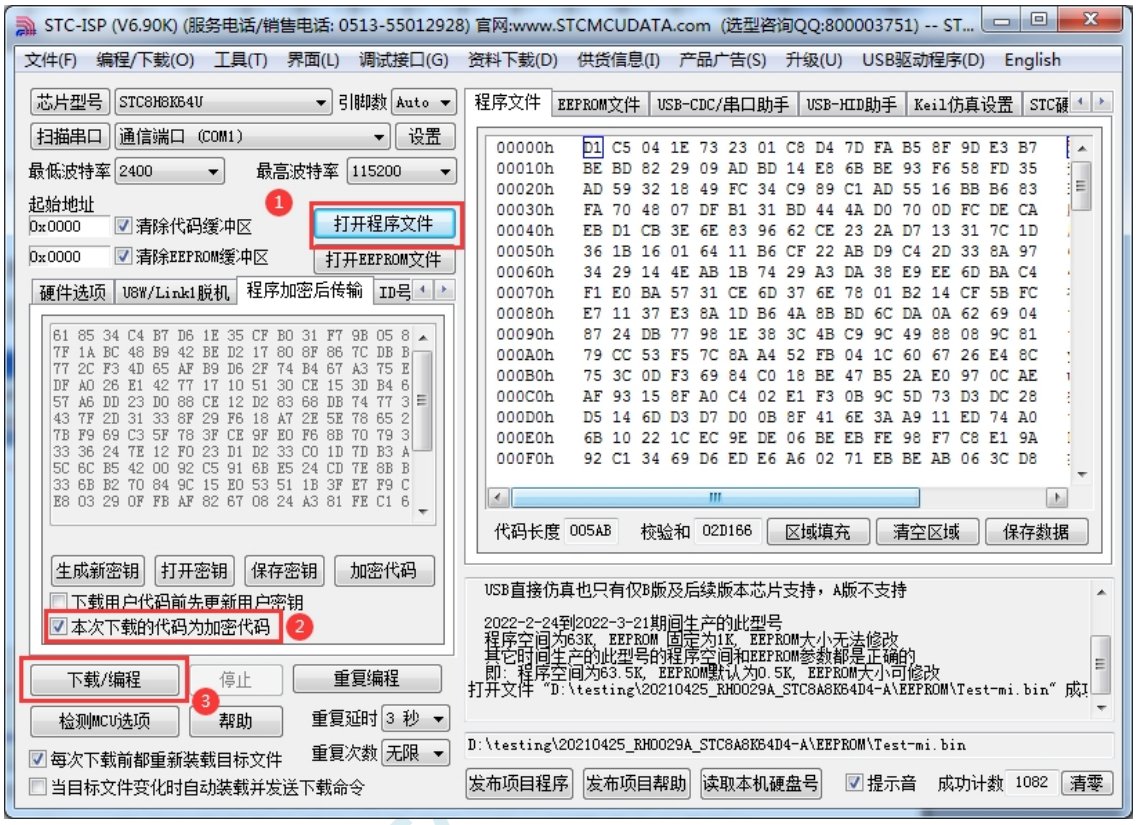
3. Update the user key into the target chip

Before updating the key, we need to open our own key. If our key is already stored in the buffer, do not open it again. As shown in the figure below, click the "Open Key" button in the "Custom Encryption Download" page to open the key file we saved before, for example, "New.k". After the key is opened, as shown in the figure below, tick the options of "Update user key before downloading user code" and "The downloaded code is encrypted code", then open our previously encrypted file, and click "Download/Programming" in the lower left corner of the interface. After opening the encrypted file, click the "Download/Programming" button at the bottom left corner of the interface, and then update the user key after the download of the target chip is completed in the normal way.



4. Encrypted update of user code

After the key is successfully updated, the target chip will have the function of receiving the encrypted code and restoring it. At this time, if you need to upgrade/update the code again, you only need to refer to the method in the second step, encrypt the target code, and then as follows



For a new STC microcontroller, steps 3 and 4 can be combined to complete, i.e., updating the key to the target microcontroller and downloading the encrypted code to the microcontroller at the same time. In the "Transmission after encryption" page, you only need to select the option of "The downloaded code is encrypted code" (the option of "Update the user key before downloading the user code" does not need to be selected), and then open the encrypted file that we have previously opened, and click on the "Download" button in the left corner of the interface. interface in the lower left corner of the "download/programming" button, according to the normal way to download the target chip can be completed with the user's own encrypted file to update the user's code for the purpose of (to prevent the burning of the program by the burning personnel through the monitoring of the serial port to analyse the code for the purpose of).

5.17.3 Combined use of release project program + encrypted transmission of the program

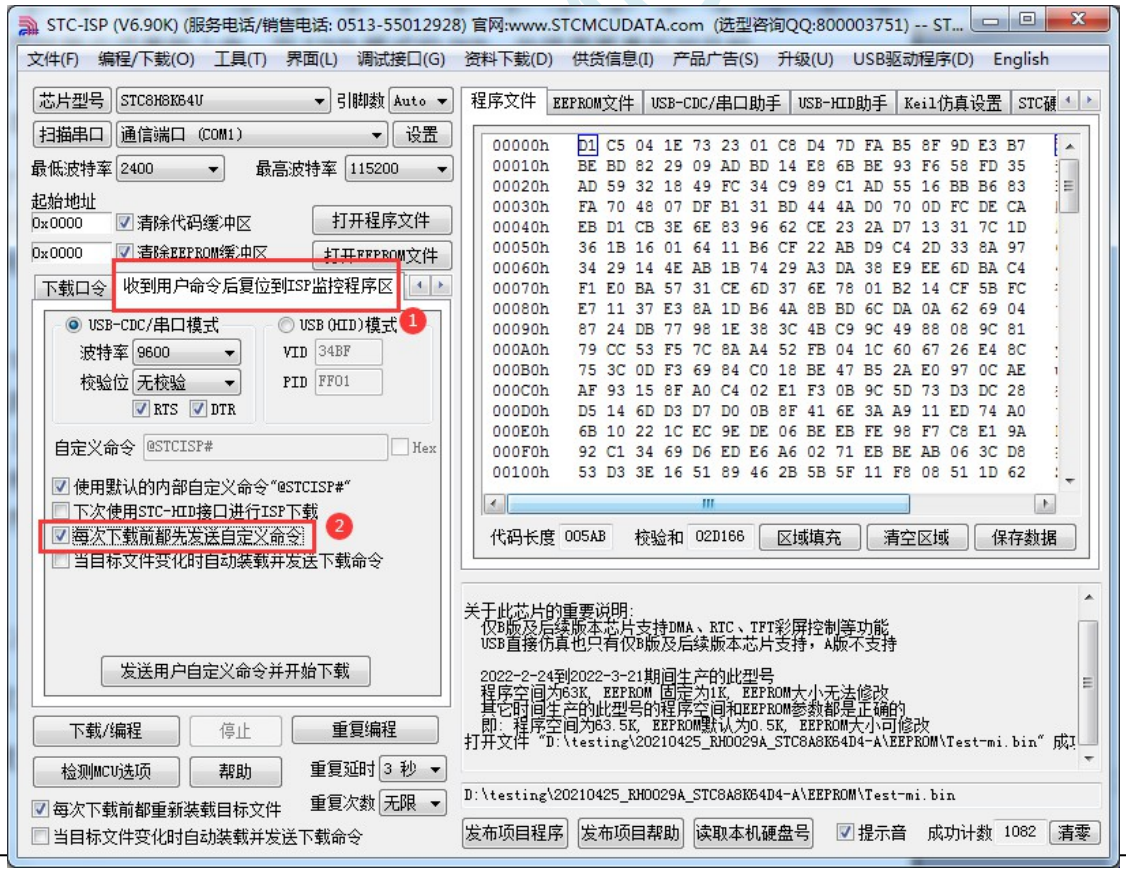
Two new special features can be used in conjunction with the release of the project programme and the encrypted transfer of the programme. Firstly, the encrypted program transfer ensures the confidentiality of the user code during the serial communication transfer when programming, while the release of the project program allows the end-user to update the software remotely (without the need for the solution company's personnel to be physically present). So the combination of the two functions, very suitable for solution companies / manufacturers in the software needs to be updated, so that the end user of the end product for the purpose of software updates, but also to ensure that the site burning personnel can not be analysed through the serial port of the useful procedures, is highly recommended for solution companies to use.

5.17.4 User-defined downloads (to enable non-stop downloads)

Downloading the user's target programme to the STC microcontroller is achieved by executing the internal ISP system code of the microcontroller and communicating with the host computer through the serial port or USB. However, the internal ISP system code of STC microcontroller will only be executed every time when the power is turned off and on again, which requires the user to re-power on the target microcontroller every time the user needs to update the programme, while the ISP in the USB mode requires the user to re-power on the target chip as well as pull down the P3.2 port to GND during the power-on process, which is not only necessary in the project in the development stage but also in the project in the development stage. For projects in the development stage, it is necessary to frequently modify and update the code, and it is very troublesome to re-power on the target chip every time it is downloaded.

STC microcontroller adds a soft reset register (IAP_CONTR) in the hardware design, which allows the user to set this register to decide whether the CPU resets to re-execute the user code or resets to the ISP area to execute the ISP system code. When 0x20 is written to the IAP_CONTR register, the CPU resets and re-executes the user code; when 0x60 is written to the IAP_CONTR register, the CPU resets and resets to the ISP area to execute the ISP system code.

To realise non-stop ISP download, users can design a code in the program, such as detecting a special key, or monitoring the serial port and waiting for a special serial port command, and when the download condition is detected and met, then the software will trigger the soft reset register to reset to the ISP area to execute the ISP system code, so as to realise non-stop ISP download. When the triggering condition is an external key, it is enough to monitor the key status in real time in the user code. In order to synchronise the STC-ISP software with the user-triggered soft reset, it is necessary to use the function "Reset to ISP Monitor Area after Receiving User Command" provided in the STC-ISP software.



The steps to implement a non-stop ISP download are as follows:

1、Write user code and add serial port command monitoring procedure in user code

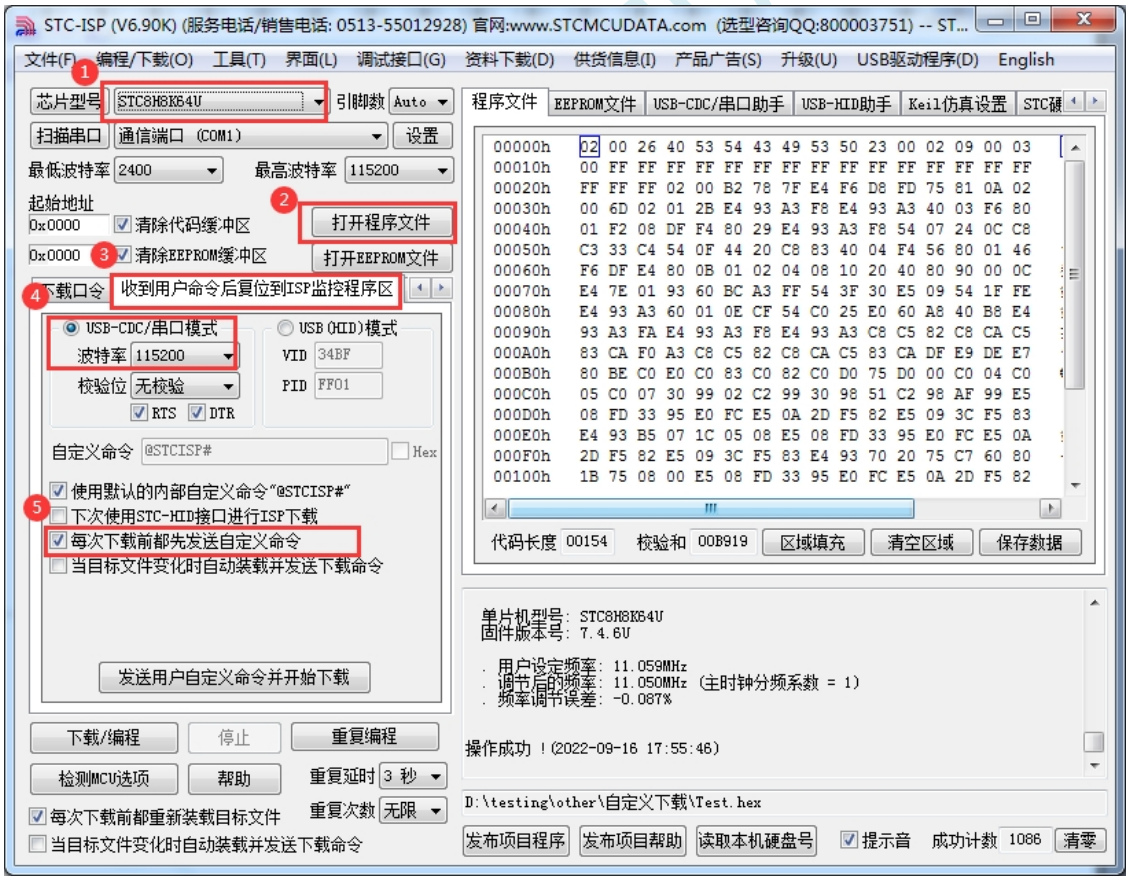
(The reference code is as follows, the test microcontroller model is STC8H8K64U)

```
#include "stc8h.h"

#define    FOSC        11059200UL
#define    BAUD        (65536 - (fosc/115200+2)/4)      The // plus 2 operation is designed to make the Keil compiler
                                                        //Automatic implementation of rounding operations

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

void uart_isr() interrupt 4
{
    char dat.

    if (TI)
    {
        TI = 0;
    }

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

        if (dat == STCISPCMD[index])        // Judge whether the received data matches the current command character or not
        {
                                        }
                                    }

        }
        else
        {



        }
```

```
index++;
//If match then index+1
if
(
S
T
C
I
S
P
C
M
D
                [index] == '\0')                    //determine if command
                matching is complete IAP_CONTR = 0x60;          //If
                match complete then soft reset to ISP

                index = 0;                          //if no match, need to
                start from the beginning if (dat == STCISPCMD[index])
                    index++;

void main()
{
    p0m0 = 0x00; p0m1 = 0x00.
```

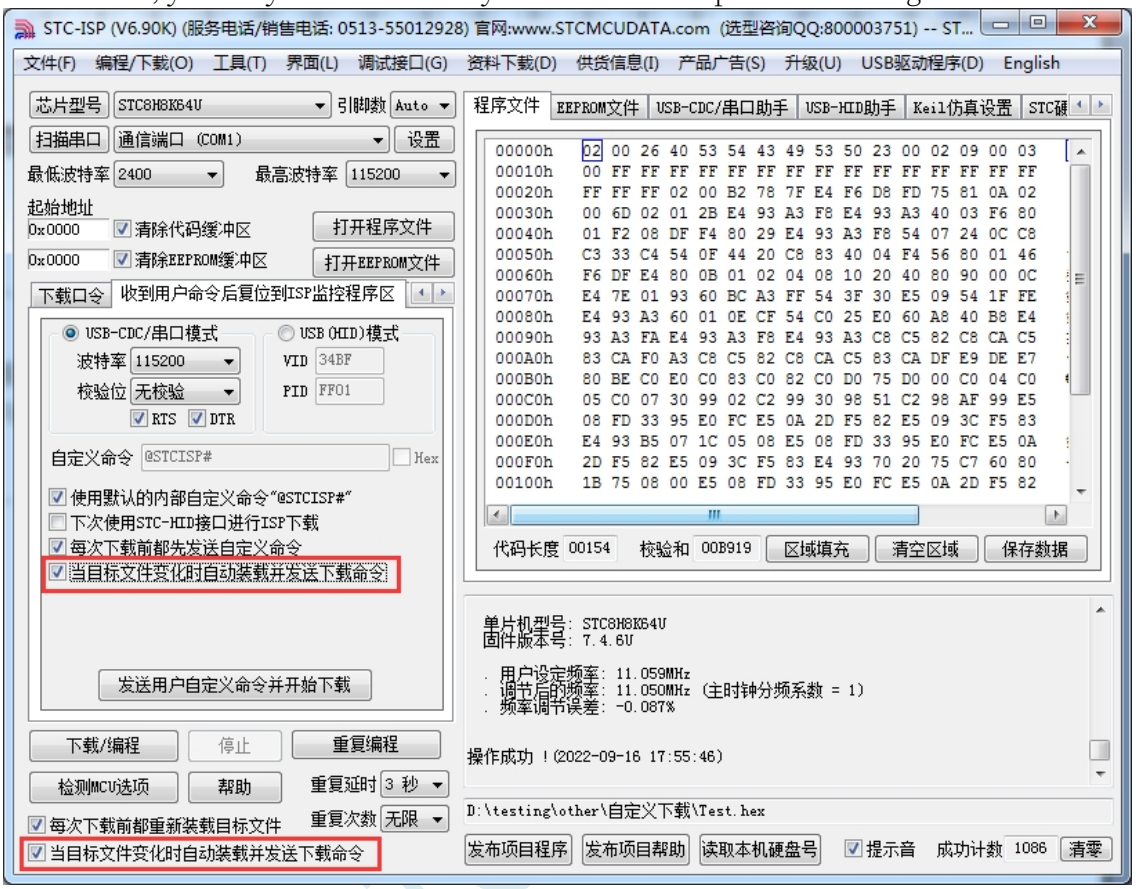
```
        p1m0 = 0x00; p1m1 = 0x00;
        p2m0 = 0x00; p2m1 = 0x00;
        p3m0 = 0x00; p3m1 = 0x00.

        SCON = 0x50;                    // Serial port initialisation
        auxr = 0x40; tmod
        = 0x00; th1 = baud
        >> 8; tl1 = baud.
        TR1 = 1;
        ES = 1;
        EA = 1;

        index = 0;                      //Initialisation command

        while (1);
    }
```

2. Set up the custom download command as shown in the following figure (the example uses the STC



3, the first time to download the target microcontroller needs to re-power on, and then each time to update only need to click on the download software in the "download/programming" button, the download software automatically sends the download command to the target microcontroller, the target microcontroller receives the command automatically reset to the system ISP area, you can realise the update of the user code without stopping the power.

4, STC-ISP can also be realised in the project development phase, completely automatic download function, that is, when the download software detects the target code has been updated, it will automatically send the download command. To realise this function, you only need to tick any one of the two options in the figure below.

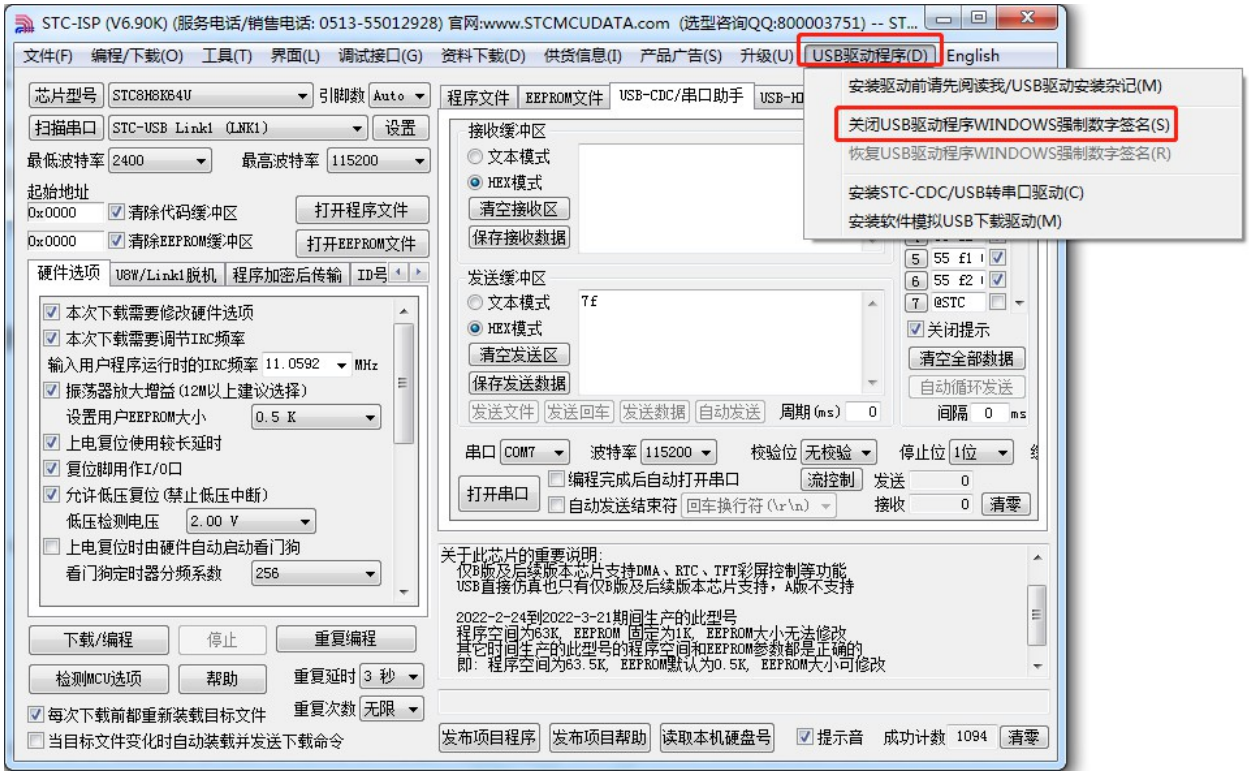


5.18 Notes on driver digital signatures

5.18.1 About driver mandatory digital signatures

Windows systems from the Vista version onwards require drivers to be digitally signed in order to be installed on 64-bit Windows systems, otherwise the driver will not be installed and the USB device will not work properly. Currently, the drivers provided by STC are not WHQL certified (Microsoft logo certified), so they may not be installed successfully on 64-bit Windows systems starting from Vista version until the mandatory digital signature of drivers is disabled.

The latest STC-ISP download software provides a simple disablement of the driver forced digital signature, with the menu catalogue shown below:

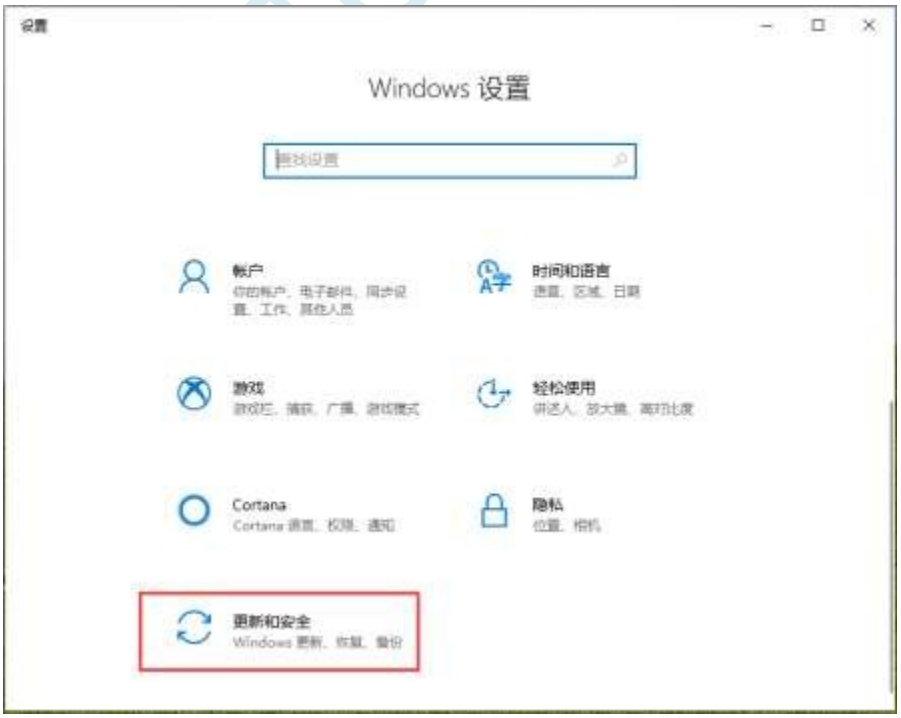


This menu item uses the TestSigning and NoIntegrityChecks parameters that are automatically set by the system's BCDEDIT utility to temporarily disable the driver's forced digital signature. For systems with Secure Boot enabled, the BCDEDIT utility cannot be used, and Secure Boot needs to be disabled in the BIOS (set Secure Boot to Disable).

Currently, WinXP/Win7-32/Win7-64 systems can disable digital signatures, but Win10 and later systems cannot. To disable digital signatures on Win10 and later systems, you still need to select Disable forced digital signatures from the system startup menu. Please refer to the next section for more details.

5.18.2 Windows 10 Steps to turn off forced digital signatures for drivers

Select the "Settings" function in the "Start" menu



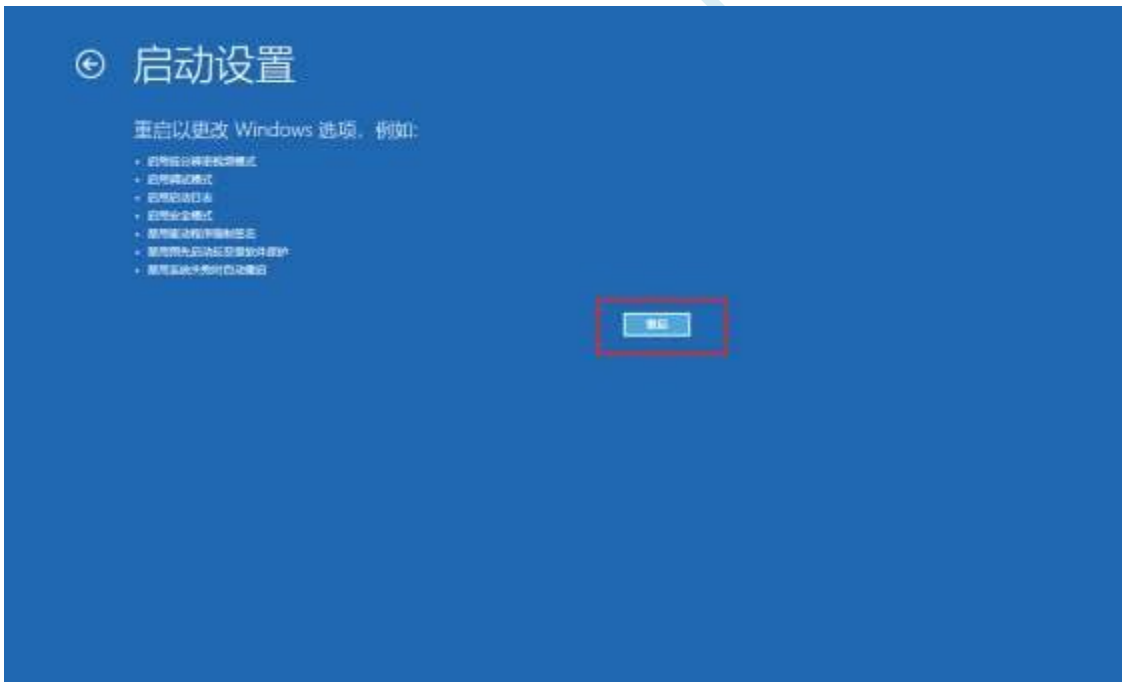

设置
主页
查找设置
更新和安全
Windows 更新
传递优化
Windows 安全中心
备份
疑难解答
恢复
激活
查找我的设备
开发者选项
恢复
重置此电脑
如果电脑未正常运行，重置电脑可能会解决问题。重置时，可以选择保留个人文件或删除个人文件，然后重新安装 Windows。
开始
高级启动
通过设备或磁盘(如 U 盘或 DVD)启动，更改你的电脑固件设置，更改 Windows 启动设置，或者从系统映像还原 Windows。 这将重新启动电脑。
立即重新启动
更多恢复选项
了解如何进行 Windows 的全新安装以便开始全新的体验
备份文件

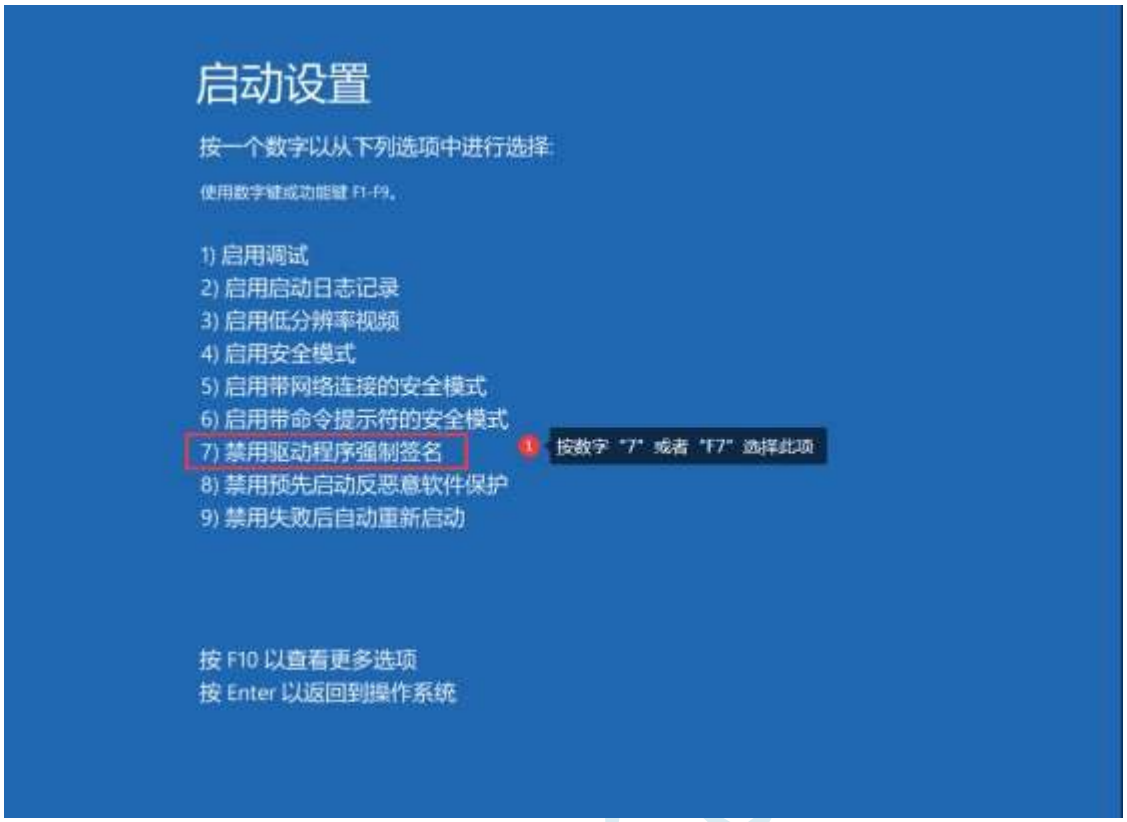

选择一个选项
继续
关闭电脑
使用设备
疑难解答


疑难解答
重置此电脑
高级选项


高级选项
系统还原
启动修复
卸载更新
命令提示符
系统映像恢复
UEFI 固件设置
查看更多恢复选项


高级选项
启动设置
查看更多恢复选项


启动设置
重启以更改 Windows 选项，例如：

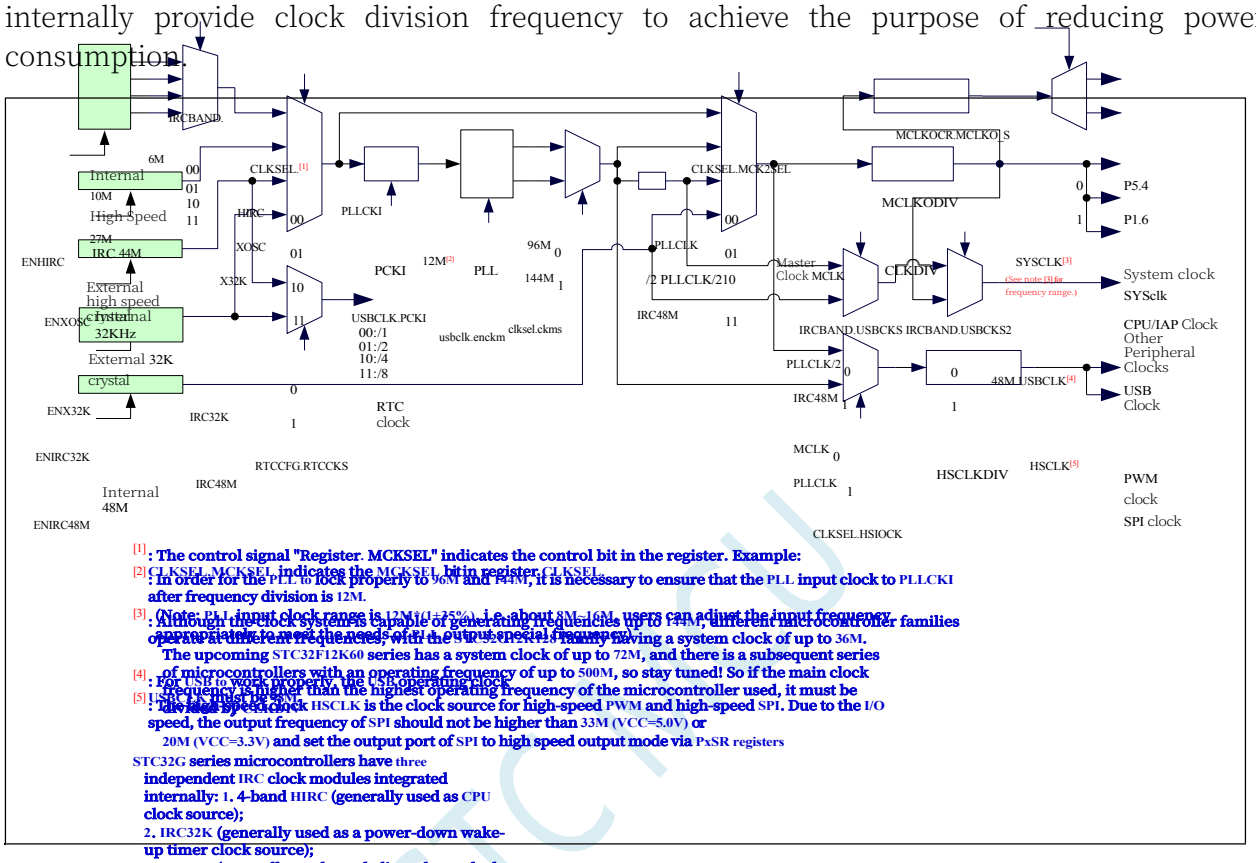


The driver's forced digital signature can be temporarily disabled after a system reboot.

6 Clock management

6.1 System Clock Control

The system clock controller provides clock sources for the CPU and all peripheral systems of the microcontroller. The system clock has four clock sources to choose from: internal high-precision IRC, internal 32KHz IRC (with large error), external crystal, and internal PLL output clock. Users can programmatically enable and disable each clock source separately, as well as internally provide clock division frequency to achieve the purpose of reducing power consumption.



[1]: The control signal "Register. MCKSEL" indicates the control bit in the register. Example: CLKSEL.MCKSEL indicates the MCKSEL bit in register CLKSEL.

[2]: In order for the PLL to lock properly to 96M and 144M, it is necessary to ensure that the PLL input clock to PLLCKI after frequency division is 12M.

[3]: (Note: PLL input clock range is 12M(1±5%), i.e. about 8M~16M, users can adjust the input frequency appropriately to meet the needs of PLL output special frequency)
Although the clock system is capable of generating frequencies up to 144M, different microcontroller families operate at different frequencies, with the STC32A8K64S4 family having a system clock of up to 36M.
The upcoming STC32F12K60 series has a system clock of up to 72M, and there is a subsequent series of microcontrollers with an operating frequency of up to 500M, so stay tuned! So if the main clock frequency is higher than the highest operating frequency of the microcontroller used, it must be divided by SYSCLK.

[4]: For USB to work properly, the USB operating clock

[5]: The high-speed clock HSCLK is the clock source for high-speed PWM and high-speed SPI. Due to the I/O speed, the output frequency of SPI should not be higher than 33M (VCC=5.0V) or 20M (VCC=3.3V) and set the output port of SPI to high speed output mode via PxSR registers

STC32G series microcontrollers have three independent IRC clock modules integrated internally: 1. 4-band HIRC (generally used as CPU clock source);
2. IRC32K (generally used as a power-down wake-up timer clock source);
3. IRC48M (generally used as a dedicated USB clock source);

System and Peripheral Clock Selection Reference Table (refer to the sample programme for detailed settings)

Master Clock Selection (MCLK)	Internal IRC	Internal High Speed IRC (HIRC)
		Internal Low Speed IRC (IRC32K)
		Internal USB dedicated 48M Hi-Speed IRC (IRC48M)
	external crystal	External high-speed crystal (XOSC)
		External low-speed crystal (X32K)
	PLL	Internal PLL Clock (PLL)
		Internal PLL clock 2 divisions (PLL/2)
High-speed peripheral clock selection (HSIOCK)	Master Clock (MCLK)	
	Internal PLL Clock (PLL)	
USB clock selection (48M)	Internal USB dedicated 48M Hi-Speed IRC (IRC48M)	
	Internal PLL clock 2 divisions (PLL/2)	
	System clock (SYSCLK)	

Note: The system clock (SYSCLK) is the clock derived from dividing the master clock (MCLK) by CLKDIV. The HSPWM/HSSPI clock (HSCLK) is the clock obtained by dividing the high-speed peripheral clock (HSIOCK) by HSCLKDIV.

PLL Input Clock Selection Reference

MCKSEL [1:0]	PLL Input Clock Source
00	Internal High Speed IRC (HIRC)
01	External high-speed crystal (XOSC)
10	External low-speed crystal (X32K)
11	Internal Low Speed IRC (IRC32K)

6.2 Related registers

Related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IRCBAND	IRC band selection	A9H	USBCKS	USBCKS2	-	-	-	-	SEL[1:0]		10xx,xxnn
LIRTRIM	IRC Frequency Trim Register	9EH	-	-	-	-	-	-	-	LIRTRIM	xxxx,xxxn
IRTRIM	IRC Frequency Adjustment Register	9FH	IRTRIM[7:0]								nnnn,nnnn
USBCLK	USB Clock Control Register	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]		0010,0000

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	Clock Select Register	7EFE00H	CKMS	HSIOCK	-	-	MCK2SEL[1:0]		MCKSEL[1:0]		00xx,0000
CLKDIV	Clock Divider Register	7EFE01H									nnnn,nnnn
HIRCCR	Internal high-speed oscillator control register	7EFE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	External Crystal Control Register	7EFE03H	ENXOSC	XITYPE	GAIN		XCFILTER[1:0]			XOSCST	000x,00x0
IRC32KCR	Internal 32K Oscillator Control Registers	7EFE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	Master Clock Output Control Register	7EFE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	Internal high-speed oscillator stabilisation time control	7EFE06H									1000,0000
IRC48MCR	Internal 48M Oscillator Control Registers	7EFE07H	ENIRC48M	-	-	-	-	-	-	IRC48MST	0xxx,xxx0
X32KCR	External 32K Crystal Control Register	7FFE08H	ENX32K	GAIN32K		-	-	-	-	X32KST	00xx,xxx0
HSCLKDIV	High Speed Clock Divider Register	7EFE0BH									0000,0010
HPLLCR	High-speed PLL control registers	7EFE0CH	ENHPLL	-	-	-	HPLLDIV[3:0]				0xxx,0000
HPLLPSCR	High-speed PLL prescaler register	7EFE0DH	-	-	-	-	HPLL_PREDIV[3:0]				xxxx,0000

6.2.1 USB Clock Control Register (USBCLK)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USBCLK	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]	

ENCKM: PLL multiplier control

0: Disable PLL frequency doubling

1: Enable PLL multiplier

PCKI[1:0]: PLL clock selection

PCKI[1:0]	PLL Clock Source
00	/1

01	/2
10	/4
11	/8

CRE: Clock chasing control bit 0: Clock chasing disabled

1: Enable clock chasing

6.2.2 System clock selection register (CLKSEL)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CLKSEL	7EFE00H	CKMS	HSIOCK	-	-	MCK2SEL[1:0]		MCKSEL[1:0]	

CKMS: Internal PLL output clock selection

0: PLL output 96MHz

1: PLL output 144MHz

HSIOCK: High Speed I/O Clock Source Selection

0: Master clock MCLK is the high-speed I/O clock source

1: PLL output 96MHz/144MHz PLLCLK for high-speed I/O clock source

MCK2SEL[1:0]: Master clock source selection

MCK2SEL[1:0]	master clock source
00	MCKSEL Selected Clock Source
01	Internal PLL output
10	Internal PLL output/2
11	Internal 48MHz high speed IRC

MCKSEL[1:0]: Master clock source selection

MCKSEL[1:0]	master clock source
00	Internal high precision IRC
01	External high speed crystal
10	External 32KHz crystal
11	Internal 32KHz low speed IRC

6.2.3 Clock division register (CLKDIV)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	7EFE01H								

CLKDIV: Master clock division frequency coefficient. The system clock SYSCLK is the clock signal after MCLK.

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

6.2.4 Internal high-speed, high-precision IRC control register (HIRCCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HIRCCR	7EFE02H	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 stabilisation flag bit. (read-only bit)

When the internal IRC is enabled from the stop state, a period of time must elapse before the oscillator frequency stabilises, and when the oscillator frequency stabilises, the clock controller will automatically set the HIRCST flag bit to 1. So when the user program needs to switch the clock to use the internal IRC, it must firstly set ENHIRC=1 to enable the oscillator, and then keep on querying the oscillator stabilisation flag bit HIRCST until the flag bit changes to 1, then the clock source can be switched.

6.2.5 External Oscillator Control Register (XOSCCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
XOSCCR	7EFE03H	ENXOSC	XITYPE	GAIN		XCFILTER[1:0]			XOSCST

ENXOSC: External crystal oscillator enable bit 0:

Disable external crystal oscillator

1: Enable external crystal oscillator XITYPE:

External clock source type

0: The external clock source is an external clock signal (or active crystal). The signal source is simply connected to the XTALI (P1.7) of the microcontroller

1: The external clock source is a crystal oscillator. The signal source is connected to XTALI (P1.7) and XTALO (P1.6) of the microcontroller XCFILTER[1:0]:

external crystal oscillator anti-interference control registers

00: This item can be selected when the external crystal oscillator frequency is 48M or below.

01: This item can be selected when the external crystal oscillator frequency is 24M or below.

1x: This option is available when the external crystal oscillator frequency is 12M and below.

GAIN: External crystal oscillator oscillation gain control bit 0:

Oscillation gain off (low gain)

1: Enable oscillation gain (high gain)

XOSCST: External crystal oscillator frequency stabilisation flag bit. (Read-only bit)

When the external crystal oscillator is enabled from the stop state, a period of time must pass before the oscillator frequency stabilises, and when the oscillator frequency stabilises, the clock controller will automatically set the XOSCST flag bit to 1. So when the user program needs to switch the clock to use the external crystal oscillator, the first step must be to set ENXOSC=1 to enable the oscillator, and then keep on querying the oscillator stability flag bit XOSCST until the flag bit changes to 1, then the clock source can be switched.

6.2.6 Internal 32KHz low-speed IRC control register (IRC32KCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IRC32KCR	7EFE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

ENIRC32K: Internal 32K low-speed IRC enable bits

0: Disable internal 32K low-speed IRC

1: Enable internal 32K low-speed IRC

IRC32KST: Internal 32K low-speed IRC frequency stabilisation flag bit. (read-only bits)

When the internal 32K low-speed IRC is enabled from the stop state, a period of time must elapse before the oscillator frequency stabilises, and when the oscillator frequency stabilises, the clock controller will automatically set the IRC32KST flag bit to 1. Therefore, when the user program needs to switch the clock to use the internal 32K low-speed IRC, firstly, it must set ENIRC32K=1 to enable the oscillator, and then it always So when the user programme needs to switch the clock to use the internal 32K low-speed IRC, it must first set ENIRC32K=1 to enable the oscillator, and then keep querying the oscillator stabilisation flag bit IRC32KST until the flag bit is changed to 1.

6.2.7 Master Clock Output Control Register (MCLKOCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
MCLKOCR	7EFE05H	MCLKO_S	MCLKODIV[6:0]						

MCLKODIV[6:0]: master clock output divider coefficients

MCLKODIV[6:0]	System clock divider output frequency
0000000	No clock output
0000001	SYSclk/1
0000010	SYSclk /2
0000011	SYSclk /3
...	...
1111110	SYSclk /126
1111111	SYSclk /127

MCLKO_S: System Clock Output Pin Selection 0: System Clock Divided Output to P5.4 Port

1: System clock divider output to port P1.6

6.2.8 High-speed oscillator stabilisation time control register (IRCDB)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IRCDB	7EFE06H								

IRCDB[7:0]: internal high-speed oscillator stabilisation time control.

IRCDB	System clock frequency
0	256 clocks
1	1 Clock
2	2 clocks
3	3 clocks
...	...
x	x clocks
...	...
255	255 clocks

6.2.9 Internal 48MHz High-Speed IRC Control Register (IRC48MCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IRC48MCR	7EFE07H	ENIRC48M	-	-	-	-	-	-	IRC48MST

ENIRC48M: Internal 48M high-speed IRC enable bit

0: Disable internal 48M high-speed IRC

1: Enable internal 48M high speed IRC

IRC48MST: Internal 48M high-speed IRC frequency stabilisation flag bit. (read-only bit)

When the internal 48M high-speed IRC is enabled from the stop state, a period of time must elapse before the oscillator frequency stabilises, and when the oscillator frequency stabilises, the clock controller will automatically set the IRC48MST flag to position 1. so when the user program needs to switch the clock to use the internal 48M high-speed IRC, it must first set ENIRC48M=1 to enable the oscillator, and then always Query the oscillator stability flag

Flag bit IRC48MST until the flag bit changes to 1 before the clock source can be switched.

6.2.10 External 32K Oscillator Control Register (X32KCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
X32KCR	7EFE08H	ENX32K	GAIN32K	-	-	-	-	-	X32KST

ENX32K: External 32K crystal oscillator enable bit

0: Disable external 32K crystal oscillator

1: Enable external 32K crystal oscillator

GAIN32K: External 32K Crystal Oscillator Oscillation Gain Control Bit

0: 32K oscillation gain off (low gain)

1: Enable 32K oscillation gain (high gain)

X32KST: External 32K crystal oscillator frequency stabilisation flag bit. (read-only bit)

6.2.11 High Speed Clock Division Register (HSCLKDIV)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HSCLKDIV	7EFE0BH								

HSCLKDIV: High-speed I/O clock division factor.

CLKDIV	System clock frequency
0	High-speed I/O clock source/1
1	High-speed I/O clock source/1
2	High-speed I/O clock source/2
3	High-speed I/O clock source/3
...	...
x	High-speed I/O clock source/x
...	...
255	High Speed I/O Clock Source/255

6.3 STC32G Series Internal IRC Frequency Adjustment

All STC32G series microcontrollers have a high precision internal IRC oscillator integrated inside. When the user uses the ISP download software to download, the ISP download software will automatically adjust the frequency according to the frequency selected/set by the user, and the general frequency value can be adjusted to less than ±0.3%, and the temperature drift of the adjusted frequency in the full temperature range (-40℃~85℃) can reach -1.35%~1.30%.

The internal IRC of the STC32G series has 4 frequency bands with centre frequencies of 6MHz, 10MHz, 27MHz and 44MHz, each with an adjustable range of about ±27% (note: different chips and different generation batches may have a manufacturing error of about 5% or so)

Note: For general users, the internal IRC frequency adjustment can be ignored, because the frequency adjustment work has been done automatically during the ISP download. Therefore, if the user does not need to adjust the frequency by himself, then the following 4 registers should not be modified freely, otherwise it may lead to changes in the operating frequency.

The internal IRC frequency is adjusted mainly using the following four registers

6.3.1 IRC Band Selection Register (IRCBAND)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IRCBAND	A9H	USBCKS	USBCKS2	-	-	-	-	SEL[1:0]	

USBCKS/USBCKS2: USB Clock Select Registers

USBCKS	USBCKS2	USB Clock
0	0	PLLCLK/2
1	0	IRC48M
x	1	System clock SYSCLK

SEL[1:0]: band selection

00: select 6MHz band

01: Select 10MHz band

10: Select 27MHz band

11: Select 44MHz band

6.3.2 Internal IRC Frequency Adjustment Register (IRTRIM)

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

IRTRIM[7:0]: Internal high-precision IRC frequency adjustment registers

IRTRIM can adjust the IRC frequency in 256 levels, and the frequency value adjusted in each level is linearly distributed in the whole, with local fluctuations. Macro, each level of frequency adjustment is about 0.24%, i.e., the frequency when IRTRIM is (n+1) is about 0.24% faster than the frequency when IRTRIM is (n). However, since the IRC frequency adjustment is not 0.24 per cent at every level (the maximum value of the adjusted frequency at each level is about 0.55 per cent, the minimum value is about 0.02 per cent, and the overall average is about 0.24 per cent), local fluctuations can occur.

6.3.3 Internal IRC Frequency Trim Register (LIRTRIM)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LIRTRIM	9EH	-	-	-	-	-	-		LIRTRIM

LIRTRIM: Internal high-precision IRC frequency trimming registers

6.3.4 Clock division register (CLKDIV)

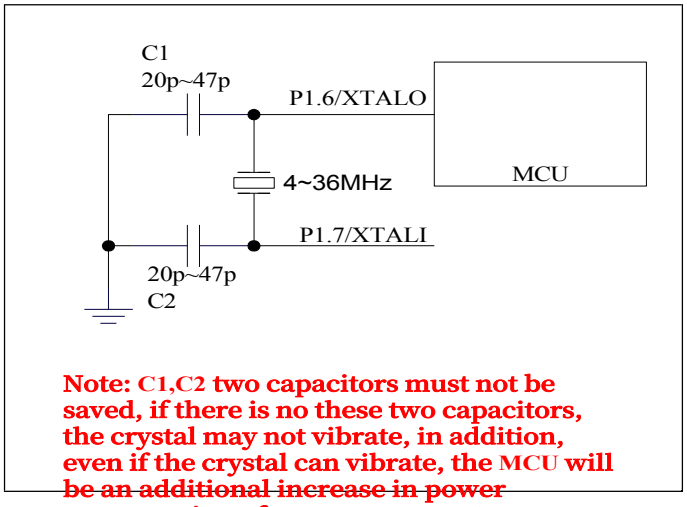
notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	7EFE01H								

CLKDIV: Master clock division frequency coefficient. The system clock SYSCLK is the clock signal after MCLK.

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

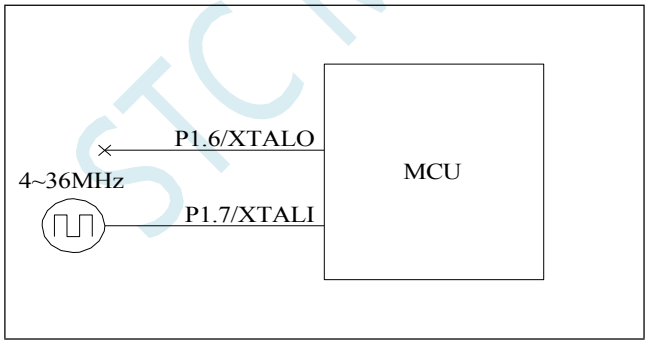
6.4 External Crystal and External Clock Circuit

6.4.1 External Crystal Input Circuit



Note: C1,C2 two capacitors must not be saved, if there is no these two capacitors, the crystal may not vibrate, in addition, even if the crystal can vibrate, the MCU will be an additional increase in power consumption of 5 to 8mA!

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



Note: When the internal clock is used as the master clock source, P1.6/P1.7 can be used as normal I/Os.

6.5 sample procedure

6.5.1 Selects internal high-speed IRC (HIRC) as system clock source

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set the external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

                                            //Note: The system will silently start the internal high-speed HIRC when the chip is powered on.
                                            // and is selected as the system clock, so it is not normally necessary to make a
                                            //Set up as follows
    HIRCCR = 0x80;                          //Start internal high-speed IRC
    while (! (HIRCCR & 1));                 //Wait for the clock to stabilise
    CLKSEL = 0x00;                          //Select internal high speed HIRC

    while (1);
}
```

6.5.2 Select internal IRC (IRC32K) as system clock source

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files
```

```
Test_Init()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
```

```
                                        //Assign 0 to set the CPU to execute the programme
                                        as fast as possible.
    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    IRC32KCR = 0x80.                    //Start internal 32K IRC
    while (! (IRC32KCR &                //Wait for clock to stabilise
    1)); CLKDIV = 0x00;                 //Clocks are not frequency-divided
    CLKSEL = 0x03;                      //Select internal 32K

    while (1);
}
```

6.5.3 Selects the internal 48M IRC (IRC48M) as the system clock source

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as
                                        possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    IRC48MCR = 0x80.                    //Start internal 48M IRC
    while (! (IRC48MCR & 1));           //Wait for the clock to stabilise
```

```
    CLKDIV = 0x02.                  //clock 2 division
    CLKSEL = 0x0c.                  //Select IRC48M
```

```
    while (1);
}
```

6.5.4 Selecting an external high-speed crystal (XOSC) as the system clock source

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    XOSCCR = 0xc0.                      //start external crystal
    while (! (XOSCCR & 1));             //Wait for the clock to stabilise
    CLKDIV = 0x00.                      //clock not divided
    CLKSEL = 0x01.                      //select external crystal

    while (1);
}
```

6.5.5 Selecting an external low-speed crystal (X32K) as the system clock source

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

void main()
```

```
{
    EAXFR = 1;          //Enable access to XFR
    CKCON = 0x00;       //Set the external data bus speed to fastest
    WTST = 0x00;        //set the program code wait parameter.
                        //Assign 0 to set the CPU to execute the programme as fast as possible.
```

```
    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    x32kcr = 0xc0.                          //Activate external 32K crystal
    while (! (X32KCR & 1));                 //Wait for clock to stabilise
    CLKDIV = 0x00;                          //Clocks are not frequency-divided
    CLKSEL = 0x02;                          //Select external 32K crystal

    while (1);
}
```

6.5.6 Selecting the internal PLL as the system clock source

//Test operating frequency is 12MHz

```
//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files

void delay()
{
    int i;

    for (i=0; i<100; i++);
}

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set the external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.
```

```
    clksel &= ~0x80;                    //Select 96M of the PLL as the output clock of the PLL.
//  clksel |= 0x80.                     //Select 144M of the PLL as the output clock of the PLL.

    usbclk &= ~0x60; usbclk |=
    0x00.                               //Select 1 division if the /PLL input clock is 12M.
//  USBCLK |= 0x20.                     //Select 2 divisions if the /PLL input clock is 24M.
//  USBCLK |= 0x40.                     //Select 4 divisions if the /PLL input clock is 48M.
//  USBCLK |= 0x60.                     //Select 8 divisions if the /PLL input clock is 96M.

    USBCLK |= 0x80.                     //Start PLL

    delay();                            //Wait for PLL lock, more than 50us is recommended.

    CLKDIV = 0x04;                      //Clock 4 divisions, master clock selects high speed frequency before, //Clock 4 divisions, master clock selects high speed frequency before
    CLKSEL &= 0xf0.                     //Must set the crossover coefficient first, otherwise the programme will fall off.
                                        //Select PLL clock source
    CLKSEL |= 0x04.

    while (1);
}
```

6.5.7 Select the master clock (MCLK) as the high-speed peripheral clock source

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
```

```
    p5m0 = 0x00;
    p5m1 = 0x00.

    HSCLKDIV = 0x00;
    CLKSEL &= ~0x40.                    //Select master clock (MCLK) as high-speed peripheral clock
    source

    while (1);
}
```

6.5.8 Selecting the internal PLL clock as a high-speed peripheral clock source

```
//Test operating frequency is 12MHz

//#include "stc8h.h"
#include "stc32g.h"                                   // see download software for header files

void main()
{
    EAXFR = 1;                                        //Enable access to XFR
    CKCON = 0x00;                                     //Set the external data bus speed to fastest
    WTST = 0x00;                                      //set the program code wait parameter.
                                                      //Assign 0 to set the CPU to execute the programme as fast as
                                                      possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

CLKSEL &= ~0x80.                                      //Select 96M of the PLL as the output clock of the PLL.
  //CLKSEL |= 0x80; //Select 144M of PLL as PLL output clock.  //CLKSEL |= 0x80; //Select 144M of PLL as the output
                                                                clock of PLL.

    USBCLK &= ~0x60.
    USBCLK |= 0x00.                                   //PLL input clock is 12M, then select 1 division.
    //USBCLK |= 0x20.                                 //PLL input clock is 24M, then select 2 divisions.
    //USBCLK |= 0x40.                                 //PLL input clock is 48M then select 4 divisions.
    //USBCLK |= 0x60.                                 //PLL input clock is 96M then select 8 divisions.

    USBCLK |= 0x80.                                   //start PLL

    delay();                                          //Wait for PLL to lock frequency, more than 50us is
    recommended.

    HSCLKDIV = 0x00;
    CLKSEL |= 0x40.                                   //Select PLL clock as high speed peripheral clock source

    while (1);
}
```

6.5.9 Select the system clock (SYSCLK) as the USB clock source

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
```

```
void main()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    IRCBAND |= 0x40; //Select system clock (SYSCLK) as USB clock source.    //Select system clock (SYSCLK) as USB clock source.

    while (1);
}
```

6.5.10 Selecting the Internal PLL Clock as the USB Clock Source

```
//Test operating frequency is 12MHz

//#include "stc8h.h"
#include "stc32g.h"                             // see download software for header files

void main()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

CLKSEL &= ~0x80.                                //Select 96M of the PLL as the output clock of the PLL.
  //CLKSEL |= 0x80; //Select 144M of PLL as PLL output clock.  //CLKSEL |= 0x80; //Select 144M of PLL as the output
```

clock of *PLL*.

```
    USBCLK &= ~0x60.
    USBCLK |= 0x00;                 //Select 1 division if the /PLL input clock is 12M.
//  USBCLK |= 0x20.                 //Select 2 divisions if the /PLL input clock is 24M.
//  USBCLK |= 0x40.                 //Select 4 divisions if the /PLL input clock is 48M.
//  USBCLK |= 0x60.                 //Select 8 divisions if the /PLL input clock is 96M.

    USBCLK |= 0x80.                 //Start PLL

    delay();                        //Wait for PLL lock, more than 50us is
                                    recommended.

    IRCBAND &=~ 0xc0.               //Select PLL clock (96M/2=48M) as USB clock source

    while (1);
}
```

6.5.11 Selects internal USB-specific 48M IRC as USB clock source

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 //see download software for header files

void main()
{
    EAXFR = 1;                      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign 0 to set the CPU to execute the programme as fast as
                                    possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    IRC48MCR = 0x80.                //Start internal 48M IRC
    while (! (IRC48MCR & 1));       //Wait for the clock to stabilise

    IRCBAND |= 0x80; //Select IRC48M as USB clock source.    //Select IRC48M as USB clock source.
    IRCBAND &= ~0x40.

    while (1);
}
```

6.5.12 Master Clock Split Output

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files

void main()
{
    EAXFR = 1;                                         //Enable access to XFR
    CKCON = 0x00;                                      //Set the external data bus speed to fastest
    WTST = 0x00;                                       //set the program code wait parameter.
                                                       //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    //  MCLKOCR = 0x01.                                //Master clock output to port P5.4.
//MCLKOCR = 0x02.                                      //Master clock 2 divider output to port P5.4.
MCLKOCR = 0x04.                                        //Master clock 4-division output to port P5.4.
    //MCLKOCR = 0x84.                                  //Master clock 4-division output to port P1.6.

    while (1);
}
```

7 Automatic frequency calibration, automatic frequency tracking (CRE)

product lines	auto-frequency tracking (auto-tracking)
STC32G12K128 Series	
STC32G8K64 Series	●
STC32F12K60 Series	●

The STC32G series has a built-in frequency auto-calibration module (CRE). The CRE module uses an external 32.768KHz crystal to automatically adjust the IRTRIM register of the internal high-speed IRC (HIRC) to achieve the auto-frequency calibration function. When auto-calibration is required, just set the target frequency count and error range according to the given formula, then start the CRE module, the hardware will carry out the auto-frequency calibration, and when the frequency of the HIRC reaches the error range set by the user, the calibration completion flag will be set.

7.1 Related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CRECR	CRE Control Register	7EFDA8H	ENCRE	MONO	UPT[1:0]		CREHF	CREINC	CREDEC	CRERDY	0000,0000
CRECNTH	CRE Calibration target register	7EFDA9H	CNT [15:8]								0000,0000
CRECNTL	CRE Calibration target register	7EFDAAH	CNT [7:0]								0000,0000
CRERES	CRE Resolution Control Register	7EFDABH	RES[7:0]								0000,0000

7.1.1 CRE Control Register (CRECR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CRECR	7EFDA8H	ENCRE	MONO	UPT[1:0]		CREHF	CREINC	CREDEC	CRERDY

ENCRE: CRE module control bit 0:
Disables the CRE module.
1: Enable the CRE module.

MONO: automatic calibration of stride control

0: Single-step mode. The hardware automatically increments or decrements IRTRIM by 1 per calibration cycle.

1: Two-step mode. The hardware automatically increments or decrements IRTRIM by 2 per calibration cycle.

Single-step mode provides higher accuracy of calibrated IRC than two-step mode, but auto-calibration takes longer than two-step mode.

UPT[1:0]: CRE calibration period selection

UPT[1:0]	calibration period
00	1ms
01	4ms
10	32ms
11	64ms

CREHF: High frequency mode selection

0: Low frequency mode (target frequency less than or equal to 50MHz).

1: High frequency mode (target frequency > 50MHz).

CREINC: CRE calibration is being up-regulated. Read only bits. CREDEC: The CRE calibration is in the down state. Read only bits. CRERDY: CRE calibration is complete. Read-only bit.

0: CRE calibration function is not activated or calibration is not completed.

1: CRE calibration is complete.

7.1.2 CRE Calibration Count Value Register (CRECNT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CRECNTH	7EFDA9H	CRECNT [15:8]							
CRECNTL	7EFDAAH	CRECNT[7:0]							

CRECNT[15:0]: 16-bit calibration count value. Calculation formula for the target calibration value:

Low frequency mode (CREHF=0): CRECNT = (16 * target frequency (Hz)) / 32768

High frequency mode (CREHF=1): CRECNT = (8 * target frequency (Hz)) / 32768

(see sample programme for detailed settings)

7.1.3 CRE Calibration Error Value Registers (CRERES)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CRERES	7EFDABH	CRERES[7:0]							

CRERES[7:0]: 8-bit calibration error value (resolution control).

Since the resolution of the internal high-speed IRC is much lower than that of the external 32.768K crystal, the final calibration value cannot be exactly the same as the target value set by CRECNT, so an error range must be set through the CRERES register.

Calibration error calculation formula:

CRERES = range of error (%) * target calibration value

(Tolerance range of 1 to 0.3 per cent is generally acceptable, and it is not recommended to exceed this range.)

(see sample programme for detailed settings)

7.2 sample procedure

7.2.1 Automatic calibration of internal high-speed IRC (HIRC)

For example, the target frequency for calibration is 22.1184 MHz, and the calibration error range is ±0.5 per cent.

Then you need to set CREHF to 0 and CRECNT to (16*22118400)/32768 = 10800 (2A30H).

i.e. CRECNTH set to 2AH, CRECNTL set to 30H, CRERES set to 10800 * 0.5% = 54 (36H)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                  // see download software for header files

#define      CNT22M          (16 * 22118400L) / 32768)      //calibrate target frequency to 22.1184M #define
RES22M      (CNT22M *5 / 1000)      //Set the calibration error to 0.5 per cent.

void main()
{
    EAXFR = 1;                                       //Enable access to XFR
    CKCON = 0x00;                                    //Set the external data bus speed to fastest
    WTST = 0x00;                                     //set the program code wait parameter.
                                                     //Assign 0 to set the CPU to execute the programme as fast as possible.


    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    X32KCR = 0xc0.                                   //Start external 32K crystal
    while (! (X32KCR & 1));                          //Wait for the clock to stabilise

    IRCBAND &= ~0x03.
    IRCBAND |= 0x02.                                 //Select 27M band
    CLKSEL = 0x00.                                   //Select internal high-speed HIRC as system clock

    CRECNTH = CNT22M >> 8;                           //Set target calibration value CRECNTL = CNT22M.
    CRERES = RES22M.                                 //Set the calibration error
    CRECR = 0x90;                                    //Enable the CRE function and set the calibration period to 4ms.

    while (1)
    {
```

```
if(CALECR & 0x01)
{
                                        //Frequency auto-calibration complete
```

```
        }
    }
}
```

8 Reset, Watchdog, Wake-on-Drop Dedicated Timer and Power Management

8.1 system reset

The reset of STC32G series microcontroller is divided into hardware reset and software reset.

During a hardware reset, all register values are reset to their initial values and the system re-reads all hardware options. At the same time, the power-up wait time is performed according to the power-up wait time set by the hardware options. Hardware reset mainly includes:

- Power-on reset
- Low pressure reset
- Reset pin reset **(low level reset)**
- watchdog reset

During a software reset, the values of all registers are reset to their initial values except for the clock-related registers, which remain unchanged; a software reset does not re-read all hardware options. Software reset mainly includes:

- Write the reset triggered by SWRST of IAP_CONTR.

Related registers

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

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
RSTFLAG	Reset Flag Register	7EFE99H	-	-	-	LVDRSTF	WDTRSTF	SWRSTF	ROMOVF	EXRSTF	xxx1,0100
RSTCR0	Reset Control Register 0	7EFE9AH	-	-	-	-	RSTTM34	TSTTM2	-	RSTTM01	xxxx,00x0
RSTCR1	Reset Control Register 1	7EFE9BH	-	-	-	-	RSTUR4	RSTUR3	RSTUR2	RSTUR1	xxxx,0000
RSTCR2	Reset Control Register 2	7EFE9CH	RSTCAN2	RSTCAN	RSTLIN	RSTRTC	RSTPWMB	RSTPWMA	RSTI2C	RSTSPI	0000,0000
RSTCR3	Reset Control Register 3	7EFE9DH	RSTFPU	RSTDMA	RSTLCM	RSTLCD	RSTLED	RSTTKS	RSTCMP	RSTADC	0000,0000
RSTCR4	Reset Control Register 4	7EFE9EH	-	-	-	-	-	-	-	RSTMDU	xxxx,xxx0
RSTCR5	Reset Control Register 5	7EFE9FH	-	-	-	-	-	-	-	-	xxxx,xxxx

8.1.1 Watchdog Control Register (WDT_CONTR)

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

WDT_FLAG: Watchdog Overflow Flag

When watchdog overflow occurs, the hardware automatically sets this bit to 1, which needs to be cleared by software. EN_WDT: Watchdog enable bit

0: no effect on the microcontroller

1: Start watchdog timer

CLR_WDT: Watchdog timer clearing

0: no effect on the microcontroller

1: Clear the watchdog timer, hardware reset this bit automatically IDL_WDT: Watchdog control bit in IDLE mode

0: Watchdog stops counting in IDLE mode.

1: Watchdog continues counting in IDLE mode.

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

WDT_PS[2:0]	crossover factor	Overflow time at 12M mains	Overflow time at 20M mains
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 seconds	≈ 629 ms
101	64	≈ 2.10 seconds	≈ 1.26 seconds
110	128	≈ 4.20 seconds	≈ 2.52 seconds
111	256	≈ 8.39 seconds	≈ 5.03 seconds

The watchdog overflow time is calculated as follows:

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

8.1.2 IAP Control Register (IAP_CONTR)

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

SWBS: Software Reset Boot Selection

0: Code execution starts from the user programme area after a software reset. The data in the user data area remains unchanged.

1: Code execution starts from the system ISP area after a software reset. The data in the user data area is initialised.

SWRST: Software Reset Trigger Bit 0: No effect on microcontroller

1: Trigger software reset

8.1.3 Reset Configuration Register (RSTCFG)

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

ENLVR: Low voltage reset control bit

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

1: Enable low voltage reset. Automatically reset when the system detects a low voltage event P54RST: RST pin function selection

0: RST pin is used as a normal I/O port (P54)

1: RST pin is used as reset pin **(low level reset)**

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

LVDS[1:0]	Low Voltage Detection Threshold Voltage
00	2.0V
01	2.4V
10	2.7V
11	3.0V

8.1.4 Reset Flag Register (RSTFLAG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RSTFLAG	7EFE99H	-			LVDRSTF	WDTRSTF	SWRSTF	ROMOVF	EXRSTF

LVDRSTF: LVD Low Voltage
Reset Flag Read 0:
meaningless
Read 1: current reset is triggered by LVD low voltage reset
(power-on reset default value is 1) Write 0: no effect
Write 1: Clear LVDRST flag bit

WDTRSTF: Watchdog reset flag
Read 0: meaningless (power-on reset
defaults to 0) Read 1: current reset is
triggered by watchdog overflow
Write 0: no effect
Write 1: Clear WDTRST flag bit

SWRSTF: soft reset
flag read 0: no
significance
Read 1: The current reset is triggered by a software write of SWRST (IAP_CONTR.5)
(the default value of user programme reset is 1) Write 0: No effect
Write 1: Clear the SWRST flag bit

ROMOVF: code area overflow flag
Read 0: meaningless (power-on reset defaults to 0)
Read 1: The current reset was triggered by an overflow in the code
area caused by the CPU executing code into a non-program area Write
0: No effect
Write 1: Clear the ROMOV flag bit

EXRSTF: External Reset Flag
Read 0: meaningless (power-on reset defaults to 0)
Read 1: The current reset is triggered by the external
reset pin (P5.4/RST) being pulled low Write 0: No
effect
Write 1: Clear the EXRST flag bit

Instructions for soft resetting the user program to the system area for USB-ISP download:
P3.2 needs to be grounded at the same time to enter the USB-ISP download mode when powering up, in order to facilitate the user to control the ISP download independently, it is added that

When the user programme is soft reset to the system area, the USB-ISP download can be carried out without grounding P3.2. The judgement of this function is to judge whether the SWRSTF register bit is 1 or not after entering the ISP, if it is 1, it means the user soft reset to the system area, then it does not need P3.2 grounding, otherwise it needs P3.2 grounding. If user needs to soft reset to system area or perform USB-ISP download after key reset or watchdog reset, keep the SWRSTF register bit as 1. Otherwise, please write 1 to the SWRSTF register bit to clear the SWRSTF during user code initialisation.

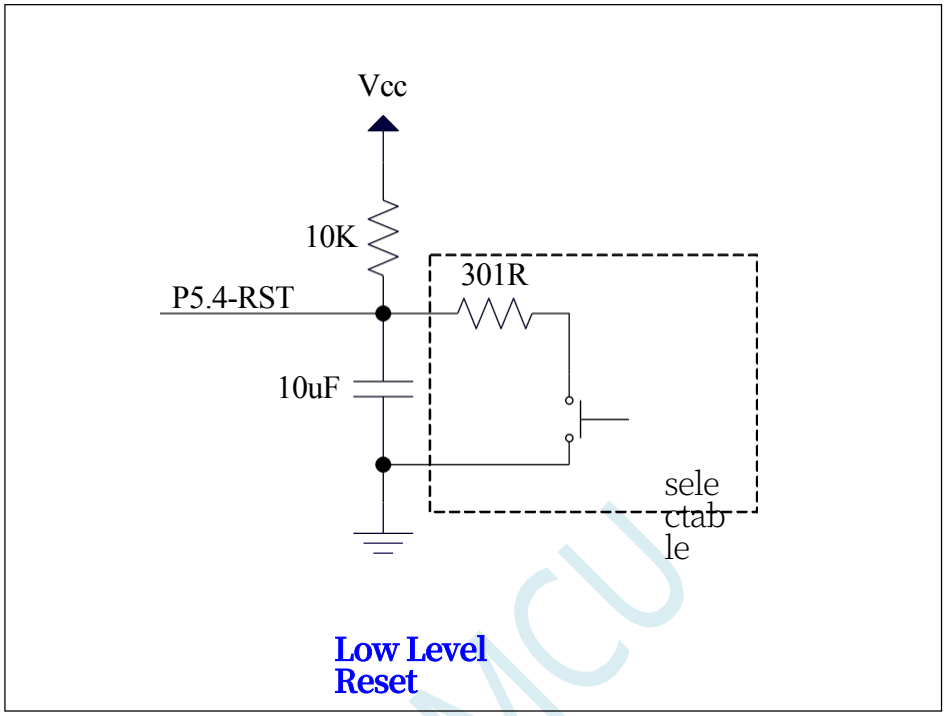
8.1.5 Reset Control Register (RSTCRx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RSTCR0	7EFE9AH	-	-	-	-	RSTTM34	TSTTM2	-	RSTTM01
RSTCR1	7EFE9BH	-	-	-	-	RSTUR4	RSTUR3	RSTUR2	RSTUR1
RSTCR2	7EFE9CH	RSTCAN2	RSTCAN	RSTLIN	RSTRTC	RSTPWMB	RSTPWMA	RSTI2C	RSTSPI
RSTCR3	7EFE9DH	RSTFPU	RSTDMA	RSTLCM	RSTLCD	RSTLED	RSTTKS	RSTCMP	RSTADC
RSTCR4	7EFE9EH	-	-	-	-	-	-	-	RSTMDU

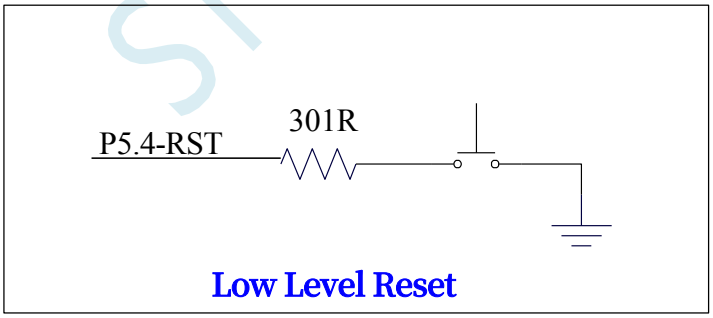
RSTTMn: TIMER0/1/2/3/4 reset control bit RSTUARTn: UART1/2/3/4 reset control bit RSTSPI: SPI reset control bit RSTI2C: I2C (SSB) reset control bit RSTPWMA: PWMA reset control bit RSTPWMB: PWMB reset control bit RSTRTC: RTC RSTLIN: LIN reset control bit RSTCAN: CAN reset control bit RSTCAN2: CAN2 reset control bit RSTADC: ADC reset control bit RSTCMP: CMP (Comparator) reset control bit RSTTKS: TKS (TouchKey) reset control bit RSTLED: LED driver reset control bit RSTLCD: LCD driver reset control bit RSTLCM: LCM driver reset control bit RSTDMA: DMA reset control bit RSTFPU: FPU reset control bit RSTMDU: MDU32 reset control bit

Write 1: Reset the corresponding peripheral module module, software clear required

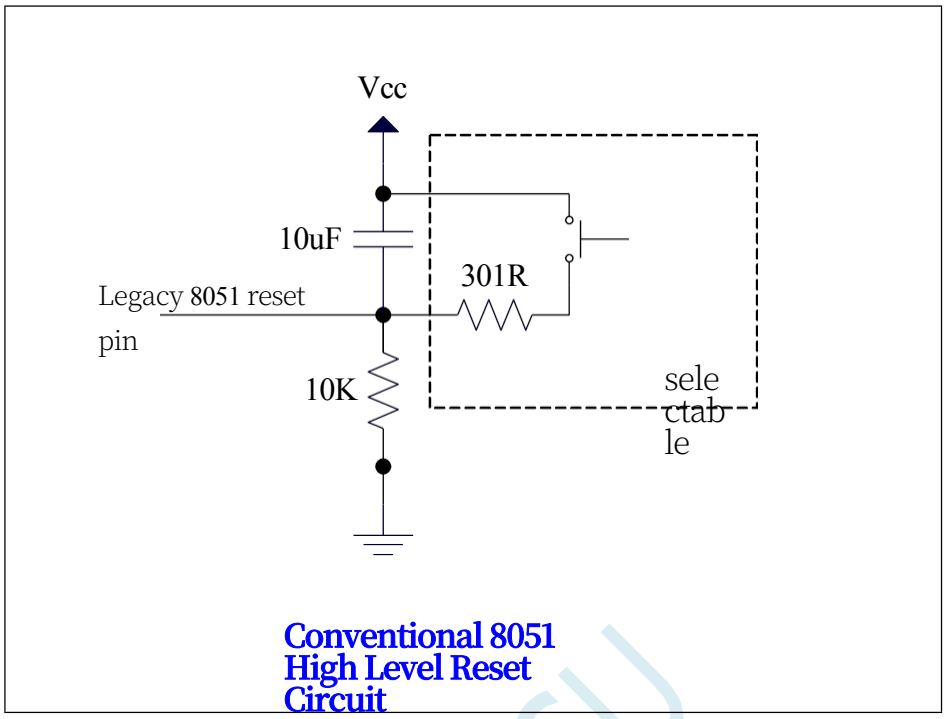
8.1.6 Low-level power-up reset reference circuit (not normally required)



8.1.7 Low Level Key Reset Reference Circuit



8.1.8 Legacy 8051 High-Level Power-Up Reset Reference Circuit



Conventional 8051 High Level Reset Circuit

The above figure shows the high level reset circuit of traditional 8051, the reset of STC32G is low level reset, which is different from the traditional reset circuit.

8.2 System Power Management

notation	descriptions	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

8.2.1 Power Control Register (PCON)

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

LVDF: Low Voltage Detection Flag Bit. When the system detects a low voltage event, the hardware automatically sets this bit to 1 and requests an interrupt to the CPU. This bit needs to be cleared by user software.

POF: Power On Flag Bit. When the hardware automatically sets this bit to 1.

PD: Power-down mode control bit

0: no effect

1: The microcontroller enters the clock stop mode/power down mode, the CPU and all peripherals stop working. The hardware is automatically cleared after wake-up.

(**Note: In Clock Stop mode, the CPU and all peripherals stop working, but the data in SRAM and XRAM are always maintained**).

IDL: IDLE (idle) mode control

bit 0: no effect

1: The microcontroller enters IDLE mode, only the CPU stops working, other peripherals are still running. Hardware is automatically cleared after wake-up

8.3 Power-down wake-up timer

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

8.3.1 Wake-up from power-down timer count registers (WKTCL, WKTCH)

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

WKTEN: Wake-up from power-down timer enable control bit 0: disable Wake-up from power-down timer

1: Enable power-down wake-up timer

If the built-in Wake-on-Drop timer of STC32G series MCU is allowed (WKTEN position 1 in WKTCH register is set by software), when the MCU enters into the power-down mode/shutdown mode, the Wake-on-Drop timer starts counting, and when the counted value is equal to the value set by the user, Wake-on-Drop timer wakes the MCU up. After the MCU wakes up, the programme starts to execute from the next statement of the last statement that set the MCU to enter the power-down mode. After waking up from power-down mode, you can read the contents of WKTCH and WKTCL to get the sleep time of MCU in power-down mode.

Please note that the value that the user writes to the registers {WKTCH[6:0],WKTCL[7:0]} must be 1 less than the actual count value, e.g., if the user wants to count 10 times, then 9 should be written to the registers {WKTCH[6:0],WKTCL[7:0]}. Similarly, if the user wants to count 32767 times, 7FFEH (i.e. 32766) should be written to {WKTCH[6:0],WKTCL[7:0]}. (**The count value 0 and the count value 32767 are internally reserved and cannot be used by the user**.)

The internal wake-up timer has its own internal clock, where the time for the wake-up timer to count once is determined by this clock. The clock frequency of the internal Wake-on-Wake-up Timer is about 32KHz, but of course the error is large. Users can read the contents of RAM area F8H and F9H (F8H stores the high byte of the frequency, F9H stores the low byte) to get the clock frequency of the internal Wake-on-Wake-up Timer recorded in the factory.

The formula for calculating the counting time of the Wake-on-Drop Dedicated Timer is shown below: (F_{wt} is the clock frequency of the internal Wake-on-Drop Dedicated Timer that we obtained from RAM areas F8H and F9H)

$$\text{Wake-up from power-down timer timing time} = \frac{(16 \times \text{counts} + 106) \times}{F_{wt}} \text{ (microseconds)}$$

Assuming F_{wt} = 32KHz, there are:

{WKTCH[6:0], WKTCL[7:0]}	Wake on Power Down Dedicated Timer Count Time
~~0 (internal reservation)~~	
1	$10^6 \div 32K \times 16 \times (1+1) \approx 1$ ms
9	$10^6 \div 32K \times 16 \times (1+9) \approx 5$ ms
99	$10^6 \div 32K \times 16 \times (1+99) \approx 50$ ms
999	$10^6 \div 32K \times 16 \times (1+999) \approx 0.5$ sec.
4095	$10^6 \div 32K \times 16 \times (1+4095) \approx 2$ sec.

32766	$10^6 \div 32K \times 16 \times (1+32766) \approx 16$ sec.
~~32767 (internally reserved)~~	

8.4 sample procedure

8.4.1 Watchdog Timer Applications

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                  // see download software for header files

void main()
{
    EAXFR = 1;                                       //Enable access to XFR
    CKCON = 0x00;                                    //Set the external data bus speed to fastest
    WTST = 0x00;                                     //set the program code wait parameter.
                                                     //Assign 0 to set the CPU to execute the programme as fast as
                                                     possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

//  WDT_CONTR = 0x23.                                // Enable watchdog, overflow time is about 0.5s.
    WDT_CONTR = 0x24.                                //Enable watchdog, overflow time is about 1s.
// WDT_CONTR = 0x27;                                 //     enable     watchdog,
    overflow time is about 8s P32 = 0;               //test port

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

        Display().                                   //Display module
        Scankey().                                   // key scanning module
        MotorDriver();                               //Motor driver module
    }
}
```

8.4.2 Soft reset for custom downloads

```
//Tested operating frequency is 11.0592MHz
```

```
#include "stc32g.h"                          // see download software for header files

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign 0 to set the CPU to execute the programme as fast as possible.

    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 0 at the same time.
        }
    }
}
```

8.4.3 Low Voltage Detection

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // See download software for header files

#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

void Lvd_Isr() interrupt 6
{
    LVDF = 0;                                // clear interrupt flag
```

```
    P32 = ~P32;                    //test port
}

void main()
{
```

```
    eaxfr = 1; ckcon =
    0x00; wtst = 0x00.                  // Enable access to XFR
                                        //Set external data bus speed to fastest
                                        //Set the parameter for waiting for the programme
                                        code.
                                        //Assign 0 to set the CPU to execute the programme
                                        as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    LVDF = 0;                           // Test port
//  rstcfg = enlvr | lvd3v0; rstcfg = lvd3v0.
                                        //Low-voltage reset at 3.0V, no LVD interrupt.
    ELVD = 1.                           // Enable low voltage interrupt at 3.0V
    EA = 1;                             // Enable LVD Interrupt


    while (1);
}
```

8.4.4 Battery saving mode

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void INT0_Isr() interrupt 0
{
    P34 = ~P34;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as
                                        possible.

    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, used to wake up MCU.
    EA = 1;
    _nop_().
    _nop_().
    _nop_().
    _nop_().
    IDL = 1;                        //MCU enters IDLE mode
//  PD = 1;                         //MCU enters power-down mode
    _nop_().
    _nop_().
    _nop_().
    _nop_().
    P35 = 0;

    while (1);
}
```

8.4.5 Use INT0/INT1/INT2/INT3/INT4 pin interrupt to wake up in power saving mode

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 // see download software for header files
#include "intrins.h"

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()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    IT0 = 0; // Enable INT0 rising and falling edge interrupts.   //enable INT0 rising and falling edge interrupt
//  IT0 = 1; // enable INT0 falling edge interrupt                //Enable INT0 falling edge interrupt.
    EX0 = 1;                                                     //Enable INT0 interrupt

    IT1 = 0.                                                     //Enable INT1 rising and falling edge interrupts
//  IT1 = 1; // enable INT1 falling edge interrupt               //Enable INT1 falling edge interrupt.
    EX1 = 1;                                                     //Enable INT1 interrupt

    EX2 = 1; //Enable INT2 falling edge interrupt.               //Enable INT2 falling edge interrupt.
    EX3 = 1; //Enable INT3 falling edge interrupt.               //Enable INT3 falling edge interrupt.
    EX4 = 1; //Enable INT4 falling edge interrupt.               //Enable INT4 falling edge interrupt.

    EA = 1;

    PD = 1; //MCU enters power-down mode.                        //MCU enters power-down mode
    _nop_().                                                     //After waking up in power-down mode, the MCU will execute this statement first.
                                                                 //Then enter the interrupt service routine
    _nop_().
    _nop_().
    _nop_().

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

8.4.6 Use T0/T1/T2/T3/T4 pin interrupt to wake up to power saving mode

```
//Tested operating frequency is 11.0592MHz
```

```
#include "stc32g.h"                    //see download software for header files
#include "intrins.h"
```

```
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()
{
    EAXFR = 1;                                   //Enable access to XFR
    CKCON = 0x00;                                //Set the external data bus speed to fastest
    WTST = 0x00;                                 //set the program code wait parameter.
                                                 //Assign 0 to set the CPU to execute the programme as fast as
                                                 possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    TMOD = 0x00.
    TL0 = 0x66.                                  //65536-11.0592m/12/1000
    TH0 = 0xfc.
    TR0 = 1;                                     //start the timer
    ET0 = 1;                                     //Enable timer interrupt

    TL1 = 0x66.                                  //65536-11.0592m/12/1000
    TH1 = 0xfc.
    TR1 = 1;                                     //start the timer
    ET1 = 1;                                     //Enable timer interrupt

    T2L = 0x66.                                  //65536-11.0592m/12/1000
    T2H = 0xfc.
```

```
    T2R = 1;                                    //start the timer
```

```
    ET2 = 1;                                    //Enable timer interrupt

    T3L = 0x66.                                 //65536-11.0592m/12/1000
    T3H = 0xfc.
    T3R = 1;                                    //start the timer
    ET3 = 1;                                    //Enable timer interrupt

    T4L = 0x66.                                 //65536-11.0592m/12/1000
    T4H = 0xfc.
    T4R = 1;                                    //start the timer
    ET4 = 1;                                    //Enable timer interrupt

    EA = 1;

    PD = 1; //MCU enters power-down mode.       //MCU enters power-down mode
    _nop_().                                    //Waking up from power-down will not enter the interrupt service routine immediately, //the interrupt service routine will not enter the interrupt service routine immediately after waking up from power-down.
                                                //Instead, it waits until the timer overflows before entering the interrupt service routine
    _nop_().
    _nop_().
    _nop_().

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

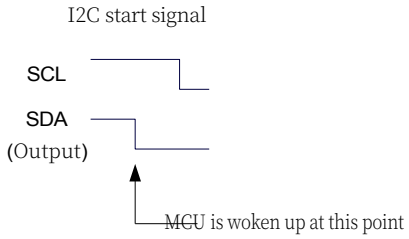
8.4.7 Wake-up to power saving mode using RxD/RxD2/RxD3/RxD4 pin interrupts

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             //see download software for header files
#include "intrins.h"

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}
```

```
void main()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
```

```
    WTST = 0x00.                    //Set the parameters for the programme code wait.
                                    //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    s1_s1 = 0; s1_s0 = 0.           //RXD/P3.0 Wake-up on falling edge
//  s1_s1 = 0; s1_s0 = 1.           //RXD_2/P3.6 Wake-up on falling edge
//  s1_s1 = 1; s1_s0 = 0.           //RXD_3/P1.6 Wake-up on falling edge
//  s1_s1 = 1; s1_s0 = 1.           //RXD_4/P4.3 Wake-up on falling edge

    S2_S = 0;                       //RXD2/P1.0 Wake-up on falling edge
//  S2_S = 1;                       //RXD2_2/P4.6 Wake-up on falling edge

    S3_S = 0;                       //RXD3/P0.0 Wake-up on falling edge
//  S3_S = 1;                       //RXD3_2/P5.0 Wake-up on falling edge

    S4_S = 0;                       //RXD4/P0.2 Wake-up on falling edge
//  S4_S = 1;                       //RXD4_2/P5.2 Wake-up on falling edge

    ES = 1;                         //Enable serial port interrupt
    ES2 = 1;                        //Enable serial port interrupt
    ES3 = 1;                        //Enable serial port interrupt
    ES4 = 1;                        //Enable serial port interrupt
    EA = 1;

    PD = 1;                         //MCU enters power-down mode
    _nop_().                        //Wake-up from power-down does not enter the interrupt service routine, //Wake-up from power-down does not enter the interrupt service routine.
    _nop_().
    _nop_().
    _nop_().

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

8.4.8 Wake-up MCU power saving mode using I2C SDA pin




```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void i2c_isr() interrupt 24
{
    I2CSLST &= ~0x40.
}

void main()
{
    eaxfr = 1; ckcon =                       // Enable access to XFR
    0x00; wtst = 0x00.                       //Set external data bus speed to fastest
                                             //Set the parameter for waiting for the programme code.
                                             //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    I2C_S1 = 0; I2C_S0 = 0.                  //SDA/P1.4 Wake-up on falling edge
//  i2c_s1 = 0; i2c_s0 = 1.                  //SDA_2/P2.4 Wake-up on falling edge
//  i2c_s1 = 1; i2c_s0 = 1.                  //SDA_4/P3.3 Wake-up on falling edge

    i2ccfg = 0x80;                           // Enable Slave Mode for I2C Module
    i2cslcr = 0x40; ea =                     // Enable start signal interrupt
    1.

    PD = 1;                                  //MCU enters power-down mode
    _nop_().                                 //Wake-up from power-down does not enter interrupt service routine
    _nop_().
    _nop_().
    _nop_().
```

```
    while (1)
```

```
    {
        P11 = ~P11.
    }
}
```

8.4.9 Wake up to power saving mode using the power-down wake-up timer

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign 0 to set the CPU to execute the programme as fast as possible.

    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 approximately 1 second.
    WKTCH = 0x87.

    while (1)
    {
        _nop_().
        _nop_().
        PD = 1; //MCU enters power-down mode.  //MCU enters power-down mode
        _nop_().
        _nop_().
        _nop_().
        _nop_().
        P11 = ~P11.
    }
}
```

8.4.10 LVD interrupt wake-up power saving mode, recommended to use with power-down wake-up timer

It is not recommended to activate the LVDs and comparators in the clock-off power-saving mode, otherwise the hardware system will also automatically activate the internal 1.19V high accuracy

This high precision reference source has corresponding anti-temperature drift and tuning lines, which will add about 300uA extra power consumption, while the MCU only consumes about 0.4uA current when it enters the clock stop mode with 3.3V operating voltage, so it is not recommended to turn on the LVD and comparator when it enters the clock stop mode. If you do need to use them, it is recommended to turn on the power-down wake-up timer, which will only increase the power consumption by about 1.4uA, which is generally acceptable for the system. Let the wake up from power down timer wake up the MCU every 5 seconds, after wake up, we can use LVD, comparator, ADC to detect the external battery voltage, the detection work takes about 1mS before entering the clock stop/power saving mode, the average current increase is less than 1uA, then the overall power consumption is about 2.8uA (0.4uA + 1.4uA + 1uA).

//Tested operating frequency is *11.0592MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                                  // See download software for header files
#include "intrins.h"

#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

void LVD_Isr() interrupt 6
{
    LVDF = 0;                                        // clear interrupt flag
    P10 = !P10;                                      //test port
}

void main()
{
    EAXFR = 1;                                       //Enable access to XFR
    CKCON = 0x00;                                    //Set the external data bus speed to fastest
    WTST = 0x00;                                     //set the program code wait parameter.
                                                     //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    LVDF = 0;                                        //Power up needs to clear the interrupt flag
    rstcfg = lvd3v0.                                 //set LVD voltage to 3.0V
    ELVD = 1; //Enable LVD interrupt.                //Enable LVD interrupt
    EA = 1;

    PD = 1; //MCU enters power-down mode.             //MCU enters power-down mode
```

```
    _nop_();                    //Enter the interrupt service routine immediately after waking
    up from power-down.
    _nop_().
    _nop_().
    _nop_();

    while (1)
```

```
    {
        P11 = ~P11.
    }
}
```

8.4.11 Comparator interrupt wake-up power saving mode, recommended to use with power-down wake-up timer

It is not recommended to start the LVD and comparator in the power saving mode, otherwise the hardware system will also automatically start the internal 1.19V high precision reference source, which has the corresponding anti-temperature drift and tuning lines, and will increase the power consumption by about 300uA, while the MCU only consumes about 0.4uA current at the 3.3V operating voltage when it enters into the power saving mode, so it is not recommended to turn on the LVD and comparator when entering into the power saving mode. Therefore, it is not recommended to turn on the LVD and comparator when entering the clock stop mode. If you really need to use them, it is recommended to turn on the power-down wake-up timer, which will only increase the power consumption by about 1.4uA, which is generally acceptable for the system. Let the wake up from power down timer wake up the MCU every 5 seconds, and then use LVD, comparator and ADC to detect the external battery voltage, the detection takes about 1mS before entering the clock stop/power saving mode, the average current increase is less than 1uA, and the overall power consumption is about 2.8uA (0.4uA + 1.4uA + 1uA).

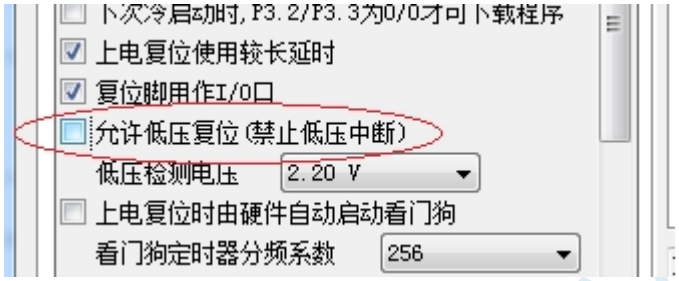
```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void CMP_Isr() interrupt 21
{
    CMPIF = 0;                          // clear interrupt flag
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.

    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;
    CMPEN = 1;                          //Enable comparator module
    pie = 1; nie = 1.                   //Enable comparator edge interrupt
    CMPOE = 1;                          //Enable comparator output
    EA = 1;

    PD = 1; //MCU enters power-down mode.     //MCU enters power-down mode
```

```
    _nop_();                                    // Enter the interrupt service routine immediately after waking
    up from power-down.
    _nop_().
    _nop_().
    _nop_().

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

8.4.12 Detecting the operating voltage (battery voltage) using the LVD function

If you need to use the LVD function to detect the battery voltage, you need to remove the low-voltage reset function during the ISP download, as shown below, the tick box of the hardware option "Allow low-voltage reset (disable low-voltage interrupt)" needs to be removed




```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             // see download software for header files
#include "intrins.h"

#define FOSC            11059200UL              //Define as unsigned long integer to avoid overflow.
#define T1MS            (65536 - FOSC/4/100)

#define     LVD2V0          0x00                //LVD@2.0V
#define     LVD2V4          0x01                //LVD@2.4V
#define     LVD2V7          0x02                //LVD@2.7V
#define     LVD3V0          0x03                //LVD@3.0V

void delay()
{
    int i;

    for (i=0; i<100; i++)
    {
        _nop_().
        _nop_().
        _nop_().
        _nop_().
    }
```

```
}

void main()
{
```

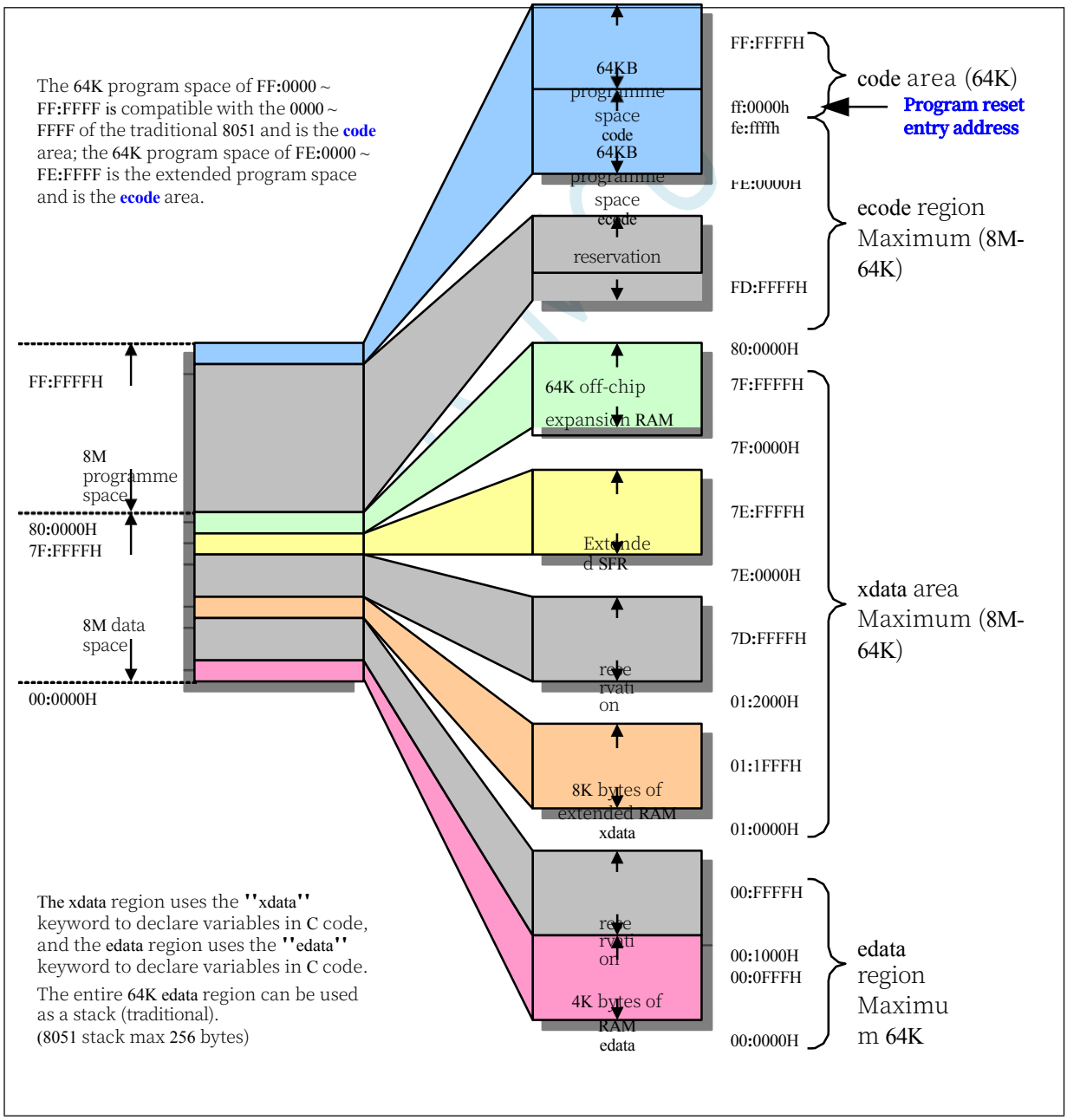
```
    unsigned char power.

    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    LVDF = 0;
    RSTCFG = LVD3V0.

    while (1)
    {
        power = 0x0f.

        RSTCFG = LVD3V0.
        delay();
        LVDF = 0.
        delay();
        if (LVDF)
        {
            power >>= 1;
            RSTCFG = LVD2V7.
            delay();
            LVDF = 0.
            delay();
            if (LVDF)
            {
                power >>= 1;
                RSTCFG = LVD2V4.
                delay();
                LVDF = 0.
                delay();
                if (LVDF)
                {
                    power >>= 1;
                    RSTCFG = LVD2V0.
                    delay();
                    LVDF = 0.
                    delay();
                    if (LVDF)
                    {
                        power >>= 1;
                    }
                }
            }
        }
        RSTCFG = LVD3V0.
```

```
            P2 = ~power.        //P2.3~P2.0 Display battery level
        }
}
```

9 Memory (32-bit access, 16-bit access, 8-bit access)

The program memory and data memory of STC32G series microcontrollers are uniformly addressed.STC32G series microcontrollers provide 24-bit addressing space and can access up to 16M memory (8M data memory + 8M program memory). Since there is no bus for accessing external programme memory, all programme memories of the microcontroller are on-chip Flash memories and cannot access external programme memories.

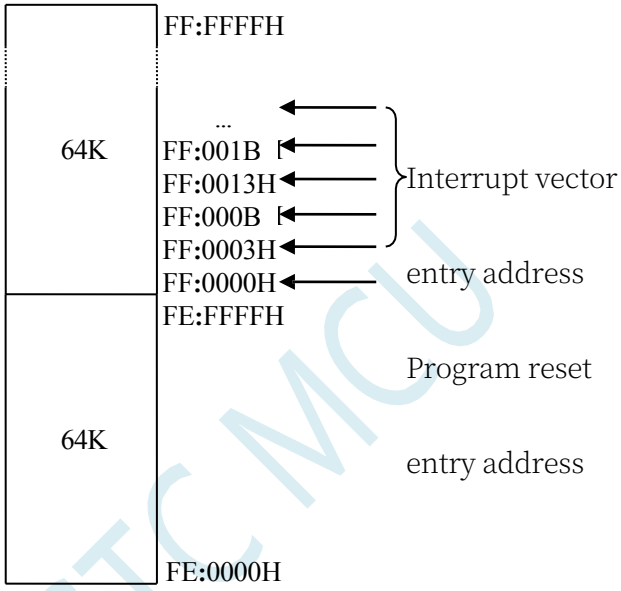
The STC32G series microcontrollers have a large data memory integrated inside.The data memory inside the STC32G series microcontrollers is physically and logically divided into two address spaces: internal RAM (edata) and internal extended RAM (xdata).



9.1 program memory

The programme memory is used to store information such as user programmes, fixed and unchanging data, and tables.

Microcontroller Series	Flash programme memory (ROM)	address range
STC32G12K128 Series	128K bytes	FE:0000H to FF:FFFFH
STC32G8K64 Series	64K bytes	FF:0000H to FF:FFFFH
STC32F12K60 Series	60K bytes	FF:0000H to FF:EFFFH



(Comparison of STC32G series and conventional 8051 interrupt entry address)

	STC32G Series	Legacy 8051
Reset Entry Address	FF:0000H	0000H
INT0 Interrupt entry address	FF:0003H	0003H
TIMER0 Interrupt entry address	FF:000BH	000BH
INT1 Interrupt entry address	FF:0013H	0013H
TIMER1 Interrupt entry address	FF:001BH	001BH
UART Interrupt Entry Address	FF:0023H	0023H

After the microcontroller is reset, the content of the programme counter (PC) is FF:0000H, and the execution of the programme starts from the FF:0000H unit. Also the entry address of the interrupt service program (also known as the interrupt vector) is located in the programme memory cell. In the programme memory, each interrupt has a fixed entry address, and when the interrupt occurs and is responded to, the microcontroller will automatically jump to the corresponding interrupt entry address to execute the program. The entrance address of the interrupt service programme for external interrupt 0 (INT0) is FF:0003H, the entrance address of the interrupt service programme for timer/counter 0

(TIMER0) is FF:000BH, the entrance address of the interrupt service programme for external interrupt 1 (INT1) is FF:0013H, the entrance address of the interrupt service programme for timer/counter 1 (TIMER1) is FF:001BH. for Timer/Counter1 (TIMER1) is FF:001BH, and so on. For more information about the entry address (interrupt vector) of the interrupt service program, please refer to the Interrupt Introduction section.

Since the interval between neighbouring interrupt entry addresses is only 8 bytes, it is generally not possible to save the complete interrupt service routine, so the

The address area of the interrupt response holds an unconditional transfer instruction that points to the space that actually holds the interrupt service routine for execution.

All STC32G series microcontrollers include Flash data memory (EEPROM). Data can be read/written in byte units and erased in 512-byte page units, and can be programmed and erased online more than 100,000 times, which improves flexibility and convenience of use.

9.1.1 Program read wait control register (WTST)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
WTST	E9H	WTST[7:0]							

WTST[7:0]: CPU wait time control for reading programme memory

WTST[7:0]	Waiting Clock Count
0	0 clocks
1	1 Clock
...	...
7	7 clocks
8	reservation
...	reservation
255	reservation

Number of actual execution clocks per instruction = number of instruction clocks + number of wait clocks in program memory

Note: For STC32G12K128 series of version A chips, the default value of WTST register is 7 at power-on, when the user's operating frequency is below 35MHz, it is recommended to modify it to 0, which can speed up the CPU to run the programme.

9.2 Data memory (32-bit access, 16-bit access, 8-bit access)

The STC32G's edata area provides single-clock read/write access to 32-BIT/16-BIT/8-BIT data, and the xdata area provides read/write access to 16-BIT/8-BIT data. edata SRAM is currently designed to have a maximum storage depth of 64K bytes, and xdata SRAM is designed to have a maximum storage depth of 8M bytes. The SRAM in the xdata area has a maximum storage depth of 8M bytes.

In the future, new 32-BIT SFR32 special function registers will be added (e.g., ADC_DATA32), such as mapping the logical address of the SFR32 to the edata area, 32-BIT/16-BIT/8-BIT access to the added special function registers can be supported;

Special function registers 16-BIT SFR16 will be added in the future (e.g. ADC_DATA16), e.g. by mapping the logical address of SFR16 to the xdata area to support 16-BIT/8-BIT access to additional special function registers.

The RAM integrated inside the STC32G series microcontrollers can be used to store intermediate results of program execution and process data.

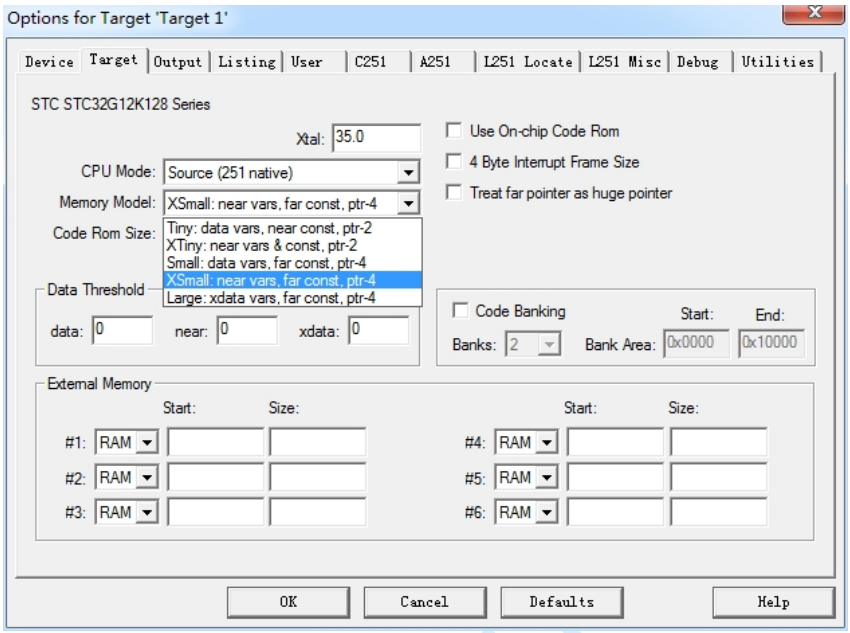
Microcontroller Series	Internal RAM (edata)	Internal Expansion RAM (xdata)
STC32G12K128 Series	4K bytes	8K bytes
STC32G8K64 Series	2K bytes	6K bytes
STC32F12K60 Series	8K bytes	4K bytes

EDATA: single clock access, fast access speed, can be used as a stack, high cost;

XDATA: Access requires 2~3 clocks, slow access speed, addressing space up to 8M, low cost.

9.2.1 Keil Options Memory Model Settings

For the memory modes of the STC32G series projects, there are five modes in the Keil environment as shown below:



The various models are compared in the table below:

Memory Model	Default variable type (Data memory)	Default Constant Type (programme memory)	Default Pointer Variable	Pointer access range
Tiny mode	data	near	2 bytes	00:0000 ~ 00:FFFF
XTiny Mode	edata	near	2 bytes	00:0000 ~ 00:FFFF
Small mode	data	far	4 bytes	00:0000 ~ FF:FFFF
XSmall Mode	**edata**	**far**	**4 bytes**	**00:0000 ~ FF:FFFF**
Large mode	xdata	far	4 bytes	00:0000 ~ FF:FFFF

Since the program logical address of STC32G is from FE:0000H to FF:FFFFH, it needs to use 24-bit address line to access it correctly, the default constant type (programme memory type) must be "far" type, and the default pointer variable must be 4 bytes.

Therefore, it is not recommended to use "Tiny" and "XTiny" modes, and it is recommended to use "XSmall" mode, which defines the variables in the internal RAM (edata) by default, with fast access speed and 12K edata available for STC32G12K128 series chips; or "Small" mode, which has 12K edata available for use. We recommend to use "XSmall" mode, which defines variables in internal RAM (edata) by default, single clock access, fast access speed, and STC32G12K128 series chips have 12K edata can be used; or "Small" mode, which defines variables in internal RAM (data) by default, single clock access, fast access speed, (data is only 128 bytes, when the user needs to use the internal RAM, it is only 128 bytes, when the user needs to use the internal RAM, it is only 128 bytes. This mode defines variables in internal RAM (data) by default, with single clock access and fast access speed, (data is only 128 bytes by default, when the user needs more than 128 bytes of RAM, the Keil compiler will report an error, and then the user needs to switch the memory mode to XSmall); it is

not recommended to use the "Large" mode, although this mode can also correctly access the full 16M addressing space of the STC32G, but the "Large" mode can not be used for accessing all the 16M addressing space. Although this mode can also access all 16M addressing space of STC32G correctly, "Large" mode defines variables in internal extended RAM (xdata) by default, which requires 2~3 clocks for accessing, and the access speed is slow.

C language variable declaration recommendations for STC32G series microcontrollers:

1, when the user variable demand is small, it is recommended not to use "edata, xdata" and other keywords to declare variables, but use the following way to declare variables directly:

```
charbCounter = 1;        //declare byte
variable    intwCounter = 100;    //declare
double-byte variable              longdwCounter =
0x1234; //declare 4-byte variable
```

Then set "Memory Model" to "XSmall" in the project options to let the compiler automatically assign declared variables to edata.
district

2、 When the user variable demand is close to or exceeds the size of (microcontroller edata capacity-1K), it is recommended to use the "xdata" keyword to force the exceeding part to be allocated to the xdata area, as shown below:

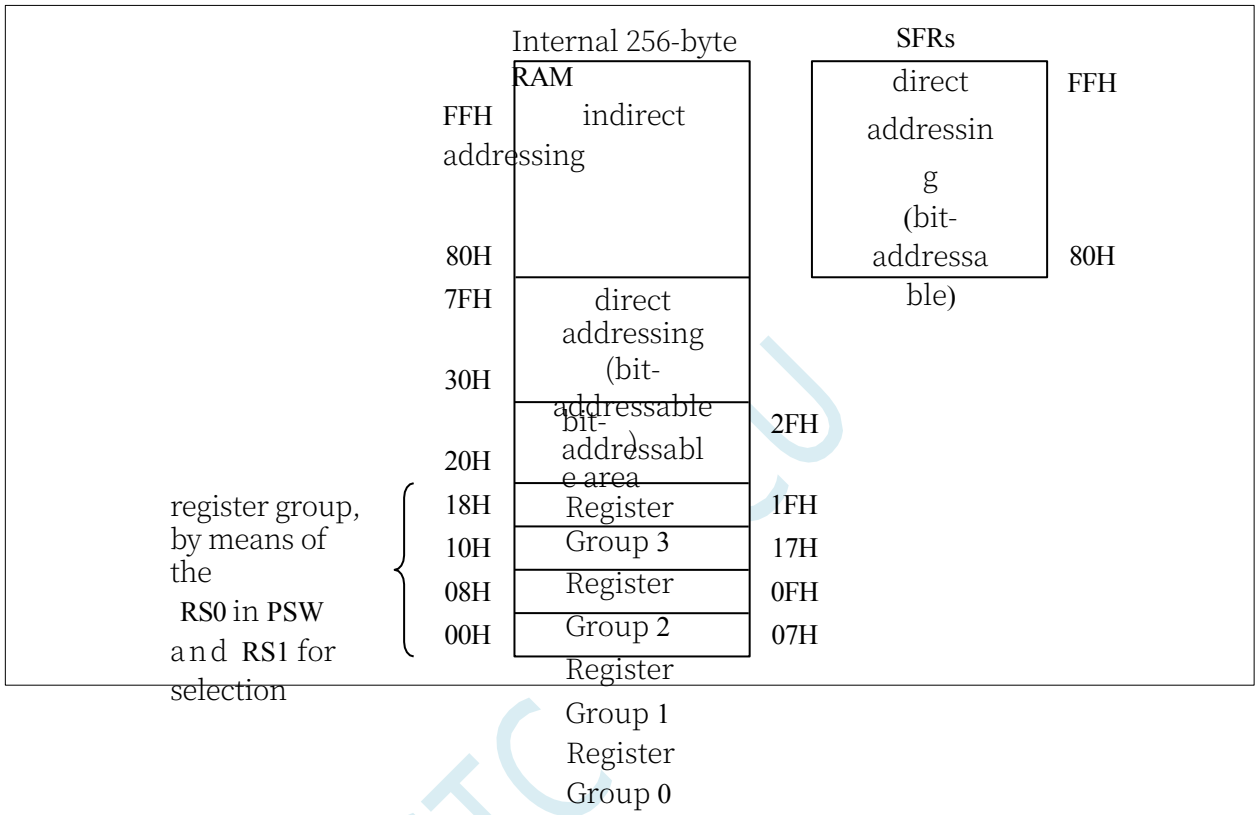
```
intxdata pBuffer = 5;          // use the xdata keyword to force allocation into the xdata region
```

9.2.2 Internal edata-RAM (declared in C with the keyword edata)

The internal edata-RAM is 4K bytes in total, and the 256 bytes at the low end of the 4K bytes are fully compatible with the 256 bytes of DATA of the 8051, which can be divided into two.

The low 128-byte data memory is compatible with the legacy 8051 and is directly addressable. Indirect addressing is also possible. The high 128-byte RAM (extended in 8052) can only be addressed indirectly. The special function registers are distributed in the 80H~FFH area and can only be addressed directly.

The structure of the low-end 256-byte RAM is shown below:



The STC32G12K128's stack is placed in EDATA, which is designed to have a theoretical depth of 64K, and actually puts 4K Bytes; the STC32G12K128's common extension XDATA, which is designed to have a theoretical depth of (8M-64K), and actually puts 8K Bytes; so the STC32G12K128's SRAM is 12K (4K) in total. edata + 8K xdata).

Declare variables in the EDATA area in C code to achieve 32-bit/16-bit/8-bit read/write operations with a single clock.

```
char (computing) edata bCounter.   // Declare byte variables in EDATA area (8-bit read/write operation with single clock)
int edata wCounter.   //Declare double-byte variables in EDATA area (16-bit read/write operation with single clock)
long edata dwCounter.   // Declare 4-byte variable in EDATA area (32-bit read/write operation with single clock)
```

8-bit/16-bit read/write operations can be achieved by declaring variables in the XDATA area in C code.

```
char (computing) xdata bCounter.   // Declare byte variables in XDATA area (3/2 clocks for 8-bit read/write operations)
int              xdata wCounter.   // Declare double-byte variables in the XDATA area (3/2 clocks for 16-bit read/write operations)
```

9.2.3 Programme Status Register (PSW)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PSW	D0H	CY	AC	F0	**RS1**	**RS0**	OV	-	P

CY: Feed Flag AC:
Auxiliary Feed Flag
F0: General Purpose
Flag OV: Overflow
Flag P: Even Check
Flag for ACC

RS1, RS0: Operating register selection bits

RS1	RS0	Operating register group (R0~R7)
0	0	Group 0 (00H-07H)
0	1	Group 1 (08H-0FH)
1	0	Group 2 (10H-17H)
1	1	Group 3 (18H-1FH)

The address of the bit-addressable area is from 20H~2FH, which is a total of 16 byte units. 20H~2FH units can be accessed by byte like ordinary RAM units, or any bit in the unit can be accessed individually, with a total of 128 bits, corresponding to a logical bit address range of 00H~7FH. The address range of the bit address is 00H~7FH, and the address of the lower 128 bytes in the internal RAM is 00H~7FH, which is actually fundamentally different; the bit address points to a bit, while the byte address points to a byte unit. 7FH, from the outside, the two addresses are the same, in fact, they have a fundamental difference; bit address points to a bit, while the byte address points to a byte unit, in the program using different instructions to distinguish.

9.2.4 Programme Status Register 1 (PSW1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PSW1	D1H	CY	AC	N	RS1	RS0	OV	Z	-

CY: Carry Flag (same as CY in PSW)
AC: Auxiliary feed flag (same as AC in PSW)
N: Negative flag bit of calculation result, N is 1 if the highest bit of the operation result is 1, otherwise it is 0 RS1, RS0: Work register selection bit (same as RS0, RS1 in PSW) OV: Overflow flag (same as OV in PSW)
Z: Zero flag bit, Z is 1 if the result is 0, otherwise it is 0.

9.2.5 Internal xdata-RAM (C declaration keyword is xdata)

In addition to the integrated expansion RAM, the STC32G series microcontrollers access the internal expansion RAM in the same way as the conventional 8051 microcontrollers access the external expansion RAM, but without affecting the signals on the P0 and P2 ports, as well as the RD, WR and ALE ports.

In assembly language, the internal extended RAM is accessed by the MOVX instruction.

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

In C, you can just declare the storage type using the xdata keyword. For example:

```
unsigned char xdata i.
```

(It is strongly recommended that variables not be declared using the pdata keyword)

The accessibility of the microcontroller's internal extended RAM is controlled by the EXTRAM bit in the Auxiliary Register AUXR.

9.2.6 Auxiliary Register (AUXR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	**EXTRAM**	S1BRT

EXTRAM: Extended RAM access control

0: Access to external extended RAM is disabled.

1: Enables access to external extended RAM.

9.2.7 External xdata-RAM

The STC32G series microcontrollers have the ability to expand the 64KB external data memory. The WR/RD/ALE signals should be valid during the access to the external data memory.The special function register BUS_SPEED, which controls the speed of the external 64K-byte data bus for STC32G series microcontrollers, is described as follows:

9.2.8 Bus speed control register (BUS_SPEED)

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

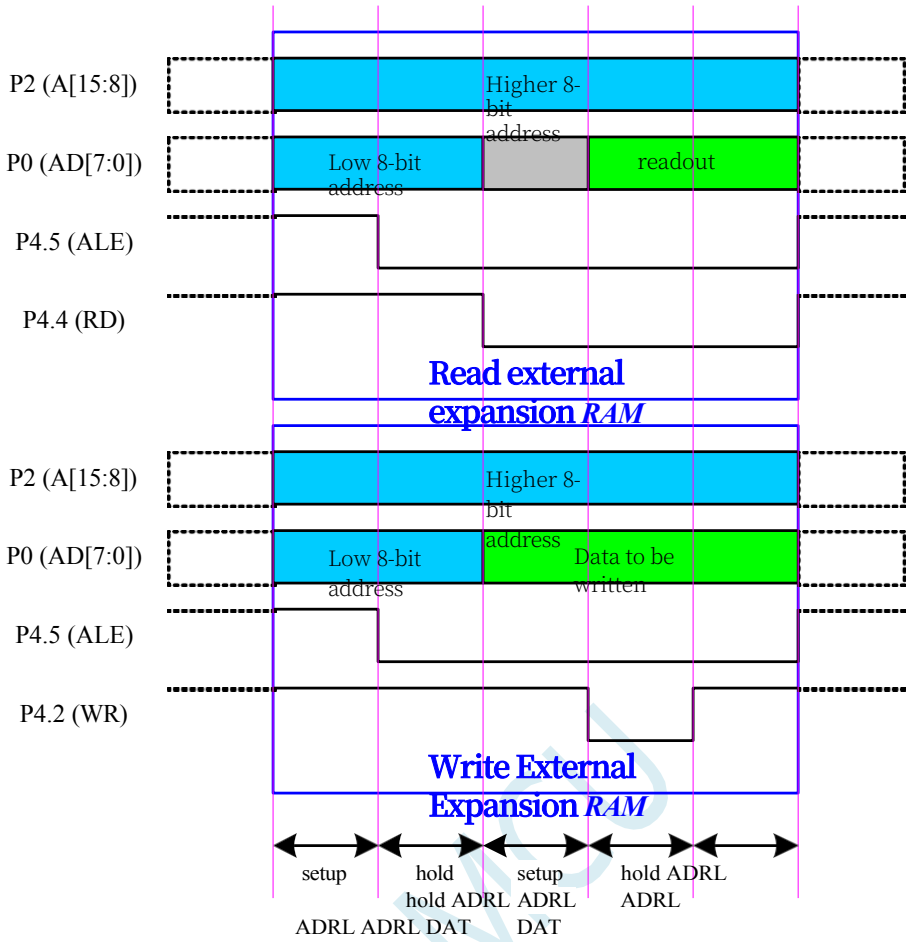
RW_S[1:0]: RD/WR control line select bits

00: P4.4 for RD, P4.2 for WR

x1: Reserved

SPEED[2:0]: bus read/write speed control (read/write data read/write control signal and data signal ready time and hold time)

The timing for reading and writing the external expansion RAM is shown below:



9.2.9 External Data Bus Clock Control Register (CKCON)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CKCON	EAH	-	-	-	T1M	T0M	CKCON[2:0]		

T1M:
Reserved
T0M:
Reserved

CKCON[2:0]: external data bus clock control register (power-on reset value is 7, recommended to set to 0)

CKCON[2:0]	Waiting Clock Count
0	0 clocks
1	1 Clock
...	...
7	7 clocks

9.2.10 Program memory cache control register (ICHERCR)

This feature is currently only available for the STC32F12K60 series.

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
ICHERCR	F7H	CON	HIT	CLR	-	-	-	-	EN

CON: CACHE control bit (write register only)

0: Enable configuration register access

1: Enable statistics register access

HIT: Hit/Miss statistic toggle, valid for writing when CON=1, used for testing (write registers only)

0: Selection of access miss count register (MISS)

1: Selection of the access hit count register (HIT)

The miss register and hit register are both 32 bits wide (4 bytes) and require 4 consecutive reads from the ICHERCR to obtain the

CLR: Clear hit/miss statistics, valid for writing when CON=1, used for testing (write registers only)

0: Write 0 No action

1: Write 1 Clear statistics

EN: CACHE function control bit, write valid when CON=0. Writes to this bit are protected by TA register writes (register writes only)

0: Disable CACHE function

1: Turn on the CACHE function

Note: Due to the hardware design of multiple cache lines, each time the CACHE function is turned off and then on again, it will need to wait for about 64 CPU clocks, and the hardware will automatically stop the CPU instruction instruction until the end of the 64 clocks. No additional processing is required by the user.

The reading function of the ICHERCR register depends on the setting values of CON and HIT. When CON=1 and HIT=0, the content read by ICHERCR register is the statistic value of the number of misses; when CON=1 and HIT=1, the content read by ICHERCR register is the statistic value of the number of hits.

Programming code to turn on the CACHE function:

```
CLR     EA          ;Turn off interrupts (required)
MOV     TA,#0AAH    Write Trigger Command Sequence 1
                    No other instructions can be given here.
MOV     TA,#55H     Write Trigger Command Sequence 2
                    No other instructions can be given here.
MOV     ICHERCR,#1  The write protection is temporarily disabled
                    and the EN bit in ICHERCR can be modified.
                    The ;EN bit enters the write-protect state
                    again
SETB    EA          ;Turn on interrupts (if necessary)
```

Procedure code to disable the CACHE function:

```
CLR     EA          ;Turn off interrupts (required)
MOV     TA,#0AAH    Write Trigger Command Sequence 1
                    No other instructions can be given here.
MOV     TA,#55H     Write Trigger Command Sequence 2
```

```
                                No other instructions can be given here.
    MOV     ICHERCR,#0          The write protection is temporarily disabled
                                and the EN bit in ICHERCR can be modified.
                                The ;EN bit enters the write-protect state
                                again
    SETB    EA                  ;Turn on interrupts (if necessary)
```

The program code that reads the number of CACHE hits:

```
CACHE_ON                        ;first must turn on the
CACHE function MOV              ICHERCR,#0C0H   ;select
read hit count register MOV     R0,ICHERCR      ;read
hit count BIT[31:24] MOV        R1,ICHERCR
        ;BIT[23:16] MOV         R2,ICHERCR      ;Read
hit count of BIT[15:8] MOV      R3,ICHERCR
        ;BIT[7:0] read hit count
                                DR0 is the number of CACHE hits.
```

The program code that reads the number of CACHE misses:

```
CACHE_ON                        ; first the CACHE
function must be turned on MOV  ICHERCR,#080H
;select read miss count register MOV     R0,ICHERCR
        ;read miss count BIT[31:24] MOV
R1,ICHERCR                      ;read miss count
BIT[23:16] MOV                  R2,ICHERCR      ;Read
miss count of BIT[15:8] MOV     R3,ICHERCR;     Read the
number of misses BIT[7:0] MOV
                                DR0 is the number of CACHE misses.
```

CACHE How hit rate is calculated: hits/(hits + misses)*100%

Procedure code to clear CACHE statistics:

```
CACHE_ON                        ; CACHE must be turned on first!
MOV         ICHERCR,#0A0H
```

9.2.11 Bit Addressable Data Memory in STC32G Series Microcontrollers

The bit-addressable data memory inside the STC32G series microcontrollers consists of two parts: the first part has the address range of the DATA area of the 20H~7FH, and the address range of the second part is Special Function Register SFR: 80H~FFH, and these two parts of the area are explained separately below.

DATA area

00H~1FH of DATA area is the mapping area of registers R0~R7, which is not bit-addressable, and there are 96 bytes from 20H~7FH, each byte is bit-addressable.

Method of defining assembly code:

```
BVAR1    BIT    20H.0    ;Define bit variable BVAR1
BVAR2    BIT    50H.1    ;Define the bit variable BVAR2
BVAR3    BIT    70H.2    ;Define bit variable BVAR3
```

The use of assembly code:

```
SETB     BVAR1            ;BVAR1 = 1
CLR      BVAR2            ;BVAR2 = 0
CPL      BVAR3            ;BVAR3 = ~BVAR3
```

C code definition methods:

```
char ebdata      ebdata; flag.                    // Define a byte variable in the bit-addressable area.
sbit  bVar1  =   flag^0; //use sbit to declare bit variable bVar1 in flag   // use sbit to declare the bit variable bVar1 in flag
sbit  bVar2  =   flag^7.                          // use sbit to declare the bit variable bVar2 in flag
bit   ebdata     bVar3.                           // Use bit ebdata to directly declare
```

the bit variable bVar3 C code usage:

```
bVar1 = 1; //bit variable set to 1    //bit variable set to 1
bVar2 = 0;                            // Bit variable clear 0
bVar3 = ~bVar3                        //Bit variable inversion
```

Special Function Register (SFR) Area

All SFR area 80H ~ FFH, a total of 128 bytes, each byte can be bit-addressable. Assembly code definition method:

```
BVAR1    BIT    80H.0    ;Define bit variable BVAR1
BVAR2    BIT    87H.1    ;Define bit variable BVAR2
BVAR3    BIT    FFH.2    Defines the bit variable BVAR3.
```

The use of assembly code:

```
SETB     BVAR1            ;BVAR1 = 1
CLR      BVAR2            ;BVAR2 = 0
CPL      BVAR3            ;BVAR3 = ~BVAR3
```

C code definition methods:

```
sfr   P0     =sfr   0x80;          //define SFR
sbit  P00    sbit P00; =           P0^0;     // use sbit to declare the SFR bit in the SFR
sfr   PCON   =sfr   0x87.          //define SFR
```

```
sbit    PD      =       PCON^1.         //use sbit to declare the SFR bit in the SFR
sfr     ACC     =sfr    0xE0;           //define SFR
sbit    ACC7    =       ACC^7.          // use sbit to declare the SFR bit in the SFR
```

C code usage:

```
PD = 0; //bit variable set to 1       //Bit variable set to 1
```

```
P00 = 1; //Bit variable clear 0                    //Bit variable clear 0
ACC7 = ~ACC7                                       //Bit variable inversion
```

Note: Bit variables do not support array and pointer type definitions

9.2.12 Extended SFR Enable Register EAXFR Usage Description

The STC32G series microcontroller memory addresses are addressed as follows:

shore	address range	functionality	instructions
data area	00:0000H - 00:FFFFH	Data area (edata)	stacks
	01:0000h - 01:ffffh	Internal extended data area (xdata)	
	02:0000h - 7d:ffffh	data retention area	
	7e:0000h - 7e:ffffh	**Extended SFR zone (XFR)**	**EAXFR (P_SW2.7) needs to be enabled to access the**
	7f:0000h - 7f:ffffh	External data area (xdata)	EXTRAM (AUXR.1) needs to be enabled to access it.
code area	80:0000h - fd:ffffh	code reservation (computing)	
	FE:0000H - FE:FFFFH	Extended code area (ecode)	
	ff:0000h - ff:ffffh	Code area (code)	

If you need to access the extended SFR of XFR area, you need to set EAXFR (P_SW2.7) to 1 first. Since the memory address of STC32G is addressed linearly and the address of XFR area is an independent address area, you can **write 1 to the EAXFR register during the initialisation of the system (e.g., EAXFR = 1;), and keep it at 1 afterwards.** The address of XFR area is an independent address area.

Note: Since the EAXFR control bits and the S2_S, S3_S and other bit register bits share P_SW2, it is recommended to use the bit operation statement (e.g., S2_S = 0; S3_S = 1;) directly when setting S2_S, S3_S in P_SW2, and avoid to manipulate the P_SW2 registers directly. If the code does need to modify P_SW2 directly, it must also use the and or instruction (e.g. P_SW2 &= ~0x01; P_SW2 |= 0x02;).

notation	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 disabled

1: Enables access to the XFR.

When you need to access the XFR, you must set EAXFR to 1 before you can read or write to the XFR normally. It is recommended to set EAXFR to 1 during initialisation and not to change it later.

9.3 Unique ID numbers and important parameters stored in read-only special function registers. ID numbers and important parameters stored in the special function registers (CHIPID)

product lines	CHIPID
STC32G12K128 Series	●
STC32G8K64 Series	●
STC32F12K60 Series	●

The read-only special function register CHIPID of STC32G series microcontrollers stores some special parameters related to the chip, including: the global unique ID number, the frequency of the 32K power-down wake-up timer, the value of the internal 1.19V reference signal source, and the IRC parameters. The contents of the CHIPID can only be read in the user programme and cannot be modified. Using the data in the CHIPID to encrypt the user program is the optimal solution recommended by STC.

Related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID0	Hardware numbers ID00	7EFDE0H	Globally unique ID number (byte 0)								nnnn,nnnn
CHIPID1	Hardware number ID01	7EFDE1H	Globally unique ID number (1st byte)								nnnn,nnnn
CHIPID2	Hardware Digital ID02	7EFDE2H	Globally unique ID number (byte 2)								nnnn,nnnn
CHIPID3	Hardware Digital ID03	7EFDE3H	Globally unique ID number (byte 3)								nnnn,nnnn
CHIPID4	Hardware Digital ID04	7EFDE4H	Globally unique ID number (byte 4)								nnnn,nnnn
CHIPID5	Hardware Digital ID05	7EFDE5H	Globally unique ID number (byte 5)								nnnn,nnnn
CHIPID6	Hardware Digital ID06	7EFDE6H	Globally unique ID number (byte 6)								nnnn,nnnn
CHIPID7	Hardware Digital ID07	7EFDE7H	Internal 1.19V reference signal source (high byte)								nnnn,nnnn
CHIPID8	Hardware Digital ID08	7EFDE8H	Internal 1.19V reference signal source (low byte)								nnnn,nnnn
CHIPID9	Hardware Digital ID09	7EFDE9H	32K Power-down wake-up timer frequency (high byte)								nnnn,nnnn
CHIPID10	Hardware Digital ID10	7EFDEAH	32K Power-down wake-up timer frequency (low byte)								nnnn,nnnn
CHIPID11	Hardware Digital ID11	7EFDEBH	IRC parameters for 22.1184MHz (27M band)								nnnn,nnnn
CHIPID12	Hardware Digital ID12	7EFDECH	IRC parameters for 24MHz (27M band)								nnnn,nnnn
CHIPID13	Hardware number ID13	7EFDEDH	IRC parameters for 27MHz (27M band)								nnnn,nnnn
CHIPID14	Hardware number ID14	7EFDEEH	IRC parameters for 30MHz (27M band)								nnnn,nnnn
CHIPID15	Hardware Digital ID15	7EFDEFH	IRC parameters for 33.1776MHz (27M band)								nnnn,nnnn
CHIPID16	Hardware Digital ID16	7EFDF0H	IRC parameters for 35MHz (44M band)								nnnn,nnnn
CHIPID17	Hardware Digital ID17	7EFDF1H	IRC parameters for 36.864MHz (44M band)								nnnn,nnnn
CHIPID18	Hardware Digital ID18	7EFDF2H	IRC parameters for 40MHz (44M band)								nnnn,nnnn
CHIPID19	Hardware number ID19	7EFDF3H	IRC parameters for 44.2368MHz (44M band)								nnnn,nnnn
CHIPID20	Hardware Digital ID20	7EFDF4H	IRC parameters for 48MHz (44M band)								nnnn,nnnn
CHIPID21	Hardware numbers ID21	7EFDF5H	VRTRIM parameters for the 6M band								nnnn,nnnn

CHIPID25	Hardware numbers ID25	7EFDF9H	00H	nnnn,nnnn
CHIPID26	Hardware numbers ID26	7EFDFAH	End-of-user-program space address (high byte)	nnnn,nnnn
CHIPID27	Hardware numbers ID27	7EFDFBH	Chip testing time (years)	nnnn,nnnn
CHIPID28	Hardware numbers ID28	7EFDFCH	Chip testing time (months)	nnnn,nnnn
CHIPID29	Hardware Digital ID29	7EFDFDH	Chip test time (days)	nnnn,nnnn
CHIPID30	Hardware Digital ID30	7EFDFEH	Chip package form number	nnnn,nnnn
CHIPID31	Hardware numbers ID31	7EFDFFH	5AH	nnnn,nnnn

9.3.1 CHIP's Globally Unique ID Number Interpretation

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID0	Hardware numbers ID00	7EFDE0H	Globally unique ID number (byte 0)								nnnn,nnnn
CHIPID1	Hardware number ID01	7EFDE1H	Globally unique ID number (1st byte)								nnnn,nnnn
CHIPID2	Hardware Digital ID02	7EFDE2H	Globally unique ID number (byte 2)								nnnn,nnnn
CHIPID3	Hardware Digital ID03	7EFDE3H	Globally unique ID number (byte 3)								nnnn,nnnn
CHIPID4	Hardware Digital ID04	7EFDE4H	Globally unique ID number (byte 4)								nnnn,nnnn
CHIPID5	Hardware Digital ID05	7EFDE5H	Globally unique ID number (byte 5)								nnnn,nnnn
CHIPID6	Hardware Digital ID06	7EFDE6H	Globally unique ID number (byte 6)								nnnn,nnnn

[CHIPID0, CHIPID1]: 16-bit MCU ID for distinguishing different MCU models (high bit comes first). [CHIPID2, CHIPID3]: 16-bit test bench number (high bit comes first).

[CHIPID4, CHIPID5, CHIPID6]: 24-bit test flow number (high bit first).

9.3.2 CHIP's Internal Reference Signal Source Interpretation

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID7	Hardware Digital ID07	7EFDE7H	Internal 1.19V reference signal source (high byte)								nnnn,nnnn
CHIPID8	Hardware Digital ID08	7EFDE8H	Internal 1.19V reference signal source (low byte)								nnnn,nnnn

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

The standard value is 1190 (04A6H) in mV, i.e. 1.19 V. However, the actual chip is subject to manufacturing errors. The voltage value of the internal reference signal source is not affected by the operating voltage VCC, so the internal reference signal source can be used in combination with an ADC to calibrate the ADC or with a comparator to detect the operating voltage.

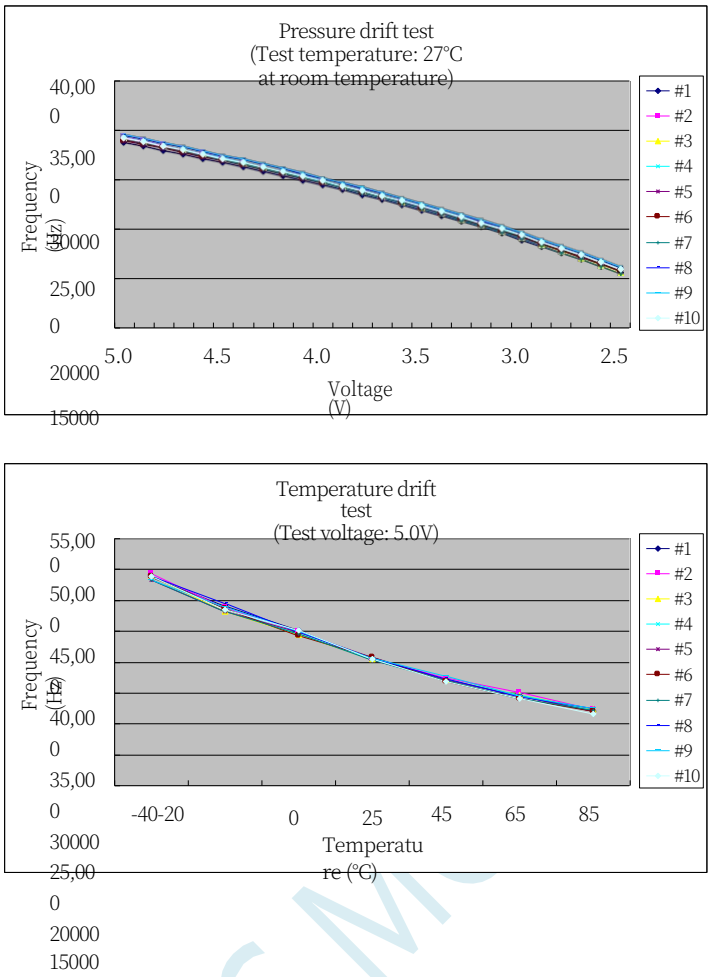
9.3.3 CHIP's Internal 32K IRC Oscillation Frequency Interpretation

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID9	Hardware Digital ID09	7EFDE9H	32K Power-down wake-up timer frequency (high byte)								nnnn,nnnn
CHIPID10	Hardware Digital ID10	7EFDEAH	32K Power-down wake-up timer frequency (low byte)								nnnn,nnnn

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

The standard value is 32768 (8000H) in Hz, i.e. 32.768 KHz, but the actual chip is subject to manufacturing errors and has large temperature and voltage drift.

The internal 32K oscillator voltage drift test linearity graph and temperature drift linearity graph are shown below:


Pressure drift test
(Test temperature: 27℃
at room temperature)
Frequency (Hz)
40,000
35,000
30000
25,000
20000
15000
5.0
4.5
4.0
3.5
3.0
2.5
Voltage (V)
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10

Temperature drift test
(Test voltage: 5.0V)
Frequency (Hz)
55,000
50,000
45,000
40,000
35,000
30000
25,000
20000
15000
-40-20
0
25
45
65
85
Temperature (℃)
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10

9.3.4 CHIP's High Precision IRC Parameter Interpretation

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID11	Hardware Digital ID11	7EFDEBH	IRC parameters for 22.1184MHz (27M band)								nnnn,nnnn
CHIPID12	Hardware Digital ID12	7EFDECH	IRC parameters for 24MHz (27M band)								nnnn,nnnn
CHIPID13	Hardware number ID13	7EFDEDH	IRC parameters for 27MHz (27M band)								nnnn,nnnn
CHIPID14	Hardware number ID14	7EFDEEH	IRC parameters for 30MHz (27M band)								nnnn,nnnn
CHIPID15	Hardware Digital ID15	7EFDEFH	IRC parameters for 33.1776MHz (27M band)								nnnn,nnnn
CHIPID16	Hardware Digital ID16	7EFDF0H	IRC parameters for 35MHz (44M band)								nnnn,nnnn
CHIPID17	Hardware Digital ID17	7EFDF1H	IRC parameters for 36.864MHz (44M band)								nnnn,nnnn
CHIPID18	Hardware Digital ID18	7EFDF2H	IRC parameters for 40MHz (44M band)								nnnn,nnnn
CHIPID19	Hardware number ID19	7EFDF3H	IRC parameters for 44.2368MHz (44M band)								nnnn,nnnn
CHIPID20	Hardware Digital ID20	7EFDF4H	IRC parameters for 48MHz (44M band)								nnnn,nnnn
CHIPID21	Hardware numbers ID21	7EFDF5H	VRTRIM parameters for the 6M band								nnnn,nnnn
CHIPID22	Hardware numbers ID22	7EFDF6H	VRTRIM parameters for the 10M band								nnnn,nnnn
CHIPID23	Hardware numbers ID23	7EFDF7H	VRTRIM parameters for the 27M band								nnnn,nnnn
CHIPID24	Hardware Digital ID24	7EFDF8H	VRTRIM parameters for the 44M band								nnnn,nnnn

STC32G series microcontrollers supporting CHIPID function, the internal integrated high-precision IRC is divided into 4 frequency bands, the reference voltage value corresponding to each band has been calibrated in the factory, when selecting different frequency bands, you only need to fill the voltage calibration value of the corresponding band into the VRTRIM register. 4 frequency bands have the centre frequency of 6MHz, 10MHz, 27MHz and 44MHz, respectively. The centre frequencies of the four frequency bands are 6MHz, 10MHz, 27MHz and 44MHz respectively. Due to manufacturing errors, the centre frequencies may deviate by ±5%, so in order to get an accurate user frequency, the frequencies can be fine-tuned and calibrated using IRTRIM. In order to get the accurate user frequency, IRTRIM can be used to fine-tune the frequency calibration. When you download the user programme using the official STC download software, the system will automatically set the VRTRIM and IRTRIM registers according to the user's set frequency. Meanwhile, CHIPID is also preset with IRTRIM values for 10 commonly used frequencies and reference voltage calibration values for 4 frequency bands, allowing users to dynamically modify the operating frequency during the running of the programme.

[CHIPID11 : CHIPID20]: IRTRIM values for 10 common frequencies. The notes in parentheses are the corresponding frequency bands.

[CHIPID21 : CHIPID24] : Reference voltage value calibration value for 4 frequency bands.

To modify the frequency dynamically, the user only needs to read out a frequency calibration value from [CHIPID11 : CHIPID20] and write it to the IRTRIM register, and at the same time read out a voltage calibration value from [CHIPID21 : CHIPID24] and write it to the VRTRIM register according to the frequency band corresponding to the frequency. For detailed operation, please refer to the sample programs in the following sections.

9.3.5 CHIP's Test Time Parameter Interpretation

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID27	Hardware numbers ID27	7EFDFBH	Chip testing time (years)								nnnn,nnnn
CHIPID28	Hardware numbers ID28	7EFDFCH	Chip testing time (months)								nnnn,nnnn
CHIPID29	Hardware Digital ID29	7EFDFDH	Chip test time (days)								nnnn,nnnn

The year, month and day parameters of the test time are BCD codes.(e.g. CHIPID27 = 0x21, CHIPID28 = 0x11, CHIPID29 = 0x18, the production test date of the target chip is 18 November 2021)

9.3.6 CHIP's Chip Package Form Number Interpretation

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID30	Hardware Digital ID30	7EFDFEH	Chip package form number								nnnn,nnnn

Package Number	Package form		Package Number	Package form
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

9.4 sample procedure

9.4.1 Read internal 1.19V reference signal source value

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - (FOSC / 115200+2) / 4)

//Define as unsigned long integer to avoid overflow.

The // plus 2 operation is designed to make the Keil compiler
//Automatic implementation of rounding operations

#define     VREFH_ADDR
CHIPID7 #define
VREFL_ADDR
CHIPID8

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;
    T1x12 = 1.
    busy = 0;
}

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

void main()
{
```

```
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
```

```
    WTST = 0x00.                        //Set the parameter for waiting for the programme code.
                                        //Assign 0 to set the CPU to execute the programme as fast as possible.

    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;
    UartSend(VREF_ADDRH).               //read the high byte of the internal 1.19V reference signal source

    UartSend(VREF_ADDRL).               //read the low byte of the internal 1.19V reference signal source

    while (1);
}
```

9.4.2 Read globally unique ID numbers

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL  //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)
                                        //The // plus 2 operation is designed to make the Keil compiler
                                        //Automatic implementation of rounding operations

#define     ID_ADDR

(&CHIPID0) 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;
    T1x12 = 1.
    busy = 0;
}

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

void main()
{
    char i;

    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign 0 to set the CPU to execute the programme as fast as
                                possible.


    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;

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

    while (1);
}
```


9.4.3 Reads the frequency of the 32K power-down wake-up timer

//Tested operating frequency is *11.0592MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                                  // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL               //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)
                                                     The // plus 2 operation is designed to make the Keil compiler
                                                     //Automatic implementation of rounding operations

#define     F32K_ADDRH      CHIPID9
#define     F32K_ADDRL
CHIPID10

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;
    T1x12 = 1.
    busy = 0;
}

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

void main()
{
    EAXFR = 1;                                       //Enable access to XFR
    CKCON = 0x00;                                    //Set the external data bus speed to fastest
    WTST = 0x00;                                     //set the program code wait parameter.
                                                     //Assign 0 to set the CPU to execute the programme as fast as
                                                     possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
```

```
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

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

    UartSend(F32K_ADDRH).                    //read the 32K frequency high byte
    UartSend(F32K_ADDRL).                    //read the low byte of 32K frequency

    while (1);
}
```

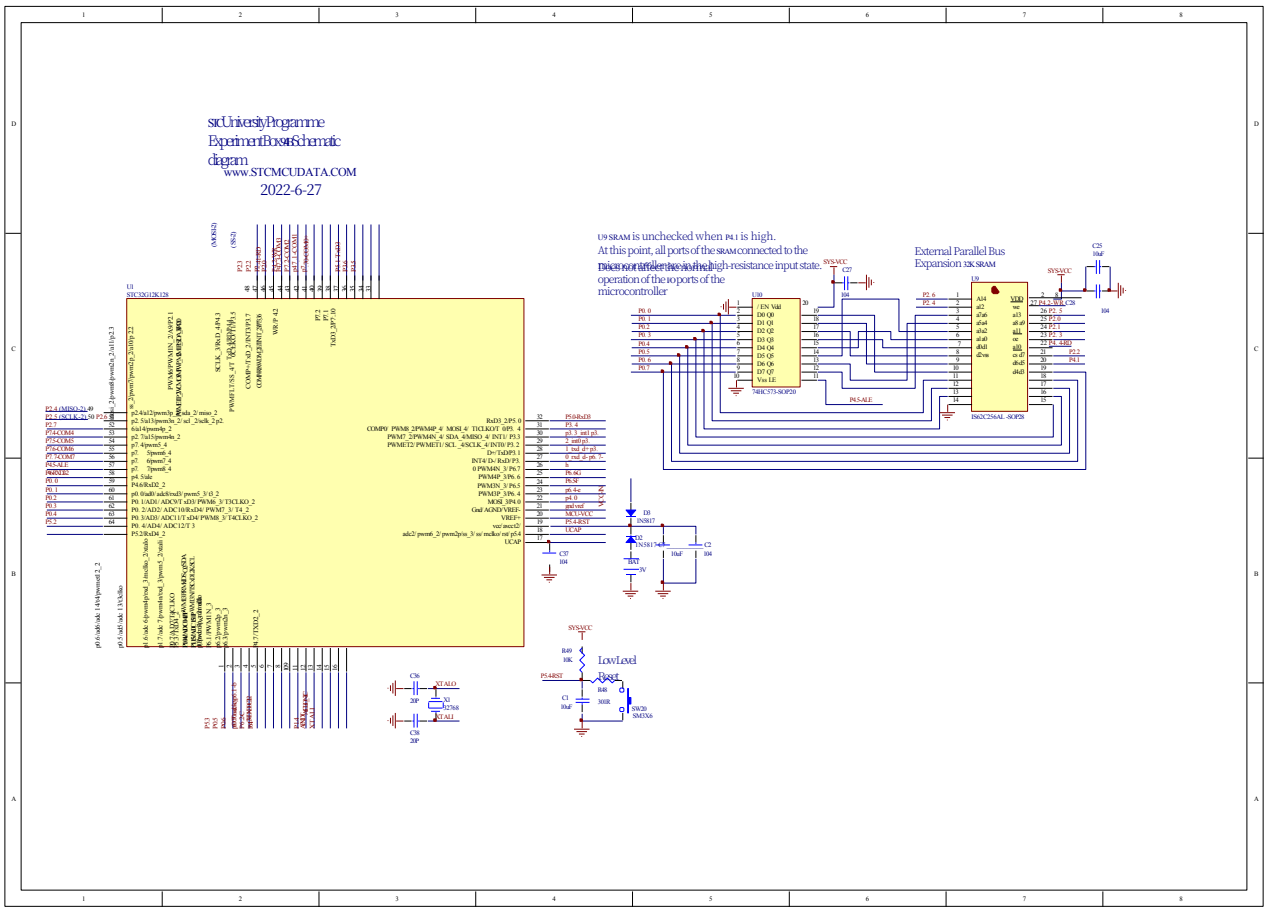
9.4.4 User-defined internal IRC frequency

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

#define   T22M_ADDR     CHIPID11     //22.1184MHz
#define   T24M_ADDR     CHIPID12     //24MHz
#define   T27M_ADDR     CHIPID13     //27MHz
#define   T30M_ADDR     CHIPID14     //30MHz
#define   T33M_ADDR     CHIPID15     //33.1776MHz
#define   T35M_ADDR     CHIPID16     //35MHz
#define   T36M_ADDR     CHIPID17     //36.864MHz
#define   T40M_ADDR     CHIPID18     //40MHz
#define   T44M_ADDR     CHIPID19     //44.2368MHz
#define   T48M_ADDR     CHIPID20     //48MHz
#define   VRT6M_ADDR    CHIPID21     //VRTRIM_6M
#define   VRT10M_ADDR   CHIPID22     //VRTRIM_10M
#define   VRT27M_ADDR   CHIPID23     //VRTRIM_27M
#define   VRT44M_ADDR   CHIPID24     //VRTRIM_44M

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign 0 to set the CPU to execute the programme as fast as
                                             possible.

    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 22.1184MHz
    //CLKDIV = 0x04.
    //IRTRIM = T22M_ADDR.
    //VRTRIM = VRT27M_ADDR.
    //IRCBAND = 0x02.
    //CLKDIV = 0x00.

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

// // Select 27MHz
    //CLKDIV = 0x04.
    //IRTRIM = T27M_ADDR.
    //VRTRIM = VRT27M_ADDR.
    //IRCBAND = 0x02.
    //CLKDIV = 0x00.

// // Select 30MHz
    //CLKDIV = 0x04.
    //IRTRIM = T30M_ADDR.
    //VRTRIM = VRT27M_ADDR.
    //IRCBAND = 0x02.
    //CLKDIV = 0x00.

// // Select 33.1776MHz
    //CLKDIV = 0x04.
    //IRTRIM = T33M_ADDR.
    //VRTRIM = VRT27M_ADDR.
    //IRCBAND = 0x02.
    //CLKDIV = 0x00.

// // Select 35MHz
    //CLKDIV = 0x04.
    //IRTRIM = T35M_ADDR.
    //VRTRIM = VRT44M_ADDR.
    //IRCBAND = 0x03.
    //CLKDIV = 0x00.

// // Select 44.2368MHz
    //CLKDIV = 0x04.
    //IRTRIM = T44M_ADDR.
    //VRTRIM = VRT44M_ADDR.
    //IRCBAND = 0x03.
    //CLKDIV = 0x00.

// // Select 48MHz
    //CLKDIV = 0x04.
    //IRTRIM = T48M_ADDR.
    //VRTRIM = VRT44M_ADDR.
    //IRCBAND = 0x03.
```

```
    //CLKDIV = 0x00.

    while (1);
}
```

9.4.5 Read and write off-chip extended RAM




```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

#define     EXRAMB          ((unsigned char volatile far
*)0x7f0000) #define          EXRAMW((unsigned      int volatile
far *)0x7f0000) #define      EXRAMD       ((unsigned long
volatile far *)0x7f0000)

void main()
{
    char x8;
    int x16;
    long x32.

    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
```

```
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    EXTRAM = 1;                     //Enable access to off-chip
    BUS_SPEED = 2;                  //Set the external bus access speed

    x8 = EXRAMB[0x0100];            //read 1 byte of data from 0x0100 address of external
    expansion RAM to variable x8 x16 = EXRAMW[0x0200];  //read 2 bytes of data from 0x0400 address
    of external expansion RAM to variable x16 x32 = EXRAMD[0x0300]; //read 4 bytes of data from 0x0C00
    address of external expansion RAM to variable x32 x32 = EXRAMD[0x0300].  // Read 4 bytes of data
    from 0x0C00 address of external expansion RAM to variable x32.
                                    //Note: The multibyte data format in Keil uses the BE (big-endian) format.
                                    //i.e., the high byte is stored at the lower address and the low byte is stored at
                                    the higher address

    EXRAMB[0x0101] = x8;            //write the data of variable x8 to address
    0x0101 of external expansion RAM EXRAMB[0x0205] = x16;  //write the data of
    variable x16 to address 0x040A of external expansion RAM EXRAMB[0x030c] = x32. //write
    the data of variable x32 to address 0x0C30 of the external expansion RAM.

    while (1);
}
```

10 Special Function Registers (SFR, XFR)

10.1 STC32G12K128 Series

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H	P7	LINICR	LINAR	LINDR	USBADR	S4CON	S4BUF	RSTCFG
F0H	B	CANICR			USBCON	IAP_TPS	IAP_ADDRE	ICHECR
E8H	P6	WTST	CKCON	MXAX	USBDAT	DMAIR	IP3H	AUXINTIF
E0H	ACC	P7M1	P7M0	DPS			CMPCR1	CMPCR2
D8H					USBCLK	T4T3M	ADCCFG	IP3
D0H	PSW	PSW1	TH4	TL4	TH3	TL3	TH2	TL2
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	P_SW3	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	AUXR2
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	DPXL	SPH		PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
7EFEF8	PWMB_CCR6L	PWMB_CCR7H	PWMB_CCR7L	PWMB_CCR8H	PWMB_CCR8L	PWMB_BKR	PWMB_DTR	PWMB_OISR
7EFEF0	PWMB_PSCRH	PWMB_PSCRL	PWMB_ARRH	PWMB_ARRL	PWMB_RCR	PWMB_CCR5H	PWMB_CCR5L	PWMB_CCR6H
7EFEE8	PWMB_CCMR1	PWMB_CCMR2	PWMB_CCMR3	PWMB_CCMR4	PWMB_CCER1	PWMB_CCER2	PWMB_CNTRH	PWMB_CNTRL
7EFEE0	PWMB_CR1	PWMB_CR2	PWMB_SMCR	PWMB_ETR	PWMB_IER	PWMB_SR1	PWMB_SR2	PWMB_EGR
7EFED8	PWMA_CCR2L	PWMA_CCR3H	PWMA_CCR3L	PWMA_CCR4H	PWMA_CCR4L	PWMA_BKR	PWMA_DTR	PWMA_OISR
7EFED0	PWMA_PSCRH	PWMA_PSCRL	PWMA_ARRH	PWMA_ARRL	PWMA_RCR	PWMA_CCR1H	PWMA_CCR1L	PWMA_CCR2H
7EFEC8	PWMA_CCMR1	PWMA_CCMR2	PWMA_CCMR3	PWMA_CCMR4	PWMA_CCER1	PWMA_CCER2	PWMA_CNTRH	PWMA_CNTRL
7EFEC0	PWMA_CR1	PWMA_CR2	PWMA_SMCR	PWMA_ETR	PWMA_IER	PWMA_SR1	PWMA_SR2	PWMA_EGR
7EFEB8				CANAR	CANDR			
7EFEB0	PWMA_ETRPS	PWMA_ENO	PWMA_PS	PWMA_IOAUX	PWMB_ETRPS	PWMB_ENO	PWMB_PS	PWMB_IOAUX
7EFEA8	ADCTIM				T3T4PIN	ADCEXCFG	CMPEXCFG	
7EFEA0	TM0PS	TM1PS	TM2PS	TM3PS	TM4PS			
7EFE98	SPFUNC	RSTFLAG	RSTCR0	RSTCR1	RSTCR2	RSTCR3	RSTCR4	RSTCR5
7EFE88	I2CMSAUX							
7EFE80	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
7EFE70	YEAR	MONTH	DAY	HOUR	MIN	SEC	SSEC	
7EFE68	INIYEAR	INIMONTH	INIDAY	INIHOUR	INIMIN	INISEC	INISSEC	
7EFE60	RTCCR	RTCCFG	RTCIEN	RTCIF	ALAHOUR	ALAMIN	ALASEC	ALASSEC
7EFE50	LCMIFCFG	LCMIFCFG2	LCMIFCR	LCMIFSTA	LCMIFDATL	LCMDATH		

7EFE30	P0IE	P1IE	P2IE	P3IE	P4IE	P5IE	P6IE	P7IE
7EFE28	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR	P6DR	P7DR
7EFE20	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR	P6SR	P7SR
7EFE18	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
7EFE10	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
7EFE08	X32KCR			HSCLKDIV				
7EFE00	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	IRC48MCR
7EFDF8	CHIPID[24]	CHIPID[25]	CHIPID[26]	CHIPID[27]	CHIPID[28]	CHIPID[29]	CHIPID[30]	CHIPID[31]
7EFDF0	CHIPID[16]	CHIPID[17]	CHIPID[18]	CHIPID[19]	CHIPID[20]	CHIPID[21]	CHIPID[22]	CHIPID[23]
7EFDE8	CHIPID[8]	CHIPID[9]	CHIPID[10]	CHIPID[11]	CHIPID[12]	CHIPID[13]	CHIPID[14]	CHIPID[15]
7EFDE0	CHIPID[0]	CHIPID[1]	CHIPID[2]	CHIPID[3]	CHIPID[4]	CHIPID[5]	CHIPID[6]	CHIPID[7]
7EFDC8	USART2CR1	USART2CR2	USART2CR3	USART2CR4	USART2CR5	USART2GTR	USART2BRH	USART2BRL
7EFDC0	USARTCR1	USARTCR2	USARTCR3	USARTCR4	USARTCR5	USARTGTR	USARTBRH	USARTBRL
7EFDB0					S2CFG	S2ADDR	S2ADEN	
7EFD60	PINIPL	PINIPH						
7EFD40	P0WKUE	P1WKUE	P2WKUE	P3WKUE	P4WKUE	P5WKUE	P6WKUE	P7WKUE
7EFD30	P0IM1	P1IM1	P2IM1	P3IM1	P4IM1	P5IM1	P6IM1	P7IM1
7EFD20	P0IM0	P1IM0	P2IM0	P3IM0	P4IM0	P5IM0	P6IM0	P7IM0
7EFD10	P0INTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF	P6INTF	P7INTF
7EFD00	P0INTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE	P6INTE	P7INTE
7EFBF8	HSSPI_CFG	HSSPI_CFG2	HSSPI_STA					
7EFBF0	HSPWMA_CFG	HSPWMA_ADR	HSPWMA_DAT		HSPWMB_CFG	HSPWMB_ADR	HSPWMB_DAT	
7EFAF8	DMA_ARBCFG	DMA_ARBSTA						
7EFAA8	DMA_I2CT_AMTH	DMA_I2CT_DONEH	DMA_I2CR_AMTH	DMA_I2CR_DONEH		DMA_I2C_CR	DMA_I2C_ST1	DMA_I2C_ST2
7EFAA0	DMA_I2CR_CFG	DMA_I2CR_CR	DMA_I2CR_STA	DMA_I2CR_AMT	DMA_I2CR_DONE	DMA_I2CR_RXAH	DMA_I2CR_RXAL	
7EFA98	DMA_I2CT_CFG	DMA_I2CT_CR	DMA_I2CT_STA	DMA_I2CT_AMT	DMA_I2CT_DONE	DMA_I2CT_TXAH	DMA_I2CT_TXAL	
7EFA90	DMA_UR3T_AMTH	DMA_UR3T_DONEH	DMA_UR3R_AMTH	dma_ur3r_doneh	DMA_UR4T_AMTH	DMA_UR4T_DONEH	DMA_UR4R_AMTH	DMA_UR4R_DONEH
7EFA88	DMA_UR1T_AMTH	DMA_UR1T_DONEH	DMA_UR1R_AMTH	DMA_UR1R_DONEH	DMA_UR2T_AMTH	DMA_UR2T_DONEH	DMA_UR2R_AMTH	dma_ur2r_doneh
7EFA80	DMA_M2M_AMTH	DMA_M2M_DONEH			DMA_SPI_AMTH	DMA_SPI_DONEH	DMA_LCM_AMTH	DMA_LCM_DONEH
7EFA78	DMA_LCM_RXAL							
7EFA70	DMA_LCM_CFG	DMA_LCM_CR	DMA_LCM_STA	DMA_LCM_AMT	DMA_LCM_DONE	DMA_LCM_TXAH	DMA_LCM_TXAL	DMA_LCM_RXAH
7EFA68	DMA_UR4R_CFG	DMA_UR4R_CR	DMA_UR4R_STA	DMA_UR4R_AMT	DMA_UR4R_DONE	DMA_UR4R_RXAH	DMA_UR4R_RXAL	
7EFA60	DMA_UR4T_CFG	DMA_UR4T_CR	DMA_UR4T_STA	DMA_UR4T_AMT	DMA_UR4T_DONE	DMA_UR4T_TXAH	DMA_UR4T_TXAL	
7EFA58	DMA_UR3R_CFG	DMA_UR3R_CR	DMA_UR3R_STA	DMA_UR3R_AMT	DMA_UR3R_DONE	DMA_UR3R_RXAH	dma_ur3r_rxal	
7EFA50	DMA_UR3T_CFG	DMA_UR3T_CR	DMA_UR3T_STA	DMA_UR3T_AMT	DMA_UR3T_DONE	DMA_UR3T_TXAH	DMA_UR3T_TXAL	
7EFA48	DMA_UR2R_CFG	DMA_UR2R_CR	DMA_UR2R_STA	DMA_UR2R_AMT	DMA_UR2R_DONE	DMA_UR2R_RXAH	DMA_UR2R_RXAL	
7EFA40	DMA_UR2T_CFG	DMA_UR2T_CR	DMA_UR2T_STA	DMA_UR2T_AMT	DMA_UR2T_DONE	DMA_UR2T_TXAH	DMA_UR2T_TXAL	
7EFA38	DMA_UR1R_CFG	DMA_UR1R_CR	DMA_UR1R_STA	DMA_UR1R_AMT	DMA_UR1R_DONE	DMA_UR1R_RXAH	DMA_UR1R_RXAL	
7EFA30	DMA_UR1T_CFG	DMA_UR1T_CR	DMA_UR1T_STA	DMA_UR1T_AMT	DMA_UR1T_DONE	DMA_UR1T_TXAH	DMA_UR1T_TXAL	
7EFA28	DMA_SPI_RXAL	DMA_SPI_CFG2						
7EFA20	DMA_SPI_CFG	DMA_SPI_CR	DMA_SPI_STA	DMA_SPI_AMT	DMA_SPI_DONE	DMA_SPI_TXAH	DMA_SPI_TXAL	DMA_SPI_RXAH
7EFA18	DMA_ADC_RXAL	DMA_ADC_CFG2	DMA_ADC_CHSW0	DMA_ADC_CHSW1				
7EFA10	DMA_ADC_CFG	DMA_ADC_CR	DMA_ADC_STA					DMA_ADC_RXAH
7EFA08	DMA_M2M_RXAL							

7EFA00	DMA_M2M_CFG	DMA_M2M_CR	DMA_M2M_STA	DMA_M2M_AMT	DMA_M2M_DONE	DMA_M2M_TXAH	DMA_M2M_TXAL	DMA_M2M_RXAH

10.2 STC32G8K64 Series

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		LINICR	LINAR	LINDR		S4CON	S4BUF	RSTCFG
F0H	B	CANICR				IAP_TPS	IAP_ADDRE	ICHECR
E8H		WTST	CKCON	MXAX		DMAIR	IP3H	AUXINTIF
E0H	ACC			DPS			CMPCR1	CMPCR2
D8H						T4T3M	ADCCFG	IP3
D0H	PSW	PSW1	TH4	TL4	TH3	TL3	TH2	TL2
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	P_SW3	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	AUXR2
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	DPXL	SPH		PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
7EFEF8	PWMB_CCR6L	PWMB_CCR7H	PWMB_CCR7L	PWMB_CCR8H	PWMB_CCR8L	PWMB_BKR	PWMB_DTR	PWMB_OISR
7EFEF0	PWMB_PSCRH	PWMB_PSCRL	PWMB_ARRH	PWMB_ARRL	PWMB_RCR	PWMB_CCR5H	PWMB_CCR5L	PWMB_CCR6H
7EFEE8	PWMB_CCMR1	PWMB_CCMR2	PWMB_CCMR3	PWMB_CCMR4	PWMB_CCER1	PWMB_CCER2	PWMB_CNTRH	PWMB_CNTRL
7EFEE0	PWMB_CR1	PWMB_CR2	PWMB_SMCR	PWMB_ETR	PWMB_IER	PWMB_SR1	PWMB_SR2	PWMB_EGR
7EFED8	PWMA_CCR2L	PWMA_CCR3H	PWMA_CCR3L	PWMA_CCR4H	PWMA_CCR4L	PWMA_BKR	PWMA_DTR	PWMA_OISR
7EFED0	PWMA_PSCRH	PWMA_PSCRL	PWMA_ARRH	PWMA_ARRL	PWMA_RCR	PWMA_CCR1H	PWMA_CCR1L	PWMA_CCR2H
7EFEC8	PWMA_CCMR1	PWMA_CCMR2	PWMA_CCMR3	PWMA_CCMR4	PWMA_CCER1	PWMA_CCER2	PWMA_CNTRH	PWMA_CNTRL
7EFEC0	PWMA_CR1	PWMA_CR2	PWMA_SMCR	PWMA_ETR	PWMA_IER	PWMA_SR1	PWMA_SR2	PWMA_EGR
7EFEB8				CANAR	CANDR			
7EFEB0	PWMA_ETRPS	PWMA_ENO	PWMA_PS	PWMA_IOAUX	PWMB_ETRPS	PWMB_ENO	PWMB_PS	PWMB_IOAUX
7EFEA8	ADCTIM				T3T4PIN	ADCEXCFG	CMPEXCFG	
7EFEA0	TM0PS	TM1PS	TM2PS	TM3PS	TM4PS			
7EFE98	SPFUNC	RSTFLAG	RSTCR0	RSTCR1	RSTCR2	RSTCR3	RSTCR4	RSTCR5
7EFE88	I2CMSAUX							
7EFE80	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
7EFE70	YEAR	MONTH	DAY	HOUR	MIN	SEC	SSEC	
7EFE68	INIYEAR	INIMONTH	INIDAY	INIHOUR	INIMIN	INISEC	INISSEC	
7EFE60	RTCCR	RTCCFG	RTCIEN	RTCIF	ALAHOUR	ALAMIN	ALASEC	ALASSEC
7EFE50	LCMIFCFG	LCMIFCFG2	LCMIFCR	LCMIFSTA	LCMIFDATL	LCMDATH		
7EFE40	P0PD	P1PD	P2PD	P3PD	P4PD	P5PD	P6PD	P7PD
7EFE30	P0IE	P1IE	P2IE	P3IE	P4IE	P5IE	P6IE	P7IE

7EFE28	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR	P6DR	P7DR
7EFE20	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR	P6SR	P7SR
7EFE18	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
7EFE10	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
7EFE08	X32KCR			HSCLKDIV				
7EFE00	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	IRC48MCR
7EFDF8	CHIPID[24]	CHIPID[25]	CHIPID[26]	CHIPID[27]	CHIPID[28]	CHIPID[29]	CHIPID[30]	CHIPID[31]
7EFDF0	CHIPID[16]	CHIPID[17]	CHIPID[18]	CHIPID[19]	CHIPID[20]	CHIPID[21]	CHIPID[22]	CHIPID[23]
7EFDE8	CHIPID[8]	CHIPID[9]	CHIPID[10]	CHIPID[11]	CHIPID[12]	CHIPID[13]	CHIPID[14]	CHIPID[15]
7EFDE0	CHIPID[0]	CHIPID[1]	CHIPID[2]	CHIPID[3]	CHIPID[4]	CHIPID[5]	CHIPID[6]	CHIPID[7]
7EFDC8	USART2CR1	USART2CR2	USART2CR3	USART2CR4	USART2CR5	USART2GTR	USART2BRH	USART2BRL
7EFDC0	USARTCR1	USARTCR2	USARTCR3	USARTCR4	USARTCR5	USARTGTR	USARTBRH	USARTBRL
7EFDB0					S2CFG	S2ADDR	S2ADEN	
7EFDA8	CRECR	CRECNTH	CRECNTL	CRERES				
7EFD60	PINIPL	PINIPH						
7EFD40	P0WKUE	P1WKUE	P2WKUE	P3WKUE	P4WKUE	P5WKUE	P6WKUE	P7WKUE
7EFD30	P0IM1	P1IM1	P2IM1	P3IM1	P4IM1	P5IM1	P6IM1	P7IM1
7EFD20	P0IM0	P1IM0	P2IM0	P3IM0	P4IM0	P5IM0	P6IM0	P7IM0
7EFD10	P0INTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF	P6INTF	P7INTF
7EFD00	P0INTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE	P6INTE	P7INTE
7EFBF8	HSSPI_CFG	HSSPI_CFG2	HSSPI_STA					
7EFBF0	HSPWMA_CFG	HSPWMA_ADR	HSPWMA_DAT		HSPWMB_CFG	HSPWMB_ADR	HSPWMB_DAT	
7EFAF8	DMA_ARBCFG	DMA_ARBSTA						
7EFAA8	DMA_I2CT_AMTH	DMA_I2CT_DONEH	DMA_I2CR_AMTH	DMA_I2CR_DONEH		DMA_I2C_CR	DMA_I2C_ST1	DMA_I2C_ST2
7EFAA0	DMA_I2CR_CFG	DMA_I2CR_CR	DMA_I2CR_STA	DMA_I2CR_AMT	DMA_I2CR_DONE	DMA_I2CR_RXAH	DMA_I2CR_RXAL	
7EFA98	DMA_I2CT_CFG	DMA_I2CT_CR	DMA_I2CT_STA	DMA_I2CT_AMT	DMA_I2CT_DONE	DMA_I2CT_TXAH	DMA_I2CT_TXAL	
7EFA90	DMA_UR3T_AMTH	DMA_UR3T_DONEH	DMA_UR3R_AMTH	dma_ur3r_doneh	DMA_UR4T_AMTH	DMA_UR4T_DONEH	DMA_UR4R_AMTH	DMA_UR4R_DONEH
7EFA88	DMA_UR1T_AMTH	DMA_UR1T_DONEH	DMA_UR1R_AMTH	DMA_UR1R_DONEH	DMA_UR2T_AMTH	DMA_UR2T_DONEH	DMA_UR2R_AMTH	dma_ur2r_doneh
7EFA80	DMA_M2M_AMTH	DMA_M2M_DONEH			DMA_SPI_AMTH	DMA_SPI_DONEH	DMA_LCM_AMTH	DMA_LCM_DONEH
7EFA78	DMA_LCM_RXAL							
7EFA70	DMA_LCM_CFG	DMA_LCM_CR	DMA_LCM_STA	DMA_LCM_AMT	DMA_LCM_DONE	DMA_LCM_TXAH	DMA_LCM_TXAL	DMA_LCM_RXAH
7EFA68	DMA_UR4R_CFG	DMA_UR4R_CR	DMA_UR4R_STA	DMA_UR4R_AMT	DMA_UR4R_DONE	DMA_UR4R_RXAH	DMA_UR4R_RXAL	
7EFA60	DMA_UR4T_CFG	DMA_UR4T_CR	DMA_UR4T_STA	DMA_UR4T_AMT	DMA_UR4T_DONE	DMA_UR4T_TXAH	DMA_UR4T_TXAL	
7EFA58	DMA_UR3R_CFG	DMA_UR3R_CR	DMA_UR3R_STA	DMA_UR3R_AMT	DMA_UR3R_DONE	DMA_UR3R_RXAH	dma_ur3r_rxal	
7EFA50	DMA_UR3T_CFG	DMA_UR3T_CR	DMA_UR3T_STA	DMA_UR3T_AMT	DMA_UR3T_DONE	DMA_UR3T_TXAH	DMA_UR3T_TXAL	
7EFA48	DMA_UR2R_CFG	DMA_UR2R_CR	DMA_UR2R_STA	DMA_UR2R_AMT	DMA_UR2R_DONE	DMA_UR2R_RXAH	DMA_UR2R_RXAL	
7EFA40	DMA_UR2T_CFG	DMA_UR2T_CR	DMA_UR2T_STA	DMA_UR2T_AMT	DMA_UR2T_DONE	DMA_UR2T_TXAH	DMA_UR2T_TXAL	
7EFA38	DMA_UR1R_CFG	DMA_UR1R_CR	DMA_UR1R_STA	DMA_UR1R_AMT	DMA_UR1R_DONE	DMA_UR1R_RXAH	DMA_UR1R_RXAL	
7EFA30	DMA_UR1T_CFG	DMA_UR1T_CR	DMA_UR1T_STA	DMA_UR1T_AMT	DMA_UR1T_DONE	DMA_UR1T_TXAH	DMA_UR1T_TXAL	
7EFA28	DMA_SPI_RXAL	DMA_SPI_CFG2						
7EFA20	DMA_SPI_CFG	DMA_SPI_CR	DMA_SPI_STA	DMA_SPI_AMT	DMA_SPI_DONE	DMA_SPI_TXAH	DMA_SPI_TXAL	DMA_SPI_RXAH
7EFA18	DMA_ADC_RXAL	DMA_ADC_CFG2	DMA_ADC_CHSW0	DMA_ADC_CHSW1				
7EFA10	DMA_ADC_CFG	DMA_ADC_CR	DMA_ADC_STA					DMA_ADC_RXAH
7EFA08	DMA_M2M_RXAL							

7EFA00	DMA_M2M_CFG	DMA_M2M_CR	DMA_M2M_STA	DMA_M2M_AMT	DMA_M2M_DONE	DMA_M2M_TXAH	DMA_M2M_TXAL	DMA_M2M_RXAH

10.3 STC32F12K60 Series

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H		LINICR	LINAR	LINDR	USBADR	S4CON	S4BUF	RSTCFG
F0H	B	CANICR			USBCON	IAP_TPS	IAP_ADDRE	ICHECR
E8H		WTST	CKCON	MXAX	USBDAT	DMAIR	IP3H	AUXINTIF
E0H	ACC			DPS			CMPCR1	CMPCR2
D8H		IAP_DATA1	IAP_DATA2	IAP_DATA3	USBCLK	T4T3M	ADCCFG	IP3
D0H	PSW	PSW1	TH4	TL4	TH3	TL3	TH2	TL2
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA0	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2	P_SW3	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	AUXR2
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	DPXL	SPH		PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
7EFEF8	PWMB_CCR6L	PWMB_CCR7H	PWMB_CCR7L	PWMB_CCR8H	PWMB_CCR8L	PWMB_BKR	PWMB_DTR	PWMB_OISR
7EFEF0	PWMB_PSCRH	PWMB_PSCRL	PWMB_ARRH	PWMB_ARRL	PWMB_RCR	PWMB_CCR5H	PWMB_CCR5L	PWMB_CCR6H
7EFEE8	PWMB_CCMR1	PWMB_CCMR2	PWMB_CCMR3	PWMB_CCMR4	PWMB_CCER1	PWMB_CCER2	PWMB_CNTRH	PWMB_CNTRL
7EFEE0	PWMB_CR1	PWMB_CR2	PWMB_SMCR	PWMB_ETR	PWMB_IER	PWMB_SR1	PWMB_SR2	PWMB_EGR
7EFED8	PWMA_CCR2L	PWMA_CCR3H	PWMA_CCR3L	PWMA_CCR4H	PWMA_CCR4L	PWMA_BKR	PWMA_DTR	PWMA_OISR
7EFED0	PWMA_PSCRH	PWMA_PSCRL	PWMA_ARRH	PWMA_ARRL	PWMA_RCR	PWMA_CCR1H	PWMA_CCR1L	PWMA_CCR2H
7EFEC8	PWMA_CCMR1	PWMA_CCMR2	PWMA_CCMR3	PWMA_CCMR4	PWMA_CCER1	PWMA_CCER2	PWMA_CNTRH	PWMA_CNTRL
7EFEC0	PWMA_CR1	PWMA_CR2	PWMA_SMCR	PWMA_ETR	PWMA_IER	PWMA_SR1	PWMA_SR2	PWMA_EGR
7EFEB8				CANAR	CANDR			
7EFEB0	PWMA_ETRPS	PWMA_ENO	PWMA_PS	PWMA_IOAUX	PWMB_ETRPS	PWMB_ENO	PWMB_PS	PWMB_IOAUX
7EFEA8	ADCTIM				T3T4PIN	ADCEXCFG	CMPEXCFG	
7EFEA0	TM0PS	TM1PS	TM2PS	TM3PS	TM4PS			
7EFE98	SPFUNC	RSTFLAG	RSTCR0	RSTCR1	RSTCR2	RSTCR3	RSTCR4	RSTCR5
7EFE88	I2CMSAUX							
7EFE80	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
7EFE70	YEAR	MONTH	DAY	HOUR	MIN	SEC	SSEC	
7EFE68	INIYEAR	INIMONTH	INIDAY	INIHOUR	INIMIN	INISEC	INISSEC	
7EFE60	RTCCR	RTCCFG	RTCIEN	RTCIF	ALAHOUR	ALAMIN	ALASEC	ALASSEC
7EFE50	LCMIFCFG	LCMIFCFG2	LCMIFCR	LCMIFSTA	LCMIFDATL	LCMDATH		
7EFE40	P0PD	P1PD	P2PD	P3PD	P4PD	P5PD	P6PD	P7PD

7EFE30	P0IE	P1IE	P2IE	P3IE	P4IE	P5IE	P6IE	P7IE
7EFE28	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR	P6DR	P7DR
7EFE20	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR	P6SR	P7SR
7EFE18	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
7EFE10	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
7EFE08	X32KCR			HSCLKDIV	HPLLCR	HPLLPSCR		
7EFE00	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	IRC48MCR
7EFDF8	CHIPID[24]	CHIPID[25]	CHIPID[26]	CHIPID[27]	CHIPID[28]	CHIPID[29]	CHIPID[30]	CHIPID[31]
7EFDF0	CHIPID[16]	CHIPID[17]	CHIPID[18]	CHIPID[19]	CHIPID[20]	CHIPID[21]	CHIPID[22]	CHIPID[23]
7EFDE8	CHIPID[8]	CHIPID[9]	CHIPID[10]	CHIPID[11]	CHIPID[12]	CHIPID[13]	CHIPID[14]	CHIPID[15]
7EFDE0	CHIPID[0]	CHIPID[1]	CHIPID[2]	CHIPID[3]	CHIPID[4]	CHIPID[5]	CHIPID[6]	CHIPID[7]
7EFDC8	USART2CR1	USART2CR2	USART2CR3	USART2CR4	USART2CR5	USART2GTR	USART2BRH	USART2BRL
7EFDC0	USARTCR1	USARTCR2	USARTCR3	USARTCR4	USARTCR5	USARTGTR	USARTBRH	USARTBRL
7EFDB0					S2CFG	S2ADDR	S2ADEN	
7EFDA8	CRECR	CRECNTH	CRECNTL	CRERES				
7EFDA0	I2S_MD							
7EFD98	I2S_CR	I2S_SR	I2S_DRH	I2S_DRL	I2S_PRH	I2S_PRL	I2S_CFGH	I2S_CFGL
7EFD60	PINIPL	PINIPH						
7EFD40	P0WKUE	P1WKUE	P2WKUE	P3WKUE	P4WKUE	P5WKUE	P6WKUE	P7WKUE
7EFD30	P0IM1	P1IM1	P2IM1	P3IM1	P4IM1	P5IM1	P6IM1	P7IM1
7EFD20	P0IM0	P1IM0	P2IM0	P3IM0	P4IM0	P5IM0	P6IM0	P7IM0
7EFD10	P0INTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF	P6INTF	P7INTF
7EFD00	P0INTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE	P6INTE	P7INTE
7EFBF8	HSSPI_CFG	HSSPI_CFG2	HSSPI_STA					
7EFBF0	HSPWMA_CFG	HSPWMA_ADR	HSPWMA_DAT		HSPWMB_CFG	HSPWMB_ADR	HSPWMB_DAT	
7EFAF8	DMA_ARBCFG	DMA_ARBSTA						
7EFAC0	DMA_I2ST_AMTH	DMA_I2ST_DONEH	DMA_I2SR_AMTH	DMA_I2SR_DONEH				
7EFAB8	DMA_I2SR_CFG	DMA_I2SR_CR	DMA_I2SR_STA	DMA_I2SR_AMT	DMA_I2SR_DONE	DMA_I2SR_RXAH	DMA_I2SR_RXAL	
7EFAB0	DMA_I2ST_CFG	DMA_I2ST_CR	DMA_I2ST_STA	DMA_I2ST_AMT	DMA_I2ST_DONE	DMA_I2ST_TXAH	DMA_I2ST_TXAL	
7EFAA8	DMA_I2CT_AMTH	DMA_I2CT_DONEH	DMA_I2CR_AMTH	DMA_I2CR_DONEH		DMA_I2C_CR	DMA_I2C_ST1	DMA_I2C_ST2
7EFAA0	DMA_I2CR_CFG	DMA_I2CR_CR	DMA_I2CR_STA	DMA_I2CR_AMT	DMA_I2CR_DONE	DMA_I2CR_RXAH	DMA_I2CR_RXAL	
7EFA98	DMA_I2CT_CFG	DMA_I2CT_CR	DMA_I2CT_STA	DMA_I2CT_AMT	DMA_I2CT_DONE	DMA_I2CT_TXAH	DMA_I2CT_TXAL	
7EFA90	DMA_UR3T_AMTH	DMA_UR3T_DONEH	DMA_UR3R_AMTH	dma_ur3r_doneh	DMA_UR4T_AMTH	DMA_UR4T_DONEH	DMA_UR4R_AMTH	DMA_UR4R_DONEH
7EFA88	DMA_UR1T_AMTH	DMA_UR1T_DONEH	DMA_UR1R_AMTH	DMA_UR1R_DONEH	DMA_UR2T_AMTH	DMA_UR2T_DONEH	DMA_UR2R_AMTH	dma_ur2r_doneh
7EFA80	DMA_M2M_AMTH	DMA_M2M_DONEH			DMA_SPI_AMTH	DMA_SPI_DONEH	DMA_LCM_AMTH	DMA_LCM_DONEH
7EFA78	DMA_LCM_RXAL							
7EFA70	DMA_LCM_CFG	DMA_LCM_CR	DMA_LCM_STA	DMA_LCM_AMT	DMA_LCM_DONE	DMA_LCM_TXAH	DMA_LCM_TXAL	DMA_LCM_RXAH
7EFA68	DMA_UR4R_CFG	DMA_UR4R_CR	DMA_UR4R_STA	DMA_UR4R_AMT	DMA_UR4R_DONE	DMA_UR4R_RXAH	DMA_UR4R_RXAL	
7EFA60	DMA_UR4T_CFG	DMA_UR4T_CR	DMA_UR4T_STA	DMA_UR4T_AMT	DMA_UR4T_DONE	DMA_UR4T_TXAH	DMA_UR4T_TXAL	
7EFA58	DMA_UR3R_CFG	DMA_UR3R_CR	DMA_UR3R_STA	DMA_UR3R_AMT	DMA_UR3R_DONE	DMA_UR3R_RXAH	dma_ur3r_rxal	
7EFA50	DMA_UR3T_CFG	DMA_UR3T_CR	DMA_UR3T_STA	DMA_UR3T_AMT	DMA_UR3T_DONE	DMA_UR3T_TXAH	DMA_UR3T_TXAL	
7EFA48	DMA_UR2R_CFG	DMA_UR2R_CR	DMA_UR2R_STA	DMA_UR2R_AMT	DMA_UR2R_DONE	DMA_UR2R_RXAH	DMA_UR2R_RXAL	
7EFA40	DMA_UR2T_CFG	DMA_UR2T_CR	DMA_UR2T_STA	DMA_UR2T_AMT	DMA_UR2T_DONE	DMA_UR2T_TXAH	DMA_UR2T_TXAL	
7EFA38	DMA_UR1R_CFG	DMA_UR1R_CR	DMA_UR1R_STA	DMA_UR1R_AMT	DMA_UR1R_DONE	DMA_UR1R_RXAH	DMA_UR1R_RXAL	

7EFA30	DMA_UR1T_CFG	DMA_UR1T_CR	DMA_UR1T_STA	DMA_UR1T_AMT	DMA_UR1T_DONE	DMA_UR1T_TXAH	DMA_UR1T_TXAL	
7EFA28	DMA_SPI_RXAL	DMA_SPI_CFG2						
7EFA20	DMA_SPI_CFG	DMA_SPI_CR	DMA_SPI_STA	DMA_SPI_AMT	DMA_SPI_DONE	DMA_SPI_TXAH	DMA_SPI_TXAL	DMA_SPI_RXAH
7EFA18	DMA_ADC_RXAL	DMA_ADC_CFG2	DMA_ADC_CHSW0	DMA_ADC_CHSW1				
7EFA10	DMA_ADC_CFG	DMA_ADC_CR	DMA_ADC_STA					DMA_ADC_RXAH
7EFA08	DMA_M2M_RXAL							
7EFA00	DMA_M2M_CFG	DMA_M2M_CR	DMA_M2M_STA	DMA_M2M_AMT	DMA_M2M_DONE	DMA_M2M_TXAH	DMA_M2M_TXAL	DMA_M2M_RXAH

10.4 Special Function Register List (SFR: 0x80-0xFF)

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 port	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
SP	stack pointer	81H									0000,0111
DPL	Data pointer (low byte)	82H									0000,0000
DPH	Data pointer (high byte)	83H									0000,0000
DPXL	Data pointer (highest byte)	84H									0000,0000
SPH	Stack pointer high byte	85H									0000,0000
PCON	Power Control Register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	Timer Control Register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	Timer Mode Register	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	Timer 0 Low 8-bit Register	8AH									0000,0000
TL1	Timer 1 Low 8-bit Register	8BH									0000,0000
TH0	Timer 0 High 8-bit Register	8CH									0000,0000
TH1	Timer 1 High 8-bit Register	8DH									0000,0000
AUXR	Auxiliary Register 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT	0000,0001
INTCLKO	Interrupt and Clock Output Control Registers	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	P1 port	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P1M1	P1 Port Configuration Register 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	P1 Port Configuration Register 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P0M1	P0 Port Configuration Register 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	P0 Port Configuration Register 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P2M1	P2 Port Configuration Register 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	P2 Port Configuration Register 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
AUXR2	Auxiliary Register 2	97H	-	-	-	-	CANSEL	CAN2EN	CANEN	LINEN	xxxx,0000
SCON	Serial Port 1 Control Register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	Serial Port 1 Data Register	99H									0000,0000
S2CON	Serial Port 2 Control Register	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
S2BUF	Serial Port 2 Data Register	9BH									0000,0000
IRCBAND	IRC Band Selection Detection	9DH	USBCKS	USBCKS2	-	-	-	-	SEL[1:0]		10xx,xxnn

LIRTRIM	IRC Frequency Trim Register	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		xxxx,xxnn
IRTRIM	IRC Frequency Adjustment Register	9FH	IRTRIM[7:0]								nnnn,nnnn
P2	P2 port	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
BUS_SPEED	Bus Speed Control Register	A1H	RW_S[1:0]					SPEED[2:0]			00xx,x000

P_SW1	Peripheral Port Switching Register 1	A2H	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]		nn00,0000
IE	Interrupt Allow Register	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
SADDR	Serial Port 1 Slave Address Register	A9H									0000,0000
WKTCL	Power-down wake-up timer low byte	AAH									1111,1111
WKTCH	Power-down wake-up timer high byte	ABH	WKTEN								0111,1111
S3CON	Serial Port 3 Control Register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	Serial Port 3 Data Register	ADH									0000,0000
TA	DPTR Timing Control Register	AEH									0000,0000
IE2	Interrupt Allow Register 2	AFH	EUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	0000,0000
P3	P3 port	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P3M1	P3 Port Configuration Register 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	1111,1100
P3M0	P3 Port Configuration Register 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	0000,0000
P4M1	P4 Port Configuration Register 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	P4 Port Configuration Register 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
IP2	Interrupt priority control register 2	B5H	PUSB	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	High Interrupt Priority Control Register 2	B6H	PUSBH	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H	0000,0000
IPH	High Interrupt Priority Control Register	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP	Interrupt Priority Control Register	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
SADEN	Serial Port 1 Slave Address Mask Register	B9H									0000,0000
P_SW2	Peripheral Port Switching Register 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000
P_SW3	Peripheral Port Switching Register 2	BBH	I2S_S[1:0]		S2SPI_S[1:0]		S1SPI_S[1:0]		CAN2_S[1:0]		0000,0000
ADC_CONTR	ADC Control Register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				0000,0000
ADC_RES	ADC conversion result high register	BDH									0000,0000
ADC_RESL	ADC conversion result low register	BEH									0000,0000
P4	P4 port	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 Data Register	C2H									0000,0000
IAP_ADDRH	IAP High Address Register	C3H									0000,0000
IAP_ADDRL	IAP Low Address Register	C4H									0000,0000
IAP_CMD	IAP Command Register	C5H	-	-	-	-	-	CMD[2:0]			xxxx,x000
IAP_TRIG	IAP Trigger Register	C6H									0000,0000
IAP_CONTR	IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx

P5	P5 port	C8H	-	-	P55	P54	P53	P52	P51	P50	xx11,1111
P5M1	P5 Port Configuration Register 1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx11,1111
P5M0	P5 Port Configuration Register 0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx00,0000
P6M1	P6 Port Configuration Register 1	CBH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1	1111,1111
P6M0	P6 Port Configuration 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	0000,0000
PSW1	Program Status Word 1 Register	D1H	CY	AC	N	RS1	RS0	OV	Z		0000,000x
T4H	Timer 4 High Byte	D2H									0000,0000
T4L	Timer 4 Low Byte	D3H									0000,0000

T3H	Timer 3 High Byte	D4H								0000,0000
T3L	Timer 3 Low Byte	D5H								0000,0000
T2H	Timer 2 High Byte	D6H								0000,0000
T2L	Timer 2 Low Byte	D7H								0000,0000
IAP_DATA1	IAP Data Register 1	D9H								0000,0000
IAP_DATA2	IAP Data Register 2	DAH								0000,0000
IAP_DATA3	IAP Data Register 3	DBH								0000,0000
USBCLK	USB Clock Control Register	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]	0010,0000
T4T3M	Timer 4/3 Control Register	DDH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
ADCCFG	ADC Configuration Register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000
IP3	Interrupt priority control register 3	DFH	-	-	-	-	PI2S	PRTC	PS4	PS3	xxxx,0000
ACC	accumulator (computing)	E0H									0000,0000
P7M1	P7 Port Configuration Register 1	E1H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1	1111,1111
P7M0	P7 Port Configuration Register 0	E2H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0	0000,0000
DPS	DPTR Pointer Selector	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
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
P6	P6 port	E8H	P67	P66	P65	P64	P63	P62	P61	P60	1111,1111
WTST	Program reads the wait control register	E9H									0000,0111
CKCON	XRAM Control Register	EAH	-	-	-	T1M	T0M	CKCON[2:0]			0000,0111
MXAX	MOVX Extended Address Register	EBH									0000,0001
USBDAT	USB Data Register	ECH									0000,0000
DMAIR	FMU DMA Instruction Register	EDH									0000,0000
IP3H	High Interrupt Priority Control Register 3	EEH	-	-	-	-	PI2SH	PRTCH	PS4H	PS3H	xxxx,0000
AUXINTIF	Extended External Interrupt Flag Register	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
B	B Register	F0H									0000,0000
CANICR	CANBUS Interrupt Control Register	F1H	PCAN2H	CAN2IF	CAN2IE	PCAN2L	PCANH	CANIF	CANIE	PCANL	0000,0000
USBCON	USB Control Register	F4H	ENUSB	USBRST	PS2M	PUEN	PDEN	DFREC	DP	DM	0000,0000
IAP_TPS	IAP Wait Time Control Register	F5H	-	-	IAP_TPS[5:0]						xx00,0000
IAP_ADDRE	IAP Extended High Address Register	F6H									1111,1111
ICHECR	CACHE Control Register	F7H	CON	HIT	CLR					EN	000x,xxx0
P7	P7 port	F8H	P77	P76	P75	P74	P73	P72	P71	P70	1111,1111
LINICR	LINBUS Interrupt Control Register	F9H					PLINH	LINIF	LINIE	PLINL	0000,0000
LINAR	LINBUS Address Register	FAH									0000,0000
LINDR	LINBUS Data Register	FBH									0000,0000
USBADR	USB Address Register	FCH	BUSY	AUTORD	UADR[5:0]						0000,0000

S4CON	Serial Port 4 Control Register	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	Serial Port 4 Data Register	FEH									0000,0000
RSTCFG	Reset Configuration Register	FFH	-	ENLVR	-	P54RST	-	-	LVDS[1:0]		x0x0,xx00

10.5 Extended Special Function Register List (XFR: 0x7EFE00-0x7EFEFF)

The following special function registers are Extended SFR (XFR) with logical address located in XDATA area. Before accessing, you need to set the highest bit (EAXFR) of P_SW2 register to 1, and then use MOV @DRk, Rm and MOV Rm, @DRk instructions to access them, for example:

```
MOV A,#00H
MOV WR6,#WORD0 CLKSEL      CLKSEL can be replaced by the register to be accessed.
MOV WR4,#WORD2 CLKSEL
MOV @DR4,R11
```

respond in singing

```
MOV WR6,#WORD0 CLKSEL      CLKSEL can be replaced by the register to be accessed.
MOV WR4,#WORD2 CLKSEL
MOV R11,@DR4
```

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	Clock Select Register	7EFE00H	CKMS	HSIOCK	-	-	MCK2SEL[1:0]		MCKSEL [1:0]		00xx,0000
CLKDIV	Clock Divider Register	7EFE01H									0000,0100
HIRCCR	Internal high-speed oscillator control register	7EFE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	External Crystal Control Register	7EFE03H	ENXOSC	XITYPE	GAIN	-	XCFILTER[1:0]		-	XOSCST	000x,00x0
IRC32KCR	Internal 32K Oscillator Control Registers	7EFE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	Master Clock Output Control Register	7EFE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	Internal high-speed oscillator de-jitter control	7EFE06H									1000,0000
IRC48MCR	Internal 48M Oscillator Control Registers	7EFE07H	ENIRC48M	-	-	-	-	-	-	IRC48MST	1xxx,xxx0
X32KCR	External 32K Crystal Control Register	7FFE08H	ENX32K	GAIN32K	-	-	-	-	-	X32KST	00xx,xxx0
HSCLKDIV	High Speed Clock Divider Register	7EFE0BH									0000,0010
HPLLCR	High-speed PLL control registers	7EFE0CH	ENHPLL	-	-	-	HPLLDIV[3:0]				0xxx,0000
HPLLPSCR	High-speed PLL prescaler register	7EFE0DH	-	-	-	-	HPLL_PREDIV[3:0]				xxxx,0000
P0PU	P0 Port Pull-up Resistor Control Register	7EFE10H									0000,0000
P1PU	P1 Port Pull-up Resistor Control Register	7EFE11H									0000,0000
P2PU	P2 Port Pull-Up Resistor Control Register	7EFE12H									0000,0000
P3PU	P3 Port Pull-up Resistor Control Register	7EFE13H									0000,0000
P4PU	P4 Port Pull-Up Resistor	7EFE14H									0000,0000

P4NCS	P4 Port Schmitt Trigger Control Register	7EFE1CH						0000,0000
P5NCS	P5 Port Schmitt Trigger Control Register	7EFE1DH	-	-	-			xxx0,0000
P6NCS	P6 Port Schmitt Trigger Control Register	7EFE1EH						0000,0000
P7NCS	P7 Port Schmitt Trigger Control Register	7EFE1FH						0000,0000
P0SR	P0 Port Level Shift Rate Register	7EFE20H						1111,1111
P1SR	P1 Port Level Shift Rate Register	7EFE21H						1111,1111
P2SR	P2 Port Level Shift Rate Register	7EFE22H						1111,1111
P3SR	P3 Port Level Shift Rate Register	7EFE23H						1111,1111
P4SR	P4 Port Level Shift Rate Register	7EFE24H						1111,1111
P5SR	P5 Port Level Shift Rate Register	7EFE25H	-	-	-			xxx1,1111
P6SR	P6 Port Level Shift Rate Register	7EFE26H						1111,1111
P7SR	P7 Port Level Shift Rate Register	7EFE27H						1111,1111
P0DR	P0 Port Drive Current Control Register	7EFE28H						1111,1111
P1DR	P1 Port Drive Current Control Register	7EFE29H						1111,1111
P2DR	P2 Port Drive Current Control Register	7EFE2AH						1111,1111
P3DR	P3 Port Drive Current Control Register	7EFE2BH						1111,1111
P4DR	P4 Port Drive Current Control Register	7EFE2CH						1111,1111
P5DR	P5 Port Drive Current Control Register	7EFE2DH	-	-	-			xxx1,1111
P6DR	P6 Port Drive Current Control Register	7EFE2EH						1111,1111
P7DR	P7 Port Drive Current Control Register	7EFE2FH						1111,1111
P0IE	P0 Port Input Enable Control Register	7EFE30H						1111,1111
P1IE	P1 Port Input Enable Control Register	7EFE31H						1111,1111
P2IE	P2 Port Input Enable Control Register	7EFE32H						1111,1111
P3IE	P3 Port Input Enable Control Register	7EFE33H						1111,1111
P4IE	P4 Port Input Enable Control Register	7EFE34H						1111,1111
P5IE	P5 Port Input Enable Control Register	7EFE35H	-	-	-			xxx1,1111
P6IE	P6 Port Input Enable	7EFE36H						1111,1111

	Control Register									
P7IE	P7 Port Input Enable Control Register	7EFE37H								1111,1111
P0PD	P0 Port Pull Down Resistor Control Register	7EFE40H								0000,0000
P1PD	P1 Port Pull Down Resistor Control Register	7EFE41H								0000,0000
P2PD	P2 Port Pull Down Resistor Control Register	7EFE42H								0000,0000
P3PD	P3 Port Pull Down Resistor Control Register	7EFE43H								0000,0000
P4PD	P4 Port Pull Down Resistor Control Register	7EFE44H								0000,0000
P5PD	P5 Port Pull Down Resistor Control Register	7EFE45H	-	-	-					xxx1,1111
P6PD	P6 Port Pull Down Resistor Control Register	7EFE46H								0000,0000
P7PD	P7 Port Pull-Down Resistor Control Register	7EFE47H								0000,0000
LCMIFCFG	LCM Interface Configuration Register	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFCFG2	LCM Interface Configuration Register 2	7EFE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000
LCMIFCR	LCM Interface Control Register	7EFE52H	ENLCMIF	-	-	-	-	CMD[2:0]			0xxx,x000
LCMIFSTA	LCM Interface Status Register	7EFE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
LCMIFDATL	LCM Interface Low Byte Data	7EFE54H	LCMIFDAT[7:0]								0000,0000
LCMIFDATH	LCM Interface High Byte Data	7EFE55H	LCMIFDAT[15:8]								0000,0000
RTCCR	RTC Control Register	7EFE60H	-	-	-	-	-	-	-	RUNRTC	xxxx,xxx0
RTCCFG	RTC Configuration Register	7EFE61H	-	-	-	-	-	-	RTCCKS	SETRTC	xxxx,xx00

RTCIEN	RTC Interrupt Enable Register	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I	0000,0000
RTCIF	RTC Interrupt Request Register	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF	0000,0000
ALAHOUR	RTC Hourly value of the alarm	7EFE64H	-	-	-						xxx0,0000
ALAMIN	RTC Minute value of the alarm	7EFE65H	-	-							xx00,0000
ALASEC	RTC Alarm Clock seconds value	7EFE66H	-	-							xx00,0000
ALASSEC	1/128 second value for RTC alarms	7EFE67H	-								x000,0000
INIYEAR	RTC Year Initialisation	7EFE68H	-								x000,0000
INIMONTH	RTC Month Initialisation	7EFE69H	-	-	-	-					xxxx,0000
INIDAY	RTC Day Initialisation	7EFE6AH	-	-	-						xxx0,0000
INIHOUR	RTC Hourly Initialisation	7EFE6BH	-	-	-						xxx0,0000
INIMIN	RTC Minute Initialisation	7EFE6CH	-	-							xx00,0000
INISEC	RTC second initialisation	7EFE6DH	-	-							xx00,0000
INISSEC	RTC1/128 seconds initialisation	7EFE6EH	-								x000,0000
YEAR	Annualised value of RTC	7EFE70H	-								x000,0000
MONTH	Monthly count of RTC	7EFE71H	-	-	-	-					xxxx,0000
DAY	Daily count of RTC	7EFE72H	-	-	-						xxx0,0000
HOUR	Hourly count of RTC	7EFE73H	-	-	-						xxx0,0000
MIN	Minute count value of RTC	7EFE74H	-	-							xx00,0000
SEC	Seconds value of RTC	7EFE75H	-	-							xx00,0000
SSEC	1/128 second count value for RTC	7EFE76H	-								x000,0000
I2CCFG	I² C Configuration Register	7EFE80H	ENI2C	MSSL	MSSPEED[6:1]						0000,0000
I2CMSCR	I² C Host Control Register	7EFE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I² C Host Status Register	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx10
I2CSLCR	I² C Slave Control Registers	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I² C Slave Status Register	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I² C Slave Address Registers	7EFE85H	SLADR [6:0]							MA	0000,0000
I2CTXD	I² C Data Transmission Register	7EFE86H									0000,0000
I2CRXD	I² C Data Receiving Register	7EFE87H									0000,0000
I2CMSAUX	I² C Host Auxiliary Control Registers	7EFE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
SPFUNC	Auxiliary Control Register	7EFE98H	-	-	-	-	-	-	-	BKSWR	xxxx,xxx0
RSTFLAG	Reset Flag Register	7EFE99H	-	-	-	LVDRSTF	WDTRSTF	SWRSTF	ROMOVF	EXRSTF	xxx1,0100
RSTCR0	Reset Control Register 0	7EFE9AH	-	-	-	-	RSTTM34	TSTTM2	-	RSTTM01	xxxx,00x0
RSTCR1	Reset Control Register 1	7EFE9BH	-	-	-	-	RSTUR4	RSTUR3	RSTUR2	RSTUR1	xxxx,0000
RSTCR2	Reset Control Register 2	7EFE9CH	RSTCAN2	RSTCAN	RSTLIN	RSTRTC	RSTPWMB	RSTPWMA	RSTI2C	RSTSPI	0000,0000
RSTCR3	Reset Control Register 3	7EFE9DH	RSTFPU	RSTDMA	RSTLCM	RSTLCD	RSTLED	RSTTKS	RSTCMP	RSTADC	0000,0000
RSTCR4	Reset Control Register 4	7EFE9EH	-	-	-	-	-	-	-	RSTMDU	xxxx,xxx0
RSTCR5	Reset Control Register 5	7EFE9FH	-	-	-	-	-	-	-	-	xxxx,xxxx
TM0PS	Timer 0 Clock prescaler	7EFEA0H									0000,0000

	Register										
TM1PS	Timer 1 Clock prescaler register	7EFEA1H									0000,0000
TM2PS	Timer 2 Clock Preshunt Register	7EFEA2H									0000,0000
TM3PS	Timer 3 Clock Preshunt Register	7EFEA3H									0000,0000
TM4PS	Timer 4 Clock Preshunt Register	7EFEA4H									0000,0000
ADCTIM	ADC Timing Control Register	7EFEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010, 1010
T3T4PIN	T3/T4 Selection Register	7EFEACH	-	-	-	-	-	-	-	T3T4SEL	xxxx,xxx0

ADCEXCFG	ADC Extended Configuration Register	7EFEADH	-	-	ADCETRS [1:0]		-	CVTIMESEL [2:0]			xx00,x000
CMPEXCFG	Comparator Extended Configuration Register	7EFEAEH	CHYS[1:0]		-	-	-	CMPNS	CMPPS[1:0]		CHYS[1:0]
PWMA_ETRPS	ETR Selection Register for PWMA	7EFEB0H						BRKAPS	ETRAPS[1:0]		xxxx,x000
PWMA_ENO	PWMA output enable control	7EFEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P	0000,0000
PWMA_PS	PWMA Output Pin Select Register	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMA_IOAUX	PWMA Auxiliary Register	7EFEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P	0000,0000
PWMB_ETRPS	ETR Selection Register for PWMB	7EFEB4H						BRKBPS	ETRBPS[1:0]		xxxx,x000
PWMB_ENO	PWMB output enable control	7EFEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P	x0x0,x0x0
PWMB_PS	PWMB Output Pin Select Register	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMB_IOAUX	PWMB Auxiliary Register	7EFEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P	x0x0,x0x0
CANAR	CANBUS Address Register	7EFEBBH									0000,0000
CANDR	CANBUS Data Register	7EFEBCH									0000,0000
PWMA_CR1	PWMA control register 1	7EFEC0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMA_CR2	PWMA Control Register 2	7EFEC1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMA_SMCR	PWMA Slave Mode Control Register	7EFEC2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMA_ETR	PWMA External Trigger Register	7EFEC3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMA_IER	PWMA Interrupt Enable Register	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA status register 1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA status register 2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMA_EGR	PWMA Event Generation Register	7EFEC7H	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG	0000,0000
PWMA_CCMR1	PWMA Capture Mode Register 1	7EFEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		0000,0000
	PWMA Compare Mode Register 1		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]		0000,0000
PWMA_CCMR2	PWMA Capture Mode Register 2	7EFEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		0000,0000
	PWMA Compare Mode Register 2		IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		0000,0000
PWMA_CCMR3	PWMA Capture Mode Register 3	7EFECAH	OC3CE	OC3M [2:0]			OC3PE	OC3FE	CC3S[1:0]		0000,0000
	PWMA Compare Mode Register 3		IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]		0000,0000
PWMA_CCMR4	PWMA Capture Mode Register 4	7EFECBH	OC4CE	OC4M [2:0]			OC4PE	OC4FE	CC4S[1:0]		0000,0000
	PWMA Compare Mode Register 4		IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]		0000,0000
PWMA_CCER1	PWMA Capture Compare Enable Register 1	7EFECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E	0000,0000

PWMA_CCER2	PWMA Capture Compare Enable Register 2	7EFECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	0000,0000
PWMA_CNTRH	PWMA Counter High Byte	7EFECEH	CNT [15:8]								0000,0000
PWMA_CNTRL	PWMA Counter Low Byte	7EFECFH	CNT [7:0]								0000,0000
PWMA_PSCRH	PWMA prescaled high byte	7EFED0H	PSC [15:8]								0000,0000
PWMA_PSCRL	PWMA prescaled low byte	7EFED1H	PSC[7:0]								0000,0000
PWMA_ARRH	PWMA Auto Reload Register High Byte	7EFED2H	ARR[15:8]								0000,0000
PWMA_ARRL	PWMA Auto Reload Register Low Byte	7EFED3H	ARR[7:0]								0000,0000
PWMA_RCR	PWMA Repeat Counter Register	7EFED4H	REP[7:0]								0000,0000
PWMA_CCR1H	PWMA Compare Capture Register 1 high	7EFED5H	CCR1[15:8]								0000,0000
PWMA_CCR1L	PWMA Compare Capture Register 1 Low	7EFED6H	CCR1[7:0]								0000,0000
PWMA_CCR2H	PWMA Compare Capture Register 2 high	7EFED7H	CCR2[15:8]								0000,0000
PWMA_CCR2L	PWMA Compare Capture Register 2 Low	7EFED8H	CCR2[7:0]								0000,0000
PWMA_CCR3H	PWMA Compare Capture Register 3 high	7EFED9H	CCR3[15:8]								0000,0000

PWMA_CCR3L	PWMA Compare Capture Register 3 Low	7EFEDAH	CCR3[7:0]							0000,0000
PWMA_CCR4H	PWMA Compare Capture Register 4 high	7EFEDBH	CCR4[15:8]							0000,0000
PWMA_CCR4L	PWMA Compare Capture Register 4 Low	7EFEDCH	CCR4[7:0]							0000,0000
PWMA_BKR	PWMA Brake Register	7EFEDDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		0000,000x
PWMA_DTR	PWMA Deadband Control Register	7EFEDEH	DTG[7:0]							0000,0000
PWMA_OISR	PWMA Output Idle Status Register	7EFEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	0000,0000
PWMB_CR1	PWMB control register 1	7EFEE0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWMB_CR2	PWMB Control Register 2	7EFEE1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWMB_SMCR	PWMB Slave Mode Control Register	7EFEE2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWMB_ETR	PWMB External Trigger Register	7EFEE3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWMB_IER	PWMB Interrupt Enable Register	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB status register 1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB status register 2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x
PWMB_EGR	PWMB Event Generation Register	7EFEE7H	BG	TG	COMG	CC8G	CC7G	CC6G	CC5G	UG	0000,0000
PWMB_CCMR1	PWMB Capture Mode Register 1	7EFEE8H	OC5CE	OC5M [2:0]			OC5PE	OC5FE	CC5S[1:0]		0000,0000
	PWMB Compare Mode Register 1		IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]		0000,0000
PWMB_CCMR2	PWMB Capture Mode Register 2	7EFEE9H	OC6CE	OC6M [2:0]			OC6PE	OC6FE	CC6S[1:0]		0000,0000
	PWMB Compare Mode Register 2		IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]		0000,0000
PWMB_CCMR3	PWMB Capture Mode Register 3	7EFEEAH	OC7CE	OC7M [2:0]			OC7PE	OC7FE	CC7S[1:0]		0000,0000
	PWMB Compare Mode Register 3		IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]		0000,0000
PWMB_CCMR4	PWMB Capture Mode Register 4	7EFEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]		0000,0000
	PWMB Compare Mode Register 4		IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]		0000,0000
PWMB_CCER1	PWMB Capture Compare Enable Register 1	7EFEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWMB_CCER2	PWMB Capture Compare Enable Register 2	7EFEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E	xx00,xx00
PWMB_CNTRH	PWMB Counter High Byte	7EFEEEH	CNT [15:8]								0000,0000
PWMB_CNTRL	PWMB Counter Low Byte	7EFEEFH	CNT [7:0]								0000,0000
PWMB_PSCRH	PWMB Preshared High Byte	7EFEF0H	PSC [15:8]								0000,0000
PWMB_PSCRL	PWMB prescaled low byte	7EFEF1H	PSC[7:0]								0000,0000
PWMB_ARRH	PWMB Auto Reload Register High Byte	7EFEF2H	ARR[15:8]								0000,0000
PWMB_ARRL	PWMB Auto Reload Register Low Byte	7EFEF3H	ARR[7:0]								0000,0000
PWMB_RCR	PWMB Repeat Counter	7EFEF4H	REP[7:0]								0000,0000

	Register										
PWMB_CCR5H	PWMB Compare Capture Register 5 high	7EFEF5H	CCR1[15:8]								0000,0000
PWMB_CCR5L	PWMB Compare Capture Register 5 Low	7EFEF6H	CCR1[7:0]								0000,0000
PWMB_CCR6H	PWMB Compare Capture Register 6 high	7EFEF7H	CCR2[15:8]								0000,0000
PWMB_CCR6L	PWMB Compare Capture Register 6 Low	7EFEF8H	CCR2[7:0]								0000,0000
PWMB_CCR7H	PWMB Compare Capture Register 7 high	7EFEF9H	CCR3[15:8]								0000,0000
PWMB_CCR7L	PWMB Compare Capture Register 7 Low	7EFEFAH	CCR3[7:0]								0000,0000
PWMB_CCR8H	PWMB Compare Capture Register 8 Upper	7EFEFBH	CCR4[15:8]								0000,0000
PWMB_CCR8L	PWMB Compare Capture Register 8 Low	7EFEFCH	CCR4[7:0]								0000,0000
PWMB_BKR	PWMB Brake Register	7EFEFDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]	-	0000,000x
PWMB_DTR	PWMB Deadband Control Register	7EFEFEH	DTG[7:0]								0000,0000
PWMB_OISR	PWMB Output Idle Status Register	7EFEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5	x0x0,x0x0

10.6 Extended Special Function Register List (XFR: 0x7EFD00-0x7EFDFF)

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	P0 Port Interrupt Enable Register	7EFD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 Port Interrupt Enable Register	7EFD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 Port Interrupt Enable Register	7EFD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 Port Interrupt Enable Register	7EFD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 Port Interrupt Enable Register	7EFD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 Port Interrupt Enable Register	7EFD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	xx00,0000
P6INTE	P6 Port Interrupt Enable Register	7EFD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE	0000,0000
P7INTE	P7 Port Interrupt Enable Register	7EFD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE	0000,0000
P0INTF	P0 Port Interrupt Flag Register	7EFD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 Port Interrupt Flag Register	7EFD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 Port Interrupt Flag Register	7EFD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 Port Interrupt Flag Register	7EFD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P4INTF	P4 Port Interrupt Flag Register	7EFD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 Port Interrupt Flag Register	7EFD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	xx00,0000
P6INTF	P6 Port Interrupt Flag Register	7EFD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF	0000,0000
P7INTF	P7 Port Interrupt Flag Register	7EFD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF	0000,0000
P0IM0	P0 port interrupt mode register 0	7EFD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 port interrupt mode register 0	7EFD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 port interrupt mode register 0	7EFD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 port interrupt mode register 0	7EFD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P4IM0	P4 port interrupt mode register 0	7EFD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0	0000,0000
P5IM0	P5 port interrupt mode register 0	7EFD25H	-	-	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	xx00,0000
P6IM0	P6 port interrupt mode	7EFD26H	P67IM0	P66IM0	P65IM0	P64IM0	P63IM0	P62IM0	P61IM0	P60IM0	0000,0000

	register 0										
P7IM0	P7 port interrupt mode register 0	7EFD27H	P77IM0	P76IM0	P75IM0	P74IM0	P73IM0	P72IM0	P71IM0	P70IM0	0000,0000
P0IM1	P0 port interrupt mode register 1	7EFD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1 port interrupt mode register 1	7EFD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 port interrupt mode register 1	7EFD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 port interrupt mode register 1	7EFD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P4IM1	P4 Port Interrupt Mode Register 1	7EFD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1	0000,0000
P5IM1	P5 Port Interrupt Mode Register 1	7EFD35H	-	-	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	xx00,0000
P6IM1	P6 Port Interrupt Mode Register 1	7EFD36H	P67IM1	P66IM1	P65IM1	P64IM1	P63IM1	P62IM1	P61IM1	P60IM1	0000,0000
P7IM1	P7 Port Interrupt Mode Register 1	7EFD37H	P77IM1	P76IM1	P75IM1	P74IM1	P73IM1	P72IM1	P71IM1	P70IM1	0000,0000
P0WKUE	P0 port interrupt wake-up enable	7EFD40H	P07WKUE	P06WKUE	P05WKUE	P04WKUE	P03WKUE	P02WKUE	P01WKUE	P00WKUE	0000,0000
P1WKUE	P1 port interrupt wake-up enable	7EFD41H	P17WKUE	P16WKUE	P15WKUE	P14WKUE	P13WKUE	P12WKUE	P11WKUE	P10WKUE	0000,0000
P2WKUE	P2 port interrupt wake-up enable	7EFD42H	P27WKUE	P26WKUE	P25WKUE	P24WKUE	P23WKUE	P22WKUE	P21WKUE	P20WKUE	0000,0000
P3WKUE	P3 port interrupt wake-up enable	7EFD43H	P37WKUE	P36WKUE	P35WKUE	P34WKUE	P33WKUE	P32WKUE	P31WKUE	P30WKUE	0000,0000
P4WKUE	P4 port interrupt wake-up enable	7EFD44H	P47WKUE	P46WKUE	P45WKUE	P44WKUE	P43WKUE	P42WKUE	P41WKUE	P40WKUE	0000,0000
P5WKUE	P5 port interrupt wake-up enable	7EFD45H	-	-	P55WKUE	P54WKUE	P53WKUE	P52WKUE	P51WKUE	P50WKUE	xx00,0000
P6WKUE	P6 port interrupt wake-up enable	7EFD46H	P67WKUE	P66WKUE	P65WKUE	P64WKUE	P63WKUE	P62WKUE	P61WKUE	P60WKUE	0000,0000

P7WKUE	P7 port interrupt wake-up enable	7EFD47H	P77WKUE	P76WKUE	P75WKUE	P74WKUE	P73WKUE	P72WKUE	P71WKUE	P70WKUE	0000,0000
PINIPL	I/O Port Interrupt Priority Low Register	7EFD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O Port Interrupt Priority High Register	7EFD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
FSHWUPPRD	FLASH Wake-Up Wait Time Register	7EFD68H									0011,1100
UR1TOCR	Serial Port 1 Timeout Control Register	7EFD70H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR1TOSR	Serial port 1 timeout status register	7EFD71H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
UR1TOTH	Serial Port 1 Timeout Length Control Register	7EFD72H	TM[15:8]								0000,0000
UR1TOTL	Serial Port 1 Timeout Length Control Register	7EFD73H	TM[7:0]								0000,0000
UR2TOCR	Serial Port 2 Timeout Control Register	7EFD74H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR2TOSR	Serial Port 2 Timeout Status Register	7EFD75H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
UR2TOTH	Serial Port 2 Timeout Length Control Register	7EFD76H	TM[15:8]								0000,0000
UR2TOTL	Serial Port 2 Timeout Length Control Register	7EFD77H	TM[7:0]								0000,0000
UR3TOCR	Serial Port 3 Timeout Control Register	7EFD78H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR3TOSR	Serial Port 3 Timeout Status Register	7EFD79H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
UR3TOTH	Serial Port 3 Timeout Length Control Register	7EFD7AH	TM[15:8]								0000,0000
UR3TOTL	Serial Port 3 Timeout Length Control Register	7EFD7BH	TM[7:0]								0000,0000
UR4TOCR	Serial Port 4 Timeout Control Register	7EFD7CH	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR4TOSR	Serial Port 4 Timeout Status Register	7EFD7DH	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
UR4TOTH	Serial Port 4 Timeout Length Control Register	7EFD7EH	TM[15:8]								0000,0000
UR4TOTL	Serial Port 4 Timeout Length Control Register	7EFD7FH	TM[7:0]								0000,0000
SPITOCR	SPI Timeout Control Register	7EFD80H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
SPITOSR	SPI Timeout Status Register	7EFD81H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
SPITOTH	SPI Timeout Length Control Register	7EFD82H	TM[15:8]								0000,0000
SPITOTL	SPI Timeout Length Control Register	7EFD83H	TM[7:0]								0000,0000
I2CTOCR	I2C Timeout Control Register	7EFD84H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
I2CTOSR	I2C Timeout Status Register	7EFD85H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
I2CTOTH	I2C Timeout Length Control Register	7EFD86H	TM[15:8]								0000,0000
I2CTOTL	I2C Timeout Length	7EFD87H	TM[7:0]								0000,0000

	Control Register										
I2SCR	I2S Control Register	7EFD98H	TXEIE	RXNEIE	ERRIE	FRF	-	-	TXDMAEN	RXDMAEN	0000,xx00
I2SSR	I2S Status Register	7EFD99H	-	FRE	BUY	OVR	UDR	CHSID	TXE	RXNE	x000,0000
I2SDRH	I2S Data Register High Byte	7EFD9AH	DR[15:8]								0000,0000
I2SDRL	I2S Data Register Low Byte	7EFD9BH	DR[7:0]								0000,0000
I2SPRH	I2S Crossover Register High Byte	7EFD9CH	-	-	-	-	-	-	MCKOE	ODD	xxxx,xx00
I2SPRL	I2S Crossover Register Low Byte	7EFD9DH	DIV[7:0]								0000,0000
I2SCFGH	I2S Configuration Register High Byte	7EFD9EH	-	-	-	-	-	I2SE	I2SCFG[1:0]		xxxx,x000
I2SCFGL	I2S Configuration Register Low Byte	7EFD9FH	PCMSYNC	-	STD[1:0]		CKPOL	DATLEN[1:0]		CHLEN	0x00,0000
I2SMD	I2S Slave Mode Control Register	7EFDA0H	MD[7:0]								0000,0000
CRECR	CRE Control Register	7EFDA8H	ENCRE	MONO	UPT[1:0]		CREHF	CREINC	CREDEC	CRERDY	0000,0000
CRECNTH	CRE Calibration target register	7EFDA9H	CNT [15:8]								0000,0000
CRECNTL	CRE Calibration target register	7EFDAAH	CNT [7:0]								0000,0000
CRERES	CRE Resolution Control Register	7EFDABH	RES[7:0]								0000,0000
S2CFG	Serial Port 2 Configuration Register	7EFDB4H	-	S2MOD0	S2M0x6	-	-	-	-	W1	000x,xxx0
S2ADDR.	Serial 2 Slave Address Register	7EFDB5H									0000,0000
S2ADEN	Serial Port 2 Slave Address Mask Register	7EFDB6H									0000,0000

USARTCR1	Serial Port 1 Control Register 1	7EFDC0H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA	0000,0000
USARTCR2	Serial Port 1 Control Register 2	7EFDC1H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE	0000,0000
USARTCR3	Serial Port 1 Control Register 3	7EFDC2H	IrDA_LPBAUD[7:0]								0000,0111
USARTCR4	Serial Port 1 Control Register 4	7EFDC3H	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]		xxxx,0000
USARTCR5	Serial Port 1 Control Register 5	7EFDC4H	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC	0000,0000
USARTGTR	Serial Port 1 Protection Time Register	7EFDC5H									0000,0000
USARTBRH	Serial Port 1 Baud Rate Register	7EFDC6H	USARTBR[15:8]								0000,0000
USARTBRL	Serial Port 1 Baud Rate Register	7EFDC7H	USARTBR[7:0]								0000,0000
USART2CR1	Serial Port 2 Control Register 1	7EFDC8H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA	0000,0000
USART2CR2	Serial Port 2 Control Register 2	7EFDC9H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE	0000,0000
USART2CR3	Serial Port 2 Control Register 3	7EFDCAH	IrDA_LPBAUD[7:0]								0000,0000
USART2CR4	Serial Port 2 Control Register 4	7EFDCBH	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]		xxxx,0000
USART2CR5	Serial Port 2 Control Register 5	7EFDCCH	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNCAN	0000,0000
USART2GTR	Serial Port 2 Protection Time Register	7EFDCDH									0000,0000
USART2BRH	Serial Port 2 Baud Rate Register	7EFDCEH	USART2BR[15:8]								0000,0000
USART2BRL	Serial Port 2 Baud Rate Register	7EFDCFH	USART2BR[7:0]								0000,0000
CHIPID0	Hardware numbers ID00	7EFDE0H	Globally unique ID number (byte 0)								nnnn,nnnn
CHIPID1	Hardware number ID01	7EFDE1H	Globally unique ID number (1st byte)								nnnn,nnnn
CHIPID2	Hardware Digital ID02	7EFDE2H	Globally unique ID number (byte 2)								nnnn,nnnn
CHIPID3	Hardware Digital ID03	7EFDE3H	Globally unique ID number (byte 3)								nnnn,nnnn
CHIPID4	Hardware Digital ID04	7EFDE4H	Globally unique ID number (byte 4)								nnnn,nnnn
CHIPID5	Hardware Digital ID05	7EFDE5H	Globally unique ID number (byte 5)								nnnn,nnnn
CHIPID6	Hardware Digital ID06	7EFDE6H	Globally unique ID number (byte 6)								nnnn,nnnn
CHIPID7	Hardware Digital ID07	7EFDE7H	Internal 1.19V reference signal source (high byte)								nnnn,nnnn
CHIPID8	Hardware Digital ID08	7EFDE8H	Internal 1.19V reference signal source (low byte)								nnnn,nnnn
CHIPID9	Hardware Digital ID09	7EFDE9H	32K Power-down wake-up timer frequency (high byte)								nnnn,nnnn
CHIPID10	Hardware Digital ID10	7EFDEAH	32K Power-down wake-up timer frequency (low byte)								nnnn,nnnn
CHIPID11	Hardware Digital ID11	7EFDEBH	IRC parameters for 22.1184MHz (27M band)								nnnn,nnnn
CHIPID12	Hardware Digital ID12	7EFDECH	IRC parameters for 24MHz (27M band)								nnnn,nnnn
CHIPID13	Hardware number ID13	7EFDEDH	IRC parameters for 27MHz (27M band)								nnnn,nnnn
CHIPID14	Hardware number ID14	7EFDEEH	IRC parameters for 30MHz (27M band)								nnnn,nnnn
CHIPID15	Hardware Digital ID15	7EFDEFH	IRC parameters for 33.1776MHz (27M band)								nnnn,nnnn
CHIPID16	Hardware Digital ID16	7EFDF0H	IRC parameters for 35MHz (44M band)								nnnn,nnnn
CHIPID17	Hardware Digital ID17	7EFDF1H	IRC Parameters for 36.864MHz (44M band)								nnnn,nnnn

CHIPID18	Hardware Digital ID18	7EFDF2H	IRC parameters for 40MHz (44M band)	nnnn,nnnn
CHIPID19	Hardware number ID19	7EFDF3H	IRC parameters for 44.2368MHz (44M band)	nnnn,nnnn
CHIPID20	Hardware Digital ID20	7EFDF4H	IRC parameters for 48MHz (44M band)	nnnn,nnnn
CHIPID21	Hardware numbers ID21	7EFDF5H	VRTRIM parameters for the 6M band	nnnn,nnnn
CHIPID22	Hardware numbers ID22	7EFDF6H	VRTRIM parameters for the 10M band	nnnn,nnnn
CHIPID23	Hardware numbers ID23	7EFDF7H	VRTRIM parameters for the 27M band	nnnn,nnnn
CHIPID24	Hardware Digital ID24	7EFDF8H	VRTRIM parameters for the 44M band	nnnn,nnnn
CHIPID25	Hardware numbers ID25	7EFDF9H	00H	nnnn,nnnn
CHIPID26	Hardware numbers ID26	7EFDFAH	End-of-user-program space address (high byte)	nnnn,nnnn
CHIPID27	Hardware numbers ID27	7EFDFBH	Chip testing time (years)	nnnn,nnnn

CHIPID28	Hardware numbers ID28	7EFDFCH	Chip testing time (months)	nnnn,nnnn
CHIPID29	Hardware Digital ID29	7EFDFDH	Chip test time (days)	nnnn,nnnn
CHIPID30	Hardware Digital ID30	7EFDFEH	Chip package form number	nnnn,nnnn
CHIPID31	Hardware numbers ID31	7EFDFFH	5AH	nnnn,nnnn

10.7 Extended Special Function Register List (XFR: 0x7EFB00-0x7EFBFF)

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
HSPWMA_CFG	High-speed PWMA Configuration Register	7EFBF0H	-	-	-	-	AUTORD	INTEN	ASYNCEN	1	xxxx,0001
HSPWMA_ADR	High-speed PWMA address register	7EFBF1H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMA_DAT	High-speed PWMA data register	7EFBF2H	DATA[7:0]								0000,0000
HSPWMB_CFG	High-speed PWMB Configuration Register	7EFBF4H	-	-	-	-	AUTORD	INTEN	ASYNCEN	1	xxxx,0001
HSPWMB_ADR	High-speed PWMB address register	7EFBF5H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMB_DAT	High-speed PWMB data register	7EFBF6H	DATA[7:0]								0000,0000
HSSPI_CFG	High Speed SPI Configuration Register	7EFBF8H	SS_HLD[3:0]				SS_SETUP[3:0]				0011,0011
HSSPI_CFG2	High Speed SPI Configuration Register 2	7EFBF9H	-	-	HSSPIEN	FIFOEN	SS_DACT[3:0]				xx00,0011
HSSPI_STA	High Speed SPI Status Register	7EFBFAH	-	-	-	-	TXFULL	TXEMPT	RXFULL	RXEMPT	xxxx,0000

10.8 Extended Special Function Register List (XFR: 0x7EFA00-0x7EFAFF)

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_M2M_CFG	M2M_DMA Configuration Registers	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_CR	M2M_DMA control registers	7EFA01H	ENM2M	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA Status Register	7EFA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_M2M_AMT	M2M_DMA Total bytes transferred	7EFA03H									0000,0000
DMA_M2M_DONE	M2M_DMA Transmission Completion Bytes	7EFA04H									0000,0000
DMA_M2M_TXAH	M2M_DMA Send High Address	7EFA05H									0000,0000

DMA_M2M_TXAL	M2M_DMA Transmit Low Address	7EFA06H									0000,0000
DMA_M2M_RXAH	M2M_DMA Receive High Address	7EFA07H									0000,0000
DMA_M2M_RXAL	M2M_DMA Receive Low Address	7EFA08H									0000,0000
DMA_ADC_CFG	ADC_DMA Configuration Register	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_CR	ADC_DMA Control Register	7EFA11H	ENADC	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_ADC_STA	ADC_DMA Status Register	7EFA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_ADC_RXAH	ADC_DMA Receive High Address	7EFA17H									0000,0000
DMA_ADC_RXAL	ADC_DMA Receive Low Address	7EFA18H									0000,0000
DMA_ADC_CFG2	ADC_DMA Configuration Register 2	7EFA19H	-	-	-	-	CVTIMESEL [3:0]				xxxx,0000
DMA_ADC_CHSW0	ADC_DMA channel enable	7EFA1AH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0	0000,0001
DMA_ADC_CHSW1	ADC_DMA channel enable	7EFA1BH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	1000,0000
DMA_SPI_CFG	SPI_DMA Configuration Register	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_CR	SPI_DMA Control Register	7EFA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO	000x,xxx0

DMA_SPI_STA	SPI_DMA Status Register	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_SPI_AMT	SPI_DMA Total bytes transferred	7EFA23H									0000,0000
DMA_SPI_DONE	SPI_DMA transfer completion byte count	7EFA24H									0000,0000
DMA_SPI_TXAH	SPI_DMA Send High Address	7EFA25H									0000,0000
DMA_SPI_TXAL	SPI_DMA Send Low Address	7EFA26H									0000,0000
DMA_SPI_RXAH	SPI_DMA Receive High Address	7EFA27H									0000,0000
DMA_SPI_RXAL	SPI_DMA Receive Low Address	7EFA28H									0000,0000
DMA_SPI_CFG2	SPI_DMA Configuration Register 2	7EFA29H	-	-	-	-	-	WRPSS	SSS[1:0]		xxxx,x000
DMA_UR1T_CFG	UR1T_DMA Configuration Registers	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_CR	UR1T_DMA Control Registers	7EFA31H	ENUR1T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA Status Register	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1T_AMT	UR1T_DMA Total bytes transferred	7EFA33H									0000,0000
DMA_UR1T_DONE	UR1T_DMA transfer completion byte count	7EFA34H									0000,0000
DMA_UR1T_TXAH	UR1T_DMA Send High Address	7EFA35H									0000,0000
DMA_UR1T_TXAL	UR1T_DMA Send Low Address	7EFA36H									0000,0000
DMA_UR1R_CFG	UR1R_DMA Configuration Registers	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_CR	UR1R_DMA Control Registers	7EFA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR1R_STA	UR1R_DMA Status Register	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR1R_AMT	UR1R_DMA Total bytes transferred	7EFA3BH									0000,0000
DMA_UR1R_DONE	UR1R_DMA Transmission Completion Byte Count	7EFA3CH									0000,0000
DMA_UR1R_RXAH	UR1R_DMA Receive High Address	7EFA3DH									0000,0000
DMA_UR1R_RXAL	UR1R_DMA Receive Low Address	7EFA3EH									0000,0000
DMA_UR2T_CFG	UR2T_DMA Configuration Registers	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_CR	UR2T_DMA Control Registers	7EFA41H	ENUR2T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA Status Register	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA Total bytes transferred	7EFA43H									0000,0000
DMA_UR2T_DONE	UR2T_DMA Transfer Completion Bytes	7EFA44H									0000,0000
DMA_UR2T_TXAH	UR2T_DMA Send High Address	7EFA45H									0000,0000

DMA_UR2T_TXAL	UR2T_DMA Send Low Address	7EFA46H									0000,0000
DMA_UR2R_CFG	UR2R_DMA Configuration Registers	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_CR	UR2R_DMA Control Registers	7EFA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR2R_STA	UR2R_DMA Status Register	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR2R_AMT	UR2R_DMA Total bytes transferred	7EFA4BH									0000,0000
DMA_UR2R_DONE	UR2R_DMA Transmission Completion Bytes	7EFA4CH									0000,0000
DMA_UR2R_RXAH	UR2R_DMA Receive High Address	7EFA4DH									0000,0000
DMA_UR2R_RXAL	UR2R_DMA Receive Low Address	7EFA4EH									0000,0000
DMA_UR3T_CFG	UR3T_DMA Configuration Registers	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_CR	UR3T_DMA Control Registers	7EFA51H	ENUR3T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA Status Register	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA Total bytes transferred	7EFA53H									0000,0000
DMA_UR3T_DONE	UR3T_DMA Transfer Completion Bytes	7EFA54H									0000,0000
DMA_UR3T_TXAH	UR3T_DMA Send High Address	7EFA55H									0000,0000
DMA_UR3T_TXAL	UR3T_DMA Send Low Address	7EFA56H									0000,0000
DMA_UR3R_CFG	UR3R_DMA Configuration Registers	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000

DMA_UR3R_CR	UR3R_DMA Control Registers	7EFA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR3R_STA	UR3R_DMA Status Register	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR3R_AMT	UR3R_DMA Total bytes transferred	7EFA5BH									0000,0000
DMA_UR3R_DONE	UR3R_DMA Transmission Completion Byte Count	7EFA5CH									0000,0000
DMA_UR3R_RXAH	UR3R_DMA Receive High Address	7EFA5DH									0000,0000
dma_ur3r_rxal	UR3R_DMA Receive Low Address	7EFA5EH									0000,0000
DMA_UR4T_CFG	UR4T_DMA Configuration Registers	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_CR	UR4T_DMA Control Registers	7EFA61H	ENUR4T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA Status Register	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA Total bytes transferred	7EFA63H									0000,0000
DMA_UR4T_DONE	UR4T_DMA Transfer Completion Bytes	7EFA64H									0000,0000
DMA_UR4T_TXAH	UR4T_DMA Send High Address	7EFA65H									0000,0000
DMA_UR4T_TXAL	UR4T_DMA Send Low Address	7EFA66H									0000,0000
DMA_UR4R_CFG	UR4R_DMA Configuration Registers	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_CR	UR4R_DMA Control Registers	7EFA69H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR4R_STA	UR4R_DMA Status Register	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_UR4R_AMT	UR4R_DMA Total bytes transferred	7EFA6BH									0000,0000
DMA_UR4R_DONE	UR4R_DMA Transmission Completion Bytes	7EFA6CH									0000,0000
DMA_UR4R_RXAH	UR4R_DMA Receive High Address	7EFA6DH									0000,0000
DMA_UR4R_RXAL	UR4R_DMA Receive Low Address	7EFA6EH									0000,0000
DMA_LCM_CFG	LCM_DMA Configuration Register	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]		0xxx,0000
DMA_LCM_CR	LCM_DMA Control Register	7EFA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	-	0000,0xxx
DMA_LCM_STA	LCM_DMA Status Register	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_LCM_AMT	LCM_DMA Total bytes transferred	7EFA73H									0000,0000
DMA_LCM_DONE	LCM_DMA transfer completion byte count	7EFA74H									0000,0000
DMA_LCM_TXAH	LCM_DMA Send High Address	7EFA75H									0000,0000
DMA_LCM_TXAL	LCM_DMA Send Low Address	7EFA76H									0000,0000
DMA_LCM_RXAH	LCM_DMA Receive High Address	7EFA77H									0000,0000
DMA_LCM_RXAL	LCM_DMA Receive Low	7EFA78H									0000,0000

	Address			
DMA_M2M_AMTH	M2M_DMA Total bytes transferred	7EFA80H		0000,0000
DMA_M2M_DONEH	M2M_DMA transfer completion byte count	7EFA81H		0000,0000
DMA_SPI_AMTH	SPI_DMA Total bytes transferred	7EFA84H		0000,0000
DMA_SPI_DONEH	SPI_DMA transfer completion byte count	7EFA85H		0000,0000
DMA_LCM_AMTH	LCM_DMA Total bytes transferred	7EFA86H		0000,0000
DMA_LCM_DONEH	LCM_DMA transfer completion byte count	7EFA87H		0000,0000
DMA_UR1T_AMTH	UR1T_DMA Total bytes transferred	7EFA88H		0000,0000
DMA_UR1T_DONEH	UR1T_DMA transfer completion byte count	7EFA89H		0000,0000
DMA_UR1R_AMTH	UR1R_DMA Total bytes transferred	7EFA8AH		0000,0000
DMA_UR1R_DONEH	UR1R_DMA Transmission Completion Byte Count	7EFA8BH		0000,0000
DMA_UR2T_AMTH	UR2T_DMA Total bytes transferred	7EFA8CH		0000,0000
DMA_UR2T_DONEH	UR2T_DMA Transfer Completion Bytes	7EFA8DH		0000,0000
DMA_UR2R_AMTH	UR2R_DMA Total bytes transferred	7EFA8EH		0000,0000
dma_ur2r_doneh	UR2R_DMA Transmission Completion Bytes	7EFA8FH		0000,0000
DMA_UR3T_AMTH	UR3T_DMA Total bytes transferred	7EFA90H		0000,0000

DMA_UR3T_DONEH	UR3T_DMA Transfer Completion Bytes	7EFA91H								0000,0000
DMA_UR3R_AMTH	UR3R_DMA Total bytes transferred	7EFA92H								0000,0000
dma_ur3r_doneh	UR3R_DMA Transmission Completion Byte Count	7EFA93H								0000,0000
DMA_UR4T_AMTH	UR4T_DMA Total bytes transferred	7EFA94H								0000,0000
DMA_UR4T_DONEH	UR4T_DMA Transfer Completion Bytes	7EFA95H								0000,0000
DMA_UR4R_AMTH	UR4R_DMA Total bytes transferred	7EFA96H								0000,0000
DMA_UR4R_DONEH	UR4R_DMA Transmission Completion Bytes	7EFA97H								0000,0000
DMA_I2CT_CFG	I2CT_DMA Configuration Registers	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]	0xxx,0000
DMA_I2CT_CR	I2CT_DMA Control Registers	7EFA99H	ENI2CT	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2CT_STA	I2CT_DMA Status Register	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF	xxxx,x0x0
DMA_I2CT_AMT	I2CT_DMA Total bytes transferred	7EFA9BH								0000,0000
DMA_I2CT_DONE	I2CT_DMA Transfer Completion Bytes	7EFA9CH								0000,0000
DMA_I2CT_TXAH	I2CT_DMA Send High Address	7EFA9DH								0000,0000
DMA_I2CT_TXAL	I2CT_DMA Send Low Address	7EFA9EH								0000,0000
DMA_I2CR_CFG	I2CR_DMA Configuration Registers	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]	0xxx,0000
DMA_I2CR_CR	I2CR_DMA Control Registers	7EFAA1H	ENI2CR	TRIG	-	-	-	-	-	CLRFIFO	00xx,xxx0
DMA_I2CR_STA	I2CR_DMA Status Register	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF	xxxx,xx00
DMA_I2CR_AMT	I2CR_DMA Total bytes transferred	7EFAA3H								0000,0000
DMA_I2CR_DONE	I2CR_DMA Transmission Completion Bytes	7EFAA4H								0000,0000
DMA_I2CR_RXAH	I2CR_DMA Receive High Address	7EFAA5H								0000,0000
DMA_I2CR_RXAL	I2CR_DMA Receive Low Address	7EFAA6H								0000,0000
DMA_I2CT_AMTH	I2CT_DMA Total bytes transferred	7EFAA8H								0000,0000
DMA_I2CT_DONEH	I2CT_DMA Transfer Completion Bytes	7EFAA9H								0000,0000
DMA_I2CR_AMTH	I2CR_DMA Total bytes transferred	7EFAAAH								0000,0000
DMA_I2CR_DONEH	I2CR_DMA Transmission Completion Byte Count	7EFAABH								0000,0000
DMA_I2C_CR	I2C_DMA Control Register	7EFAADH	RDSEL	-	-	-	-	ACKERR	INTEN	BMMEN	0xxx,x000
DMA_I2C_ST1	I2C_DMA Status Register	7EFAAEH	COUNT[7:0]								0000,0000
DMA_I2C_ST2	I2C_DMA Status Register	7EFAAFH	COUNT[15:8]								0000,0000
DMA_I2ST_CFG	I2ST_DMA Configuration	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]		0xxx,0000

	Registers									
DMA_I2ST_CR	I2ST_DMA Control Registers	7EFAB1H	ENI2ST	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2ST_STA	I2ST_DMA Status Register	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF	xxxx,x0x0
DMA_I2ST_AMT	I2ST_DMA Total number of bytes transferred	7EFAB3H									0000,0000
DMA_I2ST_DONE	I2ST_DMA Transfer Completion Byte Count	7EFAB4H									0000,0000
DMA_I2ST_TXAH	I2ST_DMA Send High Address	7EFAB5H									0000,0000
DMA_I2ST_TXAL	I2ST_DMA Send Low Address	7EFAB6H									0000,0000
DMA_I2SR_CFG	I2SR_DMA Configuration Registers	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]		0xxx,0000
DMA_I2SR_CR	I2SR_DMA Control Registers	7EFAB9H	ENI2SR	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_I2SR_STA	I2SR_DMA Status Register	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF	xxxx,xx00
DMA_I2SR_AMT	I2SR_DMA Total bytes transferred	7EFABBH									0000,0000
DMA_I2SR_DONE	I2SR_DMA Transfer Completion Byte Count	7EFABCH									0000,0000
DMA_I2SR_RXAH	I2SR_DMA Receive High Address	7EFABDH									0000,0000
DMA_I2SR_RXAL	I2SR_DMA Receive Low Address	7EFABEH									0000,0000
DMA_I2ST_AMTH	I2ST_DMA Total bytes transferred	7EFAC0H									0000,0000
DMA_I2ST_DONEH	I2ST_DMA Transfer Completion Byte Count	7EFAC1H									0000,0000

DMA_I2SR_AMTH	I2SR_DMA Total bytes transferred	7EFAC2H									0000,0000
DMA_I2SR_DONEH	I2SR_DMA Transfer Completion Byte Count	7EFAC3H									0000,0000
DMA_ARB_CFG	DMA President Configuration Register	7EFAF8H	WTRREN	-	-	-	STASEL[3:0]-				0xxx,0000
DMA_ARB_STA	DMA President Status Register	7EFAF9H									0000,0000

11 I/O Port

product lines	Maximum number of I/O ports
STC32G12K128 Series	**60**
STC32G8K64 Series	**45**
STC32F12K60 Series	**45**

All I/O ports of STC32G series microcontrollers have four working modes: quasi-bidirectional port/weak pull-up (standard 8051 output port mode), push-pull output/strong pull-up, high-resistance input (current can neither flow in nor out), and open-drain output. The operating modes of the I/O ports can be easily configured by software.

Notes on I/O:

1. The state of P3.0 and P3.1 ports is weak pull-up/quasi-bidirectional port mode after power-up

Except P3.0 and P3.1, all other IO ports are in high resistance input state after power on, users must set the IO port mode before using the IO port.

3、If the chip does not need to use USB for ISP download when powering up, the three I/O ports P3.0/P3.1/P3.2 should not be low at the same time, otherwise it will enter the USB download mode and can't run the user code.

4、When the chip is powered on, if P3.0 and P3.1 are low at the same time, the P3.2 port will be switched from high-resistance input to bidirectional mode for a short time, which can be used to read the external state of the P3.2 port to judge whether it is necessary to enter the USB download mode.

5、When using P5.4 as reset pin, the 4K pull-up resistor inside the port will be open all the time; however, when P5.4 is used as a normal I/O port, based on the special consideration that this I/O port shares the same pin with the reset pin, the 4K pull-up resistor inside the port will still be open for about 6.5 milliseconds, and then turn off again automatically (when the user's circuit design needs to use the P5.4 to drive the external circuits, please (When using the P5.4 port to drive external circuits, please make sure to consider the 6.5ms high level at power-on).

11.1 I/O Port Related Registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 port	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111

P1	P1 port	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	P2 port	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	P3 port	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P4	P4 port	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
P5	P5 port	C8H	-	-	P55	P54	P53	P52	P51	P50	xx11,1111
P6	P6 port	E8H	P67	P66	P65	P64	P63	P62	P61	P60	1111,1111
P7	P7 port	F8H	P77	P76	P75	P74	P73	P72	P71	P70	1111,1111
P0M0	P0 Port Configuration Register 0	94H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	0000,0000
P0M1	P0 Port Configuration Register 1	93H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	1111,1111
P1M0	P1 Port Configuration Register 0	92H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	0000,0000

P1M1	P1 Port Configuration Register 1	91H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	1111,1111
P2M0	P2 Port Configuration Register 0	96H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	0000,0000
P2M1	P2 Port Configuration Register 1	95H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	1111,1111
P3M0	P3 Port Configuration Register 0	B2H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	0000,0000
P3M1	P3 Port Configuration Register 1	B1H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	1111,1100
P4M0	P4 Port Configuration Register 0	B4H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	0000,0000
P4M1	P4 Port Configuration Register 1	B3H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	1111,1111
P5M0	P5 Port Configuration Register 0	CAH	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1	xx00,0000
P5M1	P5 Port Configuration Register 1	C9H	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0	xx11,1111
P6M0	P6 Port Configuration Register 0	CCH	P67M1	P66M1	P65M1	P64M1	P63M1	P62M1	P61M1	P60M1	0000,0000
P6M1	P6 Port Configuration Register 1	CBH	P67M0	P66M0	P65M0	P64M0	P63M0	P62M0	P61M0	P60M0	1111,1111
P7M0	P7 Port Configuration Register 0	E2H	P77M1	P76M1	P75M1	P74M1	P73M1	P72M1	P71M1	P70M1	0000,0000
P7M1	P7 Port Configuration Register 1	E1H	P77M0	P76M0	P75M0	P74M0	P73M0	P72M0	P71M0	P70M0	1111,1111

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0PU	P0 Port Pull-up Resistor Control Register	7EFE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 Port Pull-Up Resistor Control Register	7EFE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 Port Pull-Up Resistor Control Register	7EFE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 Port Pull-up Resistor Control Register	7EFE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	P4 Port Pull-Up Resistor Control Register	7EFE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	P5 Port Pull-Up Resistor Control Register	7EFE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU	xx00,0000
P6PU	P6 Port Pull-Up Resistor Control Register	7EFE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU	0000,0000
P7PU	P7 Port Pull-Up Resistor Control Register	7EFE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU	0000,0000
P0NCS	P0 Port Schmitt Trigger Control Register	7EFE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 Port Schmitt Trigger Control Register	7EFE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 Port Schmitt Trigger Control Register	7EFE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 Port Schmitt Trigger	7EFE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000

	Control Register										
P4NCS	P4 Port Schmitt Trigger Control Register	7EFE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	P5 Port Schmitt Trigger Control Register	7EFE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS	xx00,0000
P6NCS	P6 Port Schmitt Trigger Control Register	7EFE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS	0000,0000
P7NCS	P7 Port Schmitt Trigger Control Register	7EFE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS	0000,0000
P0SR	P0 Port Level Shift Rate Register	7EFE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	P1 Port Level Shift Rate Register	7EFE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	P2 Port Level Shift Rate Register	7EFE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	P3 Port Level Shift Rate Register	7EFE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	P4 Port Level Shift Rate Register	7EFE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	P5 Port Level Shift Rate Register	7EFE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR	xx11,1111
P6SR	P6 Port Level Shift Rate Register	7EFE26H	P67SR	P66SR	P65SR	P64SR	P63SR	P62SR	P61SR	P60SR	1111,1111
P7SR	P7 Port Level Shift Rate Register	7EFE27H	P77SR	P76SR	P75SR	P74SR	P73SR	P72SR	P71SR	P70SR	1111,1111
P0DR	P0 Port Drive Current Control Register	7EFE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 Port Drive Current Control Register	7EFE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 Port Drive Current Control Register	7EFE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 Port Drive Current Control Register	7EFE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111

P4DR	P4 Port Drive Current Control Register	7EFE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111
P5DR	P5 Port Drive Current Control Register	7EFE2DH	-	-	P5DR	P54DR	P53DR	P52DR	P51DR	P50DR	xx11,1111
P6DR	P6 Port Drive Current Control Register	7EFE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR	1111,1111
P7DR	P7 Port Drive Current Control Register	7EFE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR	1111,1111
P0IE	P0 Port Input Enable Control Register	7EFE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE	1111,1111
P1IE	P1 Port Input Enable Control Register	7EFE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111
P2IE	P2 Port Input Enable Control Register	7EFE32H	P27IE	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE	1111,1111
P3IE	P3 Port Input Enable Control Register	7EFE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE	1111,1111
P4IE	P4 Port Input Enable Control Register	7EFE34H	P47IE	P46IE	P45IE	P44IE	P43IE	P42IE	P41IE	P40IE	1111,1111
P5IE	P5 Port Input Enable Control Register	7EFE35H	-	-	P55IE	P54IE	P53IE	P52IE	P51IE	P50IE	xx11,1111
P6IE	P6 Port Input Enable Control Register	7EFE36H	P67IE	P66IE	P65IE	P64IE	P63IE	P62IE	P61IE	P60IE	1111,1111
P7IE	P7 Port Input Enable Control Register	7EFE37H	P77IE	P76IE	P75IE	P74IE	P73IE	P72IE	P71IE	P70IE	1111,1111
P0PD	P0 Port Pull Down Resistor Control Register	7EFE40H	P07PD	P06PD	P05PD	P04PD	P03PD	P02PD	P11PD	P00PD	0000,0000
P1PD	P1 Port Pull Down Resistor Control Register	7EFE41H	P17PD	P16PD	P15PD	P14PD	P13PD	P12PD	P11PD	P10PD	0000,0000
P2PD	P2 Port Pull Down Resistor Control Register	7EFE42H	P27PD	P26PD	P25PD	P24PD	P23PD	P22PD	P11PD	P20PD	0000,0000
P3PD	P3 Port Pull Down Resistor Control Register	7EFE43H	P37PD	P36PD	P35PD	P34PD	P33PD	P32PD	P11PD	P30PD	0000,0000
P4PD	P4 Port Pull Down Resistor Control Register	7EFE44H	P47PD	P46PD	P45PD	P44PD	P43PD	P42PD	P11PD	P40PD	0000,0000
P5PD	P5 Port Pull Down Resistor Control Register	7EFE45H	-	-	P55PD	P54PD	P53PD	P52PD	P11PD	P50PD	xx00,0000
P6PD	P6 Port Pull Down Resistor Control Register	7EFE46H	P67PD	P66PD	P65PD	P64PD	P63PD	P62PD	P11PD	P60PD	0000,0000
P7PD	P7 Port Pull-Down Resistor Control Register	7EFE47H	P77PD	P76PD	P75PD	P74PD	P73PD	P72PD	P11PD	P70PD	0000,0000

11.1.1 Port Data Register (Px)

notation	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 level to port buffer Write 1: Output high level to port buffer Read: Read the level on the port pin directly

11.1.2 Port mode configuration registers (PxM0, PxM1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0M0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0
P0M1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1
P1M0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0
P1M1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1
P2M0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0
P2M1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1
P3M0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0
P3M1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1
P4M0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0
P4M1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1
P5M0	CAH	-	-	P55M0	P54M0	P53M0	P52M0	P51M0	P50M0
P5M1	C9H	-	-	P55M1	P54M1	P53M1	P52M1	P51M1	P50M1
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 port

PnM1.x	PnM0.x	Pn.x port operating mode
0	0	quasi-bidirectional port
0	1	push-pull output
1	0	High Resistance Input
1	1	open-drain output

11.1.3 Port pull-up resistor control register (PxPU)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	7EFE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	7EFE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	7EFE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	7EFE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	7EFE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	7EFE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	7EFE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	7EFE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

Port internal 4.1K pull-up resistor control bit (note: pull-up resistors on P3.0 and P3.1 ports may be slightly smaller)

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

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

11.1.4 Port Schmitt Trigger Control Register (PxNCS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	7EFE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS
P1NCS	7EFE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS
P2NCS	7EFE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS
P3NCS	7EFE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS
P4NCS	7EFE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS
P5NCS	7EFE1DH	-	-	P55NCS	P54NCS	P53NCS	P52NCS	P51NCS	P50NCS
P6NCS	7EFE1EH	P67NCS	P66NCS	P65NCS	P64NCS	P63NCS	P62NCS	P61NCS	P60NCS
P7NCS	7EFE1FH	P77NCS	P76NCS	P75NCS	P74NCS	P73NCS	P72NCS	P71NCS	P70NCS

Port Schmitt Trigger Control Bit

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

1: **Disable the** Schmitt trigger function of the port.

11.1.5 Port Level Shift Speed Control Register (PxSR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0SR	7EFE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR
P1SR	7EFE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR
P2SR	7EFE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR
P3SR	7EFE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR
P4SR	7EFE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR
P5SR	7EFE25H	-	-	P55SR	P54SR	P53SR	P52SR	P51SR	P50SR
P6SR	7EFE26H	P57SR	P66SR	P65SR	P64SR	P63SR	P62SR	P61SR	P60SR
P7SR	7EFE27H	P77SR	P76SR	P75SR	P74SR	P73SR	P72SR	P71SR	P70SR

Controls the speed of port level transitions

0: Fast level shifting, corresponding up and down strokes will be larger

1: Slow level conversion speed, the corresponding up and down stroke is relatively small

11.1.6 Port Drive Current Control Register (PxDR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0DR	7EFE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR
P1DR	7EFE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR
P2DR	7EFE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR
P3DR	7EFE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR
P4DR	7EFE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR
P5DR	7EFE2DH	-	-	P55DR	P54DR	P53DR	P52DR	P51DR	P50DR
P6DR	7EFE2EH	P67DR	P66DR	P65DR	P64DR	P63DR	P62DR	P61DR	P60DR
P7DR	7EFE2FH	P77DR	P76DR	P75DR	P74DR	P73DR	P72DR	P71DR	P70DR

Control port drive capability

0: Enhanced drive capacity

1: General drive capacity

11.1.7 Port Digital Signal Input Enable Control Register (PxIE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0IE	7EFE30H	P07IE	P06IE	P05IE	P04IE	P03IE	P02IE	P11IE	P00IE
P1IE	7EFE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE
P2IE	7EFE32H	P27IE	P26IE	P25IE	P24IE	P23IE	P22IE	P21IE	P20IE
P3IE	7EFE33H	P37IE	P36IE	P35IE	P34IE	P33IE	P32IE	P31IE	P30IE
P4IE	7EFE34H	P47IE	P46IE	P45IE	P44IE	P43IE	P42IE	P41IE	P40IE
P5IE	7EFE35H	-	-	P55IE	P54IE	P53IE	P52IE	P51IE	P50IE
P6IE	7EFE36H	P67IE	P66IE	P65IE	P64IE	P63IE	P62IE	P61IE	P60IE
P7IE	7EFE37H	P77IE	P76IE	P75IE	P74IE	P73IE	P72IE	P71IE	P70IE

Digital signal input enable control

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

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

11.1.8 Port pull-down resistor control register (PxPD)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0PD	7EFE40H	P07PD	P06PD	P05PD	P04PD	P03PD	P02PD	P01PD	P00PD
P1PD	7EFE41H	P17PD	P16PD	P15PD	P14PD	P13PD	P12PD	P11PD	P10PD
P2PD	7EFE42H	P27PD	P26PD	P25PD	P24PD	P23PD	P22PD	P21PD	P20PD
P3PD	7EFE43H	P37PD	P36PD	P35PD	P34PD	P33PD	P32PD	P31PD	P30PD
P4PD	7EFE44H	P47PD	P46PD	P45PD	P44PD	P43PD	P42PD	P41PD	P40PD
P5PD	7EFE45H	-	-	P55PD	P54PD	P53PD	P52PD	P51PD	P50PD

P6PD	7EFE46H	P67PD	P66PD	P65PD	P64PD	P63PD	P62PD	P61PD	P60PD
P7PD	7EFE47H	P77PD	P76PD	P75PD	P74PD	P73PD	P72PD	P71PD	P70PD

Port internal 10K pull-down resistor control bit

0: Disable the internal pull-down resistor of the port

1: Enable the internal pull-down resistor of the port

Note: STC32G12K128-Beta series does not have this function

11.2 Configuring I/O Ports

The configuration of each I/O needs to be set using two registers.

Taking the P0 port as an example, to configure the P0 port, you need to use the P0M0 and P0M1 registers for configuration, as shown in the following figure:

That is, bit 0 of P0M0 and bit 0 of P0M1 are combined to configure the mode of the P0.0 port that is, bit 1 of P0M0 and bit 1 of P0M1 are combined to configure the mode of the P0.1 port All other I/O configurations are similar to this.

The combination of PnM0 and PnM1 is shown in the table below.

PnM1	PnM0	I/O Port Operating Modes
0	0	Quasi-bidirectional port (conventional 8051 port mode, weak pull-up) Pour current up to 20mA, pull current 270-150μA (manufacturing error exists)
0	1	Push-pull output (strong pull-up output, up to 20mA, with current limiting resistor)
1	0	High resistance input (current can neither flow in nor out)
1	1	Open-Drain output with internal pull-up resistor disconnected The open-drain mode can read the external state as well as output externally (high or low). If you want to read the external state correctly or need to output a high level externally, you need to add an external pull-up resistor, otherwise you can't read the external state and can't output a high level externally.

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

Attention:

Although each I/O port can withstand a 20mA sink current in weak pull-up (quasi-bidirectional port)/strong push-pull output/open-drain mode (still need to add current-limiting resistors, such as 1K, 560Ω, 472Ω, etc.), and a 20mA pull current in strong push-pull output (also need to add current-limiting resistors), it is recommended that the whole chip's operating current should not exceed 90mA, i.e., the current flowing into the device from VCC should not exceed 90mA, and the overall inflow/outflow currents are recommended not to exceed 90mA. However, the whole chip is recommended not to exceed 90mA, i.e. the current flowing in from VCC is recommended not to exceed 90mA, the current flowing out from GND is recommended not to exceed 90mA, and the overall inflow/outflow current is recommended not to exceed 90mA.

11.3 Structure of I/O

11.3.1 Quasi-bidirectional port (weak pull-up)

The quasi-bidirectional port (weak pull-up) output type can be used as an output and input function without reconfiguring the port output state. This is because the port output is weakly driven when it is 1, allowing an external device to pull it low. When the pin output is low, it has strong drive capability and can absorb the

The quasi-bidirectional port has 3 pull-up transistors to suit different needs. The quasi-bidirectional port has 3 pull-up transistors to suit different needs.

One of the three pull-up transistors, called the "weak pull-up", turns on when the port register is 1 and the pin itself is 1. This pull-up provides the basic drive current to make the quasi-bidirectional port output 1. This pull-up provides the basic drive current to make the quasi-bidirectional port output 1. If a pin outputs 1 and is pulled low by an external device, the weak pull-up is turned off and the "very weak pull-up" is kept on, in order to pull this pin strongly low, the external device must have enough current sinking capability to bring the voltage on the pin down to below the threshold voltage. For a 5V microcontroller, the current of the "weak pull-up" transistor is about 250uA; for a 3.3V microcontroller, the current of the "weak pull-up" transistor is about 150uA.

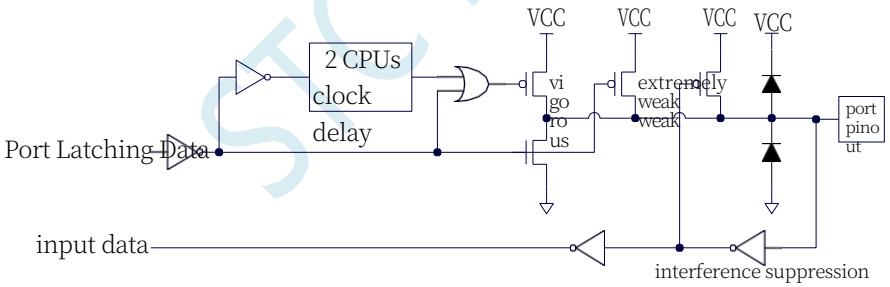
The second pull-up transistor, called the "very weak pull-up", turns on when the port is latched to 1. This very weak pull-up source generates a very weak pull-up current to pull the pin high when the pin is idle. For a 5V microcontroller, the current of the "very weak pull-up" transistor is about 18uA; for a 3.3V microcontroller, the current of the "very weak pull-up" transistor is about 5uA.

The third pull-up transistor is called the "strong pull-up". This pull-up is used to accelerate the transition from a quasi-bidirectional port to a quasi-bidirectional port when the port latch jumps from 0 to 1.

Logic 0 to logic 1 transition. When this occurs, the strong pull-up is turned on for about 2 clocks to allow the pin to be quickly pulled up to a high level. The quasi-bidirectional port (weak pull-up) has a Schmitt trigger input and an interference suppression circuit. The quasi-bidirectional port (weak pull-up) reads external

The state is latched to '1' before the correct external state can be read.

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

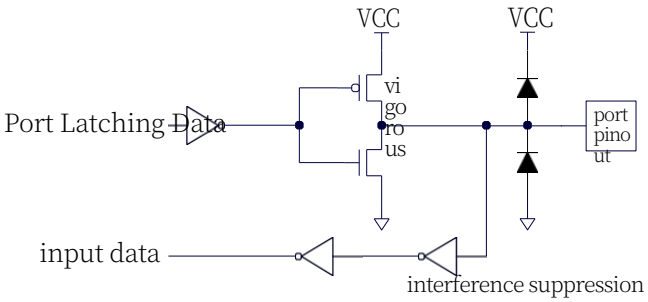


11.3.2 push-pull output

The pull-down structure of the strong push-pull output configuration is the same as that of the open-drain outputs as well as the quasi-bidirectional port, but provides a continuous

strong pull-up when the latch is 1. Push-pull mode is generally used where higher drive currents are required.

The strong push-pull pinout is shown below:

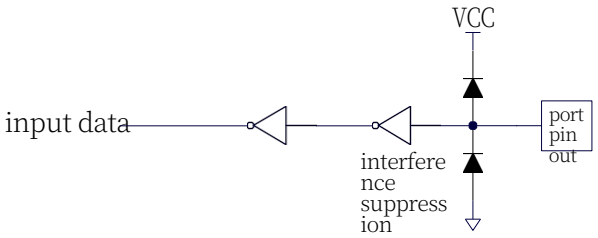


11.3.3 High Resistance Input

Current can neither flow in nor out

Inputs with a Schmitt trigger input and an interference suppression circuitry

The high resistance input pinout is shown below:



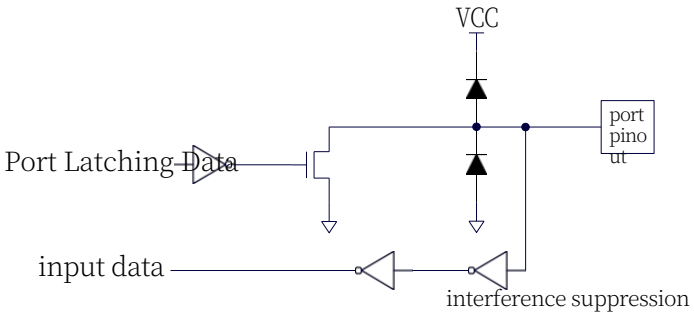
11.3.4 open-drain output

The open-drain mode allows both external status reading and external output (high or low). An external pull-up resistor is required to read the external state correctly or to output a high level externally.

When the port latch is 0, the open-drain output turns off all pull-up transistors. When used as a logic output high, this configuration must have an external pull-up, typically externalised through a resistor to VCC. if an external pull-up resistor is available, the open-drain I/O port can also read the external state, i.e., the I/O port that is configured for open-drain mode at this time can also be used as an input I/O port. The pull-down in this way is the same as for a quasi-bidirectional port.

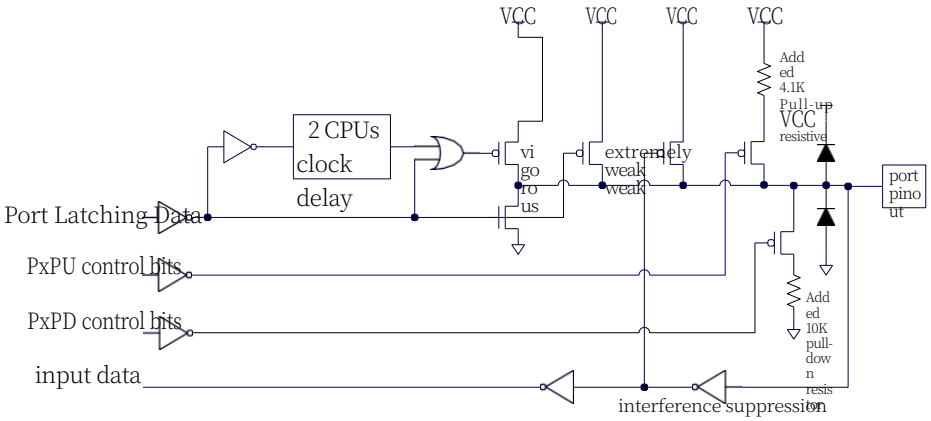
The open-drain port has a Schmitt trigger input and an interference suppression circuit.

The output port configuration is shown below:



11.3.5 Added 4.1K pull-up resistor and 10K pull-down resistor.

All I/O ports of the STC32G series internally enable a pull-up resistor of approximately 4.1K (due to manufacturing error, the pull-up resistor may range from 3K to 5K) and a pull-down resistor of approximately 10K (due to manufacturing error, the pull-down resistor may range from 8K to 12K).



Port pull-up resistor control register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	7EFE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	7EFE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	7EFE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	7EFE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	7EFE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	7EFE15H	-	-	P55PU	P54PU	P53PU	P52PU	P51PU	P50PU
P6PU	7EFE16H	P67PU	P66PU	P65PU	P64PU	P63PU	P62PU	P61PU	P60PU
P7PU	7EFE17H	P77PU	P76PU	P75PU	P74PU	P73PU	P72PU	P71PU	P70PU

Port internal 4.1K pull-up resistor control bit (note: pull-up resistors on P3.0 and P3.1 ports may be slightly smaller)

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

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

Port pull-down resistor control register (PxPD)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0PD	7EFE40H	P07PD	P06PD	P05PD	P04PD	P03PD	P02PD	P01PD	P00PD
P1PD	7EFE41H	P17PD	P16PD	P15PD	P14PD	P13PD	P12PD	P11PD	P10PD
P2PD	7EFE42H	P27PD	P26PD	P25PD	P24PD	P23PD	P22PD	P21PD	P20PD
P3PD	7EFE43H	P37PD	P36PD	P35PD	P34PD	P33PD	P32PD	P31PD	P30PD
P4PD	7EFE44H	P47PD	P46PD	P45PD	P44PD	P43PD	P42PD	P41PD	P40PD
P5PD	7EFE45H	-	-	P55PD	P54PD	P53PD	P52PD	P51PD	P50PD

P6PD	7EFE46H	P67PD	P66PD	P65PD	P64PD	P63PD	P62PD	P61PD	P60PD
P7PD	7EFE47H	P77PD	P76PD	P75PD	P74PD	P73PD	P72PD	P71PD	P70PD

Port internal 10K pull-down resistor control bit

0: Disable the internal pull-down resistor of the port

1: Enable the internal pull-down resistor of the port

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

When the user needs the I/O port to output a faster frequency externally, the external output speed of the I/O port can be increased by increasing the I/O port drive current and increasing the I/O port level conversion speed.

Setting the PxSR register can be used to control the I/O port level transition speed, when set to 0 the corresponding I/O port is fast flip-flop, set to

1 is a slow turnover.

Setting the PxDR register can be used to control the size of the I/O port drive current, when set to 1, the I/O output is normal drive current, when set to 0, it is strong drive current.

11.3.7 How to set the I/O port current drive capability

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

Setting the PxDR register can be used to control the size of the I/O port drive current, when set to 1, the I/O output is normal drive current, when set to 0, it is strong drive current.

11.3.8 How to Reduce External Radiation from I/O Ports

Setting the PxSR register can be used to control the I/O port level transition speed, and setting the PxDR register can be used to control the I/O port drive current size.

When it is necessary to reduce the external radiation of the I/O port, it is necessary to set the PxSR register to 1 to reduce the I/O port level conversion speed, and at the same time, it is necessary to set the PxDR register to 1 to reduce the I/O drive current, so as to ultimately achieve the reduction of the external radiation of the I/O port.

11.4 sample procedure

11.4.1 Port Mode Setting (for all I/O)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     //see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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 resistance input mode
    P2M1 = 0xff.

    P3M0 = 0xcc.                        //Set P3.0~P3.1 as bidirectional port mode
P3M1 = 0xf0.                            //Set P3.2~P3.3 as push-pull output
                                                mode
                                    //Set P3.4~P3.5 to high resistance input mode.
                                        //Set P3.6~P3.7 to open-drain mode.

    while (1);
}
```

11.4.2 Bidirectional port read/write operations (for all I/O)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     //see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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 to bidirectional port mode.
    p0m1 = 0x00.

    P0 = 0xff.                      //All /P0 ports output high
    _nop_().                        //
    _nop_();                        //
    P0 = 0x00.                      /P0 port all output low level
    _nop_().                        //
    _nop_().                        //

    P00 = 1;                        //P0.0 port output high level
    _nop_().                        //
    _nop_().                        //
    P00 = 0;                        //P0.0 port output low level
    _nop_().                        //
    _nop_().                        //

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

    while (1);
}
```

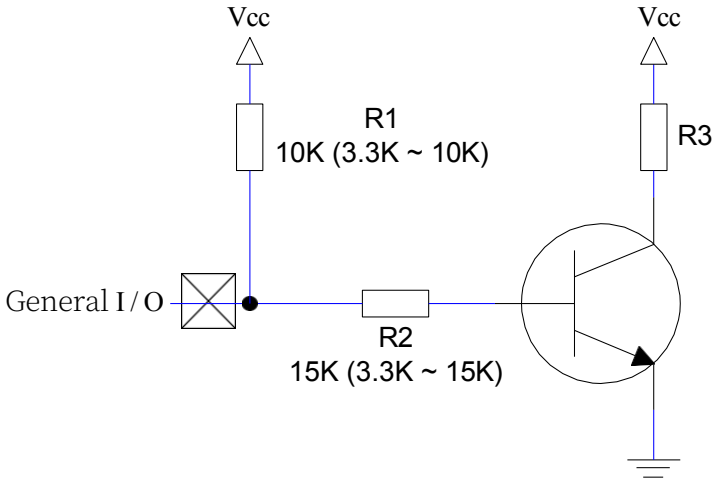
11.4.3 Open the I/O port internal pull-up resistor (applies to all I/O)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 //see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
```

```
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00.
```

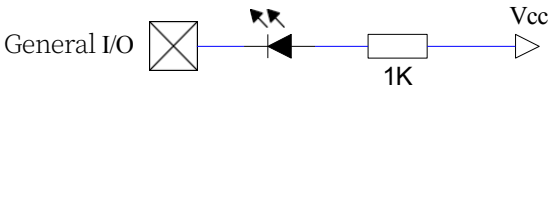
```
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    P0PU = 0x0f.                    //Turn on the internal pull-up resistor of port P0.0~P0.3.

    P1PU = 0xf0.                    //Turn on the internal pull-up resistor of P1.4~P1.7 ports

    while (1);
}
```

11.5 A typical triode control circuit

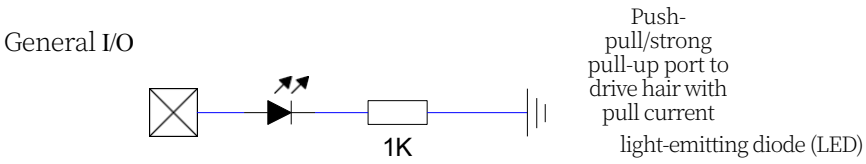


If weak pull-up control is used, it is recommended to add pull-up resistor R1 (3.3K to 10K). If pull-up resistor R1 (3.3K to 10K) is not added, it is recommended that the value of R2 is above 15K or a strong push-pull output is used.

11.6 Typical Light Emitting Diode Control Circuit

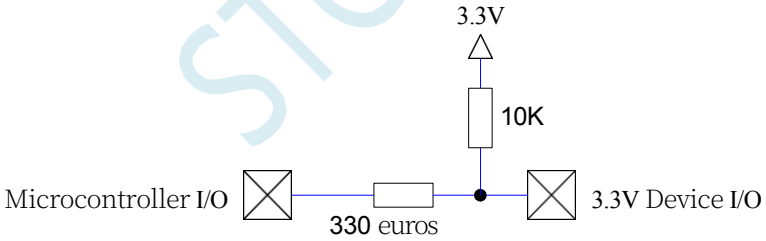


(Current limiting resistors should be greater than 1K as much as possible, not less than 470 ohms)



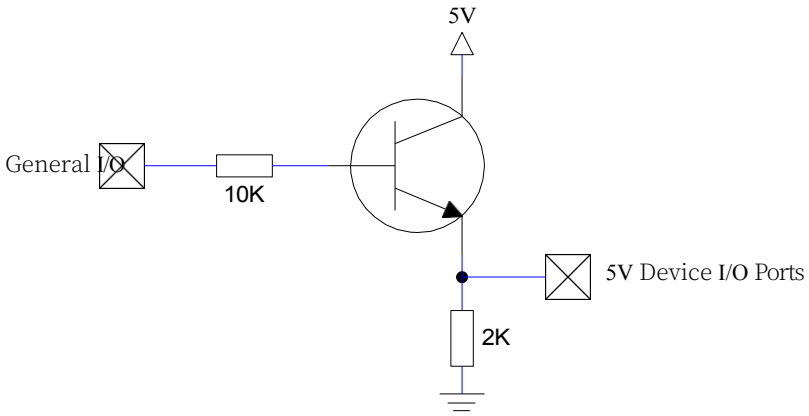
11.7 Mixed Voltage Power Systems 3V/5V Device I/O Port Interconnections

When STC series wide-voltage microcontrollers operate at 5V, if you need to connect a 3.3V device directly, in order to prevent the 3.3V device from not being able to withstand 5V, you can string a 330Ω current-limiting resistor from the I/O port of the corresponding microcontroller to the I/O port of the 3.3V device first, and then initialize the program to set the I/O port of the microcontroller to an open-drain configuration, disconnect the internal pull-up resistor, and add a 10K pull-up resistor outside the corresponding I/O port of the 3.3V device to Vcc of the 3.3V device so that the high level is 3.3V and the low level is 0V. Add a 10K pull-up resistor to the Vcc of the 3.3V device so that the high level is 3.3V and the low level is 0V, and all inputs and outputs are normal.



When the STC wide-voltage microcontroller works at 3V, if you need to connect a 5V device directly, you can connect an isolation diode in series with the corresponding I/O port to isolate the high-voltage part if the I/O port is an input. When the external signal voltage is higher than the operating voltage of the microcontroller, the I/O port is pulled up to a high level internally, so the state of the read I/O port is high; when the external signal voltage is low, it is on, and the I/O port is clamped at 0.7V, and the state of the microcontroller's read I/O port is low when the external signal voltage is less than 0.8V.

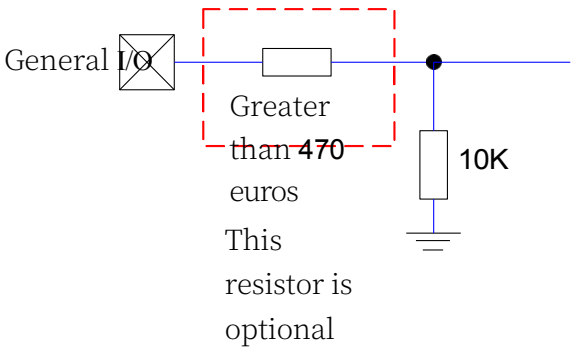
When the STC wide voltage microcontroller operates at 3V, if it is necessary to connect a 5V device directly, if the corresponding I/O port is an output, it can be isolated by an NPN transistor with the following circuit:



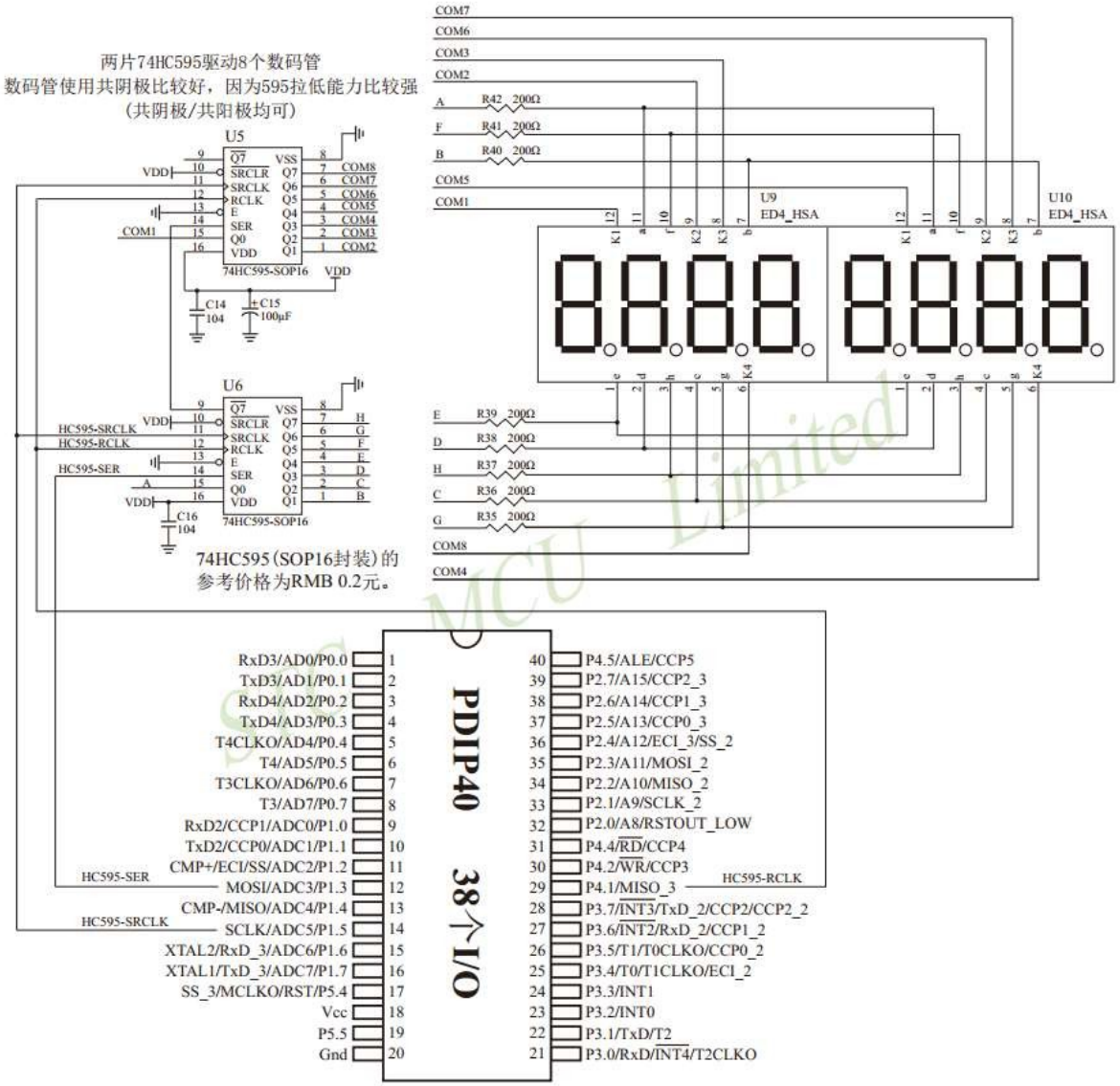
11.8 How to make the I/O port go low on power-on reset

Ordinary 8051 microcontroller power-on reset common I / O port for the weak pull-up (quasi-bidirectional port) high output, and many practical applications require power-on some I / O port for the low output, otherwise the control of the system (such as motors) will be inaccurate action, the STC microcontroller due to both the weak pull-up output and a strong push-pull output, it can be very easy to solve this problem.

Now we can add a pull-down resistor (about 10K) to the I/O port of STC microcontroller, so that at power-on reset, except for the download ports P3.0 and P3.1 which are weak pull-ups (quasi-bidirectional), all other I/O ports are in high-resistance input mode, and there is a pull-down resistor externally, so that the I/O port is low externally at power-on reset. 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 I/O port drive current can be up to 20mA, so you can definitely drive this port to a high level output.



11.9 The use of 74HC595 drive 8 digital tube (serial



expansion, 3 wires) of the circuit diagram

12 interruption system

The interrupt system is set up to provide the CPU with the ability to handle external emergencies in real time.

When the CPU of the central processing machine is dealing with something, an urgent event occurs in the outside world, which requires the CPU to suspend the current work and deal with the urgent event, and then return to the original place where it was interrupted to continue the original work after dealing with the event, and this process is called an interrupt. This process is called interrupt. The component that realises this function is called the interrupt system, and the source of the request for CPU interrupt is called the interrupt source. The interrupt system of microcomputer generally allows more than one interrupt source, and when several interrupt sources request interrupt from the CPU at the same time to serve it, there exists the problem of which interrupt source request the CPU responds to in priority. The CPU always responds to the interrupt request with the highest priority first.

While the CPU is processing an interrupt source request (executing the corresponding interrupt service routine), another interrupt source request occurs with a higher priority than it. If the CPU is able to pause the service procedure for the original interrupt source, switch to the interrupt request source with a higher priority, and then return to the original low-level interrupt service procedure after processing, such a process is called interrupt nesting. Such an interrupt system is called a multilevel interrupt system, and an interrupt system without interrupt nesting is called a single-level interrupt system.

The user can block the corresponding interrupt request by switching off the general interrupt allow bit (EA/IE.7) or the allow bit of the corresponding interrupt, or turn on the corresponding interrupt allow bit to make the CPU respond to the corresponding interrupt request. Each interrupt source can be independently controlled by software to be in the on interrupt or off interrupt state, and the priority level of some of the interrupts can be set by software. A high priority interrupt request can interrupt a low priority interrupt, and vice versa, a low priority interrupt request cannot interrupt a high priority interrupt. When two interrupts of the same priority are generated at the same time, the query order will determine which interrupt the system responds to first.

12.1 STC32G Series Interrupt Sources

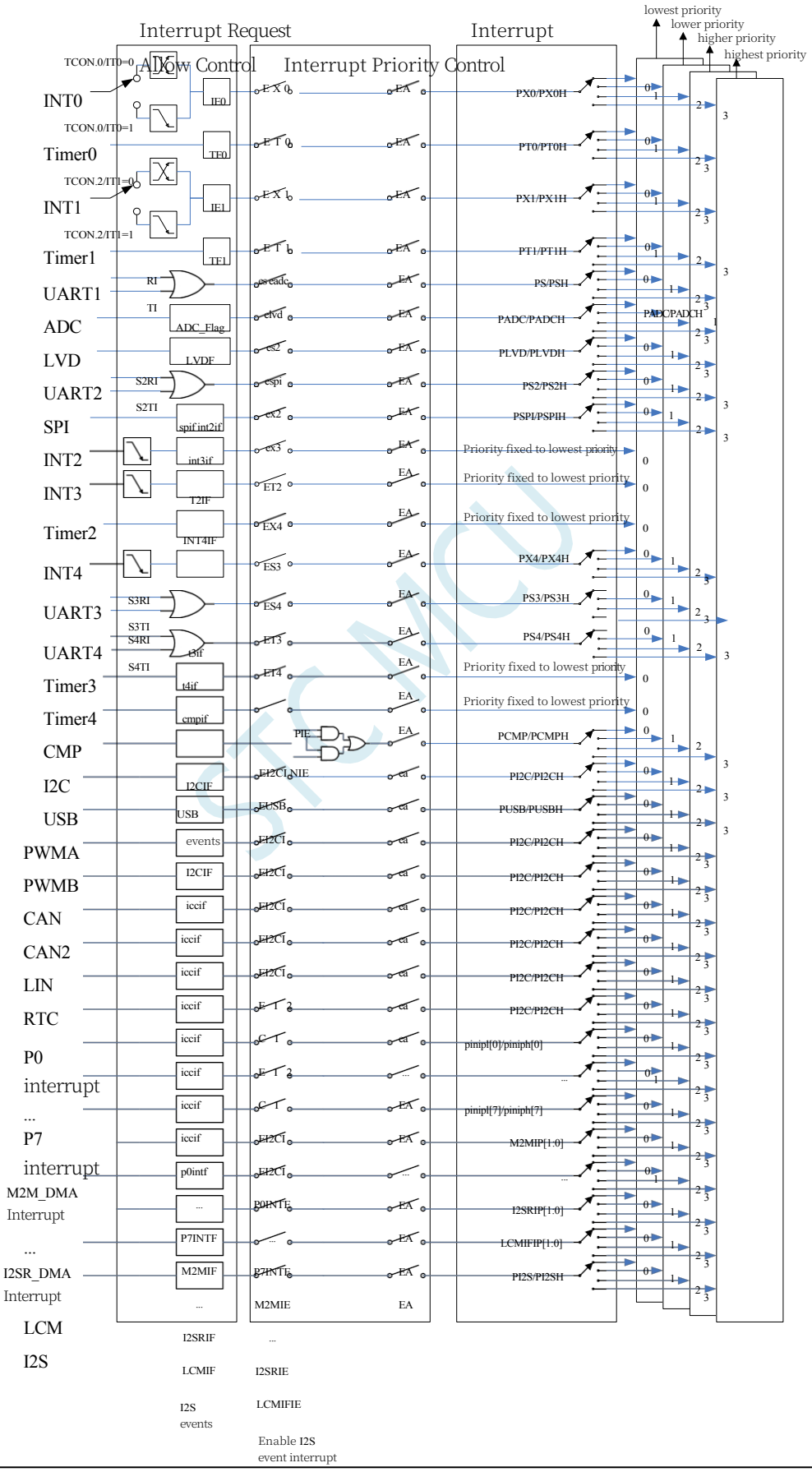
source of interruption	STC32G12K128 Series	STC32G8K64 Series	STC32F12K60 Series
External Interrupt 0 Interrupt (INT0)	√	√	√
Timer 0 Interrupt (Timer0)	√	√	√
External Interrupt 1 Interrupt (INT1)	√	√	√
Timer 1 interrupt (Timer1)	√	√	√
Serial port 1 interrupt (UART1)	√	√	√
Analogue to Digital	√	√	√

Conversion Interrupt (ADC)			
Low Voltage Detection Interrupt (LVD)	√	√	√
Serial 2 Interrupt (UART2)	√	√	√
Serial Peripheral Interface Interrupt (SPI)	√	√	√
External Interrupt 2 Interrupt (INT2)	√	√	√
External Interrupt 3 Interrupt (INT3)	√	√	√
Timer 2 Interrupt (Timer2)	√	√	√
External Interrupt 4 Interrupt (INT4)	√	√	√
Serial port 3 interrupt (UART3)	√	√	√
Serial 4 Interrupt (UART4)	√	√	√
Timer 3 Interrupt (Timer3)	√	√	√

Timer 4 Interrupt (Timer4)	√	√	√
Comparator Interrupt (CMP)	√	√	√
I2C Bus Interrupt	√	√	√
USB interrupt	√		√
PWMA	√	√	√
PWMB	√	√	√
CAN interrupt	√	√	√
CAN2 Interrupt	√	√	√
LIN Interrupt	√	√	√
RTC interrupt	√	√	√
P0 port interrupt	√	√	√
P1 port interrupt	√	√	√
P2 port interrupt	√	√	√
P3 port interrupt	√	√	√
P4 port interrupt	√	√	√
P5 port interrupt	√	√	√
P6 port interrupt	√		
P7 port interrupt	√		
M2M_DMA Interrupt	√	√	√
ADC_DMA Interrupt	√	√	√
SPI_DMA Interrupt	√	√	√
Serial port 1 sends DMA interrupt	√	√	√
Serial port 1 receive DMA interrupt	√	√	√
Serial port 2 sends DMA interrupt	√	√	√
Serial port 2 receive DMA interrupt	√	√	√
Serial port 3 sends DMA interrupt	√	√	√
Serial port 3 receive DMA interrupt	√	√	√
Serial Port 4 Send DMA Interrupt	√	√	√
Serial port 4 receive DMA interrupt	√	√	√

LCM_DMA Interrupt	√	√	√
LCM Interrupt	√	√	√
I2C Transmit DMA Interrupt	√	√	√
I2C Receive DMA Interrupt	√	√	√
I2S Interrupt			√
I2S Transmit DMA Interrupt			√
I2S receive DMA interrupt			√

12.2 STC32G Interrupt Structure Diagram



12.3 STC32G Series Interrupt List

(Table 1)

source of interruption	interrupt vector		sequence	priority setting	priority	interrupt request bit	Interrupt Allow Bit
	STC32G	STC8G/H					
INT0	FF0003H	0003H	0	PX0PX0H	0/1/2/3	IE0	EX0
Timer0	FF000BH	000BH	1	PT0,PT0H	0/1/2/3	TF0	ET0
INT1	FF0013H	0013H	2	PX1, PX1H	0/1/2/3	IE1	EX1
Timer1	FF001BH	001BH	3	PT1,PT1H	0/1/2/3	TF1	ET1
UART1	FF0023H	0023H	4	PS, PSH	0/1/2/3	RI || TI	ES
ADC	FF002BH	002BH	5	PADC, PADCH	0/1/2/3	ADC_FLAG	EADC
LVD	FF0033H	0033H	6	PLVD, PLVDH	0/1/2/3	LVDF	ELVD
UART2	FF0043H	0043H	8	PS2, PS2H	0/1/2/3	S2RI || S2TI	ES2
SPI	FF004BH	004BH	9	PSPI,PSPIH	0/1/2/3	SPIF	ESPI
INT2	FF0053H	0053H	10		0	INT2IF	EX2
INT3	FF005BH	005BH	11		0	INT3IF	EX3
Timer2	FF0063H	0063H	12		0	T2IF	ET2
INT4	FF0083H	0083H	16	PX4, PX4H	0/1/2/3	INT4IF	EX4
UART3	FF008BH	008BH	17	PS3,PS3H	0/1/2/3	S3RI || S3TI	ES3
UART4	FF0093H	0093H	18	PS4, PS4H	0/1/2/3	S4RI || S4TI	ES4
Timer3	FF009BH	009BH	19		0	T3IF	ET3
Timer4	FF00A3H	00A3H	20		0	T4IF	ET4
CMP	FF00ABH	00ABH	21	PCMP, PCMPH	0/1/2/3	CMPIF	PIE|NIE
I2C	FF00C3H	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
						STAIF	ESTAI
						RXIF	ERXI
						TXIF	ETXI
						STOIF	ESTOI
USB	FF00CBH	00CBH	25	PUSB, PUSBH	0/1/2/3	USB Events	EUSB
PWMA	FF00D3H	00D3H	26	PPWMA, PPWMAH	0/1/2/3	PWMA_SR	PWMA_IER
PWMB	FF00DBH	00DBH	27	PPWMB,PPWMBH	0/1/2/3	PWMB_SR	PWMB_IER

(Table 2)

source of interruption	interrupt vector		sequence	priority setting	priority	interrupt request bit	Interrupt Allow Bit
	STC32G	STC8G/H					
CANBUS	FF00E3H	00E3H	28	PCANL,PCANH	0/1/2/3	ALI	ALIM
						EWI	EWIM
						EPI	EPIM
						RI	RIM
						TI	TIM
						BEI	BEIM
						DOI	DOIM
CAN2BUS	FF00EBH	00EBH	29	PCAN2L,PCAN2H	0/1/2/3	ALI	ALIM
						EWI	EWIM
						EPI	EPIM
						RI	RIM
						TI	TIM
						BEI	BEIM
						DOI	DOIM
LINBUS	FF00F3H	00F3H	30	PLINL,PLINH	0/1/2/3	ABORT	ABORTE
						ERR	ERRE
						RDY	RDYE
						LID	LIDE
RTC	FF0123H	0123H	36	PRTC, PRTCH	0/1/2/3	ALAIF	EALAI
						DAYIF	EDAYI
						HOURIF	EHOURI
						MINIF	EMINI
						SECIF	ESECI
						SEC2IF	ESEC2I
						SEC8IF	ESEC8I
						SEC32IF	ESEC32I

(Table 3)

source of interruption	interrupt vector		sequence	priority setting	priority	interrupt request bit	Interrupt Allow Bit
	STC32G	STC8G/H					
P0 Interrupt	FF012BH	012BH	37	pinipl[0], piniph[0]	0/1/2/3	P0INTF	P0INTE
P1 Interrupt	FF0133H	0133H	38	pinipl[1], piniph[1]	0/1/2/3	P1INTF	P1INTE
P2 Interrupt	FF013BH	013BH	39	pinipl[2], piniph[2]	0/1/2/3	P2INTF	P2INTE
P3 Interrupt	FF0143H	0143H	40	pinipl[3], piniph[3]	0/1/2/3	P3INTF	P3INTE
P4 Interrupt	FF014BH	014BH	41	pinipl[4], piniph[4]	0/1/2/3	P4INTF	P4INTE
P5 Interrupt	FF0153H	0153H	42	pinipl[5], piniph[5]	0/1/2/3	P5INTF	P5INTE
P6 Interruptions	FF015BH	015BH	43	pinipl[6], piniph[6]	0/1/2/3	P6INTF	P6INTE
P7 Interruptions	FF0163H	0163H	44	pinipl[7], piniph[7]	0/1/2/3	P7INTF	P7INTE
DMA_M2M Interrupt	FF017BH	017BH	47	M2MIP[1:0]	0/1/2/3	M2MIF	M2MIE
DMA_ADC Interrupt	FF0183H	0183H	48	ADCIP[1:0]	0/1/2/3	ADCIF	ADCIE
DMA_SPI Interrupt	FF018BH	018BH	49	SPIIP[1:0]	0/1/2/3	SPIIF	SPIIE
DMA_UR1T Interrupt	FF0193H	0193H	50	UR1TIP[1:0]	0/1/2/3	UR1TIF	UR1TIE
DMA_UR1R Interrupt	FF019BH	019BH	51	UR1RIP[1:0]	0/1/2/3	UR1RIF	UR1RIE
DMA_UR2T interrupt	FF01A3H	01A3H	52	UR2TIP[1:0]	0/1/2/3	UR2TIF	UR2TIE
DMA_UR2R Interrupt	FF01ABH	01ABH	53	UR2RIP[1:0]	0/1/2/3	UR2RIF	UR2RIE
DMA_UR3T Interrupt	FF01B3H	01B3H	54	UR3TIP[1:0]	0/1/2/3	UR3TIF	UR3TIE
DMA_UR3R Interrupt	FF01BBH	01BBH	55	UR3RIP[1:0]	0/1/2/3	UR3RIF	UR3RIE
DMA_UR4T Interrupt	FF01C3H	01C3H	56	UR4TIP[1:0]	0/1/2/3	UR4TIF	UR4TIE
DMA_UR4R Interrupt	FF01CBH	01CBH	57	UR4RIP[1:0]	0/1/2/3	UR4RIF	UR3RIE
DMA_LCM Interrupt	FF01D3H	01D3H	58	LCMIP[1:0]	0/1/2/3	LCMIF	LCMIE
LCM Interrupt	FF01DBH	01DBH	59	LCMIFIP[1:0]	0/1/2/3	LCMIFIF	LCMIFIE
DMA_I2CT Interrupt	FF01E3H	01E3H	60	I2CTIP[1:0]	0/1/2/3	I2CTIF	I2CTIE
DMA_I2CR Interrupt	FF01EBH	01EBH	61	I2CRIP[1:0]	0/1/2/3	I2CRIF	I2CRIE
					0/1/2/3	TXE	TXEIE
					0/1/2/3	RXNE	RXNEIE

I2S interrupt	FF01F3H	01F3H	62	PI2S,PI2SH	0/1/2/3	FRE OVR UDR	ERRIE
DMA_I2ST Interrupt	FF01FBH	01FBH	63	I2STIP[1:0]	0/1/2/3	I2STIF	I2STIE
DMA_I2SR Interrupt	FF0203H	0203H	64	I2SRIP[1:0]	0/1/2/3	I2SRIF	I2SRIE

Declaring Interrupt Service Programs in C

INT0_Routine(void) interrupt 0; 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    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    I2C_Routine(void)       interrupt  24;
void    USB_Routine(void)       interrupt  25;
void    PWMA_Routine(void)      interrupt  26;
void    PWMB_Routine(void)      interrupt  27;
void    CAN_Routine(void)       interrupt  28;
void    CAN2_Routine(void)      interrupt  29;
void    LIN_Routine(void)       interrupt  30;
```

C interrupt service programme with interrupt number more than 31 cannot be declared by interrupt directly, please refer to the section "About the interrupt number more than 31 in Keil compilation error handling" in the chapter "Development Environment Establishment and ISP Download". Please refer to the section "Handling of Compilation Errors in Keil when the Interrupt Number is Greater than 31" in the section "Establishment of Development Environment and ISP Download". Assembly language is not affected.

12.4 Interrupt Related Registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	Interrupt Allow Register	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000,0000
IE2	Interrupt Allow Register 2	AFH	EUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	0000,0000
IP	Interrupt Priority Control Register	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0	x000,0000
IPH	High Interrupt Priority Control Register	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	x000,0000
IP2	Interrupt priority control register 2	B5H	PUSB	PI2C	PCMP	PX4	PPWMB	PPWMA	PSPI	PS2	0000,0000
IP2H	High Interrupt Priority Control Register 2	B6H	PUSBH	PI2CH	PCMPH	PX4H	PPWMBH	PPWMAH	PSPIH	PS2H	0000,0000
IP3	Interrupt priority control register 3	DFH	-	-	-	-	PI2S	PRTC	PS4	PS3	xxxx,0000
IP3H	High Interrupt Priority Control Register 3	EEH	-	-	-	-	PI2SH	PRTCH	PS4H	PS3H	xxxx,0000
PCON	Power Control Register	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	Timer Control Register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
INTCLKO	Interrupt and Clock Output Control Registers	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
AUXINTIF	Extended External Interrupt Flag Register	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	Serial Port 1 Control Register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	Serial Port 2 Control Register	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
S3CON	Serial Port 3 Control Register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	Serial Port 4 Control Register	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
ADC_CONTR	ADC Control Register	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]				0000,0000
SPSTAT	SPI Status Register	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
CMPCR1	Comparator Control Register 1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES	0000,xx00
CANICR	CANBUS Interrupt Control Register	F1H	PCAN2H	CAN2IF	CAN2IE	PCAN2L	PCANH	CANIF	CANIE	PCANL	0000,0000
LINICR	LINBUS Interrupt Control Register	F9H					PLINH	LINIF	LINIE	PLINL	0000,0000

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
LCMIFCFG	LCM Interface Configuration Register	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000
LCMIFSTA	LCM Interface Status Register	7EFE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0
RTCIEN	RTC Interrupt Enable Register	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I	0000,0000
RTCIF	RTC Interrupt Request Register	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF	0000,0000
I2CMSCR	I² C Host Control Register	7EFE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000

I2CMSST	I² C Host Status Register	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx10
I2CSLCR	I² C Slave Control Registers	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I² C Slave Status Register	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
PWMA_IER	PWMA Interrupt Enable Register	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWMA_SR1	PWMA status register 1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWMA_SR2	PWMA status register 2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWMB_IER	PWMB Interrupt Enable Register	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWMB_SR1	PWMB status register 1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWMB_SR2	PWMB status register 2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x

P0INTE	P0 Port Interrupt Enable Register	7EFD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 Port Interrupt Enable Register	7EFD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 Port Interrupt Enable Register	7EFD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 Port Interrupt Enable Register	7EFD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 Port Interrupt Enable Register	7EFD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 Port Interrupt Enable Register	7EFD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	xx00,0000
P6INTE	P6 Port Interrupt Enable Register	7EFD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE	0000,0000
P7INTE	P7 Port Interrupt Enable Register	7EFD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE	0000,0000
P0INTF	P0 Port Interrupt Flag Register	7EFD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 Port Interrupt Flag Register	7EFD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 Port Interrupt Flag Register	7EFD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 Port Interrupt Flag Register	7EFD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P4INTF	P4 Port Interrupt Flag Register	7EFD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 Port Interrupt Flag Register	7EFD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	xx00,0000
P6INTF	P6 Port Interrupt Flag Register	7EFD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF	0000,0000
P7INTF	P7 Port Interrupt Flag Register	7EFD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF	0000,0000
PINIPL	I/O Port Interrupt Priority Low Register	7EFD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O Port Interrupt Priority High Register	7EFD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
UR1TOCR	Serial Port 1 Timeout Control Register	7EFD70H	ENTO	ENTOI	SCALE	-	-	-	-		000x,xxxx
UR1TOSR	Serial port 1 timeout status register	7EFD71H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
UR2TOCR	Serial Port 2 Timeout Control Register	7EFD74H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR2TOSR	Serial Port 2 Timeout Status Register	7EFD75H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
UR3TOCR	Serial Port 3 Timeout Control Register	7EFD78H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR3TOSR	Serial Port 3 Timeout Status Register	7EFD79H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
UR4TOCR	Serial Port 4 Timeout Control Register	7EFD7CH	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
UR4TOSR	Serial Port 4 Timeout Status Register	7EFD7DH	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
SPITOCR	SPI Timeout Control	7EFD80H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx

	Register										
SPITOSR	SPI Timeout Status Register	7EFD81H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
I2CTOCR	I2C Timeout Control Register	7EFD84H	ENTO	ENTOI	SCALE	-	-	-	-	-	000x,xxxx
I2CTOSR	I2C Timeout Status Register	7EFD85H	-	-	-	-	-	-	-	TOIF	xxxx,xxx0
I2SCR	I2S Control Register	7EFD98H	TXEIE	RXNEIE	ERRIE	FRF	-	-	TXDMAEN	RXDMAEN	0000,xx00
I2SSR	I2S Status Register	7EFD99H	-	FRE	BUY	OVR	UDR	CHSID	TXE	RXNE	x000,0000
DMA_M2M_CFG	M2M_DMA Configuration Registers	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_STA	M2M_DMA Status Register	7EFA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_ADC_CFG	ADC_DMA Configuration Register	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_STA	ADC_DMA Status Register	7EFA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_SPI_CFG	SPI_DMA Configuration Register	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_STA	SPI_DMA Status Register	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_UR1T_CFG	UR1T_DMA Configuration Registers	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_STA	UR1T_DMA Status Register	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1R_CFG	UR1R_DMA Configuration Registers	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000
DMA_UR1R_STA	UR1R_DMA Status Register	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR2T_CFG	UR2T_DMA Configuration Registers	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_STA	UR2T_DMA Status Register	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0

DMA_UR2R_CFG	UR2R_DMA Configuration Registers	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]		0xxx,0000
DMA_UR2R_STA	UR2R_DMA Status Register	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR3T_CFG	UR3T_DMA Configuration Registers	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]		0xxx,0000
DMA_UR3T_STA	UR3T_DMA Status Register	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3R_CFG	UR3R_DMA Configuration Registers	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]		0xxx,0000
DMA_UR3R_STA	UR3R_DMA Status Register	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR4T_CFG	UR4T_DMA Configuration Registers	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_STA	UR4T_DMA Status Register	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4R_CFG	UR4R_DMA Configuration Registers	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_STA	UR4R_DMA Status Register	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_LCM_CFG	LCM_DMA Configuration Register	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]		0xxx,0000
DMA_LCM_STA	LCM_DMA Status Register	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_I2CT_CFG	I2CT_DMA Configuration Registers	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]		0xxx,0000
DMA_I2CT_STA	I2CT_DMA Status Register	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF	xxxx,x0x0
DMA_I2CR_CFG	I2CR_DMA Configuration Registers	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]		0xxx,0000
DMA_I2CR_STA	I2CR_DMA Status Register	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF	xxxx,xx00
DMA_I2ST_CFG	I2ST_DMA Configuration Registers	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]		0xxx,0000
DMA_I2ST_STA	I2ST_DMA Status Register	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF	xxxx,x0x0
DMA_I2SR_CFG	I2SR_DMA Configuration Registers	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]		0xxx,0000
DMA_I2SR_STA	I2SR_DMA Status Register	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF	xxxx,xx00

12.4.1 Interrupt enable register (interrupt allow bit)

IE (Interrupt Enable Register)

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

EA: Total Interrupt Allow Control Bit The function of EA is to make the interrupt allow form multi-level control. That is, each interrupt source is firstly controlled by EA; secondly, it is also controlled by each interrupt source's own interrupt allow control bit.

0: CPU blocks all interrupt requests

1: CPU open interrupt

ELVD: Low voltage detect interrupt allow bit. 0: Low voltage detect interrupt disabled

1: Allow low-voltage detection interrupt EADC: A/D conversion interrupt

allow bit. 0: A/D conversion
interrupt disabled
1: Allow A/D conversion
interrupt
ES: Serial port 1 interrupt
allow bit.
0: Disable serial port 1 interrupt
1: Allow serial port 1 interrupt
ET1: Overflow interrupt allow bit for timer/counter T1.
0: Disable T1 interrupt
1: Allow T1 interrupt
EX1: External interrupt 1 interrupt allow bit.

0: Disable INT1 interrupt

1: Allow INT1 interrupt

ET0: Overflow interrupt allow bit for timer/counter T0.

0: Disable T0 interrupt

1: Allow T0 interrupt

EX0: External interrupt 0 interrupt allow bit.

0: Disable INT0 interrupt

1: Allow INT0 interrupt

IE2 (Interrupt Enable Register 2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IE2	AFH	EUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

EUSB: USB interrupt allow bit.

0: Disable USB interrupt

1: Allow USB interruptions

ET4: Overflow interrupt allow bit for timer/counter T4.

0: Disable T4 interrupt

1: Allow T4 interrupt

ET3: Overflow interrupt allow bit for timer/counter T3.

0: Disable T3 interrupt

1: Allow T3 interrupt

ES4: Serial port 4 interrupt allow bit.

0: Disable serial port 4 interrupt

1: Allow serial port 4 interrupt

ES3: Serial port 3 interrupt allow bit.

0: Disable serial port 3 interrupt

1: Allow serial port 3 interrupt

ET2: Overflow interrupt allow bit for timer/counter T2.

0: Disable T2 interrupt

1: T2 interrupt allowed ESPI: SPI interrupt allowed bit.

0: SPI interrupt disabled

1: Allow SPI interrupt

ES2: Serial port 2 interrupt allow bit.

0: Disable serial port 2 interrupt

1: Allow serial port 2 interrupt

INTCLKO (External Interrupt and Clock Output Control Register)

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

EX4: External Interrupt 4 Interrupt Allow bit.

0: Disable INT4 interrupt

1: Allow INT4 interrupt

EX3: External Interrupt 3 Interrupt Allow bit.

0: Disable INT3 interrupt

1: Allow INT3 interrupt

EX2: External Interrupt 2 Interrupt Allow bit.

0: Disable INT2 interrupt

1: Allow INT2 interrupt

CMPCR1 (Comparator Control Register 1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES

PIE: Comparator rising edge interrupt allow bit. 0: Comparator rising edge interrupt disabled

1: Allow comparator rising edge interrupt

NIE: Comparator falling edge interrupt allow bit. 0: Comparator falling edge interrupt disabled

1: Allow comparator falling edge interrupt

CANICR (CAN Bus Interrupt Control Register)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CANICR	F1H	PCAN2H	CAN2IF	CAN2IE	PCAN2L	PCANH	CANIF	CANIE	PCANL

CANIE: CAN interrupt allow bit. 0: CAN interrupt disabled

1: Allow CAN interrupt

CAN2IE: CAN2 interrupt allow bit. 0: CAN2 interrupt disabled

1: Allow CAN2 interrupt

LINICR (LIN Bus Interrupt Control Register)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LINICR	F9H	-	-	-	-	PLINH	LINIF	LINIE	PLINL

LINIE: LIN interrupt allow bit. 0: LIN interrupt disabled

1: LIN interrupt allowed

LCM Interface Configuration Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80

LCMIFIE: LCM interface
interrupt allow bit. 0:
Disable LCM interface
interrupt

1: Allow LCM interface interrupt

RTC Interrupt Enable Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RTCIEN	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I

EALAI: Alarm clock interrupt enable bit 0: disable alarm clock interrupt
1: Enable alarm clock interrupt

EDAYI: One-day (24-hour) interrupt enable bit
0: Closing of one-day interruptions
1: Enabling one-day interruptions

EHOURI: One hour (60 minutes) interrupt enable bit
0: Closing of hourly interruptions
1: Enable hourly interrupt

EMINI: One-minute (60 seconds) interrupt enable bit
0: Closing of hourly interruptions
1: Enable hourly interrupt

ESECI: One second interrupt enable bit 0: Disable second interrupt
1: Enable second interrupt

ESEC2I: 1/2 second interrupt enable bit
0: Disable 1/2 second interrupt
1: Enable 1/2 second interrupt

ESEC8I: 1/8 second interrupt enable bit 0: Disable 1/8 second interrupt
1: Enable 1/8 second interrupt

ESEC32I: 1/32 second interrupt enable bit 0: Disable 1/32 second interrupt
1: Enable 1/32 second interrupt

I2C Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	7EFE81H	EMSI	-	-	-	MSCMD[3:0]			
I2CSLCR	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI: I^2C Host Mode Interrupt

Allow bit. 0: I² C host mode interrupt disabled
1: Allow I² C host mode interrupts
ESTAI: I² C slave receive START event interrupt allow bit. 0: Disable I² C slave receive START event interrupt
1: Allow I² C slave receive START event interrupt ERXI: I² C slave receive data completion event interrupt allow bit. 0: Disable I² C slave receive data completion event interrupt
1: Allow I² C slave receive data completion event interrupt

ETXI: I² C Slave send data completion
event interrupt allow bit. 0: Disable I² C
slave send data completion event
interrupt
1: Allow I² C slave to send data completion event interrupt

ESTOI: I² C slave receive STOP event
interrupt allow bit. 0: Disable I² C
slave receive STOP event interrupt
1: Allow I² C slaves to receive STOP event interrupts

PWMA Interrupt Enable Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	7EFEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE

BIE: PWMA brake interrupt
allow bit. 0: PWMA
brake interrupt disabled
1: Allow PWMA brake interrupt

TIE: PWMA trigger interrupt
allow bit. 0: PWMA
trigger interrupt
disabled
1: Allow PWMA to trigger an interrupt

COMIE: PWMA compare
interrupt allow bit. 0:
PWMA compare interrupt
disabled
1: Allow PWMA compare interrupt

CC4IE: PWMA capture compare channel 4
interrupt allow bit. 0: PWMA capture
compare channel 4 interrupt
disabled
1: Allow PWMA capture compare channel 4 interrupt

CC3IE: PWMA capture compare channel 3
interrupt allow bit. 0: PWMA capture
compare channel 3 interrupt
disabled
1: Allow PWMA capture compare channel 3 interrupt

CC2IE: PWMA capture compare channel 2
interrupt allow bit. 0: PWMA capture
compare channel 2 interrupt
disabled
1: Allow PWMA capture compare channel 2 interrupt

CC1IE: PWMA capture compare channel 1
interrupt allow bit. 0: PWMA capture

compare channel 1 interrupt
disabled
1: Allow PWMA capture compare channel 1 interrupt
UIE: PWMA update interrupt
allow bit. 0: PWMA
update interrupt
disabled
1: Allow PWMA update interrupt

PWMB Interrupt Enable Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_IER	7EFEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE

BIE: PWMB brake interrupt
allow bit. 0: PWMB
brake interrupt
disabled
1: Allow PWMB brake interrupt

TIE: PWMB trigger interrupt
allow bit. 0: PWMB
trigger interrupt
disabled
1: Allow PWMB to trigger an interrupt

COMIE: PWMB compare
interrupt allow bit. 0:
PWMB compare interrupt
disabled
1: Allow PWMB compare interrupt

CC8IE: PWMB capture compare channel 8
interrupt allow bit. 0: PWMB Capture
Compare Channel 8 Interrupt
Disable
1: Allows PWMB to capture compare channel 8 interrupts

CC7IE: PWMB capture compare channel 7
interrupt allow bit. 0: PWMB capture
compare channel 7 interrupt
disabled
1: Allow PWMB to capture compare channel 7 interrupt

CC6IE: PWMB capture compare channel 6
interrupt allow bit. 0: PWMB capture
compare channel 6 interrupt
disabled
1: Allows PWMB to capture compare channel 6 interrupts

CC5IE: PWMB capture compare channel 5
interrupt allow bit. 0: PWMB capture
compare channel 5 interrupt
disabled
1: Allows PWMB to capture the compare channel 5 interrupt.

UIE: PWMB update interrupt
allow bit. 0: PWMB
update interrupt
disabled
1: Allow PWMB update interrupt

Port Interrupt Enable Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	7EFD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	7EFD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	7EFD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	7EFD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P4INTE	7EFD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	7EFD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE

P6INTE	7EFD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE
P7INTE	7EFD07H	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

Serial Port 1 Timeout Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UR1TOCR	7EFD70H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI: Serial port 1 timeout interrupt allow bit. 0: Disable serial port 1 timeout interrupt

1: Allow serial port 1 timeout interrupt

Serial Port 2 Timeout Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UR2TOCR	7EFD74H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI: Serial port 2 timeout interrupt allowed bit. 0: Disable serial port 2 timeout interrupt

1: Allow serial port 2 timeout interrupt

Serial Port 3 Timeout Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UR3TOCR	7EFD78H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI: Serial port 1 timeout interrupt allowed bit. 0: Disable serial port 3 timeout interrupt

1: Allow serial port 3 timeout interrupt

Serial Port 4 Timeout Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UR4TOCR	7EFD7CH	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI: Serial port 1 timeout interrupt allowed bit. 0: Disable serial port 4 timeout interrupt

1: Allow serial port 4 timeout interrupt

SPI Timeout Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
SPITOCR	7EFD80H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI: SPI timeout interrupt allow bit. 0: SPI timeout interrupt disabled

1: Allow SPI timeout interrupt

I2C Timeout Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CTOCR	7EFD84H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI: I2C timeout interrupt allowed bit. 0: I2C timeout interrupt disabled

1: Allow I2C timeout interrupt

I2S Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2SCR	7EFD98H	TXEIE	RXNEIE	ERRIE	FRF	-	-	TXDMAEN	RXDMAEN

TXEIE: Output buffer air interrupt allow bit. 0: Disable output buffer air break
1: Allow output buffer air break
RXNEIE: Input buffer off-air interrupt allow bit. 0: Input buffer non-air disconnect disable
1: Allow input buffer to be disconnected out-of-air
ERRIE: Error interrupt allow bit. 0: Error interrupt disable
1: Allow error interruptions

DMA_I2CT_CFG	I2CT_DMA Configuration Registers	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]	I2CTPTY[1:0]	0xxx,0000
DMA_I2CR_CFG	I2CR_DMA Configuration Registers	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]	I2CRPTY[1:0]	0xxx,0000
DMA_I2ST_CFG	I2ST_DMA Configuration Registers	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]	I2STPTY[1:0]	0xxx,0000
DMA_I2SR_CFG	I2SR_DMA Configuration Registers	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]	I2SRPTY[1:0]	0xxx,0000

DMA Interrupt Enable Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	
DMA_ADC_CFG	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]	
DMA_SPI_CFG	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	
DMA_UR1T_CFG	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	
DMA_UR1R_CFG	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	
DMA_UR2T_CFG	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	
DMA_UR2R_CFG	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	
DMA_UR3T_CFG	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	
DMA_UR3R_CFG	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	
DMA_UR4R_CFG	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	
DMA_UR4R_CFG	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	
DMA_LCM_CFG	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]	
DMA_I2CT_CFG	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]	
DMA_I2CR_CFG	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]	
DMA_I2ST_CFG	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]	
DMA_I2SR_CFG	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]	

M2MIE: DMA_M2M (memory to memory DMA) interrupt

allow bit. 0: Disable DMA_M2M interrupt

1: DMA_M2M interrupt allowed

ADCIE: DMA_ADC (ADC DMA) interrupt

allow bit. 0: Disable DMA_ADC interrupt

1: Allow DMA_ADC interrupt

SPIIE: DMA_SPI (SPI DMA) interrupt allow bit.

0: Disable DMA_SPI interrupt

1: Allow DMA_SPI interrupt

UR1TIE: DMA_UR1T (serial port 1 send DMA) interrupt allow bit. 0: Disable DMA_UR1T interrupt

1: DMA_UR1T interrupt allowed

UR1RIE: DMA_UR1R (serial port 1 receive DMA) interrupt allow bit. 0: Disable DMA_UR1R interrupt

1: DMA_UR1R interrupt allowed

UR2TIE: DMA_UR2T (serial port 2 send DMA) interrupt allow bit. 0: Disable DMA_UR2T interrupt

1: DMA_UR2T interrupt allowed

UR2RIE: DMA_UR2R (serial port 2 receive DMA) interrupt allow bit. 0: Disable DMA_UR2R interrupt

1: DMA_UR2R interrupt allowed

UR3TIE: DMA_UR3T (serial port 3 send DMA) interrupt allow bit. 0: Disable DMA_UR3T interrupt

1: DMA_UR3T interrupt allowed

UR3RIE: DMA_UR3R (serial port 3 receive DMA) interrupt allow bit. 0: Disable DMA_UR3R interrupt

1: DMA_UR3R interrupt allowed

UR4TIE: DMA_UR4T (serial port 4 send DMA) interrupt allow bit. 0: Disable DMA_UR4T interrupt

1: DMA_UR4T interrupt allowed

UR4RIE: DMA_UR4R (serial port 4 receive DMA) interrupt allow bit. 0: Disable DMA_UR4R interrupt

1: DMA_UR4R interrupt allowed

LCMIE: DMA_LCM (LCM interface DMA) interrupt allow bit. 0: Disable DMA_LCM interrupt

1: Allow DMA_LCM interrupt

I2CTIE: DMA_I2CT (I2C transmit DMA) interrupt allow bit. 0: Disable DMA_I2CT interrupt

1: DMA_I2CT interrupt allowed

I2CRIE: DMA_I2CR (I2C receive DMA) interrupt allow bit. 0: Disable DMA_I2CR interrupt

1: DMA_I2CR interrupt allowed

I2STIE: DMA_I2ST (I2S transmit DMA) interrupt allow bit. 0: Disable DMA_I2ST interrupt

1: DMA_I2ST interrupt allowed

I2SRIE: DMA_I2SR (I2S receive DMA) interrupt allow bit. 0: Disable DMA_I2SR interrupt

1: DMA_I2SR interrupt allowed

12.4.2 Interrupt request register (interrupt flag bit)

Timer Control Register

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

TF1: Timer 1 overflow interrupt flag. It is automatically cleared by hardware in the interrupt service programme. TF0: Timer 0 overflow interrupt flag. It is automatically cleared by hardware in the interrupt service routine. IE1: External interrupt 1 interrupt request flag. It is automatically cleared by hardware in the interrupt service programme. IE0: External interrupt 0 interrupt request flag. It is automatically cleared by hardware in the interrupt service routine.

Interrupt Flag Auxiliary Register

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

INT4IF: external interrupt 4 interrupt request flag. It is automatically cleared by hardware in the interrupt service programme. INT3IF: external interrupt 3 interrupt request flag. It is automatically cleared by hardware in interrupt service programme. INT2IF: external interrupt 2 interrupt request flag. It is automatically cleared by hardware in interrupt service programme. T4IF: Timer 4 overflow interrupt flag. It is automatically cleared by hardware in the interrupt service routine. T3IF: Timer 3 overflow interrupt flag. It is automatically cleared by hardware in interrupt service routine. T2IF: Timer 2 overflow interrupt flag. It is automatically cleared by hardware in the interrupt service routine.

Serial Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
S2CON	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI
S4CON	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

TI: Serial port 1 transmit completion interrupt request flag. Need to be cleared by software. RI:

Serial port 1 receive completion interrupt request flag. Software clearing is required. S2TI: Serial port 2 transmit completion interrupt request flag. Software clearing is required. S2RI: Serial port 2 receive completion interrupt request flag. Software clearing is required. S3TI: Serial port 3 transmit completion interrupt request flag. Software clearing is required. S3RI: Serial port 3 receive completion interrupt request flag. Software clearing is required. S4TI: Serial port 4 transmit completion interrupt request flag. Software clearing is required. S4RI: Serial port 4 receive completion interrupt request flag. Software clearing is required.

Power Management Registers

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

LVDF: Low Voltage Detection Interrupt Request Flag. Software clearing is required.

ADC Control Register

notation	structural particle: used before a verb or adjective, linking it preceding the verb or adjective sites	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 conversion completion interrupt request flag. Needs to be cleared by software.

SPI Status Register

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

SPIF: SPI data transfer completion interrupt request flag.

It needs software to write "1" to clear it.

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES

Comparator Control Register 1

CMPIF: Comparator Interrupt Request Flag. Needs to be cleared by software.

CANICR (CAN Bus Interrupt Control Register)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CANICR	F1H	PCAN2H	CAN2IF	CAN2IE	PCAN2L	PCANH	CANIF	CANIE	PCANL

CANIF: CAN interrupt request flag. (Read only) CAN2IF: CAN2 interrupt request flag. (read-only)

LINICR (LIN Bus Interrupt Control Register)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LINICR	F9H	-	-	-	-	PLINH	LINIF	LINIE	PLINL

LINIF: LIN Interrupt Request Flag. (read-only)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFSTA	7EFE53H	-	-	-	-	-	-	-	LCMIFIF

LCM Interface Status Register

LCMIFIF: LCM interface interrupt request flag. Needs to be cleared by software.

RTC Interrupt Request Flag Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RTCIF	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF

ALAIF: Alarm clock interrupt request bit. Software clearing is required. DAYIF: Day (24-hour) interrupt request bit. Software clearing is required.
HOURIF: One hour (60 minutes) interrupt request bit. Software clearing is required. MINIF: One minute (60 seconds) interrupt request bit. Software clearing is required. SECIF: One second interrupt request bit. Software clearing is required.
SEC2IF: 1/2 second interrupt request bit. Software clear required SEC8IF: 1/8 second interrupt request bit. Software clear required. SEC32IF: 1/32 second interrupt request bit. Software clear required.

I2C Status Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO
I2CSLST	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO

MSIF: I² C Host Mode Interrupt Request Flag. Software clearing is required. ESTAI: I² C Slave receive START event interrupt request flag. Software clearing required.
ERXI: I² C Slave receive data completion event interrupt request flag. Software clearing is required. ETXI: I² C Slave transmit data completion event interrupt request flag. Software clearing is required. ESTOI: I² C Slave receive STOP event interrupt request flag. Software clearing is required.

PWMA Status Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	7EFEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
PWMA_SR2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-

BIF: PWMA brake interrupt request flag. Need to be cleared by software. TIF: PWMA trigger interrupt request flag. Need to be cleared by software. COMIF: PWMA compare interrupt request flag. Software clearing is required.

CC4IF: Capture Compare Interrupt Request Flag occurs on PWMA channel 4. Software clearing is required. CC3IF: PWMA channel 3 capture compare interrupt request flag. Software clearing is required. CC2IF: PWMA channel 2 capture compare interrupt request flag. Software clearing is required. CC1IF: PWMA channel 1 capture compare interrupt request flag. Software clearing is required. UIF: PWMA update interrupt request flag. Software clearing is required.

CC4OF: Repeat Capture Interrupt Request Flag occurs on PWMA channel 4. Software clearing is required. CC3OF: PWMA channel 3 repeat capture interrupt request flag. Software clearing is required. CC2OF: PWMA channel 2 repeat capture interrupt request flag. Software clearing is required.

CC1OF: Repeat Capture Interrupt Request Flag occurs on PWMA channel 1. Software clearing is required.

PWMB Status Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_SR1	7EFEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF
PWMB_SR2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

BIF: PWMB brake interrupt request flag. Need to be cleared by software. TIF: PWMB trigger interrupt request flag. Software clearing is required. COMIF: PWMB compare interrupt request flag. Software clearing is required.

CC8IF: Capture Compare Interrupt Request Flag occurs on PWMB channel 8. Software clearing is required. CC7IF: PWMB channel 7 capture compare interrupt request flag. Software clearing is required. CC6IF: PWMB channel 6 capture compare interrupt request flag. Software clearing is required. CC5IF: PWMB channel 5 capture compare interrupt request flag. Software clearing is required. UIF: PWMB update interrupt request flag. Software clearing is required.

CC8OF: Repeat Capture Interrupt Request Flag occurs on PWMB channel 8. Software clearing is required. CC7OF: PWMB channel 7 repeat capture interrupt request flag. Software clearing is required. CC6OF: PWMB channel 6 repeat capture interrupt request flag. Software clearing is required. CC5OF: PWMB channel 5 repeat capture interrupt request flag. Software clearing is required.

Port Interrupt Flag Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	7EFD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	7EFD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	7EFD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	7EFD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P4INTF	7EFD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF
P5INTF	7EFD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF
P6INTF	7EFD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF
P7INTF	7EFD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF

PnINTF.x: port interrupt request flag bit (n=0~7, x=0~7) 0: no interrupt request on Pn.x port

1: There is an interrupt request on the Pn.x port, if the interrupt is enabled, it will enter the

interrupt service routine. Software clearing is required.

Serial port 1 timeout status register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UR1TOSR	7EFD71H	-	-	-	-	_	_	_	TOIF

TOIF: serial port 1 timeout interrupt request flag. Need to be cleared by software.

Serial Port 2 Timeout Status Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UR2TOSR	7EFD75H	-	-	-	-	_	_	_	TOIF

TOIF: serial port 2 timeout interrupt request

flag. Needs to be cleared by software. Serial

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UR3TOSR	7EFD79H	-	-	-	-	_	_	_	TOIF

port 3 timeout status register

TOIF: Serial port 3 timeout interrupt request

flag. Software clearing is required. Serial **port**

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UR4TOSR	7EFD7DH	-	-	-	-	_	_	_	TOIF

4 timeout status register

TOIF: serial port 4 timeout interrupt request flag. Software clearing is required.

SPI Timeout Status Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
SPITOSR	7EFD81H	-	-	-	-	_	_	_	TOIF

TOIF: SPI Timeout Interrupt Request Flag. Needs to be cleared by software.

I2C Timeout Status Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CTOSR	7EFD75H	-	-	-	-	_	_	_	TOIF

TOIF: I2C Timeout Interrupt Request Flag. Needs to be cleared by software.

I2S Timeout Status Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2STOSR	7EFD79H	-	-	-	-	_	_	_	TOIF

TOIF: I2S Timeout Interrupt Request Flag. Needs to be cleared by software.

DMA Interrupt Flag Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	7EFA02H	-	-	-	-	-	-	-	M2MIF
DMA_ADC_STA	7EFA12H	-	-	-	-	-	-	-	ADCIF
DMA_SPI_STA	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF
DMA_UR1T_STA	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF
DMA_UR1R_STA	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF
DMA_UR2T_STA	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF

DMA_UR2R_STA	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF
DMA_UR3T_STA	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF
DMA_UR3R_STA	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF
DMA_UR4T_STA	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF
DMA_UR4R_STA	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF
DMA_LCM_STA	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF
DMA_I2CT_STA	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF
DMA_I2CR_STA	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF
DMA_I2ST_STA	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF
DMA_I2SR_STA	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF

M2MIF: DMA_M2M (memory to memory DMA) interrupt request flag. Software clearing is required. ADCIF: DMA_ADC (ADC DMA) interrupt request flag. Software clearing is required. SPIIF: DMA_SPI (SPI DMA) interrupt request flag. Software clearing is required. UR1TIF: DMA_UR1T (Serial Port 1 Transmit DMA) interrupt request flag. Requires software clearing. UR1RIF: DMA_UR1R (Serial Port 1 Receive DMA) interrupt request flag. Software clearing is required. UR2TIF: DMA_UR2T (serial port 2 send DMA) interrupt request flag. Software clearing is required. UR2RIF: DMA_UR2R (serial port 2 receive DMA) interrupt request flag. Software clearing is required. UR3TIF: DMA_UR3T (serial port 3 transmit DMA) interrupt request flag. Software clearing is required. UR3RIF: DMA_UR3R (serial port 3 receive DMA) interrupt request flag. Software clearing is required. UR4TIF: DMA_UR4T (serial port 4 transmit DMA) interrupt request flag. Software clearing is required. UR4RIF: DMA_UR4R (serial port 4 receive DMA) interrupt request flag. Software clearing is required. LCMIF: DMA_LCM (LCM interface DMA) interrupt request flag. Software clearing is required. I2CTIF: DMA_I2CT (I2C transmit DMA) interrupt request flag. Software clearing is required. I2CRIF: DMA_I2CR (I2C receive DMA) interrupt request flag. Software clearing is required. I2STIF: DMA_I2ST (I2S transmit DMA) interrupt request flag. Software clearing is required. I2SRIF: DMA_I2SR (I2S receive DMA) interrupt request flag. Software clearing is required.

12.4.3 Interrupt Priority Register

Interrupt Priority Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	PPWMA	PLVD	PADC	PS	PT1	PX1	PT0	PX0
IPH	B7H	PPWMAH	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	PUSB	PI2C	PCMP	PX4	PPWMB	PUSB	PSPI	PS2
IP2H	B6H	PUSBH	PI2CH	PCMPH	PX4H	PPWMBH	PUSBH	PSPIH	PS2H
IP3	DFH	-	-	-	-	PI2S	PRTC	PS4	PS3

IP3H	EEH	-	-	-	-	PI2SH	PRTCH	PS4H	PS3H

PX0H,PX0: External Interrupt 0 Interrupt Priority Control Bit 00: INT0 Interrupt Priority Level 0 (lowest level)
01: INT0 Interrupt priority level 1 (lower level)
10: INT0 Interrupt priority level 2 (higher level)

11: INT0 Interrupt priority 3 (highest level)

PT0H,PT0: Timer 0 interrupt priority control
bit 00: Timer 0 interrupt priority is level 0 (lowest level)
01: Timer 0 interrupt priority 1 (lower level)
10: Timer 0 interrupt priority 2 (higher level)
11: Timer 0 interrupt priority 3 (highest level)

PX1H,PX1: External Interrupt 1 Interrupt Priority Control Bit 00: INT1 Interrupt Priority Level 0 (lowest level)
01: INT1 Interrupt priority level 1 (lower level)
10: INT1 Interrupt priority level 2 (higher level)
11: INT1 Interrupt priority 3 (highest level)

PT1H,PT1: Timer 1 interrupt priority control
bit 00: Timer 1 interrupt priority is level 0 (lowest level)
01: Timer 1 interrupt priority 1 (lower level)
10: Timer 1 interrupt priority 2 (higher level)
11: Timer 1 interrupt priority 3 (highest level)

PSH,PS: Serial port 1 interrupt priority control bit
00: Serial port 1 interrupt priority 0 (lowest level)
01: Serial port 1 interrupt priority 1 (lower level)
10: Serial port 1 interrupt priority 2 (higher level)
11: Serial port 1 interrupt priority 3 (highest level)

PADCH,PADC: ADC interrupt priority control bit 00: ADC interrupt priority is level 0 (lowest level)
01: ADC interrupt priority level 1 (lower level)
10: ADC interrupt priority level 2 (higher)
11: ADC interrupt priority is level 3 (highest level)

PLVDH,PLVD: Low voltage detection interrupt priority control bit
00: LVD interrupt priority is level 0 (lowest level)
01: LVD interrupt priority 1 (lower level)
10: LVD interrupt priority 2 (higher level)
11: LVD interrupt priority 3 (highest level)

PS2H,PS2: Serial Port 2 Interrupt Priority Control Bit 00: Serial Port 2 Interrupt Priority Level 0 (lowest level)
01: Serial port 2 interrupt priority 1 (lower level)
10: Serial port 2 interrupt priority 2 (higher level)
11: Serial port 2 interrupt priority 3 (highest level)

PS3H,PS3: Serial 3 interrupt priority

control bit 00: Serial 3 interrupt
priority is level 0 (lowest level)
01: Serial port 3 interrupt priority 1 (lower level)
10: Serial port 3 interrupt priority 2 (higher level)
11: Serial port 3 interrupt priority 3 (highest level)

PS4H,PS4: Serial Port 4 Interrupt Priority
Control Bit 00: Serial Port 4 Interrupt
Priority is Level 0 (lowest level)
01: Serial port 4 interrupt priority 1 (lower level)

10: Serial port 4 interrupt priority 2 (higher level)

11: Serial port 4 interrupt priority 3 (highest level)

PSPIH,PSPI: SPI interrupt priority control bit 00: SPI interrupt priority is level 0 (lowest level)

01: SPI interrupt priority level 1 (lower level)

10: SPI interrupt priority level 2 (higher level)

11: SPI interrupt priority 3 (highest level)

PPWMAH,PPWMA: Advanced PWMA Interrupt Priority Control Bit 00: Advanced PWMA interrupt priority is level 0 (lowest level)

01: Advanced PWMA interrupt priority 1 (lower level)

10: Advanced PWMA interrupt priority 2 (higher level)

11: Advanced PWMA interrupt priority is level 3 (highest level) PPWMBH,PPWMB: Advanced PWMB interrupt priority control bit

00: Advanced PWMB interrupt priority is level 0 (lowest level)

01: Advanced PWMB interrupt priority 1 (lower level)

10: Advanced PWMB Interrupt Priority 2 (higher level)

11: Advanced PWMB Interrupt Priority 3 (highest level)

PX4H,PX4: External Interrupt 4 Interrupt Priority Control Bit 00: INT4 Interrupt Priority is Level 0 (lowest level)

01: INT4 Interrupt priority level 1 (lower level)

10: INT4 Interrupt priority level 2 (higher level)

11: INT4 Interrupt priority 3 (highest level)

PCMPH,PCMP: Comparator Interrupt Priority Control Bit 00: CMP interrupt priority is level 0 (lowest level)

01: CMP interrupt priority level 1 (lower level)

10: CMP interrupt priority 2 (higher level)

11: CMP interrupt priority 3 (highest level)

PI2CH,PI2C: I2C interrupt priority control bit 00: I2C interrupt priority level 0 (lowest level)

01: I2C interrupt priority level 1 (lower level)

10: I2C interrupt priority level 2 (higher level)

11: I2C interrupt priority level 3 (highest level)

PUSBH,PUSB: USB interrupt priority control bit 00: USB interrupt priority is level 0 (lowest level)

01: USB interrupt priority 1 (lower level)
10: USB interrupt priority 2 (higher level)
11: USB interrupt priority 3 (highest level)

PRTCH,PRTC: RTC interrupt priority
control bit 00: RTC interrupt priority
is level 0 (lowest level)
01: RTC interrupt priority level 1 (lower level)
10: RTC interrupt priority level 2 (higher level)
11: RTC interrupt priority level 3 (highest level)

PI2SH,PI2S: I2S interrupt priority
control bit 00: I2S interrupt priority
level 0 (lowest level)

01: I2S interrupt priority level 1 (lower level)
10: I2S interrupt priority level 2 (higher level)
11: I2S interrupt priority 3 (highest level)

CANICR (CAN Bus Interrupt Control Register)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CANICR	F1H	PCAN2H	CAN2IF	CAN2IE	PCAN2L	PCANH	CANIF	CANIE	PCANL

PCANH,PCANL: CAN interrupt priority
control bit 00: CAN interrupt priority
level 0 (lowest level)
01: CAN interrupt priority level 1 (lower level)
10: CAN interrupt priority level 2 (higher level)
11: CAN interrupt priority level 3
(highest level) PCAN2H,PCAN2L: CAN2
interrupt priority control bit 00: CAN2
interrupt priority level 0 (lowest level)
01: CAN2 interrupt priority level 1 (lower level)
10: CAN2 interrupt priority 2 (higher level)
11: CAN2 interrupt priority 3 (highest level)

LINICR (LIN Bus Interrupt Control Register)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LINICR	F9H	-	-	-	-	PLINH	LINIF	LINIE	PLINL

PLINH,PLINL: LIN interrupt priority
control bit 00: LIN interrupt priority
level 0 (lowest level)
01: LIN interrupt priority level 1 (lower level)
10: LIN interrupt priority 2 (higher level)
11: LIN interrupt priority 3 (highest level)

LCM Interface Configuration Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFE50H	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 is level 0 (lowest level)
01: LCM interface interrupt priority 1 (lower level)
10: LCM interface interrupt priority 2 (higher level)
11: LCM interface interrupt priority 3 (highest level)

Port Interrupt Priority Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
~~PINIPL~~	~~7EFD60H~~	~~P7IP~~	~~P6IP~~	~~P5IP~~	~~P4IP~~	~~P3IP~~	~~P2IP~~	~~P1IP~~	~~P0IP~~
PINIPH	7EFD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH

P0IPH,P0IP: P0 port interrupt priority
control bit 00: P0 port interrupt priority 0 (lowest level)
01: P0 port interrupt priority level 1 (lower level)
10: P0 port interrupt priority level 2 (higher level)
11: P0 port interrupt priority 3 (highest level)

P1IPH,P1IP: P1 port interrupt priority
control bit 00: P1 port interrupt priority is level 0 (lowest level)
01: P1 port interrupt priority 1 (lower level)
10: P1 port interrupt priority 2 (higher level)
11: P1 port interrupt priority 3 (highest level)

P2IPH,P2IP: P2 port interrupt priority
control bit 00: P2 port interrupt priority is level 0 (lowest level)
01: P2 port interrupt priority 1 (lower level)
10: P2 port interrupt priority 2 (higher level)
11: P2 port interrupt priority 3 (highest level)

P3IPH,P3IP: P3 port interrupt priority
control bit 00: P3 port interrupt priority is level 0 (lowest level)
01: P3 port interrupt priority level 1 (lower level)
10: P3 port interrupt priority 2 (higher level)
11: P3 port interrupt priority 3 (highest level)

P4IPH,P4IP: P4 port interrupt priority
control bit 00: P4 port interrupt priority is level 0 (lowest level)
01: P4 port interrupt priority 1 (lower level)
10: P4 port interrupt priority 2 (higher level)
11: P4 port interrupt priority 3 (highest level)

P5IPH,P5IP: P5 port interrupt priority
control bit 00: P5 port interrupt priority is level 0 (lowest level)
01: P5 port interrupt priority 1 (lower level)
10: P5 port interrupt priority 2 (higher level)
11: P5 port interrupt priority 3 (highest level)

P6IPH,P6IP: P6 port interrupt priority
control bit 00: P6 port interrupt priority is level 0 (lowest level)
01: P6 port interrupt priority 1 (lower level)
10: P6 port interrupt priority 2 (higher level)
11: P6 port interrupt priority 3 (highest level)

P7IPH,P7IP: P7 port interrupt priority
control bit 00: P7 port interrupt priority is level 0 (lowest level)

01: P7 port interrupt priority 1 (lower level)

10: P7 port interrupt priority 2 (higher level)

11: P7 port interrupt priority 3 (highest level)

DMA Interrupt Priority Control Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0

DMA_M2M_CFG	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]	M2MPTY[1:0]
DMA_ADC_CFG	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]	ADCPTY[1:0]
DMA_SPI_CFG	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]	SPIPTY[1:0]
DMA_UR1T_CFG	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]	UR1TPTY[1:0]
DMA_UR1R_CFG	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]	UR1RPTY[1:0]
DMA_UR2T_CFG	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]	UR2TPTY[1:0]
DMA_UR2R_CFG	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]	UR2RPTY[1:0]
DMA_UR3T_CFG	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]	UR3TPTY[1:0]
DMA_UR3R_CFG	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]	UR3RPTY[1:0]
DMA_UR4T_CFG	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]	UR4TPTY[1:0]
DMA_UR4R_CFG	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]	UR4RPTY[1:0]
DMA_LCM_CFG	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]	LCMPTY[1:0]
DMA_I2CT_CFG	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]	I2CTPTY[1:0]
DMA_I2CR_CFG	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]	I2CRPTY[1:0]
DMA_I2ST_CFG	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]	I2STPTY[1:0]
DMA_I2SR_CFG	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]	I2SRPTY[1:0]

M2MIP: DMA_M2M (Memory to Memory DMA) Interrupt Priority Control Bit 00: DMA_M2M Interrupt Priority Level 0 (lowest level)
01: DMA_M2M interrupt priority level 1 (lower level)
10: DMA_M2M interrupt priority level 2 (higher level)
11: DMA_M2M interrupt priority level 3 (highest level) ADCIP: DMA_ADC (ADC DMA) interrupt priority control bit 00: DMA_ADC interrupt priority level 0 (lowest level)
01: DMA_ADC interrupt priority level 1 (lower level)
10: DMA_ADC interrupt priority level 2 (higher level)
11: DMA_ADC interrupt priority 3 (highest level)
SPIIP: DMA_SPI (SPI DMA) Interrupt Priority Control Bit 00: DMA_SPI Interrupt Priority Level 0 (lowest level)
01: DMA_SPI interrupt priority is level 1 (lower level)
10: DMA_SPI interrupt priority level 2 (higher level)
11: DMA_SPI interrupt priority 3 (highest level)
UR1TIP: DMA_UR1T (Serial Port 1 Send DMA) Interrupt Priority Control Bit 00: DMA_UR1T Interrupt Priority Level 0 (lowest level)
01: DMA_UR1T Interrupt priority level 1 (lower level)
10: DMA_UR1T Interrupt priority level 2 (higher level)
11: DMA_UR1T Interrupt priority 3 (highest level)
UR1RIP: DMA_UR1R (Serial Port 1 Receive DMA) Interrupt Priority Control Bit 00: DMA_UR1R Interrupt Priority Level 0 (lowest level)
01: DMA_UR1R Interrupt priority level 1 (lower level)
10: DMA_UR1R Interrupt priority level 2 (higher level)

11: DMA_UR1R Interrupt priority level 3 (highest level)

UR2TIP: DMA_UR2T (Serial Port 2 Send DMA) Interrupt

Priority Control Bit 00: DMA_UR2T Interrupt

Priority Level 0 (lowest level)

01: DMA_UR2T Interrupt priority level 1 (lower level)

10: DMA_UR2T interrupt priority level 2 (higher level)
11: DMA_UR2T interrupt priority 3 (highest level)
UR2RIP: DMA_UR2R (Serial Port 2 Receive DMA)
Interrupt Priority Control Bit 00: DMA_UR2R
Interrupt Priority Level 0 (lowest level)
01: DMA_UR2R Interrupt priority level 1 (lower level)
10: DMA_UR2R interrupt priority level 2 (higher level)
11: DMA_UR2R interrupt priority 3 (highest level)
UR3TIP: DMA_UR3T (Serial Port 3 Send DMA) Interrupt
Priority Control Bit 00: DMA_UR3T Interrupt
Priority Level 0 (lowest level)
01: DMA_UR3T Interrupt priority level 1 (lower level)
10: DMA_UR3T interrupt priority level 2 (higher level)
11: DMA_UR3T interrupt priority 3 (highest level)
UR3RIP: DMA_UR3R (Serial Port 3 Receive DMA)
Interrupt Priority Control Bit 00: DMA_UR3R
Interrupt Priority Level 0 (lowest level)
01: DMA_UR3R Interrupt priority level 1 (lower level)
10: DMA_UR3R Interrupt priority level 2 (higher level)
11: DMA_UR3R Interrupt priority 3 (highest level)
UR4TIP: DMA_UR4T (Serial Port 4 Transmit DMA)
Interrupt Priority Control Bit 00: DMA_UR4T
Interrupt Priority Level 0 (lowest level)
01: DMA_UR4T Interrupt priority level 1 (lower level)
10: DMA_UR4T interrupt priority level 2 (higher level)
11: DMA_UR4T interrupt priority 3 (highest level)
UR4RIP: DMA_UR4R (Serial Port 4 Receive DMA)
Interrupt Priority Control Bit 00: DMA_UR4R
Interrupt Priority Level 0 (lowest level)
01: DMA_UR4R Interrupt priority level 1 (lower level)
10: DMA_UR4R Interrupt priority level 2 (higher level)
11: DMA_UR4R Interrupt priority 3 (highest level)
LCMIP: DMA_LCM (LCM Interface DMA) Interrupt
Priority Control Bit 00: DMA_LCM Interrupt
Priority Level 0 (lowest level)
01: DMA_LCM interrupt priority is level 1 (lower level)
10: DMA_LCM interrupt priority level 2 (higher level)
11: DMA_LCM interrupt priority level 3 (highest
level) I2CTIP: DMA_I2CT (I2C send DMA) interrupt
priority control bit 00: DMA_I2CT interrupt priority
level 0 (lowest level)
01: DMA_I2CT interrupt priority level 1 (lower level)
10: DMA_I2CT interrupt priority level 2 (higher level)
11: DMA_I2CT interrupt priority level 3 (highest level)
I2CRIP: DMA_I2CR (I2C receive DMA) interrupt

priority control bit 00: DMA_I2CR interrupt priority level 0 (lowest level)
01: DMA_I2CR interrupt priority level 1 (lower level)
10: DMA_I2CR interrupt priority level 2 (higher level)
11: DMA_I2CR interrupt priority level 3 (highest level) I2STIP: DMA_I2ST (I2S transmit DMA) interrupt priority control bit 00: DMA_I2ST interrupt priority level 0 (lowest level)

01: DMA_I2ST interrupt priority level 1 (lower level)
10: DMA_I2ST interrupt priority level 2 (higher level)
11: DMA_I2ST interrupt priority level 3 (highest level) I2SRIP: DMA_I2SR (I2S receive DMA) interrupt priority control bit 00: DMA_I2SR interrupt priority level 0 (lowest level)
01: DMA_I2SR interrupt priority level 1 (lower level)
10: DMA_I2SR interrupt priority level 2 (higher level)
11: DMA_I2SR interrupt priority level 3 (highest level)

12.5 sample procedure

12.5.1 INT0 interrupts (rising and falling edges), can support both rising and falling edges

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void INT0_Isr() interrupt 0
{
    if (P32)                                 //Determine the rising and falling edges
    {
        P10 = !P10;                          //test port
    }
    else
    {
        P11 = !P11;                          //test port
    }
}

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    IT0 = 0; // Enable INT0 rising and falling edge interrupts.   //enable INT0 rising and falling edge interrupt
    EX0 = 1; // Enable INT0 interrupt.                            //Enable INT0 interrupt
    EA = 1;

    while (1);
}
```

12.5.2 INT0 Interrupt (falling edge)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files
#include "intrins.h"

void INT0_Isr() interrupt 0
{
    P10 = !P10;                             //test port
}

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign a value of 0 to set the CPU to execute the
                                            programme as fast as possible.

    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.   //enable INT0 falling edge interrupt
    EX0 = 1; // Enable INT0 interrupt.               //Enable INT0 interrupt
    EA = 1;

    while (1);
}
```

12.5.3 INT1 interrupts (rising and falling edges), can support both rising and falling edges

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files
#include "intrins.h"

void INT1_Isr() interrupt 2
{
```

```
    if (INT1)                                   //Determine the rising and falling edges
    {
        P10 = !P10;                             //test port
    }
    else
    {
        P11 = !P11;                             //test port
    }
}

void main()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign a value of 0 to set the CPU to execute the
                                                programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    IT1 = 0.                                    //Enable INT1 rising and falling edge interrupts
    EX1 = 1;                                    //Enable INT1 interrupt
    EA = 1;

    while (1);
}
```

12.5.4 INT1 Interrupt (falling edge)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             // see download software for header files
#include "intrins.h"

void INT1_Isr() interrupt 2
{
    P10 = !P10;                                 //test port
}

void main()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
```

```
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.
    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);
}
```

12.5.5 INT2 interrupt (falling edge), only falling edge interrupt is supported

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     //see download software for header files
#include "intrins.h"

void INT2_Isr() interrupt 10
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
```

```
    p5m0 = 0x00;
    p5m1 = 0x00.
```

```
    EX2 = ;                                 //Enable INT2 interrupt
    EA = 1;

    while (1);
}
```

12.5.6 INT3 interrupt (falling edge), only falling edge interrupt is supported

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         //see download software for header files
#include "intrins.h"

void INT3_Isr() interrupt 11
{
    P10 = !P10;                             //test port
}

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set the external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign a value of 0 to set the CPU to execute the
                                            programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    EX3 = 1; // Enable INT3 interrupt.      //Enable INT3 interrupt
    EA = 1;

    while (1);
}
```

12.5.7 INT4 interrupt (falling edge), only falling edge interrupt is supported

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                    // see download software for header files
```

```
#include "intrins.h"

void INT4_Isr() interrupt 16
{
    P10 = !P10;                                   //test port
}

void main()
{
    EAXFR = 1;                                    //Enable access to XFR
    CKCON = 0x00;                                 //Set the external data bus speed to fastest
    WTST = 0x00;                                  //set the program code wait parameter.
                                                  //Assign a value of 0 to set the CPU to execute the
                                                  programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    EX4 = 1;                                      //Enable INT4 interrupt
    EA = 1;

    while (1);
}
```

12.5.8 Timer 0 Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                               // see download software for header files
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                   //test port
}

void main()
{
    EAXFR = 1;                                    //Enable access to XFR
    CKCON = 0x00;                                 //Set the external data bus speed to fastest
    WTST = 0x00;                                  //set the program code wait parameter.
                                                  //Assign a value of 0 to set the CPU to execute the
                                                  programme as fast as possible.

    p0m0 = 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
    EA = 1;

    while (1);
}
```

12.5.9 Timer 1 Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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 timer interrupt
    EA = 1;

    while (1);
}
```

12.5.10 Timer 2 Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files
#include "intrins.h"

void TM2_Isr() interrupt 12
{
    P10 = !P10;                             //test port
}

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set the external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign a value of 0 to set the CPU to execute the
                                            programme as fast as possible.

    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.
    T2R = 1;                                //start the timer
    ET2 = 1;                                //Enable timer interrupt
    EA = 1;

    while (1);
}
```

12.5.11 Timer 3 Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

void TM3_Isr() interrupt 19
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                                      //Set the external data bus speed to fastest
    WTST = 0x00;                                       //set the program code wait parameter.
                                                       //Assign a value of 0 to set the CPU to execute the
                                                       programme as fast as possible.

    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.
    T3R = 1;                            //start the timer
    ET3 = 1;                            //Enable timer interrupt
    EA = 1;

    while (1);
}
```

12.5.12 Timer 4 Interrupt

```
//Tested operating frequency is
11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // See download software for header files
#include "intrins.h"
```

```
void TM4_Isr() interrupt 20
{
    P10 = !P10.                 //Test port
```

```
}

void main()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign a value of 0 to set the CPU to execute the
                                programme as fast as possible.

    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.
    T4R = 1;                    //start the timer
    ET4 = 1;                    //Enable timer interrupt
    EA = 1;

    while (1);
}
```

12.5.13 UART1 Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"             // see download software for header files
#include "intrins.h"

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()
{
```

```
    eaxfr = 1; ckcon = 0x00; wtst = 0x00.        //Enable access to XFR
                                                 //Set external data bus speed to fastest
                                                 //Set the parameter for waiting for the programme code.
                                                 //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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;
    S1BRT = 1;
    T2x12 = 1.
    T2R = 1.                                     //Start timer
    ES = 1;                                      //Enable serial port interrupt
    ea = 1; sbuf = 0x5a.                         //Send test data

    while (1);
}
```

12.5.14 UART2 Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                              // see download software for header files
#include "intrins.h"

void UART2_Isr() interrupt 8
{
    if (S2TI)
    {
        S2TI = 0;                                //clear interrupt flag
        P12 = !                                  //test port
    }
    if (S2RI)
    {
        S2RI = 0;                                //clear interrupt flag
        P13 = !P13;                              //test port
    }
void main()
{
```

```
}

void main()
{
```

```
    eaxfr = 1; ckcon =                  //Enable access to XFR
    0x00; wtst = 0x00.                  //Set external data bus speed to fastest
                                        //Set the parameter for waiting for the programme code.
                                        //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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 = 0x50; t2l =
    0xe8.                               //65536-11059200/115200/4=0FFE8H
    T2H = 0xff.
    t2x12 = 1; t2r = 1.                 //Start timer
    ES2 = 1;                            //Enable serial port interrupt
    ea = 1; s2buf =
    0x5a.                               //Send test data

    while (1);
}
```

12.5.15 UART3 Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     //see download software for header files
#include "intrins.h"

void UART3_Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;                       //clear interrupt flag
        P12 = !                         //test port
    }
    if (S3RI)
    {
        S3RI = 0;                       //clear interrupt flag
        P13 = !P13;                     //test port
    }
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
```

```
    WTST = 0x00;                    //set the program code wait parameter.
```

```
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.
    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.
    t2x12 = 1; t2r = 1.                             //Start timer
    ES3 = 1;                                        //Enable serial port interrupt
    ea = 1; s3buf =
    0x5a.                                           //Send test data

    while (1);
}
```

12.5.16 UART4 Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                 // see download software for header files
#include "intrins.h"

void UART4_Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0;                                   // clear interrupt flag
        P12 = !                                     //test port
    }
    if (S4RI)
    {
        S4RI = 0;                                   // clear interrupt flag
        P13 = !P13;                                 //test port
    }
}

void main()
{
    EAXFR = 1;                                      //Enable access to XFR
    CKCON = 0x00;                                   //Set the external data bus speed to fastest
    WTST = 0x00;                                    //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.
```

```
    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.
    t2x12 = 1; t2r = 1.                     //Start timer
    ES4 = 1;                                //Enable serial port interrupt
    ea = 1; s4buf =
    0x5a.                                   //Send test data

    while (1);
}
```

12.5.17 ADC Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files
#include "intrins.h"

void ADC_Isr() interrupt 5
{
    ADC_FLAG = 0;                           // clear interrupt flag
    P0 = ADC_RES.                           //test port
    P2 = ADC_RESL.                          //test port
}

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set the external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign a value of 0 to set the CPU to execute the
                                            programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00.
```

```
    p5m0 = 0x00;
    p5m1 = 0x00.

    ADCCFG = 0x00;
    ADC_CONTR = 0xc0.                    //Enable and start the ADC module
    EADC = 1; //Enable ADC interrupt.    //Enable ADC interrupt
    EA = 1;

    while (1);
}
```

12.5.18 LVD Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                      // see download software for header files
#include "intrins.h"

#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

void LVD_Isr() interrupt 6
{
    LVDF = 0;                            // clear interrupt flag
    P10 = !P10;                          //test port
}

void main()
{
    EAXFR = 1;                           //Enable access to XFR
    CKCON = 0x00;                        //Set the external data bus speed to fastest
    WTST = 0x00;                         //set the program code wait parameter.
                                         //Assign a value of 0 to set the CPU to execute the
                                         programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    LVDF = 0;                            //Power up needs to clear the interrupt flag
    rstcfg = lvd3v0.                     //set LVD voltage to 3.0V
    ELVD = 1; //Enable LVD interrupt.    //Enable LVD interrupt
    EA = 1;
```

```
    while (1);
}
```

12.5.19 Comparator Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     //see download software for header files
#include "intrins.h"

void CMP_Isr() interrupt 21
{
    CMPIF = 0;                          //clear interrupt flag
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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;
    CMPEN = 1;                          //Enable comparator module
    PIE = NIE = 1;                      //enable comparator edge interrupt
    EA = 1;

    while (1);
}
```

12.5.20 SPI Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
```

```
#include "stc32g.h"                          //see download software for header files
#include "intrins.h"

void SPI_Isr() interrupt 9
{
    SPIF = 1;                                //clear interrupt flag
    P10 = !P10;                              //test port
}

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    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 host mode
    SPSTAT = 0xc0.                           //clear interrupt flag
    ESPI = 1; //Enable SPI interrupt.        //Enable SPI interrupt
    EA = 1;
    SPDAT = 0x5a;                            //Send test data

    while (1);
}
```

12.5.21 I2C interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          //see download software for header files
#include "intrins.h"

void I2C_Isr() interrupt 24
{
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40.                    //clear interrupt flag
        P10 = !P10;                          //test port
    }
}
```

```
void main()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign a value of 0 to set the CPU to execute the
                                programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    I2CCFG = 0xc0.              //Enable I2C host mode
    I2CMSCR = 0x80.             //Enable I2C interrupts; //Enable I2C interrupts; //Enable
    I2C interrupts.

    EA = 1;

    I2CMSCR = 0x81.             //send start command

    while (1);
}
```

13 Common I/O ports can be interrupted, not traditional external interrupts.

product lines	I/O Interrupt	I/O Interrupt Priority	I/O interrupt wake-up function
STC32G12K128 Series	●	**4 levels**	●
STC32G8K64 Series	●	**4 levels**	●
STC32F12K60 Series	●	**4 levels**	●

The STC32G series supports all I/O interrupts and four interrupt modes: falling edge interrupt, rising edge interrupt, low level interrupt, and high level interrupt. Each group of I/O ports has an independent interrupt entry address, and each I/O can set the interrupt mode independently.

Note: Do not use the falling-edge interrupt and rising-edge interrupt of the STC32G12K128-Beta version of the chip for the time being.

13.1 I/O Port Interrupt Related Registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	P0 Port Interrupt Enable Register	7EFD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 Port Interrupt Enable Register	7EFD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 Port Interrupt Enable Register	7EFD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 Port Interrupt Enable Register	7EFD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 Port Interrupt Enable Register	7EFD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 Port Interrupt Enable Register	7EFD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE	xx00,0000
P6INTE	P6 Port Interrupt Enable Register	7EFD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE	0000,0000
P7INTE	P7 Port Interrupt Enable Register	7EFD07H	P77INTE	P76INTE	P75INTE	P74INTE	P73INTE	P72INTE	P71INTE	P70INTE	0000,0000
P0INTF	P0 Port Interrupt Flag Register	7EFD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 Port Interrupt Flag Register	7EFD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 Port Interrupt Flag Register	7EFD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 Port Interrupt Flag	7EFD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000

	Register										
P4INTF	P4 Port Interrupt Flag Register	7EFD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 Port Interrupt Flag Register	7EFD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF	xx00,0000
P6INTF	P6 Port Interrupt Flag Register	7EFD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF	0000,0000
P7INTF	P7 Port Interrupt Flag Register	7EFD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF	0000,0000
P0IM0	P0 port interrupt mode register 0	7EFD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 port interrupt mode register 0	7EFD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 port interrupt mode register 0	7EFD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 port interrupt mode register 0	7EFD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P4IM0	P4 port interrupt mode register 0	7EFD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0	0000,0000
P5IM0	P5 port interrupt mode register 0	7EFD25H	-	-	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0	xx00,0000
P6IM0	P6 port interrupt mode register 0	7EFD26H	P67IM0	P66IM0	P65IM0	P64IM0	P63IM0	P62IM0	P61IM0	P60IM0	0000,0000
P7IM0	P7 port interrupt mode register 0	7EFD27H	P77IM0	P76IM0	P75IM0	P74IM0	P73IM0	P72IM0	P71IM0	P70IM0	0000,0000
P0IM1	P0 port interrupt mode register 1	7EFD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000

P1IM1	P1 port interrupt mode register 1	7EFD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 port interrupt mode register 1	7EFD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 port interrupt mode register 1	7EFD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P4IM1	P4 Port Interrupt Mode Register 1	7EFD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1	0000,0000
P5IM1	P5 Port Interrupt Mode Register 1	7EFD35H	-	-	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1	xx00,0000
P6IM1	P6 Port Interrupt Mode Register 1	7EFD36H	P67IM1	P66IM1	P65IM1	P64IM1	P63IM1	P62IM1	P61IM1	P60IM1	0000,0000
P7IM1	P7 Port Interrupt Mode Register 1	7EFD37H	P77IM1	P76IM1	P75IM1	P74IM1	P73IM1	P72IM1	P71IM1	P70IM1	0000,0000
PINIPL	I/O Port Interrupt Priority Low Register	7EFD40H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O Port Interrupt Priority High Register	7EFD41H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
P0WKUE	P0 Port Interrupt Wakeup Enable Register	7EFD42H	P07WKUE	P06WKUE	P05WKUE	P04WKUE	P03WKUE	P02WKUE	P01WKUE	P00WKUE	0000,0000
P1WKUE	P1 Port Interrupt Wakeup Enable Register	7EFD43H	P17WKUE	P16WKUE	P15WKUE	P14WKUE	P13WKUE	P12WKUE	P11WKUE	P10WKUE	0000,0000
P2WKUE	P2 Port Interrupt Wakeup Enable Register	7EFD44H	P27WKUE	P26WKUE	P25WKUE	P24WKUE	P23WKUE	P22WKUE	P21WKUE	P20WKUE	0000,0000
P3WKUE	P3 Port Interrupt Wakeup Enable Register	7EFD45H	P37WKUE	P36WKUE	P35WKUE	P34WKUE	P33WKUE	P32WKUE	P31WKUE	P30WKUE	0000,0000
P4WKUE	P4 Port Interrupt Wakeup Enable Register	7EFD46H	P47WKUE	P46WKUE	P45WKUE	P44WKUE	P43WKUE	P42WKUE	P41WKUE	P40WKUE	0000,0000
P5WKUE	P5 Port Interrupt Wakeup Enable Register	7EFD47H	-	-	P55WKUE	P54WKUE	P53WKUE	P52WKUE	P51WKUE	P50WKUE	xx00,0000
P6WKUE	P6 Port Interrupt Wakeup Enable Register	7EFD60H	P67WKUE	P66WKUE	P65WKUE	P64WKUE	P63WKUE	P62WKUE	P61WKUE	P60WKUE	0000,0000
P7WKUE	P7 Port Interrupt Wakeup Enable Register	7EFD61H	P77WKUE	P76WKUE	P75WKUE	P74WKUE	P73WKUE	P72WKUE	P71WKUE	P70WKUE	0000,0000

13.1.1 Port Interrupt Enable Register (PxINTE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	7EFD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	7EFD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	7EFD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	7EFD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P4INTE	7EFD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	7EFD05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE
P6INTE	7EFD06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE
P7INTE	7EFD07H	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

13.1.2 Port Interrupt Flag Register (PxINTF)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	7EFD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	7EFD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	7EFD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	7EFD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P4INTF	7EFD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF
P5INTF	7EFD15H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF
P6INTF	7EFD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF

P7INTF	7EFD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF

PnINTF.x: port interrupt request flag bit (n=0~7,
x=0~7) 0: no interrupt request on Pn.x port
1: There is an interrupt request on the Pn.x port, if the interrupt is enabled, it will enter the interrupt service programme. The flag bit needs to be cleared 0 by software.

13.1.3 Port interrupt mode configuration registers (PxIM0, PxIM1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0IM0	7EFD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0
P0IM1	7EFD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1
P1IM0	7EFD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0
P1IM1	7EFD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1
P2IM0	7EFD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0
P2IM1	7EFD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1
P3IM0	7EFD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0
P3IM1	7EFD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1
P4IM0	7EFD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0
P4IM1	7EFD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1
P5IM0	7EFD25H	-	-	P55IM0	P54IM0	P53IM0	P52IM0	P51IM0	P50IM0
P5IM1	7EFD35H	-	-	P55IM1	P54IM1	P53IM1	P52IM1	P51IM1	P50IM1
P6IM0	7EFD26H	P67IM0	P66IM0	P65IM0	P64IM0	P63IM0	P62IM0	P61IM0	P60IM0
P6IM1	7EFD36H	P67IM1	P66IM1	P65IM1	P64IM1	P63IM1	P62IM1	P61IM1	P60IM1
P7IM0	7EFD27H	P77IM0	P76IM0	P75IM0	P74IM0	P73IM0	P72IM0	P71IM0	P70IM0
P7IM1	7EFD37H	P77IM1	P76IM1	P75IM1	P74IM1	P73IM1	P72IM1	P71IM1	P70IM1

Configure the mode of the port

PnIM1.x	PnIM0.x	Pn.x port interrupt mode
0	0	Falling edge interrupt
0	1	Rising edge interrupt
1	0	low level interrupt (computing)
1	1	High Level Interrupt

13.1.4 Port Interrupt Priority Control Registers (PINIPL, PINIPH)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PINIPL	7EFD60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP
PINIPH	7EFD61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH

PxIPH,PxIP: Px port interrupt priority
control bit 00: Px port interrupt

priority is level 0 (lowest level)
01: Px port interrupt priority 1 (lower level)
10: Px port interrupt priority 2 (higher level)
11: Px port interrupt priority 3 (highest level)

13.1.5 Port interrupt power-down wake-up enable register (PxWKUE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P0WKUE	7EFD40H	P07WKUE	P06WKUE	P05WKUE	P04WKUE	P03WKUE	P02WKUE	P01WKUE	P00WKUE
P1WKUE	7EFD41H	P17WKUE	P16WKUE	P15WKUE	P14WKUE	P13WKUE	P12WKUE	P11WKUE	P10WKUE
P2WKUE	7EFD42H	P27WKUE	P26WKUE	P25WKUE	P24WKUE	P23WKUE	P22WKUE	P21WKUE	P20WKUE
P3WKUE	7EFD43H	P37WKUE	P36WKUE	P35WKUE	P34WKUE	P33WKUE	P32WKUE	P31WKUE	P30WKUE
P4WKUE	7EFD44H	P47WKUE	P46WKUE	P45WKUE	P44WKUE	P43WKUE	P42WKUE	P41WKUE	P40WKUE
P5WKUE	7EFD45H	-	-	P55WKUE	P54WKUE	P53WKUE	P52WKUE	P51WKUE	P50WKUE
P6WKUE	7EFD46H	P67WKUE	P66WKUE	P65WKUE	P64WKUE	P63WKUE	P62WKUE	P61WKUE	P60WKUE
P7WKUE	7EFD47H	P77WKUE	P76WKUE	P75WKUE	P74WKUE	P73WKUE	P72WKUE	P71WKUE	P70WKUE

PnxWKUE: port interrupt power-down wake-up enable control bit (n=0~7, x=0~7) 0: disable Pn.x port interrupt power-down wake-up function

1: Enable Wake-on-Disconnect function in Pn.x port

13.2 sample procedure

13.2.1 P0 port falling edge interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                                         //Enable access to XFR
    CKCON = 0x00;                                      //Set the external data bus speed to fastest
    WTST = 0x00;                                       //set the program code wait parameter.
                                                       //Assign a value of 0 to set the CPU to execute the
                                                       programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    P0IM0 = 0x00.                                      //falling edge interrupt
    P0IM1 = 0x00.
    P0INTE = 0xff; //Enable P0 interrupt.              // Enable P0 interrupt

    EA = 1;

    while (1);
}

//Cannot be compiled directly in KEIL because the interrupt vector is greater than 31.
//Must borrow interrupt entry address 13.
void common_isr() interrupt 13
{
    unsigned char intf.

    intf = P0INTF;
    if (intf)
    {
        P0INTF = 0x00.                                 {
        if (intf & 0x01)
        {

        }
        if (intf & 0x02)
```

//P0.0 port

interrupt

```
            }                                                   //P0.1 port interrupt
            if (intf & 0x04)
            {

            }                                                   //P0.2 port interrupt
            if (intf & 0x08)
            {

            }                                                   //P0.3 port interrupt
            if (intf & 0x10)
            {

            }                                                   //P0.4 port interrupt
            if (intf & 0x20)
            {

            }                                                   //P0.5 port interrupt
            if (intf & 0x40)
            {

            }                                                   //P0.6 port interrupt
            if (intf & 0x80)
            {

            }                                                   //P0.7 port interrupt
        }
}

// ISR.ASM
// Save the following code as ISP.ASM and just add the file to the project

            CSEG        AT 012BH            P0 port interrupt entry address
            JMP         P0INT_ISR
P0INT_ISR.
            JMP         006BH               Borrow the entry address of interrupt 13.
            END
```

13.2.2 P1 port rising edge interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                             // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                                                  //Enable access to XFR
    CKCON = 0x00;                                               //Set the external data bus speed to fastest
    WTST = 0x00;                                                //set the program code wait parameter.
                                                                //Assign a value of 0 to set the CPU to execute the
                                                                programme as fast as possible.

    P0M0 = 0x00.
```

```
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    P1IM0 = 0xff;                                   //Rising edge interrupt
    P1IM1 = 0x00;
    P1INTE = 0xff; //Enable P1 interrupt.           //Enable P1 interrupt

    EA = 1;

    while (1);
}

//Cannot be compiled directly in KEIL because the interrupt vector is greater than 31.
//Must borrow interrupt entry address 13.
void common_isr() interrupt 13
{
    unsigned char intf.

    intf = P1INTF;
    if (intf)
    {
        P1INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P1.0 port interrupt
        }
        if (intf & 0x02)
        {
                                                    //P1.1 port interrupt
        }
        if (intf & 0x04)
        {
                                                    //P1.2 port interrupt
        }
        if (intf & 0x08)
        {
                                                    //P1.3 port interrupt
        }
        if (intf & 0x10)
        {
                                                    //P1.4 port interrupt
        }
        if (intf & 0x20)
        {
                                                    //P1.5 port interrupt
        }
        if (intf & 0x40)
        {
                                                    //P1.6 port interrupt
        }
```

```
            if (intf & 0x80)
            {
                                                    //P1.7 port interrupt
            }
        }
}

// ISR.ASM
// Save the following code as ISP.ASM and just add the file to the project

            CSEG        AT 0133H                    P1 port interrupt entry address
            JMP         P1INT_ISR
P1INT_ISR.
            JMP         006BH                       Borrow the entry address of interrupt 13.
            END
```

13.2.3 P2 port low level interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                 // see download software for header files
#include "intrins.h"

void main()
{
        EAXFR = 1;                                  //Enable access to XFR
        CKCON = 0x00;                               //Set the external data bus speed to fastest
        WTST = 0x00;                                //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.

        p0m0 = 0x00;
        p0m1 = 0x00;
        p1m0 = 0x00;
        p1m1 = 0x00;
        p2m0 = 0x00;
        p2m1 = 0x00;
        p3m0 = 0x00;
        p3m1 = 0x00;
        p4m0 = 0x00;
        p4m1 = 0x00;
        p5m0 = 0x00;
        p5m1 = 0x00.

        P2IM0 = 0x00.                               //Low level interrupt
        P2IM1 = 0xff;
        P2INTE = 0xff; //Enable P2 interrupt.       //Enable P2 interrupt

        EA = 1;

        while (1);
}

//Cannot be compiled directly in KEIL because the interrupt vector is greater than 31.
```

```
//Must borrow interrupt entry address 13.
void common_isr() interrupt 13
{
    unsigned char intf.

    intf = P2INTF;
    if (intf)
    {
        P2INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P2.0 port interrupt

        }
        if (intf & 0x02)
        {
                                                    //P2.1 port interrupt

        }
        if (intf & 0x04)
        {
                                                    //P2.2 port interrupt

        }
        if (intf & 0x08)
        {
                                                    //P0.3 port interrupt

        }
        if (intf & 0x10)
        {
                                                    //P2.4 port interrupt

        }
        if (intf & 0x20)
        {
                                                    //P2.5 port interrupt

        }
        if (intf & 0x40)
        {
                                                    //P2.6 port interrupt

        }
        if (intf & 0x80)
        {
                                                    //P2.7 port interrupt

        }
    }
}

// ISR.ASM
// Save the following code as ISP.ASM and just add the file to the project

            CSEG        AT 013BH                    P2 port interrupt entry address
            JMP         P2INT_ISR
P2INT_ISR.
            JMP         006BH                       Borrow the entry address of interrupt 13.
            END
```

13.2.4 P3 port high interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                   // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                                        //Enable access to XFR
    CKCON = 0x00;                                     //Set the external data bus speed to fastest
    WTST = 0x00;                                      //set the program code wait parameter.
                                                      //Assign a value of 0 to set the CPU to execute the
                                                      programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    P3IM0 = 0xff;                                     //high level interrupt
    P3IM1 = 0xff;
    P3INTE = 0xff; //Enable P3 port interrupt.        //Enable P3 interrupt

    EA = 1;

    while (1);
}

//Cannot be compiled directly in KEIL because the interrupt vector is greater than 31.
//Must borrow interrupt entry address 13.
void common_isr() interrupt 13
{
    unsigned char intf.

    intf = P3INTF;
    if (intf)
    {
        P3INTF = 0x00;
        if (intf & 0x01)
        {
                                                      //P3.0 port interrupt
        }
        if (intf & 0x02)
        {
                                                      //P3.1 Port Interrupt
        }
        if (intf & 0x04)
        {
```

```
                                                    //P3.2 Port Interrupt
        }
        if (intf & 0x08)
        {
                                                    //P3.3 Port Interrupt
        }
        if (intf & 0x10)
        {
                                                    //P3.4 Port Interrupt
        }
        if (intf & 0x20)
        {
                                                    //P3.5 Port Interrupt
        }
        if (intf & 0x40)
        {
                                                    //P3.6 Port Interrupt
        }
        if (intf & 0x80)
        {
                                                    //P3.7 Port Interrupt
        }
    }
}

// ISR.ASM
// Save the following code as ISP.ASM and just 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 interrupt 13.
            END
```

14 Timer/Counter (24-bit Timer, 8-bit Prescaler) (+16-bit auto-reload)

STC32G series microcontrollers have five 24-bit timers/counters (8-bit prescaler + 16-bit counter). 5 16-bit timers T0, T1, T2, T3 and T4 have both counting and timing modes of operation. For timer/counter T0 and T1, their corresponding control bits C/T in special function register TMOD are used to select whether T0 or T1 is a timer or a counter. For Timer/Counter T2, the control bit T2_C/T in Special Function Register AUXR is used to select whether T2 is a timer or counter. For Timer/Counter T3, use control bit T3_C/T in Special Function Register T4T3M to select whether T3 is a timer or counter. For Timer/Counter T4, the control bit T4_C/T in Special Function Register T4T3M is used to select whether T4 is a timer or a counter. The core component of the timer/counter is an addition counter, which essentially counts pulses. Only the source of counting pulses is different: if the counting pulses come from the system clock, then it is the timing mode, in this case, the timer/counter will get a counting pulse every 12 clocks or every 1 clock, and the counting value will be added 1; if the counting pulses come from the external pins of the microcontroller, then it is the counting mode, and the counting value will be added 1 for every incoming pulse.

When Timer/Counter T0, T1 and T2 are working in Timing mode, T0x12, T1x12 and T2x12 in Special Function Register AUXR determine whether it is System Clock/12 or System Clock/1 (undivided) for T0, T1 and T2 to count respectively. When Timer/Counter T3 and T4 are working in Timing mode, T3x12 and T4x12 in Special Function Register T4T3M determine whether it is System Clock/12 or System Clock/1 (undivided) for T3 and T4 to count respectively. When the timer/counter operates in count mode, the external pulse count is not divided into frequencies.

Timer/Counter 0 has 4 working modes: Mode 0 (16-bit auto-reload mode), Mode 1 (16-bit non-reloadable mode), Mode 2 (8-bit auto-reload mode), and Mode 3 (16-bit auto-reload mode without interrupt masking). Timer1 operates in the same modes as Timer0 except for Mode 3. T1 is invalid in Mode 3 and stops counting. The working mode of Timer T2 is fixed to 16-bit auto-reload mode, T2 can be used as timer, baud rate generator of serial port and programmable clock output. T2 can be used as timer, baud rate generator of serial port and programmable clock output.

14.1 Timer related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	Timer Control Register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	Timer Mode Register	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	Timer 0 Low 8-bit Register	8AH									0000,0000
TL1	Timer 1 Low 8-bit Register	8BH									0000,0000
TH0	Timer 0 High 8-bit Register	8CH									0000,0000

TH1	Timer 1 High 8-bit Register	8DH									0000,0000
AUXR	Auxiliary Register 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT	0000,0001
INTCLKO	Interrupt and Clock Output Control Registers	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	Power-down wake-up timer low byte	AAH									1111,1111
WKTCH	Power-down wake-up timer high byte	ABH	WKTEN								0111,1111
T4T3M	Timer 4/3 Control Register	DDH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	Timer 4 High Byte	D2H									0000,0000
T4L	Timer 4 Low Byte	D3H									0000,0000

T3H	Timer 3 High Byte	D4H		0000,0000
T3L	Timer 3 Low Byte	D5H		0000,0000
T2H	Timer 2 High Byte	D6H		0000,0000
T2L	Timer 2 Low Byte	D7H		0000,0000

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
TM0PS	Timer 0 Clock prescaler register	7EFEA0H									0000,0000
TM1PS	Timer 1 Clock prescaler register	7EFEA1H									0000,0000
TM2PS	Timer 2 Clock Preshunt Register	7EFEA2H									0000,0000
TM3PS	Timer 3 Clock Preshunt Register	7EFEA3H									0000,0000
TM4PS	Timer 4 Clock Preshunt Register	7EFEA4H									0000,0000

14.2 Timer 0/1

14.2.1 Timer 0/1 Control Register (TCON)

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

TF1: T1 overflow interrupt flag: after T1 is allowed to count, counting starts from the initial value plus 1. When overflow is generated by the hardware TF1 position **"1",** and request an interrupt to the CPU, until the CPU responds to the interrupt, then the hardware clear **"0"** (can also be queried by the software clear **"0")**. "). TR1: operation control bit of Timer T1. This bit is set and cleared by software. When GATE (TMOD.7) = 0 and TR1 = 1 it allows T1 to start counting, and TR1 = 0 prohibits T1 from counting. T1 is allowed to count when GATE (TMOD.7) = 1, TR1 = 1 and INT1 is input high.

Count.

TF0: T0 overflow interrupt flag, T0 is allowed to count, from the initial value of the beginning of the count plus 1, when the overflow is generated by the hardware set **"1" TF0,** request an interrupt to the CPU, has been to keep the CPU respond to the interrupt, only by the hardware to clear the 0 (can also be queried by the software to clear the 0).

TR0: Operation control bit of timer T0. This bit is set and cleared by software. It allows T0 to start counting when GATE (TMOD.3) = 0 and TR0 = 1, and prohibits T0 from counting when TR0 = 0. T0 is allowed to count when GATE (TMOD.3) = 1, TR0 = 1 and INT0 input is high, and T0 is prohibited when TR0 = 0.

IE1: external interrupt 1 request source (INT1/P3.3) flag. IE1=1, the external interrupt requests an interrupt from the CPU, and when the CPU responds to the interrupt, the hardware clears "0" IE1.

IT1: external interrupt source 1 trigger control bit. IT1=0, either rising edge or falling edge can trigger external interrupt 1. IT1=1, external interrupt 1 is programmed to be triggered by falling edge.

IE0: external interrupt 0 request source (INT0/P3.2) flag. IE0=1 external interrupt 0 request interrupt from CPU, when CPU responds to the external interrupt, the hardware clears "0" IE0 (edge-triggered mode).

IT0: external interrupt source 0 trigger control bit. IT0=0, either rising edge or falling edge can trigger external interrupt 0. IT0=1, external interrupt 0 is programmed to be triggered by falling edge.

14.2.2 Timer 0/1 Mode Register (TMOD)

notation	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: Controls Timer1, set to 1 to open Timer/Counter1 only when INT1 pin is high and TR1 control position 1. T0_GATE: Controls Timer0, set to 1 to open Timer/Counter0 only when INT0 pin

is high and TR0 control position 1. T1_C/T: Controls Timer1 to be used as a timer or counter, clearing it to be used as a Timer (to count the internal system clock), set 1 to use as Counter (counts external pulses on pins T1/P3.5).

T0_C/T: Controls Timer 0 to be used as a timer or counter, clear 0 to be used as a timer (counting the internal system clock), and set 1 to be used as a counter (counting external pulses on pins T0/P3.4).

T1_M1/T1_M0: Timer Timer/Counter 1 mode selection

T1_M1	T1_M0	Timer/Counter 1 Operating Mode
0	0	16-bit auto-reload mode When the 16-bit count value in [TH1,TL1] overflows, the system will automatically set the internal 16-bit

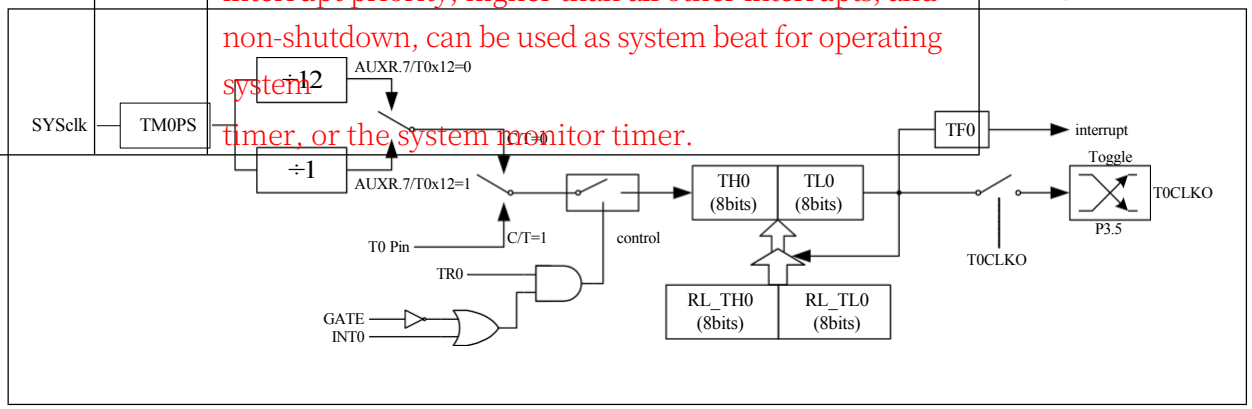
		The overloaded value in the overload register is loaded into [TH1,TL1].
0	1	16-bit not auto-reload mode When the 16-bit count value in [TH1,TL1] overflows, Timer 1 will start counting from 0
1	0	8-bit auto-reload mode When the 8-bit count value in TL1 overflows, the system will automatically reload the value in TH1 Load into TL1.
1	1	T1 stops working

T0_M1/T0_M0: Timer Timer/Counter 0 mode selection

T0_M1	T0_M0	Timer/Counter 0 Operating Mode
0	0	16-bit auto-reload mode When the 16-bit count value in [TH0,TL0] overflows, the system will automatically set the internal 16-bit The overloaded value in the overload register is loaded into [TH0,TL0].
0	1	16-bit not auto-reload mode When the 16-bit count value in [TH0,TL0] overflows, Timer 0 will start counting from 0
1	0	8-bit auto-reload mode When the 8-bit count value in TL0 overflows, the system will automatically reload the value in TH0 Load into TL0.
1	1	16-bit auto-reload mode with non-maskable interrupts Same as mode 0, non-maskable interrupt, highest interrupt priority, higher than all other interrupts, and non-shutdown, can be used as system beat for operating system timer, or the system monitor timer.

14.2.3 Timer 0 Mode 0 (16-bit auto-reload mode)

This mode uses Timer/Counter 0 as a 16-bit counter that can be reloaded automatically, as shown below:



Timer/Counter 0 Mode 0: 16-bit Auto-Reload Mode

When GATE=0 (TMOD.3), the timer counts if TR0=1. GATE=1 allows Timer 0 to be controlled by external input INT0, which enables pulse width measurement. TR0 is a control bit within the TCON

register, and the specific function description of each TCON register bit is described in the previous section on TCON register.

When C/T=0, the multiswitch is connected to the crossover output of the system clock, T0 counts the internal system clock, and T0 operates in the timing mode. When

When C/T=1, the multiswitch is connected to the external pulse input P3.4/T0, i.e. T0 operates in counting mode.

Timer 0 of STC microcontroller has two counting rates: one is 12T mode, adding 1 every 12 clocks, the same as the traditional 8051 microcontroller; the other is 1T mode, adding 1 every clock, which is 12 times faster than the traditional 8051 microcontroller.The rate of T0 is determined by T0x12 in the Special Function Register AUXR, if T0x12=0, T0 works in 12T mode; if T0x12=1, T0 works in 1T mode; if T0x12=1, T0 works in 1T mode; if T0x12=1, T0 works in 12T mode; if T0x12=1, T0 works in 1T mode. If T0x12=0, T0 works in 12T mode; if T0x12=1, T0 works in 1T mode.

Timer 0 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, Timer/Counter 0 is disabled, the content written to TL0 will be written to RL_TL0, and the content written to TH0 will be written to RL_TH0, and when TR0=1, Timer/Counter 0 is allowed to work, the content written to TL0 is not actually written to the current register TL0 but to the hidden register RL_TL0, and the content written to TH0 will be written to RL_TL0, and the content written to TH0 will be written to RL_TL0, and the content written to TH0 will be written to RL_TL0. When TL0 is allowed to work, the content written to TL0 is not actually written to the current register TL0, but to the hidden register RL_TH0, so that the 16-bit reload timer can be realised cleverly. When reading the contents of TH0 and TL0, what is read is the contents of TH0 and TL0, not the contents of RL_TH0 and RL_TL0.

When Timer 0 operates 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] into [TH0,TL0].

When T0CLKO/INT_CLKO.0=1, pin P3.5/T1 is configured as clock output T0CLKO for Timer 0. The output clock frequency is T0

Spillover rate/2.

If C/T=0, timer/counter T0 counts the internal system clock:

Output clock frequency when T0 is operating in 1T mode (AUXR.7/T0x12=1) = (SYSclk)/(TM0PS+1)/(65536-[RL_TH0, RL_TL0])/2

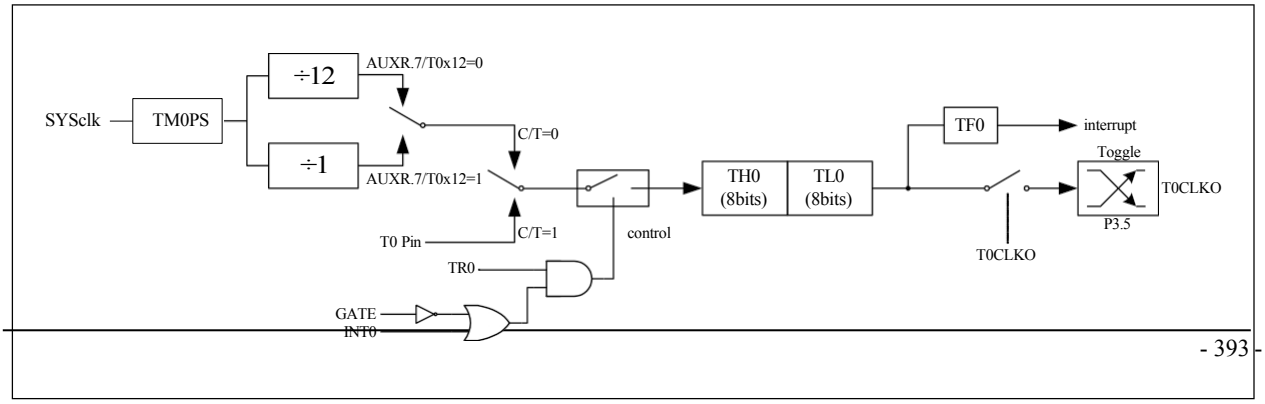
Output clock frequency when T0 is operating in 12T mode (AUXR.7/T0x12=0) = (SYSclk)/(TM0PS+1)/12/(65536-[RL_TH0, RL_TL0])/2

If C/T=1, timer/counter T0 is counting against the external pulse input (P3.4/T0):

Output clock frequency = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

14.2.4 Timer 0 Mode 1 (16-bit non-reloadable mode)

In this mode, Timer/Counter 0 operates in 16-bit non-reloadable mode as shown below:



Mode 1 of Timer/Counter 0: 16-bit Non-Reloadable Mode

In this mode, Timer/Counter 0 is configured in 16-bit non-reloadable mode and consists of 8 bits of TL0 and 8 bits of TH0.

An 8-bit overflow is scored to TH0, and TH0 counts the overflow to set the overflow flag bit TF0 in TCON.

When GATE=0 (TMOD.3), the timer counts if TR0=1. GATE=1 allows Timer 0 to be controlled by the external input INT0, which enables pulse width measurement. TR0 is a control bit in the TCON register, and the specific function description of each bit of the TCON register is described in the previous section on the TCON register.

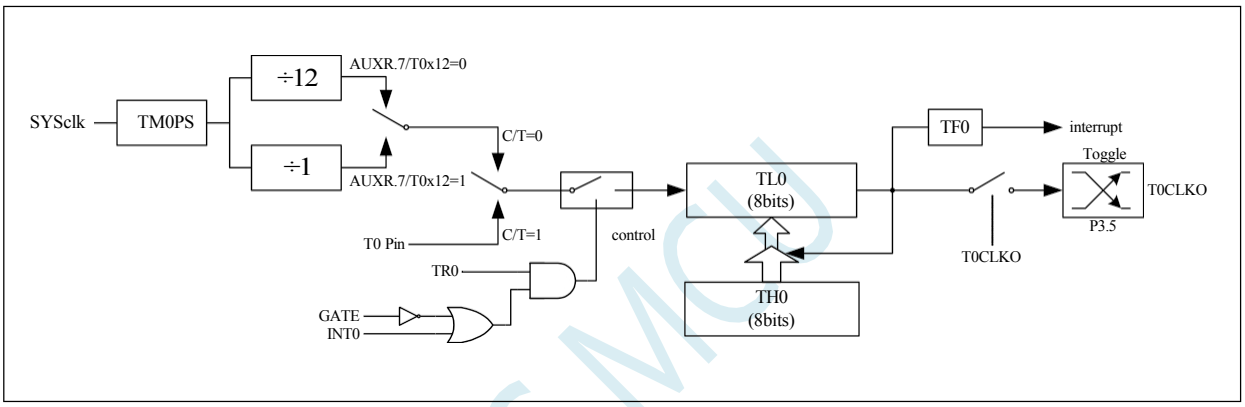
When C/T=0, the multiswitch is connected to the crossover output of the system clock, T0 counts the internal system clock, and T0 operates in the timing mode. When

When C/T=1, the multiswitch is connected to the external pulse input P3.4/T0, i.e. T0 operates in counting mode.

Timer 0 of STC microcontroller has two counting rates: one is 12T mode, adding 1 every 12 clocks, the same as the traditional 8051 microcontroller; and the other is 1T mode, adding 1 every clock, which is 12 times faster than the traditional 8051 microcontroller.The rate of T0 is determined by T0x12 in the special function register AUXR, and T0 works in 12T mode if T0x12=0; if T0x12=1, T0 works in 1T mode. The rate of T0 is determined by T0x12 in special function register AUXR.

14.2.5 Timer 0 Mode 2 (8-bit Auto-Reload Mode)

This mode uses Timer/Counter 0 as an 8-bit counter that can be reloaded automatically, as shown below:



Mode 2 for Timer/Counter 0: 8-bit Auto-Reload 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 the software, and when reloading, the content of TH0 is not

Change.

When T0CLKO/INT_CLKO.0=1, pin P3.5/T1 is configured as clock output T0CLKO for Timer 0. The output clock frequency is T0

Spillover rate/2.

If C/T=0, timer/counter T0 counts the internal system clock:

Output clock frequency when T0 is operating in 1T mode (AUXR.7/T0x12=1) = (SYSclk)/(TM0PS+1)/(256-TH0)/2

Output clock frequency when T0 is operating in 12T mode (AUXR.7/T0x12=0) = (SYSclk)/(TM0PS+1)/12/(256-TH0)/2

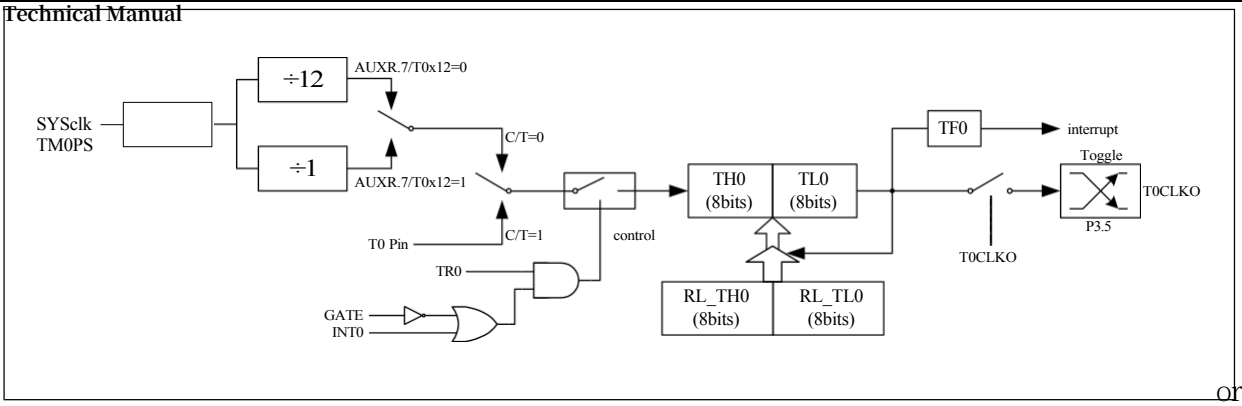
If C/T=1, timer/counter T0 is counting the external pulse input (P3.4/T0):

Output clock frequency = (T0_Pin_CLK) / (256-TH0)/2

14.2.6 Timer 0 Mode 3 (non-maskable interrupt 16-bit auto-reload, real-time OS metronome)

For Timer/Counter 0, the operating mode 3 is the same as the operating mode 0 (the schematic diagram of Timer Mode 3, below, is the same as the operating mode 0).

(Mode 0 is the same). The only difference is that when Timer/Counter 0 is working in mode 3, only ET0/IE.1 (Timer/Counter 0 interrupt allow bit) and not EA/IE.7 (general interrupt enable bit) is needed to open the interrupt of Timer/Counter 0. The Timer/Counter 0 interrupt in this mode has nothing to do with the general interrupt enable bit EA, and once Timer/Counter 0 interrupt is opened (ET0=1) in mode 3, the interrupt is unmaskable, i.e., the interrupt has the highest priority, i.e., the interrupt cannot be blocked by any interrupt. Once the Timer/Counter 0 interrupt working in mode 3 is turned on (ET0=1), the interrupt is unmaskable and has the highest priority, i.e., the interrupt cannot be interrupted by any interrupt, and after the interrupt is turned on, it is not controlled by either EA/IE.7 or ET0, and cannot be masked when EA=0 or ET0=0. Therefore, this mode is called the 16-bit auto-reload mode with non-maskable interrupt.



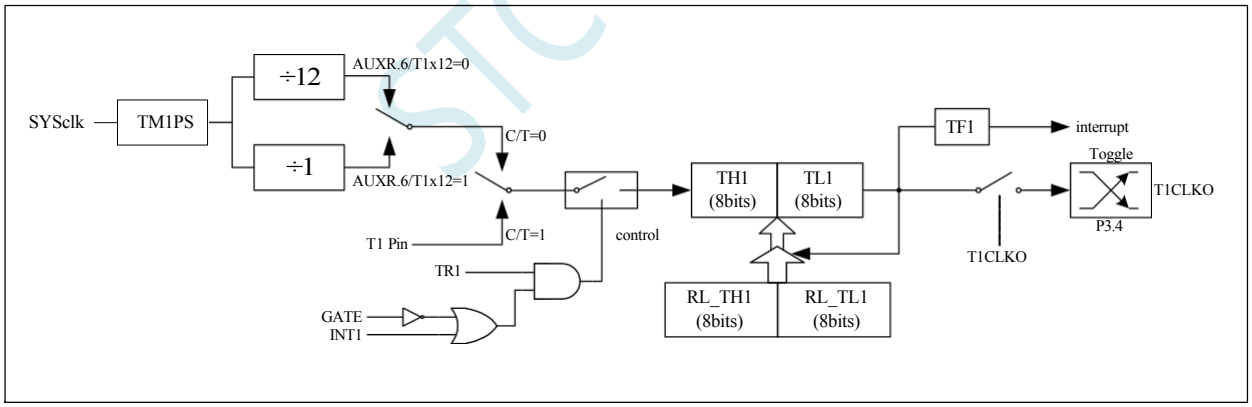
order

Mode 3 for Timer/Counter 0: 16-bit auto-reload mode without interrupt masking

Note: When Timer0 is operating in Mode 3 (16-bit auto-reload mode with non-maskable interrupts), Timer0 interrupts can be turned on by allowing ET0/IE.1 (Timer0 Interrupt Allow Bit) instead of EA/IE.7 (General Interrupt Enable Bit), the Timer0 interrupts in this mode have nothing to do with General Interrupt Enable Bit EA. Once the Timer0 interrupt is turned on in this mode, the Timer0 interrupt has the highest priority and cannot be interrupted by any other interrupts (neither interrupts with a lower priority than the Timer0 interrupt nor interrupts with a higher priority than the Timer0 interrupt can interrupt the Timer0 interrupt at this point in time), and after the interrupt is turned on, it is not under the control of EA/IE.7 nor ET0 anymore. The interrupt is neither controlled by EA/IE.7 nor ET0 after it is turned on, and clearing EA or ET0 cannot close this interrupt.

14.2.7 Timer 1 Mode 0 (16-bit auto-reload mode)

This mode uses Timer/Counter 1 as a 16-bit counter that can be reloaded automatically, as shown below:



Mode 0 for Timer/Counter 1: 16-bit Auto-Reload Mode

When GATE=0 (TMOD.7), the timer counts if TR1=1. GATE=1 allows Timer 1 to be controlled by the external input INT1, which enables pulse width measurement. TR1 is a control bit within the TCON register, and the specific function description of each TCON register bit is described in the previous section on TCON register.

When C/T=0, the multiswitch is connected to the crossover output of the system clock, T1 counts the internal system clock, and T1 operates in the timing mode. When

When C/T=1, the multiswitch is connected to the external pulse input P3.5/T1, i.e. T1 operates in counting mode.

Timer 1 of STC microcontroller has two counting rates: one is 12T mode, adding 1 every 12

clocks, the same as the traditional 8051 microcontroller; the other is 1T mode, adding 1 every clock, which is 12 times faster than the traditional 8051 microcontroller.The rate of T1 is determined by the T1x12 in the Special Function Register AUXR, if T1x12=0, T1 operates in 12T mode; if T1x12=1, T1 operates in 1T mode; if T1x12=1, T1 operates in 1T mode. If T1x12=0, T1 works in 12T mode; if T1x12=1, T1 works in 1T mode.

Timer 1 has two hidden registers, RL_TH1 and RL_TL1. RL_TH1 shares the same address as TH1, and RL_TL1 shares the same address as TL1.

The same address is shared. When TR1=0, i.e. timer/counter 1 is disabled, writes to TL1 are written to RL_TL1 at the same time, and writes to

When TR1=1, i.e. timer/counter 1 is allowed to work, the content written to TL1 is actually written to the hidden register RL_TL1 instead of the current register TL1, and the content written to TH1 is actually written to the hidden register RL_TH1 instead of the current register TH1, so that the 16-bit reload timer can be cleverly implemented. This is a clever way to implement the 16-bit reload timer. When reading the contents of TH1 and TL1, what is read is the contents of TH1 and TL1, not the contents of RL_TH1 and RL_TL1.

When Timer 1 operates in Mode 1 (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] into [TH1,TL1].

When T1CLKO/INT_CLKO.1=1, pin P3.4/T0 is configured as clock output T1CLKO for Timer 1. the output clock frequency is T1

Spillover rate/2.

If C/T=0, timer/counter T1 counts the internal system clock:

Output clock frequency when T1 is operating in 1T mode (AUXR.6/T1x12=1) = (SYSclk)/(TM1PS+1)/(65536-[RL_TH1, RL_TL1])/2

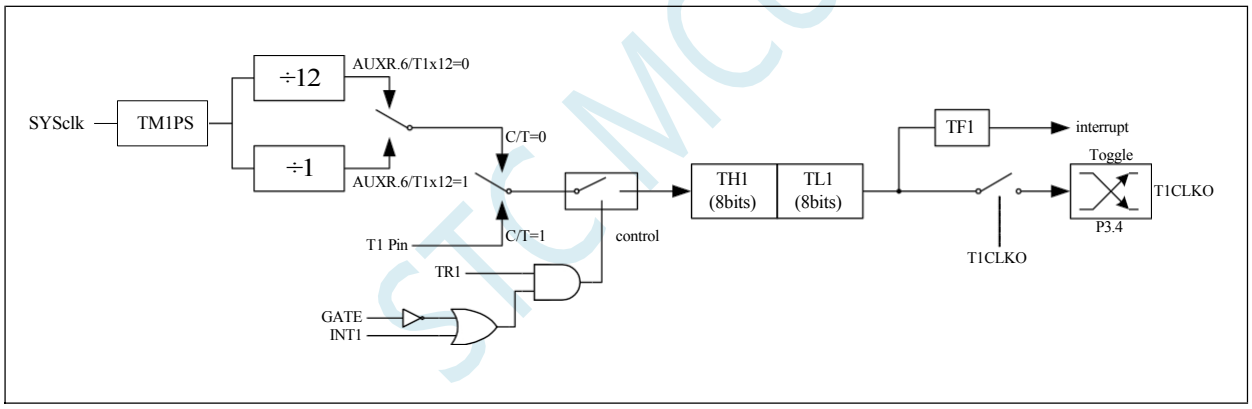
Output clock frequency when T1 is operating in 12T mode (AUXR.6/T1x12=0) = (SYSclk)/(TM1PS+1)/12/(65536-[RL_TH1, RL_TL1])/2

If C/T=1, timer/counter T1 is counting against an external pulse input (P3.5/T1):

Output clock frequency = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

14.2.8 Timer 1 Mode 1 (16-bit non-reloadable mode)

In this mode, Timer/Counter 1 operates in 16-bit non-reloadable mode as shown below:



Mode 1 of Timer/Counter 1: 16-bit Non-Reloadable Mode

In this mode, Timer/Counter 1 is configured in 16-bit non-reloadable mode, consisting of 8 bits in TL1 and 8 bits in TH1.

An 8-bit overflow is scored to TH1, and TH1 counts the overflow to set the overflow flag bit TF1 in TCON.

When GATE=0 (TMOD.7), the timer counts if TR1=1. GATE=1 allows Timer 1 to be controlled by the external input INT1, so that pulse width measurement can be realised. TR1 is a control bit in the TCON register, and the specific function description of each bit of the TCON register is shown in the previous section of TCON register.

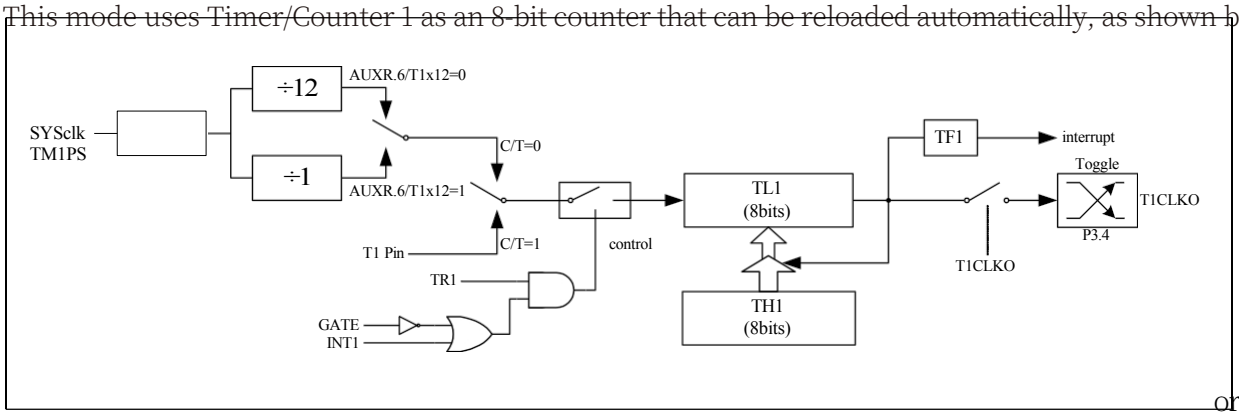
When C/T=0, the multiswitch is connected to the crossover output of the system clock, T1 counts the internal system clock, and T1 operates in the timing mode. When

When C/T=1, the multiswitch is connected to the external pulse input P3.5/T1, i.e. T1 operates in counting mode.

Timer 1 of STC microcontroller has two counting rates: one is 12T mode, adding 1 every 12 clocks, the same as the traditional 8051 microcontroller; the other is 1T mode, adding 1 every clock, which is 12 times faster than the traditional 8051 microcontroller.The rate of T1 is determined by the T1x12 in the Special Function Register AUXR, if T1x12=0, T1 operates in 12T mode; if T1x12=1, T1 operates in 1T mode; if T1x12=1, T1 operates in 1T mode. If T1x12=0, T1 works in 12T mode; if T1x12=1, T1 works in 1T mode.

14.2.9 Timer 1 Mode 2 (8-bit auto-reload mode)

This mode uses Timer/Counter 1 as an 8-bit counter that can be reloaded automatically, as shown below:



order

Mode 2 of Timer/Counter 1: 8-bit Auto-Reload 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 the software, and when reloading, the content of TH1 is not

Change.

When T1CLKO/INT_CLKO.1=1, pin P3.4/T0 is configured as clock output T1CLKO for Timer 1. the output clock frequency is T1

Spillover rate/2.

If C/T=0, timer/counter T1 counts the internal system clock:

Output clock frequency when T1 is operating in 1T mode (AUXR.6/T1x12=1) = (SYSclk)/(TM1PS+1)/(256-TH1)/2

Output clock frequency when T1 is operating in 12T mode (AUXR.6/T1x12=0) = (SYSclk)/(TM1PS+1)/12/(256-TH1)/2

If C/T=1, timer/counter T1 is counting against an external pulse input (P3.5/T1):

Output clock frequency = (T1_Pin_CLK) / (256-TH1)/2

14.2.10 Timer 0 Count Register (TL0, TH0)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

When Timer/Counter 0 operates in 16-bit mode (Mode 0, Mode 1, Mode 3), TL0 and TH0 combine to become a 16-bit register.

TL0 is the low byte and TH0 is the high byte. For 8-bit mode (mode 2), TL0 and TH0 are two separate 8-bit registers.

14.2.11 Timer 1 Count Register (TL1, TH1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

When Timer/Counter 1 operates in 16-bit mode (Mode 0, Mode 1), TL1 and TH1 combine to

become one 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.

14.2.12 Auxiliary Register 1 (AUXR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT

T0x12: Timer 0 speed control bit

0: 12T mode, i.e. CPU clock 12 divisions (FOSC/12)

1: 1T mode, i.e., CPU clock without frequency division divider (FOSC/1)

T1x12: Timer 1 speed control bit

0: 12T mode, i.e. CPU clock 12 divisions (FOSC/12)

1: 1T mode, i.e., CPU clock without frequency division divider (FOSC/1)

14.2.13 Interrupt and Clock Output Control Register (INTCLKO)

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

T0CLKO: Timer 0 clock output

control 0: Clock output

off

1: Enabling the P3.5 port is the Timer 0 clock output function.

The level of the P3.5 port is automatically flipped when an overflow occurs in the Timer 0 count.

T1CLKO: Timer 1 clock output

control 0: Clock output

off

1: Enabling the P3.4 port is the Timer 1 clock output function.

When an overflow occurs in Timer 1 count, the level of the P3.4 port is automatically flipped.

14.2.14 8-bit prescaler register for Timer 0 (TM0PS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TM0PS	7EFEA0H								

Timer 0 clock = System clock SYSclk ÷ (TM0PS + 1)

14.2.15 8-bit prescaler register for Timer 1 (TM1PS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TM1PS	7EFEA1H								

Timer 1 clock = System clock SYSclk ÷ (TM1PS + 1)

14.2.16 Timer 0 Calculation formula

timer mode	Timer speed	Periodicity formula
Mode 0/3 (16-bit auto-reload)	1T	$\text{Timer period} = \frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)}$ (automatic reloading)
	12T	$\text{Timer period} = \frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)} \times 12$ (automatic reloading)
Model 1 (16-bit without automatic reloading)	1T	$\text{Timer period} = \frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)}$ (software loading required)
	12T	$\text{Timer period} = \frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)} \times 12$ (software loading required)
Model 2 (8-bit auto-reload)	1T	$\text{Timer period} = \frac{256 - TH0}{SYSclk/(TM0PS+1)}$ (automatic reloading)
	12T	$\text{Timer period} = \frac{256 - TH0}{SYSclk/(TM0PS+1)} \times 12$ (automatic reloading)

14.2.17 Timer 1 Calculation formula

timer mode	Timer speed	Periodicity formula
Mode 0 (16-bit auto-reload)	1T	$\text{Timer period} = \frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)}$ (automatic reloading)
	12T	$\text{Timer period} = \frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)} \times 12$ (automatic reloading)
Model 1 (16-bit without	1T	$\text{Timer period} = \frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)}$ (software loading required)

automatic reloading)	12T	Timer period = $\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)} \times 12$	(software loading required)
Model 2 (8-bit auto-reload)	1T	Timer period = $\frac{256 - TH1}{SYSclk/(TM1PS+1)}$	(automatic reloading)
	12T	Timer period = $\frac{256 - TH1}{SYSclk/(TM1PS+1)} \times 12$	(automatic reloading)

14.3 Timer 2

14.3.1 Auxiliary Register 1 (AUXR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT

T2R: Timer 2 operation
control bit 0: Timer 2
stop counting
1: Timer 2 starts counting

T2_C/T: Controls whether Timer 2 is used as a timer or counter, clearing 0 is used as a timer (counting the internal system clock), setting 1 is used as a counter (counting external pulses on pin T2/P1.2).

T2x12: Timer 2 speed control bit
0: 12T mode, i.e. CPU clock 12 divisions (FOSC/12)
1: 1T mode, i.e., CPU clock without frequency division divider (FOSC/1)

14.3.2 Interrupt and Clock Output Control Register (INTCLKO)

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

T2CLKO: Timer 2 clock output
control 0: Clock output
off
1: Enabling the P1.3 port is the Timer 2 clock output function.
When an overflow occurs in Timer 2 count, the level of the P1.3 port is automatically flipped.

14.3.3 Timer 2 Count Register (T2L, T2H)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

The operating mode of Timer/Counter 2 is fixed to 16-bit reload mode, T2L and T2H are combined into one 16-bit register, T2L is the low byte and T2H is the high byte. When the 16-bit count value in [T2H,T2L] overflows, the system will automatically load the reloaded value in the internal 16-bit reload register into [T2H,T2L].

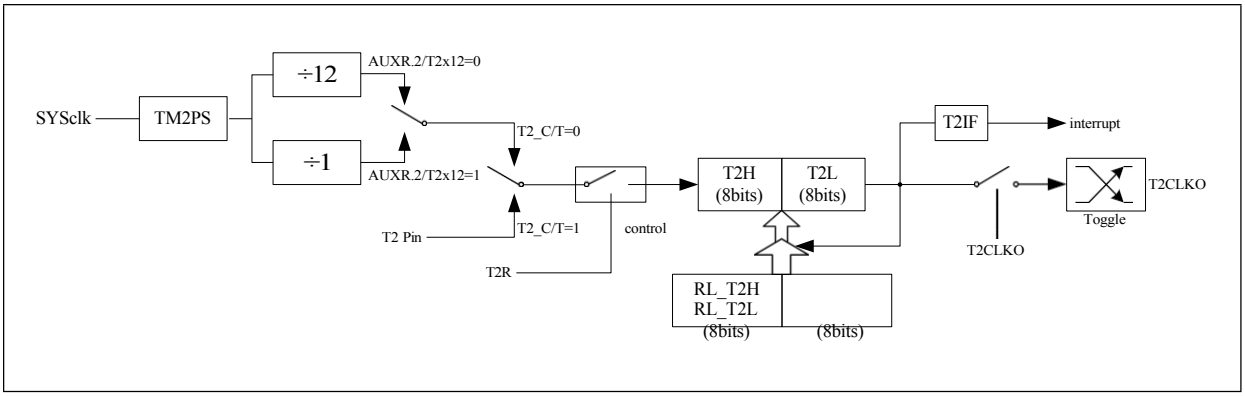
14.3.4 8-bit prescaler register for Timer 2 (TM2PS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TM2PS	FEA2H								

Timer 2 clock = System clock SYSclk ÷ (TM2PS + 1)

14.3.5 Timer 2 operating mode

The block diagram of Timer/Counter 2 is shown below:



Timer/Counter 2 Operating Mode: 16-bit Auto-Reload Mode

T2R/AUXR.4 is the control bit in the AUXR register, and the specific function of each bit of the AUXR register is described in the previous section of the AUXR register.

Introduction.

When T2_C/T=0, the multiswitch is connected to the system clock output, T 2 counts the internal system clock, and T 2 works in the timing mode. When T2_C/T=1

When the multiswitch is connected to the external pulse input T2, i.e. T2 operates in counting mode.

Timer 2 of the STC microcontroller has two counting rates: one is 12T mode, adding 1 every 12 clocks, the same as the traditional 8051 microcontroller; the other is 1T mode, adding 1 every clock, which is 12 times faster than the traditional 8051 microcontroller.The rate of T2 is determined by T2x12 in the Special Function Register AUXR, and T2 operates in 12T mode if T2x12=0; T2 operates in 1T mode if T2x12=1; and T2 operates in 1T mode if T2x12=1. If T2x12=0, T2 works in 12T mode; if T2x12=1, T2 works in 1T mode.

Timer 2 has two hidden registers, RL_T2H and RL_T2L. RL_T2H shares the same address with T2H and RL_T2L shares the same address with T2L. When T2R=0, that is, Timer/Counter 2 is prohibited to work, the content written to T2L will be written to RL_T2L at the same time, and the content written to T2H will be written to RL_T2H at the same time, when T2R=1, that is, Timer/Counter 2 is allowed to work, the content written to T2L is not actually written to the current register T2L, but written to hidden register RL_T2L, and the content written to T2H will be written to RL_T2L, and RL_T2L will be written to RL_T2L, and RL_T2L and T2H will be written to T2L. When T2L is allowed to work, writing to T2H is not actually writing to the current register T2L, but to the hidden register RL_T2H, so that the 16-bit reload timer can be realised in a clever way. When reading the contents of T2H and T2L, what is read is the contents of T2H and T2L, not the contents of RL_T2H and RL_T2L.

An overflow of [T2H,T2L] not only sets the interrupt request flag bit (T2IF), causing the CPU to go to the interrupt routine for Timer 2, but also automatically reloads the contents of [RL_T2H,RL_T2L] into [T2H,T2L].

14.3.6 Timer 2 Calculation Formula

Timer speed	Periodicity formula
1T	Timer period = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)}$ (automatic reloading)
12T	Timer period = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)} \times 12$ (automatic reloading)

14.4 Timer 3/4

14.4.1 Timer 3/4 function pin switching

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
T3T4PIN	7EFEACH	-	-	-	-	-	-	-	**T3T4SEL**

T3T4SEL: T3/T3CLKO/T4/T4CLKO pin select bit

T3T4SEL	T3	T3CLKO	T4	T4CLKO
0	P0.4	P0.5	P0.6	P0.7
1	P0.0	P0.1	P0.2	P0.3

14.4.2 Timer 4/3 Control Register (T4T3M)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	DDH	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

TR4: Timer 4 operation
control bit 0: Timer 4
stops counting
1: Timer 4 starts counting

T4_C/T: Controls whether Timer 4 is used as a timer or counter, clearing 0 is used as a timer (counting the internal system clock), setting 1 is used as a counter (counting external pulses on pin T4/P0.6).

T4x12: Timer 4 speed control bit
0: 12T mode, i.e. CPU clock 12 divisions (FOSC/12)
1: 1T mode, i.e., CPU clock without frequency division divider (FOSC/1)

T4CLKO: Timer 4 clock output
control 0: Clock output
off
1: Enabling the P0.7 port is the Timer 4 clock output function.
When an overflow occurs in Timer 4 count, the level of the P0.7 port is automatically flipped.

TR3: Timer 3 operation
control bit 0: Timer 3
stop counting
1: Timer 3 starts counting

T3_C/T: Controls whether Timer 3 is used as a timer or counter, clearing 0 is used as a timer (counting the internal system clock), setting 1 is used as a counter (counting external pulses on pin T3/P0.4).

T3x12: Timer 3 speed control bit
0: 12T mode, i.e. CPU clock 12 divisions (FOSC/12)
1: 1T mode, i.e., CPU clock without frequency division divider (FOSC/1)

T3CLKO: Timer 3 clock output
control 0: Clock output
off
1: Enabling port P0.5 is the Timer 3 clock output function.

When Timer 3 overflow occurs, the level of the P0.5 port is automatically flipped.

14.4.3 Timer 3 Count Register (T3L, T3H)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

The operating mode of Timer/Counter 3 is fixed to 16-bit reload mode, T3L and T3H are combined into one 16-bit register, T3L is the low byte and T3H is the high byte. When the 16-bit count value in [T3H,T3L] overflows, the system will automatically load the reloaded value in the internal 16-bit reload register into [T3H,T3L].

14.4.4 Timer 4 Count Register (T4L, T4H)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

The operating mode of Timer/Counter 4 is fixed to 16-bit reload mode, T4L and T4H are combined into one 16-bit register, T4L is the low byte and T4H is the high byte. When the 16-bit count value in [T4H,T4L] overflows, the system will automatically load the reloaded value in the internal 16-bit reload register into [T4H,T4L].

14.4.5 8-bit prescaler register for Timer 3 (TM3PS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TM3PS	FEA3H								

Timer 3 clock = System clock SYSclk ÷ (TM3PS + 1)

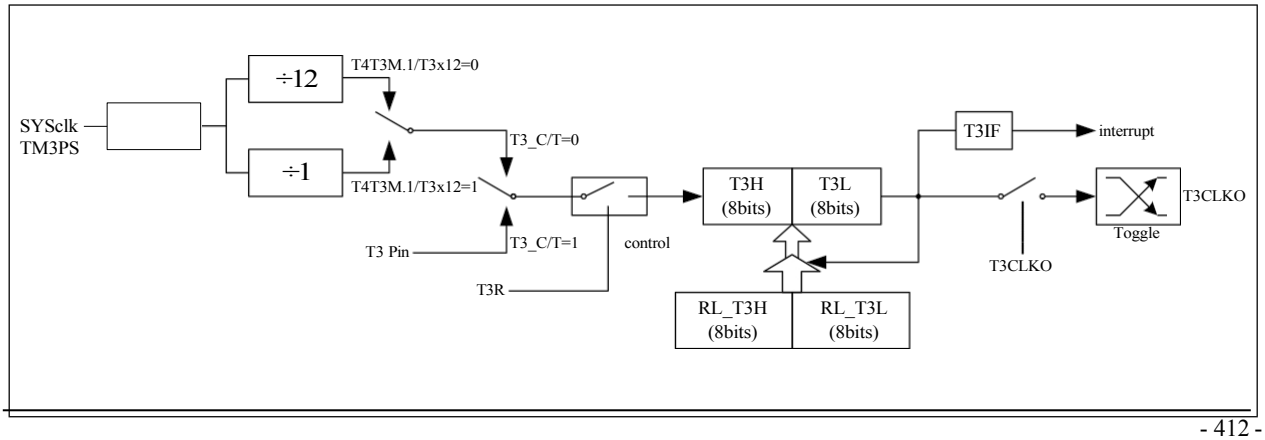
14.4.6 8-bit prescaler register for Timer 4 (TM4PS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TM4PS	FEA4H								

Timer 4 clock = System clock SYSclk ÷ (TM4PS + 1)

14.4.7 Timer 3 operating mode

The block diagram of Timer/Counter 3 is shown below:



Timer/Counter 3 Operating Mode: 16-bit Auto-Reload Mode

T3R/T4T3M.3 are control bits within the T4T3M register. The specific function description of each T4T3M register bit is described in the previous section on T4T3M register.

When T3_C/T=0, the multiswitch is connected to the system clock output, T 3 counts the internal system clock, and T 3 works in the timing mode. When T3_C/T=1

When the multiswitch is connected to the external pulse input T3, i.e. T3 operates in counting mode.

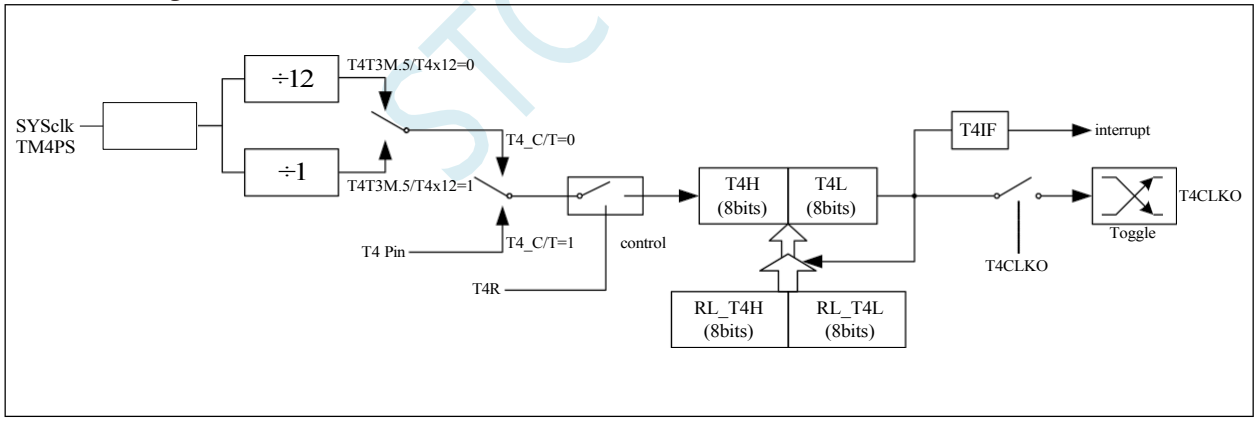
Timer 3 of the STC microcontroller has two counting rates: one is 12T mode, adding 1 every 12 clocks, the same as the traditional 8051 microcontroller; the other is 1T mode, adding 1 every clock, which is 12 times faster than the traditional 8051 microcontroller.The rate of T3 is determined by T3x12 in the Special Function Register T4T3M, and T3 works in 12T mode if T3x12=0; T3 works in 1T mode if T3x12=1; T3 works in 1T mode if T3x12=1. If T3x12=0, T3 works in 12T mode; if T3x12=1, T3 works in 1T mode.

Timer 3 has two hidden registers, RL_T3H and RL_T3L. RL_T3H shares the same address with T3H and RL_T3L shares the same address with T3L. When T3R=0, i.e. timer/counter 3 is prohibited to work, the content written to T3L will be written to RL_T3L at the same time, and the content written to T3H will be written to RL_T3H at the same time; when T3R=1, i.e. timer/counter 3 is allowed to work, the content written to T3L is not actually written to the current register T3L, but is written to hidden register RL_T3L, while the content written to T3H is not written to T3L, but is written to RL_T3L, and RL_T3L. When T3L is allowed to work, writing to T3H is not actually writing to the current register T3L, but to the hidden register RL_T3H, so that the 16-bit reload timer can be realised cleverly. When reading the contents of T3H and T3L, what is read is the contents of T3H and T3L, not the contents of RL_T3H and RL_T3L.

An overflow of [T3H,T3L] not only sets the interrupt request flag bit (T3IF), causing the CPU to go to the interrupt routine for Timer 3, but also automatically reloads the contents of [RL_T3H,RL_T3L] into [T3H,T3L].

14.4.8 Timer 4 operating mode

The block diagram of Timer/Counter 4 is shown below:



Timer/Counter 4 Operating Mode: 16-bit Auto-Reload Mode

T4R/T4T3M.7 are the control bits within the T4T3M register, and the specific function descriptions of each of the T4T3M register bits are described in the previous section on the T4T3M register.

When T4_C/T=0, the multiswitch is connected to the system clock output, T 4 counts the internal system clock, and T 4

operates in timing mode. When T4_C/T=1

When the multiswitch is connected to the external pulse input T4, i.e. T4 operates in counting mode.

Timer 4 of the STC microcontroller has two counting rates: one is 12T mode, adding 1 every 12 clocks, the same as the traditional 8051 microcontroller; the other is 1T mode, adding 1 every clock, which is 12 times faster than the traditional 8051 microcontroller.The rate of T4 is determined by T4x12 in the Special Function Register T4T3M, and T4 works in 12T mode if T4x12=0; T4 works in 1T mode if T4x12=1; T4 works in 1T mode if T4x12=1. If T4x12=0, T4 operates in 12T mode; if T4x12=1, T4 operates in 1T mode.

Timer 4 has two hidden registers, RL_T4H and RL_T4L. RL_T4H shares the same address as T4H, and RL_T4L shares the same address as T4L.

The same address is shared. When T4R=0, i.e. timer/counter 4 is prohibited to work, the content written to T4L will be written to RL_T4L at the same time, the content written to T4H will also be written to RL_T4H at the same time, when T4R=1, i.e. timer/counter 4 is allowed to work, the content written to T4L is not written to the current register T4L, but written to the hidden register RL_T4L, and the content written to T4H is also not written to the current register T4H, but written to the hidden register RL_T4H, which can cleverly implement the 16-bit address. When T4L is allowed to work, writing to T4H is not actually writing to the current register T4L, but to the hidden register RL_T4H, so that the 16-bit reload timer can be realised in a clever way. When reading the contents of T4H and T4L, what is read is the contents of T4H and T4L, not the contents of RL_T4H and RL_T4L.

An overflow of [T4H,T4L] not only sets the interrupt request flag bit (T4IF), causing the CPU to go to the interrupt routine for Timer 4, but also automatically reloads the contents of [RL_T4H,RL_T4L] into [T4H,T4L].

14.4.9 Timer 3 Calculation formula

Timer speed	Periodicity formula
1T	$\text{Timer period} = \frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)}$ (automatic reloading)
12T	$\text{Timer period} = \frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)} \times 12$ (automatic reloading)

14.4.10 Timer 4 Calculation formula

Timer speed	Periodicity formula
1T	$\text{Timer period} = \frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)}$ (automatic reloading)
12T	$\text{Timer period} = \frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)} \times 12$ (automatic reloading)

14.5 sample procedure

14.5.1 Timer 0 (mode 0-16 bit auto-reload), used as timer

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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 the timer
    ET0 = 1;                            //Enable timer interrupt
    EA = 1;

    while (1);
}
```

14.5.2 Timer 0 (mode 1-16 bits not auto-reloaded), used as timer

```
//Tested operating frequency is 11.0592MHz
```

```
#include "stc32g.h"                                    //see download software for header files
```

```
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    TL0 = 0x66.                          // Parameter at reset
    TH0 = 0xfc.
    P10 = !P10;                          //test port
}

void main()
{
    EAXFR = 1;                           //Enable access to XFR
    CKCON = 0x00;                        //Set the external data bus speed to fastest
    WTST = 0x00;                         //set the program code wait parameter.
                                         //Assign a value of 0 to set the CPU to execute the
                                         programme as fast as possible.

    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 the timer
    ET0 = 1;                             //Enable timer interrupt
    EA = 1;

    while (1);
}
```

14.5.3 Timer 0 (mode 2-8 bit auto-reload), used as timer

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                      // see download software for header files
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10;                          //test port
}

void main()
{
    EAXFR = 1;                           //Enable access to XFR
```

```
    ckcon = 0x00;                       //Set external data bus speed to fastest
    wtst = 0x00.                        //Set the parameter for waiting for the programme
                                        code.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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; th0 =                   //256-11.0592m/12/76k
    0xf4; tr0 = 1.
    ET0 = 1;                            //Start timer
    EA = 1;                             //Enable timer interrupt


    while (1);
}
```

14.5.4 Timer 0 (mode 3-16 bit auto-reload non-maskable interrupt), used as timer

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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);
}
```

14.5.5 Timer 0 (external count - extend T0 for external falling edge interrupt)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 //see download software for header files
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10;                     //test port
}

void main()
{
    EAXFR = 1;                      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    TMOD = 0x04.                    //External counting mode
    TL0 = 0xff.
```

```
    TH0 = 0xff;
    TR0 = 1;                    //start the timer
    ET0 = 1;                    //Enable timer interrupt
```

```
    EA = 1;

    while (1);
}
```

14.5.6 Timer 0 (measuring pulse width - INT0 high level width)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

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.   //TH0 is the high byte of the measured value
    TL0 = 0x00.
    TH0 = 0x00.
}

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    T0x12 = 1;                               //1T mode
    TMOD = 0x08.                             //Enable GATE, enable timing when INT0 is 1.
    TL0 = 0x00.
    TH0 = 0x00.
    while (P32); //wait for INT0 to go low   //Wait for INT0 to go low
    TR0 = 1;                                 //start the timer
    IT0 = 1; //Enable INT0 falling edge interrupt.   //enable INT0 falling edge interrupt
    EX0 = 1;
    EA = 1;

    while (1);
```

14.5.7 Timer 0 (Mode 0), Clock Divided Outputs

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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 the timer
    T0CLKO = 1;                         //Enable clock output

    while (1);
}
```

14.5.8 Timer 1 (mode 0-16 bit auto-reload), used as timer

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
```

```
    ckcon = 0x00;                       //Set external data bus speed to fastest
    wtst = 0x00.                        //Set the parameter for waiting for the programme code.
                                        //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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                    //Mode 0
    = 0x66.                             //65536-11.0592m/12/1000
    th1 = 0xfc; tr1
    = 1.                                //Start timer
    ET1 = 1;                            //Enable timer interrupt
    EA = 1;

    while (1);
}
```

14.5.9 Timer 1 (mode 1-16 bits without auto-reload), used as timer

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    TL1 = 0x66.                         // Parameter on reset
    TH1 = 0xfc.
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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);
}
```

14.5.10 Timer 1 (mode 2-8 bit auto-reload), for timing

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 // see download software for header files
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    P10 = !P10;                     //test port
}

void main()
{
    EAXFR = 1;                      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    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 the timer
    ET1 = 1;                        //Enable timer interrupt
```

```
    EA = 1;

    while (1);
}
```

14.5.11 Timer 1 (external count - extend T1 for external falling edge interrupt)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    TMOD = 0x40.                        //External counting mode
    TL1 = 0xff.
    TH1 = 0xff;
    TR1 = 1;                            //start the timer
    ET1 = 1;                            //Enable timer interrupt
    EA = 1;

    while (1);
}
```

14.5.12 Timer 1 (measuring pulse width - INT1 high level

width)

//Tested operating frequency is *11.0592MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

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.   //TH1 is the high byte of the measured value
    TL1 = 0x00.
    TH1= 0x00.
}

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    T1x12 = 1;                               //1T mode
    TMOD = 0x80.                             //Enable GATE, enable timing when INT1 is 1.
    TL1 = 0x00.
    TH1 = 0x00.
    while (INT1);                            //Wait for INT1 to go low
    TR1 = 1;                                 //start the timer
    IT1 = 1; //Enable INT1 falling edge interrupt.   //Enable INT1 falling edge interrupt.
    EX1 = 1;
    EA = 1;

    while (1);
}
```

14.5.13 Timer 1 (mode 0), clock divider output

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void main()
```

```
{
    EAXFR = 1;                          //Enable access to XFR
    ckcon = 0x00;                       //Set external data bus speed to fastest
    wtst = 0x00.                        //Set the parameter for waiting for the programme code.
                                        //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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                    //Mode 0
    = 0x66.                             //65536-11.0592m/12/1000
    th1 = 0xfc; tr1
    = 1.                                //Start timer
    T1CLKO = 1.                         //Enable clock output

    while (1);
}
```

14.5.14 Timer 1 (mode 0) does serial port 1 baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL  //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - FOSC / 115200 / 4)
                                        //The // plus 2 operation is to allow the Keil compiler to
                                        //Automatic implementation of rounding operations

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;
        s1brt = 0; tmod
        = 0x00; tl1 =
        brt.
        TH1 = BRT >> 8;
        TR1 = 1;
        T1x12 = 1.
        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()
{
        EAXFR = 1;                  //Enable access to XFR
        CKCON = 0x00;               //Set the external data bus speed to fastest
        WTST = 0x00;                //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

        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.
        }
    }
}
```

14.5.15 Timer 1 (mode 2) as serial port 1 baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (256 - (FOSC / 115200+16) / 32)

                                        //Define as unsigned long integer to avoid overflow.
                                        The //add 16 operation is designed to allow the Keil compiler to
                                        //Automatic implementation of rounding operations

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;
    s1brt = 0; tmod
    = 0x20; tl1 =
    brt; th1 = brt;
    tr1 = 1.
    T1x12 = 1.
    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()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign a value of 0 to set the CPU to execute the
                                programme as fast as possible.

    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.
        }
    }
}
```

14.5.16 Timer 2 (16-bit auto-reload), for Timing

```
//Tested operating frequency is 11.0592MHz
```

```
//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void TM2_Isr() interrupt 12
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    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.
    T2R = 1;                            //start the timer
    ET2 = 1;                            //Enable timer interrupt
    EA = 1;

    while (1);
}
```

14.5.17 Timer 2 (external count - extend T2 for external falling edge interrupt)

//Tested operating frequency is 11.0592MHz

```
//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void TM2_Isr() interrupt 12
{
    P10 = !P10;                         //test port
}
```

```
void main()
{
    EAXFR = 1;                                  //Enable access to XFR
```

```
    ckcon = 0x00; wtst =                    //Set external data bus speed to fastest
    0x00.                                   //Set the parameter for waiting for the programme code.
                                            //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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.
    t2ct = 1; t2r = 1.                      //Set external counting mode and start timer
    ET2 = 1;                                //Enable timer interrupt
    EA = 1;

    while (1);
}
```

14.5.18 Timer 2, Clock Divided Output

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set the external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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.
    T2R = 1;                                    //start the timer
    T2CLKO = 1;                                 //Enable clock output

    while (1);
}
```

14.5.19 Timer 2 do serial port 1 baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL          //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)

                                                The // plus 2 operation is to allow the Keil compiler to
                                                //Automatic implementation of rounding operations

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;
    S1BRT = 1.
    T2x12 = 1.
    T2R = 1.
    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()
{
        EAXFR = 1;       //Enable access to XFR
        CKCON = 0x00;                                   //Set the external data bus speed to fastest
        WTST = 0x00;                                    //set the program code wait parameter.
                                                        //Assign a value of 0 to set the CPU to execute the
                                                        programme as fast as possible.

        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.
                }
        }
}
```


14.5.20 Timer 2 do serial port 2 baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                     // see download software for header files
#include "intrins.h"
```

```
#define     FOSC            11059200UL                  //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)
                                                        The // plus 2 operation is to allow the Keil compiler to
                                                        //Automatic implementation of rounding operations

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

void Uart2Isr() interrupt 8
{
    if (S2TI)
    {
        S2TI = 0;
        busy = 0;
    }
    if (S2RI)
    {
        S2RI = 0;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f.
    }
}

void Uart2Init()
{
    s2con = 0x50;
    s1brt = 1; t2l =
    brt.
    T2H = BRT >> 8;
    T2x12 = 1.
    T2R = 1.
    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()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                                       //Set the external data bus speed to fastest
    WTST = 0x00;                                        //set the program code wait parameter.
                                                        //Assign a value of 0 to set the CPU to execute the
```

programme as fast as possible.

```
    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.
        }
    }
}
```

14.5.21 Timer 2 Make serial port 3 Baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL  //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)

                            The // plus 2 operation is to allow the Keil compiler to
                            //Automatic implementation of rounding operations

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

void Uart3Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;
        busy = 0;
    }
    if (S3RI)
    {
```

```
            S3RI = 0;
            buffer[wptr++] = S3BUF;
            wptr &= 0x0f.
        }
}

void Uart3Init()
{
        s3con = 0x10;
        t2l = brt.
        T2H = BRT >> 8;
        T2x12 = 1.
        T2R = 1.
        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()
{
        EAXFR = 1;       //Enable access to XFR
        CKCON = 0x00;                                   //Set the external data bus speed to fastest
        WTST = 0x00;                                    //set the program code wait parameter.
                                                        //Assign a value of 0 to set the CPU to execute the
                                                        programme as fast as possible.

        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.
        }
    }
}
```

14.5.22 Timer 2 do serial port 4 baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL  //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)

The // plus 2 operation is to allow the Keil compiler to
//Automatic implementation of rounding operations

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

void Uart4Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0.
        busy = 0;
    }
    if (S4RI)
    {
        S4RI = 0.
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f.
    }
}

void Uart4Init()
{
    s4con = 0x10;
    t2l = brt.
    T2H = BRT >> 8;
    T2x12 = 1.
    T2R = 1.
    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()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    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.
        }
    }
}
```


14.5.23 Timer 3 (16-bit auto-reload) for Timing

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
```

```
#include "intrins.h"

#define   ET2         0x04
#define   ET3         0x20
#define   ET4         0x40
#define   T2IF        0x01
#define   T3IF        0x02
#define   T4IF        0x04

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //test port
}

void main()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign a value of 0 to set the CPU to execute the
                                programme as fast as possible.

    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);
}
```

14.5.24 Timer 3 (external count - extend T3 for external falling edge interrupt)

```
//Tested operating
frequency is 11.0592MHz

//#include "stc8h.h"
#include  "stc32g.h"                //See download software for header files
#include "intrins.h"
```

```
#define    ET2         0x04
#define    ET3         0x20
#define    ET4         0x40
#define    T2IF        0x01
```

```
#define   T3IF          0x02
#define   T4IF          0x04

void TM3_Isr() interrupt 19
{
    P10 = !P10;                     //test port
}

void main()
{
    eaxfr = 1; ckcon =              // Enable access to
    0x00; wtst = 0x00.              XFR
                                    //Set external data bus speed to fastest
                                    //Set the parameter for waiting for the
                                    programme code.
                                    //Assign a value of 0 to set the CPU to
                                    execute the programme as fast as possible.

    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 = 0xff;
    t3h = 0xff.
    T4T3M = 0x0c.                   //Set external counting mode and start timer
    IE2 = ET3.                      //Enable timer interrupt
    EA = 1;

    while (1);
}
```

14.5.25 Timer 3, Clock Divided Output

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                      //Enable access to XFR
    CKCON = 0x00;                   //Set external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    p0m0 = 0x00;
```

```
    p0m1 = 0x00;
    p1m0 = 0x00.
```

```
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    T3L = 0x66.                                   //65536-11.0592m/12/1000
    t3h = 0xfc; t4t3m =
    0x09.                                         //Enable the clock output and start the timer

    while (1);
}
```

14.5.26 Timer 3 do serial port 3 baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                               // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL            //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)
                                                  The // plus 2 operation is to allow the Keil compiler to
                                                  //Automatic implementation of rounding operations

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

void Uart3Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;
        busy = 0;
    }
    if (S3RI)
    {
        S3RI = 0;
        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()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                    //Set the external data bus speed to fastest
    WTST = 0x00;                     //set the program code wait parameter.
                                     //Assign a value of 0 to set the CPU to execute the
                                     programme as fast as possible.


    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.
        }
    }
}
```

14.5.27 Timer 4 (16-bit auto-reload), for Timing

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // See download software for header files
#include "intrins.h"

#define   ET2         0x04
#define   ET3         0x20
#define   ET4         0x40
#define   T2IF        0x01
#define   T3IF        0x02
#define   T4IF        0x04

void TM4_Isr() interrupt 20
{
    P10 = !P10;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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);
}
```

14.5.28 Timer 4 (external count - extend T4 for external falling edge interrupt)

//Tested operating frequency is *11.0592MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

#define    ET2             0x04
#define    ET3             0x20
#define    ET4             0x40
#define    T2IF            0x01
#define    T3IF            0x02
#define    T4IF            0x04

void TM4_Isr() interrupt 20
{
    P10 = !P10;                          //test port
}

void main()
{
    eaxfr = 1; ckcon =                   // Enable access to
    0x00; wtst = 0x00.                   XFR
                                         //Set external data bus speed to fastest
                                         //Set the parameter for waiting for the
                                         programme code.
                                         //Assign a value of 0 to set the CPU to
                                         execute the programme as fast as possible.
    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 = 0xff;
    t4h = 0xff.
    T4T3M = 0xc0.                        //Set the external counting mode and start the timer
    IE2 = ET4.                           //enable timer interrupt
    EA = 1;

    while (1);
}
```

14.5.29 Timer 4, Clock Divided Output

//Tested operating frequency is 11.0592MHz

```
//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

void main()
```

```
    eaxfr = 1; ckcon = 0x00; wtst = 0x00.      //Enable access to XFR
                                               //Set external data bus speed to fastest
                                               //Set the parameters for the programme code wait.
                                               //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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);
}
```

14.5.30 Timer 4 do serial port 4 baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                            // see download software for header files
#include "intrins.h"

#define    FOSC        11059200UL              //Define as unsigned long integer to avoid overflow.
#define    BRT         (65536 - (FOSC / 115200+2) / 4)
                                               //The // plus 2 operation is to allow the Keil compiler to
                                               //Automatic implementation of rounding operations

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

void Uart4Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0.
        busy = 0;
    }
    if (S4RI)
    {
        S4RI = 0.
        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()
{
        EAXFR = 1;       //Enable access to XFR
        CKCON = 0x00;                                  //Set the external data bus speed to fastest
        WTST = 0x00;                                   //set the program code wait parameter.
                                                       //Assign a value of 0 to set the CPU to execute the
                                                       programme as fast as possible.

        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.
            }
        }
}
```

15 Synchronous/asynchronous serial communications (USART1, USART2)

The STC32G series microcontrollers have 2 full duplex synchronous/asynchronous serial communication interfaces (USART1 and USART2). Each serial port consists of 2 data buffers, a shift register, a serial control register and a baud rate generator. Each serial port has a

The data buffer consists of 2 receive and transmit buffers that are independent of each other and can send and receive data at the same time.

The serial port 1 and serial port 2 of STC32G series microcontrollers have four working modes, two of which have variable baud rates and the other two are fixed for different applications. Users can use software to set different baud rates and select different operating modes. The host can be queried or interrupted for receiving/transmitting, which makes it very flexible to use.

The communication ports of Serial Port 1 and Serial Port 2 can be switched to multiple ports by the switching function of the function pins, so that it is possible to time-multiplex a communication port into multiple communication ports.

15.1 Serial Port Function Pin Switching

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	**S1_S[1:0]**		CAN_S[1:0]		SPI_S[1:0]		LIN_S[1:0]	

S1_S[1:0]: Serial port 1 function pin select 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

notation	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**

S2_S: Serial Port 2 Function Pin Select Bit

S2_S	RxD2	TxD2
0	P1.0	P1.1
1	P4.6	P4.7

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW3	BBH	I2S_S		S2SPI_S[1:0]		S1SPI_S[1:0]		CAN2_S[1:0]	

S2SPI_S[1:0]: SPI function pin select bits for USART2

S2SPI_S[1:0]	S2SS	S2MOSI	S2MISO	S2SCLK
00	P1.2/P5.4[1]	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P7.4	P7.5	P7.6	P7.7

[1] 有 P1.2 port, this function is on theP1.2 port, and for models without theP1.2 port, this function is on the P5.4 口上

S1SPI_S[1:0]: SPI function pin select bit for USART1

S1SPI_S[1:0]	S1SS	S1MOSI	S1MISO	S1SCLK
00	P1.2/P5.4[1]	P1.3	P1.4	P1.5

01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P6.4	P6.5	P6.6	P6.7

[1] ：有 P1.2 port, this function is on theP1.2 port, and for models without theP1.2 port, this function is on the P5.4 口上

15.2 Serial port related registers

notation	descriptions	addr-ess	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	Serial Port 1 Control Register	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	Serial Port 1 Data Register	99H									0000,0000
S2CON	Serial Port 2 Control Register	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
S2BUF	Serial Port 2 Data Register	9BH									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	S1BRT	0000,0001
SADDR	Serial Port 1 Slave Address Register	A9H									0000,0000
SADEN	Serial Port 1 Slave Address Mask Register	B9H									0000,0000

notation	descriptions	addres s	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
S2CFG	Serial Port 2 Configuration Register	7EFDB4H	-	S2MOD0	S2M0x6	-	-	-	-	W1	000x,xxx0
S2ADDR.	Serial 2 Slave Address Register	7EFDB5H									0000,0000
S2ADEN	Serial Port 2 Slave Address Mask Register	7EFDB6H									0000,0000
USARTCR1	Serial Port 1 Control Register 1	7EFDC0H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA	0000,0000
USARTCR2	Serial Port 1 Control Register 2	7EFDC1H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE	0000,0000
USARTCR3	Serial Port 1 Control Register 3	7EFDC2H	IrDA_LPBAUD[7:0]								0000,0111
USARTCR4	Serial Port 1 Control Register 4	7EFDC3H	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]		xxxx,0000
USARTCR5	Serial Port 1 Control Register 5	7EFDC4H	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC	0000,0000
USARTGTR	Serial Port 1 Protection Time Register	7EFDC5H									0000,0000
USARTBRH	Serial Port 1 Baud Rate Register	7EFDC6H	USARTBR[15:8]								0000,0000
USARTBRL	Serial Port 1 Baud Rate Register	7EFDC7H	USARTBR[7:0]								0000,0000
USART2CR1	Serial Port 2 Control Register 1	7EFDC8H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA	0000,0000
USART2CR2	Serial Port 2 Control Register 2	7EFDC9H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE	0000,0000

USART2CR3	Serial Port 2 Control Register 3	7EFDCAH	IrDA_LPBAUD[7:0]								0000,0000
USART2CR4	Serial Port 2 Control Register 4	7EFDCBH	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]		xxxx,0000
USART2CR5	Serial Port 2 Control Register 5	7EFDCCH	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNCAN	0000,0000
USART2GTR	Serial Port 2 Protection Time Register	7EFDCDH									0000,0000
USART2BRH	Serial Port 2 Baud Rate Register	7EFDCEH	USART2BR[15:8]								0000,0000
USART2BRL	Serial Port 2 Baud Rate Register	7EFDCFH	USART2BR[7:0]								0000,0000

15.3 Serial Port 1 (Synchronous/Asynchronous Serial USART)

15.3.1 Serial Port 1 Control Register (SCON)

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

SM0/FE: When the SMOD0 bit in the PCON register is 1, this bit is the frame error detection flag bit. When the UART detects an invalid stop bit during reception, this bit is 1 through the UART receiver and must be cleared by software. When the SMOD0 bit in the PCON register is 0, this bit, along with SM1, specifies the communication operating mode for Serial Port 1, as shown in the table below:

SM0	SM1	Serial port 1 operating mode	Functional Description
0	0	Mode 0	Synchronous shift serial method
0	1	Model 1	Variable baud rate 8-bit data mode
1	0	Model 2	Fixed baud rate 9-bit data mode
1	1	Mode 3	Variable baud rate 9-bit data mode

SM2: Allow mode 2 or mode 3 multicomputer communication control bit. When serial port 1 uses mode 2 or mode 3, if SM2 bit is 1 and REN bit is 1, the receiver is in address frame filtering state. At this time, the receiver can use the received bit 9 (i.e. RB8) to screen the address frame, if RB8=1, it means that the frame is an address frame, the address information can be entered into the SBUF, and make RI is 1, and then in the interrupt service program to compare the address number; if RB8=0, it means that the frame is not an address frame, it should be thrown away and keep the RI=0. In Mode 2 or Mode 3, if SM2 is 0 and REN bit is 1, the receiver is in the state of address frame filtering. In mode 2 or 3, if the SM2 bit is 0 and the REN bit is 1, the receiver is in the address frame filtering disabled state, no matter the received RB8 is 0 or 1, it can make the received information into the SBUF and make the RI=1, at this time, RB8 is usually the parity bit. Mode 1 and mode 0 are non-multi-computer communication modes, in these two modes, SM2 should be set to 0.

REN: Allow/Disallow serial port receive control bit 0:
Disable serial port receive data
1: Allow serial port to receive data

TB8: TB8 is the 9th bit of data to be sent when Serial Port 1 is in Mode 2 or Mode 3, and is set or cleared by software as required.

This bit is not used in mode 0 and mode 1.

RB8: RB8 is the 9th bit of data received when serial port 1 is used in mode 2 or mode 3 and is generally used as a parity bit or address frame/data.

Frame flag bit. This bit is not used in Mode 0 and Mode 1.

TI: Serial port 1 transmit interrupt request flag bit. In mode 0, when the serial port sends the data at the end of the 8th bit, the hardware will set TI to 1 automatically to request an interrupt from the host, and TI must be cleared by software after responding to the interrupt. In other modes, TI is automatically set to 1 by the hardware when the stop bit starts to be sent to request an interrupt from the CPU, and TI must be cleared by software after responding to the interrupt.

RI: Serial port 1 receive interrupt request flag bit. In mode 0, when the serial port receives the end of the 8th bit of data, the hardware will automatically set RI to 1 to request an interrupt from the host, and RI must be cleared by software after responding to the interrupt. In other modes, the hardware will automatically set RI to 1 in the middle of receiving stop bit, and send an interrupt request to the CPU, and RI must be cleared by software after responding to the interrupt.

15.3.2 Serial Port 1 Data Register (SBUF)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF: Serial port 1 data receive/send buffer.SBUF is actually 2 buffers, read buffer and write buffer, the two operations correspond to two different registers respectively, 1 is write-only register (write buffer) and 1 is read-only register (read buffer). The two operations correspond to two different registers.

A read operation to SBUF actually reads the serial port receive buffer, and a write operation to SBUF triggers the serial port to start sending data.

15.3.3 Power Management Register (PCON)

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

SMOD: Serial port 1 baud rate control bit

0: Baud rate is not doubled for all modes of serial port 1

1: Serial Port 1 Mode 1 (valid when using Timer 1 of Mode 2 as baud rate generator), Mode 2, Mode 3 (valid when using Mode 2)

The baud rate is doubled when Timer 1 is used as the baud rate generator.

SMOD0: frame error detection control bit

0: no frame error detection function

1: Enable the frame error detection function. At this time, SM0/FE of SCON is FE function, that is, frame error detection flag bit.

15.3.4 Auxiliary Register 1 (AUXR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1BRT

UART_M0x6: Communication speed control for serial port 1 mode 0

0: Baud rate of serial port 1 mode 0 is not doubled, fixed at Fosc/12

1: Baud rate of serial port 1 mode 0 is 6x, i.e. fixed to Fosc/12*6 = Fosc/2

S1BRT: Serial port 1 baud rate transmitter select bit

0: Select Timer 1 as baud rate transmitter

1: Select Timer 2 as baud rate transmitter **(default)**

Note: Serial port 1 uses Timer 2 as the baud rate generator by default, and it is not recommended to use Timer 1. Timer 2 can be shared as the baud rate generator for Serial port 1, Serial port 2, Serial port 3 and Serial port 4 at the same time.

15.3.5 Serial Port 1 Mode 0, Mode 0 Baud Rate Calculation Formula

When serial port 1 selects the working mode as mode 0, the serial communication interface works in synchronous shift register mode, when the communication speed setting bit of serial port mode 0, UART_M0x6, is 0, its baud rate is fixed to 12 divisions of the system clock frequency (SYSclk/12); when UART_M0x6 is set to 1, its baud rate is fixed to 2 divisions of the system clock frequency (SYSclk/2). RxD is the data port for serial communication, TxD is the synchronous shift pulse output pin, sending and receiving 8-bit data, the lower bit comes first.

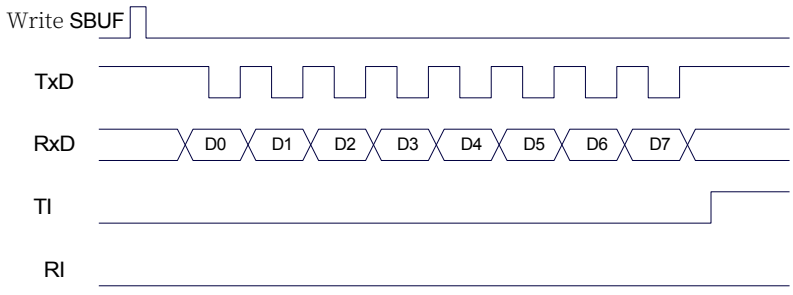
Transmission process of mode 0: When the host executes the instruction to write data into the transmit buffer SBUF, the serial port will output the 8-bit data from the RxD pin at the baud

rate of SYSclk/12 or SYSclk/2 (whether it is a 12-division frequency or a 2-division frequency is determined by the UART_M0x6), and then the interrupt flag TI will be set to 1 after the transmission, and the synchronous shift pulse signal will be output from the TxD pin. TxD pin outputs a synchronous shift pulse signal. When the write signal is active, one clock apart, the transmitter control terminal SEND is active (high), allowing the RxD to transmit data and the TxD to output synchronous shift pulses. When a frame (8-bit) of data has been sent, all the control terminals return to their original state, only TI remains high, which is an interrupt application state. Before sending data again, TI must be cleared by software.

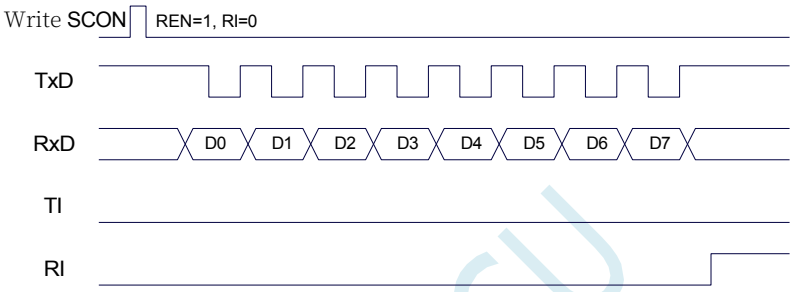
Mode 0 reception process: The mode 0 reception process is initiated by first clearing the reception interrupt request flag RI and setting the allowable reception control bit REN. When the reception process is started, RxD is the serial data input and TxD is the synchronisation pulse output. The baud rate for serial reception is either SYSclk/12 or SYSclk/2 (UART_M0x6 determines whether it is 12-division or 2-division). When the reception of a frame of data (8 bits) is completed, the

When the control signal is reset, the interrupt flag RI is set to 1, and an interrupt request is made.

When receiving again, RI must be cleared to 0 by software.



Send data (serial port 1 mode 0)



Receive data (serial port 1 mode 0)

When operating in mode 0, the multicomputer communication control bit SM2 must be cleared so that it does not affect the TB8 and RB8 bits. Since the baud rate is fixed to SYSclk/12 or SYSclk/2, there is no need for a timer to be supplied, and the microcontroller clock is used directly as the synchronisation shift pulse.

The baud rate calculation formula for Serial Port 1 Mode 0 is shown in the table below (SYSclk is the system operating frequency):

UART_M0x6	Baud rate formula
0	$\text{Baud rate} = \frac{\text{SYSclk}}{12}$
1	$\text{Baud rate} = \frac{\text{SYSclk}}{2}$

15.3.6 Serial Port 1 Mode 1, Mode 1 Baud Rate Calculation Formula

When the software sets SM0 and SM1 of SCON to "01", serial port 1 operates in mode 1. This mode is an 8-bit UART format, and a frame has 10 bits of information: 1 start bit, 8 data bits (low bit first) and 1 stop bit. The baud rate is variable, so you can set the baud rate as needed.

txD is the data transmit port, RxD is the data receive port, and the serial port is full-duplex receive/transmit.

Transmission process of mode 1: In the serial communication mode, data is output from the serial transmitter TxD. When the host executes a write SBUF instruction, the serial communication is started. The write "SBUF" signal also loads a "1" into bit 9 of the transmit shift register and informs the TX control unit to start transmitting. The shift register shifts the data to the right and sends it to the TxD port, and "0" is added to the left side of the data. When the highest bit of the data is shifted to the output position of the shift register, it is followed by bit 9 "1", and all the bits to the left of it are "0".

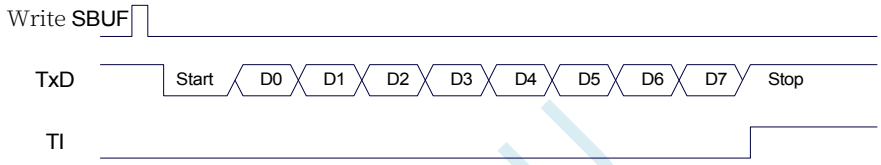
This state condition causes the TX control unit to make the last shift output, then deactivates the "SEND" permission signal, completes the transmission of one frame of information, and sets the interrupt request bit TI, i.e., TI=1, to request interrupt processing from the host computer.

Receiving process of mode 1: When the software sets the receive allow flag REN, i.e. REN=1, the receiver detects the signal from RxD port, and when it detects that the RxD port sends a falling-edge transition from "1" to "0", it starts to receive the data and immediately resets the receive counter of the baud rate generator and loads 1FFH into the shift register. The receiver prepares to receive data and immediately resets the receive counter of the baud rate generator and loads 1FFH into the shift register. The received data is shifted in from the right side of the receive shift register, and the loaded 1FFH is shifted out to the left side, and when the start bit "0" is shifted to the leftmost side of the shift register, the RX controller is shifted for the last time to complete the reception of a frame. If the following two conditions are satisfied simultaneously:

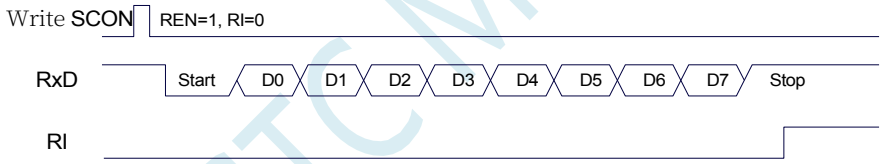
-RI=0;

-SM2=0 or received stop bit is 1.

If the above two conditions cannot be met at the same time, the received data will be invalidated and lost, regardless of whether the conditions are met or not, the receiver will detect the "1"→"0" jump on the RxD port again and continue to receive the next frame. If the reception is valid, after responding to the interrupt, the RI flag bit must be cleared by the software to 0. Normally, when the serial communication operates in mode 1, SM2 is set to "0".



Send Data (Serial Port 1 Mode 1)



Receive data (serial port 1 mode 1)

The baud rate of serial port 1 is variable and can be generated by either Timer 1 or Timer 2. When the timer is in 1T mode (12), the baud rate is variable.

The speed of the baud rate is increased by a factor of 12.

The formula for calculating the baud rate for serial port 1 mode 1 is shown in the following table: (SYSclk is the system operating frequency)

select and fix timers	timers tempo	Formula for calculating reloading value	baud
Timer 2	1T	Timer 2 reload value = 65536 - SYSclk / (4 x Baud Rate)	Baud rate = SYSclk / (4 × (65536 - number of timer reloads))
	12T	Timer 2 reload value = 65536 - SYSclk / (12 x 4 x baud rate)	Baud rate = SYSclk / (12 × 4 × (65536 - number of timer reloads))
Timer 1 mode 0	1T	Timer1 reload value = 65536 - SYSclk / (4 x Baud Rate)	Baud rate = SYSclk / (4 × (65536 - number of timer reloads))
	12T	Timer1 reload value = 65536 - SYSclk / (12 x 4 x baud rate)	Baud rate = SYSclk / (12 × 4 × (65536 - number of timer reloads))
Timer 1 Mode 2	1T	Timer1 reload value = 256 - 2^{SMOD} × SYSclk / (32 x baud rate)	Baud rate = 2SMOD×SYSclk / (32 × (256 - number of timer reloads))
	12T	Timer1 reload value = 256 - 2^{SMOD} × SYSclk / (12 x 32 x baud rate)	Baud rate = 2SMOD×SYSclk / (12 × 32 × (256 - number of timer reloads))

The following are the reload values for timers corresponding to common frequencies and

frequency (MHz)	baud	Timer 2		Timer 1 Mode 0		Timer 1 Mode 2			
						SMOD=1		SMOD=0	
		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

15.3.7 Serial Port 1 Mode 2, Mode 2 Baud Rate Calculation Formula

When SM0 and SM1 are 10, Serial Port 1 operates in Mode 2. Serial Port 1 operates in Mode 2 as a 9-bit data asynchronous communication UART mode, where a frame consists of 11 bits of information: 1 start bit, 8 data bits (lower bit first), 1 programmable bit (9th data bit), and 1 stop bit. The programmable bit (9th bit of data) is provided by TB8 in SCON, which can be set to 1 or 0 by software, or the P value of the odd/even parity bit in PSW can be loaded into TB8 (TB8 can be used as both the address data flag bit and the parity bit of the data for multi-computer communication). When receiving, the 9th bit of data is loaded into RB8 of SCON. TxD is the transmit port and RxD is the receive port to receive/transmit in full-duplex mode.

The baud rate for Mode 2 is fixed at 64 divisions of the system clock or 32 divisions (depending on the value of SMOD in PCON) The baud rate calculation formula for Serial 1 Mode 2 is shown in the table below (SYSclk is the system operating frequency):

SMOD	Baud rate formula
0	$\text{Baud rate} = \frac{\text{SYSclk}}{64}$
1	$\text{Baud rate} = \frac{\text{SYSclk}}{32}$

Compared with Mode 1, Mode 2 has basically the same functional structure and the same receive/transmit operation process and timing sequence, except that the baud rate generator source is slightly different, and the 9th data bit supplied by TB8 to the shift register is different when transmitting.

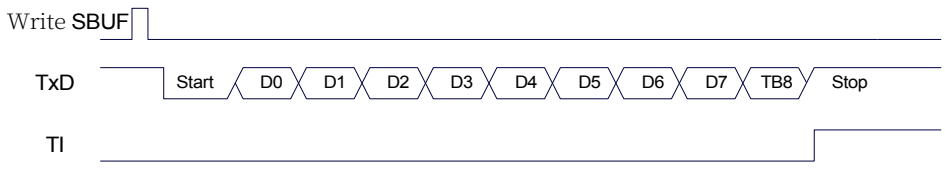
When the receiver finishes receiving a frame of information the following conditions must both be met:

-RI=0

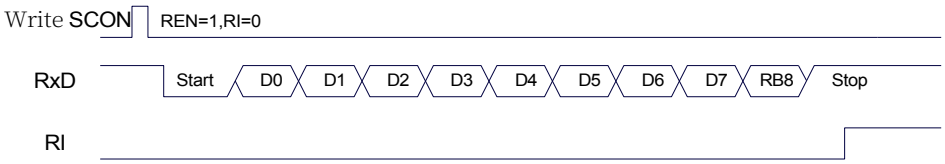
-SM2=0 or SM2=1 and received 9th data bit RB8=1.

When both of the above conditions are satisfied, the data received in the shift register is loaded into SBUF and RB8, the RI flag bit is set to 1, and an interrupt is requested from the host. If one of the above conditions is not satisfied, the data just received in the shift register is invalidated and lost, and RI is not reset; regardless of whether the above conditions are satisfied or not, the receiver resumes detecting the hopping information at the RxD input port and receives the input information for the next frame. In mode 2, the stop bits received are independent of SBUF, RB8, and RI.

The software settings of SM2 and TB8 in SCON and the convention of communication protocols facilitate multi-computer communication.



Send data (serial port 1 mode 2)



Receive data (serial port 1 mode 2)

15.3.8 Serial Port 1 Mode 3, Mode 3 Baud Rate Calculation Formula

When SM0 and SM1 are 11, Serial Port 1 operates in Mode 3. Serial Communication Mode 3 is a 9-bit data asynchronous communication UART mode in which a frame consists of 11 bits of information: 1 start bit, 8 data bits (lower bit first), 1 programmable bit (9th data bit) and 1 stop bit. The programmable bit (9th bit of data) is provided by TB8 in SCON, which can be set to 1 or 0 by software, or the P value of the odd/even parity bit in PSW can be loaded into TB8 (TB8 can be used as both the address data flag bit and the parity bit of the data for multi-computer communication). When receiving, the 9th bit of data is loaded into RB8 of SCON. TxD is the transmit port and RxD is the receive port to receive/transmit in full-duplex mode.

Compared with Mode 1, Mode 3 has basically the same functional structure except for the difference in the 9th data bit supplied by TB8 to the shift register when transmitting, and its receive/transmit operation process and timing sequence are also basically the same.

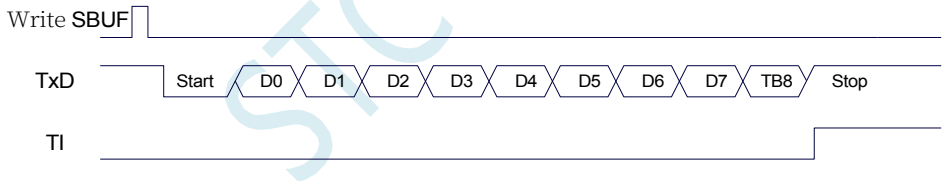
When the receiver finishes receiving a frame of information the following conditions must both be met:

-RI=0

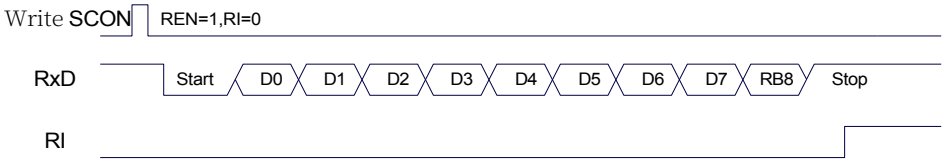
-SM2=0 or SM2=1 and received 9th data bit RB8=1.

When both of the above conditions are satisfied, the data received in the shift register is loaded into SBUF and RB8, the RI flag bit is set to 1, and an interrupt is requested from the host. If one of the above conditions is not satisfied, the data just received in the shift register is invalidated and lost, and RI is not reset; regardless of whether the above conditions are satisfied or not, the receiver resumes detecting the hopping information at the RxD input port and receives the input information for the next frame. In mode 3, the stop bits received are independent of SBUF, RB8, and RI.

The software settings of SM2 and TB8 in SCON and the communication protocol conventions facilitate multi-computer communication.



Send data (serial port 1 mode 3)



Receive data (serial port 1 mode 3)

The baud rate calculation formula for Serial Port 1 Mode 3 is identical to Mode 1. Please refer to the baud rate calculation formula for Mode 1.

15.3.9 Automatic address recognition

15.3.10 Serial Port 1 Slave Address Control Register (SADDR, SADEN)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR: Slave Address Register

SADEN: Slave Address Mask Bit Register

The automatic address recognition function is typically used in the field of multi-computer communication. The main principle is that the slave system identifies the address information in the serial port data stream from the host through the hardware comparison function, and the hardware automatically filters the slave address through the slave address of the local machine set in the registers SADDR and SADEN, and the hardware generates a serial port interrupt when the slave address information from the host matches the slave address of the local machine; otherwise, the hardware automatically discards the serial port data without generating an interrupt. When the slave address information from the host matches the slave address set by the unit, the hardware generates a serial port interrupt; otherwise, the hardware automatically discards the serial port data without generating an interrupt. When many slaves in idle mode are linked together, only the slave with matching slave address will wake up from idle mode, which can greatly reduce the power consumption of the slave MCU, and even if the slave is in normal operation, it can avoid constantly entering the serial port interrupt and reduce the system execution efficiency.

To use the automatic address recognition function of the serial port, you first need to set the serial communication mode of the MCU involved in the communication to Mode 2 or Mode 3 (usually choose Mode 3 with variable baud rate, because the baud rate of Mode 2 is fixed, which is not easy to adjust), and turn on the SM2 bit of the slave's SCON. For the 9-bit data bits of the serial port mode 2 or mode 3, the 9th data bit (stored in RB8) is the address/data flag bit, and when the 9th data bit is 1, it means that the previous 8 data bits (stored in SBUF) are the address information. When SM2 is set to 1, the slave MCU automatically filters out the non-address data (data with bit 9 as 0), and the address data in SBUF is filtered out.
(data with bit 9 as 1) is automatically compared with the local address set by SADDR and SADEN, and if the addresses match, the
RI is set to "1" and an interrupt is generated, otherwise the received serial data will not be processed.

The slave address is set through the SADDR and SADEN registers, SADDR is the slave address register, which stores the local slave address, and SADEN is the slave address mask register, which is used to set the ignore bit in the address information, and the setting method is as follows:

for example
SADDR = 11001010
SADEN = 10000001
then the matching address is 1xxxxxx0
That is, as long as bit0 is 0 and bit7 is 1 in the address data sent from the host, it can match the local address.

another example
SADDR = 11001010
SADEN = 00001111
then the matching address is xxxx1010
That is, as long as the lower 4 bits of the address data sent by the host are 1010, it can match the local address, while the higher 4 bits are ignored and can be any value.

The master can use the broadcast address (FFH) to select all slaves for communication at the same time.

15.3.11 Serial Port 1 Sync Mode Control Register 1 (USARTCR1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR1	7EFDC0H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA

LINEN: LIN mode enable bit

0: LIN mode disabled

1: Enable LIN mode

DORD: SPI mode data bit send/receive sequence

0: Send/receive data high bit first (MSB)

1: Transmit/receive data low bit

first (LSB) CLKEN: SmartCard mode

clock output control bit

0: Clock output disabled

1: Enable clock output
SPMOD: SPI mode
enable bit 0: Disable
SPI mode
1: Enable SPI mode
SPIEN: SPI enable bit
0: Disable SPI function
1: Enable SPI function
SPSLV: SPI Slave Mode Enable Bit
0: SPI is host mode
1: SPI is slave mode
CPOL: SPI Clock Polarity Control
0: SCLK is low when it is idle, the front clock edge of SCLK is rising edge, the back clock edge is falling edge
1: SCLK is high when idle, the front clock edge of SCLK is falling edge, the back clock edge is rising edge
CPHA: SPI clock phase control
0: Data SS pin is low to drive the first bit of data and change the data on the back clock edge of SCLK and sample the data on the front clock edge
1: Data is driven on the front clock edge of SCLK and sampled on the back clock edge

15.3.12 Serial Port 1 Synchronisation Mode Control Register 2 (USARTCR2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR2	7EFDC1H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE

IREN: IrDA mode enable bit
0: Disable IrDA mode
1: Enable IrDA mode IRLP:
IrDA low power mode
control bit 0: IrDA is
normal mode
1: IrDA is in low power mode
SCEN: SmartCard mode
enable bit 0: disable
SmartCard mode 1:
enable SmartCard mode
NACK: SmartCard mode NACK output enable bit
0: Disable NACK signal output
1: Enable output NACK
signal HDSEL: Single-wire
half-duplex mode enable bit
0: Disable single-wire half-
duplex mode
1: Enable single-wire half-
duplex mode PCEN: Hardware auto

generate parity bit control enable

0: Disable automatic hardware generation of the parity bit (the parity bit of the serial port is the value set by TB8)

1: Enable hardware to
generate parity bits
automatically PS: Hardware
parity bit mode selection

0: hardware automatically generates an even parity bit based on the value of SBUF

1: Hardware automatically generates an odd parity bit based on the value of SBUF
PE: Check digit error flag (must be
cleared by software) 0: No
check error

1: There is a parity error (if the received data parity bit error occurs in the process of receiving DMA from the serial port, the DMA will not stop, but the parity bit will not be stopped)

(The error flag will remain until the DMA completes)

15.3.13 Serial Port 1 Sync Mode Control Register 3 (USARTCR3)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR3	7EFDC2H	IrDA_LPBAUD[7:0]							

IrDA_LPBAUD: IrDA Low Power Mode Baud Rate Control Register

IrDA_LPBAUD[7:0]	IrDA Low Power Mode Baud Rate
0	SYSclk/16/256
1	SYSclk/16/1
2	SYSclk/16/2
...	...
255	SYSclk/16/255

15.3.14 Serial Port 1 Synchronisation Mode Control Register 4 (USARTCR4)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR4	7EFDC3H	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]	

SCCKS[1:0]: SmartCard mode clock selection

SCCKS[1:0]	SmartCard Mode Clock
00	SYSclk/64
01	SYSclk/32
10	SYSclk/16
11	SYSclk/8

SPICKS[1:0]: SPI mode clock selection

SPICKS[1:0]	SPI Mode Clock
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/2

15.3.15 Serial Port 1 Synchronisation Mode Control Register 5 (USARTCR5)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USARTCR5	7EFDC4H	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC

BRKDET: LIN from mode interval field (BREAK) detection flag

0: LIN spacing field not detected

1: LIN interval field detected, software clearing required

HDRER: LIN slave mode header (HEADER) error

0: LIN header error not detected

1: LIN header error detected, software clearing required

SLVEN: LIN slave mode enable bit

0: LIN is in host mode

1: LIN is slave mode

ASYNC: LIN auto-sync enable bit

0: Automatic synchronisation disabled

1: Enable automatic synchronisation

TXCF: Transmission Conflict Flag bit. Valid in single-wire half-duplex mode, LIN mode and SmartCard mode.

0: No transmission conflict detected (data sent is the same as data received)

1: Transmission conflict detected (data sent is different from data received)

SENDBK: Send interval field. Software writes 1 to trigger sending interval field, hardware automatically clears 0 after sending is completed.

HDRDET: LIN slave mode header (HEADER) detection flag

0: LIN header not detected

1: LIN header detected, software clearing required

SYNC: LIN slave mode synchronised field detection flag.

When a sync field is correctly analysed, the hardware sets the SYNC flag to 1. The hardware automatically clears the SYNC when receiving a marker field.

15.3.16 Serial Port 1 Synchronised Mode Protection Time Register (USARTGTR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USARTGTR	7EFDC5H								

The USARTGTR register is only valid in SmartCard mode. This register is used to give a protection time value based on the number of baud rate clocks. the TI flag is set by hardware to 1 when the data bits sent by the SmartCard are equal to the protection time value set by the USARTGTR register. note: the value of the USARTGTR register should be greater than 11

15.3.17 Serial Port 1 Synchronous Mode Baud Rate Register (USARTBR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USARTBRH	7EFDC6H	USARTBR[15:8]							
USARTBRL	7EFDC7H	USARTBR[7:0]							

Baud rate calculation formula for LIN mode, IrDA mode and SmartCard mode

$$\text{Synchronised baud rate} = \frac{\text{SYSclk}}{16 * \text{usartbr}[15:0]}$$

15.4 Serial port 2 (synchronous/asynchronous serial port USART2)

15.4.1 Serial 2 Control Register (S2CON)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0/FE	S2SM1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0/FE: When the S2MOD0 bit in the S2CFG register is 1, this bit is the frame error detection flag bit. When the UART2 detects an invalid stop bit during reception, this bit is 1 through the UART2 receiver and must be cleared by software. When the S2MOD0 bit in the S2CFG register is 0, this bit, along with S2SM1, specifies the communication operating mode of the serial port 2, as shown in the following table:

S2SM0	S2SM1	Serial 2 operating mode	Functional Description
0	0	Mode 0	Synchronous shift serial method
0	1	Model 1	Variable baud rate 8-bit data mode
1	0	Model 2	Fixed baud rate 9-bit data mode
1	1	Mode 3	Variable baud rate 9-bit data mode

S2SM2: Allow mode 2 or mode 3 multicomputer communication control bit. When Serial Port 2 uses Mode 2 or Mode 3, if the S2SM2 bit is 1 and the S2REN bit is 1, the receiver is in the address frame screening state. At this time, the received bit 9 (i.e. S2RB8) can be used to screen the address frame, if S2RB8=1, it means that the frame is an address frame, the address information can be entered into the S2BUF, and make S2RI is 1, and then in the interrupt service program to compare the address number; if S2RB8=0, it means that the frame is not an address frame, and it should be thrown away and keep S2RI=0. In mode 2 or 3, if S2SM2 bit is 0, the receiver is in an address frame screening state, if S2SM2 bit is 0, the receiver is in an address frame screening state. In mode 2 or 3, if the S2SM2 bit is 0 and the S2REN bit is 1, the receiving transceiver is in the address frame filtering disabled state, regardless of whether the received S2RB8 is 0 or 1, it can make the received information into the S2BUF, and make the S2RI=1, at this time, S2RB8 is usually the parity bit. Mode 1 and Mode 0 are non-multi-computer communication modes, and S2SM2 should be set to 0 in these two modes.

S2REN: Allow/Disallow serial port receive control bit 0:
Disable serial port receive data
1: Allow serial port to receive data

S2TB8: S2TB8 is the 9th bit of data to be sent when Serial 2 is used in Mode 2 or Mode 3, and is set or cleared by software as required.

This bit is not used in Mode 0 and Mode 1.

S2RB8: S2RB8 is the 9th bit of data received when Serial 2 is used in Mode 2 or Mode 3, usually used as a parity bit or address frame.

/Data frame flag bit. This bit is not used in Mode 0 and Mode 1.

S2TI: Serial 2 transmit interrupt request flag bit. In mode 0, when the serial port sends data at the end of bit 8, the hardware will automatically set S2TI to 1 to request an interrupt from the host computer, and S2TI must be cleared by software after responding to the interrupt. In other modes, the hardware automatically sets S2TI to 1 when the stop bit starts to be sent and requests an interrupt from the CPU, and S2TI must be cleared by software after responding to the interrupt.

S2RI: Serial port 2 receive interrupt request flag bit. In mode 0, when the serial port receives the end of the 8th bit of data, the hardware will automatically set S2RI to 1 to request an interrupt from the host, and S2RI must be cleared by software after responding to the interrupt. In other modes, the hardware automatically sets S2RI to 1 in the middle of receiving stop bit and sends an interrupt request to the CPU, and S2RI must be cleared by software after responding to the interrupt.

15.4.2 Serial Port 2 Data Register (S2BUF)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

S2BUF: Serial port 1 data receive/send buffer.S2BUF is actually 2 buffers, read buffer and write buffer, the two operations correspond to two different registers respectively, one is write-only register (write buffer) and one is read-only register (read buffer). For

A read operation to S2BUF actually reads the serial port receive buffer, and a write operation to S2BUF triggers the serial port to start sending data.

15.4.3 Serial Port 2 Configuration Register (S2CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
S2CFG	7EFDB4H	-	S2MOD0	S2M0x6					W1

S2MOD0: frame error
detection control bit 0:
no frame error
detection function
1: Enable the frame error detection function. At this time, S2SM0/FE of S2CON is FE function, that is, it is the frame error detection flag bit.

S2M0x6: Communication speed control for serial port 2 mode 0
0: The baud rate of serial port 2 mode 0 is not doubled, and is fixed to Fosc/12.
1: Baud rate of serial port 2 mode 0 is 6 times faster, i.e. fixed to Fosc/12*6 = Fosc/2

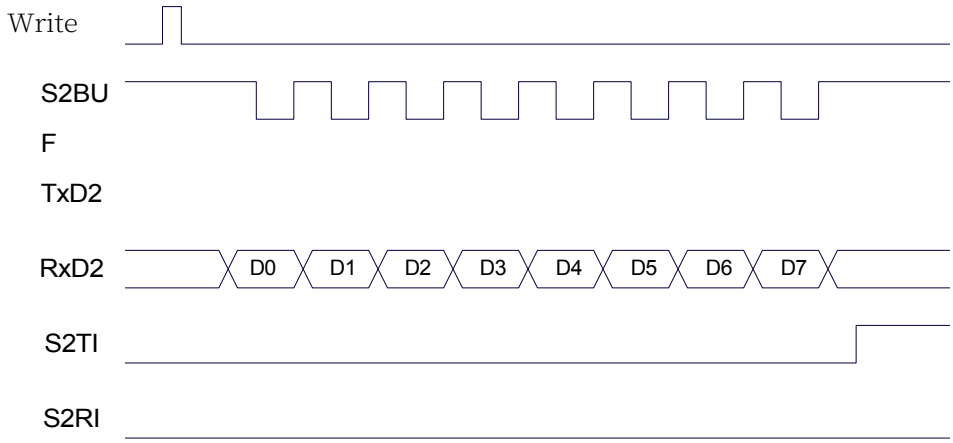
W1: This bit must be set to "1" when serial port 2 is required, otherwise unanticipated errors may occur. If the serial port is not required
2, you do not need to set W1 specifically.

15.4.4 Serial Port 2 Mode 0, Mode 0 Baud Rate Calculation Formula

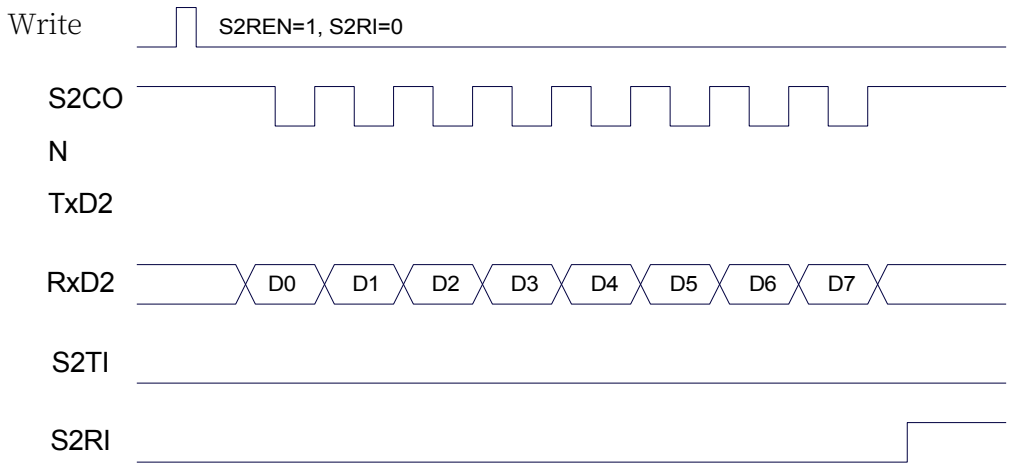
When serial port 2 selects the working mode as mode 0, the serial communication interface works in synchronous shift register mode, when the communication speed setting bit S2M0x6 of serial port mode 0 is 0, its baud rate is fixed to 12 divisions of the system clock frequency (SYSclk/12); when S2M0x6 is set to 1, its baud rate is fixed to 2 divisions of the system clock frequency (SYSclk/2). RxD2 is the data port for serial communication, and TxD2 is the synchronous shift pulse output pin, which sends and receives 8-bit data, with the lower bit first.

Transmission process of mode 0: When the host executes the instruction to write data into the transmit buffer S2BUF, the serial port will output 8-bit data at the baud rate of SYSclk/12 or SYSclk/2 (whether it is 12-division frequency or 2-division frequency is determined by S2M0x6) from the RxD2 pin (from the low bit to the high bit), and after the transmission, the interrupt flag S2TI will be set to 1, and the synchronous shift pulse signal will be outputted from the TxD2 pin. Synchronous shift pulse signal. When the write signal is active, one clock apart, the transmitter control SEND is active (high), allowing RxD2 to transmit data and TxD2 to output synchronous shift pulses. When a frame (8-bit) of data has been sent, all the control terminals return to their original state, except S2TI, which remains high and is an interrupt request state. S2TI must be cleared by software before sending data again.

Mode 0 reception process: The mode 0 reception process is initiated by first clearing the receive interrupt request flag S2RI to zero and setting the allowable reception control bit S2REN. When the reception process is started, RxD2 is the serial data input and TxD2 is the sync pulse output. The baud rate for serial reception is SYSclk/12 or SYSclk/2 (S2M0x6 determines whether it is 12-division or 2-division). When a frame of data (8 bits) is received, the control signal is reset, the interrupt flag S2RI is set to 1, and an interrupt request is made. When receiving again, S2RI must be cleared to 0 by software.



Send data (serial port 2 mode 0)



Receive data (serial port 2 mode 0)

When operating in mode 0, the multicomputer communication control bit S2SM2 must be cleared so that it does not affect the S2TB8 bit and the S2RB8 bit. Since the baud rate is fixed to SYSclk/12 or SYSclk/2, there is no need for a timer to be supplied, as the microcontroller clock is used directly as the synchronisation shift pulse.

The baud rate calculation formula for Serial Port 2 Mode 0 is shown in the table below (SYSclk is the system operating frequency):

S2M0x6	Baud rate formula
0	$\text{Baud rate} = \frac{\text{SYSclk}}{12}$
1	$\text{Baud rate} = \frac{\text{SYSclk}}{2}$

15.4.5 Serial Port 2 Mode 1, Mode 1 Baud Rate Calculation Formula

When S2SM0 and S2SM1 of S2CON are set to "01" by software, serial port 2 operates in mode 1. This mode is an 8-bit UART format, and a frame consists of 10 bits of information: 1 start bit, 8 data bits (low bit first) and 1 stop bit. The baud rate is variable, so you can set the baud rate according to your needs. txD2 is the data transmit port, RxD2 is the data receive port, and the serial port is full-duplex receive/transmit.

Transmission process of mode 1: In the serial communication mode, data is output from the serial transmitter TxD2. When the host executes an instruction to write S2BUF, the serial transmission is initiated. The write "S2BUF" signal also loads a "1" into bit 9 of the transmit shift register and informs the TX control unit to start transmission. The shift register shifts the data to the right and sends it to the TxD port, and "0" is added to the left side of the data. When the highest bit of the data is shifted to the output position of the shift register, followed by bit 9

"1", and all the bits to the left of it are "0", this state condition causes the TX control unit to make the last shift output, and then the signal "SEND" is allowed to be sent. The TX control unit makes the last shift output and then deactivates the "SEND" signal, completes the transmission of one frame of information, and sets the interrupt request bit S2TI, i.e., S2TI=1, and requests the host for interrupt processing.

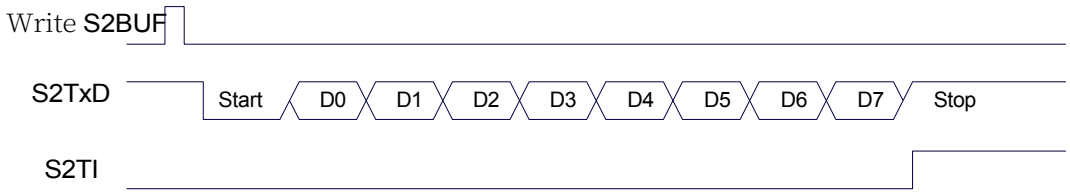
Mode 1 reception process: When the software sets the receive allow flag S2REN, i.e. S2REN=1, the receiver detects the signal of RxD2 port, and when it detects that the RxD2 port sends a falling-edge transition from "1" to "0", it starts the receiver to prepare for receiving data, and immediately resets the receive counter of the baud rate generator, and loads 1FFH into the shift register. When it detects the RxD2 port sends a falling edge jump from "1" to "0", the receiver is ready to receive the data, and reset the receive counter of the baud rate generator immediately, and load 1FFH into the shift register. The received data is shifted in from the right side of the receive shift register, and the loaded 1FFH is shifted out to the left side, and when the start bit "0" is shifted to the leftmost side of the shift register, the RX controller is shifted for the last time to complete the reception of a frame. If the following two conditions are satisfied simultaneously:

-S2RI=0;

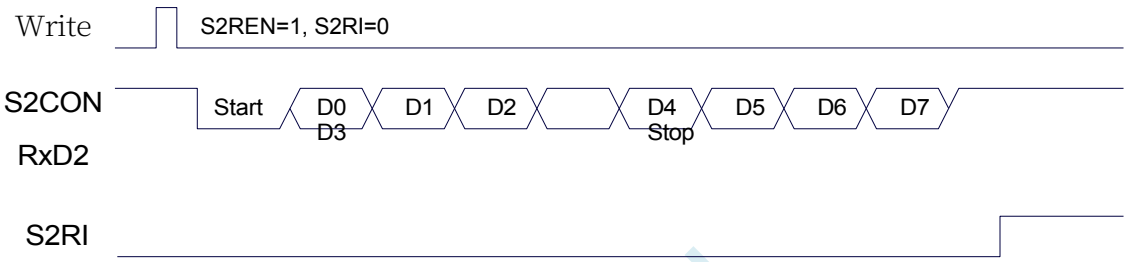
-S2SM2=0 or received stop bit is 1.

Then the received data is valid, the implementation is loaded into S2BUF, the stop bit goes into S2RB8, the S2RI flag bit is set to 1, and a request is made to the host in the

If the above two conditions cannot be satisfied at the same time, the received data will be invalidated and lost, regardless of whether the conditions are satisfied or not, the receiver will detect the "1"→"0" jump on the RxD2 port again and continue to receive the next frame. If the reception is valid, after responding to the interrupt, the S2RI flag bit must be cleared by the software to 0. Normally, when the serial communication operates in mode 1, S2SM2 is set to "0".



Send Data (Serial 2 Mode 1)



Receive data (Serial 2 mode 1)

The baud rate of serial port 2 is variable and its baud rate is fixed by timer 2. When the timer is in 1T mode (12x speed), the corresponding baud rate is increased by a factor of 12.

The baud rate calculation formula for Serial 2 Mode 1 is shown in the following table: (SYSclk is the system operating frequency)

Select Timing tool	timer speed degree (angles, temperature etc)	Formula for calculating reloading value	baud
Timer 2	1T	Timer 2 reload value = 65536 - SYSclk / (4 x Baud Rate)	Baud rate = SYSclk / (4 × (65536 - number of timer reloads))
	12T	Timer 2 reload value = 65536 - SYSclk / (12 x 4 x baud rate)	Baud rate = SYSclk / (12 × 4 × (65536 - number of timer reloads))

The following are the reload values for timers corresponding to common frequencies and common baud rates

frequency (MHz)	baud	Timer 2	
		1T mode	12T mode
11.0592	115200	FFE8H	FFFEH
	57600	FFD0H	FFFCH
	38400	FFB8H	FFFAH
	19200	FF70H	FFF4H
	9600	FEE0H	FFE8H
18.432	115200	FFD8H	-
	57600	FFB0H	-
	38400	FF88H	FFF6H
	19200	FF10H	FFECH
	9600	FE20H	FFD8H
22.1184	115200	FFD0H	FFFCH
	57600	FFA0H	FFF8H
	38400	FF70H	FFF4H
	19200	FEE0H	FFE8H
	9600	FDC0H	FFD0H

15.4.6 Serial 2 Mode 2, Mode 2 Baud Rate Calculation Formula

When S2SM0 and S2SM1 are 10, Serial Port 2 operates in Mode 2. Serial Port 2 operates in Mode 2 as a 9-bit data asynchronous communication UART mode, where the information of a

frame consists of 11 bits: 1 start bit, 8 data bits (lower bit first), 1 programmable bit (9th data bit), and 1 stop bit. The programmable bit (9th bit of data) is provided by S2TB8 in S2CON, which can be set to 1 or 0 by software, or the P value of the odd/even parity bit in the PSW can be loaded into S2TB8 (S2TB8 can be used as both the address data flag bit and the parity bit of the data for multi-computer communication). When receiving, the 9th bit of data is loaded into S2RB8 of S2CON. TxD2 is the transmitting port and RxD2 is the receiving port to receive/transmit in full duplex mode.

The baud rate for Mode 2 is fixed at 64 divisions of the system clock or 32 divisions (depending on the value of S2MOD in S2CFG) The baud rate calculation formula for Serial 2 Mode 2 is shown in the table below (SYSclk is the system operating frequency):

SMOD	Baud rate formula

0	$\text{Baud rate} = \frac{\text{SYSclk}}{64}$
1	$\text{Baud rate} = \frac{\text{SYSclk}}{32}$

Compared with Mode 1, Mode 2 has basically the same functional structure and the same receive/transmit operation process and timing sequence, except that the baud rate generator source is slightly different, and the 9th data bit supplied by S2TB8 to the shift register is different when transmitting.

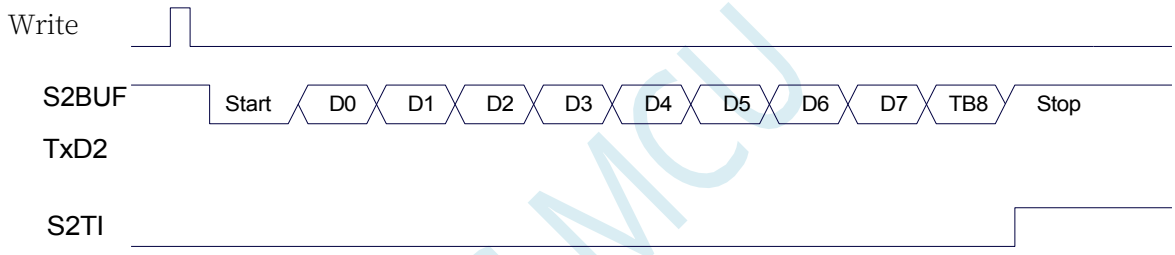
When the receiver finishes receiving a frame of information the following conditions must both be met:

-S2RI=0

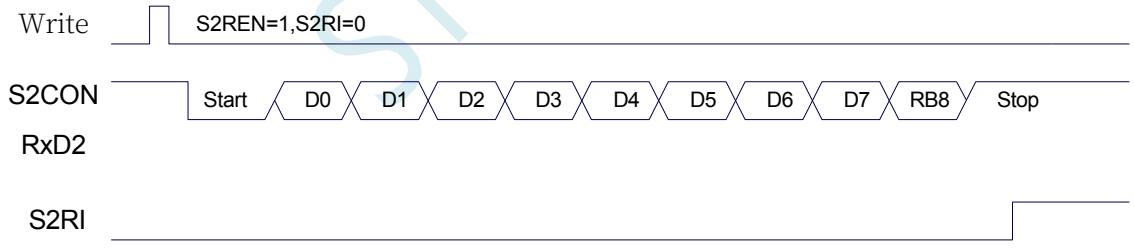
-S2SM2=0 or S2SM2=1 and received 9th data bit S2RB8=1.

When both of the above conditions are met, the received shift register data is loaded into S2BUF and S2RB8, the S2RI flag bit is set to 1, and an interrupt is requested from the host. If one of the above conditions is not met, the data just received in the shift register is invalidated and lost, and S2RI is not reset; regardless of whether the above conditions are met or not, the receiver resumes detecting the hopping information at the RxD2 input port and receives the input information for the next frame. In mode 2, the stop bits received are independent of S2BUF, S2RB8, and S2RI.

The software settings of S2SM2 and S2TB8 in S2CON and the communication protocol conventions provide convenience for multi-computer communication.



Send Data (Serial 2 Mode 2)



Receive data (serial port 2 mode 2)

15.4.7 Serial Port 2 Mode 3, Mode 3 Baud Rate Calculation Formula

When S2SM0 and S2SM1 are 11, Serial Port 2 operates in Mode 3. Serial Communication Mode 3 is a 9-bit data asynchronous communication UART mode in which a frame consists of 11 bits of information: 1 start bit, 8 data bits (lower bit first), 1 programmable bit (9th data bit), and 1 stop bit. The programmable bit (9th bit of data) is provided by S2TB8 in S2CON, which can be set to 1 or 0 by software, or the P value of the odd/even parity bit in the PSW can be loaded into S2TB8 (S2TB8 can be used as both the address data flag bit and the parity bit of the data for multi-computer communication). When receiving, the 9th bit of data is loaded into S2RB8 of S2CON. TxD2 is the transmitting port and RxD2 is the receiving port to receive/transmit in full duplex mode.

Compared with Mode 1, Mode 3 has basically the same functional structure except for the difference in the 9th data bit supplied by S2TB8 to the shift register when transmitting, and its receive/transmit operation process and timing sequence are also basically the same.

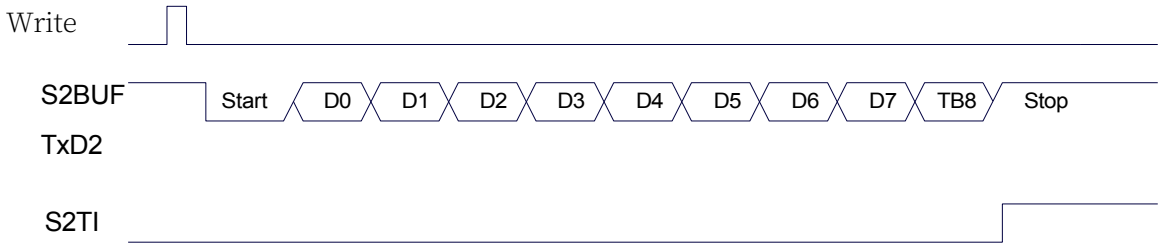
When the receiver finishes receiving a frame of information the following conditions must both be met:

-S2RI=0

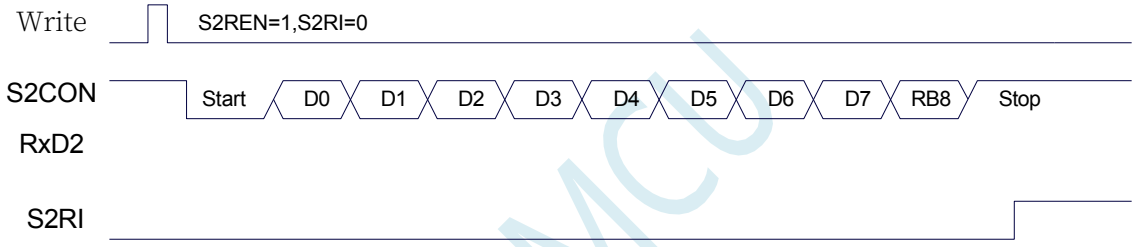
-S2SM2=0 or S2SM2=1 and received 9th data bit S2RB8=1.

When both of the above conditions are met, the received shift register data is loaded into S2BUF and S2RB8, the S2RI flag bit is set to 1, and an interrupt is requested from the host. If one of the above conditions is not met, the data just received in the shift register is invalidated and lost, and S2RI is not reset; regardless of whether the above conditions are met or not, the receiver resumes detecting the hopping information at the RxD2 input port and receives the input information for the next frame. In mode 3, the received stop bits are independent of S2BUF, S2RB8, and S2RI.

The software settings of S2SM2 and S2TB8 in S2CON and the communication protocol conventions facilitate multi-computer communication.



Send data (serial port 2 mode 3)



Receive data (serial port 2 mode 3)

The baud rate calculation formula for Serial 2 Mode 3 is identical to Mode 1. Please refer to the baud rate calculation formula for Mode 1.

15.4.8 Serial Port 2 Automatic Address Recognition

15.4.9 Serial 2 Slave Address Control Register (S2ADDR, S2ADEN)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
S2ADDR	7EFDB5H								
S2ADEN	7EFDB6H								

S2ADDR: Slave Address Register S2ADEN: Slave Address Mask Bit Register

The automatic address recognition function is typically used in multi-computer communication. The main principle is that the slave system identifies the address information in the serial port 2 data stream from the host through the hardware comparison function, and the hardware automatically filters the slave address through the slave address of the local machine set in registers S2ADDR and S2ADEN, and the hardware automatically filters the slave address, and when the slave address information from the host matches with the slave address of the local machine, the hardware generates a serial port 2 interrupt; otherwise, the hardware automatically discards the serial port 2 data without generating an interrupt. When the slave address information from the host matches the slave address set by the unit, the hardware generates the Serial 2 interrupt; otherwise, the hardware automatically discards the Serial 2 data without generating an interrupt. When many slaves in idle mode are linked together, only the slave with matching slave address will wake up from idle mode, which can greatly reduce the power consumption of the slave MCU, and even if the slave is in normal operation, it can also avoid constantly entering the Serial 2 interrupt, which reduces the efficiency of the system execution.

To use the automatic address recognition function of Serial 2, you need to set the Serial 2 communication mode of the MCU involved in the communication to Mode 1, and turn on the S2SM2 bit of the slave's S2CON. For the 9-bit data bit of Serial 2 Mode 1, the 9th data bit (stored in S2RB8) is the address/data flag bit, when the 9th data bit is 1, it means that the previous 8 data bits (stored in S2BUF) are the address signal.

Message. When S2SM2 is set to 1, the slave MCU will automatically filter out the non-address data (data with bit 9 as 0), and compare the address data in S2BUF (data with bit 9 as 1) with the local address set by S2ADDR and S2ADEN, and if the addresses match, then S2RI will be set to 1 and an interrupt will be generated. If the address matches, S2RI will be set to "1" and an interrupt will be generated, otherwise, the received serial port 2 data will not be processed.

The slave address is set through S2ADDR and S2ADEN registers, S2ADDR is the slave address register, which stores the slave address of the local machine, S2ADEN 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
S2ADDR = 11001010
S2ADEN = 10000001
then the matching address is 1xxxxxx0
That is, as long as bit0 is 0 and bit7 is 1 in the address data sent from the host, it can match the local address.

another example
S2ADDR = 11001010
S2ADEN = 00001111
then the matching address is xxxx1010
That is, as long as the lower 4 bits of the address data sent by the host are 1010, it can match the local address, while the higher 4 bits are ignored and can be any value.

The master can use the broadcast address (FFH) to select all slaves for communication at the same time.

15.4.10 Serial Port 2 Sync Mode Control Register 1 (USART2CR1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR1	7EFDC8H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPSLV	CPOL	CPHA

LINEN: LIN mode enable bit
0: LIN mode disabled
1: Enable LIN mode
DORD: SPI mode data bit send/receive sequence
0: Send/receive data high bit first (MSB)
1: Transmit/receive data low bit
first (LSB) CLKEN: SmartCard mode
clock output control bit
0: Clock output disabled
1: Enable clock output
SPMOD: SPI mode
enable bit 0: Disable
SPI mode
1: Enable SPI mode
SPIEN: SPI enable bit

0: Disable SPI function

1: Enable SPI function

SPSLV: SPI Slave Mode Enable Bit

0: SPI is host mode

1: SPI is slave mode

CPOL: SPI Clock Polarity Control

0: SCLK is low when it is idle, the front clock edge of SCLK is rising edge, the back clock edge is falling edge

1: SCLK is high when idle, the front clock edge of SCLK is falling edge, the back clock edge is rising edge

CPHA: SPI clock phase control

0: Data SS pin is low to drive the first bit of data and change the data on the back clock edge of SCLK and sample the data on the front clock edge

1: Data is driven on the front clock edge of SCLK and sampled on the back clock edge

15.4.11 Serial 2 Sync Mode Control Register 2 (USART2CR2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR2	7EFDC9H	IREN	IRLP	SCEN	NACK	HDSEL	PCEN	PS	PE

IREN: IrDA mode enable bit

0: Disable IrDA mode

1: Enable IrDA mode IRLP: IrDA low power mode control bit 0: IrDA is normal mode

1: IrDA is in low power mode

SCEN: SmartCard mode enable bit 0: disable SmartCard mode 1: enable SmartCard mode

NACK: SmartCard mode NACK output enable bit

0: Disable NACK signal output

1: Enable output NACK signal HDSEL: Single-wire half-duplex mode enable bit

0: Disable single-wire half-duplex mode

1: Enable single-wire half-duplex mode PCEN: Hardware auto generate parity bit control enable

0: Disable automatic hardware generation of the parity bit (the parity bit of the serial port is the value set by S2TB8)

1: Enable hardware to generate parity bits automatically PS: Hardware parity bit mode selection

0: Hardware automatically generates an even parity bit according to the value of S2BUF

1: Hardware automatically generates an odd parity bit based on the value of S2BUF

PE: Check digit error flag (must be cleared by software) 0: No check error

1: There is a parity error (if a receive data parity bit error occurs during the serial port receive DMA process, the DMA will not stop, but the parity bit error flag will remain until the DMA completes)

15.4.12 Serial Port 2 Sync Mode Control Register 3 (USART2CR3)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR3	7EFDCAH	IrDA_LPBAUD[7:0]							

IrDA_LPBAUD: IrDA Low Power Mode Baud Rate Control Register

IrDA_LPBAUD[7:0]	IrDA Low Power Mode Baud Rate
0	SYSclk/16/256
1	SYSclk/16/1
2	SYSclk/16/2
...	...

255	SYSclk/16/255

15.4.13 Serial Port 2 Sync Mode Control Register 4 (USART2CR4)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR4	7EFDCBH	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]	

SCCKS[1:0]: SmartCard mode clock selection

SCCKS[1:0]	SmartCard Mode Clock
00	SYSclk/64
01	SYSclk/32
10	SYSclk/16
11	SYSclk/8

SPICKS[1:0]: SPI mode clock selection

SPICKS[1:0]	SPI Mode Clock
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/2

15.4.14 Serial Port 2 Sync Mode Control Register 5 (USART2CR5)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USART2CR5	7EFDCCH	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC

BRKDET: LIN from mode interval field (BREAK) detection flag

0: LIN spacing field not detected

1: LIN interval field detected, software clearing required

HDRER: LIN slave mode header (HEADER) error

0: LIN header error not detected

1: LIN header error detected, software clearing required

SLVEN: LIN slave mode enable bit

0: LIN is in host mode

1: LIN is slave mode

ASYNC: LIN auto-sync enable bit

0: Automatic synchronisation disabled

1: Enable automatic synchronisation

TXCF: Transmission Conflict Flag bit. Valid in single-wire half-duplex mode, LIN mode and SmartCard mode.

0: No transmission conflict detected (data sent is the same as data received)

1: Transmission conflict detected (data sent is different from data received)

SENDBK: Send interval field. Software writes 1 to trigger sending interval field, hardware automatically clears 0 after sending is completed.

HDRDET: LIN slave mode header (HEADER) detection flag

0: LIN header not detected

1: LIN header detected, software clear required

SYNC: LIN slave mode synchronised field detection flag.

When a sync field is correctly analysed, the hardware sets the SYNC flag to 1. The hardware automatically clears the SYNC when receiving a marker field.

15.4.15 Serial Port 2 Synchronous Mode Protection Time Register (USART2GTR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USART2GTR	7EFDCDH								

The USARTGTR register is only valid in SmartCard mode. This register is used to give a protection time value based on the number of baud rate clocks. the S2TI flag is set by hardware to 1 when the data bits sent by the SmartCard are equal to the protection time value set by the USARTGTR register. note: the value of the USARTGTR register should be greater than 11

15.4.16 Serial Port 2 Synchronous Mode Baud Rate Register (USART2BR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USART2BRH	7EFDCEH	USART2BR[15:8]							
USART2BRL	7EFDCFH	USART2BR[7:0]							

Baud rate calculation formula for LIN mode, IrDA mode and SmartCard mode

$$\text{Synchronised baud rate} = \frac{\text{SYSclk}}{16 * \text{usart2br[15:0]}}$$

15.5 sample procedure

15.5.1 Serial port 1 uses timer 2 as a baud rate generator.

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - (FOSC / 115200+2) / 4)

                                        //Define as unsigned long integer to avoid overflow.
                                        The // plus 2 operation is to allow the Keil compiler to
                                        //Automatic implementation of rounding operations

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;
    S1BRT = 1.
    T2x12 = 1.
    T2R = 1.
    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()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    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.
        }
    }
}
```


15.5.2 Serial port 1 uses timer 1 (mode 0) as baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL          //Define as unsigned long integer to avoid overflow.
```

```
(65536 - (FOSC / 115200+2) / 4)
```

```
The // plus 2 operation is to allow the Keil compiler to
//Automatic implementation of rounding operations
```

```
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;
    s1brt = 0; tmod
    = 0x00; tl1 =
    brt.
    TH1 = BRT >> 8;
    TR1 = 1;
    T1x12 = 1.
    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()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign a value of 0 to set the CPU to execute the
                                programme as fast as possible.

    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.
        }
    }
}
```

15.5.3 Serial port 1 uses timer 1 (mode 2) as baud rate generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                 // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL              //Define as unsigned long integer to avoid overflow.
#define     BRT             (256 - (FOSC / 115200+16) / 32)
                                                    The //add 16 operation is designed to allow the Keil compiler to
                                                    //Automatic implementation of rounding operations

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;
        s1brt = 0; tmod
        = 0x20; tl1 =
        brt; th1 = brt;
        tr1 = 1.
        T1x12 = 1.
        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()
{
        EAXFR = 1;                  //Enable access to XFR
        CKCON = 0x00;               //Set the external data bus speed to fastest
        WTST = 0x00;                //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

        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.
        }
    }
}
```

15.5.4 Serial 2 Use Timer 2 as a baud rate generator.

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL      //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)

The // plus 2 operation is to allow the Keil compiler to
//Automatic implementation of rounding operations


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

void Uart2Isr() interrupt 8
{
    if (S2TI)
    {
        S2TI = 0;
        busy = 0;
    }
    if (S2RI)
    {
        S2RI = 0;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f.
    }
}

void Uart2Init()
{
    p_sw2 = 0x80;
    s2cfg = 0x01.

    s2con = 0x50;
    t2l = brt.
    T2H = BRT >> 8;
    T2x12 = 1.
    T2R = 1.
    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()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign a value of 0 to set the CPU to execute the
                                programme as fast as possible.

    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.
        }
    }
}
```


15.5.5 Serial FLASH access using the SPI interface of USART1 (DMA mode)

//Test operating frequency is *24MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"
#include "stdio.h"


#define    FOSC    24000000UL                          //System operating frequency
#define    BAUD    (65536 - (fosc/115200+2)/4)         //Debug serial port baud rate
                                                       The // plus 2 operation is to allow the Keil compiler to
                                                       //Automatic implementation of rounding operations

typedef    bit                 BOOL.
typedef    unsigned char       BYTE;
typedef    unsigned int        WORD.
typedef    unsigned long       DWORD;

sbit   S1SS     =    P2^2.
sbit   S1MOSI   =    P2^3.
sbit   S1MISO   =    P2^4.
sbit   S1SCLK   =    P2^5.

void sys_init();
void usart1_spi_init().
void usart1_tx_dma(WORD size, BYTE xdata *pdat);
void usart1_rx_dma(WORD size, BYTE xdata *pdat);
BOOL flash_is_busy();
void flash_read_id();
void flash_read_data(DWORD addr, WORD size, BYTE xdata
*pdat); void flash_write_enable().
void flash_write_data(DWORD addr, WORD size, BYTE xdata *pdat);
void flash_erase_sector(DWORD addr).

BYTE xdata buffer1[256].                               //define the buffer
BYTE xdata buffer2[256].                               //Note: If you need to use DMA to send data, you can use
DMA to send data.

                                                       // then the buffer must be defined in the xdata area.


void main()
{
    int    i;

    sys_init().                                        //System initialisation
    usart1_spi_init().                                 //USART1 enables SPI mode initialisation.

    printf("\r\nUSART_SPI_DMA test ! \r\n");
    flash_read_id().
    flash_read_data(0x0000, 0x80, buffer1); //read data from external FLASH.      //Read the data from the external FLASH.
    flash_erase_sector(0x0000).                        //Erase one sector of the external FLASH.
    flash_read_data(0x0000, 0x80, buffer1);
    for (i=0; i<128; i++)
        buffer2[i] = i;
    flash_write_data(0x0000, 0x80, buffer2);           //Data to external FLASH
    flash_read_data(0x0000, 0x80, buffer1);

    while (1);
}
```

```
void UartPutchar(char dat)
{
    while (!S2TI);
```

```
        s2ti = 0; s2buf
        = dat.

        return dat.
}

void sys_init()
{
        wtst = 0x00;
        ckcon = 0x00;
        eaxfr = 1.

        p0m0 = 0x00; p0m1 = 0x00;
        p1m0 = 0x00; p1m1 = 0x00;
        p2m0 = 0x00; p2m1 = 0x00;
        p3m0 = 0x00; p3m1 = 0x00;
        p4m0 = 0x00; p4m1 = 0x00;
        p5m0 = 0x00; p5m1 = 0x00;
        p6m0 = 0x00. P6M1 = 0x00;
        P7M0 = 0x00; P7M1 = 0x00.

        S2_S = 1; //Initialise serial port 2 for debugging.        //Initialise serial port 2 for debugging.
        S2CON = 0x52.
        t2l = baud; t2h =
        baud >> 8; t2x12
                =1.
        T2R = 1.
}

void usart1_spi_init()
{
        S1SPI_S0 = 1;                                             //Switch S1SPI to
                                                                  //P2.2/S1SS,P2.3/S1MOSI,P2.4/S1MISO,P2.5/S1SCLK
        s1spi_s1 = 0;
        scon = 0x10.                                              // Enable receive, must be set to serial mode 0

        USARTCR1 = 0x10;                                          //Enable SPI mode on USART1.
//      USARTCR1 |= 0x40;                                         //DORD=1
        USARTCR1 &= ~0x40;                                        //DORD=0
//      USARTCR1 |= 0x04;                                         //Slave Mode
        USARTCR1 &= ~0x04;                                        // Host mode
        USARTCR1 |= 0x00;                                         //CPOL=0, CPHA=0
//      USARTCR1 |= 0x01.                                         //CPOL=0, CPHA=1
//      USARTCR1 |= 0x02 ;- USARTCR1 |= 0x02                      //CPOL=1, CPHA=0
        ;- USARTCR2 |= 0x02
//      USARTCR1 |= 0x03;                                         //CPOL=1, CPHA=1
        USARTCR4 = 0x00.                                          //SPI speed of SYSCLK/4
//      USARTCR4 = 0x01.                                          //SPI speed of SYSCLK/8
//      USARTCR4 = 0x02.                                          //SPI speed of SYSCLK/16
//      USARTCR4 = 0x03;                                          //SPI speed of SYSCLK/2
        USARTCR1 |= 0x08.                                         // Enable SPI function
}

BYTE usart1_spi_shift(BYTE dat)
{
        TI = 0;
        SBUF = dat.                                               //send data
        while (!TI);                                              //The TI flag is the host mode transmit/receive data
        completion flag.
```

```
    return SBUF;                    //Read the received data
```

```
}

BOOL flash_is_busy()
{
    BYTE dat;

    S1SS = 0.

    usart1_spi_shift(0x05).                 //Send the command to read the status register.
    dat = usart1_spi_shift(0);              //read the status register

    S1SS = 1.

    return (dat & 0x01); //Detect FLASH busy flag.    //Detect FLASH busy flag
}

void flash_read_id()
{
    BYTE id[3];

    S1SS = 0;

    usart1_spi_shift(0x9f).                 //Send the command to read FLASHID.
    id[0] = usart1_spi_shift(0);            //read ID1
    id[1] = usart1_spi_shift(0);            //read ID2
    id[2] = usart1_spi_shift(0);            //read ID3

    S1SS = 1.

    printf("ReadID : ");
    printf("%02bx", id[0]);
    printf("%02bx", id[1]);
    printf("%02bx\r\n", id[2]);
}

void flash_read_data(DWORD addr, WORD size, BYTE xdata *pdat)
{
    WORD sz.
    BYTE *ptr.

    while (flash_is_busy());

    S1SS = 0.

    usart1_spi_shift(0x03);                 //Send the command to
    usart1_spi_shift((BYTE)(addr >> 16));   read FLASH data.
    usart1_spi_shift((BYTE)(addr >> 8));
    usart1_spi_shift((BYTE)(addr)).

                                            //Set the destination
                                            address

//  sz = size;
//  ptr = pdat;
//  while (sz--)
//      *ptr++ = usart1_spi_shift(0);         //Register mode read data

    usart1_rx_dma(size, pdat).              //Read data by /DMA
                                            method
```

```
    S1SS = 1.

    printf("ReadData : ");
```

```
    sz = size.
        ptr= pdat;
    for ( sz=0; sz<size; sz++)
    {
        printf("%02bx ", *ptr++);                    //Send the read data to the serial port for debugging.
        if ((sz% 16) == 15)
        {
            printf("\r\n                    ");
        }
    }
    printf("\r\n").
}

void flash_write_enable()
{
    while (flash_is_busy());

    S1SS = 0.

    usart1_spi_shift(0x06).                          //Send write enable command

    S1SS = 1.
}

void flash_write_data(DWORD addr, WORD size, BYTE xdata *pdat)
{
    WORD sz.

    sz = size;
    while (sz)
    {
        flash_write_enable();

        S1SS = 0;

        usart1_spi_shift(0x02).                      //Send write data command
        usart1_spi_shift((BYTE)(addr >> 16));
        usart1_spi_shift((BYTE)(addr >> 8));
        usart1_spi_shift((BYTE)(addr)).

//          do
//          {
//              usart1_spi_shift(*pdat++).           //Write data in register mode
//              addr++;
//
            //if ((BYTE)(addr) == 0x00)
//                  break;
        //} while   (--sz);

        usart1_tx_dma(sz, pdat);                     //DMA write data (note: data must be in one page) sz = 0; DMA write data (note: data must be in one page) sz = 0; DMA write data (note: data must be in one page)

        S1SS = 1.
    }

    printf("Program ! \r\n").
}
```

void flash_erase_sector(DWORD addr)

```
{
    flash_write_enable().

    S1SS = 0.
    usart1_spi_shift(0x20);                                  //send erase
    command usart1_spi_shift((BYTE)(addr >> 16));
    usart1_spi_shift((BYTE)(addr >>
    8)); usart1_spi_shift((BYTE)(addr));
    S1SS = 1;

    printf("Erase Sector ! \r\n").
}

void usart1_tx_dma(WORD size, BYTE xdata *pdat)
{
    size--;                                                  //DMA transfer bytes less than actual 1

    DMA_UR1T_CFG = 0x00; //Close DMA interrupt.              //Disable DMA interrupt
    DMA_UR1T_STA = 0x00; //Clear DMA status.                 //clear DMA status
    DMA_UR1T_AMT = size; //Set the number of DMA transfer bytes.    //Set the number of DMA transfer bytes
    DMA_UR1T_AMTH = size >> 8;
    DMA_UR1T_TXAL = (BYTE)pdat;                              //set buffer address (note: buffer must be
    xdata type) DMA_UR1T_TXAH = (WORD)pdat >> 8;
    DMA_UR1T_CR = 0xc0.                                      //Enable DMA, trigger serial port 1 to send data.

    while (! (DMA_UR1T_STA & 0x01));                         //Wait for DMA data transfer to complete
    DMA_UR1T_STA = 0x00; //Clear DMA status.                 //clear DMA status
    DMA_UR1T_CR = 0x00;                                      //Disable DMA
}

void usart1_rx_dma(WORD size, BYTE xdata *pdat)
{
    size--;                                                  //DMA transfer bytes less than actual 1

    DMA_UR1R_CFG = 0x00; //Close DMA interrupt.              //Disable DMA interrupt
    DMA_UR1R_STA = 0x00; //Clear DMA status.                 //clear DMA status
    DMA_UR1R_AMT = size; //Set the number of DMA transfer bytes.    //Set the number of DMA transfer bytes
    DMA_UR1R_AMTH = size >> 8;
    DMA_UR1R_RXAL = (BYTE)pdat;                              //set buffer address (note: buffer must be
    xdata type) DMA_UR1R_RXAH = (WORD)pdat >> 8;
    DMA_UR1R_CR = 0xa1; //Enable DMA, clear the receive FIFO, trigger serial port 1 to receive data.    //Enable DMA,
    clear receive FIFO, trigger serial port 1 to receive data.

                                                             //!!!!!!!!!!!!!!
usart1_tx_dma(size+1, pdat);                                 //Note: The transmit DMA must be activated at the
                                                             same time as the receive data.
                                                             //!!!!!!!!!!!!!!

    while (! (DMA_UR1R_STA & 0x01));                         //Wait for DMA data transfer to complete
    DMA_UR1R_STA = 0x00; //Clear DMA status.                 //clear DMA status
    DMA_UR1R_CR = 0x00;                                      //Disable DMA
}
```

15.5.6 The SPI interfaces of USART1 and USART2 transfer data to

each other (interrupt mode).

```
//Test operating frequency is 24MHz

//#include "stc8h.h"
```

```
#include "stc32g.h"                         // see download software for header files
#include "intrins.h"
#include "stdio.h"

#define     FOSC     24000000UL             // System Operating Frequency

typedef     bit                 bool;
typedef     unsigned char       byte;
typedef     unsigned int        word.
typedef     unsigned long       DWORD;

sbit S1SS       =   P2^2.
sbit S1MOSI     =   P2^3.
sbit S1MISO     =   P2^4.
sbit S1SCLK     =   P2^5.

sbit S2SS       =   P2^2.
sbit S2MOSI     =   P2^3.
sbit S2MISO     =   P2^4.
sbit S2SCLK     =   P2^5.

void sys_init();
void usart1_spi_init();
void usart2_spi_init();
void test().

BYTE xdata buffer1[256].                    //define the buffer
BYTE xdata buffer2[256].                    //define the buffer
BYTE rptr.
BYTE wptr;
bit     wptr;
bit

void main()
{
    int     i;

    sys_init().                             //System initialisation
    usart1_spi_init().                      //USART1 enable SPI master mode initialisation.
    usart2_spi_init().                      //USART2 enable SPI slave mode initialisation.
    EA = 1;

    for ( i=0; i<128; i++)
    {
        buffer1[i] = i;                     //initialise the buffer
        buffer2[i] = 0;
    }
    test();

    while (1);
}

void uart1_isr() interrupt UART1_VECTOR
{
    if (TI)
    {
        TI = 0;

        if (rptr < 128)
        {
```

```
                SBUF = buffer1[rptr++];
            }
            else
            {
                over = 1;
            }
        }

        if (RI)
        {
            RI = 0;
        }
}

void uart2_isr() interrupt UART2_VECTOR
{
        if (S2TI)
        {
            S2TI = 0.
        }

        if (S2RI)
        {
            S2RI = 0;
            buffer2[wptr++] = S2BUF.
        }
}

void sys_init()
{
        wtst = 0x00;
        ckcon = 0x00;
        eaxfr = 1.

        p0m0 = 0x00; p0m1 = 0x00;
        p1m0 = 0x00; p1m1 = 0x00;
        p2m0 = 0x00; p2m1 = 0x00;
        p3m0 = 0x00; p3m1 = 0x00;
        p4m0 = 0x00; p4m1 = 0x00;
        p5m0 = 0x00; p5m1 = 0x00;
        p6m0 = 0x00. P6M1 = 0x00;
        P7M0 = 0x00; P7M1 = 0x00.

        P40 = 0;
        P6 = 0xff.
}

void usart1_spi_init()
{
        S1SPI_S0 = 1;                       //Switch S1SPI to
                                            //P2.2/S1SS,P2.3/S1MOSI,P2.4/S1MISO,P2.5/S1SCLK

        s1spi_s1 = 0;
        scon = 0x10.                        // Enable receive, must be set to serial mode 0

        USARTCR1 = 0x10;                    //Enable SPI mode on USART1.
//      USARTCR1 |= 0x40;                   //DORD=1
        USARTCR1 &=                         //DORD=0
//      ~0x40; USARTCR1 |=                  //Slave Mode
        0x04; USARTCR1 &=                   //Host mode
        ~0x04.
```

```
        USARTCR1 |= 0x00.                       //CPOL=0, CPHA=0
        //USARTCR1 |= 0x01;                     //CPOL=0, CPHA=1
        //USARTCR1 |= 0x02 ;                    //CPOL=1, CPHA=0
//      USARTCR1 |= 0x03;                       //CPOL=1,
        CPHA=1USARTCR4 = 0x00;                  //SPI speed is
        SYSCLK/4
        //USARTCR4 = 0x01;                      //SPI speed is SYSCLK/8
        //USARTCR4 = 0x02;                      //SPI speed is SYSCLK/16
        //USARTCR4 = 0x03;                      //SPI speed is SYSCLK/2
        USARTCR1 |= 0x08; //Enable SPI function.  //Enable SPI function

        ES = 1;
}

void usart2_spi_init()
{
        S2SPI_S0 = 1; //Switch S2SPI to S2SPI.  //Switch S2SPI to
                                                //P2.2/S2SS,P2.3/S2MOSI,P2.4/S2MISO,P2.5/S2SCLK
        S2SPI_S1 = 0;
        S2CON = 0x10.                           // Enable receive, must be set to serial mode 0

        USART2CR1 = 0x10.                       //Enable SPI mode of USART2.
        //USART2CR1 |= 0x40.                    //DORD=1
        usart2cr1 &= ~0x40.                     //DORD=0
  USART2CR1 |= 0x04.                          //Slave mode
        //USART2CR1 &= ~0x04.                   //Host mode
        USART2CR1 |= 0x00; //CPOL=0, CPHA=0.   //CPOL=0, CPHA=0
        //USART2CR1 |= 0x01;                    //CPOL=0, CPHA=1
        //USART2CR1 |= 0x02 ;                   //CPOL=1, CPHA=0
//      USART2CR1 |= 0x03;                      //CPOL=1,
        CPHA=1USART2CR4 = 0x00;                 //SPI speed is
        SYSCLK/4
        //USART2CR4 = 0x01;                     //SPI speed is SYSCLK/8
        //USART2CR4 = 0x02;                     //SPI speed is SYSCLK/16
        //USART2CR4 = 0x03;                     //SPI speed is SYSCLK/2
        USART2CR1 |= 0x08; //Enable SPI function.  //Enable SPI function

        ES2 = 1;
}

void test()
{
        BYTE i.
        BYTE ret.

        wptr = 0;
        rptr = 0;
        over = 0;

        S1SS = 0.
        SBUF = buffer1[rptr++];                 //initiate data transfer
        while (!over);                          //Wait for 128 data transfers to complete
        S1SS = 1.

                ret= 0x5a.
        for ( i=0; i<128; i++)
        {
                if (buffer1[i] ! = buffer2[i])  //check data
                {
                        ret=
                    0xfe; break;
```

```
            }
        }
        P6 = ret.                                   //P6=0x5a indicates correct data transfer
}
```

16 Asynchronous serial communications (UART3, UART4)

The STC32G series microcontrollers have two full-duplex asynchronous serial communication interfaces (UART3 and UART4). Each serial port consists of two data buffers, a shift register, a serial control register and a baud rate generator. The data buffer of each serial port consists of two independent receive and transmit buffers, which can transmit and receive data at the same time.

Serial port 3/Serial port 4 of STC32G series microcontrollers have two working modes, the baud rate of both modes is variable. Users can set different baud rates and select different working modes by software. The host can process the receiving/transmitting procedures by querying or interrupting, which makes the use very flexible.

The communication ports of Serial Port 3 and Serial Port 4 can be switched to multiple ports by the switching function of the function pins, so that it is possible to time-multiplex a communication port into multiple communication ports.

16.1 Serial Port Function Pin Switching

notation	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**

S4_S: Serial port 4 function pin select bit

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

S3_S: Serial port 3 function pin select bit

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

16.2 Serial port related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
S3CON	Serial Port 3 Control Register	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	Serial Port 3 Data Register	ADH									0000,0000
S4CON	Serial Port 4 Control Register	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	Serial Port 4 Data Register	FEH									0000,0000

16.3 Serial Port 3 (Asynchronous Serial Port UART3)

16.3.1 Serial Port 3 Control Register (S3CON)

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

S3SM0: Specifies the communication operating mode of serial port 3, as shown in the following table:

S3SM0	Serial port 3 operating mode	Functional Description
0	Mode 0	Variable baud rate 8-bit data mode
1	Model 1	Variable baud rate 9-bit data mode

S3ST3: Select baud rate generator for serial port 3

0: Select Timer 2 as the baud rate generator for serial port 3

1: Select Timer 3 as the baud rate generator for serial port 3

S3SM2: Allows serial port 3 to allow multi-computer communication control bit in mode 1. In mode 1, if the S3SM2 bit is 1 and the S3REN bit is 1, the receiver is in the address frame filtering state. At this time, the received bit 9 (i.e., S3RB8) can be used to screen the address frame: if S3RB8=1, it means that the frame is an address frame, the address information can be entered into the S3BUF, and make S3RI to be 1, and then compare the address number in the interrupt service program; if S3RB8=0, it means that the frame is not an address frame, and it should be thrown away and keep S3RI=0. In mode 1, if S3SM2 bit is 0 and S3REN bit is 1, the receiver is in the address frame filtering state. In mode 1, if the S3SM2 bit is 0 and the S3REN bit is 1, the receiving transceiver is in the address frame screening disabled state. Regardless of whether S3RB8 is received as a 0 or a 1, the received message can be made to enter the S3BUF with S3RI=1, at which point S3RB8 is normally the checksum bit. Mode 0 is a non-multiple-computer communication mode, in which S3SM2 should be set to zero.

S3REN: Allow/Disallow serial port receive control bit 0:
Disable serial port receive data

1: Allow serial port to receive data

S3TB8: When Serial 3 is used in mode 1, S3TB8 is the 9th bit of data to be sent, generally used as a parity bit or address frame/data frame flag bit, which can be set or cleared by software as needed. in mode 0, this bit is not used.

S3RB8: When serial port 3 uses mode 1, S3RB8 is the 9th bit of data received, usually used as a parity bit or address frame/data frame marker.

Bit. This bit is not used in mode 0.

S3TI: Serial 3 transmit interrupt request flag bit. S3TI is automatically set to 1 by the hardware when the stop bit starts to be sent, so that an interrupt request is sent to the CPU, and S3TI must be cleared by the software after responding to the interrupt.

S3RI: Serial port 3 receive interrupt request flag bit. S3RI is automatically set to 1 by the hardware in the middle of the stop bit of serial reception, and sends the interrupt request to the CPU, after responding to the interrupt, S3RI must be cleared by the

software.

16.3.2 Serial Port 3 Data Register (S3BUF)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

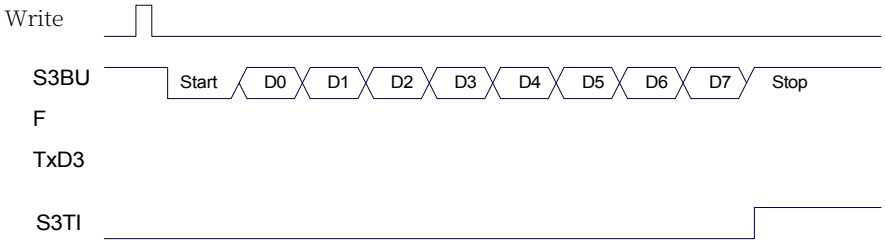
S3BUF: serial port 1 data receive/send buffer, S3BUF is actually two buffers, read buffer and write buffer, two operations correspond to two different registers, one is write-only register (write buffer), and one is read-only register (read buffer). A read operation to S3BUF actually reads the serial port receive buffer, while a write operation to S3BUF triggers the serial port to start sending data.

16.3.3 Serial Port 3 Mode 0, Mode 0 Baud Rate Calculation Formula

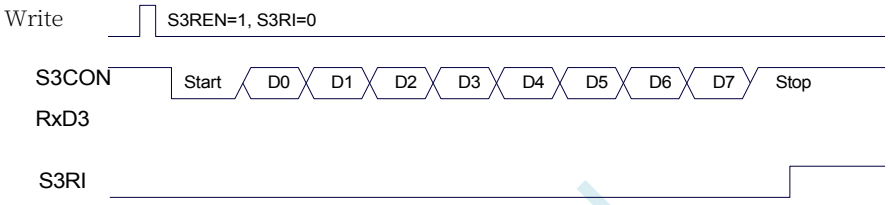
Mode 0 of serial port 3 is an 8-bit data bit variable baud rate UART mode of operation. This mode consists of 10 bits of information in a frame: 1 start bit, 2 start bits, 2 start bits, 1 start bit, and 1 start bit.

8 data bits (low bit first) and 1 stop bit. The baud rate is variable and can be set according to your needs.TxD3 is the data transmission port.

RxD3 is the data receive port, serial port full duplex receive/transmit.



Send data (serial port 3 mode 0)



Receive data (serial port 3 mode 0)

The baud rate of serial port 3 is variable and can be generated by either Timer 2 or Timer 3.

When the timer is in 1T mode (12), the baud rate is variable.

The speed of the baud rate is increased by a factor of 12.

The baud rate calculation formula for Serial 3 Mode 0 is shown in the following table: (SYSclk is the system operating frequency)

Select Timing tool	timer speed degree (angles, temperature etc)	Formula for calculating reloading value	baud
	1T	Timer 2 reload value = $65536 - \frac{SYSclk}{4 \times Baud\ Rate}$	Baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$

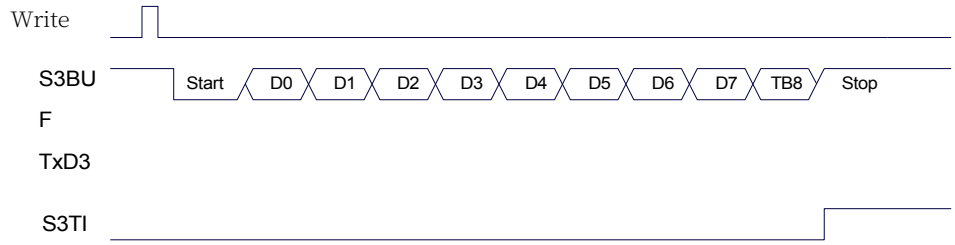
Timer 2	12T	Timer 2 reload value = 65536 - $\frac{SYSclk}{12 \times 4 \times \text{baud rate}}$	Baud rate = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{number of timer reloads})}$
Timer 3	1T	Timer 3 reload value = 65536 - $\frac{SYSclk}{4 \times \text{Baud Rate}}$	Baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$
	12T	Timer 3 reload value = 65536 - $\frac{SYSclk}{12 \times 4 \times \text{baud rate}}$	Baud rate = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{number of timer reloads})}$

16.3.4 Serial Port 3 Mode 1, Mode 1 Baud Rate Calculation Formula

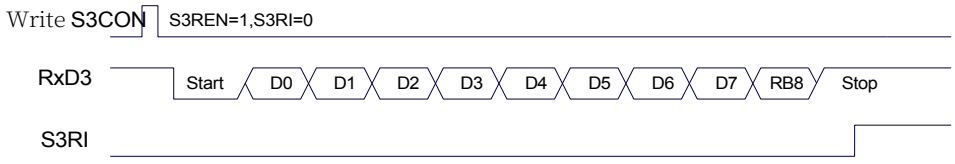
Mode 1 of Serial Port 3 is a 9-bit data bit variable baud rate UART mode of operation. This mode consists of 11 bits of information in a frame: 1 start bit, 2 start bits, 2 start bits, 1 start bit, and 1 start bit.

9 data bits (low first) and 1 stop bit. The baud rate is variable and can be set according to your needs.TxD3 is the data transmission port.

RxD3 is the data receive port, serial port full duplex receive/transmit.



Send data (serial port 3 mode 1)



Receive data (serial port 3 mode 1)

The baud rate calculation formula for Serial Port 3 Mode 1 is identical to Mode 0. Please refer to the baud rate calculation formula for Mode 0.

16.4 Serial Port 4 (Asynchronous Serial Port UART4)

16.4.1 Serial 4 Control Register (S4CON)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0: Specifies the communication operating mode of serial port 4, as shown in the following table:

S4SM0	Serial Port 4 Operating Mode	Functional Description
0	Mode 0	Variable baud rate 8-bit data mode
1	Model 1	Variable baud rate 9-bit data mode

S4ST4: Select baud rate generator for serial port 4

0: Select Timer 2 as the baud rate generator for serial port 4

1: Select Timer 4 as the baud rate generator for serial port 4

S4SM2: Allows serial port 4 to allow multi-computer communication control bit in mode 1. In mode 1, if the S4SM2 bit is 1 and the S4REN bit is 1, the receiver is in the address frame screening state. At this time, the received bit 9 (i.e. S4RB8) can be used to screen the address frame: if S4RB8=1, it means that the frame is an address frame, the address information can be entered into the S4BUF, and S4RI is 1, and then the address number can be compared in the interrupt service program; if S4RB8=0, it means that the frame is not an address frame, and it should be thrown away and keep S4RI=0. In mode 1, if S4SM2 bit is 0 and S4REN bit is 1, the receiver is in the address frame screening state. In mode 1, if the S4SM2 bit is 0 and the S4REN bit is 1, the receiving transceiver is in the address frame screening disabled state. Whether the received S4RB8 is 0 or 1, the received message can be put into S4BUF and S4RI=1, when S4RB8 is usually the parity bit. Mode 0 is a non-multi-computer communication mode, in which S4SM2 should be set to zero.

S4REN: Allow/Disallow serial port receive control bit 0:
Disable serial port receive data

1: Allow serial port to receive data

S4TB8: When Serial 4 is used in mode 1, S4TB8 is the 9th bit of data to be sent, generally used as a parity bit or address frame/data frame flag bit, which can be set or cleared by software as needed. in mode 0, this bit is not used.

S4RB8: When serial port 4 is used in mode 1, S4RB8 is the 9th bit of data received, which is usually used as the parity bit or address frame/data frame marker.

Bit. This bit is not used in mode 0.

S4TI: Serial 4 transmit interrupt request flag bit. S4TI is automatically set to 1 by hardware to request interrupt from CPU when the stop bit starts to be sent, and S4TI must be cleared by software after responding to the interrupt.

S4RI: Serial 4 receive interrupt request flag bit. S4RI is automatically set to 1 by the hardware in the middle of receiving stop bit, and send interrupt request to CPU, and S4RI must be

cleared by the software after responding to the interrupt.

16.4.2 Serial 4 Data Register (S4BUF)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	FEH								

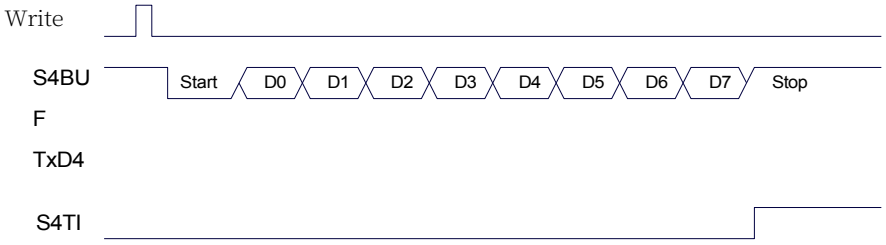
S4BUF: serial port 1 data receive/send buffer, S4BUF is actually two buffers, read buffer and write buffer, two operations correspond to two different registers, one is write-only register (write buffer), and one is read-only register (read buffer). Read operation to S4BUF actually reads the serial port receive buffer, while write operation to S4BUF triggers the serial port to start sending data.

16.4.3 Serial Port 4 Mode 0, Mode 0 Baud Rate Calculation Formula

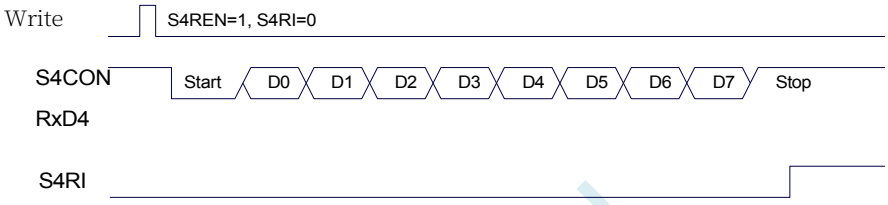
Mode 0 of serial port 4 is an 8-bit data bit variable baud rate UART mode of operation. This mode consists of 10 bits of information in a frame: 1 start bit, 2 start bits, 2 start bits, 1 start bit, and 1 start bit.

8 data bits (low first) and 1 stop bit. The baud rate is variable and can be set according to your needs.TxD4 is the data transmission port.

RxD4 is the data receive port, serial port full duplex receive/transmit.



Send data (serial port 4 mode 0)



Receive data (serial port 4 mode 0)

The baud rate of serial port 4 is variable and can be generated by either Timer 2 or Timer 4. When the timer is in 1T mode (12), the baud rate is variable.

The speed of the baud rate is increased by a factor of 12.

The formula for calculating the baud rate for serial port 4 mode 0 is shown in the following table: (SYSclk is the system operating frequency)

Select Timing tool	timer speed degree (angles, temperature etc)	Formula for calculating reloading value	baud
	1T	Timer 2 reload value = 65536 - $\frac{SYSclk}{4 \times Baud\ Rate}$	Baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$

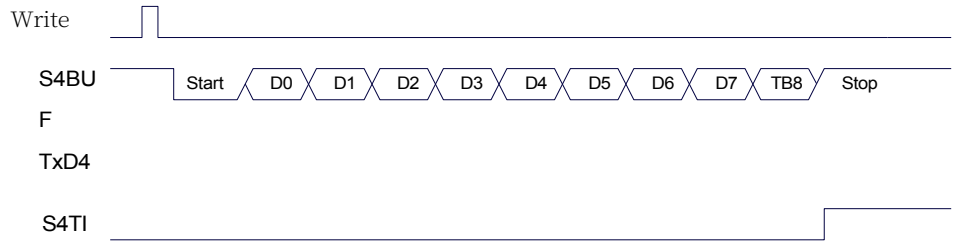
Timer 2	12T	Timer 2 reload value = 65536 - $\frac{SYSclk}{12 \times 4 \times \text{baud rate}}$	Baud rate = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{number of timer reloads})}$
Timer 4	1T	Timer 4 reload value = 65536 - $\frac{SYSclk}{4 \times \text{Baud Rate}}$	Baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$
	12T	Timer 4 reload value = 65536 - $\frac{SYSclk}{12 \times 4 \times \text{baud rate}}$	Baud rate = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{number of timer reloads})}$

16.4.4 Serial Port 4 Mode 1, Mode 1 Baud Rate Calculation Formula

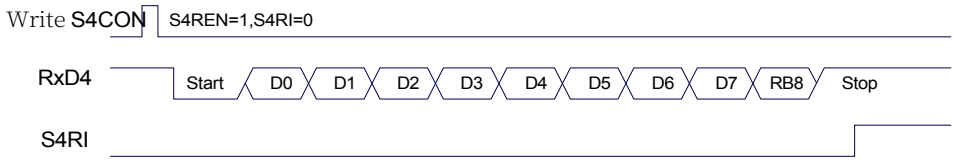
Mode 1 of serial port 4 is a 9-bit data bit variable baud rate UART mode of operation. This mode consists of 11 bits of information in a frame: 1 start bit, 2 start bits, 2 start bits, 1 start bit, and 1 start bit.

TxD4 is the data transmission port. 9 data bits (low bit first) and 1 stop bit. The baud rate is variable and can be set according to your needs.TxD4 is the data transmission port.

RxD4 is the data receive port, serial port full duplex receive/transmit.



Send Data (Serial 4 Mode 1)



Receive data (Serial 4 mode 1)

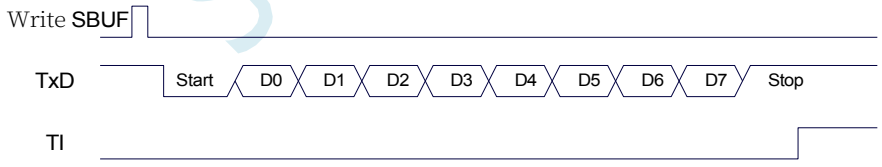
The baud rate calculation formula for Serial 4 Mode 1 is identical to Mode 0. Please refer to the baud rate calculation formula for Mode 0.

16.5 Serial Port Considerations

The following issues should be noted about the serial port interrupt request: (serial port 1, serial port 2, serial port 3, serial port 4 are similar, the following serial port 1)

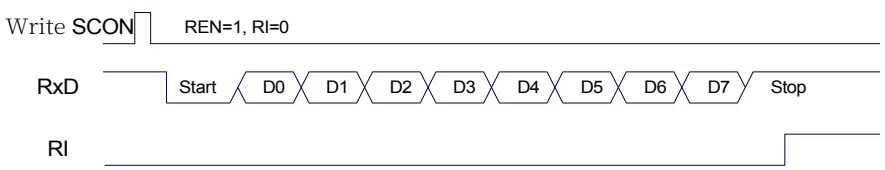
(as an example for illustration)

In 8-bit data mode, a TI interrupt request is generated after the entire stop bit is sent, as shown below:



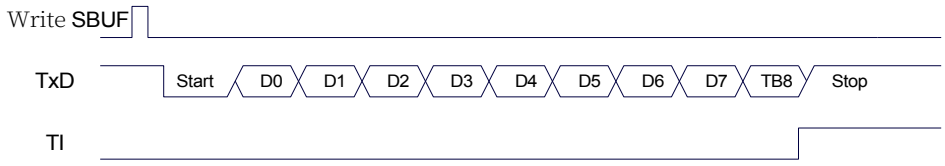
Send data (8-bit data)

In 8-bit data mode, an RI interrupt request is generated after half of the stop bits are received, as shown below:



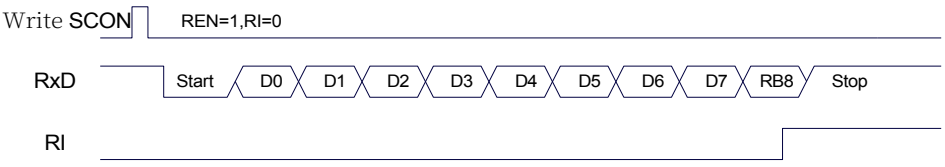
Receive data (8-bit data)

In 9-bit data mode, a TI interrupt request is generated after the entire 9th data bit is sent, as shown below:



Send data (9-bit data)

In 9-bit data mode, an RI interrupt request is generated after half of the 9th data bit is received, as shown below:



Receive data (9-bit data)

16.6 sample procedure

16.6.1 Serial port 3 uses timer 2 as a baud rate generator.

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - (FOSC / 115200+2) / 4)

                                        //Define as unsigned long integer to avoid overflow.
                                        The // plus 2 operation is to allow the Keil compiler to
                                        //Automatic implementation of rounding operations

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

void Uart3Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;
        busy = 0;
    }
    if (S3RI)
    {
        S3RI = 0;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f.
    }
}

void Uart3Init()
{
    s3con = 0x10;
    t2l = brt.
    T2H = BRT >> 8;
    T2x12 = 1.
    T2R = 1.
    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()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    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();
    ES3 = 1;
    EA = 1;
    Uart3SendStr("Uart Test ! \r\n");

    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f.
        }
    }
}
```


16.6.2 Serial Port 3 Use Timer 3 as a Baud Rate Generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - (FOSC / 115200+2) / 4)
                                    //Define as unsigned long integer to avoid overflow.
                                    The // plus 2 operation is to allow the Keil compiler to
                                    //Automatic implementation of rounding operations
```

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

void Uart3Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0;
        busy = 0;
    }
    if (S3RI)
    {
        S3RI = 0;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f.
    }
}

void Uart3Init()
{
    s3con = 0x50;
    t3l = brt.
    T3H = BRT >> 8;
    T3x12 = 1.
    T3R = 1.
    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()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    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();
    ES3 = 1;
    EA = 1;
    Uart3SendStr("Uart Test ! \r\n");

    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f.
        }
    }
}
```

16.6.3 Serial Port 4 Using Timer 2 as a Baud Rate Generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - (FOSC / 115200+2) / 4)
                                        //Define as unsigned long integer to avoid overflow.
                                        The // plus 2 operation is to allow the Keil compiler to
                                        //Automatic implementation of rounding operations

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

void Uart4Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0.
        busy = 0;
    }
    if (S4RI)
    {
        S4RI = 0.
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f.
    }
}
```

```
void Uart4Init()
{
    s4con = 0x10;
    t2l = brt.
    T2H = BRT >> 8;
    T2x12 = 1.
    T2R = 1.
    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()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;   //Set the external data bus speed to fastest
    WTST = 0x00;    //set the program code wait parameter.
                    //Assign a value of 0 to set the CPU to execute the
                    programme as fast as possible.

    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();
    ES4 = 1;
    EA = 1;
    Uart4SendStr("Uart Test ! \r\n");

    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f.
```

```
            }
        }
}
```

16.6.4 Serial Port 4 Using Timer 4 as a Baud Rate Generator

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                 // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL              //Define as unsigned long integer to avoid overflow.
#define     BRT             (65536 - (FOSC / 115200+2) / 4)

                                                    The // plus 2 operation is to allow the Keil compiler to
                                                    //Automatic implementation of rounding operations

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

void Uart4Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0.
        busy = 0;
    }
    if (S4RI)
    {
        S4RI = 0.
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f.
    }
}

void Uart4Init()
{
    s4con = 0x50;
    t4l = brt.
    t4h = brt >> 8;
    t4x12 = 1.
    T4R = 1.
    wptr = 0x00;
    rptr  = 0x00;
    busy = 0;
}

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

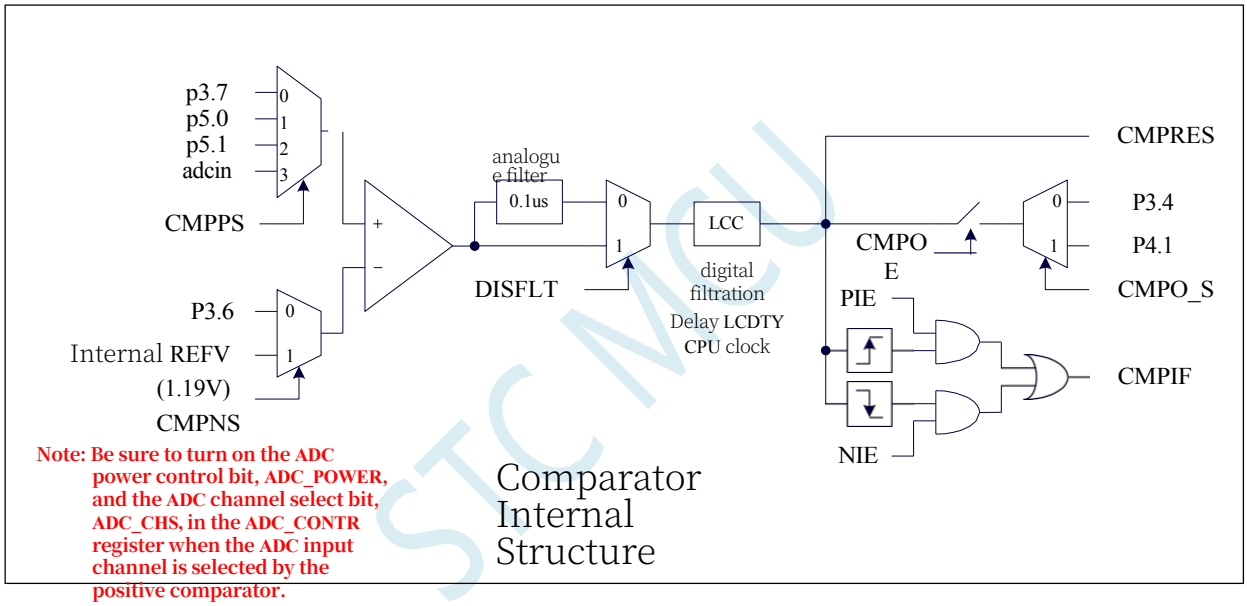
```
void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

void main()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                                   //Set the external data bus speed to fastest
    WTST = 0x00;                                    //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.

    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();
    ES4 = 1;
    EA = 1;
    Uart4SendStr("Uart Test ! \r\n");

    while (1)
    {
        if (rptr ! = wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f.
        }
    }
}
```

17 Comparator, Brown-out Detection, Internal Fixed Comparison Voltage

The STC32G series microcontrollers have an integrated comparator. The positive terminal of the comparator can be the P3.7 port, P5.0 port, P5.1 port, or the analogue input channel of the ADC, while the negative terminal can be the P3.6 port or the internal BandGap after OP REFV voltage (internal fixed comparison voltage). Multiple comparator applications can be realised by means of multiplexers and time division multiplexing.

The comparator has two levels of programmable internal filtering: analogue filtering and digital filtering. The analogue filtering filters out burrs in the comparison input signals, while the digital filtering waits for the input signals to become more stable before making a comparison. Comparison results can be obtained directly by reading internal register bits, or the comparator results can be output to an external port in either forward or reverse direction. Outputting the comparison result to an external port can be used as a trigger signal for external events and feedback signals, which can expand the range of comparison applications.

17.1 比较器内部结构图



Note: Be sure to turn on the ADC power control bit, ADC_POWER, and the ADC channel select bit, ADC_CHS, in the ADC_CONTR register when the ADC input channel is selected by the positive comparator.

17.2 Comparator Function Pin Switching

notation	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

17.3 Comparator-related registers

notation	descriptions	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	7EFEAEH	CHYS[1:0]		-	-	-	CMPNS	CMPPS[1:0]		00xx,x000

17.3.1 Comparator Control Register 1 (CMPCR1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES

CMPEN: Comparator
Module Enable Bit 0:
Disable Compare
Function
1: Enable comparison function

CMPIF: Comparator interrupt flag bit. When PIE or NIE is enabled, if corresponding interrupt signal is generated, the hardware will set CMPIF to 1 automatically and request interrupt to CPU. This flag bit must be cleared by user software.

PIE: Comparator rising edge
interrupt enable bit. 0:
Comparator rising edge
interrupt disabled.
1: Enable comparator rising edge interrupt. An interrupt request is generated when the comparison result of the enabled comparator changes from 0 to 1.

NIE: Comparator falling edge
interrupt enable bit. 0:
Comparator falling edge
interrupt disabled.
1: Enable comparator falling edge interrupt. An interrupt request is generated when the comparison result of the enable comparator changes from 1 to 0.

CMPOE: Comparator result
output control bit 0:
Comparator result output
disabled
1: Enable comparator result output. Comparator result output to P3.4 or P4.1 (set by CMPO_S in P_SW2)

CMPRES: Comparison result of the
comparator. This bit is read-only. 0:
Indicates that the CMP+ level is lower
than the CMP- level
1: Indicates that the level of CMP+ is higher than the level of CMP-.
CMPRES is the digitally filtered output signal, not the direct output result of the comparator.

17.3.2 Comparator Control Register 2 (CMPCR2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO: Comparator result output control

0: Comparator result positive output. If CMPRES is 0, P3.4/P4.1 outputs low, and vice versa outputs high.

1: Comparator result inverted output. If CMPRES is 0, P3.4/P4.1 outputs high and vice versa.

DISFLT: Analogue filter function control 0:

Enable 0.1us analogue filter function

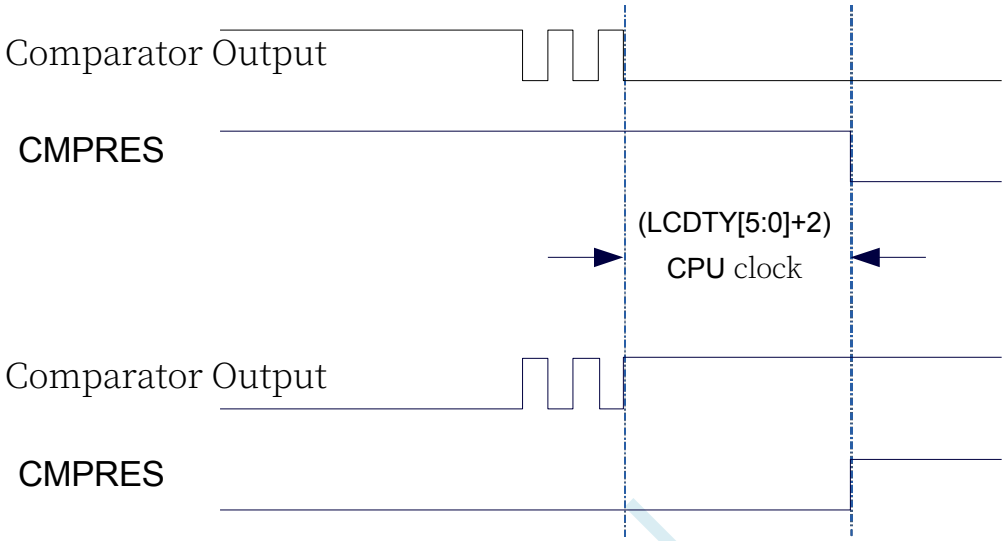
1: Turning off the 0.1us analogue filtering slightly increases the comparison speed of the comparator.

LCDTY[5:0]: Digital filter function control

The digital filter function is the digital signal de-jitter function. When the comparison result changes on the rising or falling edge, the comparator detects that the changed signal must maintain the number of CPU clocks set by LCDTY unchanged before the data change is considered valid; otherwise, the data change is considered valid.

Then it will be treated as if there is no change in the signal.

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



17.3.3 Comparator Extended Configuration Register (CMPEXCFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CMPEXCFG	7EFEAEH	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: Select the internal BandGap after OP voltage REFV as the comparator negative input source (the chip is shipped with the internal reference voltage adjusted to **1.19V**)

CMPPS[1:0]: Comparator positive input selection

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

(Note 1: Be sure to turn on the ADC power control bit ADC_POWER and the ADC channel select bit ADC_CHS in the ADC_CONTR register when the ADC input

channel is selected by the positive side of the comparator.)

17.4 sample procedure

17.4.1 Use of comparators (interrupt mode)

//Tested operating frequency is *11.0592MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void CMP_Isr() interrupt 21
{
    CMPIF = 0;                               // clear interrupt flag
    if (CMPRES)
    {
        P10 = !                              //Falling edge interrupt test port
    }
    else
    {
        P11 = !                              //Rising edge interrupt test port
    }
}

void main()
{
    eaxfr = 1; ckcon =                       // Enable access to XFR
    0x00; wtst = 0x00.                       //Set external data bus speed to fastest
                                             //Set the parameter for waiting for the
                                             programme code.
                                             //Assign a value of 0 to set the CPU to
                                             execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

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

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

```
CMPCR1 = 0x00;
```

```
    CMPCR2= 0x00;
    INVCMPO = 0;                    //Comparator forward output
//  INVCMPO = 1;                    //Comparator inverted output
    DISFLT = 0;                     //enable 0.1us filtering
//  DISFLT = 1;                     //Disable 0.1us filtering
//  cmpcr2 &= ~0x3f;                //Comparator result direct output
    cmpcr2 |= 0x10; pie =           //Comparator result output after 16 de-jittered
    nie = 1.                        clocks
//  PIE = 0;                        //Enable comparator edge interrupt
                                    //Disable comparator rising edge interrupt
//  PIE = 1;                        //Enable comparator rising edge interrupt
//  NIE = 0;                        //Disable comparator falling edge interrupt
//  NIE = 1;                        //Enable comparator falling edge interrupt
//  CMPOE = 0;                      //Disable comparator output
    CMPOE = 1;                      //Enable comparator output
    CMPEN = 1;                      //Enable Comparator Module

    EA = 1;

    while (1);
}
```

17.4.2 Use of comparators (query method)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 //see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign 0 to set the CPU to execute the programme as
                                    fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    CMPEXCFG = 0x00;
    //CMPEXCFG |= 0x40;             //CMPEXCFG |= 0x40;   //CMPEXCFG |= 0x40.      //
Comparator DC hysteresis input selection
                                    //0:0mV; 0x40:10mV; 0x80:20mV; 0xc0:30mV
```

```
CMPEXCFG &= ~0x03; //P3.7 is the CMP+ input pin.          //P3.7 is CMP+ input pin
    //CMPEXCFG |= 0x01;                                   //P5.0 is CMP+ input
                                                          pin
```

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

        cmpcr1  =  0x00;
        cmpcr2=    0x00;
        invmpcr = 0.                        //Comparator forward output
//      INVCMPO = 1;                        //Comparator inverted output
        DISFLT = 0;                         //Enable 0.1us filtering
//      DISFLT = 1;                         //Disable 0.1us filtering
//      cmpcr2 &= ~0x3f;                    //Comparator result direct output
        cmpcr2 |= 0x10; pie =               //Comparator result output after 16
        nie = 1.                            de-jittered clocks
//      PIE = 0;                            //Enable comparator edge interrupt
                                            //Disable comparator rising edge
                                            interrupt
//      PIE = 1;                            //Enable comparator rising edge
                                            interrupt
//      NIE = 0;                            //Disable comparator falling edge
                                            interrupt
//      NIE = 1;                            //Enable comparator falling edge
                                            interrupt
//      CMPOE = 0;                          //Disable comparator output
        CMPOE = 1;                          //Enable comparator output
        CMPEN = 1;                          //Enable Comparator Module

        while (1)
        {
            P10 = CMPRES.                   //Read the comparator comparison
        }                                   result

}
```

17.4.3 Comparator multiplexing applications (Comparator + ADC input channels)

Since the positive pole of the comparator selects the analogue input channel of the ADC, multiple comparators can be implemented through multi selectors and time division multiplexing.

Note: Be sure to turn on the ADC power control bit in the ADC_CONTR register when the ADC input channel is selected by the positive side of the comparator!

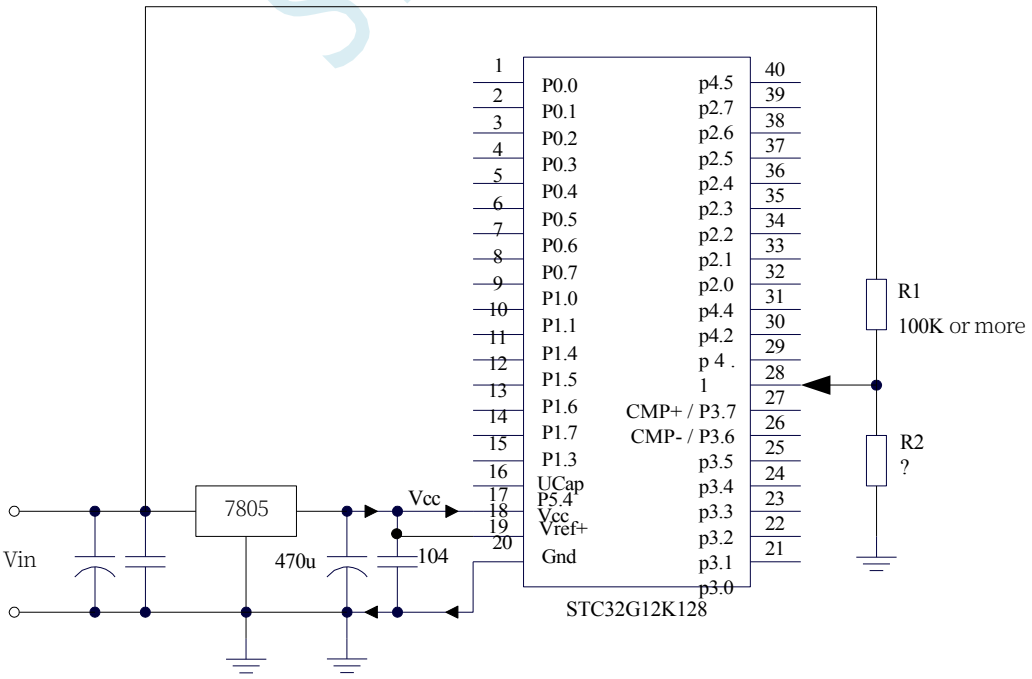
ADC_POWER and ADC channel select bits ADC_CHS

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         //see download software for header files
#include "intrins.h"

void main()
{
        EAXFR = 1;                          //Enable access to XFR
        CKCON = 0x00;                       //Set the external data bus speed to fastest
        WTST = 0x00;                        //set the program code wait parameter.
```

```
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00.
```

```
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    p1m0 &= 0xfe;                   //Set P1.0 as the input port.
    p1m1 |= 0x01.
    ADC_CONTR = 0x80.               //Enable ADC module and select P1.0 as ADC input pin


    CMPEXCFG = 0x00;
//  CMPEXCFG &= ~0x03;              //P3.7 is the CMP+ input pin.
//  CMPEXCFG |= 0x01.               //P5.0 is the CMP+ input pin.
//  CMPEXCFG |= 0x02;               //P5.1 is the CMP+ input pin.
    CMPEXCFG |= 0x03;               //ADC input pin is CMP+ input pin
    CMPEXCFG &= ~0x04;              //P3.6 is the CMP-input pin.
//  CMPEXCFG |= 0x04;               //Internal 1.19V reference voltage for CMP-input pin
    CMPEXCFG |= 0x04.

    cmpcr2  =  0x00;
    cmpcr1  =  0x00;
    cmpoe = 1.                      //Enable comparator output
    CMPEN = 1;                      //Enable Comparator Module

    while (1);
}
```

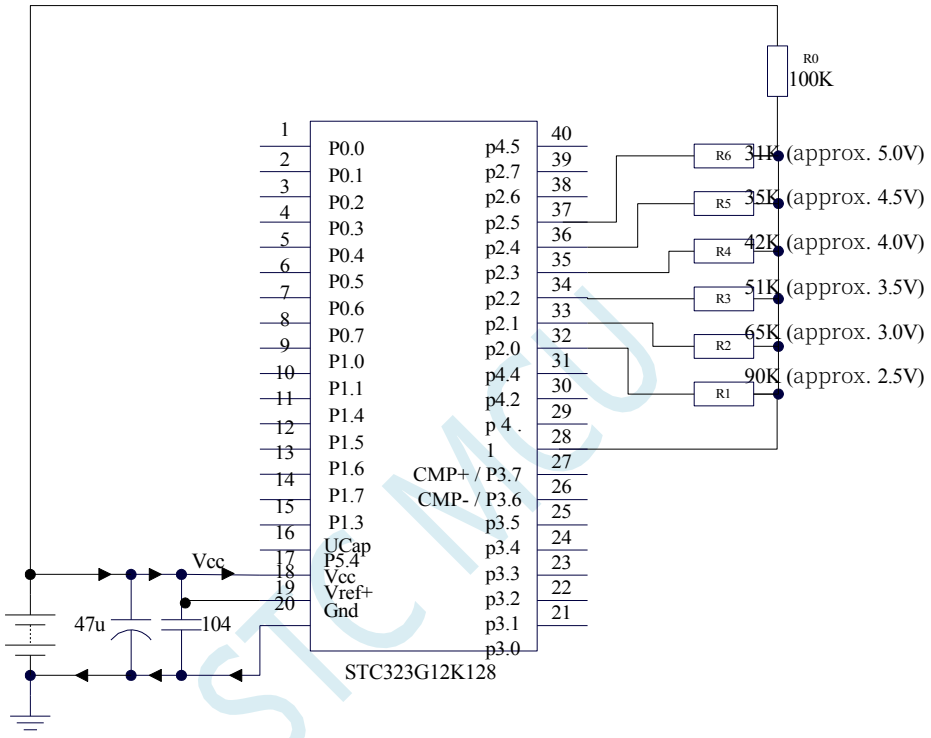
17.4.4 Comparator for external power down detection (user data should be saved in time during power down to the (in EEPROM)



Resistors R1 and R2 in the above diagram divide the voltage at the front of the regulator 7805, and the divided voltage is used as an external input to the comparator CMP+ for comparison with the internal 1.19V reference source.

Generally, when the AC voltage is 220V, the DC voltage at the front of the regulator 7805 is 11V, but when the AC voltage drops to 160V, the DC voltage at the front of the regulator 7805 is 8.5V. When the DC voltage at the front of the 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 to the positive input of the comparator, CMP+, and the input voltage at the CMP+ end is lower than the internal 1.19V reference signal source, so that there is enough time to save the data into the EEPROM when power-down detection occurs. The input voltage at CMP+ is lower than the internal 1.19V reference signal source, and a comparator interrupt can be generated at this time, so that there is sufficient time to save the data to the EEPROM during the power-down detection. When the DC voltage at the front end of the regulator 7805 is higher than 8.5V, the DC voltage at the front end is divided by resistors R1 and R2 to the positive input of the comparator, CMP+, and the input voltage at the CMP+ terminal is higher than the internal reference signal source of 1.19V, at which time the CPU can continue to work normally.

17.4.5 Comparator detects operating voltage (battery voltage)



In the above figure, the working voltage of MCU can be approximated by using the principle of resistor voltage divider (for the selected channel, the IO port of MCU outputs a low level, and the port voltage value is close to Gnd, and for the unselected channel, the IO port of MCU outputs a high in the open-drain mode, which doesn't affect other channels).

The negative end of the comparator selects the internal 1.19V reference signal source, and the positive end selects the voltage value input to the CMP+ pin after dividing by a resistor. During initialisation, ports P2.5 to P2.0 are set to open-drain mode and output high. Firstly, the P2.0 port outputs a low level, at this time, if the Vcc voltage is low, the CMP+ pin will be

set to the open-drain mode.

If the Vcc voltage is higher than 2.5V, the comparator value will be 0. Conversely, if the Vcc voltage is higher than 2.5V, the comparator value will be 1;

If it is determined that Vcc is higher than 2.5V, then P2.0 port outputs high and P2.1 port outputs low, at this time, if the Vcc voltage is lower than 3.0V, then the comparative value of the comparator will be 0, and conversely, if the Vcc voltage is higher than 3.0V, then the comparative value of the comparator will be 1;

If it is determined that Vcc is higher than 3.0V, then P2.1 port outputs high and P2.2 port outputs low, at this time, if the Vcc voltage is lower than 3.5V, then the comparative value of the comparator will be 0, and conversely, if the Vcc voltage is higher than 3.5V, then the comparative value of the comparator will be 1;

If it is determined that Vcc is higher than 3.5V, then P2.2 port outputs high and P2.3 port outputs low, at this time, if the Vcc voltage is lower than 4.0V, then the comparative value of the comparator will be 0, and vice versa, if the Vcc voltage is higher than 4.0V, then the comparative value of the comparator will be 1;

If it is determined that Vcc is higher than 4.0V, then P2.3 port outputs high and P2.4 port outputs low, at this time, if the Vcc voltage is lower than 4.5V, then the comparative value of the comparator will be 0, and vice versa, if the Vcc voltage is higher than 4.5V, then the comparative value of the comparator will be 1;

If it is determined that Vcc is higher than 4.5V, then P2.4 port outputs high and P2.5 port outputs low, at this time, if the Vcc voltage is lower than 5.0V, then the comparative value of the comparator will be 0, and vice versa, if the Vcc voltage is higher than 5.0V, the comparative value of the comparator will be 1.

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                    // see download software for header files
#include "intrins.h"

void delay ()
{
    char i;

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

void main()
{
    unsigned char v.

    eaxfr = 1; ckcon =                 // Enable access to XFR
    0x00; wtst = 0x00.                 //Set external data bus speed to fastest
                                       //Set the parameter for waiting for the
                                       programme code.
                                       //Assign a value of 0 to set the CPU to
                                       execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    p2m0 = 0x3f;                       //P2.5~P2.0 Initialisation to open-drain mode
    p2m1 = 0x3f.
    P2 = 0xff.

    CMPEXCFG = 0x00;
    CMPEXCFG &= ~0x03;                 //P3.7 is the CMP+ input pin.
//  CMPEXCFG |= 0x01.                  //P5.0 is the CMP+ input pin.
//  CMPEXCFG |= 0x02;                  //P5.1 is the CMP+ input pin.
//  CMPEXCFG |= 0x03;                  //ADC input pin is CMP+ input pin
//  CMPEXCFG &= ~0x04;                 //P3.6 is the CMP-input pin.
    CMPEXCFG |= 0x04;                  // Internal 1.19V reference voltage for
                                       CMP-input pin

    cmpcr2 = 0x10;                     //Comparator result output after 16 de-
    cmpcr1 = 0x00;                     jittered clocks
    cmpen = 1.
                                       //Enable Comparator Module

    while (1)
    {
        v = 0x00;                      //Voltage <2.5V
```

```
P2 = 0xfe;                          //P2.0 Output 0
delay();
if (! (CMPRES)) goto ShowVol;
v = 0x01.                           //Voltage > 2.5V
```

```
        P2 = 0xfd.                          //P2.1 output 0
        delay();
        if (! (CMPRES)) goto ShowVol;
        v = 0x03;                           //voltage > 3.0V
        P2 = 0xfb.                          //P2.2 output 0
        delay();
        if (! (CMPRES)) goto ShowVol;
        v = 0x07;                           //voltage > 3.5V
        P2 = 0xf7.                          //P2.3 output 0
        delay();
        if (! (CMPRES)) goto ShowVol;
        v = 0x0f;                           //voltage > 4.0V
        P2 = 0xef.                          //P2.4 output 0
        delay();
        if (! (CMPRES)) goto ShowVol;
        v = 0x1f;                           //voltage > 4.5V
        P2 = 0xdf.                          //P2.5 output 0
        delay();
        if (! (CMPRES)) goto ShowVol;
        v = 0x3f;                           //voltage > 5.0V
ShowVol.
        p2 = 0xff;
        p0 = ~v.
    }
}
```

18 IAP/EEPROM

The STC32G series microcontrollers have an integrated large-capacity EEPROM, and the internal Data Flash can be used as an EEPROM using ISP/IAP technology.

The EEPROM can be divided into several sectors, each containing 512 bytes.

Note: The write operation of EEPROM can only write 1 to 0 in byte, when you need to write 0 to 1 in byte, you have to perform sector erase operation, the read/write operation of EEPROM is performed in unit of 1 byte, while the erase operation of EEPROM is performed in unit of 1 sector (512 bytes), when you perform the erase operation, if there is any data in the target sector that needs to be preserved, you have to read these data into the RAM temporarily in advance. If there is data in the target sector that needs to be preserved during the erase operation, the data must be read into RAM in advance for temporary storage, and then written back to the EEPROM/DATA-FLASH after the erase operation is completed, along with the preserved data and the data that needs to be updated.

Therefore, when using EEPROM, it is recommended that data modified at the same time be placed in the same sector, and data not modified at the same time be placed in a different sector, not necessarily full. Erase operation of data memory is performed sector by sector (512 bytes per sector).

The EEPROM can be used to store parameter data that needs to be modified during the application and will not be lost during power down. Byte read/byte program/sector erase operations can be performed on the EEPROM in the user programme. It is recommended not to operate the EEPROM when the operating voltage is low to avoid sending data loss.

Special Note: For STC32F12K60 series microcontrollers, CACHE cannot be turned on when using the IAP register to operate the EEPROM.
function, otherwise the correct EEPROM may not be accessible.

18.1 EEPROM operation time

- Read 1 byte: 4 system clocks (easier and faster to read with MOVC command)
- Programming 1 byte: approx. 30-40us (actual programming time is 6-7.5us, but state transition times and SETUP and HOLD times for various control signals need to be added)
- Erase 1 sector (512 bytes): Approx. 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 (rounded to the nearest whole number) e.g. system operating frequency is 12MHz, then IAP_TPS is set to 12.

The system operating frequency is 22.1184MHz, then IAP_TPS is set to 22

The system operating frequency is 5.5296MHz, then IAP_TPS is set to 6

18.2 EEPROM-related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP Data Register	C2H	DATA[7:0]								0000,0000
IAP_DATA1	IAP Data Register	D9H	DATA[15:8]								0000,0000
IAP_DATA2	IAP Data Register	DAH	DATA[23:16]								0000,0000
IAP_DATA3	IAP Data Register	DBH	DATA[31:24]								0000,0000
IAP_ADDRE	IAP Extended Address Register	F6H	ADDR[23:16]								1111,1111
IAP_ADDRH	IAP High Address Register	C3H	ADDR[15:8]								0000,0000
IAP_ADDRL	IAP Low Address Register	C4H	ADDR[7:0]								0000,0000

IAP_CMD	IAP Command Register	C5H	-	-	-	-	-	CMD[2:0]			xxxx,x000
IAP_TRIG	IAP Trigger Register	C6H									0000,0000
IAP_CONTR	IAP Control Register	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
IAP_TPS	IAP Wait Time Control Register	F5H	-	-	IAP_TPS[5:0]						xx00,0000

18.2.1 EEPROM Data Register (IAP_DATA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H	DATA[7:0]							
IAP_DATA1	D9H	DATA[15:8]							
IAP_DATA2	DAH	DATA[23:16]							
IAP_DATA3	DBH	DATA[31:24]							

For EEPROM read operation, the EEPROM data read out after the command is executed is stored in the IAP_DATA register. For EEPROM write operation, the data to be written must be stored in the IAP_DATA register before the write command is executed, and then the write command is sent. The Erase EEPROM command has nothing to do with the IAP_DATA register.

Note: STC32G12K128 series and STC32G8K64 series only support single-byte read/write operation, and only IAP_DATA is valid in the data register. STC32F12K60 series support 4-byte read/write operation, and IAP_DATA, IAP_DATA1, IAP_DATA2 and IAP_DATA3 are valid in the data register.

18.2.2 EEPROM Address Register (IAP_ADDR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRE	F6H	ADDR[23:16]							
IAP_ADDRH	C3H	ADDR[15:8]							
IAP_ADDRL	C4H	ADDR[7:0]							

Target address register for EEPROM read, write, and erase operations IAP_ADDRH holds the high byte of the address and IAP_ADDRL holds the low byte of the address

18.2.3 EEPROM Command Register (IAP_CMD)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	CMD[2:0]		

CMD[2:0]: send EEPROM

operation command 000

(CMD0): null operation

001 (CMD1): read EEPROM byte command. Read 1 byte of the target address ADDR[23:0].

010 (CMD2): Write EEPROM byte command. Write 1 byte of the target address ADDR[23:0].

011 (CMD3): Erases the EEPROM sector. Erases 1 sector (512 bytes) at target address ADDR[23:9].

100 (CMD4): Null operation

101 (CMD5): Write EEPROM word (4 bytes) command. Write 4 bytes of the target address ADDR[23:2].

110 (CMD6): Erases half sector (256 bytes) of EEPROM. Erases 0.5 sector (256 bytes) of target

address ADDR[23:8].

111 (CMD7): Erase the EEPROM block. Erases 32 sectors (16K bytes) at target address ADDR[23:15].

Note: The STC32G12K128 series and STC32G8K64 series only support CMD0、CMD1、CMD2、CMD3. STC32F12K60 series supports the full range of commands.

18.2.4 EEPROM trigger register (IAP_TRIG)

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

After setting the command registers, address registers, data registers and control registers for EEPROM read, write and erase, it is necessary to write two trigger commands, 5AH and A5H (the order can not be exchanged), to the trigger register IAP_TRIG to trigger the corresponding read, write and erase operations. After the operation is completed, the contents of the EEPROM address registers IAP_ADDRH, IAP_ADDRL and the EEPROM command register IAP_CMD remain unchanged. If you want to operate the data in the next address, you need to update the values of IAP_ADDRH and IAP_ADDRL manually.

Note: For each EEPROM operation, IAP_TRIG must be written to 5AH first, and then to A5H before the corresponding command will take effect. After writing the trigger command, the CPU will be in IDLE state and wait until the corresponding IAP operation is completed, then the CPU will return from IDLE state to normal state and continue to execute CPU commands.

18.2.5 EEPROM control register (IAP_CONTR)

notation	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, (needs to be used with SWRST) 0: Execute the programme from user code after software reset
1: Execute the programme from the system ISP monitor code area after software reset.

SWRST: Software reset control bit 0: No action
1: Generating a software reset

CMD_FAIL: EEPROM operation failure status bit, need to be cleared by software 0: EEPROM operation correctly
1: EEPROM operation failure

18.2.6 EEPROM Erase Wait Time Control Register (IAP_TPS)

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

Needs to be set according to the operating frequency

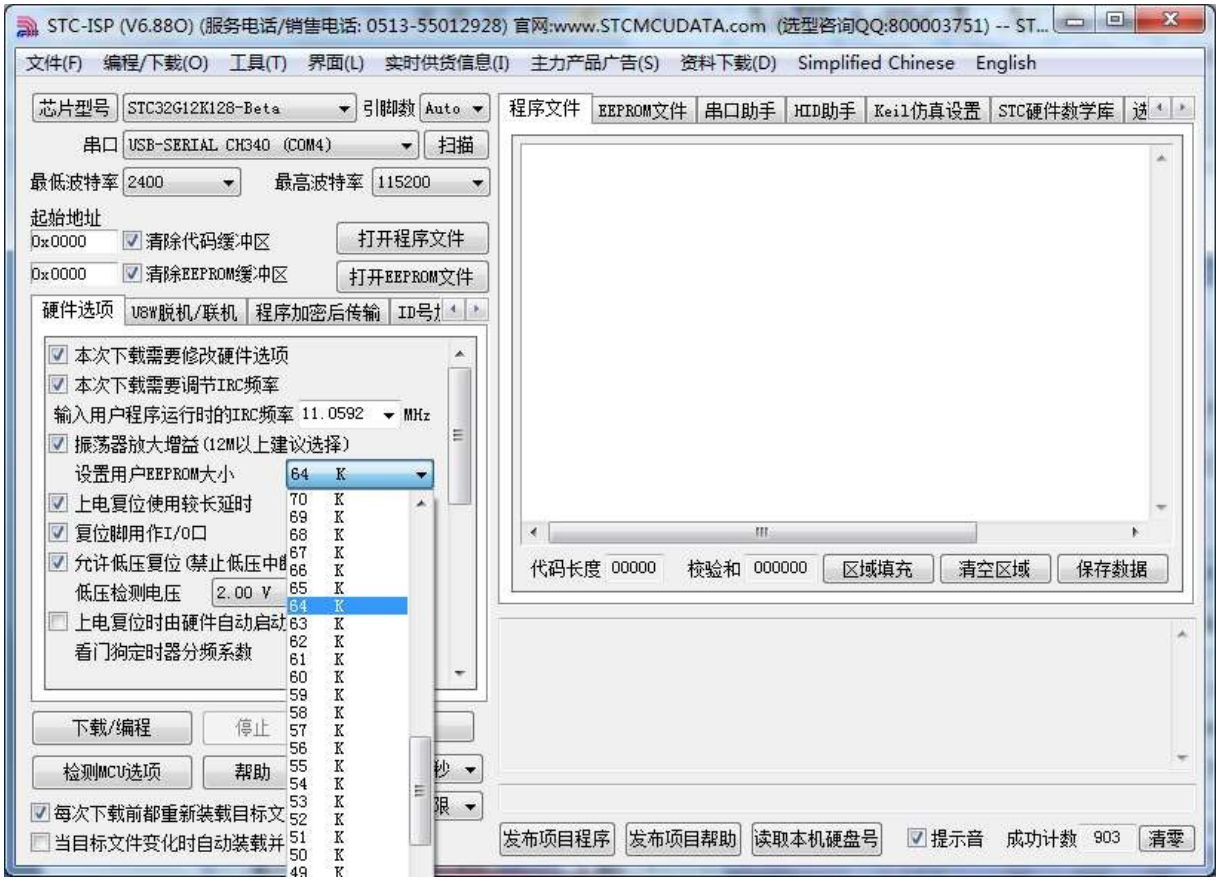
If the operating frequency is 12MHz, you need to set IAP_TPS to 12; if the operating frequency is 24MHz, you need to set IAP_TPS to 24, and so on for other frequencies.

18.3 EEPROM Size and Address

STC32G series microcontrollers have internal EEPROM for storing user data, and the internal EEPROM has three operation modes: read, write and erase, in which the erase operation is carried out by sector, each sector is 512 bytes, i.e., one sector will be erased every time the erase command is executed, whereas the read data and write data are carried out by byte, i.e., only one byte can be read out or written in every time the read or write commands are executed. or write command can only read or write one byte.

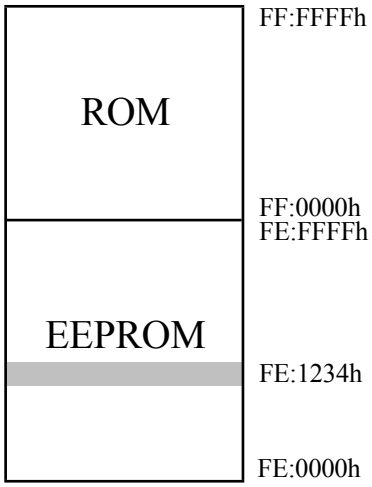
There are two ways to access the EEPROM in STC32G series microcontroller: IAP and MOV. IAP can read, write and erase the EEPROM, but MOV can only read the EEPROM, but not write or erase it. To access the EEPROM using either the IAP or MOV method, you need to set the correct destination address first. When using IAP method, the address data is the target address of the EEPROM, which starts from 0000, while if the MOV instruction is used to read the EEPROM data, the address data is the base address (FE:0000) plus the target address of the EEPROM. The following is an example of the STC32G12K128 to explain the target address in detail:

The EEPROM size of the STC32G12K128 needs to be set during the ISP download as shown in the figure below, setting the EEPROM to 64K.



The EEPROM is located at FE:0000h~FE:FFFFh in the 128K Flash memory space (Note: No matter how much the EEPROM is set to, the EEPROM is located at FE:0000h~FE:FFFFh).

EEPROM always starts at FE:0000h in Flash memory space) as shown below:



STC32G12K128

Now, when you need to read, write or erase the EEPROM unit with physical address 1234h, if you use IAP to access it, set the target address to 1234h, that is, IAP_ADDRE is set to 00h, IAP_ADDRH is set to 12h, IAP_ADDRL is set to 34h, and then set the corresponding trigger commands to correctly operate the 1234h unit. Then, set the corresponding trigger command to correctly operate unit 1234h. However, if you use MOV to read the 1234h unit of EEPROM, the target address is the base address FE:0000h plus 1234h, i.e., you must set the 32-bit register DRx to FE:1234h to read the unit correctly with MOV instruction (Note: STC32G series and STC8 series are different, and you can not read the EEPROM with MOVC)(Note: the STC32G series cannot use MOVC to read EEPROM as the STC8 series.

The following table shows the EEPROM accesses by IAP and MOV with different EEPROM sizes.

model number	magnitude	Number of sectors	IAP mode read/write/erase		MOV Read	
			starting address	end address	starting address	end address
STC32G12K128	1K	2	0:0000h	0:03FFh	FE:0000h	FE:03FFh
	2K	4	0:0000h	0:07FFh	FE:0000h	FE:07FFh
	4K	8	0:0000h	0:0FFFh	FE:0000h	FE:0FFFh
	8K	16	0:0000h	0:1FFFh	FE:0000h	FE:1FFFh
	16K	32	0:0000h	0:3FFFh	FE:0000h	FE:3FFFh
	32K	64	0:0000h	0:7FFFh	FE:0000h	FE:7FFFh
	64K	128	0:0000h	0:FFFFh	FE:0000h	FE:FFFFh
	96K	192	0:0000h	1:7FFFh	FE:0000h	FF:7FFFh
	128K	256	0:0000h	1:FFFFh	FE:0000h	FF:FFFFh

18.4 sample procedure

18.4.1 EEPROM Basic Operation

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void IapIdle()
{
    IAP_CONTR = 0;                           //Disable IAP function
    IAP_CMD = 0;                             //clear command register
    IAP_TRIG = 0;                            //clear trigger register
    IAP_ADDRE = 0x00;
    iap_addrh = 0x00; iap_addrl =
    0x00.
}

char IapRead(unsigned long addr)
{
    char dat.

    IAP_CONTR = 0x80.                        //Enable IAP
    IAP_TPS = 12;                            //set wait parameter 12MHz
    IAP_CMD = 1;                             //Set the IAP read command
    IAP_ADDRL = addr.                        //set IAP low address
    IAP_ADDRH = addr >> 8;                   //set IAP high address
    IAP_ADDRE = addr >> 16;                  //Set IAP highest address
    IAP_TRIG = 0x5a.                         //write trigger command (0x5a)
    IAP_TRIG = 0xa5.                         //write trigger command (0xa5)
    _nop_().
    _nop_().
    _nop_().
    _nop_().
    dat = IAP_DATA;                          //read IAP data
    IapIdle();                               //Disable the IAP function

    return dat.
}

void IapProgram(unsigned long addr, char dat)
{
    IAP_CONTR = 0x80.                        //Enable IAP
    IAP_TPS = 12;                            //set wait parameter 12MHz
    IAP_CMD = 2;                             //Set the IAP write command
    IAP_ADDRL = addr.                        //Set IAP low address
    IAP_ADDRH = addr >> 8;                   //set IAP high address
    IAP_ADDRE = addr >> 16;                  //Set IAP highest address
    IAP_DATA = dat;                          //write IAP data
    IAP_TRIG = 0x5a.                         //write trigger command (0x5a)
    IAP_TRIG = 0xa5.                         //write trigger command (0xa5)
    _nop_().
    _nop_().
```

```
    _nop_().
    _nop_().
    IapIdle();                              //Disable the IAP function
}

void IapErase(unsigned long addr)
{
    IAP_CONTR = 0x80.                       //Enable IAP
    IAP_TPS = 12;                           //set wait parameter 12MHz
    IAP_CMD = 3;                            //Set the IAP erase command
    IAP_ADDRL = addr.                       //set IAP low address
    IAP_ADDRH = addr >> 8;                  //set IAP high address
    IAP_ADDRE = addr >> 16;                 //Set IAP highest address
    IAP_TRIG = 0x5a.                        //write trigger command (0x5a)
    IAP_TRIG = 0xa5.                        //write trigger command (0xa5)
    _nop_().
    _nop_().
    _nop_().
    _nop_().                                //
    IapIdle();                              //Disable the IAP function
}

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set the external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign a value of 0 to set the CPU to execute the
                                            programme as fast as possible.

    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);
}
```

18.4.2 Reading EEPROM with MOV

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
```

```
#include "stc32g.h"                    // see download software for header files
```

```
#include "intrins.h"

#define      IAP_BASE

0xfe0000 void IapIdle()
{
    IAP_CONTR = 0;                          //Disable IAP function
    IAP_CMD = 0;                            //clear command register
    IAP_TRIG = 0;                           //clear trigger register
    IAP_ADDRE = 0x00;
    iap_addrh = 0x00; iap_addrl =
    0x00.
}

char IapRead(unsigned long addr)
{
    Addr = (addr & 0x1ffff) | IAP_BASE.     // Use MOV to read EEPROM and add base address.
    return *(char far *)(addr); //Use MOV to read the data.   //Use MOV to read data
}

void IapProgram(unsigned long addr, char dat)
{
    IAP_CONTR = 0x80.                       //Enable IAP
    IAP_TPS = 12;                           //set wait parameter 12MHz
    IAP_CMD = 2;                            //Set the IAP write command
    IAP_ADDRL = addr.                       //set IAP low address
    IAP_ADDRH = addr >> 8;                  //set IAP high address
    IAP_ADDRE = addr >> 16;                 //set IAP highest address
    IAP_DATA = dat;                         //write IAP data
    IAP_TRIG = 0x5a.                        //write trigger command (0x5a)
    IAP_TRIG = 0xa5.                        //write trigger command (0xa5)
    _nop_().
    _nop_().
    _nop_().
    _nop_().
    IapIdle();                              //Disable the IAP function
}

void IapErase(unsigned long addr)
{
    IAP_CONTR = 0x80.                       //Enable IAP
    IAP_TPS = 12;                           //set wait parameter 12MHz
    IAP_CMD = 3;                            //Set the IAP erase command
    IAP_ADDRL = addr.                       //set IAP low address
    IAP_ADDRH = addr >> 8;                  //set IAP high address
    IAP_ADDRE = addr >> 16;                 //Set IAP highest address
    IAP_TRIG = 0x5a.                        //write trigger command (0x5a)
    IAP_TRIG = 0xa5.                        //write trigger command (0xa5)
    _nop_().
    _nop_().
    _nop_().
    _nop_().                                //
    IapIdle();                              //Disable the IAP function
}

void main()
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set external data bus speed to fastest
```

```
    WTST = 0x00.                                //Set the parameter for waiting for the programme code.
                                                //Assign 0 to set the CPU to execute the programme as fast as possible.

    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);
}
```

18.4.3 Send EEPROM data using the serial port

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - (FOSC / 115200+2) / 4)
                                                //The // plus 2 operation is to allow the Keil compiler to
                                                //Automatic implementation of rounding operations

void UartInit()
{
    scon = 0x5a; t2l
    = brt.
    T2H = BRT >> 8;
    S1BRT = 1.
    T2x12 = 1.
    T2R = 1.
}

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_ADDRE = 0x00;
    iap_addrh = 0x00; iap_addrl =
    0x00.
}

char IapRead(unsigned long addr)
{
    char dat.

    IAP_CONTR = 0x80.                       //Enable IAP
    IAP_TPS = 12;                           //set wait parameter 12MHz
    IAP_CMD = 1;                            //Set the IAP read command
    IAP_ADDRL = addr.                       //set IAP low address
    IAP_ADDRH = addr >> 8;                  //set IAP high address
    IAP_ADDRE = addr >> 16;                 //Set IAP highest address
    IAP_TRIG = 0x5a.                        //write trigger command (0x5a)
    IAP_TRIG = 0xa5.                        //write trigger command (0xa5)
    _nop_().
    _nop_().
    _nop_().
    _nop_().
    dat = IAP_DATA;                         //read IAP data
    IapIdle();                              //Disable the IAP function

    return dat.
}

void IapProgram(unsigned long addr, char dat)
{
    IAP_CONTR = 0x80.                       //Enable IAP
    IAP_TPS = 12;                           //set wait parameter 12MHz
    IAP_CMD = 2;                            //Set the IAP write command
    IAP_ADDRL = addr.                       //set IAP low address
    IAP_ADDRH = addr >> 8;                  //set IAP high address
    IAP_ADDRE = addr >> 16;                 //Set IAP highest address
    IAP_DATA = dat;                         //write IAP data
    IAP_TRIG = 0x5a.                        //write trigger command (0x5a)
    IAP_TRIG = 0xa5.                        //write trigger command (0xa5)
    _nop_().
    _nop_().
    _nop_().
    _nop_().
    IapIdle();                              //Disable the IAP function
}

void IapErase(unsigned long addr)
{
    IAP_CONTR = 0x80.                       //Enable IAP
    IAP_TPS = 12;                           //set wait parameter 12MHz
    IAP_CMD = 3;                            //Set the IAP erase command
    IAP_ADDRL = addr.                       //set IAP low address
    IAP_ADDRH = addr >> 8;                  //set IAP high address
    IAP_ADDRE = addr >> 16;                 //Set IAP highest address
    IAP_TRIG = 0x5a.                        //write trigger command (0x5a)
    IAP_TRIG = 0xa5.                        //write trigger command (0xa5)
    _nop_().
```

```
    _nop_().
    _nop_().
    _nop_().                                    //
    IapIdle();                                  //Disable the IAP function
}

void main()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign a value of 0 to set the CPU to execute the
                                                programme as fast as possible.

    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);
}
```

18.4.4 Serial Port 1 Read/Write EEPROM-with MOV Read

(main.c)

```
//Tested operating frequency is 11.0592MHz

/*      This programme has been tested completely normal, do not provide telephone technical support, if you
can not understand, please supplement their own relevant basis.  */

/************ Functional Description of
this Program                    **************
STC32G Series EEPROM General Test
Program.
Please don't modify the programme first, download and
test it directly. PC serial port settings: baud rate
115200,8,n,1.
Sector erase, write 64 bytes, and read 64 bytes to the EEPROM.

Example command.
E 0     Perform sector erase operation on EEPROM, E means erase, number 0 is for sector 0 (decimal, 0~126,
depends on specific IC).
W 0     is a write operation to the EEPROM, W means write, the number 0 is for sector 0 (decimal, 0~126,
```

depends on the specific *IC). The number 0 is for sector 0 (decimal, 0~126, depending on the specific IC). 64* bytes are written consecutively from the start address of the sector.

R 0 performs an *IAP* read operation to *the EEPROM*, *R* means read, *and* the number *0* is for sector *0 (*decimal, *0~126*, depending on the specific *IC). The number 0 is for sector 0 (decimal, 0~126, depending on the specific IC). 64* bytes are read consecutively from the start address of the sector.

M 0 *MOVC* read operation to *EEPROM (*operation address is sector*512+offset address*), the number *0* is sector *0 (*decimal, *0~126*, depends on specific *IC).*

Read *64* consecutive bytes from the beginning of the sector.

Note: For general purpose, the programme does not identify whether the

sector is valid or not, the user decides for himself 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       0xfe0000        //Define the base address to be added when EEPROM is accessed by MOV.
#define     TimeOutSet1         5

/************ local constants ☞ Ming **************/
u8    code T_Strings[]={"Last year today, in this door, the peach blossoms of the human face reflected each other. I don't know where the human face has gone, but the peach blossoms are still smiling in spring."};

/************ local variable ☞ Ming
**************/ u8       xdatatmp[70].
u8    xdata
RX1_Buffer[UART1_BUF_LENGTH]; u8
RX1_Cnt.
u8
RX1_TimeOut; bit
      B_TX1_Busy.

/************ Local              **************/
Functions☞Ming
void UART1_config(void).
void TX1_write2buff(u8 dat).                    //Write to
                                                transmit buffer
void PrintString1(u8 *puts).                    //Send a string

/************ External Functions and Variables ☞ Ming *****************/

/*************************************************/

      u8CheckData(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    u16
      address; u8; i;

      address = 0;
      if(RX1_Cnt < 3 )        return 65535;                   //error
```

```
    if(RX1_Cnt <= 5)                                    /5 bytes or less is a sector operation, decimal, or
                                                        //Support Commands: 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)                                   //Limit to sectors 0~123
            {
                address <<= 9;
                return (address).
            }
        }
        else if(RX1_Cnt == 8)                                   /8 bytes direct address operation, hexadecimal,/8 bytes direct address operation, hexadecimal,/8 bytes direct address operation, hexadecimal
                                                                //Support commands: 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);                           //Limit to sectors 0~123
            }
        }

        return      65535;                                      //error
}

//==========================================================================
// Function: void delay_ms(u8 ms)
// Description: A delay function.
// Parameters: ms, the number of ms to delay, only 1~255ms are supported here. Auto adapt to master clock.
// Returns: none.
// Version: VER1.0
// Date: 2013-4-1
// Remarks.
//==========================================================================
void delay_ms(u8 ms)
{
    u16 i;
    do
    {
        i = MAIN_Fosc / 10000;
        while(--i) ;
    }while(---ms).
}

//Use MOV to read EEPROM.
void EEPROM_MOV_read_n(u16 EE_address, u8 *DataAddress, u16 number)
{
        u8far *pc.

    pc = EE_address + EEADDR_OFFSET;
    do
    {
        *DataAddress = *pc.                                     //Read out the data
        DataAddress++;
        pc++;
    }while(--number);
}
```

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

    UART1_config().                                  // select baud rate, 2: use Timer2 for baud rate, 2: use
    Timer2 for baud rate, 3: use Timer2 for baud rate
                                                     //Other values: Use Timer1 for baud rate.
    EA = 1;                                          // Allow total interrupt

    PrintString1("STC MCU testing EEPROM programme with serial port 1! \r\n"); //UART1 sends a character 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)                 //Limit to sectors 0~123
                    {
                        if(RX1_Buffer[0] == 'E')     //PC request to erase a sector
                        {
                            EEPROM_SectorErase(addr);
                            PrintString1("Sector erase
                            complete! \r\n");;
                        }

                        else if(RX1_Buffer[0] == 'W')    //PC request to write 64 bytes of EEPROM data.
                        {
                            EEPROM_write_n(addr,T_Strings,64);
                            PrintString1("Write operation
                            complete! \r\n");
                        }

                        else if(RX1_Buffer[0] == 'R')    //PC request to return 64 bytes of EEPROM data.
                        {
                            PrintString1("The IAP reads the following data: \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 request to return 64 bytes of EEPROM data.
                        {
                            PrintString1("MOVC read out the data as follows:\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("Command error! \r\n");
                    }
```

```
        else PrintString1("Command error! \r\n");
}
```

```
                        RX1_Cnt = 0;
                    }
                }
            }
        }
}
/*********************************************/

/*************** send a byte *******************************/
void TX1_write2buff(u8 dat)                     //write to transmit buffer
{
    B_TX1_Busy = 1; //Flag transmit busy.       //flag transmission busy
    SBUF = dat;                                 //send a byte
    while(B_TX1_Busy).                          //Wait for the transmission to finish
}

//========================================================================
// Function: void PrintString1(u8 *puts)
// Description: Serial 1 Send String function.
// Parameters: puts: Pointer to string.
// Returns: none.
// Version: VER1.0
// Date: 2014-11-28
// Remarks.
//========================================================================
void PrintString1(u8 *puts)                     // Send a string
{
    for (; *puts != 0; puts++)                  //Enter a stopper 0 and end
    {
        TX1_write2buff(*puts).
    }
}

//========================================================================
// Functions: void   UART1_config(void)
// Description: UART1 initialisation function.
// Parameters: none.
// Returns: none.
// Version: VER1.0
// Date: 2014-11-28
// Remarks.
//========================================================================
void UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01.                              //S1 BRT Use Timer1.
    AUXR |= (1<<6); //Timer1 set as 1T mode.    //Timer1 set as 1T mode
    TMOD &= ~(1<<6); //Timer1 Set As Timer      //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.       // Disable Timer1 interrupt
    INT_CLKO &= ~0x02.                          // Timer1 does not output high speed clock
    TR1 =  1; // Run Timer1.                    // Run 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: Synchronous shift output, 0x00:
```

Synchronous shift output, 0x00: Synchronous shift output, 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;                                         //Allow interrupt
    REN = 1;                                        //Allow to receive

    B_TX1_Busy = 0;
    RX1_Cnt = 0;
}

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

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

(EEPROM.c)

```
//Tested operating frequency is 11.0592MHz

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

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

//========================================================================
// Functions: void      ISP_Disable(void)
// Description: Block access to the ISP/IAP.
// Parameters: non.
// Returns: 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 ISP/IAP commands from being triggered by
    mistake.
    ISP_ADDRE = 0xff.                               //clear 0 address high byte
    ISP_ADDRH = 0xff.                               //clear 0 address high byte
    ISP_ADDRL = 0xff.                               //Clear the low byte of address 0 to point to the non-
    EEPROM area to prevent misoperation.
}

//=========================================================================
// Function: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// Description: Reads n bytes from the specified EEPROM header address into the specified buffer.
// Parameters: EE_address: First address of the EEPROM to be read.
//             DataAddress: The first address of the read data buffer.
//             number.      Length of bytes read.
// Returns: non.
// Version: V1.0, 2012-10-22
//=========================================================================
void EEPROM_read_n(u32 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0;                                         //disable interrupt
    ISP_CONTR = ISP_EN.                             //allow ISP/IAP operation
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Operating frequency setting.      //Operating frequency setting
    ISP_READ();                                     //Send byte read command, no need to re-send the
    command when the command does not need to be changed.
    do
    {
        ISP_ADDRE = EE_address / 65536;             //send address high byte (re-send address only if
        address needs to be changed) ISP_ADDRH = (EE_address / 256) % 256;      //send address high byte
        (re-send address only if address needs to be changed) ISP_ADDRL = EE_address % 256      ;
                                                    //send address low byte
        ISP_TRIG().                                 //first send 5AH, then send A5H to ISP/IAP trigger
        register.
                                                    //Every time it's needed
                                                    //The ISP/IAP command is triggered to start immediately
                                                    after sending A5H.
                                                    //CPU waits for the IAP to complete before continuing to
                                                    execute the programme.
        _nop_().
        _nop_().
        _nop_().
        _nop_().
        *DataAddress = ISP_DATA;                    //read out data
        sent to EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = 1;                                         //Re-allow interrupts
}

/****************** Sector Erase Function *****************/
//=========================================================================
// Function: void EEPROM_SectorErase(u16 EE_address)
// Description: Erases an EEPROM sector at the specified address.
// Parameters: EE_address: Address of the sector EEPROM to erase.
// Returns: non.
// Version: V1.0, 2013-5-10
//=========================================================================
void EEPROM_SectorErase(u32 EE_address)
```

```
{
    EA = 0;                                 // disable interrupt
                                            //Only sector erase, no byte erase, 512 bytes/sector.
                                            // Any byte address in a sector is a sector address.
    ISP_ADDRE = EE_address / 65536;         //send address high byte (re-send address only if address needs to be changed) ISP_ADDRH = (EE_address / 256) % 256;  //send address high byte (re-send address only if address needs to be changed) ISP_ADDRL = EE_address % 256    ;          //send address low byte
```

```
    ISP_CONTR = ISP_EN.                                 //allow ISP/IAP operation
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Operating frequency setting.      //Operating frequency setting
    ISP_ERASE();                                        //Send the sector erase command, when the command does
    not need to be changed, there is no need to re-send the command.
    ISP_TRIG().
    _nop_().
    _nop_().
    _nop_().
    _nop_().
    DisableEEPROM();
    EA = 1;                                             //Re-allow interrupts
}

//========================================================================
// Function: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// Description: Writes the buffered n bytes to the EEPROM at the specified first address.
// Parameters: EE_address: First address to write to EEPROM.
//             DataAddress: The first address of the buffer to which the source data is written.
//             number.          Length of bytes written.
// Returns: non.
// Version: V1.0, 2012-10-22
//========================================================================
void EEPROM_write_n(u32 EE_address,u8 *DataAddress,u16 number)
{
    EA = 0;                                             // disable interrupt

    ISP_CONTR = ISP_EN.                                 //allow ISP/IAP operation
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Operating frequency setting.      //Operating frequency setting
    ISP_WRITE();                                        //Send the byte write command, when the command does
    not need to be changed, there is no need to re-send the command.
    do
    {
        ISP_ADDRE = EE_address / 65536;                 //send address high byte (re-send address only if
        address needs to be changed) ISP_ADDRH = (EE_address / 256) % 256;      //send address high byte
        (re-send address only if address needs to be changed) ISP_ADDRL = EE_address % 256      ;
                                                        //send address low byte
        ISP_DATA =  *DataAddress.                       //Send data to ISP_DATA, only need to re-send when data
        change.
        ISP_TRIG().
        _nop_().
        _nop_().
        _nop_().
        _nop_().
        EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = 1;                                             //Re-allow interrupts
}
```

18.4.5 Password Erase Write-Multi-Sector Backup-Serial Port 1 Operation

(main.c)

//Tested operating frequency is *11.0592MHz*

/* This programme has been tested completely normal, do not provide telephone technical support, if you can not understand, please supplement their own relevant basis. */

/********** Functional description of this programme **********

STC32G series *EEPROM* general-purpose test program, demonstrates multi-sector backup, write with correct sector data if there is a sector error, and write with all sector errors (e.g., first time).

(run the programme) then the default value is written.

Each write is to 3 sectors, i.e. redundant backups.
Write one record for each sector, read out the saved data and parity value to compare with the source data and parity value, and return the result (correct or error message) from serial port 1 (P3.0 P3.1).
Each record is self-checking, 64 bytes of data, 2 bytes of checksum, checksum = sum of 64 bytes of data ^ 0x5555.
^0x5555 is to make sure that not all 66 data writes are zero.
If there is a sector error, write the data of the correct sector to the error sector, if all 3 sectors are wrong, then write the default value to all of them.
Erase, write, and read operations require a passphrase, and if the passphrase is incorrect, the operation is exited, and the passphrase is cleared each time the operation is exited.

Please don't modify the program first, directly download "UART-EEPROM.hex" from "03-Manual Erase Write-Multi-Sector Backup-Serial Port 1 Operation" to test it. Select the main frequency 11.0592MHZ when downloading.

PC serial port settings: baud rate 115200,8,n,1.
Sector erase, write 64 bytes, and read 64 bytes to the EEPROM.

Example command.
Use the Serial Assistant to send a single character, in both case and size.

Send E or e: to erase sector of EEPROM, E means erase, will erase sector 0, 1, 2.

W or w: Write operation to the EEPROM, W means write, will write to sector 0, 1, 2, each sector will be written continuously 64 bytes, sector 0 write
0x0000~0x003f, Sector 1 is written to 0x0200~0x023f, Sector 0 is written to 0x0400~0x043f.

Send R or r: Read operation to EEPROM, R means read, sector 0, 1, 2 will be read, each sector will be read continuously with 64 bytes, sector 0 will be read.
0x0000~0x003f, sector 1 reads 0x0200~0x023f, sector 0 reads 0x0400~0x043f.

Note: For the sake of generality, the programme does not identify whether the sector is valid or not, the user decides for himself according to the specific model.

Date: 2021-11-5
**********/

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

#define             Baudrate1               115200L

/************* local constants ☞min **************/
u8    code T_StringD[]={"Last year at this door, the peach blossoms were red. I don't know
where the human face has gone, but the peach blossoms are still smiling in spring."}; u8  code
T_StringW[]={"The peaks of the mountains are different from each other, and the heights of the
mountains are different from each other. I don't know the true face of Mount Lushan, but only
the fact that I am here in the mountain."};


/************* local                 **************/
variable ☞min
u8    xdatatmp[70].                                 //Common Data
u8    xdata SaveTmp[70].                            //Array to be
                                                    written to

bit
B_TX1_Busy; u8                                      //Serial port
u8; cmd.
```

single character

commands

```
/************ Local                ***************/

Functions☞Ming
void UART1_config(void).
void TX1_write2buff(u8 dat).
void PrintString1(u8 *puts).
```

//Write to

transmit buffer

//Send a string

```
/************ External Functions and Variables ☞Ming *****************/

/************ reads the EEPROM record and checks it, returns the result of checking, 0 is correct, 1 is error
*****************/
```

```
	u8ReadRecord(u16 addr)
{
	u8	i;
	u16 ChckSum.                                        //calculated cumulative sum
	u16 j.                                              //Reading the cumulative sum

	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++)                      //Calculate the cumulative sum
		ChckSum += tmp[i];

	j = ((u16)tmp[64]<<8) + (u16)tmp[65];               //Read the cumulative sum of the records
	j ^= 0x5555;                                        //Invert every bit to avoid all zeros.
	if(ChckSum ! = j)return 1;                          //Accumulation and error, return 1
	return	0.                                          //Accumulate and sum correctly, return 0.
}
```

/************* Write to EEPROM record and check, return check result, 0 is correct, 1 is error. ******************/

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

	for(ChckSum=0, i=0; i<64; i++)                      //Calculate the cumulative sum
		ChckSum += SaveTmp[i];

	ChckSum ^= 0x5555.                                  //Invert every bit to avoid all zeros.
	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 Sector

	for(i=0; i<66; i++)                                 //clear buffer
		tmp[i] = 0;
	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;                           //Data error, return 1
	}
	return 0; //The sum is correct, return 0.           //Accumulate and sum correctly, return 0.
}
```

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

```
void main(void)
{
	u8	i;
	u8	status;                                     //status

	UART1_config().     // select baud rate, 2: use Timer2 for baud rate, 2: use Timer2 for baud rate, 3: use Timer2 for baud rate
	                    //Other values: Use Timer1 for baud rate.
	EA = 1;                                             //Allow total interrupt

	PrintString1("STC8G-8H-8C series MCU testing EEPROM programme with serial port 1! \r\n"); //UART1 sends a string.
```

```
//Power up reads 3 sectors and checksums them, if any
sector is wrong then sends the correct one to the user.
```

```
                                                    //Sector Write Error, if all 3 sectors are in error, then write default value
    status = 0;
    if(ReadRecord(0x0000) == 0)                     //read sector 0
    {
        status |= 0x01;                             //To be correct, mark status.0=1.
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i];                    //save in write buffer
    }
    if(ReadRecord(0x0200) == 0)                     //read sector 1
    {
        status |= 0x02;                             //To be correct, mark status.1=1.
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i];                    //save in write buffer
    }
    if(ReadRecord(0x0400) == 0)                     //read sector 2
    {
        status |= 0x04;                             //To be correct, mark status.2=1.
        for(i=0; i<64; i++)
            SaveTmp[i] = tmp[i];                    //save in write buffer
    }

    if(status == 0)                                 //all sectors are in error, then write to default value
    {
        for(i=0; i<64; i++)
            SaveTmp[i] = T_StringD[i];              //read the default value
    }
    else PrintString1("Power up and read 3 sectors data are correct! \r\n");; //UART1 sends a string prompt.    //UART1 sends a string prompt

    if((status & 0x01) == 0)                        //Sector 0 error, write default value.
    {
        if(SaveRecord(0x0000) == 0)
            PrintString1("Write sector 0 correctly! \r\n"); //Write record sector 0 correctly.
        else
            PrintString1("Write sector 0 error! \r\n");     //Write Record Sector 0 Error
    }
    if((status & 0x02) == 0)                        //Sector 1 error, write default value.
    {
        if(SaveRecord(0x0200) == 0)
            PrintString1("Write sector 1 correctly! \r\n"); //Write record 1 sector correctly.    //Write record sector 1 correctly.
        else
            PrintString1("Write Sector 1 Error! \r\n");     //Write Record 1 Sector Error
    }
    if((status & 0x04) == 0)                        //Sector 2 error, write default value.
    {
        if(SaveRecord(0x0400) == 0)
            PrintString1("Write sector 2 correctly! \r\n"); //Write record sector 2 correctly.    //Write record sector 2 correctly.
        else
            PrintString1("Write Sector 2 Error! \r\n"); //Write Record 2 Sector Error!    //Write Record Sector 2 Error
    }


    while(1)
    {
        if(cmd ! = 0)                               //with serial commands
        {
```

```
        if((cmd >= 'a') && (cmd <= 'z'))
            cmd -= 0x20;                        //Lowercase to uppercase

        if(cmd == 'E')                          //PC request 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("Sector erase complete! \r\n");
    }

    else if(cmd == 'W')                         //PC request to write 64 bytes of EEPROM data.
    {
        for(i=0; i<64; i++)
            SaveTmp[i] = T_StringW[i];          //write value
        if(SaveRecord(0x0000) == 0)
            PrintString1("Write sector 0 correctly! \r\n");     //Write record sector 0 correctly.
        else
            PrintString1("Write sector 0 error! \r\n");  //Write Record Sector 0 Error
        if(SaveRecord(0x0200) == 0)
            PrintString1("Write sector 1 correctly! \r\n"); //Write record 1 sector correctly.      //Write record sector 1 correctly.
        else
            PrintString1("Write Sector 1 Error! \r\n");  //Write Record 1 Sector Error
        if(SaveRecord(0x0400) == 0)
            PrintString1("Write sector 2 correctly! \r\n"); //Write record sector 2 correctly.      //Write record sector 2 correctly.
        else
            PrintString1("Write Sector 2 Error! \r\n"); //Write Record 2 Sector Error!    //Write Record Sector 2 Error
    }

    else if(cmd == 'R')                         //PC request to return 64 bytes of EEPROM data.
    {
        if(ReadRecord(0x0000) == 0)             //read the data of sector 0
        {
            PrintString1("Read the data of sector 0 as follows:\r\n");;
            for(i=0; i<64; i++)
                TX1_write2buff(tmp[i]);         //return data to serial port
            TX1_write2buff(0x0d);               //carriage return
            TX1_write2buff(0x0a).               //line feed
        }
        else PrintString1("Error reading data from sector 0! \r\n");;

        if(ReadRecord(0x0200) == 0)             //read the data of sector 1
        {
            PrintString1("Read out the data of sector 1 as follows:\r\n");;
            for(i=0; i<64; i++)
                TX1_write2buff(tmp[i]);         //return data to serial port
            TX1_write2buff(0x0d);               //carriage return
            TX1_write2buff(0x0a).               //line feed
        }
        else PrintString1("Error reading sector 1 data! \Error reading sector 1 data!)

        if(ReadRecord(0x0400) == 0)             //read the data of sector 2
```

```
{
    PrintString1("Read out the data of sector 2 as follows:\r\n");;
    for(i=0; i<64; i++)
        TX1_write2buff(tmp[i]);          //return data to
    serial port TX1_write2buff(0x0d);    //carriage
    return line feed
    TX1_write2buff(0x0a).
}
else PrintString1("Error reading data from sector 2! \r\n");;
```

```
                }
                else PrintString1("Command error! \r\n"); cmd = 0;
            }
        }
}
/*********************************************/

/*************** send a byte *******************************/
void TX1_write2buff(u8 dat)                     //write to transmit buffer
{
    B_TX1_Busy = 1; //Flag transmit busy.       //flag transmission busy
    SBUF = dat;                                 //send a byte
    while(B_TX1_Busy).                          //Wait for the transmission to finish
}

//========================================================================
// Function: void PrintString1(u8 *puts)
// Description: Serial 1 Send String function.
// Parameters: puts: Pointer to string.
// Returns: none.
// Version: VER1.0
// Date: 2014-11-28
// Remarks.
//========================================================================
void PrintString1(u8 *puts)                     // Send a string
{
    for (; *puts ! = 0; puts++)                 //Enter a stopper 0 and end
    {
        TX1_write2buff(*puts).
    }
}

//========================================================================
// Functions: void   UART1_config(void)
// Description: UART1 initialisation function.
// Parameters: none.
// Returns: none.
// Version: VER1.0
// Date: 2014-11-28
// Remarks.
//========================================================================
void UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01.                              //S1 BRT Use Timer1.
    AUXR |= (1<<6); //Timer1 set as 1T mode.    //Timer1 set as 1T mode
    TMOD &= ~(1<<6); //Timer1 Set As Timer      //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.       // Disable Timer1 interrupt
    INT_CLKO &= ~0x02.                          // Timer1 does not output high speed clock
    TR1 =  1; // Run Timer1.                    // Run 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: Synchronous shift
                                                                output, 0x40.
                                                    //          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;                                                     // Allow interruptions
    REN = 1;                                                    //Allowed to receive

    B_TX1_Busy = 0;
}

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

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

(EEPROM.c)

```
//Tested operating frequency is 11.0592MHz

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

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

u32         PassWord.                               // Erase The passphrase required for writing.

//========================================================================
// Functions: void      ISP_Disable(void)
// Description: Block access to the ISP/IAP.
// Parameters: non.
// Returns: 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 ISP/IAP commands from being triggered by
mistake.
```

```
    ISP_ADDRE = 0xff.                       //clear 0 address high byte
    ISP_ADDRH = 0xff.                       //clear 0 address high byte
    ISP_ADDRL = 0xff.                       //Clear the low byte of address 0 to point to the non-
    EEPROM area to prevent misoperation.
}

//=========================================================================
// Function: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// Description: Reads n bytes from the specified EEPROM header address into the specified buffer.
// Parameters: EE_address: First address of the EEPROM to be read.
//             DataAddress: The first address of the read data buffer.
//             number.          Length of bytes read.
// Returns: non.
// Version: V1.0, 2012-10-22
//=========================================================================
void EEPROM_read_n(u32 EE_address,u8 *DataAddress,u16 number)
{
    if(PassWord == D_PASSWORD)              //The EEPROM will be operated only if the password is
    correct.
    {
        EA = 0;                             // disable interrupt
        ISP_CONTR = ISP_EN.                 //allow ISP/IAP operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Operating frequency setting. //Operating frequency setting
        ISP_READ();                         //Send byte read command, when the command does not
        need to change, do not need to re-send the command
        do
        {
        ISP_ADDRE = EE_address / 65536;     //send address high byte (re-send address only if
        address needs to be changed) ISP_ADDRH = (EE_address / 256) % 256;       //send address high byte
        (re-send address only if address needs to be changed) ISP_ADDRL = EE_address % 256        ;
                                            //send address low byte
            if(PassWord == D_PASSWORD)      //The operation will be triggered only if the password is
            correct.
            {
                ISP_TRIG = 0x5A.            //first send 5AH, then send A5H to ISP/IAP trigger
                register.
                                            //Every time it's needed
                ISP_TRIG = 0xA5.            //After sending A5H, ISP/IAP command is triggered
                immediately.
            }                               // The CPU waits for the IAP to complete before continuing to
            execute the program.
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            *DataAddress = ISP_DATA;        //read data sent
            to EE_address++;
            DataAddress++;
        }while(--number);

        DisableEEPROM();
        EA = 1;                             //Re-allow interrupts
    }
    PassWord = 0;                           //clear password
}

/******************** Sector Erase Function *****************/
//=========================================================================
// Function: void EEPROM_SectorErase(u16 EE_address)
```

```
// Description: Erases an EEPROM sector at the specified address.
// Parameters: EE_address: Address of the sector EEPROM to erase.
// Returns: non.
// Version: V1.0, 2013-5-10
//========================================================================
void EEPROM_SectorErase(u32 EE_address)
{
    if(PassWord == D_PASSWORD)                  //The EEPROM will be operated only if the password is
    correct.
```

```
    {
        EA = 0;                                     // disable interrupt
                                                    //Only sector erase, no byte erase, 512 bytes/sector.
                                                    // Any byte address in the sector is the sector
        address. ISP_ADDRE = EE_address / 65536;    //send address high byte (re-send address only if
        address needs to be changed) ISP_ADDRH = (EE_address / 256) % 256;      //send address high byte
        (re-send address only if address needs to be changed) ISP_ADDRL = EE_address % 256      ;
                                                    //send address low byte
        ISP_CONTR = ISP_EN.                         //allow ISP/IAP operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Operating frequency setting. //Operating frequency setting
        ISP_ERASE();                                //Send the sector erase command, the command
            does not need to change, do not need to re-send the command if (PassWord == D_PASSWORD)
                                                    //Password password is correct before
            triggering the operation
            {
                ISP_TRIG = 0x5A.                    //first send 5AH, then send A5H to ISP/IAP trigger
                register.
                                                    //Every time it's needed
                ISP_TRIG = 0xA5.                    //After sending A5H, ISP/IAP command is triggered
                immediately.
}                                                   // The CPU waits for the IAP to complete before continuing to execute the
                                                    program.
        _nop_().
        _nop_().
        _nop_().
        _nop_().
        DisableEEPROM();
        EA = 1;                                     //Re-allow interrupts
    }
    PassWord = 0;                                   //clear password
}

//=========================================================================
// Function: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// Description: Writes the buffered n bytes to the EEPROM at the specified first address.
// Parameters: EE_address: First address to write to EEPROM.
//         DataAddress: The first address of the buffer to which the source data is written.
//         number.      Length of bytes written.
// Returns: non.
// Version: V1.0, 2012-10-22
//=========================================================================
void EEPROM_write_n(u32 EE_address,u8 *DataAddress,u16 number)
{
    if(PassWord == D_PASSWORD)                      //The EEPROM will be operated only if the password is
    correct.
    {
        EA = 0;                                     // disable interrupt

        ISP_CONTR = ISP_EN.                         //allow ISP/IAP operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Operating frequency setting. //Operating frequency setting
        ISP_WRITE();                                //Send the byte write command, when the command does
        not need to be changed, there is no need to re-send the command.
        do
        {
        ISP_ADDRE = EE_address / 65536;             //send address high byte (re-send address only if
        address needs to be changed) ISP_ADDRH = (EE_address / 256) % 256;      //send address high byte
        (re-send address only if address needs to be changed) ISP_ADDRL = EE_address % 256      ;
                                                    //send address low byte
            ISP_DATA = *DataAddress;                //send data to ISP_DATA, only need to re-send
            if(PassWord == D_PASSWORD)              //trigger operation only if password is
```

```
correct
{
    ISP_TRIG = 0x5A.                    //first send 5AH, then send A5H to ISP/IAP trigger
    register.

                                        //Every time it's needed
    ISP_TRIG = 0xA5.                    //After sending A5H, ISP/IAP command is triggered
    immediately.

}                                       // The CPU waits for the IAP to complete before continuing to
execute the program.
_nop_().
_nop_().
_nop_().
```

```
                _nop_().
                EE_address++;
                DataAddress++;
            }while(--number);

            DisableEEPROM();
            EA = 1;                                    //Re-allow interrupts
        }
        PassWord = 0;                                  //clear password
}
```

19 ADC analogue-to-digital conversion, legacy DAC implementation

STC32G series microcontrollers integrate a 12-bit high-speed A/D converter, and the clock frequency of the ADC is the system frequency divided by 2 and then divided again by a user-set frequency division coefficient (the clock frequency of the ADC ranges from SYSclk/2/1 to SYSclk/2/16).

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

Note: Channel 15 of the ADC is dedicated to the measurement of the internal 1.19V reference signal source, which is factory calibrated at 1.19V, due to manufacturing and measurement errors, the actual internal reference signal source has about ±1% error compared to 1.19V. If users need to know the exact internal reference signal source value of each chip, they can connect an external accurate reference signal source and then use channel 15 of the ADC to measure and calibrate. If users need to know the exact value of the internal reference signal source of each chip, they can connect an external accurate reference signal source, and then use the 15th channel of the ADC to measure and calibrate it. when the ADC_VRef+ pin is connected to an external reference power supply, the 15th channel of the ADC can be used to inverse the voltage of the external reference power supply of the ADC_VRef+ pin; for example, by shorting the ADC_VREF+ to the MCU-VCC, the voltage of MCU-VCC can be inversely inferred. If ADC_VREF+ is shorted to MCU-VCC, the voltage of MCU-VCC can be inverted.

If the chip has an external reference power pin ADC_VRef+ for ADC, it must not be floated and must be connected to the external reference power supply or directly to VCC.

19.1 ADC-related registers

notation	descriptions	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]				0000,0000
ADC_RES	ADC conversion result high register	BDH									0000,0000
ADC_RESL	ADC conversion result low register	BEH									0000,0000
ADCCFG	ADC Configuration Register	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
ADCTIM	ADC Timing Control Register	7EFEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]					0010, 1010

19.1.1 ADC control register (ADC_CONTR), PWM trigger ADC control

notation	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 control bit

0: Turn off ADC power

1: Switch on the ADC power.

It is recommended that the ADC power be turned off before entering idle mode and power-down mode to reduce power consumption **Special attention:**

1. After powering on the internal ADC module of the MCU, wait for about 1ms for the ADC power inside the MCU to stabilise before allowing the ADC to operate;

2, appropriate to extend the sampling time for external signals, that is, the ADC internal sampling to keep the capacitor charging or discharging time, enough time, internal and external potential equal.

ADC_START: ADC conversion start control bit. Write 1 to start ADC conversion, and the hardware will clear this bit automatically after the conversion is finished.

0: No effect. Writing 0 does not stop A/D conversion even if the ADC has already started conversion.

1: Start ADC conversion, hardware automatically clears this bit to zero after conversion is complete.

ADC_FLAG: ADC conversion end flag bit. When the ADC completes a conversion, the hardware will automatically set this bit to 1 and request an interrupt from the CPU. This flag bit must be cleared by software.

ADC_EPWMT: Enable the PWM real-time trigger ADC function. For details, please refer to the 16-bit Advanced PWM Timer chapter.

ADC_CHS[3:0]: ADC analogue channel select bits

(Note: For the I/O port selected as ADC input channel, the PxM0/PxM1 registers must be set to set the I/O port mode to high-resistance input mode. In addition, if the MCU still needs to enable the ADC channel after entering the power-down mode/clock-disable mode, it is necessary to set the PxIE register to close the digital input channel to prevent the external analogue input signals from going high and low, which may cause additional power consumption).

ADC_CHS[3:0]	ADC Channel	ADC_CHS[3:0]	ADC Channel
0000	P1.0	1000	P0.0
0001	P1.1	1001	P0.1
0010	P1.2/P5.4[1]	1010	P0.2
0011	P1.3	1011	P0.3
0100	P1.4	1100	P0.4
0101	P1.5	1101	P0.5
0110	P1.6	1110	P0.6
0111	P1.7	1111	Test Internal 1.19V

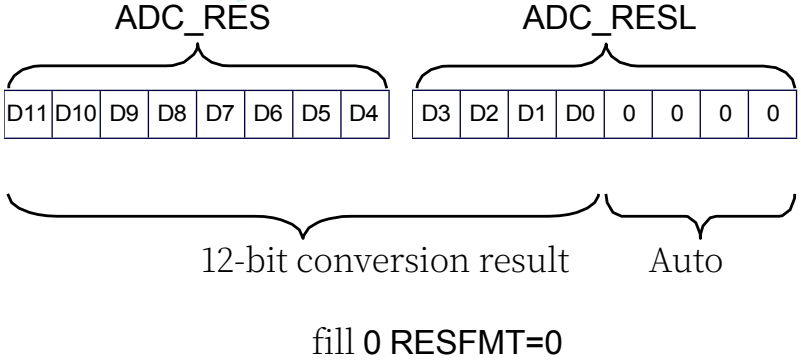
[1] ：有 P1.2 port, this function is on theP1.2 port, and for models without theP1.2 port, this function is on the P5.4 口上

19.1.2 ADC Configuration Register (ADCCFG)

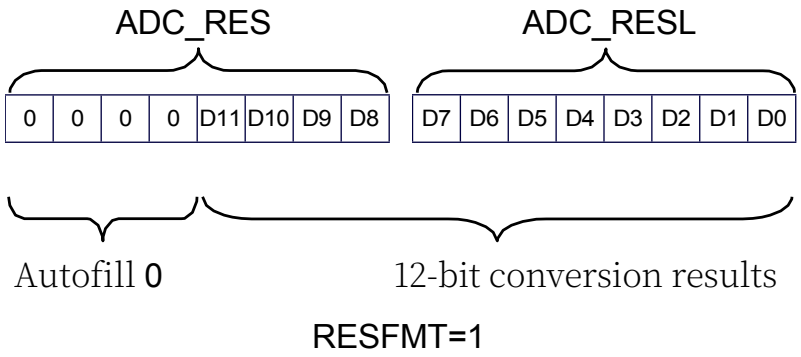
notation	address	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

RESFMT: ADC conversion result format control bit

0: the conversion result is left-aligned. ADC_RES saves the high 8 bits of the result and ADC_RESL saves the low 4 bits of the result. The format is as follows:



1: The conversion result is right-aligned. ADC_RES saves the high 4 bits of the result and ADC_RESL saves the low 8 bits of the result. The format is as follows:



SPEED[3:0]: Set ADC clock {FADC=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

19.1.3 ADC conversion result registers (ADC_RES, ADC_RESL)

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

When the A/D conversion is completed, the conversion result is automatically saved to ADC_RES and ADC_RESL. Refer to the RESFMT setting in the ADC_CFG register for the data format of the saved result.

19.1.4 ADC Timing Control Register (ADCTIM)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
ADCTIM	7EFEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				

CSSETUP: ADC Channel Selection Time Control Tsetup

CSSETUP	ADC Clock Number
0	**1 (default)**
1	2

CSHOLD[1:0]: ADC channel selection hold time control Thold

CSHOLD[1:0]	ADC Clock Number
00	1
01	**2 (default)**
10	3
11	4

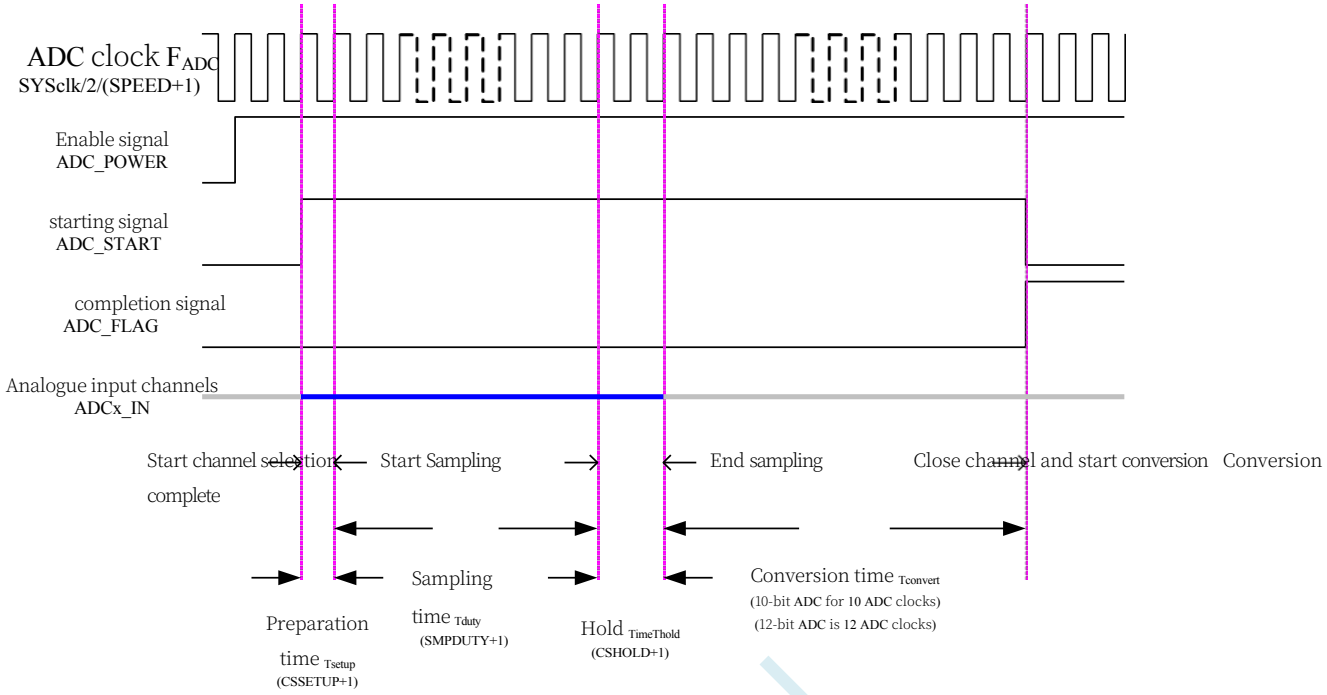
SMPDUTY[4:0]: ADC analogue signal sampling time control Tduty (**Note: SMPDUTY must not be set smaller than 01010B**)

SMPDUTY[4:0]	ADC Clock Number
00000	1
00001	2
...	...
01010	**11 (default)**
...	...
11110	31
11111	32

ADC digital-to-analogue conversion time: $T_{convert}$

12-bit ADC conversion time fixed to 12 ADC operating clocks

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



ADC overall conversion timing diagram

19.2 ADC Static Characteristics

notation	descriptions	minimum value	typical value	maximum values	unit (of measure)
RES	resolution (of a photo)	-	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 non-linear error	-	1	2	LSB
RAIN	Channel Equivalent Resistance	-	∞	-	ohm
RESD	Sample Holding Capacitor in front of the series of anti electrostatic resistance	-	700	-	ohm
CADC	Internal Sample Holding Capacitance	-	16.5	-	pF

19.3 ADC-related formulas

19.3.1 ADC speed formula

The conversion speed of the ADC is controlled by both the SPEED and ADCTIM registers in the ADCCFG register. The formula for calculating the conversion speed is shown below:

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

Attention:

- The speed of the 12-bit ADC cannot be higher than 800KHz.
- The value of SMPDUTY should not be less than 10, it is recommended to set it to 15.
- CSSETUP Power-up defaults can be used 0
- CHOLD can use the power-up default value of 1 (ADCTIM recommends setting to 3FH)

19.3.2 ADC Conversion Result Calculation Formula

$$\text{12-bit ADC conversion result} = 4096 \times \frac{\text{Input voltage of ADC converted channel Vin}}{\text{Voltage of ADC external reference source}} \quad \text{(with separate ADC\_Vref+ pin)}$$

19.3.3 Backpropagation ADC Input Voltage Calculation Formula

$$\text{Vin} = \text{Voltage of the ADC external reference source} \times \times \frac{\text{12-bit ADC conversion results}}{4096}$$

Input voltage of the channel to be converted by the ADC (with separate ADC_Vref+ pin)

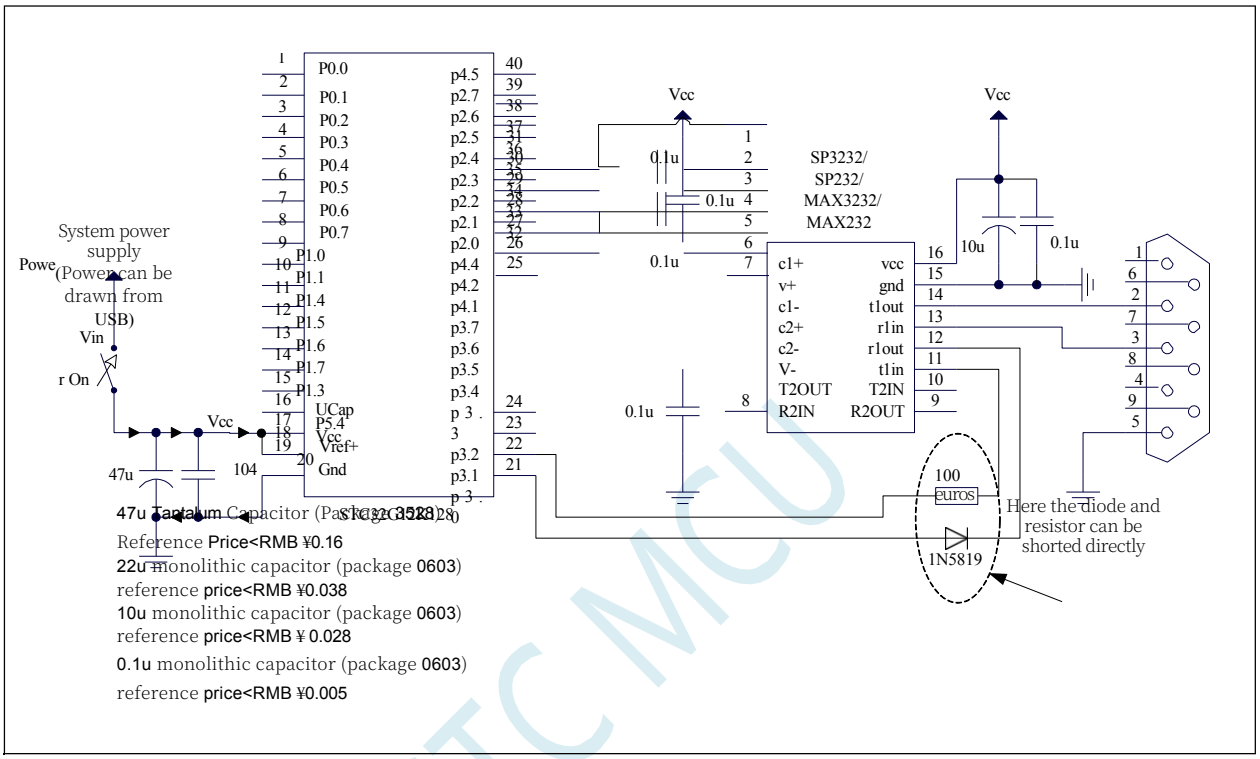
19.3.4 Backpropagation working voltage calculation formula

When it is necessary to use the ADC input voltage and the ADC conversion result to invert the operating voltage, if the target chip does not have an independent ADC_Vref+ pin, then it can be measured directly and use the following formula, if the target chip has an independent ADC_Vref+ pin, then it is necessary to connect the ADC_Vref+ pin to the Vcc pin.

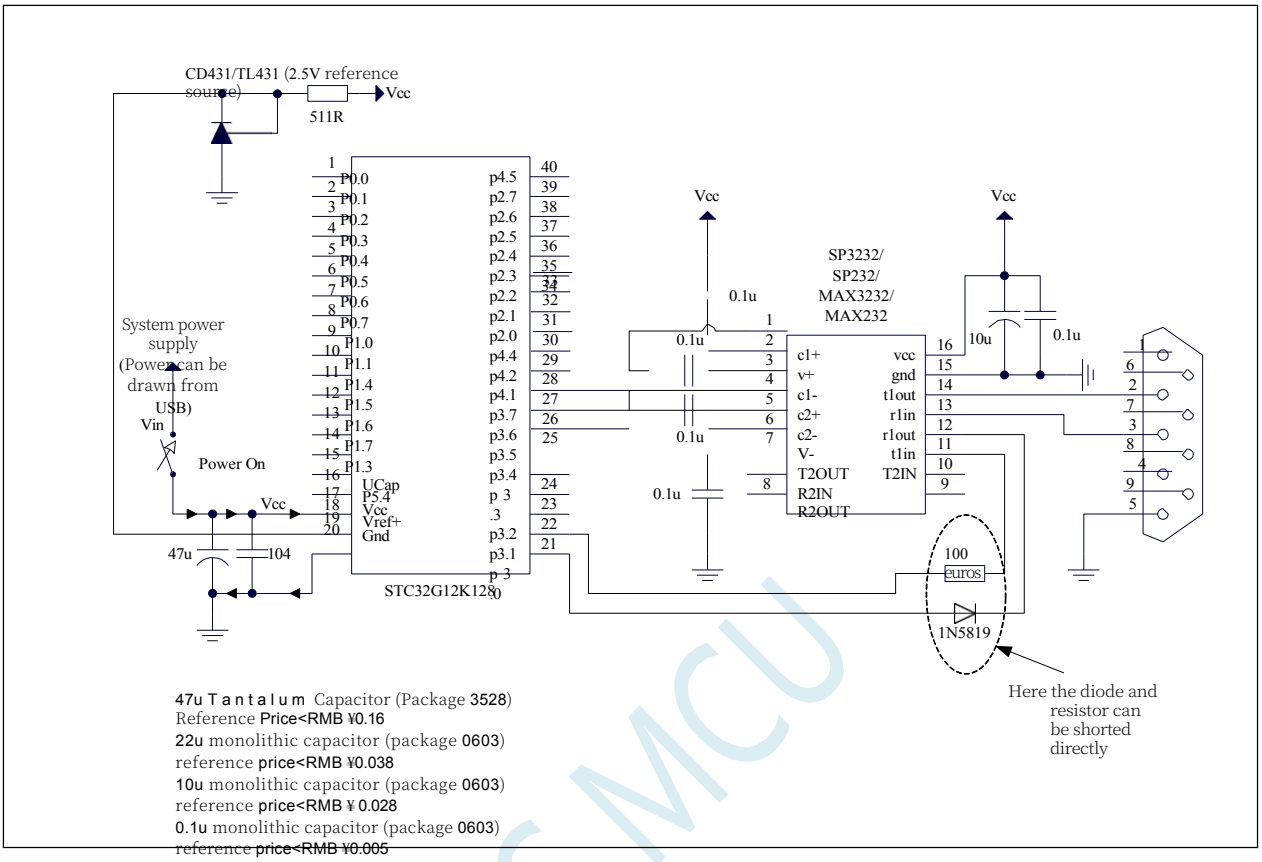
$$\text{MCU Operating Voltage Vcc} = 4096 \times \frac{\text{Input Voltage Vin of ADC Converted Channel 12-bit ADC Conversion Result}}{}$$

19.4 ADC Application Reference Circuit Diagram

19.4.1 General Precision ADC Reference Circuit Diagram



19.4.2 High Precision ADC Reference Circuit Diagram



19.5 sample procedure

19.5.1 ADC basic operation (query method)

//Tested operating frequency is *11.0592MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                    // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                         //Enable access to XFR
    CKCON = 0x00;                      //Set the external data bus speed to fastest
    WTST = 0x00;                       //set the program code wait parameter.
                                       //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    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 port
    P1M1 = 0x01.

    ADCTIM = 0x3f;                                  //set ADC internal timing
    ADCCFG = 0x0f.                                  //set ADC clock to system clock/2/16/16
    ADC_POWER = 1; //Enable the ADC module.         //Enable ADC module

    while (1)
    {
        ADC_START = 1;                              //start AD conversion
        _nop_().
        _nop_().
        while (!ADC_FLAG); //Query the ADC completion flag.  //Query ADC completion flag
        ADC_FLAG = 0;                               //clear completion flag
        P2 = ADC_RES.                               //read ADC result
    }
}
```

19.5.2 ADC basic operation (interrupt mode)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                 // see download software for header files
#include "intrins.h"

void ADC_Isr() interrupt 5
{
    ADC_FLAG = 0;                                   // clear interrupt flag
    P2 = ADC_RES.                                   //read ADC result
    ADC_START = 1; //Continue AD conversion         //Continue AD conversion
}

void main()
{
    EAXFR = 1;                                      //Enable access to XFR
    CKCON = 0x00;                                   //Set the external data bus speed to fastest
    WTST = 0x00;                                    //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
```

psm0 = 0x00;

```
    P5M1 = 0x00.

    p1m0 = 0x00;                    //Set P1.0 as the ADC port.
    p1m1 = 0x01.

    ADCTIM  =  0x3f;                //Set ADC internal timing
    ADCCFG  =  0x0f;                //Set ADC clock to system clock/2/16/16
    ADC_POWER   =                   //Enable ADC Module
    1;                              //Enable ADC interrupt
    EADC = 1;
    EA = 1;                         //Start AD conversion
    ADC_START = 1;

    while (1);
}
```

19.5.3 Formatting ADC Conversion Results

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                 //see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                      //Enable access to XFR
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    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 port
    P1M1 = 0x01.

    ADCTIM = 0x3f;                  //set ADC internal timing
    ADCCFG = 0x0f.                  //set ADC clock to system clock/2/16/16
    ADC_POWER = 1; //Enable the ADC module.  //Enable ADC module
    ADC_START = 1;                  //start AD conversion
    _nop_().
    _nop_().
    while (!ADC_FLAG); //Query the ADC completion flag.  //Query ADC completion flag
    ADC_FLAG = 0;                   //clear completion flag
```

```
    ADCCFG = 0x00;                          //Set the result left-aligned
```

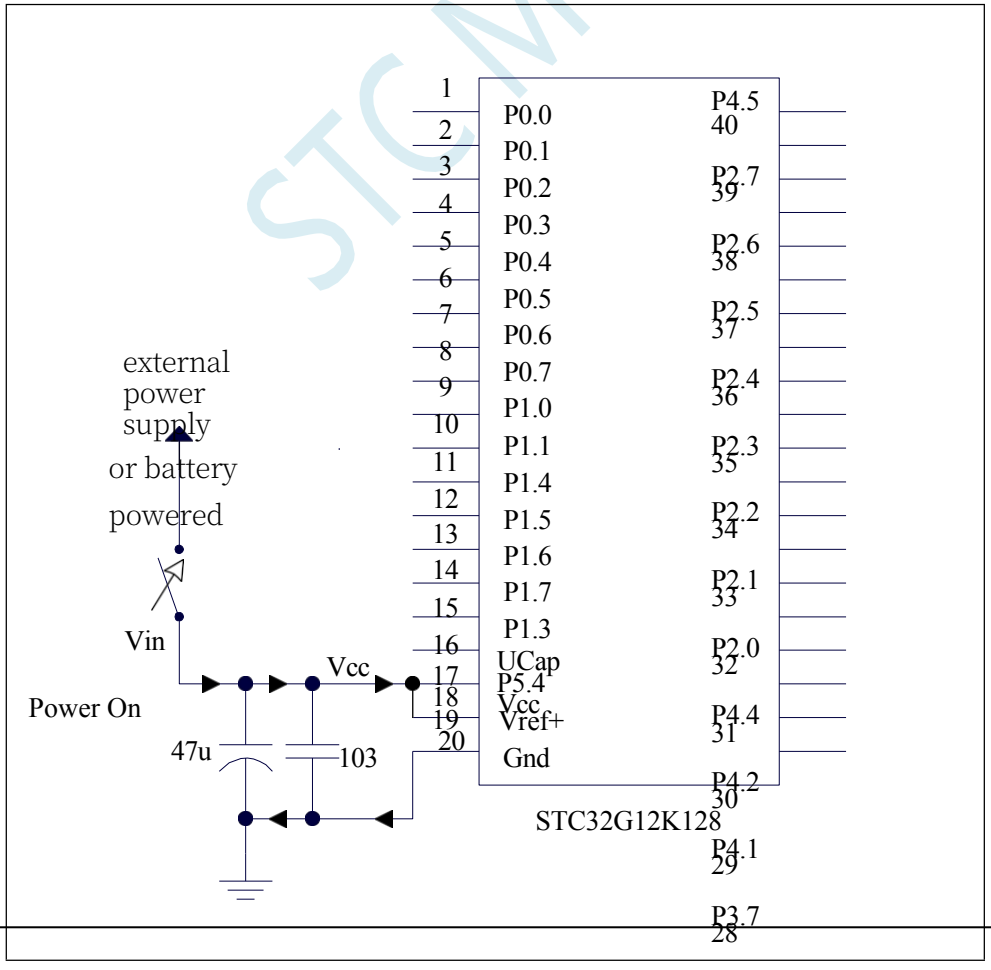
```
    ACC = ADC_RES; // A stores the upper 8 bits of the ADC's 12-bit result.      // A stores the upper 8 bits of the 12-bit result of the ADC.
    B = ADC_RESL; // B[7:4] stores the lower 4 bits of the 12-bit ADC result.    // B[7:4] store the lower 4 bits of the ADC's 12-bit result, B[3:0] are 0.

    //ADCCFG = 0x20.                                  //Set the result right-aligned
    //ACC = ADC_RES.                                  // A[3:0] store the high 4 bits of the 12-bit ADC result, A[7:4] are 0.
    //B = ADC_RESL.                                   //B stores the lower 8 bits of the ADC's 12-bit result

    while (1);
}
```

19.5.4 Measurement of external or battery voltage using ADC channel 15 (internal 1.19V reference source)

Note: The 1.19V here is not the ADC's reference voltage ADC-Vref+, but the fixed input signal source for the ADC15 channel, 1.19V

The 15th channel of the STC32G series ADC is used to measure the internal reference signal source. Since the internal reference signal source is very stable, about 1.19V, and does not change with the operating voltage of the chip, the external voltage or external battery voltage can be inverted by measuring the internal 1.19V reference signal source and then using the value of the ADC.

Below is the reference route map.



```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

#define     FOSC            11059200UL
#define     BRT             (65536 - (FOSC / 115200+2) / 4)
                                        The // plus 2 operation is to allow the Keil compiler to
                                        //Automatic implementation of rounding operations

#define     VREFH_ADDR
CHIPID7 #define
VREFL_ADDR
CHIPID8

bit         busy;
int         BGV.

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
    }
}

void UartInit()
{
    scon = 0x50;
    tmod = 0x00;
    t1x12 = 1; tl1 =
    brt.
    TH1 = BRT >> 8;
    TR1 = 1;
    busy = 0;
}

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

void ADCInit()
{
    ADCTIM = 0x3f;                      //set ADC internal timing

    ADCCFG = 0x2f.                      //Set ADC clock to system clock/2/16
    ADC_CONTR = 0x8f.                   //Enable ADC module and select channel 15.
}

int     ADCRead()
{
```

```
    ADC_START = 1;                                  //start AD conversion
    _nop_().
    _nop_().
    while (!ADC_FLAG); //Query the ADC completion flag.   //Query ADC completion flag
    ADC_FLAG = 0;                                   //clear completion flag

    res = (ADC_RES << 8) | ADC_RESL.                //read ADC result

    return res;
}

void main()
{
    int res;
    int vcc;
    int i.

    EAXFR = 1;                                      //Enable access to XFR
    CKCON = 0x00;                                   //Set the external data bus speed to fastest
    WTST = 0x00;                                    //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.

    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 = (VREFH_ADDR << 8) + VREFL_ADDR; //read the internal reference voltage value from CHIPID.   //read internal
    reference voltage value from CHIPID

    ADCInit();                                      //ADC initialisation
    UartInit();                                     //Initialise the serial port

    ES = 1;
    EA = 1;

//  ADCRead();
//  ADCRead();                                      //The first two data are recommended to be discarded

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

    vcc = (int)(4096L * BGV / res); //(12-bit ADC algorithm) calculate VREF pin voltage, i.e. battery voltage.   //(12-bit ADC
    algorithm) calculates VREF pin voltage, i.e., battery voltage.
                                                    //Note that this voltage is expressed in millivolts (mV).
```

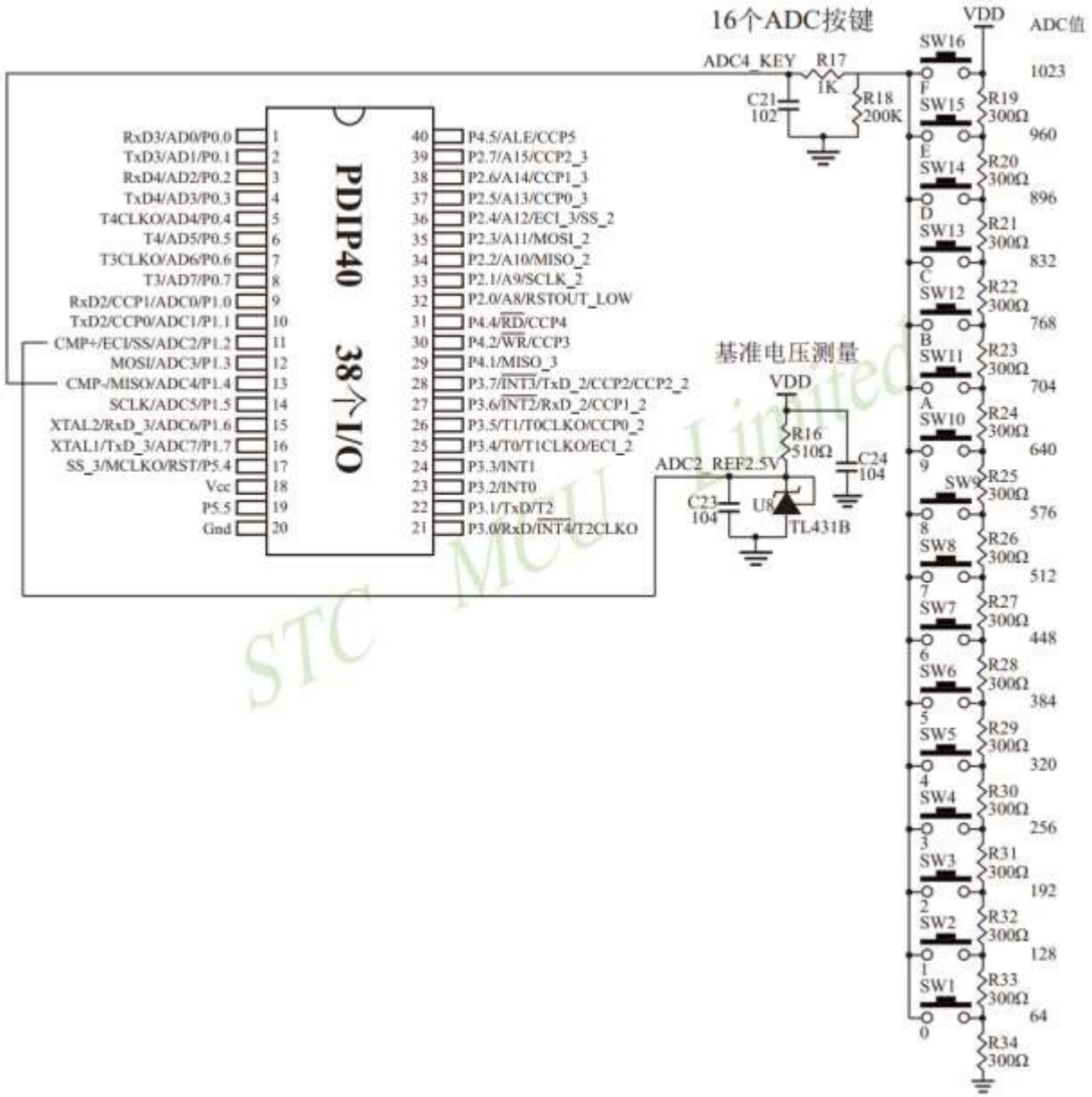
```
    UartSend(vcc >> 8);                    //Output voltage value to serial port
    UartSend(vcc).
```

```
    while (1);
}
```

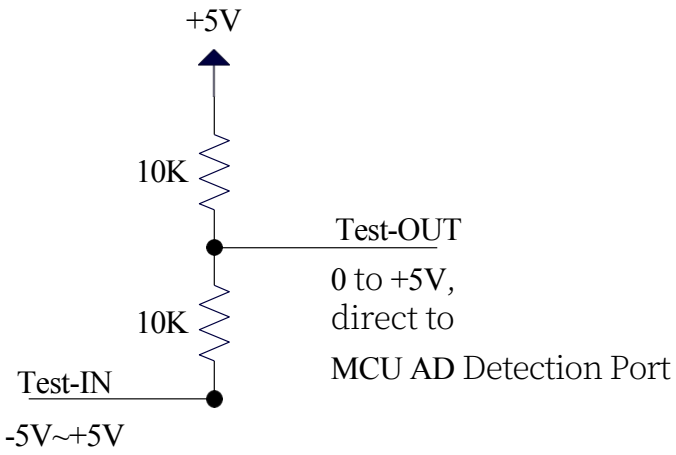
The above method uses the 15th channel of the ADC to invert the external battery voltage. In the ADC measurement range, the external measurement voltage of ADC is proportional to the measurement value of ADC, so it is also possible to use the 15th channel of ADC to inverse the external channel input voltage. Assuming that the voltage of the internal reference signal source is BGV, the ADC measurement value of the internal reference signal source is res_{bg} , and the ADC measurement value of the external channel input voltage is res_x , then the external channel input voltage V = BGV / res * res ; the external channel input voltage V = BGV / res * res ; the external channel input voltage V = BGV / res * res voltage **V_x = BGV / res_{bg} * res_x** ;

19.5.5 ADC for Key Scan Application Circuit Diagram

How to read the ADC key: Read the ADC value every 10ms or so, save the last 3 readings, and judge the key when the change is smaller. When the key is judged to be valid, a certain deviation is allowed, such as ±16 characters.

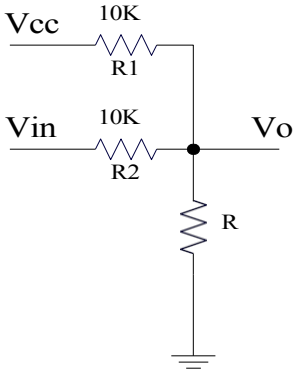


19.5.6 Negative Voltage Detection Reference Circuit Diagram

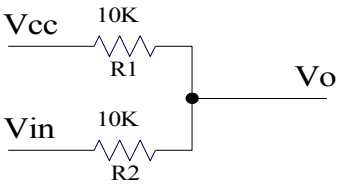


Negative Pressure Conversion Circuit

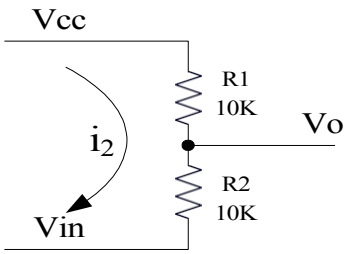
19.5.7 Common Addition Circuits in ADCs



Common Addition Circuits



Simplified Addition Circuit



Transforms into a voltage divider circuit form

Referring to the voltage divider circuit to obtain Equation 1

Equation 1: $Vo = Vin + i_2 * R2$

Equation 2: $i_2 = (Vcc - Vin) / (R1 + R2)$ {condition: current to Vo ≈ 0} .

Substituting R1=R2 into Equation 2 yields Equation 3

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

Substituting equation 3 into equation 1 yields equation 4

Equation 4: $Vo = (Vcc + Vin) / 2$

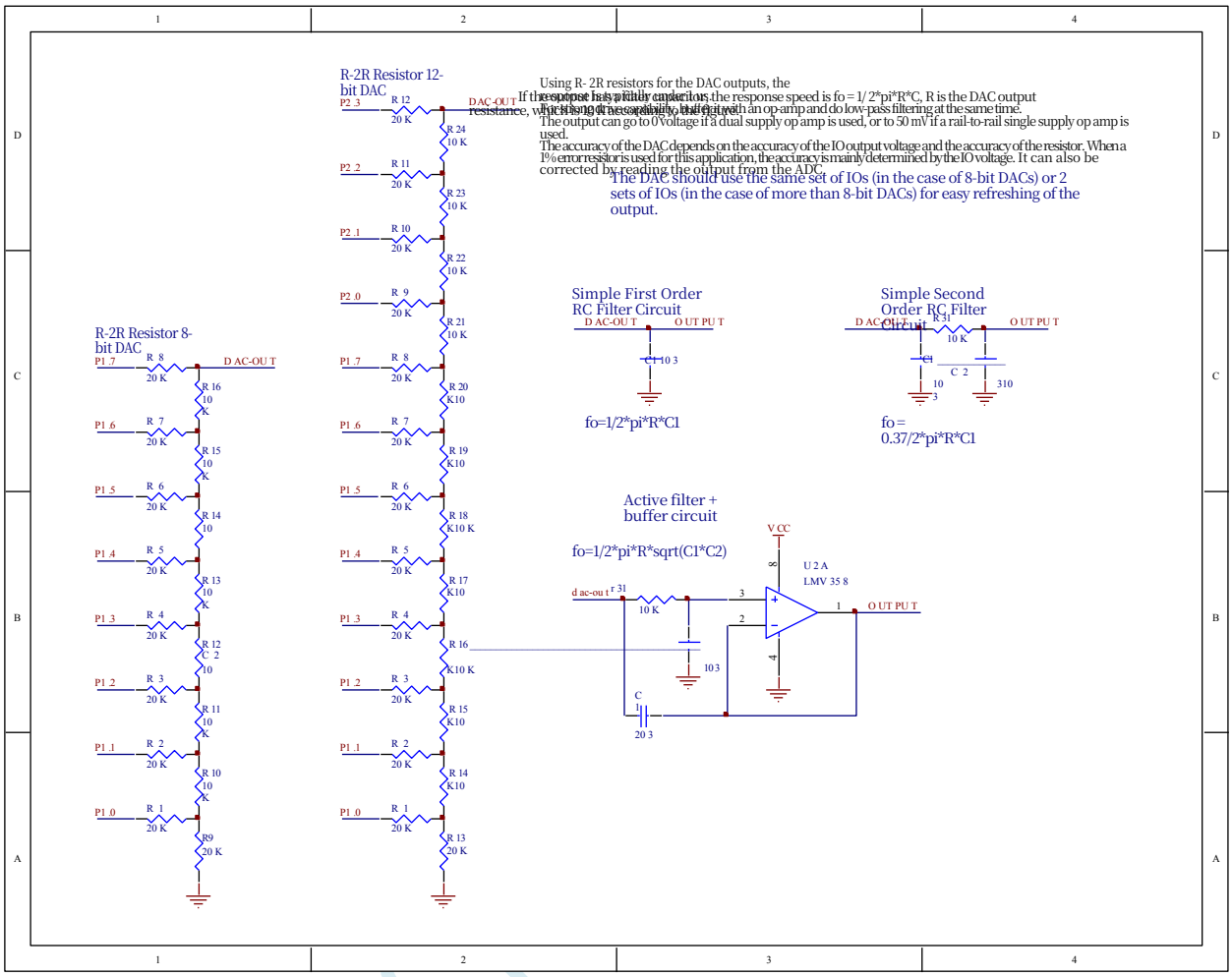
According to Equation 4, the above circuit can be viewed as an addition circuit.

In the analogue-to-digital conversion measurement of 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 raise the measured voltage to more than 0V. In this case, there are certain requirements for the measured voltage range:

Substituting the above conditions into Equation 4, the following 2 equations can be obtained (Vcc + Vin) / 2 > 0 i.e. Vin > -Vcc (Vcc + Vin) / 2 < Vcc i.e. Vin < Vcc)

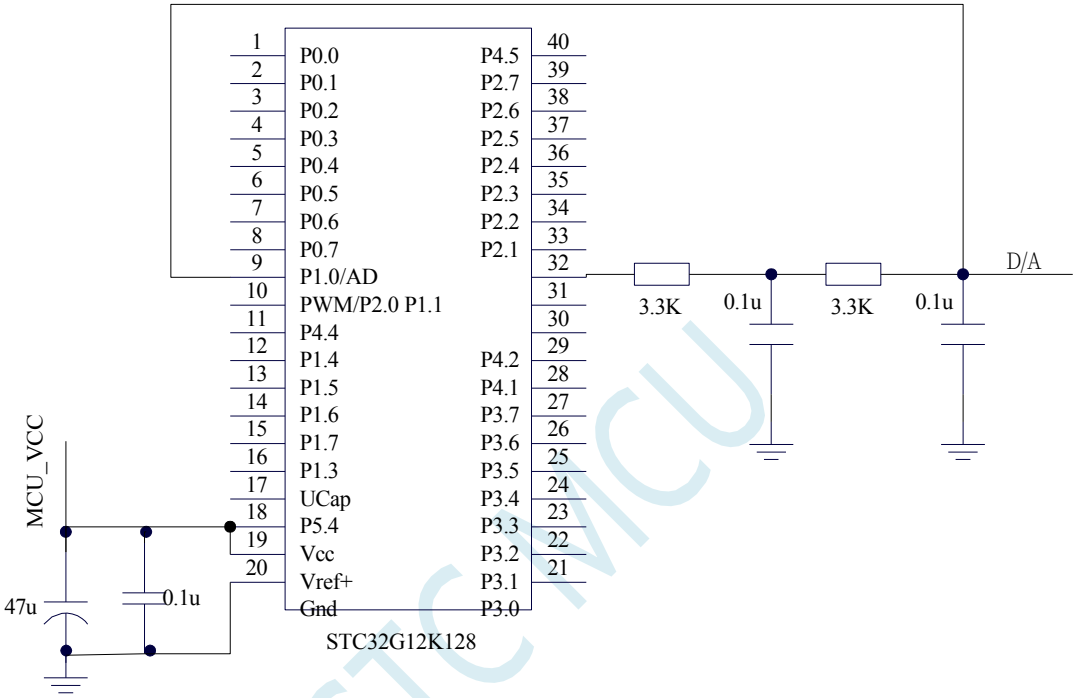
The above 2 equations can be combined: **-Vcc < Vin < Vcc**

19.6 Classic Circuit Diagram of a DAC Using I/O and R-2R



19.7 Reference Circuit Diagram for 16-bit DAC using PWM

The advanced PWM timer of STC32G series MCU can output a 16-bit PWM waveform, and then generate a 16-bit DAC signal after two stages of low-pass filtering, and the DAC signal can be changed by adjusting the duty cycle of the high level of the PWM waveform. The application circuit diagram is shown below, and the output DAC signal can be input to the ADC of MCU for feedback measurement.



20 Synchronous Serial Peripheral Interface (SPI)

The STC32G series microcontrollers incorporate a high-speed serial communication interface, the SPI interface, which is a full-duplex, high-speed, synchronous communication bus.The integrated SPI interface of the STC32G series provides two modes of operation: master mode and slave mode.

Note: The SPI modes of USART1 and USART2 can support two full SPI groups, please refer to the USART chapter for details.

20.1 SPI Function Pin Switching

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CAN_S[1:0]		**SPI_S[1:0]**		LIN_S[1:0]	

SPI_S[1:0]: SPI function pin select bits

SPI_S[1:0]	SS	MOSI	MISO	SCLK
00	P1.2/P5.4[1]	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P3.5	P3.4	P3.3	P3.2

[1] ：有 P1.2 port, this function is on theP1.2 port, and for models without theP1.2 port, this function is on the P5.4 口上

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW3	BBH	I2S_S		S2SPI_S[1:0]		S1SPI_S[1:0]		CAN2_S[1:0]	

S2SPI_S[1:0]: SPI function pin select bits for USART2

S2SPI_S[1:0]	S2SS	S2MOSI	S2MISO	S2SCLK
00	P1.2/P5.4[1]	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P7.4	P7.5	P7.6	P7.7

[1] ：有 P1.2 port, this function is on theP1.2 port, and for models without theP1.2 port, this function is on the P5.4 口上

S1SPI_S[1:0]: SPI function pin select bit for USART1

S1SPI_S[1:0]	S1SS	S1MOSI	S1MISO	S1SCLK
00	P1.2/P5.4[1]	P1.3	P1.4	P1.5
01	P2.2	P2.3	P2.4	P2.5
10	P5.4	P4.0	P4.1	P4.3
11	P6.4	P6.5	P6.6	P6.7

[1] ：有 P1.2 port, this function is on theP1.2 port, and for models without theP1.2 port, this function is on the P5.4 口上

20.2 SPI Related Registers

notation	descriptions	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)

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

SPIF: SPI interrupt flag bit.

When 1 byte of data has been transmitted/received, this bit is automatically set to 1 by the hardware and an interrupt request is made to the CPU. When the SSIG bit is set to 0, and the master/slave mode of the device changes due to a change in the level of the SS pin, this flag bit is also automatically set to 1 by the hardware to signify a change in the device mode.

Note: This flag bit must be cleared by the user by writing a 1 to this bit in software.

WCOL: SPI Write Conflict Flag bit.

When SPI writes the SPDAT register during a data transfer, hardware sets this bit to 1. Note: This flag bit must be cleared by the user by writing 1 to this bit in software.

20.2.2 SPI Control Register (SPCTL), SPI Speed Control

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

SSIG: SS pin function control bit

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

1: Ignore the SS pin function and use MSTR to determine if the device is a master or a slave

SPEN: SPI enable control bit

0: Disable SPI function

1: Enable SPI function

DORD: SPI Data Bit Transmit/Receive Sequence

0: Send/receive data high bit first (MSB)

1: transmit/receive data low bit first (LSB) MSTR: device master/slave mode selection bit

Set the host mode:

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

If SSIG = 1, only set MSTR to 1 (ignoring the level of the SS pin) to set the slave mode:

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

If SSIG = 1, you only need to set MSTR to 0 (ignoring the level of the SS pin)

CPOL: SPI Clock Polarity Control

0: SCLK is low when idle, the front clock edge of SCLK is rising edge, the back clock edge is falling edge

1: SCLK is high when idle, the front clock edge of SCLK is falling edge, the back clock edge is rising edge

CPHA: SPI Clock Phase Control

0: Data SS pin goes low to drive the first bit of data and change the data on the back clock edge of SCLK and sample the data on the front clock edge (must have SSIG = 0)

1: Data is driven on the front clock edge of SCLK and sampled on the back clock edge

SPR[1:0]: SPI clock frequency selection

SPR[1:0]	SCLK Frequency
00	SPI Input Clock/4

01	SPI Input Clock/8
10	SPI Input Clock/16
11	SPI Input Clock/2

20.2.3 SPI Data Register (SPDAT)

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

SPI transmit/receive data buffer.

20.3 SPI communication method

There are usually three types of SPI communication methods: single-master-single-slave (a host device connects to a slave device), mutual-master-slave (two devices are connected, and the devices and each other are the master and the slave), and single-master-multi-slave (a host device connects to multiple slave devices).

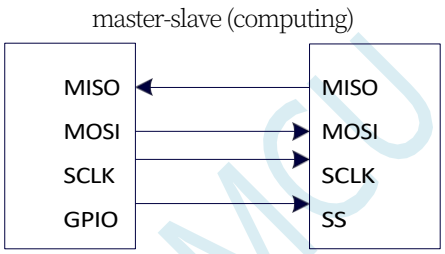
20.3.1 single master single slave (computing)

Two devices are connected, with one device fixed as the master and the other fixed as the slave.

Host Setting: SSIG is set to 1, MSTR is set to 1, fixed in host mode. The master can use any port to connect to the SS pin of the slave, and pull down the SS pin of the slave to enable the slave.

Slave Setup: SSIG is set to 0, and the SS pin is used as the chip select signal for the slave.

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



Single Master Single Slave Configuration

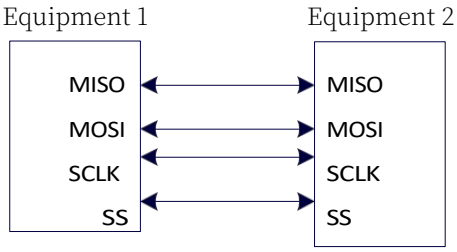
20.3.2 mutually oriented

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

Setting method 1: Both devices are initialised with SSIG set to 0, MSTR set to 1, and SS pin set to output high in bidirectional port mode. At this time, both devices are in host mode without ignoring SS. When one of the devices needs to start transmission, it can set its own SS pin to output mode and output low, pulling down the other's SS pin, so that the other device is forced to set to slave mode.

Setting method 2: When both devices are initialised, set themselves to the slave mode which ignores SS, i.e. SSIG is set to 1 and MSTR is set to 0. When one of the devices needs to start the transmission, it will first detect the level of the SS pin, and if it is high, it will set itself to the master mode which ignores SS, and then it will be able to transmit the data.

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



Mutual master-slave configuration

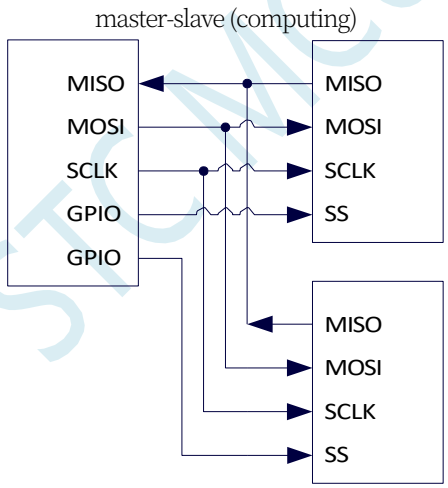
20.3.3 single master multi-slave (computing)

Multiple devices are connected, with one device fixed as a master and the others fixed as slaves.

Host Setting: SSIG is set to 1, MSTR is set to 1, and fixed to host mode. The master can use any port to connect the SS pins of each slave, and pull down the SS pin of one of the slaves to enable the corresponding slave device.

Slave Setup: SSIG is set to 0, and the SS pin is used as the chip select signal for the slave.

The single master multi-slave connection configuration diagram is shown below:



Single Master Multi-Slave Configuration

20.4 Configuring SPI

control bits			communications port				instructions
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	import ation	import ation	import ation	Disable SPI function, SS/MOSI/MISO/SCLK are all normal IOs.
1	0	0	0	exports	import ation	import ation	**Slave mode**, and is selected
1	0	0	1	high resista nce	import ation	import ation	**Slave mode**, but unchecked
1	0	1→0	0	exports	import ation	import ation	**Slave mode**, host mode that does not ignore SS and MSTR is 1. When the SS pin is pulled low, MSTR will be automatically cleared by hardware. The operating mode will be passively set to slave mode
1	0	1	1	impo rtatio n	high resista nce	high resista nce	**Host Mode**, Idle
					exports	exports	**Host mode**, active
1	1	0	x	exports	import ation	import ation	**slave mode**
1	1	1	x	import ation	exports	exports	**Host Mode**

Notes on slave mode:

When CPHA = 0, SSIG must be 0 (i.e., the SS pin cannot be ignored). SS pin must be pulled low before each serial byte is started and reset high after the serial byte has been sent. no write operation can be performed to the SPDAT register while SS pin is low, or a write conflict error will result. operation is undefined when CPHA = 0 and SSIG = 1.

When CPHA = 1, SSIG can be set to 1 (i.e., the pin can be ignored). If SSIG = 0, the SS pin can be held active low (i.e., it is always fixed low) between successive transmissions. This approach is suitable for systems with a fixed single master and single slave.

Host Mode Notes:

In SPI, transfers are always initiated by the host. If SPI is enabled (SPEN=1) and selected as the host, a host write operation to the SPI data register SPDAT initiates the SPI clock generator and the transfer of data. The data will appear on the MOSI pin one-half to one SPI bit time after the data is written to SPDAT. The data written to the host SPDAT register is shifted from the MOSI pin to the slave's MOSI pin. At the same time, data from the slave SPDAT register is shifted from the MISO pin and sent to the MISO pin of the host.

After transferring a byte, the SPI clock generator stops, the transfer completion flag (SPIF) is set, and an SPI interrupt is generated if the SPI interrupt is enabled. The two shift registers of the host and slave CPUs can be regarded as a 16-bit circular shift register. As data is shifted

from the host to the slave, it is also shifted in the opposite direction. This means that in one shift cycle, the data of the host and the slave are exchanged with each other.

Change of mode via SS

If SPEN=1, SSIG=0 and MSTR=1, SPI is enabled for host mode and the SS pin can be configured for input modalisation or quasi-bidirectional port mode. In this case, another host can drive the pin low, which selects the device as an SPI slave and sends data to it. To avoid bus contention, the SPI system clears the slave's MSTR to zero, forces MOSI and SCLK to input mode, and MISO to output mode, with SPSTAT's SPIF flag at position 1.

The user software must always detect the MSTR bit, if this bit is passively cleared by a slave select action and the user wants to continue to use the SPI as a host, the MSTR bit must be reset or it will remain in slave mode.

Write conflicts

SPI is single buffered when sending and double buffered when receiving. This prevents new data from being written to the shift registers until the previous transmission has been completed. When a write operation is performed to the data register SPDAT during transmission, the WCOL bit is set to 1 to indicate that a data write conflict error has occurred. In this case, the currently sent data continues to be sent and the newly written data will be lost.

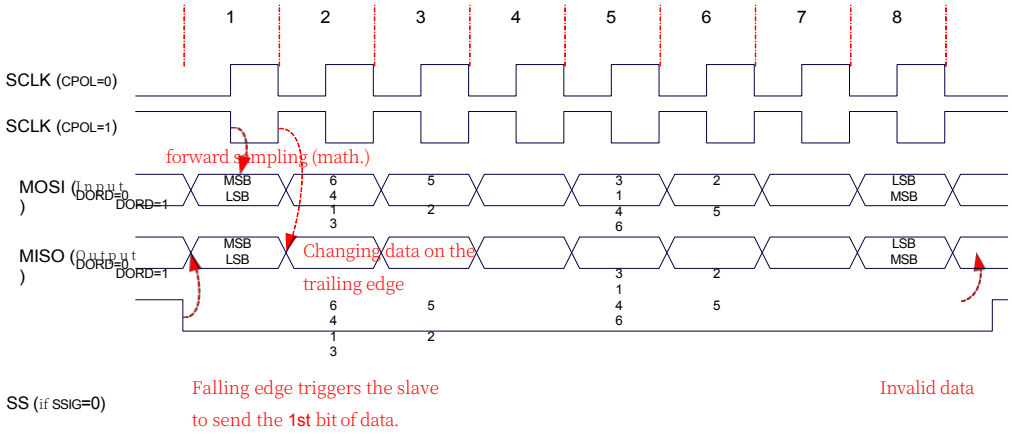
When write conflict detection is performed on either the master or the slave, it is rare for the master to have a write conflict because the master has full control of the data transfer. However, it is possible for a slave to have a write conflict because the slave has no control when the master initiates a 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 fully shifted in, otherwise the previous received data will be lost.

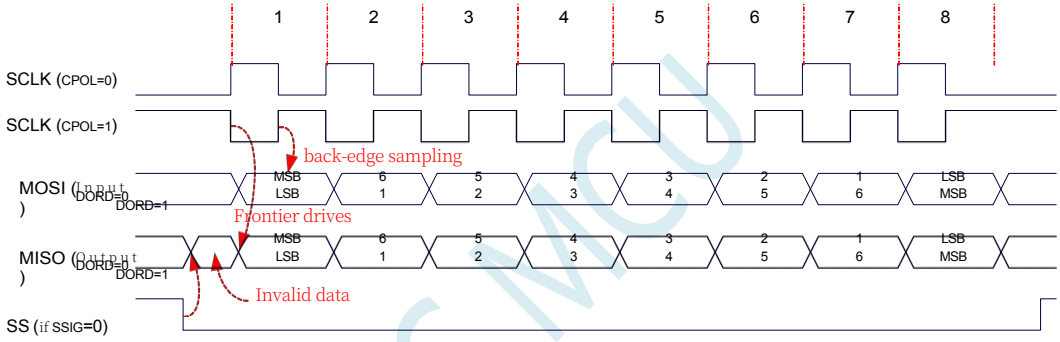
The WCOL can be cleared by software by writing a "1" to it.

20.5 data model

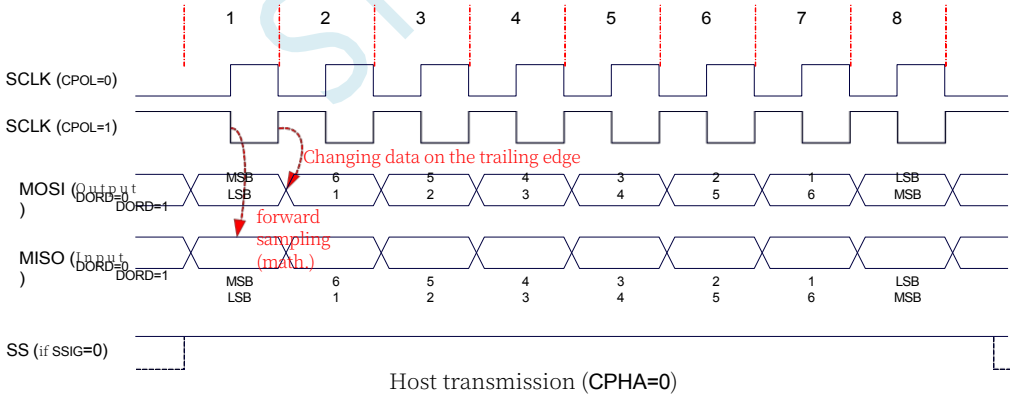
The SPI clock phase control bit CPHA allows the user to set the clock edge when data is sampled and changed. The clock polarity bit CPOL allows the user to set the clock polarity. The following diagram shows the SPI communication timing with different clock phase and polarity settings.



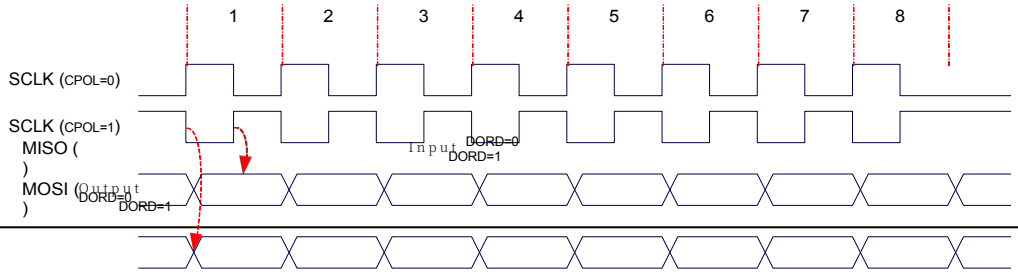
Slave transmission (CPHA=0)

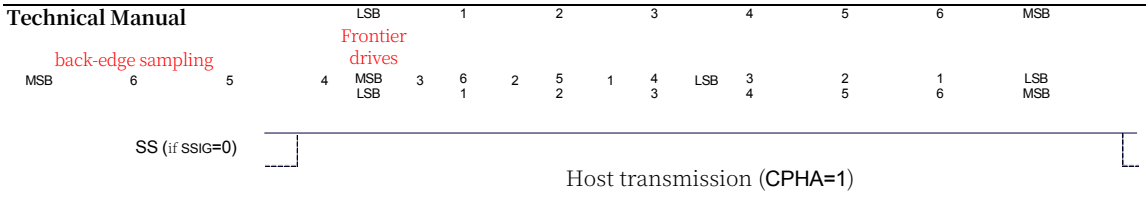


Slave transmission (CPHA=1)



Host transmission (CPHA=0)




LSB 1 2 3 4 5 6 MSB
Frontier drives
back-edge sampling
MSB 6 5 4 MSB 3 6 2 5 1 4 LSB 3 2 1 LSB
LSB 1 2 3 4 5 6 MSB
SS (if SSIG=0)
Host transmission (CPHA=1)

20.6 sample procedure

20.6.1 SPI single-master, single-slave system host program (interrupt mode)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include  "stc32g.h"                    // See download software for header files
#include "intrins.h"

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

bit         busy;

void SPI_Isr() interrupt 9
{
    SPIF = 1;                           //clear interrupt flag
    SS = 1;                             //pull up the slave's SS pin
    busy = 0;
    LED = !LED;                         //test port
}

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

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

    SPCTL = 0x50;                       //Enable SPI host mode
    SPSTAT = 0xc0.                      //clear interrupt flag
    ESPI = 1; //Enable SPI interrupt.   //Enable SPI interrupt
    EA = 1;
```

```
    while (1)
    {
        while (busy);
```

```
            busy = 1;
            SS = 0;                                 //pull down slave SS pin
            SPDAT = 0x5a;                           //Send test data
        }
}
```

20.6.2 SPI single-master-single-slave system slave programme (interrupt mode)

```
//Tested operating
frequency is 11.0592MHz

//#include "stc8h.h"
#include  "stc32g.h"                                //See download software for header files
#include "intrins.h"

sbit        LED             =     P1^1.

void SPI_Isr() interrupt 9
{
        SPIF = 1;                                   //clear interrupt flag
        SPDAT = SPDAT;                              //pass the received data back to the host computer
        LED = !LED;                                 //test port
}

void main()
{
        EAXFR = 1;                                  //Enable access to XFR
        CKCON = 0x00;                               //Set the external data bus speed to fastest
        WTST = 0x00;                                //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.


        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
        ESPI = 1; //Enable SPI interrupt.           //Enable SPI interrupt
        EA = 1;

        while (1);
}
```

20.6.3 SPI single-master-single-slave system host programme (query method)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include  "stc32g.h"                    //See download software for header files
#include "intrins.h"

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

void main()
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    LED = 1;
    SS = 1;

    SPCTL = 0x50;                       //Enable SPI host mode
    SPSTAT = 0xc0.                      //clear interrupt flag

    while (1)
    {
        SS = 0;                         //pull down slave SS pin
        SPDAT = 0x5a;                   //Send test data
        while (!SPIF);                  //Query completion flag
        SPIF = 1;                       //clear interrupt flag
        SS = 1;                         //pull up the slave's SS pin
        LED = !LED;                     //test port
    }
}
```

20.6.4 SPI single-master-single-slave system slave

programme (query method)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
```

```
#include "stc32g.h"                          //see download software for header files
#include "intrins.h"

sbit        LED             LED=

P1^1; void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    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 (!SPIF);                       //Query completion flag
        SPIF = 1;                            //clear interrupt flag
        SPDAT = SPDAT;                       //pass the received data back to the host computer
        LED = !LED;                          //test port
    }
}
```

20.6.5 SPI Mutual Master-Slave System Program (Interrupt Mode)

```
//Tested operating
frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          //See download software for header files
#include "intrins.h"

sbit        SS              =   p1^0;
sbit        LED             =   p1^1;
sbit        KEY             =   p0^0.

void SPI_Isr() interrupt 9
{
```

```
    SPIF = 1;                       //clear interrupt flag
    if (SPCTL & 0x10)
    {                               //Host mode
        SS = 1;                     //pull up the slave's SS pin
```

```
            SPCTL = 0x40;                           //reset to slave standby
        }
        else
        {                                           //Slave mode
            SPDAT = SPDAT;                          //pass the received data back to the host computer
        }
        LED = !LED;                                 //test port
}

void main()
{
        EAXFR = 1;      //Enable access to XFR
        CKCON = 0x00;                               //Set the external data bus speed to fastest
        WTST = 0x00;                                //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.

        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 for standby.
        SPSTAT = 0xc0.                              //clear interrupt flag
        ESPI = 1; //Enable SPI interrupt.           //Enable SPI interrupt
        EA = 1;

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

20.6.6 SPI Mutual Master-Slave Procedure (Query Method)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
```

```
#include "stc32g.h"                                    //see download software for header files
#include "intrins.h"

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

void main()
{
    EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                                      //Set the external data bus speed to fastest
    WTST = 0x00;                                       //set the program code wait parameter.
                                                       //Assign a value of 0 to set the CPU to execute the
                                                       programme as fast as possible.

    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 for standby.
    SPSTAT = 0xc0.                                     //clear interrupt flag

    while (1)
    {
        if (!KEY)                                      //Wait for the key to be triggered
        {
            SPCTL = 0x50;                              //Enable SPI host mode
            SS = 0;                                    //pull down slave SS pin
            SPDAT = 0x5a;                              //Send test data
            while (!KEY);                              //Wait for the key to be released
        }
        if (SPIF)
        {
            SPIF = 1;                                  //clear interrupt flag
            if (SPCTL & 0x10)
            {                                          //Host mode
                SS = 1;                                //pull up the slave's SS pin
                SPCTL = 0x40;                          //reset to slave standby
            }
            else
            {                                          //Slave mode
                SPDAT = SPDAT;                         //pass the received data back to the host computer
            }
            LED = !LED;                                //test port
        }
    }
}
```

21 High Speed SPI (HSSPI)

The STC32G series microcontrollers provide a high-speed mode (HSPSI) for SPI. High-speed SPI is based on normal SPI with the addition of high-speed mode.

When the system is running at a lower operating frequency, the high-speed SPI can operate at up to 144M. This reduces core power consumption and improves peripheral performance.

21.1 Related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
HSSPI_CFG	High Speed SPI Configuration Register	7EFBF8H	SS_HLD[3:0]				SS_SETUP[3:0]				0011,0011
HSSPI_CFG2	High Speed SPI Configuration Register 2	7EFBF9H	-	-	HSSPIEN	FIFOEN	SS_DACT[3:0]				xx00,0011
HSSPI_STA	High Speed SPI Status Register	7EFBFAH	-	-	-	-	TXFULL	TXEMPT	RXFULL	RXEMPT	xxxx,0000

21.1.1 High Speed SPI Configuration Register (HSSPI_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_CFG	7EFBF8H	SS_HLD[3:0]				SS_SETUP[3:0]			

SS_HLD[3:0]: HOLD time of SS control signal in high speed mode

SS_SETUP[3:0]: SETUP time of SS control signal in high speed mode

21.1.2 High Speed SPI Configuration Register 2 (HSSPI_CFG2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_CFG2	7EFBF9H	-	-	HSSPIEN	FIFOEN	SS_DACT[3:0]			

HSSPIEN: High Speed SPI Enable Bit

0: Switch off high-speed mode.

1: Enable high speed mode.

When the SPI speed is system clock/2 (SPCTL.SPR = 3), the enable high-speed mode must be set.

FIFOEN: FIFO mode enable bit of high-speed SPI

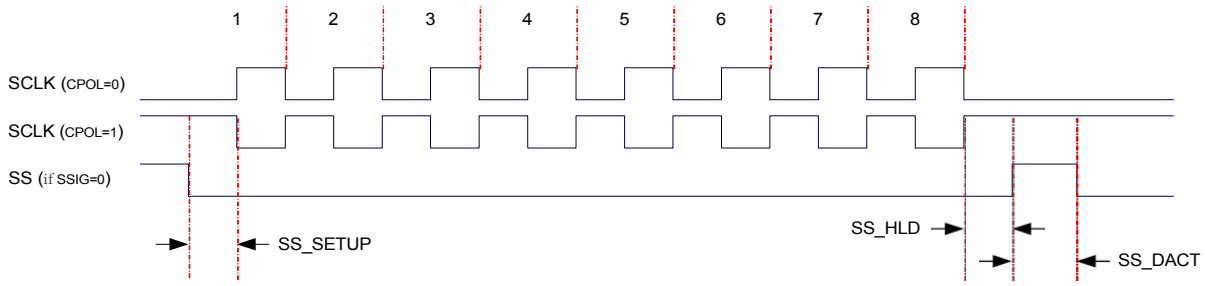
0: Disable FIFO mode.

1: Enable FIFO mode.

Meaningful only when SPI performs DMA operations, not meaningful in normal SPI mode SPI sends and receives with a FIFO depth of 4 Bytes.

SS_DACT[3:0]: DEACTIVE time of SS control signal in high speed mode

Note: The values set by SS_HLD, SS_SETUP, and SS_DACT are only meaningful for DMA in SPI host mode, and the automatic hardware output of the SS control signal is only required when SPI is in host mode and performing DMA data transfers. When the SPI speed is system clock/2 (SPCTL.SPR = 3) to perform DMA, SS_HLD, SS_SETUP, and SS_DACT must all be set to a value greater than 2.



21.1.3 High Speed SPI Status Register (HSSPI_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HSSPI_STA	7EFBFAH	-	-	-	-	TXFULL	TXEMPT	RXFULL	RXEMPT

TXFULL: send data FIFO full flag (read only) TXEMPT: send data FIFO empty flag (read only) RXFULL: receive data FIFO full flag (read only) RXEMPT: receive data FIFO empty flag (read only)

21.2 sample procedure

21.2.1 Enable SPI high-speed mode

```
//Test operating frequency is 12MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

#define     FOSC        12000000UL

#define     HSCK_MCLK       0
#define     HSCK_PLL        1
#define     HSCK_SEL        HSCK_PLL

#define     PLL_96M         0
#define     PLL_144M        1
#define     PLL_SEL         PLL_96M

#define     CKMS            0x80
#define     HSIOCK          0x40
#define     MCK2SEL_MSK     0x0c
#define     MCK2SEL_SEL1    0x00
#define     MCK2SEL_PLL     0x04
#define     MCK2SEL_PLLD2   0x08
#define     MCK2SEL_IRC48   0x0c
#define     MCKSEL_MSK      0x03
#define     MCKSEL_HIRC     0x00
#define     MCKSEL_XOSC     0x01
#define     MCKSEL_X32K     0x02
#define     MCKSEL_IRC32K   0x03

#define     ENCKM           0x80
#define     PCKI_MSK        0x60
#define     PCKI_D1         0x00
#define     PCKI_D2         0x20
#define     PCKI_D4         0x40
#define     PCKI_D8         0x60

void delay()
{
    int     i;

    for ( i=0; i<100; i++);
}

void main()
{
    EAXFR = 1;                                         //Enable access to XFR
    CKCON = 0x00; //Set the external data bus speed to fastest
    WTST = 0x00.

    //Select PLL output clock
```

```
#if   (PLL_SEL == PLL_96M)
    CLKSEL &= ~CKMS.                                    //Select 96M of PLL as PLL output clock.
#elif (PLL_SEL == PLL_144M)
    CLKSEL |= CKMS.                                     //Select 144M of PLL as the output clock of PLL.
#else
    CLKSEL &= ~CKMS.                                    //Default is to select 96M of PLL as the output clock of
    PLL.
#endif

    //Select PLL input clock divider to ensure
    input clock is 12M USBCLK &= ~PCKI_MSK.
#if   (FOSC == 12000000UL)
    USBCLK |= PCKI_D1.                                  //PLL input clock 1 division
#elif (FOSC == 24000000UL)
    USBCLK |= PCKI_D2.                                  //PLL input clock 2 divisions
#elif (FOSC == 48000000UL)
    USBCLK |= PCKI_D4.                                  //PLL input clock 4 divisions
#elif (FOSC == 96000000UL)
    USBCLK |= PCKI_D8.                                  //PLL input clock 8 divisions
#else
    USBCLK |= PCKI_D1.                                  //Default PLL input clock 1 division
#endif

    //Start PLL
    USBCLK |= ENCKM.                                    //Enable PLL multiplier

    delay();                                            //Wait for PLL to lock

    //Select HSPWM/HSSPI Clock
#if   (HSCK_SEL == HSCK_MCLK)
    CLKSEL &= ~HSIOCK.                                  //HSPWM/HSSPI selects master clock as clock source
#elif (HSCK_SEL == HSCK_PLL)
    CLKSEL |= HSIOCK.                                   //HSPWM/HSSPI Select PLL output clock as clock source
#else
    CLKSEL &= ~HSIOCK.                                  //Default HSPWM/HSSPI Selects master clock as clock
    source
#endif

    HSCLKDIV = 0; //HSPWM/HSSPI clock sources are not divided.      //HSPWM/HSSPI Clock source not divided by
    frequency

    SPCTL = 0xd0.                                       //Set SPI to host mode, speed is SPI clock /4
    HSSPI_CFG2 |= 0x20; //Enable SPI high-speed mode.  //Enable SPI high speed mode

    p1m0 = 0x28;
    p1m1 = 0x00.

    while (1)
    {
        SPSTAT = 0xc0;
        SPDAT = 0x5a.
        while (!SPIF);
    }
}
```

22 I2C bus

The STC32G series of microcontrollers incorporate an internal I² C serial bus controller. I² C is a high-speed synchronous communication bus that communicates synchronously using two lines, SCL (clock line) and SDA (data line). For the port assignment of SCL and SDA, the STC32G series of microcontrollers provide a switching mode to switch SCL and SDA to different I/O ports to facilitate users to treat one group of I² C buses as multiple groups for time-sharing multiplexing.

Compared to the standard I² C protocol, the following two mechanisms are ignored:

- No arbitration after sending the start signal (START)
- No timeout detection when clock signal (SCL) stays low

The I² C bus of the STC32G series provides two modes of operation: host mode (SCL is the output port and sends the synchronised clock signal) and slave mode (SCL is the input port and receives the synchronised clock signal)

22.1 I2C Function Pin Switching

notation	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]: I² C Function Pin Select Bits

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

22.2 I2C Related Registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I² C Configuration Register	7EFE80H	ENI2C	MSSL	MSSPEED [5:0]						0000,0000
I2CMSCR	I² C Host Control Register	7EFE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I² C Host Status Register	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx10
I2CSLCR	I² C Slave Control Registers	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I² C Slave Status Register	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I² C Slave Address Registers	7EFE85H	SLADR [6:0]							MA	0000,0000
I2CTXD	I² C Data Transmission Register	7EFE86H									0000,0000
I2CRXD	I² C Data Receiving Register	7EFE87H									0000,0000
I2CMSAUX	I² C Host Auxiliary Control Registers	7EFE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

22.3 I2C host mode

22.3.1 I2C Configuration Register (I2CCFG), Bus Speed Control

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	7EFE80H	ENI2C	MSSL	MSSPEED [5:0]					

ENI2C: I² C Function Enable Control Bit

0: Disable I² C function

1: Allow I² C function

MSSL: I² C Operating Mode Selection Bit

0: Slave mode

1: Host mode

MSSPEED[5:0]: I² C bus speed (number of wait clocks) control, **I2C bus speed = SYSCLK / 2 / (MSSPEED * 2 + 4)**

(I2C fastest speed is SYSCLK/8)

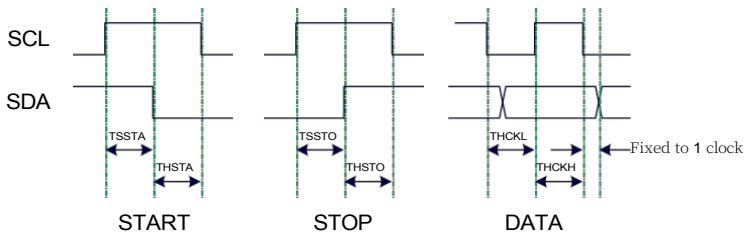
MSSPEED [5:0]	Corresponding number of clocks
0	4
1	6
2	8
...	...
x	2x+4
...	...
62	128
63	130

The wait parameter set by the MSSPEED parameter is only valid when the I² C module is operating in host mode. This wait parameter is mainly used for the following signals in host 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 T : Hold Time of SCL Low)



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

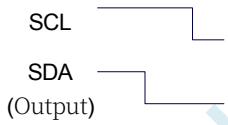
Example 2: When an I2C bus speed of 400K is required at an operating frequency of 24MHz.

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

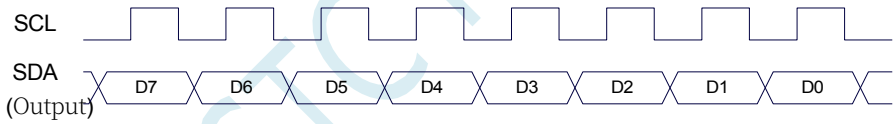
22.3.2 I2C Host Control Register (I2CMSCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	7EFE81H	EMSI	-	-	-	MSCMD[3:0]			

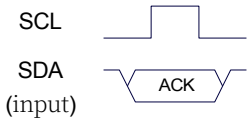
EMSI: host mode interrupt
enable control bit 0:
disable host mode
interrupts
1: Allow host mode
interrupt MSCMD[3:0]:
host command
0000: Standby, no action.
0001: Starting command.
Send the START signal. If the current I² C controller is in idle state, that is, when MSBUSY (I2CMSST.7) is 0, writing this command will make the controller enter busy state, the hardware will automatically set the MSBUSY state to position 1 and start to send the START signal; **if the current I² C controller is in busy state, writing this command will trigger sending the START signal**. The waveform of sending START signal is shown in the figure below:



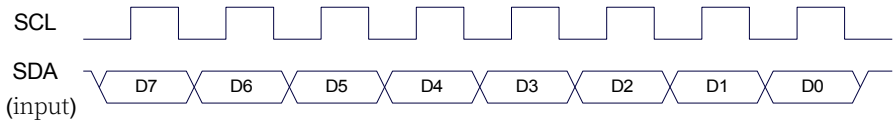
0010: Send data command.
After writing this command, the I² C bus controller generates 8 clocks on the SCL pin and sends the data in the I2CTXD register to the SDA pin bit-wise (the high data is sent first). The waveform of sending data is shown below:



0011: Receive ACK command.
After writing this command, the I² C bus controller generates 1 clock on the SCL pin and saves the data read from the SDA port to MSACKI (I2CMSST.1). The waveform of receiving ACK is shown below:

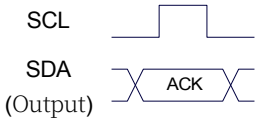


0100: Receive data command.
After writing this command, the I² C bus controller generates 8 clocks on the SCL pin and shifts the data read from the SDA port left to the I2CRXD register in order (high data is received first). The waveform of the received data is shown below:



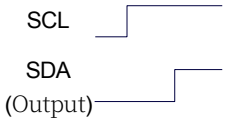
0101: Send ACK command.

After writing this command, the I² C bus controller generates 1 clock on the SCL pin and sends the data in MSACKO (I2CMSST.0) to the SDA port. The waveform of sending ACK is shown below:



0110: Cease and desist order.

Sends the STOP signal. After writing this command, the I² C bus controller starts to send the STOP signal. The waveform of the STOP signal is shown in the figure below:



0111: Reserved.

1000: Reserved.

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

This command is a combination of commands 0001, 0010, and 0011, and the controller will execute these three commands sequentially after this command is given.

1010: Send data command + receive ACK command.

This command is a combination of two commands, command 0010 and command 0011, and the controller will execute these two commands sequentially after this command is given.

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

This command is a combination of commands 0100 and 0101, the controller will execute these two commands sequentially after this command. Note: The answer signal returned by this command is fixed to ACK (0) and is not affected by the MSACKO bit.

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

This command is a combination of commands 0100 and 0101, the controller will execute these two commands sequentially after this command. Note: The answer signal returned by this command is fixed to NAK (1) and is not affected by the MSACKO bit.

22.3.3 I2C Host Auxiliary Control Register (I2CMSAUX)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	7EFE88H	-	-	-	-	-	-	-	WDTA

WDTA: I² C Data Auto Transmit Allow Bit in Host Mode

0: Disable automatic sending

1: Enable automatic sending

If the auto-send function is enabled, the I² C controller will automatically trigger the "1010" command when the MCU completes the write operation to the I2CTXD data register, i.e., it will automatically send the data and receive the ACK signal.

22.3.4 I2C Host Status Register (I2CMSST)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	7EFE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

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

0: Controller is idle

1: Controller is busy

When the I² C controller is in host mode, in the idle state, after sending the START signal, the controller enters the busy state, and the busy state will be maintained until the STOP signal is successfully sent, after which the state will return to the idle state again.

MSIF: Host mode interrupt request bit (interrupt flag bit). When the I² C controller in host mode executes the completion register I2CMSCR

The MSCMD command generates an interrupt signal, and the hardware automatically sets this bit to 1 to request an interrupt from the CPU, and the MSIF bit must be cleared by software after responding to the interrupt.

MSACKI: ACK data received after sending the "0011" command to the MSCMD bit of I2CMSCR in host mode. (Read-only bit)

MSACKO: Prepare the ACK signal to be sent out in host mode. When the "0101" command is sent to the MSCMD of I2CMSCR, it will be sent to the MSCMD of I2CMSCR, and then to the MSCMD of I2CMSCR.

After the bit, the controller automatically reads the data of this bit and sends it to the SDA as ACK.

22.4 I2C Slave Mode

22.4.1 I2C Slave Control Register (I2CSLCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI: Allowed bit for START signal interrupt received in slave mode

0: Disable interrupt when START signal is received in slave mode.

1: Interrupt occurs when the START signal is received when the slave mode is enabled.

ERXI: Interrupt Allowed bit after 1 byte of data is received in slave mode.

0: Disable interrupt after receiving data in slave mode

1: Interrupt occurs after 1 byte of data is received when slave mode is enabled.

ETXI: Interrupt allow bit after 1 byte of data has been sent in slave mode.

0: Disable interrupt after data transmission in slave mode.

1: Interrupt occurs after 1 byte of data has been sent when slave mode is enabled.

ESTOI: STOP signal received in slave mode is allowed.

0: Disable interrupt when STOP signal is received in slave mode.

1: Interrupt occurs when the STOP signal is received when the slave mode is enabled.

SLRST: Reset Slave Mode

22.4.2 I2C Slave Status Register (I2CSLST)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	7EFE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

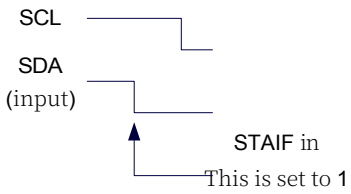
SLBUSY: I² C Controller status bit (read-only bit) when in slave mode

0: Controller is idle

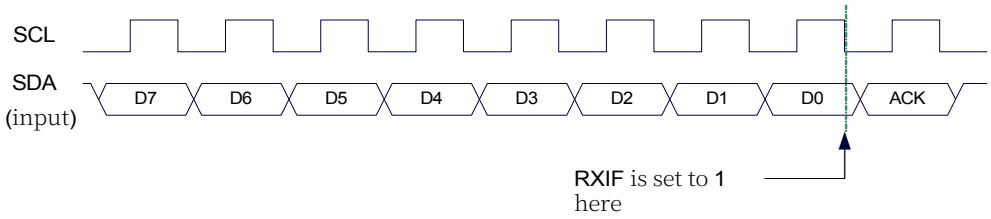
1: Controller is busy

When the I² C controller is in the slave mode, in the idle state, after receiving the START signal from the host, the controller will continue to detect the device address data after that, if the device address matches with the slave address image set in the current I2CSLADR register, the controller will enter into the busy state, and the busy state will be maintained until it successfully receives the STOP signal from the host, and the state will return to the idle state again. will be maintained until the STOP signal is successfully received from the host, after which the state will be restored to the idle state again.

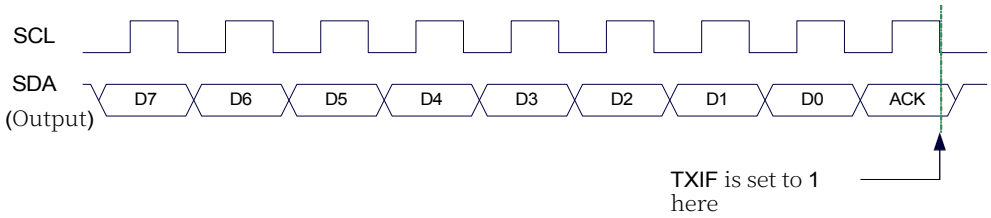
STAIF: Interrupt request bit after receiving START signal in slave mode. When the I² C controller in slave mode receives the START signal, the hardware will automatically set this bit to 1 and send a request for an interrupt to the CPU, and after responding to the interrupt, the STAIF bit must be cleared to 0 by the software.The time when STAIF is set to 1 is shown in the figure below:



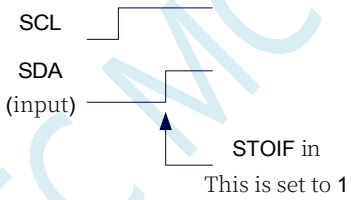
RXIF: Interrupt request bit after receiving 1 byte of data in slave mode. When the I² C controller in slave mode receives 1 byte of data, the hardware will automatically set this bit to 1 on the falling edge of the 8th clock and send a request for an interrupt to the CPU, and after responding to the interrupt, the RXIF bit must be cleared to zero by the software.The timing of when the RXIF is set to 1 is shown in the following figure:



TXIF: Interrupt request bit after sending 1 byte of data in slave mode. After the I² C controller in slave mode has sent 1 byte of data and successfully received a 1-bit ACK signal, the hardware will automatically set this bit to 1 on the falling edge of the 9th clock and send a request for an interrupt to the CPU, and after responding to the interrupt, the TXIF bit must be cleared to 0 by software. the time when the TXIF bit is set to 1 is shown in the following figure:

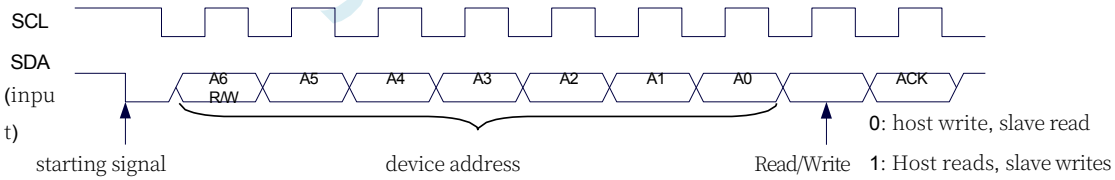


STOIF: Interrupt request bit after receiving STOP signal in slave mode. When the I² C controller in slave mode receives the STOP signal, the hardware automatically sets this bit to 1 and sends a request for interrupt to the CPU, and the STOIF bit must be cleared by software after responding to the interrupt.The time when STOIF is set to 1 is shown in the figure below:



SLACKI: ACK data received when in slave mode.

SLACKO: Prepare the ACK signal to be sent out in slave mode.



22.4.3 I2C Slave Address Register (I2CSLADR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	7EFE85H	SLADR [6:0]							MA

SLADR[6:0]: slave device address

When the I² C controller is in slave mode, the controller will continue to detect the device address data and read/write signals sent out by the host next after receiving the START signal. When the device address sent out by the host matches the slave device address set in SLADR[6:0], only then will the controller issue an interrupt request to the CPU to request the CPU to process the I² C event; otherwise, if the device address does not match,

the I²C controller will continue to continue to monitor and wait for the next start signal, and continue to match the next device address.

MA: Slave Device Address Matching Control

0: Device address must continue to match SLADR[6:0]

1: Ignore settings in SLADR, match all device addresses

22.4.4 I2C Data Registers (I2CTXD, I2CRXD)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	7EFE86H								
I2CRXD	7EFE87H								

I2CTXD is the I² C transmit data register, which holds the I² C data to be sent.

I2CRXD is the I² C Receive Data Register, which holds the I² C data of the completed reception.

22.5 sample procedure

22.5.1 I2C host mode access AT24C256 (interrupt mode)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include  "stc32g.h"                    // See download software for header files
#include "intrins.h"

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

bit         busy;

void I2C_Isr() interrupt 24
{
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40.               //clear interrupt flag
        busy = 0;
    }
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81.                     //Send START command
    while (busy);
}

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

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

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

void SendACK()
```

```
{
    I2CMSST = 0x00; //Set the ACK signal.          //Set ACK signal
    busy = 1;
    I2CMSCR = 0x85.                               //Send ACK command
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01.                               //Set 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()
{
    EAXFR = 1;                                    //Enable access to XFR
    CKCON = 0x00;                                 //Set the external data bus speed to fastest
    WTST = 0x00;                                  //set the program code wait parameter.
                                                  //Assign a value of 0 to set the CPU to execute the
                                                  programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    I2CCFG = 0xe0.                                //Enable I2C host mode
    I2CMSST = 0x00;
    EA = 1;
```

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

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

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

    while (1);
}
```

22.5.2 I2C host mode access to AT24C256 (query mode)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             // See download software for header files
#include "intrins.h"

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 data buffer
    I2CMSCR = 0x02; //Send SEND command.   //Send SEND command
    Wait();
}

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

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

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

void SendNAK()
{
    I2CMSST = 0x01.                        //Set 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()
{
    EAXFR = 1;                             //Enable access to XFR
    CKCON = 0x00;                          //Set the external data bus speed to fastest
```

```
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    I2CCFG = 0xe0.                  //Enable I2C host mode
    I2CMSST = 0x00;

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

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

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

    while (1);
}
```

22.5.3 I2C Host Mode Access to PCF8563

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                    //See download software for header files
#include "intrins.h"

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 data buffer
    I2CMSCR = 0x02; //Send SEND command.  //Send SEND command
    Wait();
}

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

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

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

void SendNAK()
{
    I2CMSST = 0x01.                    //Set 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()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign a value of 0 to set the CPU to execute the
                                programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    I2CCFG = 0xe0.              //Enable I2C host mode
    I2CMSST = 0x00;

    Start();                    //Send the start command
    SendData(0xa2).             //Send device address + write command
    RecvACK();
    SendData(0x02).             //Send the memory address
    RecvACK();
    SendData(0x00);             //Set the second value
    RecvACK();
    SendData(0x00);             //Set the minute value
    RecvACK();
    SendData(0x12).             //Set the hour value
    RecvACK();
    Stop();                     //Send the stop command

    while (1)
    {
        Start();                //Send the start command
        SendData(0xa2).         //Send device address + write command
        RecvACK();
        SendData(0x02).         //Send the memory address
        RecvACK();
```

```
Start();                                    //Send the start command
```

```
        SendData(0xa3).                         //Send device address + read command
        RecvACK();
        P0 = RecvData();                        //read the second value
        SendACK();
        P2 = RecvData();                        //Read the minute value
        SendACK();
        P3 = RecvData();                        //Read the hourly value
        SendNAK().
        Stop();                                 //Send the stop command

        Delay().
    }
}
```

22.5.4 I2C slave mode (interrupt mode)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             // see download software for header files
#include "intrins.h"

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

bit     isda;                                   //Device address flag
bit     isma.                                   //Store address flags

unsigned char           addr.
unsigned char edata     buffer[32];

void I2C_Isr() interrupt 24
{
    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40.                       //Handle the START event
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20.                       //Handle RECV event
        if (isda)
        {
           isda = 0;                            // Process RECV events (RECV DEVICE ADDR)
        }
        else if (isma)
        {
            isma = 0;                           // Process RECV events (RECV MEMORY ADDR)
            addr = I2CRXD.
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD.            // Process RECV events (RECV DATA)
        }
    }
```

```
    else if (I2CSLST & 0x10)
```

```
    {
        I2CSLST &= ~0x10.                       //Handle SEND event
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff.                      //Receive NAK then stop reading data
        }
        else
        {
            I2CTXD = buffer[++addr];            //Receive ACK and continue reading data.
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08.                       //Handle STOP event
        isda = 1;
        isma = 1;
    }
}

void main()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign a value of 0 to set the CPU to execute the
                                                programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    I2CCFG = 0x81.                              //Enable I2C slave mode
    I2CSLADR = 0x5a.                            //set slave device address register I2CSLADR = 0101_1010B
                                                //i.e. I2CSLADR[7:1]=010_1101B,MA=0B.
                                                //Since the MA is 0, the device address sent by the host
                                                must be the same as the MA address.
                                                //I2CSLADR[7:1] are the same to access this I2C slave
                                                device.
                                                // Host to send 5AH(0101_1010B) if it needs to write data.
                                                // Host needs to send 5BH(0101_1011B) if it needs to read
                                                the data.
    I2CSLST = 0x00;
    I2CSLCR = 0x78.                             //enable slave mode interrupt
    EA = 1;

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

```
    while (1);
}
```

22.5.5 I2C slave mode (query mode)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

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

bit         isda;                                      //Device address flag
bit         isma.                                      //Store address flags
unsigned char               addr;
unsigned char               buffer[32].
edata

void main()
{
    EAXFR = 1;      // Enable access to XFR
    CKCON = 0x00;                                      //Set the external data bus speed to fastest
    WTST = 0x00;                                       //set the program code wait parameter.
                                                       //Assign a value of 0 to set the CPU to execute the
                                                       programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    I2CCFG = 0x81.                                     //Enable I2C slave mode
    I2CSLADR = 0x5a.                                   //set slave device address register I2CSLADR = 0101_1010B
                                                       //i.e. I2CSLADR[7:1]=010_1101B,MA=0B.
                                                       //Since the MA is 0, the device address sent by the host
                                                       must be the same as the MA address.
                                                       //I2CSLADR[7:1] are the same to access this I2C slave
                                                       device.
                                                       // Host to send 5AH(0101_1010B) if it needs to write data.
                                                       // Host to send 5BH(0101_1011B) if it needs to read data
    I2CSLST = 0x00;
    I2CSLCR = 0x00.                                    // disable slave mode interrupt

    isda = 1;                                          // User variable initialisation
    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 RECV event
            if (isda)
            {
              isda = 0;                             // Process RECV events (RECV DEVICE ADDR)
            }
            else if (isma)
            {
                isma = 0;                           // Process RECV events (RECV MEMORY ADDR)
                addr = I2CRXD.
                I2CTXD = buffer[addr];
            }
            else
            {
                buffer[addr++] = I2CRXD.            // Process RECV events (RECV DATA)
            }
        }
        else if (I2CSLST & 0x10)
        {
            I2CSLST &= ~0x10.                       //Handle SEND event
            if (I2CSLST & 0x02)
            {
                I2CTXD = 0xff.                      //Receive NAK then stop reading data
            }
            else
            {
                I2CTXD = buffer[++addr];            //Receive ACK and continue reading data.
            }
        }
        else if (I2CSLST & 0x08)
        {
            I2CSLST &= ~0x08.                       //Handle STOP event
            isda = 1;
            isma = 1;
        }
    }
}
```

22.5.6 Host Code for Testing I2C Slave Mode Code

```
//Tested operating
frequency is 11.0592MHz

//#include "stc8h.h"
#include  "stc32g.h"                                // See download software for header files
#include "intrins.h"

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 data buffer
    I2CMSCR = 0x02; //Send SEND command.     //Send SEND command
    Wait();
}

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

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

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

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

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

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00.
```

```
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    I2CCFG = 0xe0.                          //Enable I2C host mode
    I2CMSST = 0x00;

    Start();                                //Send the start command
    SendData(0x5a).                         //Send device address (010_1101B) + write command (0B)
    RecvACK();
    SendData(0x00).                         //Send the memory address
    RecvACK();
    SendData(0x12);                         //write test data 1
    RecvACK();
    SendData(0x78).                         //Write test data 2
    RecvACK();
    Stop();                                 //Send the stop command

    Start();                                //Send the start command
    SendData(0x5a).                         //Send device address (010_1101B) + write command (0B)
    RecvACK();
    SendData(0x00);                         //Send the high byte of the memory address
    RecvACK();
    Start();                                //Send the 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 the stop command

    while (1);
}
```

23 Advanced PWM

The STC32G series microcontrollers have integrated 8-channel 16-bit advanced PWM timers, which are divided into two groups of PWMs with different cycles, named PWMA and PWMB, and can be set individually. The first group of PWM/PWMA can be configured as 4 complementary/symmetrical/deadband controlled PWMs or capture external signals, while the second group of PWM/PWMB can be configured as 4 PWM outputs or capture external signals.

The clock frequency of the first PWM/PWMA group can be the system clock after dividing by registers PWMA_PSCRH and PWMA_PSCRL, and the dividing value can be any value between 1 and 65535. The clock frequency of the second PWM/PWMB group can be the system clock divided by registers PWMB_PSCRH and PWMB_PSCRL, and the divided frequency value can be any value between 1 and 65535. The clock frequencies of the two PWMs can be set independently.

The first PWM Timer/PWMA has 4 channels (PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N), and each channel can independently realise the functions of PWM output (complementary symmetrical PWM output with deadband can be set), capture, and compare; the second PWM Timer/PWMB has 4 channels (PWM5, PWM6, PWM7, PWM8), and each channel can also independently realise the functions of PWM output, capture, and compare.
（The second PWM Timer/PWMB has 4 channels (PWM5, PWM6, PWM7, PWM8), and each channel can also independently realise PWM output, capture and compare functions. The only difference between the two groups of PWM timers is that the first group can output complementary symmetrical PWM with deadband, while the second group can only output single-ended PWM, and other functions are identical. Otherwise, the functions are exactly the same. The following introduction of the advanced PWM timers only takes the first group as an example.

When the first PWM timer is used to output PWM waveforms, the PWM1P/PWM2P/PWM3P/PWM4P outputs can be enabled independently, and the PWM1N/PWM2N/PWM3N/PWM4N outputs can be enabled independently, for example, if PWM1P outputs are enabled independently, PWM1N can no longer output independently unless PWM1P and PWM1N form a complementary symmetrical group. For example, if PWM1P is enabled separately, PWM1N can no longer be output independently, unless PWM1P and PWM1N form a complementary symmetrical output group. 4 outputs of PWMA can be set independently, for example, PWM1P and PWM2N outputs can be enabled separately, and PWM2N and PWM3N outputs can be enabled separately. If the first PWM timer is used for capture function or pulse width measurement, the input signal can only be input from the positive end of each channel, i.e., only PWM1P/PWM2P/PWM3P/PWM4P can be used for capture function and pulse width measurement.

When two sets of advanced PWM timers are used to capture external signals, you can choose either rising edge capture or falling edge capture. If you want to capture both rising and falling edges, you can connect the input signal to two PWMs at the same time, and enable one of them to capture the rising edge and the other one to capture the falling edge. What's **more, when external input signals are connected to two PWMs at the same time, the period and duty cycle of the signals can be captured at the same time**.

Comparison of three STC hardware PWMs:

Compatible with PCA/CCP/PWM of traditional 8051: outputs PWM waveforms, captures external

input signals, and outputs high-speed pulses. It can output 6-bit/7-bit/8-bit/10-bit PWM waveforms externally, and the frequency of 6-bit PWM waveforms is PCA module clock source frequency/64; the frequency of 7-bit PWM waveforms is PCA module clock source frequency/128; the frequency of 8-bit PWM waveforms is PCA module clock source frequency/256; the frequency of 10-bit PWM waveforms is PCA module clock source frequency/1024. Capture external input signal, can capture rising edge, falling edge or both rising and falling edge.

The 15-bit Enhanced PWM of STC8G series can only output PWM waveform externally without input capture function. The frequency and duty cycle of external PWM output can be set arbitrarily. Multiple complementary/symmetrical/deadband PWM waveforms can be realised by software intervention. External anomaly detection function and real-time trigger ADC conversion function are available.

STC32G/STC8H Series 16-bit Advanced PWM Timer: The most powerful PWM in STC, it can output PWM waveforms with arbitrary frequency and duty cycle. Complementary/symmetrical/deadband PWM waveforms can be output without software intervention. Capture external input signal, can capture rising edge, falling edge, or both rising edge and falling edge, when measuring external waveforms, can measure the period value and duty cycle value of the waveform at the same time. Quadrature encoding function, external anomaly detection function, and real-time trigger ADC conversion function.

In the following descriptions, PWMA represents the first PWM timer and PWMB represents the second PWM timer

Group 1 Advanced PWM Timer/PWMA

Internal Signal Description

TI1: External clock input signal 1 (PWM1P pin signal or PWM1P/PWM2P/PWM3P signal)
TI1F: TI1 signal after IC1F digital filtering
TI1FP: TI1F signal after CC1P/CC2P edge
detector **TI1F_ED**: edge signal of TI1F
TI1FP1: TI1F signal after CC1P edge detector
TI1FP2: TI1F signal after CC2P edge detector
IC1: Capture input signal of channel 1 selected
by CC1S **OC1REF**: Reference waveform output
from output channel 1 (intermediate waveform)
OC1: Main output signal of channel 1 (OC1REF signal after CC1P polarity processing)
OC1N: Complementary output signal of channel 1 (OC1REF signal after CC1NP polarity processing)

TI2: External clock input signal 2 (PWM2P
pin signal) **TI2F**: TI2 signal after IC2F digital
filtering **TI2F_ED**: TI2F edge signal
TI2FP: TI2F signal after CC1P/CC2P edge detector
TI2FP1: TI2F signal after CC1P edge detector
TI2FP2: TI2F signal after CC2P edge detector
IC2: Capture input signal of channel 2 selected
by CC2S **OC2REF**: Output reference waveform
(intermediate waveform) output from channel 2
OC2: Main output signal of channel 2 (OC2REF signal after CC2P polarity processing)
OC2N: Complementary output signal of channel 2 (OC2REF signal after CC2NP polarity processing)

TI3: External clock input signal 3 (PWM3P
pin signal) **TI3F**: TI3 signal digitally filtered
by IC3F **TI3F_ED**: Edge signal of TI3F
TI3FP: TI3F signal after CC3P/CC4P edge detector
TI3FP3: TI3F signal after CC3P edge detector
TI3FP4: TI3F signal after CC4P edge detector
IC3: Capture input signal of channel 3 selected
by CC3S **OC3REF**: Outputs the reference
waveform (intermediate waveform) output
from channel 3
OC3: Main output signal of channel 3 (OC3REF signal after CC3P polarity processing)
OC3N: Complementary output signal for channel 3 (OC3REF signal after CC3NP polarity processing)

TI4: External clock input signal 4 (PWM4P
pin signal) **TI4F**: TI4 signal after digital
filtering by IC4F **TI4F_ED**: Edge signal of
TI4F
TI4FP: TI4F signal after CC3P/CC4P edge detector
TI4FP3: TI4F signal after CC3P edge detector
TI4FP4: TI4F signal after CC4P edge detector
IC4: Capture input signal of channel 4 selected

by CC4S **OC4REF**: Output reference waveform (intermediate waveform) output from channel 4

OC4: Main output signal of channel 4 (OC4REF signal after CC4P polarity processing)

OC4N: Complementary output signal for channel 4 (OC4REF signal after CC4NP polarity processing)

ITR1: Internal trigger input signal 1

ITR2: Internal trigger input signal 2

TRC: fixed to TI1_ED

TRGI: Trigger input signal after TS multiplexer

TRGO: Trigger output signal after MMS multi selector

ETR: External trigger input signal (PWMETI1 pin signal)
ETRP: ETR signal after ETP edge detector and ETPS divider **ETRF**: ETRP signal after ETF digital filtering

BRK: Brake input signal (PWMFLT)

CK_PSC: prescaled clock, PWMA_PSCR prescaler input clock

CK_CNT: PWMA_PSCR prescaler output clock, PWM timer clock

Group 2 Advanced PWM Timer/PWMB Internal Signal Description

TI5: External clock input signal 5 (PWM5 pin signal or signal after PWM5/PWM6/PWM7 phase dissimilarity)
TI5F: TI5 signal after IC5F digital filtering
TI5FP: TI5F signal after CC5P/CC6P edge detector **TI5F_ED**: edge signal of TI5F
TI5FP5: TI5F signal after CC5P edge detector
TI5FP6: TI5F signal after CC6P edge detector
IC5: Capture input signal of channel 5 selected by CC5S **OC5REF**: Outputs reference waveform (intermediate waveform) output from channel 5

OC5: Main output signal of channel 5 (OC5REF signal after CC5P polarity processing)

TI6: External clock input signal 6 (PWM6 pin signal) **TI6F**: TI6 signal after digital filtering by IC6F **TI6F_ED**: Edge signal of TI6F
TI6FP: TI6F signal after CC5P/CC6P edge detector
TI6FP5: TI6F signal after CC5P edge detector
TI6FP6: TI6F signal after CC6P edge detector **IC6**: Capture input signal of channel 6 selected by CC6S **OC6REF**: Outputs the reference waveform (intermediate waveform) output from channel 6

OC6: Main output signal of channel 6 (OC6REF signal after CC6P polarity processing)

TI7: External clock input signal 7 (PWM7 pin signal) **TI7F**: TI7 signal after digital filtering by IC7F **TI7F_ED**: Edge signal of TI7F
TI7FP: TI7F signal after CC7P/CC8P edge detector
TI7FP7: TI7F signal after CC7P edge detector
TI7FP8: TI7F signal after CC8P edge detector **IC7**: Capture input signal of channel 7 selected by CC7S **OC7REF**: Output reference waveform (intermediate waveform) output from channel 7

OC7: Main output signal of channel 7 (OC7REF signal after CC7P polarity processing)

TI8: External clock input signal 8 (PWM8 pin signal)

TI8F: TI8 signal after digital filtering by IC8F
TI8F_ED: Edge signal of TI8F
TI8FP: TI8F signal after CC7P/CC8P edge detector
TI8FP7: TI8F signal after CC7P edge detector
TI8FP8: TI8F signal after CC8P edge detector **IC8**:
Capture input signal of channel 8 selected by
CC8S **OC8REF**: Output reference waveform
(intermediate waveform) output from channel 8
OC8: Main output signal for channel 8 (OC8REF signal after CC8P polarity processing)

23.1 brief

The only difference between PWMB and PWMA is that PWMA can output complementary symmetrical PWM with deadband, while PWMB can only output single-ended PWM. The following introduction of advanced PWM only takes PWMA as an example.

The PWMA consists of a 16-bit auto-load counter driven by a programmable prescaler.

PWMA is suitable for many different applications:

- basic timing
- Measurement of the pulse width of the input signal (input capture)
- Generates output waveforms (output comparison, PWM and single pulse modes)
- Interrupts corresponding to different events (capture, compare, overflow, brake, trigger)
- Synchronisation with PWMB or external signals (external clock, reset, trigger and enable)

The PWMA is suitable for a wide range of control applications, including those requiring an intermediate alignment mode PWM, which supports complementary outputs and dead time control.

The clock source for PWMA can be an internal clock or an external signal, which can be selected through the configuration registers.

23.2 Main characteristics

PWMA features include:

- 16-bit up, down, up/down auto-load counter
- Repeat counter that allows the timer register to be updated after a specified number of counter cycles
- 16-bit programmable (can be modified in real time) prescaler, counter clock frequency division factor of any value between 1 and 65535
- Synchronisation circuit for controlling a timer using external signals and timer interconnections
- Up to 4 independent channels can be configured as:
  - Input Capture
  - Output Comparison
  - PWM output (edge or centre aligned mode)

- Six-step PWM output
- Single pulse mode output
- Supports 4 on-channel complementary outputs with programmable dead time

- The brake input signal (PWMFLT) can place the timer output signal in a reset state or a deterministic state
- External trigger input pin (PWMETI)

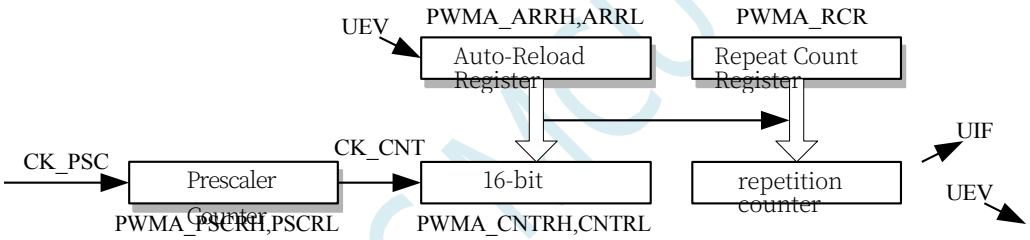
- Events that generate interruptions include:
  - Updates: Counter overflow up/down, counter initialisation (via software or internal/external trigger)
  - Trigger events (counter starts, stops, initialises or counts triggered internally/externally)
  - Input Capture
  - Output Comparison
  - Brake signal input

23.3 time base unit (in computing)

The time base unit of PWMA is included:

- 16-bit Up/Down Counter
- 16-bit Auto-Reload Register
- repetition counter
- prescaler

PWMA time base unit



The 16-bit counter, prescaler, auto-reload registers, and repeat counter registers can be read and written by software. The auto-reload register consists of a preload register and a shadow register.

The auto-reload register can be written in in both modes:

- Auto preload is enabled (ARPE bit of PWMA_CR1 register is 1).

In this mode, data written to the auto-reload register is saved in the preload register and transferred to the shadow register at the next update event (UEV).

- Auto preload is disabled (ARPE bit of PWMA_CR1 register is 0). In this mode, data written to the auto-reload register is immediately written to the shadow register. Update the event generation condition:
- The counter overflows up or down.
- The UG bit of the PWMA_EGR register is set by software.
- The clock/trigger controller generates the trigger event.

With preload enabled (ARPE=1), if an update event occurs, the value in the preload register (PWMA_ARR) is written to the shadow register and the value in the PWMA_PSCR register is

written to the prescaler.

Setting the UDIS bit of the PWMA_CR1 register disables the update event (UEV).

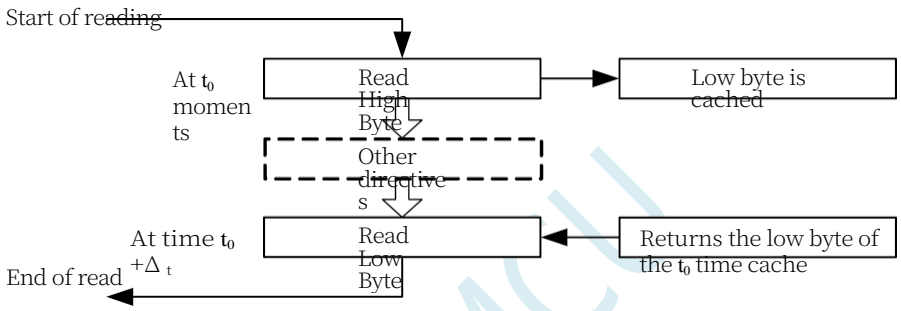
The output of the prescaler, CK_CNT, drives the counter, and CK_CNT is valid only when the counter enable bit (CEN) of the IM1_CR1 register is set.

Note: The actual counter does not start counting until one clock cycle after the CEN bit is enabled.

23.3.1 Read/Write 16-bit Counter

The write counter operation is not cached and the PWMA_CNTRH and PWMA_CNTRL registers can be written at any time, so to avoid writing a wrong value, it is generally recommended not to write a new value while the counter is running.

The read counter operates with an 8-bit buffer. The user must read the high byte of the timer first, and after the user has read the high byte, the low byte will be
The cached data is automatically cached and will be held until the read operation of the 16-bit data is completed.



23.3.2 16-bit PWMA_ARR register write operation

The value in the preloaded register is written to the 16-bit PWMA_ARR register, which is accomplished by two instructions, each writing to the
1 byte. The high byte must be written first, followed by the low byte.

The shadow register is latched when the high byte is written and held until the low byte is written.

23.3.3 prescaler

Prescaler implementation:

The PWMA's prescaler is based on a 16-bit counter controlled by a 16-bit register (PWMA_PSCR). Since this control register
The counter has a buffer so that it can be changed at runtime. The prescaler divides the counter clock frequency by any value between 1 and 65536. The prescaler value is written from the preload register, and the shadow register, which holds the currently used value, is loaded on a low byte write. Since two separate write operations are required to write the 16-bit register, it is important to ensure that the high byte is written first. The new prescaler value is adopted

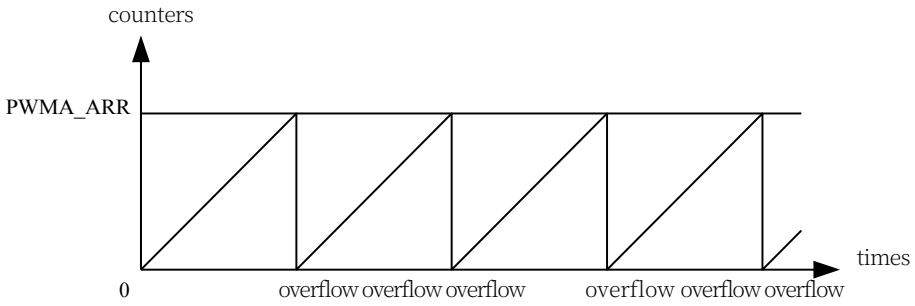
on the arrival of the next update event. Read operations to the PWMA_PSCR register are accomplished by preloading the register.

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

23.3.4 Up Count Mode

In Up Count mode, the counter counts from 0 to a user-defined comparison value (the value of the PWMA_ARR register), then starts counting from 0 again and generates a counter overflow event, at which time an update event (UEV) will be generated if the UDIS bit of the PWMA_CR1 register is 0.

Counter for upward counting mode



An update event can also be generated by software or by using a trigger controller to set the UG bit of the PWMA_EGR register.

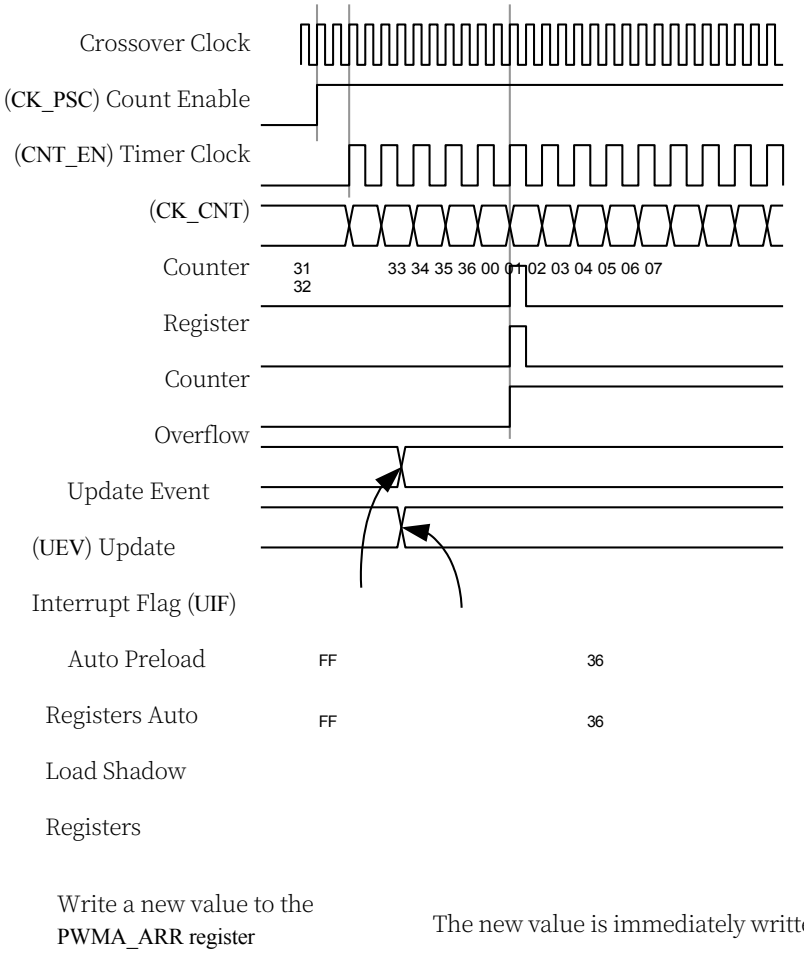
The update event can be disabled by using software to set the UDIS bit of the PWMA_CR1 register, which prevents the shadow register from being updated when the preloaded registers are updated. No update event will be generated until the UDIS bit is cleared. However, when an update event should be generated, the counter will still be cleared to 0 and the prescaler count will also be cleared to 0 (but the prescaler value will remain unchanged). In addition, if the URS bit in the PWMA_CR1 register is set (select update request), setting the UG bit will generate an update event UEV, but the hardware does not set the UIF flag
(i.e., no interrupt request is generated). This is to avoid generating both update and capture interrupts when clearing the counter in capture mode.

When an update event occurs, all registers are updated and the hardware sets the update flag bit (UIF bit of the PWMA_SR register) at the same time based on the URS bit:

- The autoload shadow register is reset to the value of the preload register (PWMA_ARR).
- The prescaler buffer is set to the value of the preload register (the contents of the PWMA_PSC register).

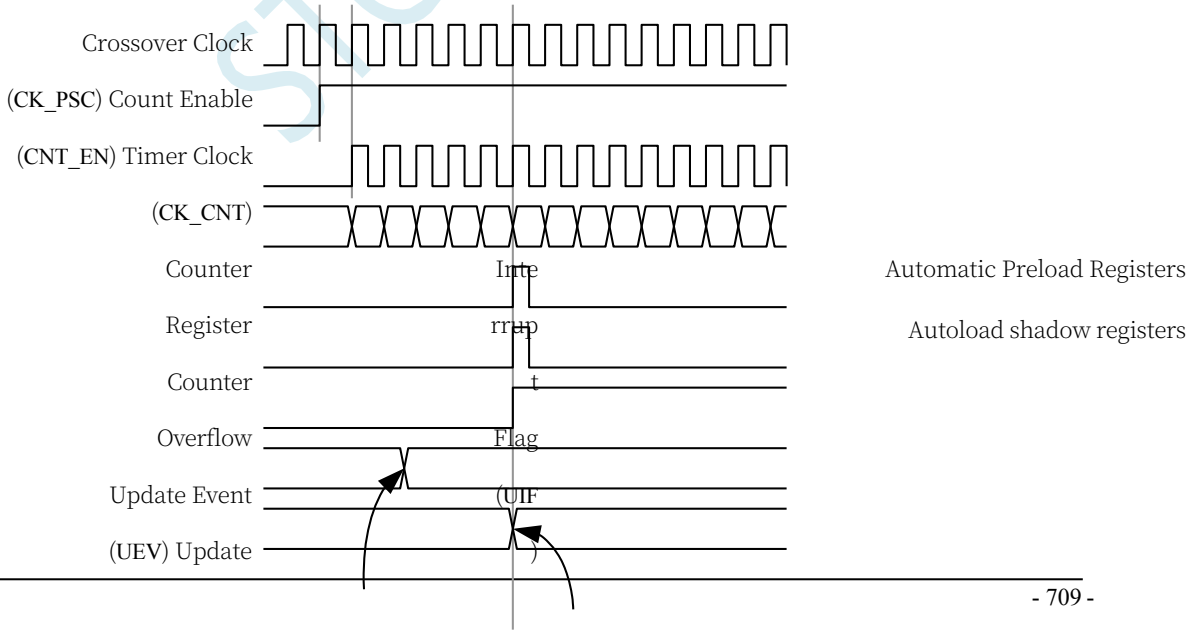
The figure below gives some examples of how the counter acts at different clock frequencies when PWMA_ARR=0x36. In the figure the prescaler frequency is 2, so the counter clock (CK_CNT) frequency is half of the prescaler clock (CK_PSC) frequency. The figure disables the auto-load function (ARPE=0), so when the counter reaches 0x36, the counter overflows and the shadow register is immediately updated and an update event is generated.

When ARPE=0 (ARR is not preloaded), the counter is updated when the prescaler is 2:



The figure below has a prescaler of 1, so the CK_CNT frequency is the same as the CK_PSC. The figure enables auto-reload (ARPE=1) so that an overflow is generated when the counter reaches 0xFF. 0x36 will be written on overflow and an update event is generated.

ARPE=1(PWMA_ARR Preload) Counter update when prescaler is 1:



FB FC FD FE FF 00 fb fc fd fe ff 00 01 02 03 04 05 06 07

FF 36

FF 36

Write a new value to the PWMA_ARR register

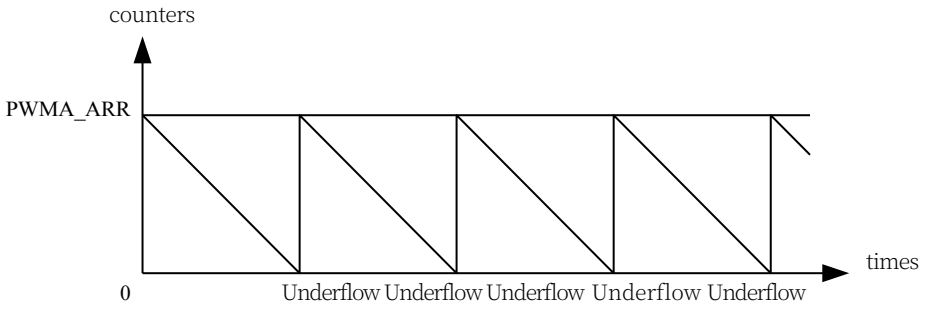
New value written to shadow register on counter overflow

23.3.5 Down Count Mode

In down mode, the counter counts down to 0 from the auto-loaded value (the value of the PWMA_ARR register), and then counts down again from the self

The dynamically loaded value restarts counting and generates a counter down overflow event. An update event (UEV) is also generated if the UDIS bit of the PWMA_CR1 register is cleared.

Counter for Down Count Mode



An update event can also be generated by software or by using a trigger controller to set the UG bit of the PWMA_EGR register.

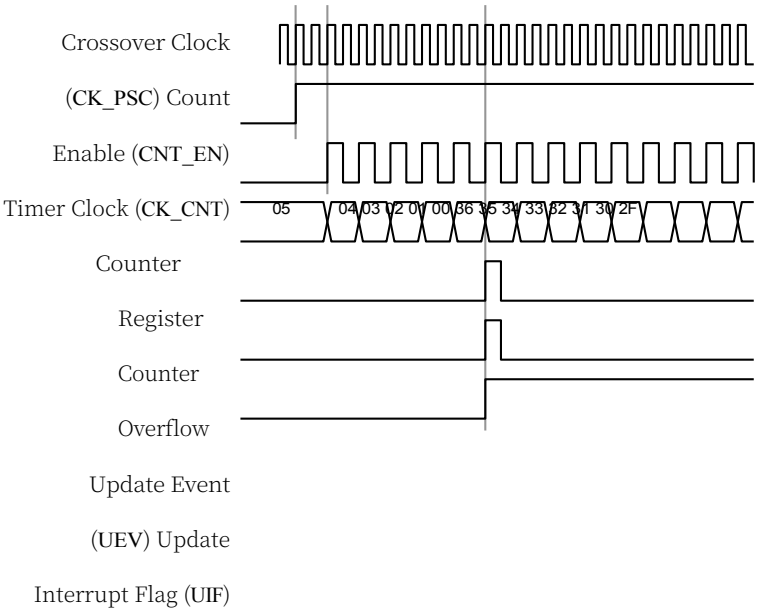
Setting the UDIS bit of the PWMA_CR1 register disables UEV events. This prevents the shadow registers from being updated when the preload registers are updated. Therefore, no update event is generated until the UDIS bit is cleared. However, the counter will still restart counting from the current autoload value and the counters for the prescaler restart from 0 (but the prescaler cannot be modified). In addition, if the URS bit in the PWMA_CR1 register is set (selecting an update request), setting the UG bit generates an update event UEV but does not set the UIF flag (and therefore does not generate an interrupt), this is to avoid generating both an update and a capture interrupt when a capture event occurs and clears the counter.

When an update event occurs, all registers are updated and the hardware sets the update flag bit (UIF bit of PWMA_SR register) at the same time based on the URS bit:

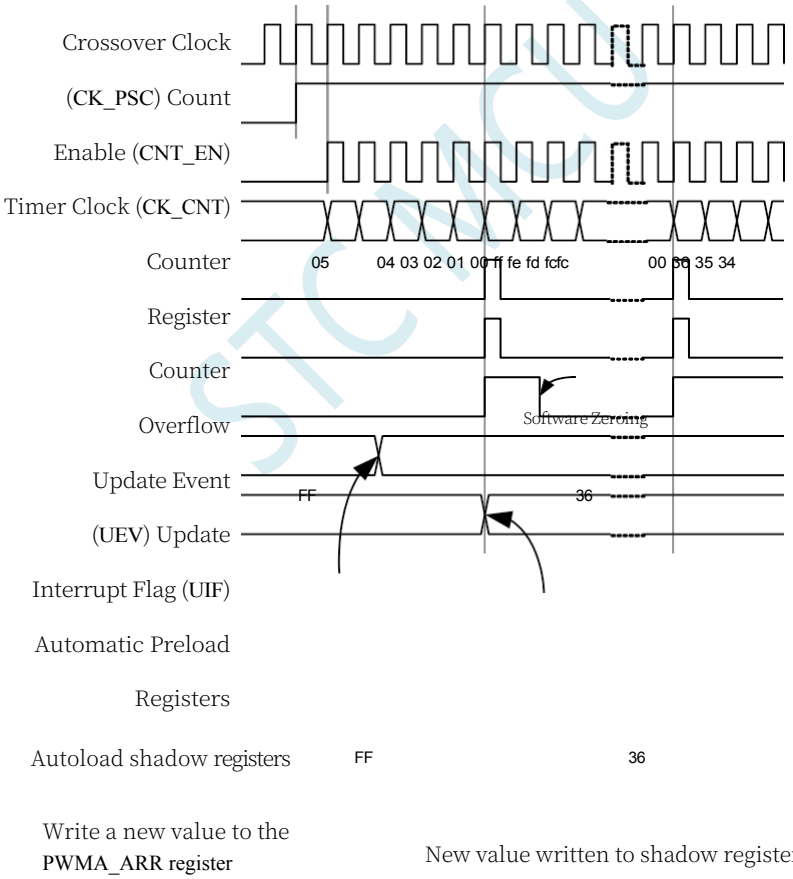
- The autoload shadow register is reset to the value of the preload register (PWMA_ARR).
- The prescaler buffer is set to the value of the preload register (the contents of the PWMA_PSC register).

Below are some graphs of the counter at different clock frequencies when PWMA_ARR=0x36. The following graph depicts that in down count mode, the new value is written at the next cycle when preload is not enabled.

ARPE=0 (ARR not preloaded), counter update at prescaler of 2:



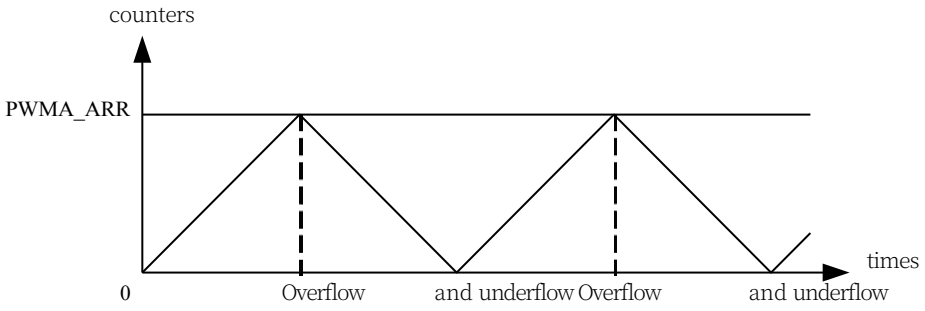
ARPE=1 (ARR preloaded), counter update when prescaler is 1



23.3.6 Middle alignment mode (count up/down)

In centre-aligned mode, the counter starts counting from 0 to the value of the PWMA_ARR register, generates a counter overflow event, then counts down from the value of the PWMA_ARR register to 0 and generates a counter underflow event; and then counts back up from 0 again.

The DIR direction bit in PWMA_CR1 cannot be written in this mode. It is updated by hardware and indicates the current count direction. Counter for Central Alignment Mode



If the timer has a repeat counter, an update event (UEV) is generated after the specified number of upward and downward overflows (the value of PWMA_RCR) have been repeated. Otherwise an update event is generated for each upward and downward overflow.

An update event can also be generated by software or by using a trigger controller to set the UG bit of the PWMA_EGR register. At this point, the counter starts counting from 0 again and the prescaler starts counting from 0 again.

Setting the UDIS bit in the PWMA_CR1 register disables UEV events. This prevents the shadow registers from being updated when the preload registers are updated. Therefore no update events are generated until the UDIS bit is cleared to 0. However, the counter will still continue to count up or down depending on the current auto-reload value. If the timer is equipped with a repeat counter, care needs to be taken when modifying it since the repeat register is not double buffered and the new repeat value will take effect immediately. In addition, if the URS bit in the PWMA_CR1 register is set (selecting an update request), setting the UG bit generates an update event UEV but does not set the UIF flag (and therefore does not generate an interrupt), this is to avoid generating both an update and a capture interrupt when a capture event occurs and clears the counter.

When an update event occurs, all registers are updated and the hardware updates the registers based on the URS bit update flag bit (in the PWMA_SR register).

(UIF bit):

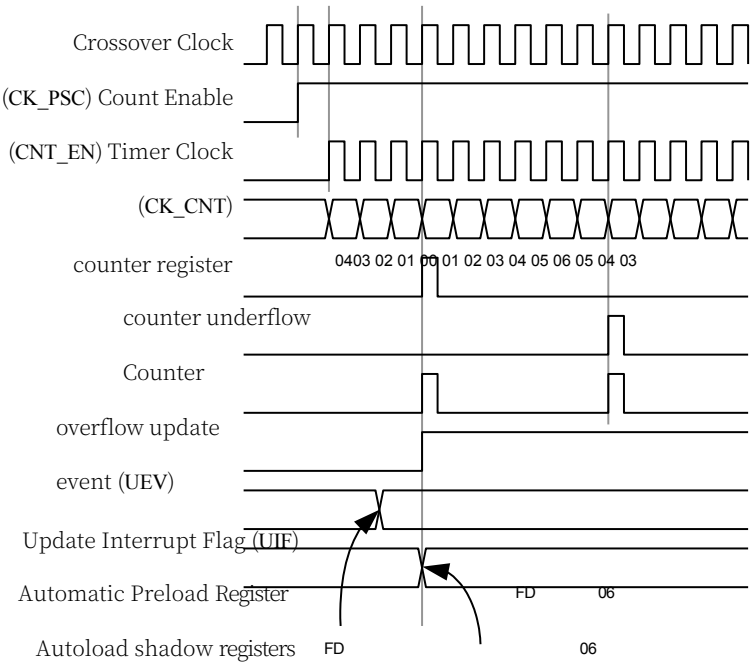
- The prescaler buffer is loaded with the value of the preload (PWMA_PSC register).
- The current autoload register is updated to the preload value (the contents of the PWMA_ARR register).

It is important to note that if an update occurs because of a counter overflow, the auto-reload register will be updated before the counter is reloaded, so the next cycle is the expected value (the counter is loaded with the new value).

Here are some examples of counter operation at

different clock frequencies: internal clock divider

factor of 1, PWMA_ARR = 0x6, ARPE = 1



Hints for using central alignment mode:

- When the central alignment mode is activated, the counter will count in the original up/down configuration. That is, the DIR bit in the PWMA_CR1 register will determine whether the counter counts up or down. In addition, software cannot modify the value of DIR bit and CMS bit at the same time.
- It is not recommended to write the counter value while the counter is counting in central alignment mode, as this will lead to unforeseen consequences. Be specific:
  - When a value larger than the auto-load value is written to the counter (PWMA_CNT>PWMA_ARR), but the counter counting direction does not change. For example, the counter has overflowed upwards, but the counter still counts upwards.
  - A value of 0 or PWMA_ARR was written to the counter, but the update event did not occur.
- The safe way to use the counters in centre-aligned mode is to generate an update event in software (by setting the UG bit of the PWMA_EGR register) before starting the counters, and not to modify the counter value while the counters are counting.

23.3.7 repetition counter

The time base unit explains how the Update Event on Counter Up/Down Overflow (UEV) is generated, however in fact it can only be generated when the Repeat Counter value reaches zero. This feature is useful for generating PWM signals.

This means that data is transferred from the preload registers to the shadow registers (PWMA_ARR auto-reload register, PWMA_PSC preload register, and also the capture/compare register PWMA_CCRx in compare mode) every N counts of overflows or underflows, and N is

the value in the PWMA_RCR repeat count register.

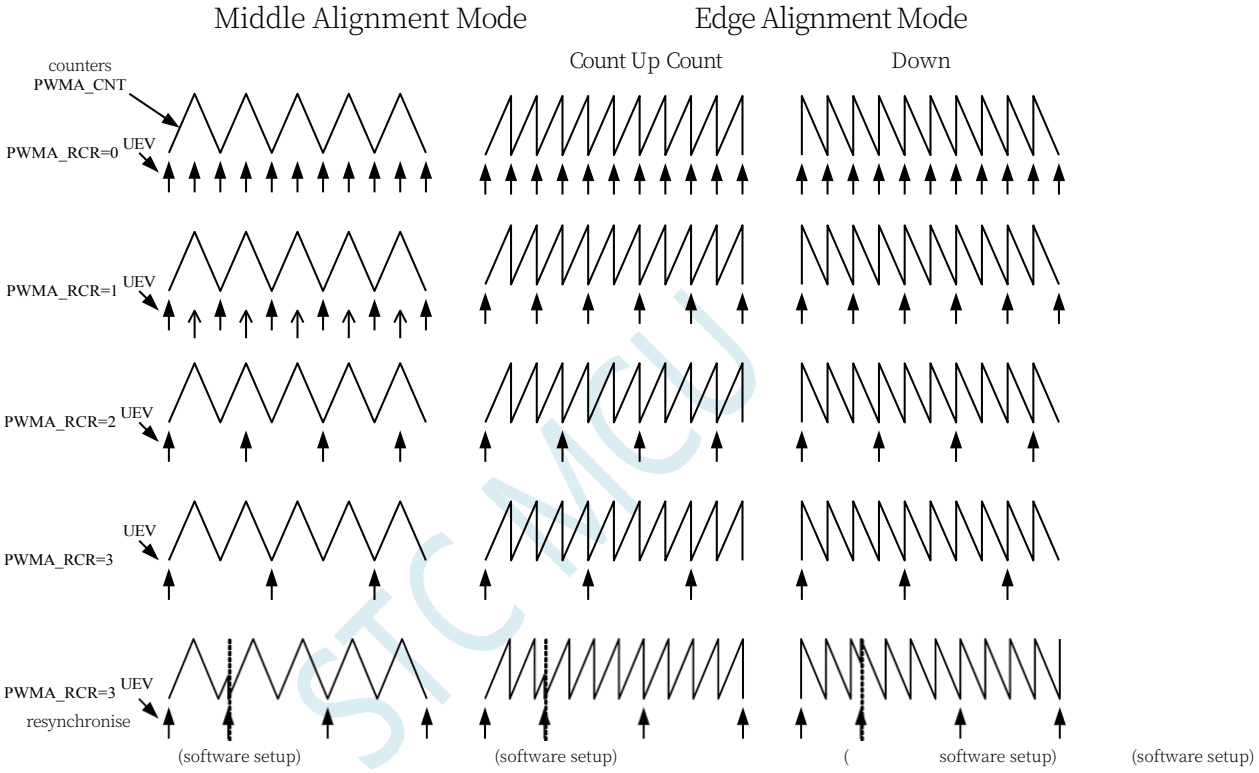
The repeat counter decreases when any of the following conditions hold:

- Each time the counter overflows upwards in upward count mode
- Each time the counter overflows downwards in the downward counting mode
- Central alignment mode at each overflow and at each underflow.

Although this limits the maximum cycle time of the PWM to 128, it is capable of updating the duty cycle 2 times per PWM cycle. In centre-aligned mode, since the waveform is symmetrical, the maximum resolution is 2*tCK_PSC if the compare register is only refreshed once in each PWM cycle.

The repeat counter is auto-loaded and the repeat rate is defined by the value of the PWMA_RCR register. When an update event is generated by software (by setting the UG bit in PWMA_EGR)or by a hardware clock/trigger controller, the update event occurs immediately regardless of the value of the Repeat Counter and the contents of the PWMA_RCR register are reloaded into the Repeat Counter.

Examples of update rates in different modes and PWMA_RCR register setting



23.4 Clock/Trigger Controller

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

23.4.1 Pre-Split Clock (CK_PSC)

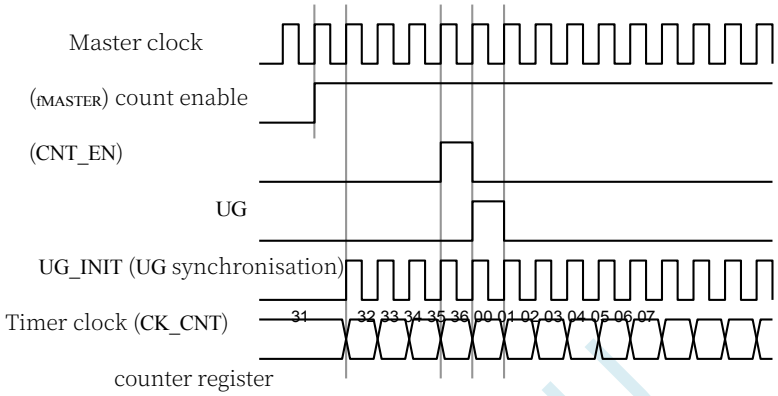
The prescaled clock (CK_PSC) of the Time Base Unit can be supplied from the following sources:

- Internal clock (fMASTER)
- External Clock Mode 1: External Clock Input (TIx)
- External clock mode 2: External trigger input ETR
- Internal Trigger Input (ITRx): uses one timer as a pre-divided clock for another timer.

23.4.2 Internal clock source (fMASTER)

If both the clock/trigger mode controller and the external trigger input are disabled (SMS=000 in the PWMA_SMCR register and ECE=0 in the PWMA_ETR register), the CEN,DIR, and UG bits are de facto control bits and can only be modified by software (the UG bit is still cleared automatically). Once the CEN bit is written to 1, the prescaler clock is provided by the internal clock.

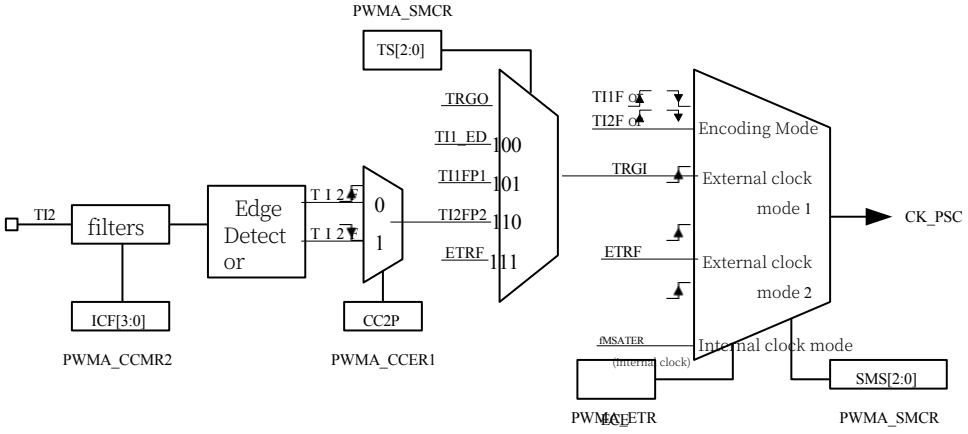
The following diagram depicts the operation of the control circuit and up counter in normal mode without prescaler. Control circuit in normal mode with $_{fMASTER}$ crossover factor 1



23.4.3 External clock source mode 1

This mode is selected when SMS=111 in PWMA_SMCR register. The counter can count on each rising or falling edge of the selected input.

TI2 External Clock Connection Example



For example, to configure the up counter to count on the rising edge of the TI2 input, use the following procedure:

1. Configure CC2S=01 in the PWMA_CCMR2 register to detect the rising edge of the TI2 input using channel 2

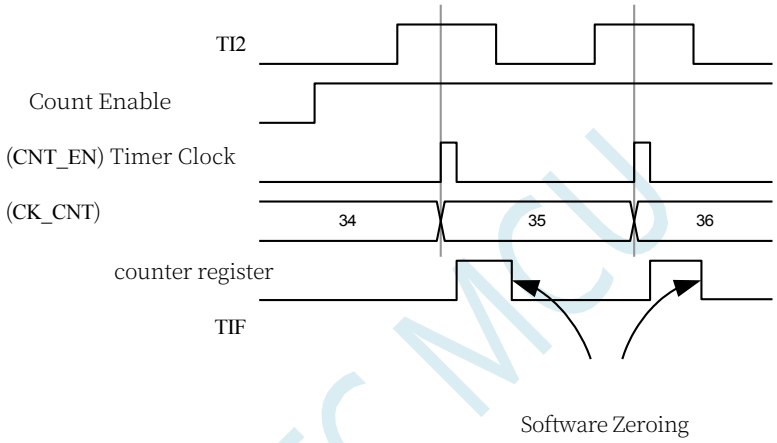
2. Configure the IC2F[3:0] bits of the PWMA_CCMR2 register to select the input filter bandwidth (if no filter is required, keep IC2F=0000) Note: The capture prescaler is not used as a trigger, so there is no need to configure it, and similarly there is no need to configure the TI2S bits, they are only used to select the input capture source.
3. Configure CC2P=0 in PWMA_CCER1 register to select rising edge polarity
4. Configure the PWMA_SMCR register with SMS=111 to configure the counter to use external clock mode 1
5. Configure TS=110 of PWMA_SMCR register to select TI2 as the input source
6. Set CEN=1 in PWMA_CR1 register to start counter

When the rising edge occurs at TI2, the counter counts once and the Trigger Identifier bit (TIF bit in the PWMA_SR1 register) is set to 1. If an interrupt is enabled (configured in the PWMA_IER register) an interrupt request is generated.

The delay between the rising edge of TI2 and the actual clock of the counter

depends on the resynchronisation circuit at the TI2 input. Control Circuit in

External Clock Mode 1

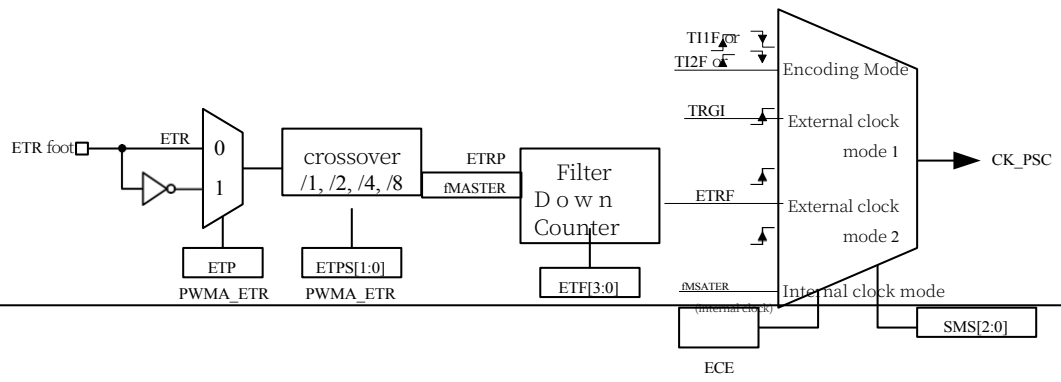


23.4.4 External clock source mode 2

The counter is capable of counting on every rising or falling edge of the externally triggered input ETR signal. Setting the PWMA_ETR register ECE

This mode is selected by writing a 1 to this bit.

General block diagram of the external trigger input:



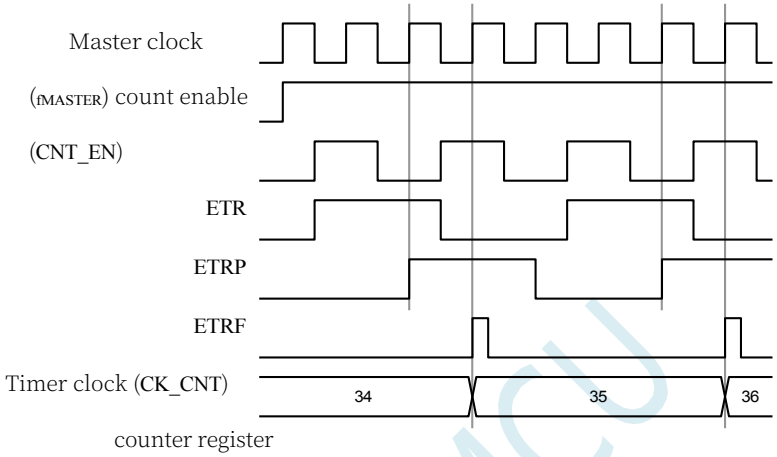
PWMA_ETR

PWMA_ETR PWMA_SMCR

For example, to configure the counter to count up every 2 rising edges of the ETR signal, use the following procedure:

1. No filter is needed in this example, configure ETF[3:0]=0000 in PWMA_ETR register
2. Set the prescaler and configure ETPS[1:0]=01 of PWMA_ETR registers
3. To select the rising edge detection of ETR, configure ETP=0 of PWMA_ETR register
4. Enable external clock mode 2, configure ECE=1 in PWMA_ETR register
5. Start the counter, write CEN=1 of PWMA_CR1 register

The counter counts every 2 ETR rising edges. Control Circuit in External Clock Mode 2



23.4.5 trigger synchronisation

The PWMA counters are synchronised to an external trigger signal using three modes:

- Standard Trigger Mode
- Reset Trigger Mode
- Gate Trigger Mode

Standard Trigger Mode

Counter enable (CEN) is dependent on events on the selected input.
In the following example, the counter starts counting up on the rising edge of the TI2 input:

1. Configure CC2P = 0 in PWMA_CCER1 register to select the rising edge of TI2 as the trigger condition.
2. Configure SMS=110 of PWMA_SMCR register to select the counter as trigger mode. Configure the PWMA_SMCR register's TS=110, select TI2 as the input source.

When a rising edge of TI2 occurs, the counter starts counting driven by the internal clock and the TIF flag is set. the delay between the rising edge of TI2 and the counter starting counting depends on the resynchronisation circuitry on the TI2 input.

Control circuit for standard trigger mode

Reset Trigger Mode

The counter and its prescaler can be reinitialised on the occurrence of a trigger input event. At the same time, if the URS bit of the PWMA_CR1 register is low, an update event UEV is also generated, and then all the preload registers (PWMA_ARR , PWMA_CCRx) are updated.

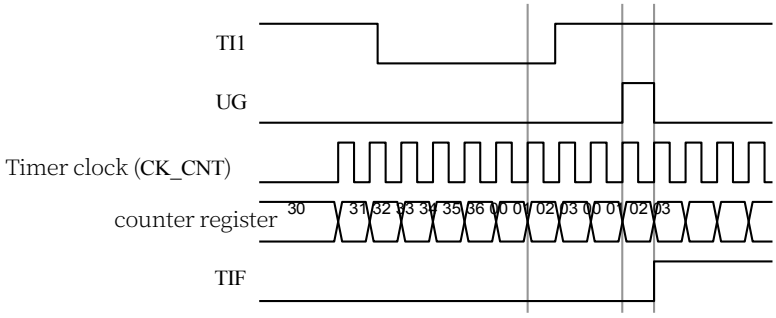
In the following example, the rising edge of the TI1 input causes the up counter to be cleared:

1. Configure CC1P=0 of the PWMA_CCER1 register to select the polarity of TI1 (only the rising edge of TI1 is detected).
2. Configure SMS=100 of PWMA_SMCR register to select the timer as reset trigger mode. Configure TS=101 of PWMA_SMCR register to select TI1 as input source.
3. Configure CEN=1 of PWMA_CR1 register to start the counter.

The counter starts counting according to the internal clock and then counts normally until TI1 has a rising edge. At this point, the counter is cleared to zero and counting resumes from zero. At the same time, the trigger flag (TIF bit in PWMA_SR1 register) is set and an interrupt request is generated if an interrupt is enabled (TIE bit in PWMA_IER register).

The following figure shows the action when the auto-reload register PWMA_ARR=0x36. The delay between the rising edge of TI1 and the actual reset of the counter depends on the resynchronisation circuitry at the TI1 input.

Control circuit in reset trigger mode



Gate Trigger Mode

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

In the following example, the counter only counts up when TI1 is low:

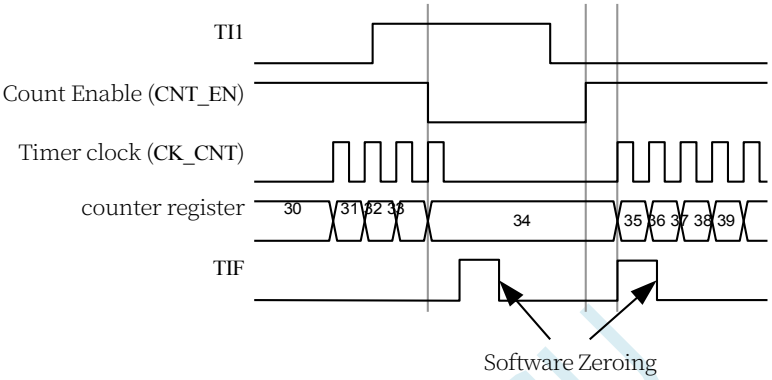
1. Configure CC1P=1 of the PWMA_CCER1 register to determine the polarity of TI1 (detects only a low level on TI1).

2. Configure SMS=101 in PWMA_SMCR register to select the timer as gate trigger mode, configure TS=101 in PWMA_SMCR register to select TI1 as input source.
3. Configure CEN=1 in the PWMA_CR1 register to start the counter (in gated mode, if CEN=0, the counter cannot start, regardless of the trigger input level).

The counter starts counting according to the internal clock as long as TI1 is low, and stops counting once TI1 goes high. When the counter starts or stops

The TIF flag bits are all set. the delay between the rising edge of TI1 and the actual stopping of the counter depends on the resynchronisation circuitry at the TI1 input.

Control Circuit in Gated Trigger Mode



External clock mode 2

Combined trigger mode

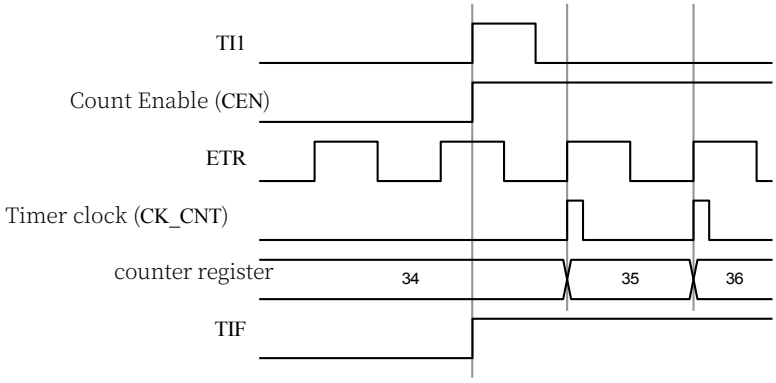
The external clock mode 2 can be used with the trigger mode of another input signal. For example, the ETR signal is used as an input to the external clock and another input signal can be used as a trigger input (standard trigger mode, reset trigger mode and gated trigger mode are supported). Note that ETR cannot be configured as TRGI via the TS bit of PWMA_SMCR register.

In the following example, once a rising edge occurs on TI1, the counter counts up on every rising edge of ETR:

1. Configure the external trigger input circuit through the PWMA_ETR register. Configure ETPS=00 to disable prescaling and ETP=0 to monitor the
   Configure ECE=1 to enable external clock mode 2 on the rising edge of the ETR signal.
2. Configure CC1P=0 of PWMA_CCER1 register to select the rising edge trigger of TI1.
3. Configure SMS=110 of PWMA_SMCR register to select the timer as trigger mode. Configure the PWMA_SMCR register's
   TS=101 to select TI1 as the input source.

When a rising edge occurs on TI1, the TIF flag is set and the counter starts counting on the rising edge of ETR. The delay between the rising edge of the TI1 signal and the actual clock of the counter depends on the resynchronisation circuit at the TI1 input. The delay between the rising edge of the ETR signal and the actual clock of the counter depends on the resynchronisation circuit at the ETRP input. Control Circuit in External Clock

Mode 2+ Trigger Mode



23.4.6 Synchronisation with PWMB

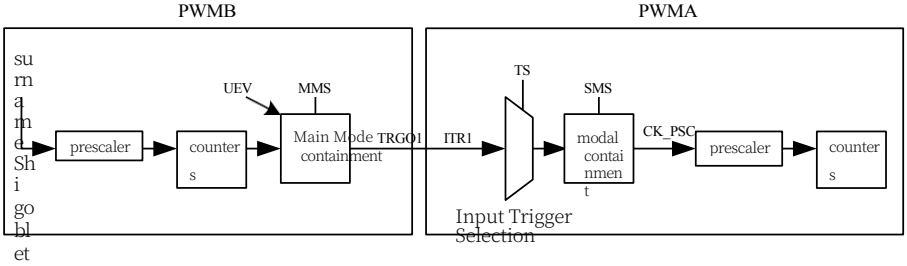
In the chip, timers are internally interconnected for timer synchronisation or linking. When a timer is configured in master mode, a trigger signal (TRGO) can be output to those timers configured in slave mode to perform reset operation, start operation, stop operation, or as a drive clock for those timers.

Using PWMB's TRGO as PWMA's pre-divided clock

For example, users can configure PWMB as the pre-divided clock for PWMA, which needs to be configured as follows:

1. Configure PWMB to be in master mode so that a periodic trigger signal is output at each update event (UEV). Configure PWMB_CR2 register with MMS=010 so that the TRGO can output a rising edge at each update event.
2. The TRGO signal output from PWMB is linked to PWMA, which needs to be configured in Trigger Slave Mode, using ITR2 as the input trigger signal. The above operation can be realised by configuring TS=010 in PWMA_SMCR register.
3. Configuring SMS=111 in the PWMA_SMCR register sets the clock/trigger controller to external clock mode 1. This operation will cause the rising edge of the periodic trigger signal TRGO output from the PWMB to drive the PWMA clock.
4. Finally, set the CEN bit of PWMB (in PWMB_CR1 register) to enable both PWMs.

Master/Trigger Slave Mode Timer Example



Enabling PWMA with PWMB

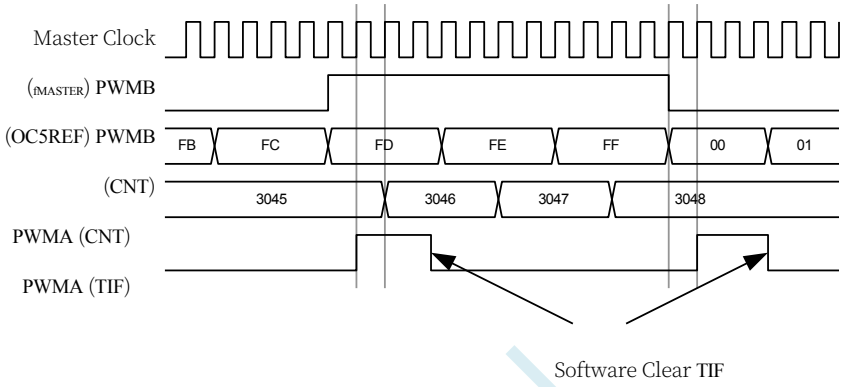
In this example, we enable PWMA with the compare output of PWMB, which only counts according to its own drive clock when the OC1REF signal of PWMB is high. Both PWMs are clocked with a 4-division f_{MASTER} (f_{CK_CNT} = fMASTER/4).

1. Configure PWMB as the master mode to output the comparison output signal (OC5REF) as the trigger signal. (Configure MMS=100 of PWMB_CR2 register).

2. Configure the waveform of the OC5REF signal of PWMB (PWMB_CMR1 register).
3. Configure PWMA to use the output of PWMB as its own trigger input signal (configure TS=010 of PWMA_SMCR register).
4. Configure the PWMA for gated trigger mode (configure SMS=101 in the PWMA_SMCR register).
5. Set the CEN bit (PWMA_CR1 register) to enable PWMA.
6. Set the CEN bit (PWMB_CR1 register) to enable PWMB.

Note: The clocks of the two PWMs are not synchronised, but only the PWMA enable signal is affected.

Output gating of PWMB triggers PWMA

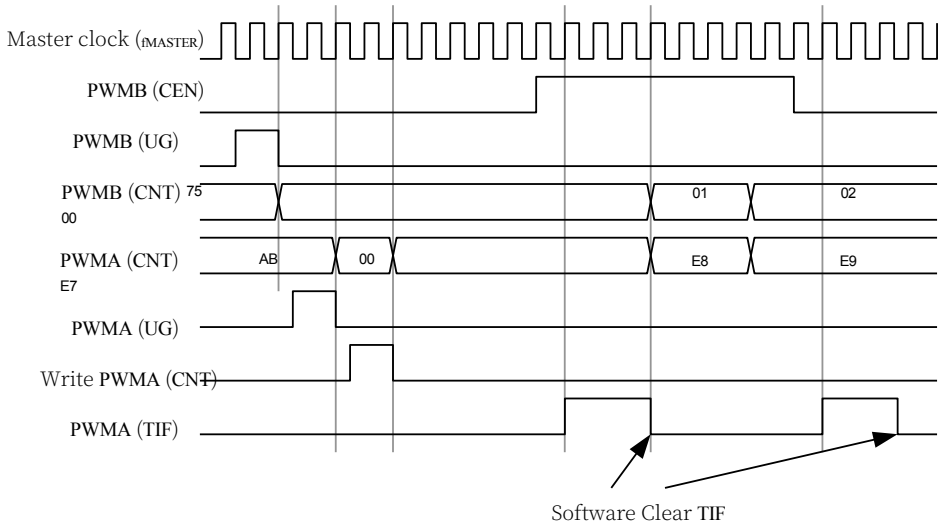


In the above figure, neither the counter nor the prescaler of the PWMA is initialised before starting, so they both start counting from the existing value. If the two timers are reset before starting the PWMB, the user can write the desired value to the PWMA counter to start counting from the specified value. The reset operation for PWMA can be achieved by software writing the UG bit of PWMA_EGR register.

In the following example, we synchronise PWMB and PWMA, PWMB is in master mode and starts counting from 0. PWMA is in trigger slave mode and starts counting from 0xE7. Both PWMs use the same crossover coefficient. When the CEN bit of PWMB_CR1 register is cleared, PWMB is disabled and PWMA stops counting.

1. Configure PWMB as the master mode to output the comparison output signal (OC5REF) as the trigger signal. (Configure MMS=100 of PWMB_CR2 register).
2. Configure the waveform of the OC5REF signal of the PWMB (PWMB_CMR1 register).
3. Configure PWMA to use the output of PWMB as its own trigger input signal (configure TS=010 of PWMA_SMCR register).
4. Configure the PWMA for gated trigger mode (configure SMS=101 in the PWMA_SMCR register).
5. Reset the PWMB by writing 1 to the UG bit (PWMB_EGR register).
6. Reset the PWMA by writing 1 to the UG bit (PWMA_EGR register).
7. Initialise the PWMA by writing 0xE7 to the PWMA counter (PWMA_CNTRL).
8. Enable PWMA by writing 1 to the CEN bit (PWMA_CR1 register).
9. PWMB is activated by writing 1 to the CEN bit (PWMB_CR1 register).

10. Stop PWMB by writing 0 to the CEN bit (PWMB_CR1 register).



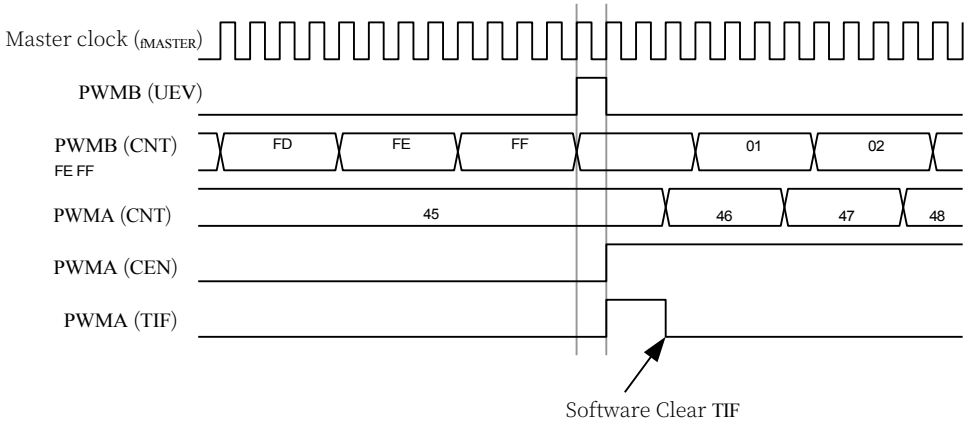
Starting PWMA with PWMB

In this example, we use the PWMB update event to start PWMA.

The PWMA starts counting from its existing value (which can be a non-zero value) according to the PWMA's own drive clock when an update event occurs in the PWMB. the PWMA automatically enables the CEN bit upon receiving a trigger signal and starts counting until the user writes a 0 to the CEN bit of the PWMA_CR1 register. both PWMs use a 4-division $_{fMASTER}$ as the drive clock (Both PWMs use a 4-division fMASTER as the drive clock ($_{fCK\_CNT}$ = fMASTER/4).

1. Configure PWMB to be the master mode and output the update signal (UEV). (Configure PWMB_CR2 register with MMS=010).
2. Configure the period of PWMB (PWMB_ARR register).
3. Configure PWMA to use the output of PWMB as the trigger signal for the input (configure TS=010 of PWMA_SMCR register).
4. Configure PWMA for trigger mode (configure SMS=110 for PWMA_SMCR register).
5. Setting the CEN bit (PWMB_CR1 register) starts PWMB.

The PWMB update event (PWMB-UEV) triggers the PWMA



As in the previous example, the user can also initialise

the counters before starting them. The **two PWMs are**

triggered synchronously with an external signal.

In this example, PWMB is enabled using the rising edge of TI1 and PWMA is enabled at the same time.To keep the timer aligned, PWMB

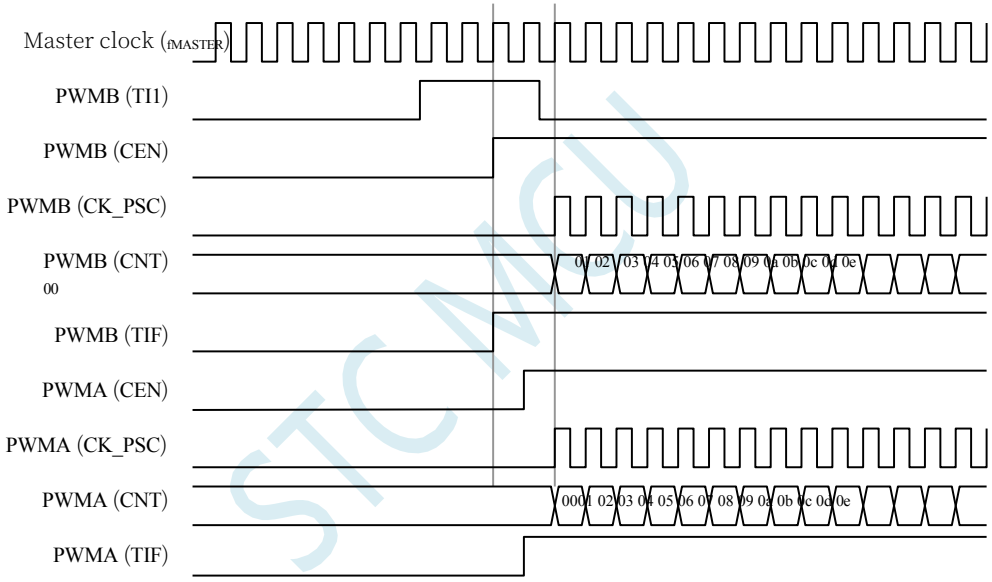
It needs to be configured in master/slave mode (slave mode for TI1 signals, master mode for PWMA).

1. Configure PWMB as master mode to output enable signal as PWMA trigger(configure MMS=001 of PWMB_CR2 register)
2. Configure PWMB to slave mode, using the TI1 signal as the trigger signal for the input (configure TS=100 of PWMB_SMCR register).
3. Configure the trigger mode of PWMB (configure SMS=110 for PWMB_SMCR register).
4. Configure the PWMB for master/slave mode (configure MSM=1 for the PWMB_SMCR register).
5. Configure PWMA to use the output of PWMB as the input trigger signal (configure TS=010 of PWMA_SMCR register).
6. Configure the trigger mode of PWMA (configure SMS=110 for PWMA_SMCR register).

When a rising edge occurs on TI1, both timers start counting synchronously and the TIF bit is set.

Note: In this example, both timers are initialised before starting (setting the UG bit), so they both start counting from 0. However, the user can also insert an offset by modifying the counter register (PWMA_CNT), in which case a delay will be inserted between the CK_PSC signal of the PWMB and the CNT_EN signal.

The TI1 signal of PWMB triggers PWMB and PWMA.



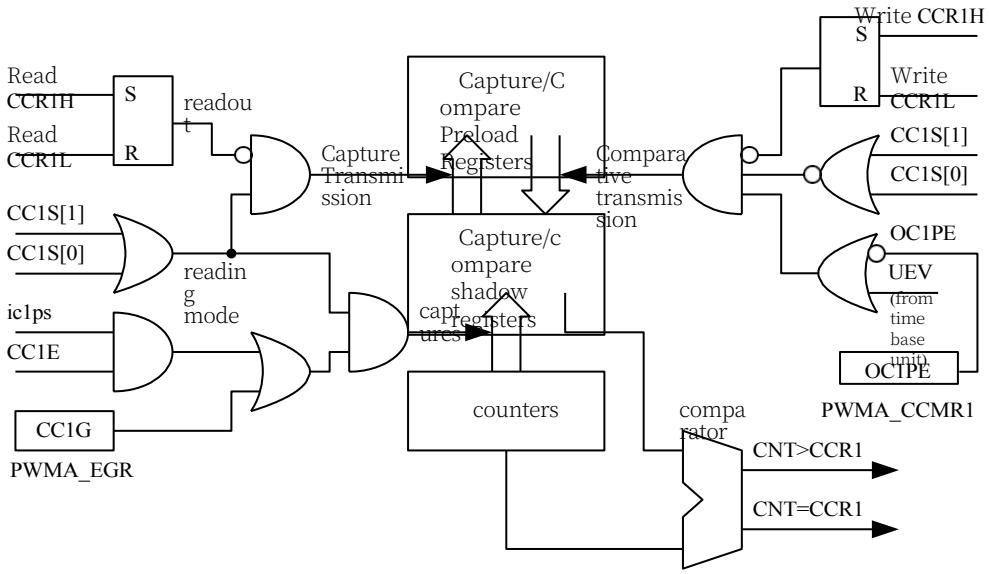
23.5 Capture/Compare Channel

PWM1P, PWM2P, PWM3P, PWM4P can be used as input capture, PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N can output comparison, this function can be configured by the capture/compare channel mode registers

(The CCiS channel selection bit of (PWMA_CMRi) is implemented, where i represents the number of channels 1~4.

Each capture/compare channel is built around a capture/compare register (containing shadow registers), including the input portion of the capture (digital filtering, multiplexing, and prescaler) and the output portion (comparator and output control).

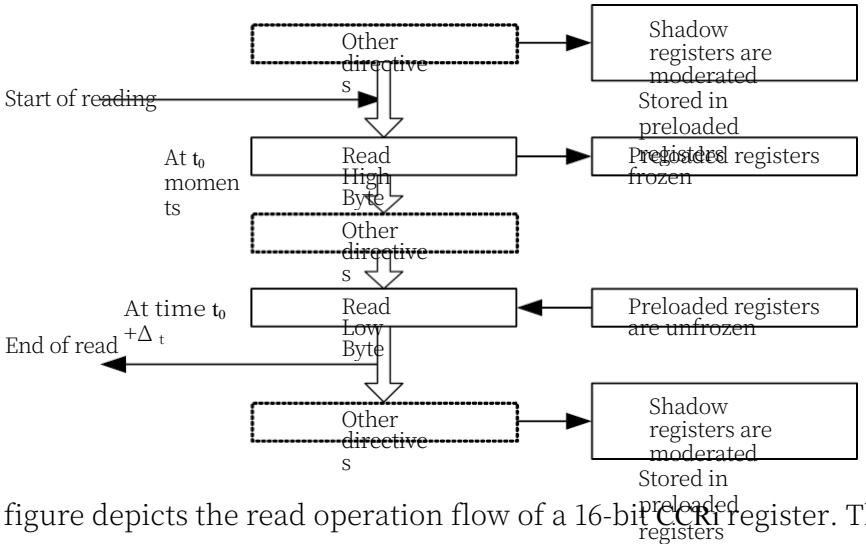
Main circuitry for capture/compare channel 1 (other channels are similar)



The capture/compare module consists of a preloaded register and a shadow register. The read/write process operates only on the preload register. In capture mode, the capture occurs on the shadow register, which is then copied to the preload register. In compare mode, the contents of the preload register are copied to the shadow register, and then the contents of the shadow register are compared to the counter.

The PWMA_CCRi register can be accessed at any time when the channel is configured in output mode (CCiS=0 in the PWMA_CCMRi register).

When the channel is configured in input mode, a read operation to the PWMA_CCRi register is similar to a counter read operation. When a capture occurs, the contents of the counter are captured into the PWMA_CCRi shadow register and later copied into the preload register. The preload register is frozen while the read operation is in progress.



The above figure depicts the read operation flow of a 16-bit CCRi register. The cached data will remain unchanged until the end of the read flow. At the end of the entire read process, if only the PWMA_CCRiL register is read, the low bit (LS) of the counter

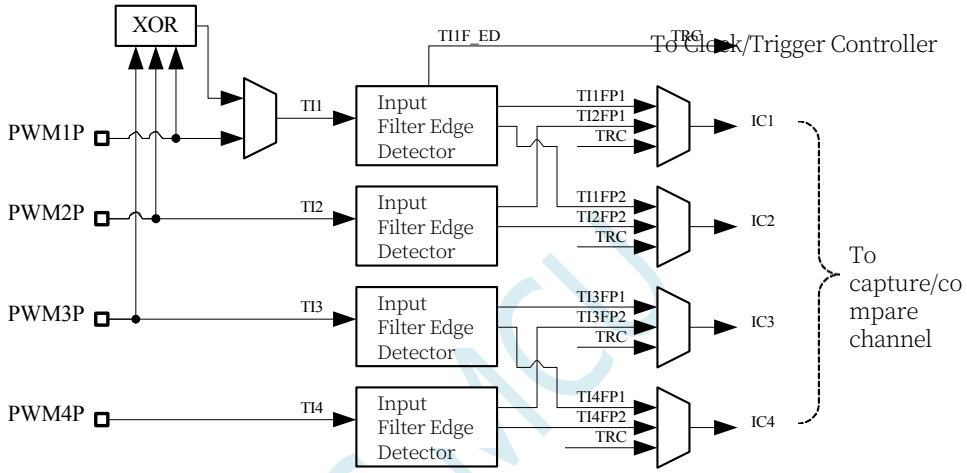
value is returned. If the high bit (MS) data is read after the low bit (LS) data is read, the same low bit data will not be returned.

23.5.1 16-bit PWMA_CCRi Register Write Flow

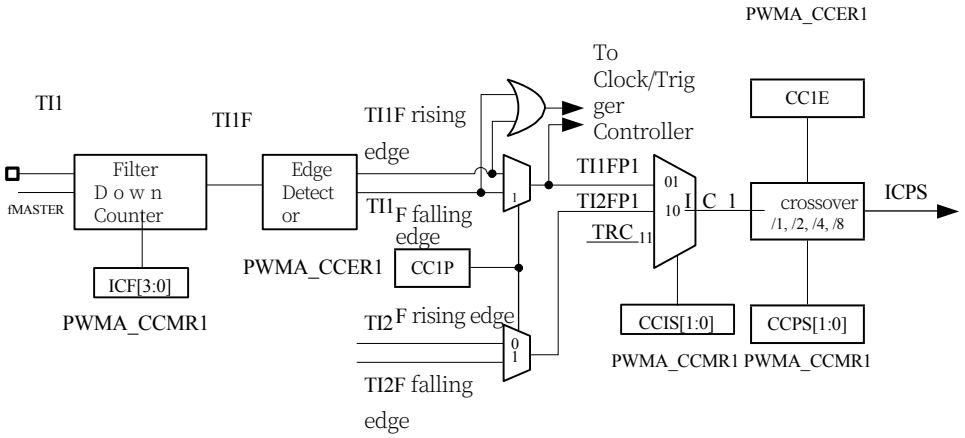
The write operation of the 16-bit PWMA_CCRi register is done through the preloaded register. Two instructions must be used to complete the entire process, one instruction corresponding to one byte. The high byte (MS) must be written first. While writing the high byte (MS), the update of the shadow register is disabled until the write operation of the low byte (LS) is completed.

23.5.2 input module

Block diagram of the input module



As shown in the figure, the input section samples the corresponding TIx input signal and generates a filtered signal, TIxF. An edge monitor with polarity selection then generates a signal (TIxFPx), which can be used as an input trigger to the trigger mode controller or as a capture control. This signal passes through a pre-divided frequency and enters the capture register (ICxPS).



23.5.3 Input capture mode

In input capture mode, the current value of the counter is latched to the capture/compare register when the corresponding edge on the ICi signal is detected

(PWMA_CCRx). When a capture event occurs, the corresponding CCiIF flag (PWMA_SR register) is set to 1.

If the CCiIE bit of the PWMA_IER register is set, i.e., an interrupt is enabled, an interrupt request will be generated. If the CCiIF flag is already high when a capture event occurs, the repeat capture flag CCiOF (PWMA_SR2 register) is set to 1. Writing CCiIF=0 or reading the capture data stored in the PWMA_CCRiL register clears CCiIF. writing CCiOF=0 clears CCiOF.

Capture on rising edge of PWM input signal

The following example illustrates how to capture the counter value into the PWMA_CCR1 register on the rising edge of the TI1 input as follows:

1. Select valid inputs: e.g. PWMA_CCR1 is connected to the TI1 input, so write to the PWMA_CCR1 register in the
   CC1S=01, at which point the channel is configured as an input and the PWMA_CCR1 register becomes read-only.
2. According to the characteristics of the input signal TIi, the filtering time of the corresponding input filter can be set by configuring the ICiF bit in the PWMA_CCMRi register. Assuming that the input signal is dithered for a maximum of 5 clock cycles, we have to configure the bandwidth of the filter to be longer than 5 clock cycles; therefore, we can sample 8 times consecutively to confirm a real edge shift on TI1, i.e., write IC1F=0011 in the TIMi_CCMR1 register, at this time, the signal is valid only if 8 consecutive samples of the same TI1 signal are received (the sampling frequency is f_{MASTER}).
3. To select the active conversion edge of the TI1 channel, write CC1P=0 (rising edge) in the PWMA_CCER1 register.
4. Configure the input prescaler. In this example, we want the capture to occur at every valid level transition moment, so the prescaler is disabled (write IC1PS=00 to the PWMA_CCMR1 register).
5. Setting CC1E=1 in the PWMA_CCER1 register allows the counter value to be captured into the capture register.
6. If required, allow the associated interrupt request by setting the CC1IE bit in the PWMA_IER register.

When an input capture occurs:

- When a valid level transition is generated, the counter value is transferred to the PWMA_CCR1 register.
- The CC1IF flag is set. CC1OF is also set to 1 when at least 2 consecutive captures have occurred and CC1IF has not been cleared.
- If the CC1IE bit is set, an interrupt is generated.

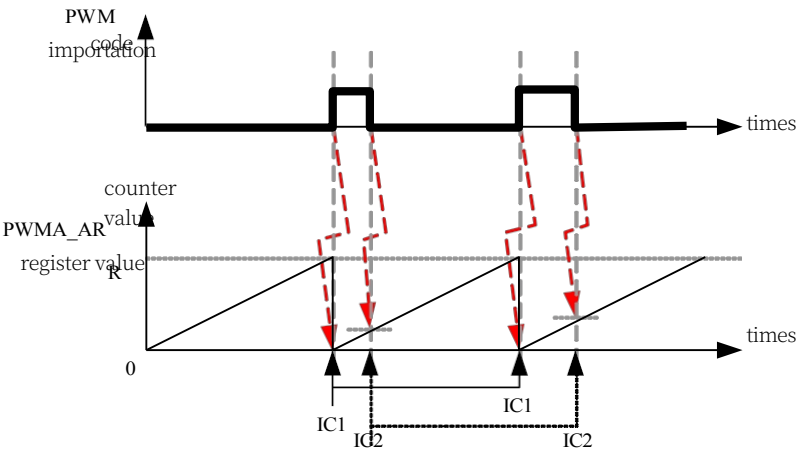
In order to handle capture overflow events (CC1OF bit), it is recommended that data be read before the duplicate capture flag is read, this is to avoid losing duplicate capture information that may be generated after the capture overflow flag is read and before the data is read.

Note: Setting the corresponding CCiG bit in the PWMA_EGR register allows you to generate an input capture interrupt through software.

PWM Input Signal Measurement

This mode is a special case of the Input Capture mode and operates the same as the Input Capture mode except for the following differences:

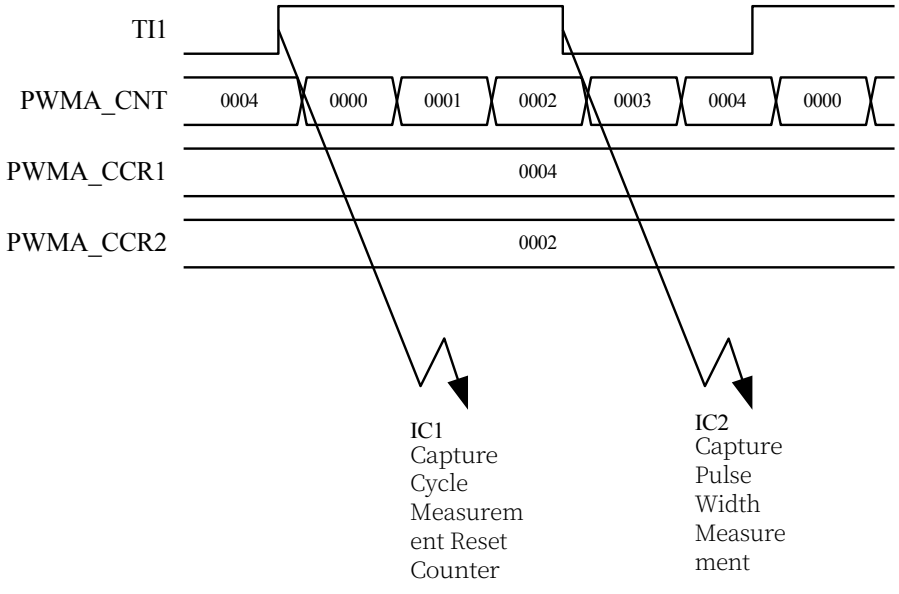
- Two ICi signals are mapped to the same TIi input.
- The polarity of the active edges of these two ICi signals is reversed.
- One of the TIiFP signals is used as the trigger input signal and the trigger mode controller is configured to reset the trigger mode.



For example, you can measure the period (PWMA_CCR1 register) and duty cycle of the PWM signal input on TI1 in the following way
(PWMA_CCR2 register). (depending on the frequency of the f_{MASTER} and the value of the prescaler)

1. Select valid input for PWMA_CCR1: Set CC1S=01 of PWMA_CCMR1 register (TI1 selected).
2. Select the active polarity of TI1FP1 (used to capture data into PWMA_CCR1 and clear the counter): set CC1P=0 (active on rising edge).
3. To select valid input for PWMA_CCR2: Set CC2S=10 of PWMA_CCMR2 register (TI1FP2 is selected).
4. Select the active polarity of TI1FP2 (capture data to PWMA_CCR2): set CC2P=1 (active on falling edge).
5. To select a valid trigger input signal: set TS=101 in PWMA_SMCR register (select TI1FP1).
6. Configure the trigger mode controller to reset trigger mode: set SMS=100 in PWMA_SMCR.
7. Enable Capture: Set CC1E=1 and CC2E=1 in PWMA_CCER1 register.

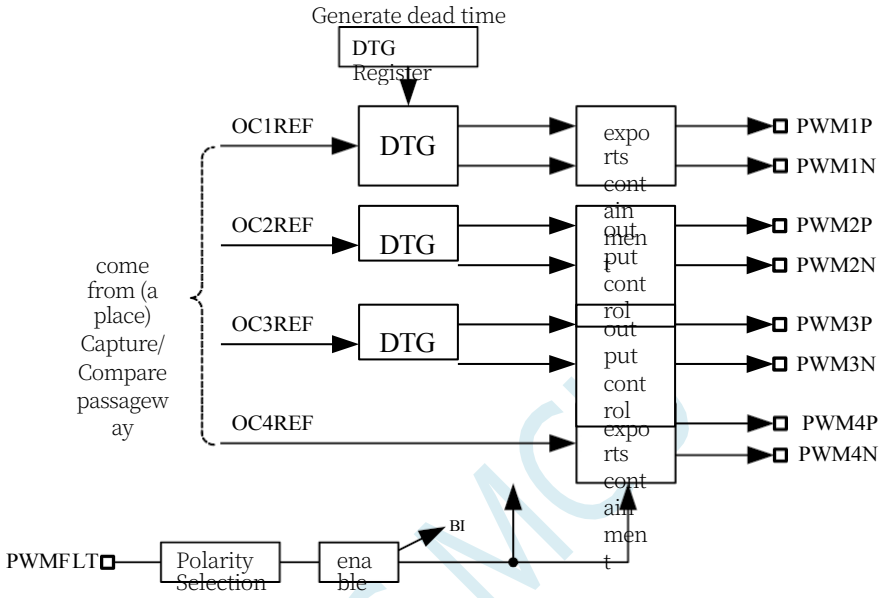
PWM Input Signal Measurement Example



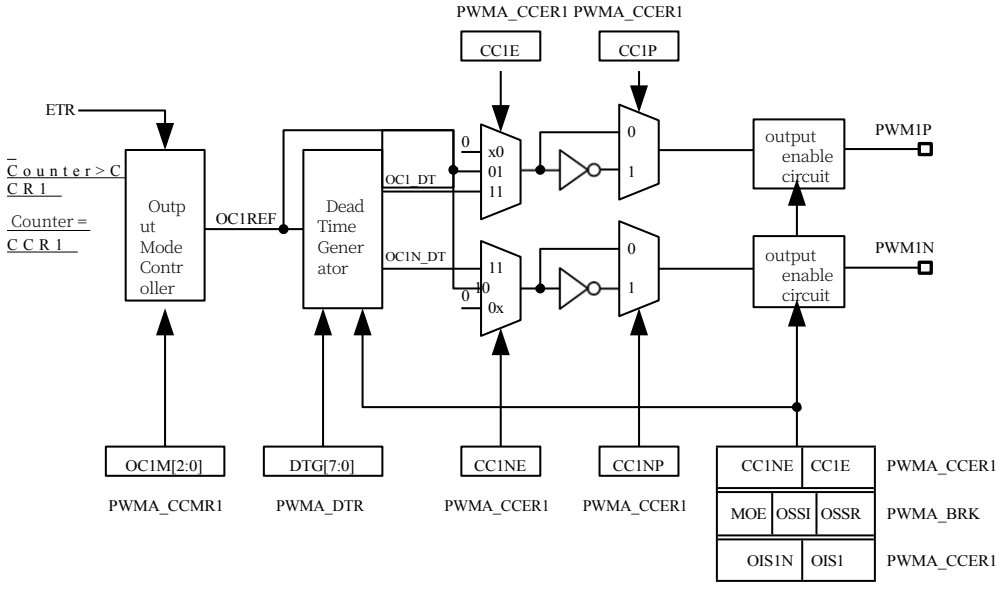
23.5.4 output module

The output module generates an intermediate waveform used as a reference, called OCiREF (highly effective). The brake function and polarity are handled at the end of the module.

Output Module Block Diagram



Channel 1 Detailed block diagram of output module with complementary outputs (similar for other channels)



23.5.5 forced output mode

In output mode, the output compare signal can be forced to a high or low state directly by software, independent of the result of the comparison between the output compare register and the counter.

Set OCiM=101 of PWMA_CCMRi register to force OCiREF signal high. Set OCiM=100 of PWMA_CCMRi register to force OCiREF signal to be low. Whether the output of OCi/OCiN is high or low depends on the CCiP/CCiNP polarity flag bit.

In this mode, the comparison between the PWMA_CCRi shadow registers and counters still takes place, the corresponding flags are modified, and the corresponding interrupts are still generated.

23.5.6 Output Comparison Mode

This mode is used to control an output waveform or to indicate that a given period of time has been reached. When the counter matches the contents of the capture/compare register, the following operations are available:

- Depending on the output comparison mode, the corresponding OCi output signal:
  - Unchanged (OCiM=000)
  - Set to active level (OCiM=001)
  - Set to invalid level (OCiM=010)
  - Flip (OCiM=011)
- Set the flag bit in the interrupt status register (CCiIF bit in the PWMA_SR1 register).
- An interrupt is generated if the corresponding interrupt enable bit (CCiIE bit in the PWMA_IER register) is set.

The OCiM bit of the PWMA_CCMRi register is used to select the output compare mode, while the CCiP bit of the PWMA_CCMRi register is used to select the valid and invalid level polarity.The OCiPE bit of the PWMA_CCMRi register is used to select whether the PWMA_CCRi register is required to use the preloaded register. In output compare mode, the update event UEV has no effect on the OCiREF and OCi outputs. The time precision is one count cycle of the counter. Output compare mode can also be used to output a single pulse.

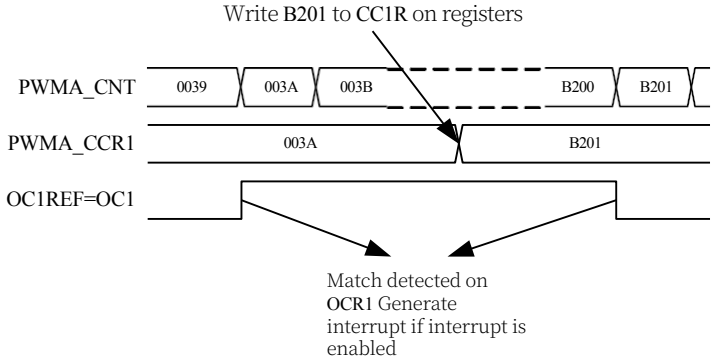
Outputs the configuration steps for the compare mode:

1. Select the counter clock (internal, external or prescaler).
2. Write the corresponding data into the PWMA_ARR and PWMA_CCRi registers.
3. To generate an interrupt request, set the CCiIE bit.
4. Select the output mode step:
   1. Set OCiM=011 to flip the output of the OCiM pin when the counter matches the CCRi
   2. Set OCiPE = 0 to disable preloaded registers
   3. Set CCiP = 0 to select high as active level
   4. Set CCiE = 1 to enable outputs

5. Set the CEN bit of the PWMA_CR1 register to start the counter.

The PWMA_CCRi register can be updated by software at any time to control the output waveform, provided that the preloaded registers are not used (OCiPE=0), otherwise the shadow registers of PWMA_CCRi can only be updated when the next update event occurs.

Output Compare Mode, Flip OC1



23.5.7 PWM Mode

Pulse Width Modulation (PWM) mode can generate a signal with a frequency determined by the PWMA_ARR register and a duty cycle determined by the PWMA_CCRi register.

Writing 110 (PWM mode 1) or 111 (PWM mode 2) to the OCiM bit in the PWMA_CCMRi register can independently set each OCi output channel to generate one PWM. the corresponding preload register must be enabled by setting the OCiPE bit in the PWMA_CCMRi register, and the preload register for auto-reload (in count-up mode or central symmetry mode) can also be enabled by setting the ARPE bit in the PWMA_CR1 register. The pre-loaded registers that can be automatically reloaded (in up-count mode or central symmetry mode) can also be set to the ARPE bit of the PWMA_CR1 register.

Since the preloaded registers are transferred to the shadow registers only when an update event occurs, all registers must be initialised by setting the UG bit of the PWMA_EGR register before the counter starts counting.

The polarity of the OCi can be set by software with the CCiP bit in the PWMA_CCERi register, which can be set to active-high or active-low.The output enable of the OCi is controlled by a combination of the CCiE, MOE, OISi, OSSR, and OSSI bits in the PWMA_CCERi and PWMA_BKR registers.

In PWM mode (Mode 1 or Mode 2), PWMA_CNT and PWMA_CCRi are always being compared (based on the counter counting direction) to determine if PWMA_CCRi≤PWMA_CNT or PWMA_CNT≤PWMA_CCRi is met.

Depending on the state of the CMS bit field in the PWMA_CR1 register, the timer is capable of generating an edge-aligned PWM signal or a centre-aligned PWM signal.

PWM Edge Aligned

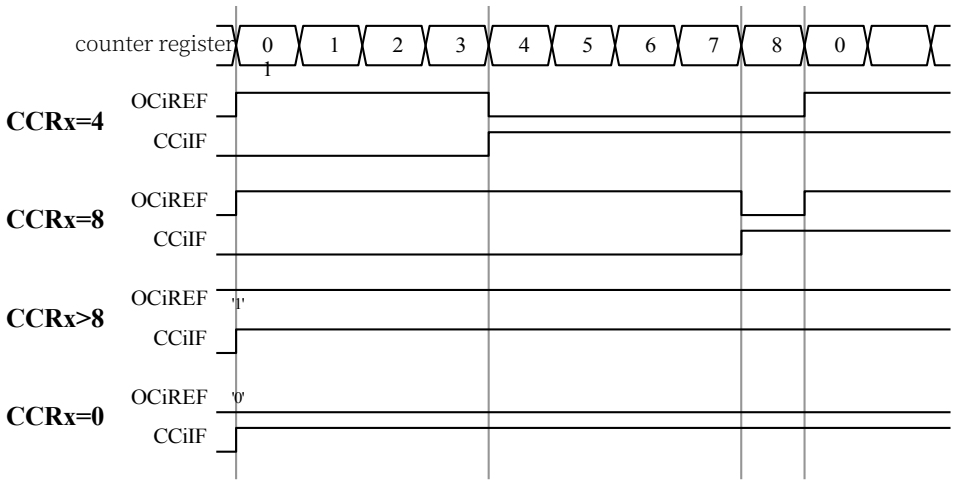
Mode Up Count

Configuration

When the DIR bit in the PWMA_CR1 register is 0, upward counting is performed.

The following is an example of PWM mode 1. The PWM reference signal OCiREF is high when PWMA_CNT<PWMA_CCRi, otherwise it is low. If the comparison value in PWMA_CCRi is greater than the auto-reload value (PWMA_ARR), OCiREF remains '1'. If the comparison value is 0, OCiREF remains '0'.

Edge aligned, PWM mode 1 waveform (ARR=8)



Down Count

Configuration

When the DIR bit of PWMA_CR1 register is 1, down count is executed.

In PWM mode 1, the reference signal OCiREF is low when PWMA_CNT>PWMA_CCRi, otherwise it is high. If the comparison value in PWMA_CCRi is greater than the auto-reload value in PWMA_ARR, OCiREF is kept as '1'. No 0% PWM waveform can be generated in this mode.

PWM Central Alignment Mode

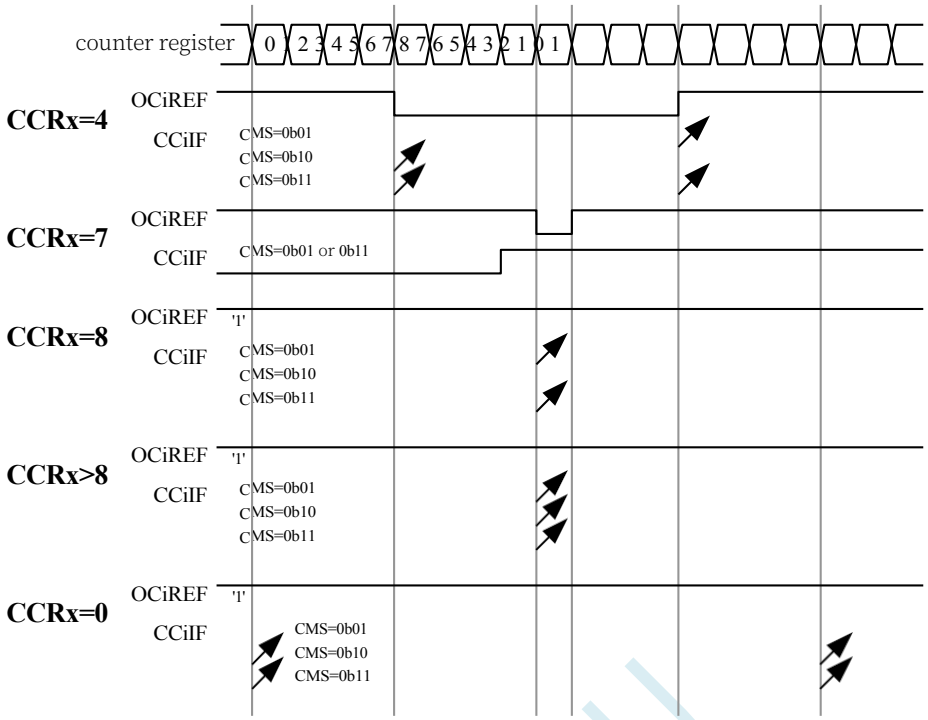
Central alignment mode when the CMS bit in the PWMA_CR1 register is not '00' (all other configurations have the same effect on the OCiREF/OCi signals).

Depending on the setting of the CMS bits, the compare flag can be set while the counter is counting up, counting down, or counting up and down.

1. The count direction bit (DIR) in the PWMA_CR1 register is updated by hardware; do not modify it with software.

Some examples of centre-aligned PWM waveforms are given below:

- PWMA_ARR=8
- PWM Mode 1
- The flag bit is set in the following three cases:
  - Only when the counter counts down (CMS=01)
  - Only when the counter is counting up (CMS=10)
  - During counter up and down counting (CMS=11)
- Centrally aligned PWM waveform (ARR=8)



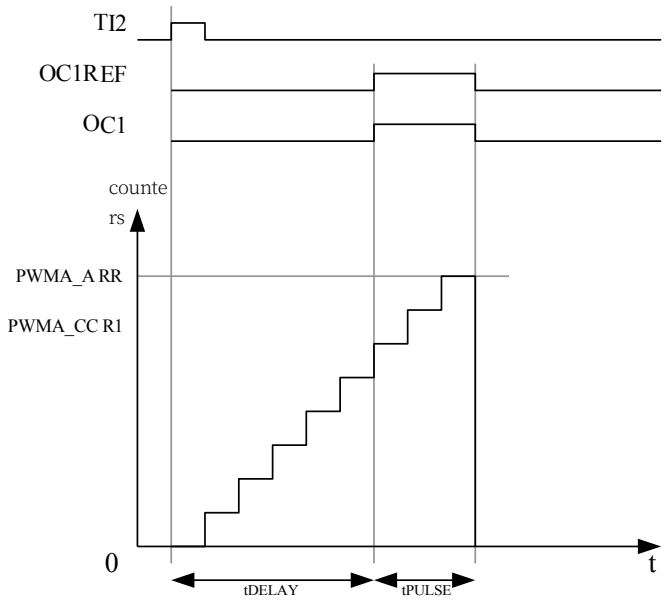
Single pulse mode

One-pulse mode (OPM) is a special case of one of the many modes described previously. This mode allows the counter to respond to an excitation and generate a pulse with a controlled pulse width after a programmable delay.

The counter can be activated by the clock/trigger controller to generate waveforms in output compare mode or PWM mode. Setting the OPM bit in the PWMA_CR1 register selects the single pulse mode, in which the counter automatically stops at the next update event UEV. A pulse is generated only when the comparison value is different from the initial value of the counter. Prior to startup (when the timer is waiting to be triggered), it must be configured as follows:

- Upward counting mode: counter CNT < CCRi ≤ ARR.
- Downward counting mode: Counter CNT > CCRi.

Example of a single pulse pattern



For example, delaying tDELAY after a rising edge is detected on the TI2 input pin generates a positive pulse of tPULSE width on OC1: (assuming IC2 is used as the trigger source for triggering channel 1)

- Set CC2S=01 in PWMA_CCMR2 register to map IC2 to TI2.
- Set CC2P=0 in PWMA_CCER1 register to enable IC2 to detect the rising edge.
- Set TS=110 of PWMA_SMCR register to enable IC2 to act as a trigger source (TRGI) for the clock/trigger controller.
- Set SMS=110 (trigger mode) of PWMA_SMCR register, IC2 is used to start the counter. the waveform of OPM is determined by the value written to the compare register (taking into account the clock frequency and the counter prescaler).
- t_{DELAY} is defined by the value in the PWMA_CCR1 register.
- t_{PULSE} is defined by the difference between the autoload value and the comparison value (PWMA_ARR - PWMA_CCR1).
- Assuming that a waveform from 0 to 1 is to be generated when a comparison match occurs, and a waveform from 1 to 0 is to be generated when the counter reaches the preloaded value, first set OCiM=111 in PWMA_CCMR1 register, enter PWM mode 2, selectively set OC1PE=1 in PWMA_CCMR1 register according to the need, set ARPE in PWMA_CR1 register to enable the preload register, then fill the comparison value in PWMA_CCR1 register, fill the auto-load value in PWMA_ARR register, and set UG bit to generate a preload register. ARPE in PWMA_CR1 register to enable the preload register, then fill in the comparison value in PWMA_CCR1 register, fill in the auto-load value in PWMA_ARR register, set the UG bit to generate an update event, and then wait for an external triggering event on TI2.

In this example, the DIR and CMS bits in the PWMA_CR1 register should be set low.

Since only one pulse is needed, set OPM=1 in the PWMA_CR1 register to stop counting at the next update event (when the counter flips from the autoload value to 0).

OCx fast enable (special case)

In single pulse mode, edge detection on the TIi input pin sets the CEN bit to start the counter, and then comparison operations between the counter and the comparison value produce the output of a single pulse. However, these operations require a certain number of clock cycles, so it limits the minimum delay tDELAY that can be obtained.

If you want to output the waveform with minimum delay, you can set the OCiFE bit in the PWMA_CCMRi register, which forces OCiREF (and OCx) to respond directly to the excitation without relying on the result of the comparison any more, and the output waveform is the same as that when the comparison is matched.OCiFE is only available when the channel is matched.

Functions when set to PWMA and PWMB modes.

Complementary outputs and deadband insertion

The PWMA is able to output two complementary signals and manages the instantaneous switching off and on of the outputs, a period of time commonly referred to as the dead time, which should be adjusted by the user according to the connected output devices and their characteristics (level shifting delays, power switching delays, etc.).

Configuring the CCiP and CCiNP bits in the PWMA_CCERi register allows the polarity to be selected independently for each output (main output OCi
(or complementary output OCiN).

The complementary signals OCi and OCiN are controlled by a combination of the following control bits: the CCiE and CCiNE bits in the PWMA_CCERi register and the MOE, OISi, OISiN, OSSI and OSSR bits in the PWMA_BKR register. In particular, the deadband control is activated on transition to the IDLE state (MOE decreases to 0).

Setting both the CCiE and CCiNE bits will insert a deadband, as well as the MOE bit if a brake circuit is present. Each channel has an 8-bit deadband generator. The reference signal OCiREF generates 2 outputs OCi and OCiN.

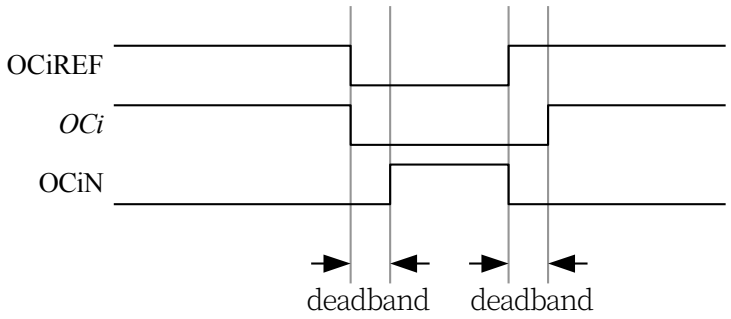
If OCi and OCiN are highly valid:

- The OCi output signal is the same as the reference signal, except that its rising edge is delayed relative to that of the reference signal.
- The OCiN output signal is the opposite of the reference signal, except that its rising edge has a delay relative to the falling edge of the reference signal.

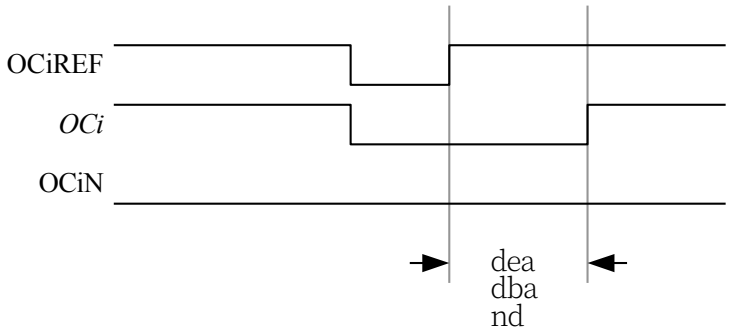
If the delay is greater than the currently valid output width (OCi or OCiN), the corresponding pulse is not generated.

The following graphs show the relationship between the output signal of the deadband generator and the current reference signal OCiREF. (Assuming CCiP=0, CCiNP=0, MOE=1, CCiE=1 and CCiNE=1)

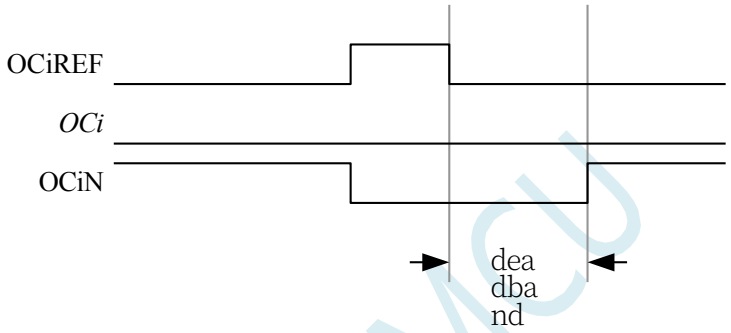
Complementary output with deadband insertion



Deadband waveform delay greater than negative pulse



Deadband waveform delay greater than positive pulse



The deadband delay is the same for each channel and is programmatically

configured by the DTG bit in the PWMA_DTR register. **Redirect OCiREF to OCi or**

OCiN

In output mode (forced output, output comparison, or PWM output), by configuring the PWMA_CCERi register's CCiE and

CCiNE bit, OCiREF can be redirected to the output of OCi or OCiN.

This function can send a special waveform (e.g. PWM or static active level) on one of the outputs when the complementary outputs are at an invalid level. Another function is to have both outputs at the same time at an invalid level, or at the same time at an active level (when they are still complementary outputs with deadband).

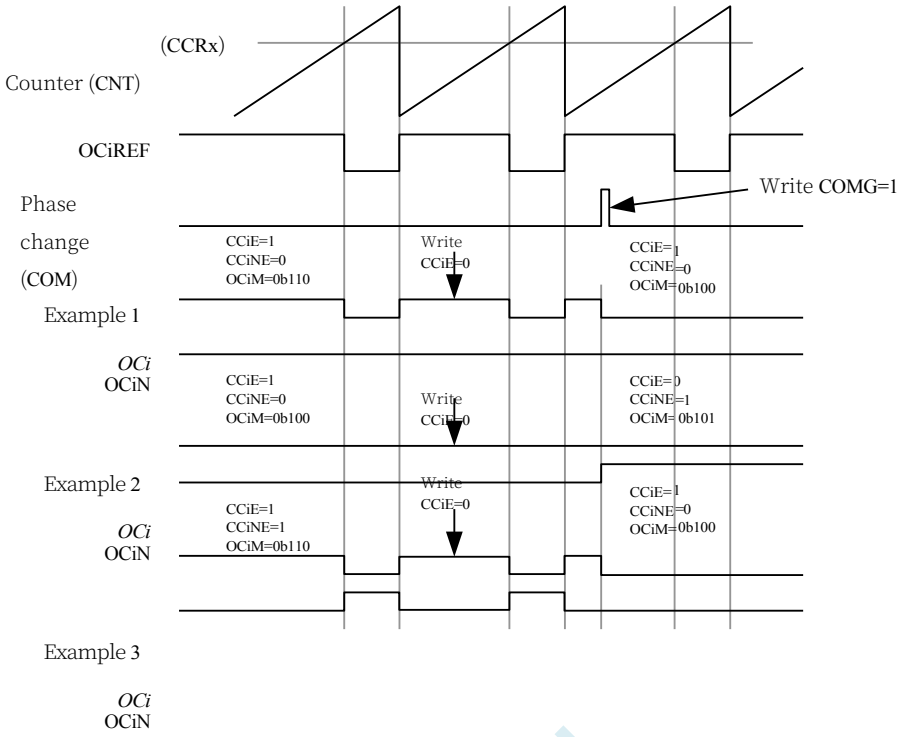
Note: When only OCiN is enabled (CCiE=0, CCiNE=1), it is not inverted and is effective immediately when OCiREF goes high. For example, if CCiNP=0, then OCiN=OCiREF. On the other hand, when both OCi and OCiN are enabled (CCiE=CCiNE=1), OCi is active when OCiREF is high; and OCiN, on the contrary, becomes active when OCiREF is low.

Six-step PWM output for motor control

When complementary outputs are required on a channel, the preload bits are OCiM, CCiE, and CCiNE. these preload bits are transferred to the shadow register bits when a COM phase change event occurs. This allows you to pre-set the next step configuration and repair change the configuration of all channels at the same time at the same moment.COM can be generated by software by setting the COMG bit in the PWMA_EGR register, or by hardware on the rising edge of TRGI.

The following figure shows the OCx and OCxN outputs for three different configurations when a COM event occurs.

Example of generating a six-step PWM, using COM (OSSR=1)



23.5.8 Using the brake function (PWMFLT)

The brake function is commonly used in motor control. When the brake function is used, the output enable signal and disable level are modified according to the corresponding control bits (MOE, OSSI and OSSR bits in PWMA_BKR register).

After system reset, the brake circuit is disabled and the MOE bit is low. Setting the BKE bit in the PWMA_BKR register enables the brake function. The polarity of the brake input signal can be selected by configuring the BKP bit in the same register.BKE and BKP can be modified simultaneously.

The MOE falling edge can be asynchronous with respect to the clock module, so a resynchronisation circuit is set up between the actual signal (acting on the output) and the synchronisation control bit (in the PWMA_BKR register). This resynchronisation circuit creates a delay between the asynchronous and synchronous signals. In particular, if MOE=1 is written when it is low, a delay (null instruction) must be inserted before it is read to get the correct value. This is because the write is to an asynchronous signal and the read is to a synchronous signal.

When braking occurs (a selected level appears at the brake input), the following actions are available:

- The MOE bit is cleared asynchronously, placing the output in an invalid, idle, or reset state (selected by the OSSI bit). This feature remains in effect when the MCU's oscillator is turned off.
- Once MOE=0, each output channel outputs the level set by the OISi bit of the PWMA_OISR

register. If OSSI=0, the timer no longer controls the output enable signal, otherwise the output enable signal is always high.

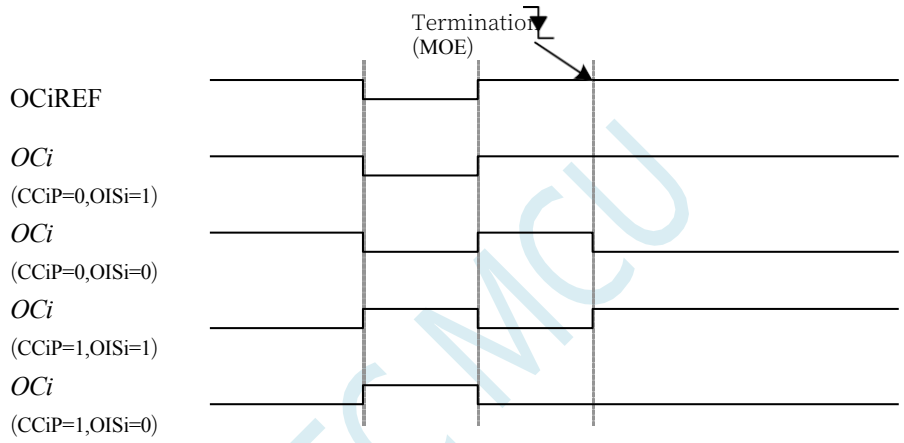
- When using complementary outputs:
  - The output is first placed in the reset state i.e. the invalid state (depending on polarity). This is asynchronous operation and is valid even when the timer is not clocked.
  - If the timer clock is still present, the deadband generator will re-activate and drive the output port after the deadband according to the levels indicated by the OISi and OISiN bits. Even in this case, OCi and OCiN cannot be driven to valid levels at the same time. Note: Because of the re-synchronisation of the MOE, the dead time is a bit longer than usual (about 2 clock cycles).
- If the BIE bit in the PWMA_IER register is set, an interrupt is generated when the brake status flag (BIF bit in the PWMA_SR1 register) is '1'.

- If the AOE bit in the PWMA_BKR register is set, the MOE bit is automatically set at the next update event UEV. This can be used for waveform control for example, otherwise the MOE remains low until it is set to '1' again. This feature can be used for safety purposes, where you can connect the brake input to a power-driven alarm output, thermal sensor or other safety device.

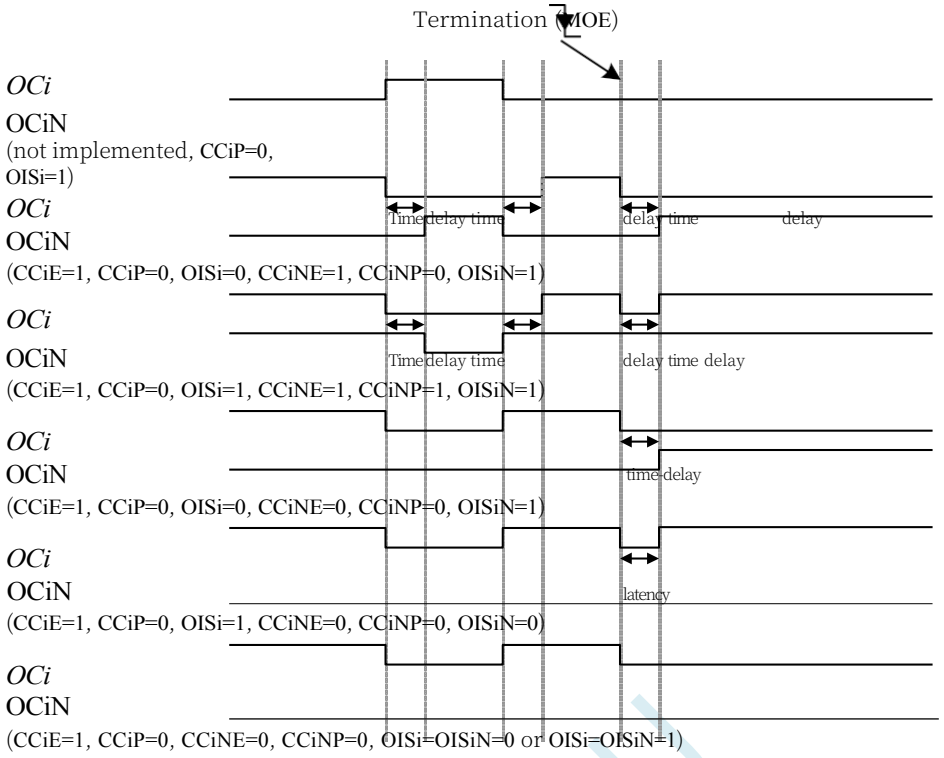
Note: The brake input is level active. Therefore, MOE cannot be set at the same time (automatically or via software) when the brake input is active, and the status flag BIF cannot be cleared.

The brakes are generated by the BRK input (BKIN), whose active polarity is programmable and enabled or disabled by the BKE bit of the PWMA_BKR register. In addition to the brake input and output management, write-protection is implemented in the brake circuit to secure the application. It allows the user to freeze several configuration parameters (OCi polarity and state when disabled, OCiM configuration, brake enable and polarity). The user can select one of the three levels of protection via the LOCK bit in the PWMA_BKR register. The LOCK bit field can only be modified once after MCU reset.

Outputs for brake response (channels without complementary outputs)



Outputs with complementary outputs for brake response (PWMA complementary outputs)



23.5.9 Clearing the OCiREF signal on an external event

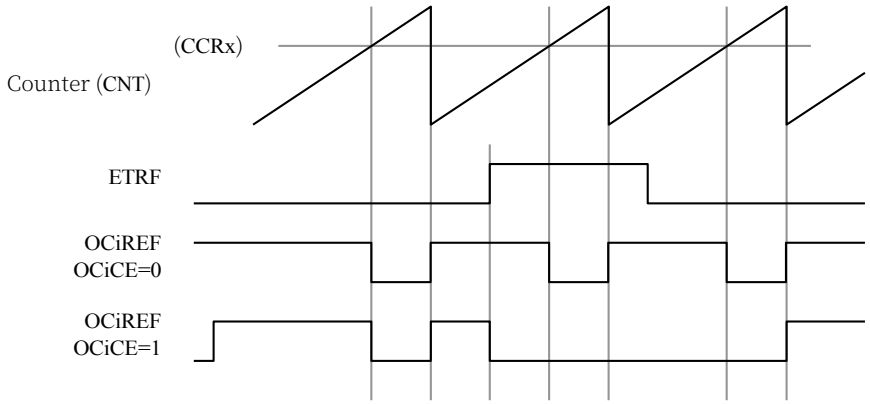
For a given channel, a high level at the ETRF input (setting the corresponding OCiCE bit in the PWMA_CCMRi register to '1') can pull the OCiREF signal low, and the OCiREF signal will remain low until the next update event UEV occurs. this function can only be used for the Output This function can only be used in Output Compare Mode and PWM Mode, but not in Forced Mode.

For example, the OCiREF signal can be linked to the output of a comparator to control the current. In this case, the ETR must be configured as follows:

1. The externally triggered prescaler must be off: ETPS[1:0]=00 in the PWMA_ETR register.
2. External clock mode 2 must be disabled: ECE=0 in the PWMA_ETR register.
3. The external trigger polarity (ETP) and external trigger filter (ETF) can be configured as required.

The following figure shows the action of the OCiREF signal when the ETRF input goes high, corresponding to different values of OCiCE. In this example, the timer PWMA is placed in PWM mode.

ETR Clear OCiREF of PWMA



23.5.10 Encoder Interface Mode

The encoder interface mode is

generally used for motor control.

The method of selecting the

encoder interface mode is:

- If the counter only counts on the edge of TI2, set SMS=001 in the PWMA_SMCR register;
- If counting only on the TI1 edge, set SMS=010;
- If the counter counts on both the TI1 and TI2 edges, set SMS=011.

The TI1 and TI2 polarity can be selected by setting the CC1P and CC2P bits in the PWMA_CCER1 register; the input filter can also be programmed if required.

Two inputs, TI1 and TI2, are used as the interface to the incremental encoder. Assuming that the counter has been activated (CEN=1 in the PWMA_CR1 register), the counter counts each time a valid hop is generated on either TI1FP1 or TI2FP2.TI1FP1 and TI2FP2 are the signals from TI1 and TI2 after passing them through the input filters and polarity control. Without filtering and polarity control, TI1FP1 = TI1 and TI2FP2 = TI2. The counting pulses and direction signals are generated according to the hopping sequence of the two input signals. Depending on the hopping sequence of the two input signals, the counter counts up or down, while the hardware sets the DIR bit of the PWMA_CR1 register accordingly. Whether the counter counts on TI1, on TI2, or on both TI1 and TI2, a trip on either input (TI1 or TI2) recalculates the DIR bit.

The encoder interface mode is basically equivalent to using an external clock with direction selection. This means that the counter only counts continuously between 0 and the auto-loaded value of the PWMA_ARR register (either 0 to ARR counting or ARR to 0 counting, depending on the direction) Therefore, PWMA_ARR must be configured before counting starts, and in this mode the capture, comparator, prescaler, repeat counter, trigger output characteristics, etc. still work as usual. Encoder mode and External Clock Mode 2 are not compatible and therefore cannot be operated at the same time.

In the encoder interface mode, the counter is automatically modified according to the speed and direction of the incremental encoder, so that the contents of the counter always indicate the position of the encoder and the direction of counting corresponds to the direction of rotation of the connected sensor.

The following table lists all possible combinations (assuming that TI1 and TI2 do not change at the same time). Counting direction versus encoder signal

active edge	Relative signal level (TI1FP1 corresponds to TI2, TI2FP2 corresponds to TI1)	TI1FP1 Signalling		TI2FP2 Signalling	
		go up	go down	go up	go down
TI1 count only	your (honorific)	down count	Upward counting	non-counting	non-counting
	lower (one's head)	Upward counting	down count	non-counting	non-counting

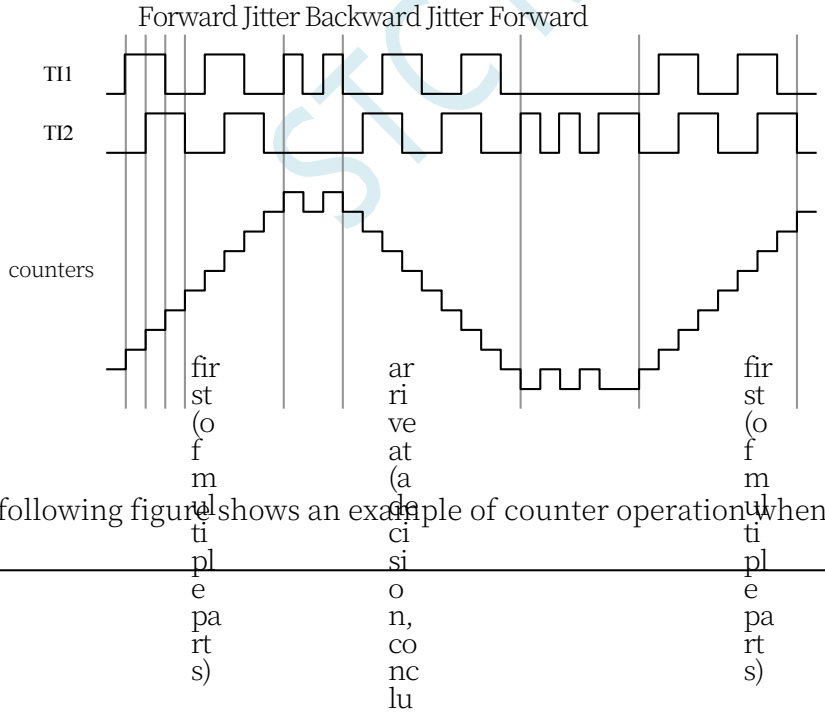
TI2 count only	your (honorific)	non-counting	non-counting	Upward counting	down count
	lower (one's head)	non-counting	non-counting	down count	Upward counting
Counting on TI1 and TI2	your (honorific)	down count	Upward counting	Upward counting	down count
	lower (one's head)	Upward counting	down count	down count	Upward counting

An external incremental encoder can be connected directly to the MCU without the need for external interface logic. However, a comparator is generally used to convert the differential output of the encoder to a digital signal, which greatly increases the immunity to noise interference. The third signal from the encoder output represents the mechanical zero point, which can be connected to an external interrupt input and trigger a counter reset.

Below is an example of counter operation showing the generation and direction control of the counting signal. It also shows how input jitter is suppressed when a double edge is selected; jitter may be generated when the sensor's position is close to a transition point. In this example, we assume the following configuration:

- CC1S=01 (PWMA_CCMR1 register, IC1FP1 mapped to TI1)
- CC2S=01 (PWMA_CMR2 register, IC2FP2 mapped to TI2)
- CC1P=0 (PWMA_CCER1 register, IC1 not inverted, IC1=TI1)
- CC2P=0 (PWMA_CCER1 register, IC2 not inverted, IC2=TI2)
- SMS=011 (PWMA_SMCR register, all inputs are valid on rising and falling edges).
- CEN=1 (PWMA_CR1 register, counter enable)

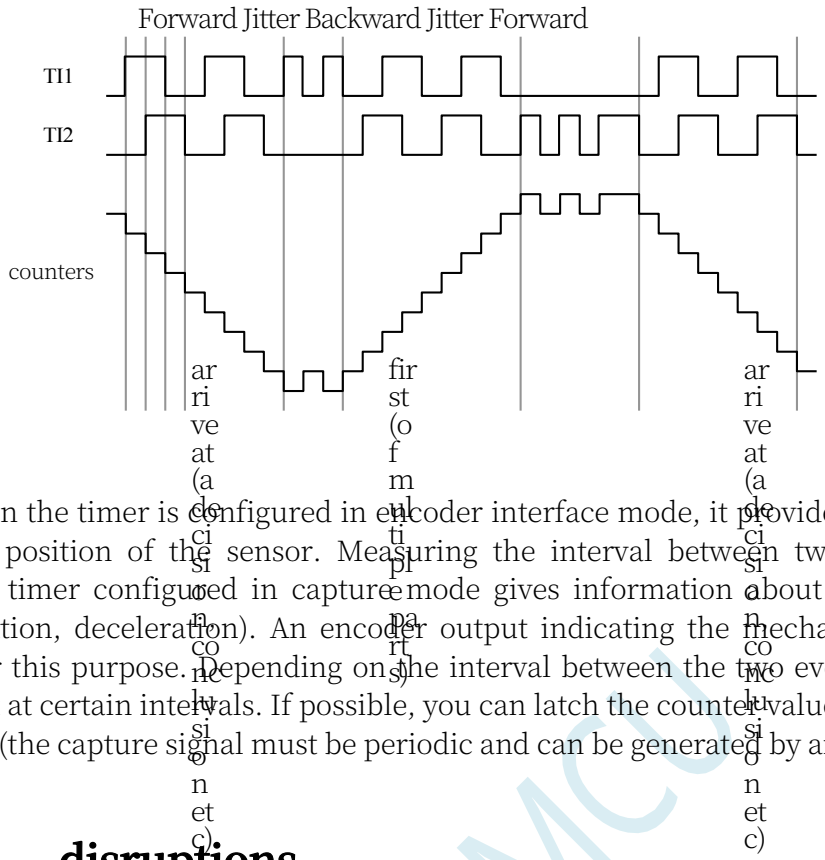
Example of counter operation in encoder mode



The following figure shows an example of counter operation when IC1 polarity is inverted

(CC1P=1, other configurations are the same as in the previous example)

Example of encoder interface mode for IC1 inversion



When the timer is configured in encoder interface mode, it provides information about the current position of the sensor. Measuring the interval between two encoder events using another timer configured in capture mode gives information about the dynamics (velocity, acceleration, deceleration). An encoder output indicating the mechanical zero point can be used for this purpose. Depending on the interval between the two events, the counter can be read out at certain intervals. If possible, you can latch the counter value to a third input capture register (the capture signal must be periodic and can be generated by another timer).

23.6 disruptions

PWMA/PWMB have 8 interrupt request sources each:

- Brake interruption
- trigger an interrupt
- COM event interrupt
- Input Capture/Output Compare 4 Interrupts
- Input Capture/Output Compare 3 Interrupts
- Input Capture/Output Compare 2 Interrupt
- Input Capture/Output Compare 1 Interrupt
- Update event interrupts (e.g. counter overflow, underflow and initialisation)

In order to use the interrupt feature, set the corresponding interrupt enable bits in the PWMA_IER/PWMB_IER registers: i.e., the BIE, TIE, COMIE, CCiIE, and UIE bits, for each interrupt channel being used. Each of the above interrupt sources can also be generated by software by setting the corresponding bits in the PWMA_EGR/PWMB_EGR registers.

23.7 PWMA/PWMB Register Description

23.7.1 Function pin switching (PWMx_PS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWMB_PS	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	

C1PS[1:0]: Advanced PWM channel 1 output pin select bits

C1PS[1:0]	PWM1P	PWM1N
00	P1.0	P1.1
01	P2.0	P2.1
10	P6.0	P6.1
11	-	-

C2PS[1:0]: Advanced PWM channel 2 output pin select bits

C2PS[1:0]	PWM2P	PWM2N
00	P1.2/P5.4[1]	P1.3
01	P2.2	P2.3
10	P6.2	P6.3
11	-	-

[1] ：有 P1.2 port, this function is on theP1.2 port, and for models without theP1.2 port, this function is on the P5.4 口上

C3PS[1:0]: Advanced PWM channel 3 output pin select bits

C3PS[1:0]	PWM3P	PWM3N
00	P1.4	P1.5
01	P2.4	P2.5
10	P6.4	P6.5
11	-	-

C4PS[1:0]: Advanced PWM channel 4 output pin select bits

C4PS[1:0]	PWM4P	PWM4N
00	P1.6	P1.7
01	P2.6	P2.7
10	P6.6	P6.7
11	P3.4	P3.3

C5PS[1:0]: Advanced PWM channel 5 output pin select bits

C5PS[1:0]	PWM5
00	P2.0
01	P1.7
10	P0.0
11	P7.4

C6PS[1:0]: Advanced PWM channel 6 output pin select bits

C6PS[1:0]	PWM6
00	P2.1
01	P5.4
10	P0.1

11	P7.5

C7PS[1:0]: Advanced PWM channel 7 output pin select bits

C7PS[1:0]	PWM7
00	P2.2
01	P3.3
10	P0.2
11	P7.6

C8PS[1:0]: Advanced PWM channel 8 output pin select bits

C8PS[1:0]	PWM8
00	P2.3
01	P3.4
10	P0.3
11	P7.7

23.7.2 Advanced PWM Function Pin Select Register (PWMx_ETRPS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETRPS	7EFEB0H						BRKAPS	ETRAPS[1:0]	
PWMB_ETRPS	7EFEB4H						BRKBPS	ETRBPS[1:0]	

ETRAPS[1:0]: External trigger pin ERI selection bit for advanced PWMA

ETRAPS [1:0]	PWMETI
00	P3.2
01	P4.1
10	P7.3
11	-

ETRBPS[1:0]: External trigger pin ERIB selection bits for advanced PWMBs

ETRBPS [1:0]	PWMETI2
00	P3.2
01	P0.6
10	-
11	-

BRKAPS: Brake Foot PWMFLT Selection Bit for Advanced PWMAs

BRKAPS	PWMFLT
0	P3.5
1	Comparator output

BRKBPS: Brake pin PWMFLT2 select bit for advanced PWMBs

BRKBPS	PWMFLT2
0	P3.5
1	Comparator output

23.7.3 Output enable register (PWMx_ENO)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ENO	7EFEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P

PWMB_ENO	7EFEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P

ENO8P: PWM8 output control bit
0: Disable PWM8 output
1: Enable PWM8 output
ENO7P: PWM7 output control bit 0: Disable PWM7 output
1: Enable PWM7 output
ENO6P: PWM6 output control bit 0: Disable PWM6 output
1: Enable PWM6 output
ENO5P: PWM5 output control bit 0: Disable PWM5 output
1: Enable PWM5 output
ENO4N: PWM4N output control bit 0: Disable PWM4N output
1: Enable PWM4N output
ENO4P: PWM4P output control bit 0: Disable PWM4P output
1: Enable PWM4P output
ENO3N: PWM3N output control bit 0: Disable PWM3N output
1: Enable PWM3N output
ENO3P: PWM3P output control bit 0: Disable PWM3P output
1: Enable PWM3P output
ENO2N: PWM2N output control bit 0: Disable PWM2N output
1: Enable PWM2N output
ENO2P: PWM2P output control bit 0: Disable PWM2P output
1: Enable PWM2P output
ENO1N: PWM1N output control bit 0: Disable PWM1N output
1: Enable PWM1N output
ENO1P: PWM1P output control bit 0: Disable PWM1P output
1: Enable PWM1P output

23.7.4 Output additional enable register (PWMx_IOAUX)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX	7EFEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P
PWMB_IOAUX	7EFEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P

AUX8P: PWM8 output additional control bits

0: PWM8 output directly controlled by ENO8P

1: The output of PWM8 is controlled by ENO8P and PWMB_BKR.

AUX7P: PWM7 output additional control bits

0: PWM7 output directly controlled by ENO7P

1: The output of PWM7 is controlled by ENO7P and PWMB_BKR.

AUX6P: PWM6 output additional control bits

0: PWM6 output directly controlled by ENO6P

1: The output of PWM6 is controlled by ENO6P and PWMB_BKR.

AUX5P: PWM5 output additional control bits

0: PWM5 output directly controlled by ENO5P

1: The output of PWM5 is controlled by ENO5P and PWMB_BKR.

AUX4N: PWM4N output additional control bits

0: Output of PWM4N directly controlled by ENO4N

1: The output of PWM4N is jointly controlled by ENO4N and PWMA_BKR

AUX4P: PWM4P output additional control bits

0: PWM4P output directly controlled by ENO4P

1: The output of PWM4P is controlled by both ENO4P and PWMA_BKR.

AUX3N: PWM3N output additional control bits

0: PWM3N output directly controlled by ENO3N

1: The output of PWM3N is controlled by both ENO3N and PWMA_BKR.

AUX3P: PWM3P output additional control bits

0: PWM3P output directly controlled by ENO3P

1: The output of PWM3P is controlled by both ENO3P and PWMA_BKR.

AUX2N: PWM2N output additional control bits

0: Output of PWM2N directly controlled by ENO2N

1: The output of PWM2N is jointly controlled by ENO2N and PWMA_BKR

AUX2P: PWM2P output additional control bits

0: PWM2P output directly controlled by ENO2P

1: The output of PWM2P is controlled by both ENO2P and PWMA_BKR.

AUX1N: PWM1N output additional control bits

0: PWM1N output directly controlled by ENO1N

1: The output of PWM1N is controlled by ENO1N and PWMA_BKR.

AUX1P: PWM1P output additional control bits

0: PWM1P output directly controlled by ENO1P

1: The output of PWM1P is controlled by ENO1P and PWMA_BKR.

23.7.5 Control Register 1 (PWMx_CR1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR1	7EFEC0H	ARPEA	CMSA[1:0]		DIRA	OPMA	URSA	UDISA	CENA
PWMB_CR1	7EFEE0H	ARPEB	CMSB[1:0]		DIRB	OPMB	URSB	UDISB	CENB

ARPEn: Automatic Reloading Permissible Element (n=A,B)

0: PWMn_ARR register is not buffered, it can be directly written to

1: PWMn_ARR register buffered by preloaded buffer

CMSn[1:0]: Selection of alignment mode (n= A,B)

CMSn[1:0]	registration mode	instructions
00	Edge Alignment Mode	The counter counts up or down according to the direction bit (DIR).
01	Central Alignment Mode 1	The counter alternately counts up and down. The output compare interrupt flag bit for channels configured as outputs is set to 1 only when the counter counts down.
10	Central Alignment Mode 2	The counter alternately counts up and down. The output compare interrupt flag bit for channels configured as outputs is set to 1 only when the counter counts up.
11	Central Alignment Mode 3	The counter alternately counts up and down. The Output Compare Interrupt Flag bit for channels configured as outputs is set to 1 for both counter up and counter down counts.

Note 1: With the counter on (CEN=1), switching from edge alignment mode to centre alignment mode is not allowed. Note 2: In Central Alignment Mode, Encoder Mode (SMS=001, 010, 011) must be disabled.

DIRn: Counter counting direction

(n= A,B) 0: Counter counts up;

1: The counter counts down.

Note: This bit is read-only when the counter is configured for Central Alignment Mode or Encoder Mode.

OPMn: Single pulse mode (n= A , B) 0:

Counter does not stop when an update event occurs;

1: When the next update event occurs, the CEN bit is cleared and the counter stops.

URSn: update request source (n= A,B)

0: If the UDIS is allowed to generate update events, either of the following events generates an update interrupt:

- Register is updated (counter overflow/underflow)
- Software Setting UG Bit
- Clock/trigger controller generated updates

1: If the UDIS is allowed to generate update events, an update interrupt is generated and the UIF is set to 1 only when the following events occur:

- Register is updated (counter overflow/underflow)

UDISn: update disabled (n= A,B)

0: Generate an update (UEV) event once the following events have occurred:

- Counter overflow/underflow
- Generate software update events

- Hardware reset generated by the clock/trigger mode controller The cached registers are loaded with their preloaded values.

1: No update event is generated and the shadow registers (ARR, PSC, CCRx) hold their values. If the UG bit is set or the clock/trigger controller issues a hardware reset, the counter and prescaler are reinitialised.

CENn: Counter allowed (n=

A,B) 0: Counter disabled;

1: Enable the counter.

Note: External Clock, Gated Mode and Encoder Mode can only work after the CEN bit has been set in software. Trigger mode, however, can automatically set the CEN bit in hardware.

23.7.6 Control Register 2 (PWMx_CR2), and the real-time trigger ADC

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR2	7EFEC1H	TI1S	MMSA [2:0]			-	COMSA	-	CCPCA
PWMB_CR2	7EFEE1H	TI5S	MMSB[2:0]			-	COMSB	-	CCPCB

TI1S: TI1 selection for the first PWM/PWMA group

0: PWM1P input pin connected to TI1 (input for digital filter);

1: PWM1P, PWM2P, and PWM3P pins are connected to TI1 of the first PWM via heterodyne. TI5S: TI5 selection of the second PWM/PWMB.

0: PWM5 input pin is connected to TI5 (input for digital filter);

1: PWM5, PWM6 and PWM7 pins are connected to TI5 of the second set of PWMs after heterodyne. MMSA[2:0]: Master Mode Selection

MMSA [2:0]	Master Mode	instructions
000	reset (a dislocated joint, an electronic device etc)	The UG bit of the PWMA_EGR register is used as a trigger output (TRGO). If the trigger input (clock/trigger controller configured in reset mode) generates a reset bit, the signal on the TRGO is delayed relative to the actual reset
001	enable	The counter enable signal is used as a trigger output (TRGO). Its used to start the ADC in order to control the enabling of the ADC over a period of time.The counter enable signal is generated by a logical or of the CEN control bit and the trigger input signal in gated mode. Unless master/slave mode is selected, there is a delay on TRGO when the counter enable signal is controlled by the trigger input. **Note: When you need to use PWM to trigger ADC conversion, you need to set ADC_POWER, ADC_CHS and ADC_EPWMT in ADC_CONTR register, and when PWM generates the internal signal of TRGO, the system will automatically set ADC_START to start ADC conversion. For details, please refer to the sample program ""CEN start using PWM". PWMA timer to trigger ADC in real time"**
010	update	The update event is selected as a trigger output (TRGO)
011	Comparison Pulse	Once a capture or a successful comparison has occurred, when the CC1IF flag is set to 1

		When the trigger output sends a positive pulse (TRGO)
100	comparisons	The OC1REF signal is used as a trigger output (TRGO)
101	comparisons	The OC2REF signal is used as a trigger output (TRGO)
110	comparisons	The OC3REF signal is used as a trigger output (TRGO)
111	comparisons	The OC4REF signal is used as a trigger output (TRGO)

MMSB[2:0]: Master mode selection

MMSB[2:0]	Master Mode	instructions
000	reset (a dislocated joint, an electronic device etc)	The UG bit of the PWMB_EGR register is used as a trigger output (TRGO). If the trigger input (clock/trigger controller configured in reset mode) generates a reset bit, the signal on the TRGO is delayed relative to the actual reset
001	enable	The counter enable signal is used as a trigger output (TRGO). It is used to start the The counter enables multiple PWMs for control over a period of time from PWMs.

		The enable signal is generated by the logic or of the CEN control bit and the trigger input signal in gated mode. Unless master/slave mode is selected, when the counter enable signal There is a delay on the TRGO when controlled by a trigger input.
010	update	The update event is selected as a trigger output (TRGO)
011	Comparison Pulse	Once a capture or a successful comparison has occurred, when the CC5IF flag is set to 1 When the trigger output sends a positive pulse (TRGO)
100	comparisons	The OC5REF signal is used as a trigger output (TRGO)
101	comparisons	The OC6REF signal is used as a trigger output (TRGO)
110	comparisons	The OC7REF signal is used as a trigger output (TRGO)
111	comparisons	The OC8REF signal is used as a trigger output (TRGO)

Note: Only the TRGO of the first PWM (PWMA) can be used to trigger the startup ADC.
Note: Only the TRGO of the second PWM (PWMB) group can be used for the ITR2 of the first PWM (PWMA) group.

COMSn: update control selection for capture/compare control bits (n=A,B)

0: When CCPCn=1, these control bits are updated only when COMG is set to 1

1: When CCPCn=1, these control bits are updated only when COMG position 1 or a rising edge of TRGI occurs

CCPCn: Capture/compare preload control bits (n= A,B) 0:

CCIE, CCINE, CCiP, CCiNP and OCIM bits are not preloaded

1: The CCIE, CCINE, CCiP, CCiNP, and OCIM bits are preloaded; when this bit is set, they are only available when COMG is set.

bit after being updated.

Note: This bit only works for channels with complementary outputs.

23.7.7 Slave Mode Control Register (PWMx_SMCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SMCR	7EFEC2H	MSMA	TSA [2:0]			-	SMSA[2:0]		
PWMB_SMCR	7EFEE2H	MSMB	TSB[2:0]			-	SMSB[2:0]		

MSMn: Master/Slave mode

(n= A,B) 0: no effect

1: Events on the trigger input (TRGI) are delayed to allow perfect synchronisation between PWMn and its slave PWM (via TRGO) TSA[2:0]: Trigger source selection

TSA [2:0]	trigger source
000	-
001	-

010	Internal trigger ITR2
011	-
100	Edge detector for TI1 (TI1F_ED)
101	Filtered Timer Input 1 (TI1FP1)
110	Filtered Timer Input 2 (TI2FP2)
111	External Trigger Input (ETRF)

TSB[2:0]: Trigger source selection

TSB[2:0]	trigger source

000	-
001	-
010	-
011	-
100	Edge Detector for TI5 (TI5F_ED)
101	Filtered Timer Input 1 (TI5FP5)
110	Filtered Timer Input 2 (TI5FP6)
111	External Trigger Input (ETRF)

Note: These bits can only be changed at SMS=000 to avoid false edge detection on change.

SMSA[2:0]: clock/trigger/slave mode selection

SMSA[2:0]	functionality	instructions
000	Internal clock mode	If CEN=1, the prescaler is driven directly by the internal clock
001	Encoder mode 1	Depending on the level of TI1FP1, the counter counts up/down on the edge of TI2FP2
010	Encoder Mode 2	Depending on the level of TI2FP2, the counter counts up/down on the edge of TI1FP1
011	Encoder Mode 3	Depending on the level of the other input, the counter is at the edge of TI1FP1 and TI2FP2 Up/Down Count
100	reset mode	Reinitialise the counter on the rising edge of the selected trigger input (TRGI). and generates a signal to update the register
101	Door control mode	When the trigger input (TRGI) is high, the counter is clocked on. Once the trigger input goes low, the counter stops (but does not reset). Counter Start Movements and stops are controlled
110	Trigger Mode	The counter is started (but not reset) on the rising edge of the trigger input TRGI, and only the Counter activation is controlled.
111	External clock Mode 1	The rising edge of the selected trigger input (TRGI) drives the counter. Note: Do not use the gating mode if TI1F_ED is selected as the trigger input (TS=100). This is because TI1F_ED only outputs one each time TI1F changes The gated mode, however, is to check the level of the trigger input

SMSB[2:0]: clock/trigger/slave mode selection

SMSB[2:0]	functionality	clarification
000	Internal clock mode	If CEN=1, the prescaler is driven directly by the internal clock
001	Encoder mode 1	Depending on the level of TI5FP5, the counter counts up/down on the edge of TI6FP6
010	Encoder Mode 2	Depending on the level of TI6FP6, the counter counts up/down on the edge of TI5FP5
011	Encoder Mode 3	Depending on the level of the other input, the counters at the edges of TI5FP5 and TI6FP6 move towards the Up/Down Count
100	reset mode	Reinitialise the counter on the rising edge of the selected trigger input (TRGI). and generates a signal to update the register
101	Door control mode	When the trigger input (TRGI) is high, the counter's clock turns on. Once the trigger

		and stops are controlled
110	Trigger Mode	The counter is started (but not reset) on the rising edge of the trigger input TRGI, and only the counting The start-up of the counting machine is controlled.
111	External clock mode 1	The rising edge of the selected trigger input (TRGI) drives the counter. Note: Do not use gating mode if TI5F_ED is selected as the trigger input (TS=100). This is because TI5F_ED only outputs one each time the TI5F changes The gated mode, however, is to check the level of the trigger input

23.7.8 External trigger register (PWMx_ETR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETR	7EFEC3H	ETP1	ECEA	ETPSA[1:0]		ETFA [3:0]			
PWMB_ETR	7EFEE3H	ETP2	ECEB	ETPSB[1:0]		ETFB[3:0]			

ETPn: Polarity of external trigger ETR
(n= A,B) 0: High or rising edge active
1: Active low or falling edge

ECEn: External clock enable
(n= A,B) 0: Disable external clock mode 2
1: Enable external clock mode 2, the counter is clocked at the active edge of ETRF.

Note 1: ECE set to 1 has the same effect as selecting External Clock Mode 1 which connects the TRGI to the ETRF (PWMn_SMCR register, SMS=111, TS=111).

Note 2: External Clock Mode 2 can be used in conjunction with the following modes: Trigger Standard Mode; Trigger Reset Mode; Trigger Gated Mode. However, TRGI must not be connected to ETRF at this time (TS cannot be 111 in PWMn_SMCR register).

Note 3: External Clock Mode 1 and External Clock Mode 2 are enabled at the same time, and the external clock input is ETRF.

ETPSn: External Trigger Prescaler The frequency of the external trigger signal EPRP must not exceed a maximum of fMASTER/4. A prescaler can be used to reduce the frequency of the ETRP, which is useful when the frequency of the EPRP is very high: (n= A,B)
00: Prescaler off
01: Frequency of EPRP/2
02: Frequency of EPRP/4
03: Frequency of EPRP/8

ETFn[3:0]: external trigger filter selection, this bit field defines the sampling frequency and digital filter length of the ETRP. (n=A,B)

ETFn[3:0]	Number of clocks	ETF[3:0]	Number of clocks
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160

23.7.9 Interrupt Enable Register (PWMx_IER)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	7EFEC4H	BIEA	TIEA	COMIEA	CC4IE	CC3IE	CC2IE	CC1IE	UIEA
PWMB_IER	7EFEE4H	BIEB	TIEB	COMIEB	CC8IE	CC7IE	CC6IE	CC5IE	UIEB

BIEn: Brake interrupt allowed
(n= A,B) 0: Brake interrupt prohibited;
1: brake interrupt allowed.
TIE: Trigger interrupt enable (n= A,B) 0: Trigger interrupt disabled;
1: Enable trigger interrupt.
COMIE: Enable COM interrupt (n= A,B)
0: Disable COM interrupt;
1: Allow COM interrupt.
CCnIE: Allow capture/compare n interrupts (n=1,2,3,4,5,6,7,8)
0: Capture/compare n interrupts are disabled;
1: Allow capture/compare n interrupts.
UIEn: update interrupt allowed
(n= A,B) 0: update interrupt disabled;
1: Allow updates to be interrupted.

23.7.10 Status Register 1 (PWMx_SR1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	7EFEC5H	BIFA	TIFA	COMIFA	CC4IF	CC3IF	CC2IF	CC1IF	UIFA
PWMB_SR1	7EFEE5H	BIFB	TIFB	COMIFB	CC8IF	CC7IF	CC6IF	CC5IF	UIFB

BIFn: Brake Interrupt Flag. Once the brake input is valid, the bit is 1 by hardware. if the brake input is not valid, the bit can be cleared 0 by software.(n= A,B)
0: No brake events generated
1: Valid level detected on brake input TIFn: Trigger interrupt flag. This position is 1 by hardware when a trigger event occurs. cleared 0 by software.(n= A,B)
0: No trigger event generated
1: Trigger interrupt and wait for response
COMIFn: COM interrupt flag. This bit is set to 1 by hardware and cleared to 0 by software once a COM event is generated (n=A,B) 0: no COM event generated
1: COM interrupt waiting for response
CC8IF: capture/compare 8 interrupt marker, refer to CC1IF description CC7IF: capture/compare 7 interrupt marker, refer to CC1IF description CC6IF: capture/compare 6 interrupt marker, refer to CC1IF description CC5IF:

capture/compare 5 interrupt marker, refer to CC1IF description CC4IF: capture/compare 4 interrupt marker, refer to CC1IF description CC3IF: capture/compare 3 interrupt marker, refer to CC1IF description CC2IF: capture/compare 2 interrupt marker, refer to CC1IF description CC1IF: capture/compare 1 interrupt marker.

If channel CC1 is configured for output mode:

This bit is set to 1 by hardware when the counter value matches the comparison value, except in centre-symmetric mode. It is cleared to 0 by software.

0: No match occurred;

1: The value of PWMA_CNT matches the value of PWMA_CCR1.

NOTE: In Centrosymmetric mode, when the counter value is 0, it counts up, and when the counter value is ARR, it counts down (it counts up from 0 to ARR-1, and then down from ARR to 1). Therefore, for all SMS bit values, neither value is flagged. However, if CCR1>ARR, CC1IF is set to 1 when the CNT reaches the ARR value.

If channel CC1 is configured for input mode:

This bit is set to 1 by hardware when a capture event occurs and it is cleared to 0 by software or by reading PWMA_CCR1L.

0: No input capture generation

1: Counter value has been captured to PWMA_CCR1

UIFn: Update Interrupt Flag This bit is set to 1 by hardware when an update event is generated. it is cleared to 0 by software.(n=A,B) 0: No update event generated

1: Update event wait response. This bit is set to 1 by hardware when the register is updated

- If UDIS=0 in PWMn_CR1 register, when the counter overflows or underflows
- If UDIS=0 and URS=0 in PWMn_CR1 register, when setting the UG bit of PWMn_EGR register, the software reinitialises the counter CNT.
- If UDIS=0 and URS=0 in PWMn_CR1 register, when counter CNT is re-initialised by trigger event

23.7.11 Status Register 2 (PWMx_SR2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR2	7EFEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-
PWMB_SR2	7EFEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

CC8OF: Capture/Compare 8 Repeat Capture Marker. See CC1OF description. CC7OF: Capture/compare 7 duplicate capture marks. See CC1OF description. CC6OF: Capture/compare 6 duplicate capture marks. See CC1OF description. CC5OF: Capture/compare 5 repeat capture markers. See CC1OF description. CC4OF: Capture/compare 4 repeat capture markers. See CC1OF description. CC3OF: Capture/compare 3 repeat capture markers. See CC1OF description. CC2OF: Capture/Compare 2 repeat capture markers. See CC1OF description.

CC1OF: Capture/Compare 1 Repeat Capture Flag. This marker can be set by hardware to 1 only if the corresponding channel is configured for input capture. write 0 can be
Clear this bit.

0: No duplicate captures are generated;

1: The state of CC1IF is already 1 when the counter value is captured into the PWMA_CCR1 register.

23.7.12 Event Generation Register (PWMx_EGR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_EGR	7EFEC7H	BGA	TGA	COMGA	CC4G	CC3G	CC2G	CC1G	UGA
PWMB_EGR	7EFEE7H	BGB	TGB	COMGB	CC8G	CC7G	CC6G	CC5G	UGB

BGn: Generate a brake event. This bit is set to 1 by software to generate a brake event, which is automatically cleared by hardware 0 (n= A,B) 0: no action

1: Generate a brake event. At this time, MOE=0, BIF=1, if the corresponding interrupt is turned on (BIE=1), the corresponding interrupt will be generated TGn: Generate a trigger event. This bit is set to 1 by software to generate a trigger event, which is automatically cleared by hardware (n= A,B)

0: no action

1: TIF=1, if the corresponding interrupt is turned on (TIE=1), the corresponding interrupt is generated

COMGn: Capture/compare events to generate control updates. This bit is set to 1 by software and cleared automatically by hardware (n= A,B) 0: no action

1: CCPC=1 allows the CCIE, CCINE, CCiP, CCiNP, and OCIM bits to be updated. Note: This bit is only valid for channels with complementary outputs

CC8G: Generates a Capture/Compare 8 event. Described with reference to CC1G

CC7G: Generates a Capture/Compare 7 event. Described with reference to CC1G

CC6G: Generates a Capture/Compare 6 event. CC1G Description CC5G: Generates a Capture/Compare 5 event. CC1G Description

CC4G: Generates a Capture/Compare 4 event. CC1G Description CC3G: Generates a Capture/Compare 3 event. CC1G Description

CC2G: Generates a Capture/Compare 2 event. Refer to CC1G Description

CC1G: Generate capture/compare 1 event. Generates a capture/compare 1 event. This bit is set to 1 by software to generate a capture/compare event that is automatically cleared by hardware.

0: No action;

1: Generate a capture/compare event on channel CC1.

If channel CC1 is configured as an output: set CC1IF=1 and generate the corresponding interrupt if it is switched on.

If channel CC1 is configured as input: the current counter value is captured to the PWMA_CCR1 register, CC1IF=1 is set, and the corresponding interrupt is generated if it is turned on. If CC1IF is already 1, set CC1OF=1.

UGn: Generate update event This bit is set to 1 by software and cleared to 0 automatically by hardware.(n= A,B) 0: No action;

1: Reinitialise the counter and generate an update event.

Note that the prescaler counter is also cleared to 0 (but the prescaler coefficient remains unchanged). The counter is cleared to 0 if in centre-symmetric mode or if DIR=0 (counting up); if DIR=1 (counting down) the counter takes the value of PWMn_ARR.

23.7.13 Capture/Compare Mode Register 1 (PWMx_CCMR1)

The channel can be used as input (capture mode) or output (compare mode), and the direction of the channel is defined by the corresponding CCnS bit. The other bits of this register function differently in input and output modes.OCxx describes the function of the channel in output mode and ICxx describes the function of the channel in input mode. It is therefore important to note that the same bit functions differently in output and input modes.

Channel configured for compare output mode

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	7EFEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
PWMB_CCMR1	7EFEE8H	OC5CE	OC5M [2:0]			OC5PE	OC5FE	CC5S[1:0]	

OCnCE: Output Compare n Clear Enable. This bit is used to enable the use of an external event on the

PWMETI pin to clear the output signal of channel n

(OCnREF) (n=1,5)

0: OCnREF is not affected by the ETRF input;

1: Once the ETRF input is detected high, OCnREF=0.

OCnM[2:0]: output compare n mode. These 3 bits define the action of the output reference signal OCnREF, which determines the OCn

OCnREF is active high and the active level of OCn depends on the CCnP bit. (n=1,5)

OCnM[2:0]	paradigm	instructions
000	freeze (loan, wage, price etc)	Comparison between PWMn_CCR1 and PWMn_CNT does not work for OCnREF
001	Setting the channel n when matching	When PWMn_CCR1=PWMn_CNT, OCnREF outputs high

	Output is active level	
010	Setting the channel n when matching Output is at an invalid level	When PWMn_CCR1 = PWMn_CNT, OCnREF outputs low
011	flips	When PWMn_CCR1 = PWMn_CNT, flip OCnREF
100	Force to invalid level	Force OCnREF to be low
101	Force to active level	Force OCnREF to be high
110	PWM mode 1	OCnREF outputs high when PWMn_CNT<PWMn_CCR1 in up count, otherwise OCnREF outputs low In down count, when PWMn_CNT > PWMn_CCR1 OCnREF output low, otherwise OCnREF output high
111	PWM Mode 2	OCnREF outputs low when PWMn_CNT<PWMn_CCR1 in up count, otherwise OCnREF outputs high In down count, when PWMn_CNT > PWMn_CCR1 OCnREF output high, otherwise OCnREF output low

Note 1: This bit cannot be modified once the LOCK level is set to 3 (LOCK bit in the PWMn_BKR register) and CCnS=00 (the channel is configured as an output).

Note 2: In PWM Mode 1 or PWM Mode 2, the OCnREF level changes only when the comparison result is changed or when switching from freeze mode to PWM mode in the output comparison mode.

Note 3: On channels with complementary outputs, these bits are preloaded. If CCPC=1 in the PWMn_CR2 register, the OCM bits take a new value from the preloaded bits only when a COM event occurs.

OCnPE: output compare n preload enable (n=1,5)

0: Disable the preload function of PWMn_CCR1 register, the PWMn_CCR1 register can be written at any time, and the newly written value takes effect immediately.

1: Enable the preload function of PWMn_CCR1 register, read/write operation only operates on the preloaded register, the preloaded value ofPWMn_CCR1 is loaded into the current register when the update event arrives.

Note 1: This bit cannot be modified once the LOCK level is set to 3 (LOCK bit in the PWMn_BKR register) and CCnS=00 (the channel is configured as an output).

Note 2: For correct operation, the preload function must be enabled in PWM mode. However, it is not necessary in single pulse mode (OPM=1 in PWMn_CR1 register).

OCnFE: Output Compare n Fast Enable. This bit is used to speed up the response of the CC outputs to triggered input events. (n=1,5)

0: Depending on the value of the counter and CCRn, CCn operates normally, even if the trigger is open. The minimum delay to activate the CCn output is 5 clock cycles when the input of the flip-flop has a valid edge.

1: The active edge of the input to the flip-flop acts as if a comparison match has occurred.

Therefore, OC is set to the comparison level independent of the comparison result. The delay between the active edge of the sampling flip-flop and the CC1 output is reduced to 3 clock cycles. oCFE only functions when the channel is configured in PWMA or PWMB mode.

CC1S[1:0]: capture/compare 1 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC1S[1:0]	orientations	input leg
00	exports	
01	importation	IC1 mapped on TI1FP1
10	importation	IC1 mapped on TI2FP1
11	importation	IC1 is mapped on TRC. This mode operates only when the internal trigger input has been

		When selected (by the TS bit of the PWMA_SMCR register)

CC5S[1:0]: capture/compare 5 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC5S[1:0]	orientations	input leg
00	exports	
01	importation	IC5 mapped on TI5FP5
10	importation	IC5 mapped on TI6FP5
11	importation	IC5 is mapped on TRC. This mode operates only when the internal trigger inputs have been When selected (by the TS bit of the PWM5_SMCR register)

Note: CC1S is writable only when the channel is off (CC1E=0 for PWMA_CCER1 register). Note: CC5S is writable only when the channel is off (CC5E=0 for PWM5_CCER1 register).

Channel configured for capture input mode

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	7EFEC8H	IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
PWMB_CCMR1	7EFEE8H	IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]	

ICnF[3:0]: input capture n filter selection, this bit field defines the sampling frequency of TIn and the digital filter length. (n=1,5)

ICnF[3:0]	Number of clocks	ICnF[3:0]	Number of clocks
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

Note: Even for channels with complementary outputs, this bit field is not preloaded and does not take into account the value of CCPC (PWMn_CR2 register)

ICnPSC[1:0]: input/capture n prescaler. These two bits define the prescaler coefficients for the CCn input (IC1). (n=1,5)

00: no prescaler, each edge detected on the capture input triggers a capture

01: Capture triggered every 2 events

10: Capture triggered every 4 events

11: Capture triggered every 8 events

CC1S[1:0]: capture/compare 1 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC1S[1:0]	orientations	input leg
00	exports	
01	importation	IC1 mapped on TI1FP1
10	importation	IC1 mapped on TI2FP1
11	importation	IC1 is mapped on TRC. This mode operates only when the internal trigger input has been When selected (by the TS bit of the PWMA_SMCR

CC5S[1:0]: capture/compare 5 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC5S[1:0]	orientations	input leg
00	exports	
01	importation	IC5 mapped on TI5FP5
10	importation	IC5 mapped on TI6FP5
11	importation	IC5 is mapped on TRC. This mode operates only when the internal trigger inputs have been When selected (by the TS bit of the PWM5_SMCR register)

Note: CC1S is writable only when the channel is off (CC1E=0 for PWMA_CCER1 register). Note: CC5S is writable only when the channel is off (CC5E=0 for PWM5_CCER1 register).

23.7.14 Capture/Compare Mode Register 2 (PWMx_CCMR2)

Channel configured for compare output mode

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	7EFEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	
PWMB_CCMR2	7EFEE9H	OC6CE	OC6M [2:0]			OC6PE	OC6FE	CC6S[1:0]	

OCnCE: Output Compare n Clear Enable. This bit is used to enable the use of external events on the PWMETI pin to clear the output signal of channel n (OCnREF) (n=2,6)

0: OCnREF is not affected by the ETRF input;

1: OCnREF=0 once ETRF input high is detected

OCnM[2:0]: output compare 2 mode, refer to OC1M.(n=2,6) OCnPE: output compare 2 preload enable, refer to OP1PE.(n=2,6)

CC2S[1:0]: capture/compare 2 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC2S[1:0]	orientations	input leg
00	exports	
01	importation	IC2 mapped on TI2FP2
10	importation	IC2 mapped on TI1FP2
11	importation	IC2 is mapped on the TRC.

CC6S[1:0]: capture/compare 6 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC6S[1:0]	orientations	input leg
00	exports	
01	importation	IC6 mapped on TI6FP6
10	importation	IC6 mapped on TI5FP6
11	importation	IC6 is mapped on the TRC.

Channel configured for capture input mode

notation	address	B7	B6	B5	B4	B3	B2	B1	B0

PWMA_CCMR2	7EFEC9H	IC2F[3:0]	IC2PSC[1:0]	CC2S[1:0]
PWMB_CCMR2	7EFEE9H	IC6F[3:0]	IC6PSC[1:0]	CC6S[1:0]

ICnF[3:0]: input capture n filter selection, refer to IC1F.(n=2,6)
ICnPSC[1:0]: input/capture n prescaler, refer to IC1PSC.(n=2,6)
CC2S[1:0]: capture/compare 2 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC2S[1:0]	orientations	input leg
00	exports	
01	importation	IC2 mapped on TI2FP2
10	importation	IC2 mapped on TI1FP2
11	importation	IC2 is mapped on the TRC.

CC6S[1:0]: capture/compare 6 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC6S[1:0]	orientations	input leg
00	exports	
01	importation	IC6 mapped on TI6FP6
10	importation	IC6 mapped on TI5FP6
11	importation	IC6 is mapped on the TRC.

23.7.15 Capture/Compare Mode Register 3 (PWMx_CCMR3)

Channel configured for compare output mode

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	7EFECAH	OC3CE	OC3M [2:0]			OC3PE	OC3FE	CC3S[1:0]	
PWMB_CCMR3	7EFEEAH	OC7CE	OC7M [2:0]			OC7PE	OC7FE	CC7S[1:0]	

OCnCE: Output Compare n Clear Enable. This bit is used to enable the use of external events on the PWMETI pin to clear the output signal of channel n
(OCnREF) (n=3,7)
0: OCnREF is not affected by the ETRF input;
1: OCnREF=0 once ETRF input high is detected
OCnM[2:0]: output compare 3 mode, refer to OC1M.(n=3,7) OCnPE: output compare 3 preload enable, refer to OP1PE.(n=3,7)

CC3S[1:0]: capture/compare 3 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC3S[1:0]	orientations	input leg
00	exports	
01	importation	IC3 mapped on TI3FP3
10	importation	IC3 mapped on TI4FP3
11	importation	IC3 is mapped on TRC.

CC7S[1:0]: capture/compare 7 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC7S[1:0]	orientations	input leg
00	exports	
01	importation	IC7 mapped on TI7FP7
10	importation	IC7 mapped on TI8FP7
11	importation	IC7 is mapped on TRC.

Channel configured for capture input mode

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	7EFECAH	IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	

PWMB_CCMR3	7EFEEAH	IC7F[3:0]	IC7PSC[1:0]	CC7S[1:0]

ICnF[3:0]: input capture n filter selection, refer to IC1F.(n=3,7)
ICnPSC[1:0]: input/capture n prescaler, refer to IC1PSC.(n=3,7)

CC3S[1:0]: capture/compare 3 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC3S[1:0]	orientations	input leg
00	exports	
01	importation	IC3 mapped on TI3FP3
10	importation	IC3 mapped on TI4FP3
11	importation	IC3 is mapped on TRC.

CC7S[1:0]: capture/compare 7 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC7S[1:0]	orientations	input leg
00	exports	
01	importation	IC7 mapped on TI7FP7
10	importation	IC7 mapped on TI8FP7
11	importation	IC7 is mapped on TRC.

23.7.16 Capture/Compare Mode Register 4 (PWMx_CMCR4)

Channel configured for compare output mode

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	7EFECBH	OC4CE	OC4M [2:0]			OC4PE	OC4FE	CC4S[1:0]	
PWMB_CCMR4	7EFEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]	

OCnCE: Output Compare n Clear Enable. This bit is used to enable the use of external events on the PWMETI pin to clear the output signal of channel n
(OCnREF) (n=4,8)
0: OCnREF is not affected by the ETRF input;
1: OCnREF=0 once ETRF input high is detected
OCnM[2:0]: output compare n mode, refer to OC1M.(n=4,8) OCnPE: output compare n preload enable, refer to OP1PE.(n=4,8)

CC4S[1:0]: capture/compare 4 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC4S[1:0]	orientations	input leg
00	exports	
01	importation	IC4 mapped on TI4FP4
10	importation	IC4 mapped on TI3FP4
11	importation	IC4 is mapped on the TRC.

CC8S[1:0]: capture/compare 8 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC8S[1:0]	orientations	input leg
00	exports	
01	importation	IC8 mapped on TI8FP8
10	importation	IC8 mapped on TI7FP8
11	importation	IC8 is mapped on TRC.

Channel configured for capture input mode

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	7EFECBH	IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]	
PWMB_CCMR4	7EFEEBH	IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]	

ICnF[3:0]: input capture n filter selection, refer to IC1F.(n=4,8)
ICnPSC[1:0]: input/capture n prescaler, refer to IC1PSC.(n=4,8)

CC4S[1:0]: capture/compare 4 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC4S[1:0]	orientations	input leg
00	exports	
01	importation	IC4 mapped on TI4FP4
10	importation	IC4 mapped on TI3FP4
11	importation	IC4 is mapped on the TRC.

CC8S[1:0]: capture/compare 8 selection. These two bits define the direction of the channel (input/output), and the selection of the input pins

CC8S[1:0]	orientations	input leg
00	exports	
01	importation	IC8 mapped on TI8FP8
10	importation	IC8 mapped on TI7FP8
11	importation	IC8 is mapped on TRC.

23.7.17 Capture/Compare Enable Register 1 (PWMx_CCER1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER1	7EFECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
PWMB_CCER1	7EFEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E

CC6P: OC6 input capture/compare output polarity. Reference CC1P CC6E: OC6 Input capture/compare output enable. Reference CC1E CC5P: OC5 Input capture/compare output polarity. Reference CC1P CC5E: OC5 Input capture/compare output enable. Reference CC1E CC2NP: OC2N Compare output polarity. CC1NP CC2NE: OC2N compare output enable. Reference CC1NE CC2P: OC2 Input capture/compare output polarity. Reference CC1P CC2E: OC2 input capture/compare output enable. Reference CC1E CC1NP: OC1N Compare Output Polarity

0: Active high;

1: Active low.

Note 1: This bit cannot be modified once the LOCK level (LOCK bit in the PWMA_BKR register) is set to 3 or 2 and CC1S=00 (channel is configured for output).

Note 2: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA_CR2 register), this bit is only available if the

The CC1NP bit takes a new value from the preload bit only when the COM event occurs.

CC1NE: OC1N compare output enable

0: Turns off the comparison output.

1: Turns on the compare output, whose output level depends on the values of the MOE, OSSI, OSSR, OIS1, OIS1N and CC1E bits.

Note: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA_CR2 register), this bit is only loaded if the COM

The CC1NE bit takes the new value from the preload bit only when the event occurs.

CC1P: OC1 input
capture/compare output
polarity CC1 channel
configured as output: 0:
active high
1: Active Low
The CC1 channel is configured as input or capture:
0: Capture occurs on the rising edge of TI1F or TI2F;
1: Capture occurs on the falling edge of TI1F or TI2F.
Note 1: This bit cannot be modified once the LOCK level (LOCK bit in the PWMA_BKR register) is set to 3 or 2. Note 2: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA_CR2 register), the bit is only available if the

The CC1P bit takes a new value from the preload bit only when the COM event occurs.

CC1E: OC1 input capture/compare output enable
0: Disables the input capture/compare output;
1: Enables input capture/comparison output.
Note: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA_CR2 register), this bit is only loaded if the COM

The CC1E bit only takes the new value from the preload bit when the event occurs.

Control bits for complementary output channels OCi and OCiN with brake function

control bits					output state	
MOE	OSSI	OSSR	CCiE	CCiNE	OCi Output Status	OCiN Output Status
1	X	0	0	0	output suppression	output suppression
		0	0	1	output suppression	OCiREF with polarity
		0	1	0	OCiREF with polarity	output suppression
		0	1	1	OCiREF with polarity and deadband	Inverted OCiREF with polarity and deadband
		1	0	0	output suppression	output suppression
		1	0	1	closed state (output enabled and disabled) OCi=CCiP	OCiREF with polarity
		1	1	0	OCiREF with polarity	closed state (output enabled and disabled) OCiN=CCiNP
		1	1	1	OCiREF with polarity and deadband	Inverted OCiREF with polarity and deadband
0	0	X	X	X	output suppression	
	1				Off state (output enabled and invalid level) asynchronously: OCi=CCiP, OCiN=CCiNP; then, if the clock is present: after a dead time OCi=OCi, OCiN=OCiN, assuming that OISi and OISiN do not both correspond to valid levels for OCi and OCiN.	

Note: The status of the external I/O pins whose pins are connected to the complementary OCi and OCiN channels depends on the OCi and OCiN channel status and GPIO Registers.

23.7.18 Capture/Compare Enable Register 2 (PWMx_CCER2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER2	7EFECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E

PWMB_CCER2	7EFEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E

CC8P: OC8 input capture/compare output polarity. Reference CC1P CC8E: OC8 Input capture/compare output enable. Reference CC1E CC7P: OC7 Input capture/compare output polarity. Reference CC1P CC7E: OC7 Input capture/compare output enable. Reference CC1E CC4NP: OC4N Compare output polarity. Reference CC1NP CC4NE: OC4N compare output enable. Reference CC1NE CC4P: OC4 Input capture/compare output polarity. Reference CC1P CC4E: OC4 input capture/compare output enable. Reference CC1E CC3NP: OC3N compare output polarity. Reference CC1NP CC3NE: OC3N compare output enable. Reference CC1NE CC3P: OC3 Input capture/compare output polarity. Reference CC1P CC3E: OC3 input capture/compare output enable. Reference CC1E

23.7.19 Counter High 8 bits (PWMx_CNTRH)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRH	7EFECEH	CNT1[15:8]							
PWMB_CNTRH	7EFEEEH	CNT2 [15:8]							

CNTn[15:8]: high 8-bit value of the counter (n=A,B)

23.7.20 Counter low 8 bits (PWMx_CNTRL)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRL	7EFECFH	CNT1[7:0]							
PWMB_CNTRL	7EFEEFH	CNT2 [7:0]							

CNTn[7:0]: low 8-bit value of the counter (n=A,B)

23.7.21 Prescaler high 8 bits (PWMx_PSCRH), output frequency calculation equation

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRH	7EFED0H	PSC1[15:8]							
PWMB_PSCRH	7EFEF0H	PSC2[15:8]							

PSCn[15:8]: high 8-bit value of the prescaler. (n=A,B)

The prescaler is used to divide the CK_PSC. The clock frequency of the counter (PSCR[15:0]+1) (fCK_CNT) is equal to fCK_PSC().

PSCR contains the value loaded into the current prescaler register when an update event is generated (update events include the counter being UG-loaded by TIM_EGR).

bit is cleared to 0 or by a slave controller operating in reset mode). This means that an update event

must be generated in order for the new value to take effect.

PWM Output Frequency Calculation Formula

The output frequencies of PWMA and PWMB are calculated by the same formula, and each group can be set to a different frequency.

registration mode	PWM output frequency calculation formula
edge alignment	$\text{PWM output frequency} = \frac{\text{System operating frequency SYSclk}}{(\text{PWMx\_PSCR} + 1) \times (\text{PWMx\_ARR} + 1)}$

centre-aligned	$\text{PWM output frequency} = \frac{\text{System operating frequency SYSclk}}{(\text{PWMx\_PSCR} + 1) \times \text{PWMx\_ARR} \times 2}$

23.7.22 Prescaler low 8 bits (PWMx_PSCRL)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRL	7EFED1H	PSC1[7:0]							
PWMB_PSCRL	7EFEF1H	PSC2[7:0]							

PSCn[7:0]: low 8-bit value of the prescaler. (n=A,B)

23.7.23 Auto Reload Register High 8 bits (PWMx_ARRH)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRH	7EFED2H	ARR1[15:8]							
PWMB_ARRH	7EFEF2H	ARR2[15:8]							

ARRn[15:8]: automatic reloading of high 8-bit values (n= A,B)

The ARR contains the value that will be loaded into the actual auto-reload register. When the value of Auto Reload is 0, the counter does not operate.

23.7.24 Auto Reload Register Low 8 bits (PWMx_ARRL)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRL	7EFED3H	ARR1[7:0]							
PWMB_ARRL	7EFEF3H	ARR2[7:0]							

ARRn[7:0]: automatic reloading of the lower 8 bits (n=A,B)

23.7.25 Repeat Counter Register (PWMx_RCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_RCR	7EFED4H	REP1[7:0]							
PWMB_RCR	7EFEF4H	REP2[7:0]							

REPn[7:0]: Repeat counter value (n= A,B)

With preload enabled, these bits allow the user to set the update rate of the compare registers (i.e., periodic transfers from the preloaded register to the current register); allowing the generation of update interrupts also affects the rate at which update interrupts are generated. Each time the down counter REP_CNT reaches 0, an update event is generated and the counter REP_CNT starts counting again from the REP value. Since REP_CNT only reloads the REP value when a cycle update event U_RC occurs, the new value written to the PWMn_RCR register will only take effect when the next cycle update event occurs. This means that in PWM mode, (REP+1) corresponds to:

— The number of PWM cycles in edge-aligned mode;

— The number of PWM half-cycles in centrosymmetric mode.

23.7.26 Capture/Compare Register 1/5 High 8 bits

(PWMx_CCR1H)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	7EFED5H	CCR1[15:8]							
PWMB_CCR5H	7EFEF5H	CCR5[15:8]							

CCRn[15:8]: capture/compare the high 8-bit value of n (n=1,5)

If the CCn channel is configured as an output: the CCRn contains the loaded current comparison value (preloaded value). If the preload function is not selected in the PWMn_CCMR1 register (OCnPE bit), the written value is immediately transferred to the current register. Otherwise this preloaded value is transferred to the current capture/compare n register only when an update event occurs. The current compare value is compared with the value of counter PWMn_CNT and an output signal is generated on the OCn port.

If the CCn channel is configured as an input: the CCRn contains the counter value at the time of the last input capture event (this register is read-only at this time).

23.7.27 Capture/Compare Register 1/5 Low 8-bit (PWMx_CCR1L)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1L	7EFED6H	CCR1[7:0]							
PWMB_CCR5L	7EFEF6H	CCR5[7:0]							

CCRn[7:0]: capture/compare the lower 8 bits of n (n=1,5)

23.7.28 Capture/Compare Register 2/6 High 8 bits (PWMx_CCR2H)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2H	7EFED7H	CCR2[15:8]							
PWMB_CCR6H	7EFEF7H	CCR6[15:8]							

CCRn[15:8]: capture/compare the high 8-bit value of n (n=2,6)

23.7.29 Capture/Compare Register 2/6 Low 8 bits (PWMx_CCR2L)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2L	7EFED8H	CCR2[7:0]							
PWMB_CCR6L	7EFEF8H	CCR6[7:0]							

CCRn[7:0]: capture/compare the lower 8 bits of n (n=2,6)

23.7.30 Capture/Compare Register 3/7 High 8 bits (PWMx_CCR3H)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3H	7EFED9H	CCR3[15:8]							
PWMB_CCR7H	7EFEF9H	CCR7[15:8]							

CCRn[15:8]: capture/compare the high 8-bit value of n (n=3,7)

23.7.31 Capture/Compare Register 3/7 Low 8 bits (PWMx_CCR3L)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3L	7EFEDAH	CCR3[7:0]							
PWMB_CCR7L	7EFEFAH	CCR7[7:0]							

CCRn[7:0]: capture/compare the lower 8 bits of n (n=3,7)

23.7.32 Capture/Compare Register 4/8 High 8 bits (PWMx_CCR4H)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4H	7EFEDBH	CCR4[15:8]							
PWMB_CCR8H	7EFEFBH	CCR8[15:8]							

CCRn[15:8]: capture/compare the high 8-bit value of n (n=4,8)

23.7.33 Capture/Compare Register 4/8 Low 8 bits (PWMx_CCR4L)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4L	7EFEDCH	CCR4[7:0]							
PWMB_CCR8L	7EFEFCH	CCR8[7:0]							

CCRn[7:0]: capture/compare the lower 8 bits of n (n=4,8)

23.7.34 Brake register (PWMx_BKR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_BKR	7EFEDDH	MOEA	AOEA	BKPA	BKEA	OSSRA	OSSIA	LOCKA[1:0]	
PWMB_BKR	7EFEFDH	MOEB	AOEB	BKPB	BKEB	OSSRB	OSSIB	LOCKB[1:0]	

MOEn: Master Output Enable. This bit is cleared asynchronously by hardware as soon as the brake input is active, and can be set to 1 by software or automatically, depending on the set value of the AOE bit, which is only active for channels configured as outputs. (n=A,B)

0: Disable OC and OCN outputs or force to idle state

1: Enables the OC and OCN outputs if the corresponding enable bit (CCIE bit of PWMn_CCERX register) is set.

AOEn: Auto Output Enable

(n=A,B) 0: MOE can only be set to 1 by software;

1: MOE can be set to 1 by software or automatically set to 1 at the next update event (if the brake input is not valid).

Note: Once the LOCK level (LOCK bit in PWMn_BKR register) is set to 1, this bit cannot be modified BKPn: Brake input polarity (n= A,B)

0: Brake input active low

1: Brake input active high

Note: Once the LOCK level (LOCK bit in PWMn_BKR register) is set to 1, this bit cannot be modified BKEn: Brake function enable (n= A,B)

0: Brake input disabled (BRK)

1: Switch on brake input (BRK)

Note: Once the LOCK level (LOCK bit in PWMn_BKR register) is set to 1, this bit cannot be modified. OSSRn: "Off State" selection in operation mode. This bit is valid when MOE=1 and the channel is set to output (n=A,B).

0: Disable OC/OCN output when PWM is not operating (OC/OCN enable output signal = 0);

1: When PWM is not working, once CCiE=1 or CCiNE=1, first turn on OC/OCN and output invalid level, then

set OC/OCN.

Enable output signal = 1.

Note: Once the LOCK level (LOCK bit in PWMn_BKR register) is set to 2, this bit cannot be modified. OSSIn: Idle mode "off state" selection. This bit is valid when MOE=0 and the channel is set to output. (n=A,B)

0: Disable OC/OCN output when PWM is not operating (OC/OCN enable output signal = 0);

1: When PWM is not operating, once CCiE=1 or CCiNE=1, OC/OCN first outputs its idle level, then OC/OCN

Enable output signal = 1.

Note: Once the LOCK level (LOCK bit in the PWMn_BKR register) is set to 2, this bit cannot be modified.

LOCKn[1:0]: lock setting. This bit provides write protection against software errors (n= A,B)

LOCKn[1:0]	Protection level	Protection content
00	unprotected	Registers are not write-protected
01	Locking level 1	The BKE, BKP, and AOE bits of the PWMn_BKR register cannot be written to the PWMn_BKR register and the OISI bit of the PWMn_OISR register
10	Locking level 2	Cannot write to each bit in lock level 1. The CC polarity bit and the OSSR/OSSI bit cannot be written either.
11	Locked level 3	cannot write to each of the bits in lock level 2. The CC control bit cannot be written either

Note: Since the BKE,BKP,AOE,OSSR and OSSI bits can be locked (dependent on the LOCK bit), the first time PWMn_BKR is written registers when they must be set.

23.7.35 Deadband register (PWMx_DTR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DTR	7EFEDEH	DTGA[7:0]							
PWMB_DTR	7EFEFEH	DTGB[7:0]							

DTGn[7:0]: deadband generator setting. (n= A,B)

These bits define the deadband duration between insertion of complementary outputs. (t_{CK_PSC} is the clock pulse for PWMn)

DTGn[7:5]	dead time
000	DTGn[7:0] * t_{CK_PSC}
001	
010	
011	
100	(64 + DTGn[6:0]) * 2 * t_{CK_PSC}
101	
110	(32 + DTGn[5:0]) * 8 * t_{CK_PSC}
111	(32 + DTGn[4:0]) * 16 * t_{CK_PSC}

Note: Once the LOCK level (LOCK bit in the PWMx_BKR register) is set to 1, 2 or 3, this bit cannot be modified.

23.7.36 Output Idle Status Register (PWMx_OISR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_OISR	7EFEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1
PWMB_OISR	7EFEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5

OIS8: OC8 output level in idle state OIS7: OC7 output level in idle state OIS6: OC6 output level in idle state OIS5: OC5 output level in idle state OIS4N: OC4N

output level in idle state OIS4:
OC4 output level in idle state

OIS3N: OC3N output level during idle state OIS3: OC3 output level during idle state OIS2N: OC2N output level during idle state OIS2: OC2 output level during idle state OIS1N: OC1N output level during idle state

0: When MOE=0, then OC1N=0 after one dead time;

1: When MOE=0, then OC1N=1 after one dead time.

Note: Once the LOCK level (LOCK bit in the PWMx_BKR register) is set to 1, 2 or 3, this bit cannot be modified.

OIS1: OC1 output level in idle state

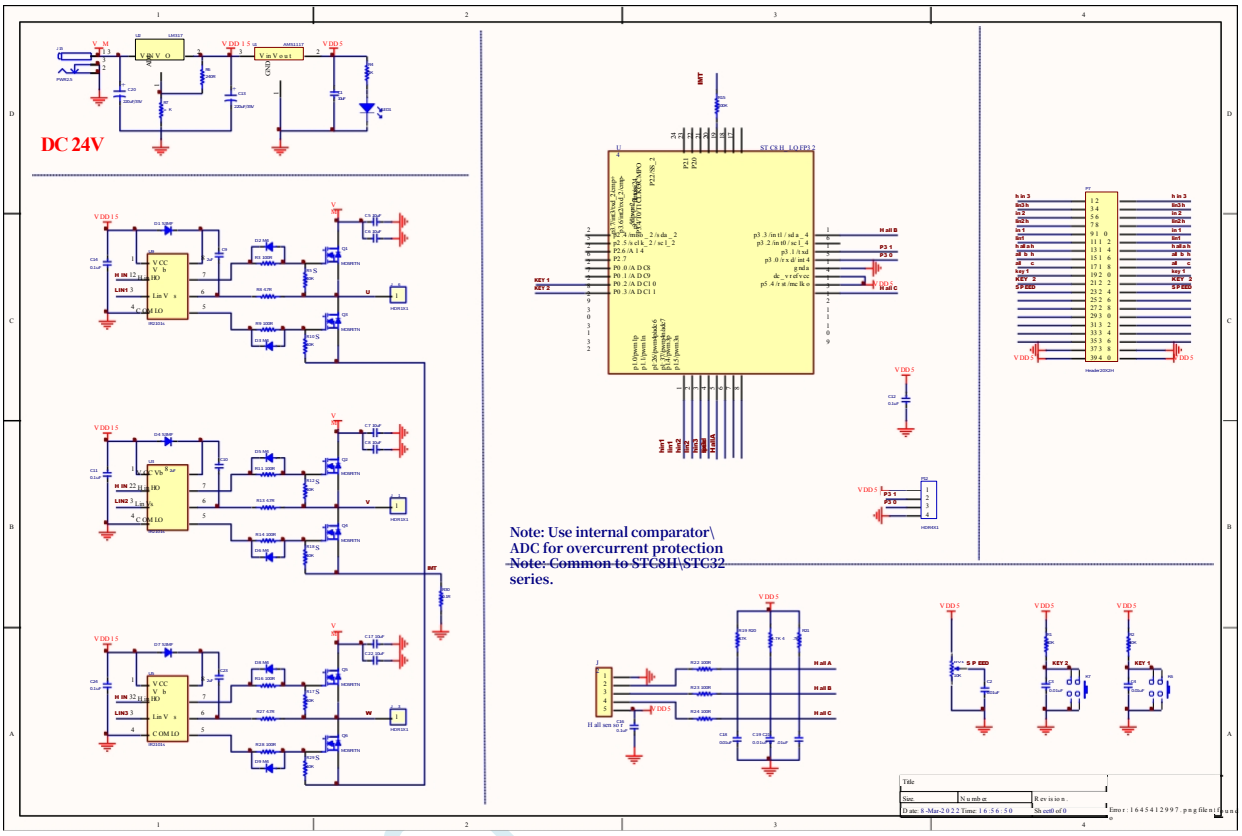
0: When MOE=0, if OC1N is enabled, OC1=0 after one deadband;

1: When MOE=0, if OC1N is enabled, OC1=1 after one deadband.

Note: Once the LOCK level (LOCK bit in the PWMx_BKR register) is set to 1, 2 or 3, this bit cannot be modified.

23.8 sample procedure

23.8.1 BLDC Brushless DC Motor with HALL




```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    //see download software for header files
#include "intrins.h"

typedef unsigned char  u8;
typedef unsigned int   u16.

#define  TRUE   1
#define  FALSE  0

#define  RV09_CH  6

#define  PWMA_Period ((u16)0x0180)
#define  PWMA_STPulse  ((u16)342)

#define  START  0x1A
#define  RUN    0x1B
#define  STOP   0x1C
#define  IDLE   0x1D

#define  PWMA_OCMODE_MASK
```

```
#define PWMA_OCCE_DISABLE ((u8)0x00)
#define PWMA_OCCE_ENABLE ((u8)0x80)
```

```
#define   PWMA_OCCE_DISABLE      ((u8)0x00)
#define   PWMA_OCMODE_TIMING     ((u8)0x00)
#define   PWMA_OCMODE_ACTIVE     ((u8)0x10)
#define   PWMA_OCMODE_INACTIVE   ((u8)0x20)
#define   PWMA_OCMODE_TOGGLE     ((u8)0x30)
#define   PWMA_FORCE_INACTIVE    ((u8)0x40)
#define   PWMA_FORCE_ACTIVE      ((u8)0x50)
#define   PWMA_OCMODE_PWM1       ((u8)0x60)
#define   PWMA_OCMODE_PWM2       ((u8)0x70)
#define   CC1_POLARITY_HIGH      ((u8)0x02)
#define   CC1N_POLARITY_HIGH     ((u8)0x08)
#define   CC2_POLARITY_HIGH      ((u8)0x20)
#define   CC2N_POLARITY_HIGH     ((u8)0x80)
#define   CC1_POLARITY_LOW       ((u8)~0x02)
#define   CC1N_POLARITY_LOW      ((u8)~0x08)
#define   CC2_POLARITY_LOW       ((u8)~0x20)
#define   CC2N_POLARITY_LOW      ((u8)~0x80)
#define   CC1_OCENABLE           ((u8)0x01)
#define   CC1N_OCENABLE          ((u8)0x04)
#define   CC2_OCENABLE           ((u8)0x10)
#define   CC2N_OCENABLE          ((u8)0x40)
#define   CC1_OCDISABLE          ((u8)~0x01)
#define   CC1N_OCDISABLE         ((u8)~0x04)
#define   CC2_OCDISABLE          ((u8)~0x10)
#define   CC2N_OCDISABLE         ((u8)~0x40)
#define   CC3_POLARITY_HIGH      ((u8)0x02)
#define   CC3N_POLARITY_HIGH     ((u8)0x08)
#define   CC4_POLARITY_HIGH      ((u8)0x20)
#define   CC4N_POLARITY_HIGH     ((u8)0x80)
#define   CC3_POLARITY_LOW       ((u8)~0x02)
#define   CC3N_POLARITY_LOW      ((u8)~0x08)
#define   CC4_POLARITY_LOW       ((u8)~0x20)
#define   CC4N_POLARITY_LOW      ((u8)~0x80)
#define   CC3_OCENABLE           ((u8)0x01)
#define   CC3N_OCENABLE          ((u8)0x04)
#define   CC4_OCENABLE           ((u8)0x10)
#define   CC4N_OCENABLE          ((u8)0x40)
#define   CC3_OCDISABLE          ((u8)~0x01)
#define   CC3N_OCDISABLE         ((u8)~0x04)
#define   CC4_OCDISABLE          ((u8)~0x10)
#define   CC4N_OCDISABLE         ((u8)~0x40)

void LED_OUT(u8 X).                                  //LED single-byte serial shift function

unsigned char code LED_0F[] =
{
    0xc0,0xf9,0xa4,0xb0.
    0x99,0x92,0x82,0xF8.
    0x80,0x90,0x8c,0xbf.
    0xc6,0xa1,0x86,0xff.
    0xbf
};

#define   DIO          P23          //Serial Data Input
#define   RCLK         P24          //Clock pulse signal - rising edge active
#define   SCLK         P25          //Punch-in signal ---- valid on rising edge

void DelayXus(unsigned char delayTime);
```

void DelayMs(unsigned char delayTime).

```
unsigned int ADC_Convert(u8 ch);
void PWM_Init(void).
void SPEED_ADJ();
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP().
unsigned char KEY_detect();
void LED4_Display (unsigned int dat,unsigned char num).

unsigned char Display_num=1;
unsigned int Display_dat=0;
unsigned int Motor_speed;
unsigned char Motor_sta = IDLE;
unsigned char BRK_occur=0;
unsigned int PWMB_CAP1_v=0;
unsigned int CAP1_avg=0;
unsigned char CAP1_cnt=0;
unsigned long CAP1_sum=0;

void main(void)
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign a value of 0 to set the CPU to execute the
                                                programme as fast as possible.


    P_SW2 = 0x80.

    p1 = 0x00;
    p0m1 = 0x0c;
    p0m0 = 0x01;
    p1m1 = 0xc0;
    p1m0 = 0x3f;
    p2m1 = 0x00;
    p2m0 = 0x38;
    p3m1 = 0x28;
    p3m0 = 0x00.

    ET0=1.
    TR0=1;

    ADCCFG = 0x0f.
    ADC_CONTR = 0x80.

    PWMA_ENO = 0x3F.                            //PWMA output enable
    PWMB_ENO = 0x00.                            //PWMB output enable
    PWMA_PS = 0x00; //PWMA_PS = 0x00; //PWMA_PS = 0x00 //PWMA pin selection
    PWMB_PS = 0xd5.                             //PWMB pin selection

/*************************************************************
Output compare mode PWMx_duty = [CCRx/(ARR + 1)]*100
*************************************************************/

/************PWMB connect hall sensor ****************/

////////////time base unit ////////////
    PWMB_PSCRL = 15;
    PWMB_ARRH = 0xff.                           //automatically reload registers, counter overflow point
    PWMB_ARRL = 0xff.
    PWMB_CCR8H = 0x00;
    PWMB_CCR8L = 0x05.
```

```
////////// channel configuration //////////
    PWMB_CCMR1 = 0x43; //Channel mode configuration.          //Channel mode configuration
    PWMB_CCMR2 = 0x41.
    PWMB_CCMR3 = 0x41;
    PWMB_CCMR4 = 0x70;
    PWMB_CCER1 = 0x11;
    PWMB_CCER2 = 0x11.

////////// Mode Configuration
    ////////// PWMB_CR2 = 0xf0;
    PWMB_CR1 = 0x81;
    PWMB_SMCR = 0x44.

////////// enable & interrupt configuration //////////
    PWMB_BKR = 0x80.                                          //main output enable
    PWMB_IER = 0x02.                                          //Enable interrupt


/************PWMA Control Motor Phase Change ****************/
////////// time base unit //////////
    PWMA_PSCRH = 0x00.                                        //pre-scaler register
    PWMA_PSCRL = 0x00;
    PWMA_ARRH = (u8)(PWMA_Period >> 8);
    PWMA_ARRL = (u8)(PWMA_Period).

////////// channel configuration //////////
    PWMA_CCMR1 = 0x70; //Channel mode configuration.          //Channel mode configuration
    PWMA_CCMR2 = 0x70.
    PWMA_CCMR3 = 0x70.
    PWMA_CCER1 = 0x11; //Configure the channel output enable and polarity.    //configure channel output enable and
    polarity
    PWMA_CCER2 = 0x01; //Configure the channel output enable and polarity.    //configure channel output enable and
    polarity
    PWMA_OISR = 0xAA; //Configure the output level of each channel when MOE=0.    //Configure the output level of
    each channel when MOE=0.

////////// Mode Configuration
    ////////// PWMA_CR1 = 0xA0;
    PWMA_CR2 = 0x24;
    PWMA_SMCR = 0x20.

////////// enable & interrupt
    configuration //////////
    PWMA_BKR = 0x1c;
    PWMA_CR1 |= 0x01; //Enable counter.                      //Enable counter

    EA = 1;
    while (1)
    {
        P22=~P22.
        Display_dat = Motor_speed.                            //Motor_speed

        switch(Motor_sta)
        {
        case START.
            MOTOR_START().
            Motor_sta = RUN;
            break;
```

```
    case RUN:
        SPEED_ADJ();
        if((KEY_detect() == 2)||(BRK_occur == TRUE))
            Motor_sta = STOP;
```

```
                break;
            case STOP.
                MOTOR_STOP().
                Motor_sta = IDLE;
                break;
            case IDLE.
                if(KEY_detect()==1)
                    Motor_sta = START;
                BRK_occur = FALSE;
                Motor_speed = 0;
                CAP1_avg = 0;
                CAP1_cnt = 0;
                CAP1_sum = 0;
                break;
            }
        }
}

void TIM0_ISR() interrupt 1
{
    TH0=0xf0.
    if(Display_num>8)
        Display_num=1;
    LED4_Display(Display_dat,Display_num);
    Display_num=(Display_num<<1);
}

void PWMA_ISR() interrupt 26
{
    if((PWMA_SR1 & 0x20))
    {
        switch(RD_HALL())
        {
        case 3.
            PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR3 |= PWMA_FORCE_INACTIVE;
            PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR1 |= PWMA_OCMODE_PWM2.
            break;
        case 2.
            PWMA_CCER1 &= CC2N_POLARITY_LOW;
            PWMA_CCER2 |= CC3N_POLARITY_HIGH.
            break;
        case 6.
            PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR1 |= PWMA_FORCE_INACTIVE;
            PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR2 |= PWMA_OCMODE_PWM2.
            break;
        case 4.
            PWMA_CCER1 |= CC1N_POLARITY_HIGH;
            PWMA_CCER2 &= CC3N_POLARITY_LOW.
            break;
        case 5.
            PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR2 |= PWMA_FORCE_INACTIVE;
            PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR3 |= PWMA_OCMODE_PWM2.
            break;
```

```
            case 1.
                PWMA_CCER1 &= CC1N_POLARITY_LOW;
                PWMA_CCER1 |= CC2N_POLARITY_HIGH.
                break;
            }

            CAP1_sum += PWMB_CAP1_v;
            CAP1_cnt++;
            if(CAP1_cnt==128)
            {
                CAP1_cnt=0;
                CAP1_avg = (CAP1_sum>>7);
                CAP1_sum = 0;
                Motor_speed = 5000000/CAP1_avg.
            }

            PWMA_SR1 &=~0x20.                       //Clear
        }
        if((PWMA_SR1 & 0x80))                       //BRK
        {
            BRK_occur = TRUE;
            PWMA_SR1 &=~0x80.                       //Clear
        }
}

void PWMB_ISR() interrupt 27
{
        if((PWMB_SR1 & 0x02))
        {
            PWMB_CAP1_v = PWMB_CCR5H.
            PWMB_CAP1_v = (PWMB_CAP1_v<<8) + PWMB_CCR5L;
            PWMB_SR1 &=~0x02.
        }
}

void DelayXus(unsigned char delayTime)
{
        int i = 0;
        while( delayTime--)
        {
            for( i = 0 ; i < 1 ; i++).
        }
}

void DelayXms( unsigned char delayTime )
{
        int i = 0;
        while( delayTime--)
        {
            for( i = 0 ; i < 2 ; i++)
            {
                DelayXus(100).
            }
        }
}

unsigned int ADC_Convert(u8 ch)
{
        u16 res=0;
```

```
    ADC_CONTR &= ~0x0f.
    ADC_CONTR |= ch.
    ADC_CONTR |= 0x40.
    DelayXus(1).
    while (! (ADC_CONTR & 0x20));
    ADC_CONTR &= ~0x20.

    res = ADC_RES.
    res = (res<<<2)+(ADC_RESL>>6);
    return res;
}

void SPEED_ADJ()
{
    u16 ADC_result.

    ADC_result = (ADC_Convert(RV09_CH)/3);
    PWMA_CCR1H = (u8)(ADC_result >> 8);                    //counter
    compare value PWMA_CCR1L = (u8)(ADC_result);
    PWMA_CCR2H = (u8)(ADC_result >> 8);
    PWMA_CCR2L = (u8)(ADC_result);
    PWMA_CCR3H = (u8)(ADC_result >> 8);
    PWMA_CCR3L = (u8)(ADC_result);
}

unsigned char RD_HALL()
{
    unsigned char Hall_sta = 0;

    (P17)?          (Hall_sta|=0x01)          :
    (Hall_sta&=~0x01);  (P54)?  (Hall_sta|=0x02)
    :          (Hall_sta&=~0x02);          (P33)?
    (Hall_sta|=0x04) : (Hall_sta&=~0x04); (P33)?

    return Hall_sta;
}

void MOTOR_START()
{
    u16 temp.
    u16 ADC_result.

    PWMA_CCR1H = (u8)(PWMA_STPulse >> 8);              //counter
    compare value PWMA_CCR1L = (u8)(PWMA_STPulse);
    PWMA_CCR2H = (u8)(PWMA_STPulse >> 8);
    PWMA_CCR2L = (u8)(PWMA_STPulse);
    PWMA_CCR3H = (u8)(PWMA_STPulse >> 8);
    PWMA_CCR3L = (u8)(PWMA_ STPulse);
    PWMA_BKR |= 0x80.                                  //Main output enable is equivalent to the master switch
    PWMA_IER |= 0xA0.                                  //Enable interrupt

    switch(RD_HALL())
    {
    case 1.
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 |= CC2N_POLARITY_HIGH;
        PWMA_CCER2 &= CC3N_POLARITY_LOW;
        PWMA_CCMR3 &= ~PWMA_OCMODE_ MASK;
        PWMA_CCMR3 |= PWMA_FORCE_INACTIVE;
```

```
PWMA_FORCE_INACTIVE; PWMA_CCMR3 |=
PWMA_FORCE_INACTIVE MASK;
PWMA_CCMR3 |= PWMA_FORCE_INACTIVE.
```

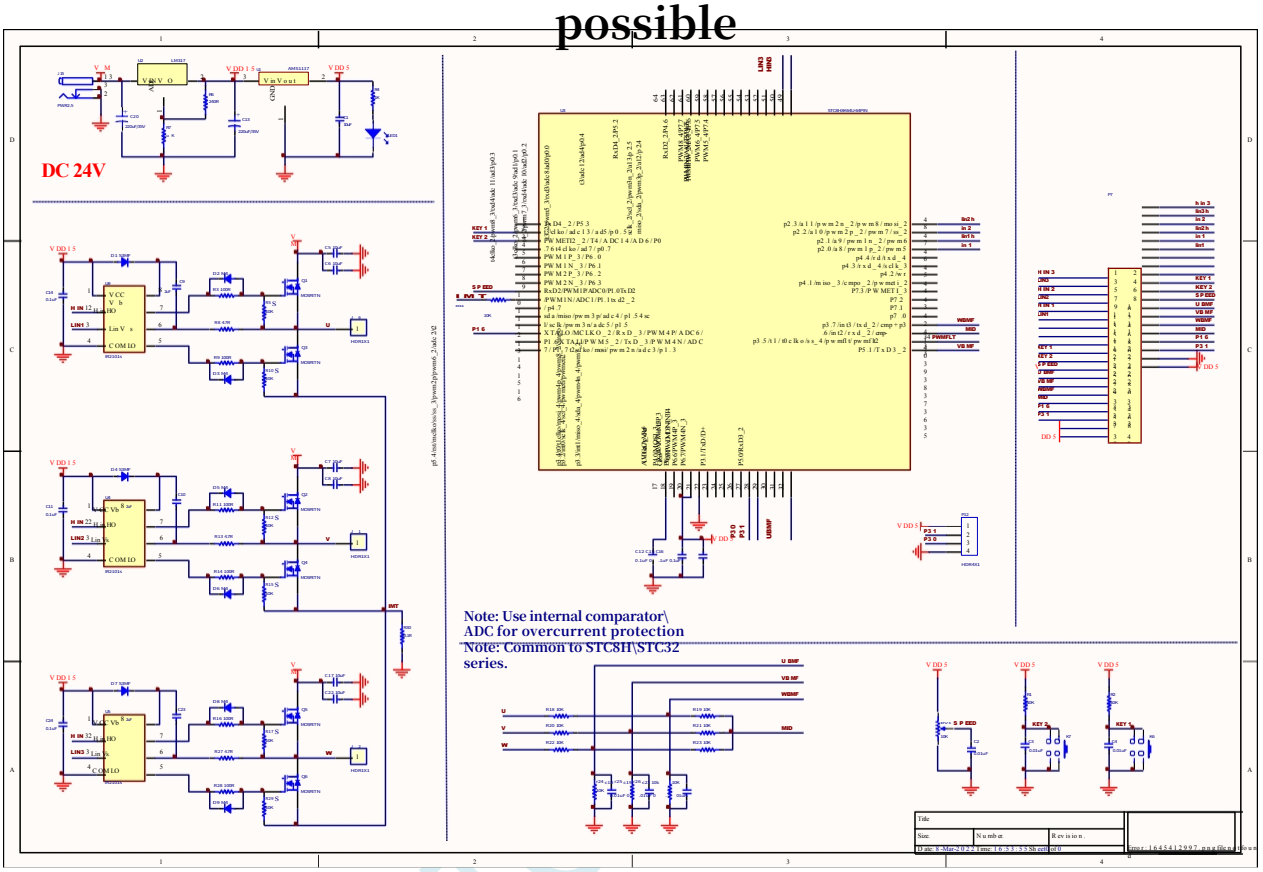
```
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 |= PWMA_FORCE_INACTIVE;
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 |= PWMA_OCMODE_PWM2.
        break;
    case 3.
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 |= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 |= PWMA_FORCE_ INACTIVE;
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 |= PWMA_OCMODE_PWM2;
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 &= CC2N_ POLARITY_LOW;
        PWMA_CCER2 |= CC3N_POLARITY_HIGH.
        break;
    case 2.
        PWMA_CCER1 &= CC1N_POLARITY_LOW;
        PWMA_CCER1 &= CC2N_POLARITY_LOW;
        PWMA_CCER2 |= CC3N_POLARITY_HIGH;
        PWMA_CCMR1 &= ~PWMA_OCMODE_ MASK;
        PWMA_CCMR1 |= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 |= PWMA_OCMODE_PWM2;
        PWMA_CCMR3 &= ~PWMA_OCMODE_ MASK;
        PWMA_CCMR3 |= PWMA_FORCE_INACTIVE.
        break;
    case 6.
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 |= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 |= PWMA_OCMODE_PWM2;
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 |= PWMA_FORCE_INACTIVE;
        PWMA_CCER1 &=~PWMA_FORCE_INACTIVE
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 |= PWMA_FORCE_INACTIVE;
        PWMA_CCER1 |= CC1N_POLARITY_HIGH;
        PWMA_CCER1 &= CC2N_POLARITY_LOW
        PWMA_CCER1 |= CC1N_POLARITY_HIGH;
        PWMA_CCER1 &= CC2N_POLARITY_LOW
        break;
    case 4.
        PWMA_CCER1 |= CC1N_POLARITY_HIGH;
        PWMA_CCER1 &= CC2N_POLARITY_LOW;
        PWMA_CCER2 &= CC3N_POLARITY_LOW;
        PWMA_CCMR1 &= ~PWMA_OCMODE_ MASK;
        PWMA_CCMR1 |= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 |= PWMA_FORCE_INACTIVE;
        PWMA_CCMR3 &= ~PWMA_ OCMODE_MASK;
        PWMA_CCMR3 |= PWMA_OCMODE_PWM2.
        break;
    case 5.
        PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR1 |= PWMA_FORCE_INACTIVE;
        PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR2 |= PWMA_FORCE_ INACTIVE;
        PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
        PWMA_CCMR3 |= PWMA_OCMODE_PWM2;
```

PWMA_CCER1 &= CC1N_POLARITY_LOW;
PWMA_CCER1 |= CC2N_POLARITY_HIGH;
PWMA_CCER1 |= CC2N_POLARITY_HIGH;
PWMA_CCER1 |= CC2N_POLARITY_HIGH
POLARITY_HIGH; PWMA_CCER2 &=
CC3N_POLARITY_LOW.

```
            break;
        }
        ADC_result = (ADC_Convert(RV09_CH)/3);

        for(temp = PWMA_STPulse; temp > ADC_result; temp--)
        {
            PWMA_CCR1H = (u8)(temp >> 8);                    //counter compare value
            PWMA_CCR1L = (u8)(temp);
            PWMA_CCR2H = (u8)(temp >> 8);
            PWMA_CCR2L = (u8)(temp);
            PWMA_CCR3H = (u8)(temp >> 8);
            PWMA_CCR3L = (u8)(temp).
            DelayXms(10).
        }
}

void MOTOR_STOP()
{
        PWMA_BKR &= ~0x80.
        PWMA_IER &= ~0xA0.
}

void LED4_Display (u16 dat,u8 num)
{
        switch(num)
        {
        case 0x01.
            LED_OUT(LED_0F[(dat/1)%10]);
            LED_OUT(0x01).
            RCLK = 0;
            RCLK = 1;
            break;
        case 0x02.
            LED_OUT(LED_0F[(dat/10)%10]);
            LED_OUT(0x02).
            RCLK = 0;
            RCLK = 1;
            break;
        case 0x04.
            LED_OUT(LED_0F[(dat/100)%10]);
            LED_OUT(0x04).
            RCLK = 0;
            RCLK = 1;
            break;
        case 0x08.
            LED_OUT(LED_0F[(dat/1000)%10]);
            LED_OUT(0x08).
            RCLK = 0;
            RCLK = 1;
            break;
        }
}

void LED_OUT(u8 X)
{
        u8 i;

        for(i=8;i>=1;i--)
        {
```

```
            if (X&0x80) DIO=1;
            else DIO=0;
            X<<=1;
            SCLK = 0.
            SCLK = 1;
        }
}

unsigned char KEY_detect()
{
        if(!P02)
        {
            DelayXms(10);
            if(!P02)
            {
                return 1;
            }
            else return 0;
        }
        else if(!P03)
        {
            DelayXms(10);
            if(!P03)
            {
                return 2;
            }
            else return 0;
        }
        else return 0;
}
```

23.8.2 BLDC Brushless DC Motor Drive (without HALL) -Hall-free 100,000 rpm operation possible




```
//Tested operating frequency is 11.0592MHz
//Tested operating frequency is 11.0592MHz
//This routine fulfils the following function: Control of a Hall-less motor via 3 PWM channels.
//This routine is for 57BL02 motor demonstration at 24V without load only.

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

typedef unsigned char  u8;
typedef unsigned int   u16.

#define  TRUE   1
#define  FALSE  0

#define  RV09_CH  6

#define  PWMA_Period ((u16)280)
#define  PWMA_STPulse
```

```
((u16)145)

#define    START           0x1A
#define    RUN             0x1B
```

```
#define    STOP                 0x1C
#define    IDLE                 0x1D

#define    PWMA_OCMODE_MASK         ((u8)0x70)
#define    PWMA_OCCE_ENABLE         ((u8)0x80)
#define    PWMA_OCCE_DISABLE        ((u8)0x00)
#define    PWMA_OCMODE_TIMING       ((u8)0x00)
#define    PWMA_OCMODE_ACTIVE       ((u8)0x10)
#define    PWMA_OCMODE_INACTIVE     ((u8)0x20)
#define    PWMA_OCMODE_TOGGLE       ((u8)0x30)
#define    PWMA_FORCE_INACTIVE      ((u8)0x40)
#define    PWMA_FORCE_ACTIVE        ((u8)0x50)
#define    PWMA_OCMODE_PWM1         ((u8)0x60)
#define    PWMA_OCMODE_PWM2         ((u8)0x70)
#define    CC1_POLARITY_HIGH        ((u8)0x02)
#define    CC1N_POLARITY_HIGH       ((u8)0x08)
#define    CC2_POLARITY_HIGH        ((u8)0x20)
#define    CC2N_POLARITY_HIGH       ((u8)0x80)
#define    CC1_POLARITY_LOW         ((u8)~0x02)
#define    CC1N_POLARITY_LOW        ((u8)~0x08)
#define    CC2_POLARITY_LOW         ((u8)~0x20)
#define    CC2N_POLARITY_LOW        ((u8)~0x80)
#define    CC1_OCENABLE             ((u8)0x01)
#define    CC1N_OCENABLE            ((u8)0x04)
#define    CC2_OCENABLE             ((u8)0x10)
#define    CC2N_OCENABLE            ((u8)0x40)
#define    CC1_OCDISABLE            ((u8)~0x01)
#define    CC1N_OCDISABLE           ((u8)~0x04)
#define    CC2_OCDISABLE            ((u8)~0x10)
#define    CC2N_OCDISABLE           ((u8)~0x40)
#define    CC3_POLARITY_HIGH        ((u8)0x02)
#define    CC3N_POLARITY_HIGH       ((u8)0x08)
#define    CC4_POLARITY_HIGH        ((u8)0x20)
#define    CC4N_POLARITY_HIGH       ((u8)0x80)
#define    CC3_POLARITY_LOW         ((u8)~0x02)
#define    CC3N_POLARITY_LOW        ((u8)~0x08)
#define    CC4_POLARITY_LOW         ((u8)~0x20)
#define    CC4N_POLARITY_LOW        ((u8)~0x80)
#define    CC3_OCENABLE             ((u8)0x01)
#define    CC3N_OCENABLE            ((u8)0x04)
#define    CC4_OCENABLE             ((u8)0x10)
#define    CC4N_OCENABLE            ((u8)0x40)
#define    CC3_OCDISABLE            ((u8)~0x01)
#define    CC3N_OCDISABLE           ((u8)~0x04)
#define    CC4_OCDISABLE            ((u8)~0x10)
#define    CC4N_OCDISABLE           ((u8)~0x40)

void UART_INIT();
void DelayXus(unsigned char delayTime);
void DelayXms( unsigned char delayTime);
unsigned int ADC_Convert(u8 ch).
void PWM_Init(void);
void SPEED_ADJ().
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP().
unsigned char KEY_detect();

unsigned char
```

```
Timer0_cnt=0x00;
```

```
unsigned int HA=0;
unsigned int Motor_speed.
unsigned char Motor_sta = IDLE;
unsigned char BRK_occur=0;
unsigned int PWMB_CAP1_v=0;
unsigned int CAP1_avg=0;
unsigned char CAP1_cnt=0;
unsigned long CAP1_sum=0;
unsigned char

void main(void)
{
    unsigned int temp=0;
    unsigned int
    ADC_result=0;

    CKCON = 0x00;                    //Set the external data bus speed to fastest
    WTST = 0x00;                     //set the program code wait parameter.
                                     //Assign a value of 0 to set the CPU to execute the
                                     programme as fast as possible.

    P_SW2= 0x80;
    P1 = 0x00;
    P0M1 = 0x0C;
    P0M0 = 0x01;
    P1M1 = 0xc0;
    P1M0 = 0x3F;
    P2M1 = 0x00;
    P2M0 = 0x38;
    P3M1 = 0x88;
    P3M0 = 0x02.

    ET0=1;
    TR0=0.
    ADCCFG = 0x0f;
    ADC_CONTR = 0x80.

    PWMA_ENO = 0x3F.                 //PWMA output enable
    PWMB_ENO = 0x00.                 //PWMB output enable
    PWMA_PS = 0x00;                          //PWMApin select
    PWMB_PS = 0xD5.                  //PWMB pin selection

    /*---------------------------------------------------------*
    Output Compare Mode PWMx_duty = [CCRx/(ARR + 1)]*100
    *---------------------------------------------------------*/
    /************PWMB BMF Input ****************/
    ////////// Time base unit //////////
    PWMB_PSCRL = 15;
    PWMB_ARRH = 0xff.                //auto-reload register, counter overflow point
    PWMB_ARRL = 0xff.
    PWMB_CCR8H = 0x00;
    PWMB_CCR8L = 0x05.
    ////////// Channel Configuration //////////
    PWMB_CCMR1 = 0xf3; //Channel mode configuration.   //Channel mode configuration
    PWMB_CCMR2 = 0xf1.
    PWMB_CCMR3 = 0xf1;
    PWMB_CCMR4 = 0x70;
    PWMB_CCER1 = 0x11;
    PWMB_CCER2 = 0x11.
```

```
//////////// Mode Configuration ////////////
    PWMB_CR2 = 0xf0.
    PWMB_CR1 = 0x81.
```

```
    PWMB_SMCR = 0x44.
    //////////// Enabling & Interrupt Configuration ////////////
    PWMB_BKR = 0x80.                                    //main output enable
    PWMB_IER = 0x02.                                    //Enable interrupt
    /***********PWMA Control Motor Phase Change ***************/
    //////////// Time base unit ////////////
    PWMA_PSCRH = 0x00.                                  //pre-scaler register
    PWMA_PSCRL = 0x00;
    PWMA_ARRH = (u8)(PWMA_Period >> 8);
    PWMA_ARRL = (u8)(PWMA_Period).
    //////////// Channel Configuration ////////////
    PWMA_CCMR1 = 0x70; //Channel mode configuration.      //Channel mode configuration
    PWMA_CCMR2 = 0x70.
    PWMA_CCMR3 = 0x70.
    PWMA_CCER1 = 0x11; //Configure the channel output enable and polarity.    //configure channel output enable and polarity
    PWMA_CCER2 = 0x01; //Configure the channel output enable and polarity.    //configure channel output enable and polarity
    PWMA_OISR = 0xAA; //Configure the output level of each channel when MOE=0.    //Configure the output level of each channel when MOE=0.
    //////////// Mode Configuration ////////////
    PWMA_CR1 = 0xA0.
    PWMA_CR2 = 0x24;
    PWMA_SMCR = 0x20;
    PWMA_BKR = 0x0c.
    //////////// Enabling & Interrupt Configuration ////////////
    PWMA_CR1 |= 0x01; //Enable counter.                 //Enable counter
    EA = 1;

    UART_INIT();

    while (1)
    {
        switch(Motor_sta)
        {
            case START.
                MOTOR_START().
                Motor_sta = RUN.
                for(temp = PWMA_STPulse; temp > ADC_result; temp--)          //open loop start
                {
                    ADC_result = (ADC_Convert(RV09_CH)/4);
                    PWMA_CCR1H = (u8)(temp >> 8);
                    PWMA_CCR1L = (u8)(temp);
                    PWMA_CCR2H = (u8)(temp >> 8);
                    PWMA_CCR2L = (u8)(temp);
                    PWMA_CCR3H = (u8)(temp >> 8);
                    PWMA_CCR3L = (u8)(temp).
                    DelayXms(10).
                }
                break;
            case RUN.
                SPEED_ADJ().                            //Motor speed control
                if((BRK_occur == TRUE))
                Motor_sta = STOP;
                break;
            case STOP.
                MOTOR_STOP().
                Motor_sta = IDLE;
                break;
            case IDLE.
```

```
if(KEY_detect()==1)
Motor_sta = START;                    //Start the motor
```

```
                BRK_occur = FALSE;
                Motor_speed = 0;
                CAP1_avg = 0;
                CAP1_cnt = 0;
                CAP1_sum = 0;
                break;
            }
        }
    }

    void TIM0_ISR() interrupt 1
    {
        if(Motor_sta == START)
        {
            if(Timer0_cnt<0xe0) Timer0_cnt++;
            TH0=Timer0_cnt.

            switch(HA%6)
            {
            case 0.
                PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
                PWMA_CCMR3 |= PWMA_FORCE_INACTIVE;
                PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
                PWMA_CCMR1 |= PWMA_OCMODE_PWM2.
                break;
            case 1.
                PWMA_CCER1 &= CC2N_POLARITY_LOW;
                PWMA_CCER2 |= CC3N_POLARITY_HIGH.
                  break;
            case 2.
                PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
                PWMA_CCMR1 |= PWMA_FORCE_INACTIVE;
                PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
                PWMA_CCMR2 |= PWMA_OCMODE_PWM2.
                break;
            case 3.
                PWMA_CCER1 |= CC1N_POLARITY_HIGH;
                PWMA_CCER2 &= CC3N_POLARITY_LOW.
                break;
            case 4.
                PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
                PWMA_CCMR2 |= PWMA_FORCE_INACTIVE;
                PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
                PWMA_CCMR3 |= PWMA_OCMODE_PWM2.
                break;
            case 5.
                PWMA_CCER1 &= CC1N_POLARITY_LOW;
                PWMA_CCER1 |= CC2N_POLARITY_HIGH.
                break;
            }
            HA++.
        }

        if(Motor_sta == RUN)
        {
            TR0=0;
            switch(RD_HALL())
            {
            case 3.
```

```
            PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR3 |= PWMA_FORCE_INACTIVE;
            PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR1 |= PWMA_OCMODE_PWM2.
            break;
        case 1.
            PWMA_CCER1 &= CC2N_POLARITY_LOW;
            PWMA_CCER2 |= CC3N_POLARITY_HIGH.
            break;
        case 5.
            PWMA_CCMR1 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR1 |= PWMA_FORCE_INACTIVE;
            PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR2 |= PWMA_OCMODE_PWM2.
            break;
        case 4.
            PWMA_CCER1 |= CC1N_POLARITY_HIGH;
            PWMA_CCER2 &= CC3N_POLARITY_LOW.
            break;
        case 6.
            PWMA_CCMR2 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR2 |= PWMA_FORCE_INACTIVE;
            PWMA_CCMR3 &= ~PWMA_OCMODE_MASK;
            PWMA_CCMR3 |= PWMA_OCMODE_PWM2.
            break;
        case 2.
            PWMA_CCER1 &= CC1N_POLARITY_LOW;
            PWMA_CCER1 |= CC2N_POLARITY_HIGH.
            break;
        }
    }
}

void PWMA_ISR() interrupt 26
{
    if((PWMA_SR1 & 0x20))
    {
        P00=0;
        CAP1_sum += PWMB_CAP1_v;
        CAP1_cnt++;
        if(CAP1_cnt==128)
        {
            CAP1_cnt=0;
            CAP1_avg = (CAP1_sum>>7);
            CAP1_sum = 0;
            Motor_speed = 5000000/CAP1_avg.
        }
        PWMA_SR1 &=~0x20.                       //Clear
    }
    if((PWMA_SR1 & 0x80))                       //BRK
    {
        BRK_occur = TRUE;
        PWMA_SR1 &=~0x80.                       //Clear
    }
}

void PWMB_ISR() interrupt 27
{
    unsigned char ccr_tmp=0;
```

```
        if((PWMB_SR1 & 0X02))
        {
            ccr_tmp = PWMB_CCR5H;
            if(ccr_tmp>1)                                   //software filter
            {
                PWMB_CAP1_v = ccr_tmp;
                PWMB_CAP1_v = (PWMB_CAP1_v<<8) + PWMB_CCR5L;
                if(Motor_sta == RUN)                        //Commutation delay timing
                {
                    TR0=1;
                    TH0 = 256-(PWMB_CAP1_v>>9);
                }
            }
            PWMB_SR1 &=~0X02.
        }
}

void UART_INIT()
{
        scon   =   0x50;                                    //8-bit variable baud rate
        auxr   =   0x40;                                    //Timer 1 is in 1T mode.
        tmod = 0x20.                                        //Timer 1 is mode 0 (16-bit
                                                            auto-reload)

        TL1 = 254.
        TH1 = 254.
//      ET1 = 0;
        TR1 = 1;
}

void DelayXus(unsigned char delayTime)
{
        int i = 0;
        while( delayTime--)
        {
            for( i = 0 ; i < 1 ; i++).
        }
}

void DelayXms( unsigned char delayTime )
{
        int i = 0;
        while( delayTime--)
        {
            for( i = 0 ; i < 2 ; i++)
            {
                DelayXus(100).
            }
        }
}

unsigned int ADC_Convert(u8 ch)
{
        u16 res=0;

        ADC_CONTR &= ~0x0f.
        ADC_CONTR |= ch.
        ADC_CONTR |= 0x40.
        DelayXus(1).
```

```
    while (! (ADC_CONTR & 0x20));
    ADC_CONTR &= ~0x20.

    res = ADC_RES.
    res = (res<<2)+(ADC_RESL>>6);

    if (res < 360) res=360;
    if (res > 900) res=900;

    return res;
}

void SPEED_ADJ()
{
    u16 ADC_result.

    ADC_result = (ADC_Convert(RV09_CH)/4);          //Speed knob ADC sampling
    PWMA_CCR1H = (u8)(ADC_result >> 8);             //counter compare value
    PWMA_CCR1L = (u8)(ADC_result);
    PWMA_CCR2H = (u8)(ADC_result >> 8);
    PWMA_CCR2L = (u8)(ADC_result);
    PWMA_CCR3H = (u8)(ADC_result >> 8);
    PWMA_CCR3L = (u8)(ADC_result);
}

unsigned char RD_HALL()                             //Read Hall Sensor
{
    unsigned char Hall_sta = 0;

    DelayXus(40).
    (P17)? (Hall_sta|=0x01) : (Hall_sta&=~0x01);
    (P54)? (Hall_sta|=0x02) : (Hall_sta&=~0x02);
    (P33)? (Hall_sta|=0x04) : (Hall_sta&=~0x04);
    (P33)?

    return Hall_sta;
}

void MOTOR_START()
{
    PWMA_CCR1H = (u8)(PWMA_STPulse >> 8);           //counter compare value
    PWMA_CCR1L = (u8)(PWMA_STPulse);
    PWMA_CCR2H = (u8)(PWMA_STPulse >> 8);
    PWMA_CCR2L = (u8)(PWMA_STPulse);
    PWMA_CCR3H = (u8)(PWMA_STPulse >> 8);
    PWMA_CCR3L = (u8)(PWMA_ STPulse);
    PWMA_BKR |= 0x80.                               //Main output enable is equivalent to the master switch
    PWMA_IER = 0x00.                                //Enable interrupt
    TR0 = 1;

    while (HA < 6*20);

    PWMA_IER = 0xa0.                                //Enable interrupt
}

void MOTOR_STOP()
{
    PWMA_BKR &= ~0x80.
    PWMA_IER &= ~0x20.
}
```

```
unsigned char KEY_detect()
{
    if(!P37)
    {
        DelayXms(10);
        if(!P37)
        {
            return 1;
        }
        else return 0;
    }
    else if(!P03)
    {
        DelayXms(10);
        if(!P03)
        {
            return 2;
        }
        else return 0;
    }
    else return 0;
}
```

23.8.3 Encoder Implementation with Advanced PWM

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files

#define   MAIN_Fosc   11059200L// Define Master Clock

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

The PWMA module operates in encoder mode. The PWMA module can only be connected to one encoder.
Serial port 1 (RXD-->P3.0 TXD-->P3.1) returns the reading result, serial port setting 115200,8,n,1.
Encoder A phase input: PWM1P (P1.0)
Encoder B phase input: PWM2P (P1.2)

 Encoder modes.          Mode 1: Add or subtract two
                                 edges per pulse 2.
          Mode 2: Add or subtract two edges per pulse 2.
          Mode 3: Add or subtract two edges per pulse 4.

*****************************************************/

unsigned int    pulse;                  // Encoder pulses
bit             B_Change.               // Encoder count
                                        change

bit             B_TX1_Busy.             // Send busy flag

void            PWMA_config(void);
void            UART1_config(unsigned long brt);    // brt:
void            UART1_TxByte(unsigned char dat).    communication
```

baud rate

```
void main(void)
{
    unsigned int j.

    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the programme as fast as possible.


    p1m1 = 0x00;
    p1m0 = 0x00.

    UART1_config(115200UL).             // brt: communication baud rate

    EA = 1;

    PWMA_config();
    pulse = 10;

    while (1)
    {
        if(B_Change)
        {
            B_Change = 0;
            j = pulse.
            UART1_TxByte(j/10000+'0');          //convert to decimal text and send it
            UART1_TxByte((j%10000)/1000+'0');
            UART1_TxByte((j%1000)/100+'0');
            UART1_TxByte((j%100)/10+'0');
            UART1_TxByte( j%10+'0').
            UART1_TxByte(0x0d);
            UART1_TxByte(0x0a).
        }
    }
}

//========================================================================
// Function: void PWMA_config(void)
// Description: PPWM configuration function.
// Parameters: noe.
// Returns: none.
// Version: V1.0, 2021-5-10
// Remarks.
//========================================================================
void PWMA_config(void)
{
    PWMA_PSCR = 0;                      //pre-scaler Fck_cnt = Fck_psc/(PSCR[15:0]+1),
                                        //Edge Aligned PWM Frequency =SYSclk/((PSCR+1)*(ARR+1)),
                                        // Central alignment frequency =SYSclk/((PSCR+1)*(ARR+1)*2).
    PWMA_ARR = 0xffff.                  //Auto Reload Register, control PWM cycle.
    PWMA_CNTR = 0;                      //Clear the encoder counter value
    PWMA_ENO = 0;                       //IO disable output PWM

    PWMA_CCMR1 = 0x01+(10<<4); //Channel 1 mode configuration, configure as input channel.      //Channel 1 mode configuration, configured as input channel, //Channel 1 mode configuration, configured as input channel.
```

//0~15 Corresponding to the number of input filter clocks.
//1 2 4 8 12 16 24 32 48 64 80 96 128 160 192 256

PWMA_CCMR2 = 0x01+(10<<4); //Channel 2 mode configuration, configure as input channel. //Channel 2 mode configuration, configured as input channel.

//Channel 2 mode configuration, configured as input channel.
//0~15 Corresponding to the number of input filter clocks.

```
                                                        //1 2 4 8 12 16 24 32 48 64 80 96 128 160 192 256
    PWMA_SMCR = 2; //Encoder mode, mode 1 or mode 2: add or subtract two edges for each pulse.      //Encoder mode, mode 1 or mode 2: add or subtract 2 to both edges of each pulse.
                                                        //Mode 3: Add or subtract four edges per pulse 4.
    PWMA_CCER1 = 0x55.                                  //Configure channel input enable and polarity, allow input, falling edge
    PWMA_PS = 0; //IO select P1.0 P1.2.                 //IO select P1.0 P1.2
    PWMA_IER = 0x02.                                    //Enable interrupt
    PWMA_CR1 = 0x01.                                    //Enable counter to allow automatic reloading of register buffers.
                                                        //Edge aligned mode, count up, /etc.
                                                        //bit7=1: write auto-reload register buffer (will not be disturbed in this cycle), //bit7=1: write auto-reload register buffer (will not be disturbed in this cycle)
                                                        //    =0: Write auto-reload register book directly (cycle may be messed up)
}

//========================================================================
// Function: void PWMA_ISR(void) interrupt PWMA_VECTOR
// Description: PWMA interrupt handler.
// Parameters: None
// Returns: none.
// Version: V1.0, 2021-6-1
//========================================================================
void PWMA_ISR(void) interrupt 26
{
    if(PWMA_SR1 & 0x02)                                 //Encoder interrupt
    {
        pulse = PWMA_CNTR.                              //read the current encoder count value
        B_Change = 1;                                   //flag existing capture value
    }
    PWMA_SR1 = 0;
}

//========================================================================
// Function:            voidUART1_config(u32 brt)
// Description: UART1 initialisation function.
// Parameters: brt.     brt.    Baud rate.
// Returns: none.
// Version: VER1.0
// Date: 2018-4-2
// Remarks.
//========================================================================
void UART1_config(unsigned long brt)                    // brt: communication baud rate
{
    Brt = 65536UL - (MAIN_Fosc / 4) / brt.
    AUXR &= ~0x01.                                      //S1 BRT Use Timer1.
    AUXR |= (1<<6); //Timer1 set as 1T mode.            //Timer1 set as 1T mode
    TMOD &= 0x0f.                                       //Timer1 16bits AutoReload.
    TH1 = (unsigned char)(brt >> 8);
    TL1 = (unsigned char)brt.
    TR1 = 1; // Run Timer1.                             // Run Timer1
    P_SW1 &= ~0xc0; //Serial port 1 switches to P3.0 P3.1.    //Serial port 1 switch to P3.0 P3.1
    SCON = (SCON & 0x3f) | (1<<6); // 8-bit data, 1 start bit, 1 stop bit, no parity.    // 8-bit data, 1-bit start, 1-bit stop, no parity
    ES = 1;                                             // Allow interrupt
    REN = 1;                                            // Allow to receive
}

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

```
// Function: void UART1_TxByte(u8 dat)
// Description: Serial 1 query send a byte function.
// Parameters: dat: Bytes of data to send.
// Returns: none.
// Version: VER1.0
```

```
// Date: 2018-4-2
// Remarks.
//========================================================================
void UART1_TxByte(unsigned char dat)
{
    B_TX1_Busy = 1; //Flag transmit busy.          //flag transmission busy
    SBUF = dat;                                   //send a byte
    while(B_TX1_Busy).                            //Wait for the transmission to complete
}

//========================================================================
// Function: void UART1_int      (void) interrupt UART1_VECTOR
// Description: Serial 1 interrupt function
// Parameters: none.
// Returns: none.
// Version: VER1.0
// Date: 2018-4-2
// Remarks.
//========================================================================
void UART1_int (void) interrupt 4
{
    if(RI)
        RI = 0;

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

23.8.4 Quadrature encoder mode

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                               // see download software for header files
#include "intrins.h"

unsigned char cnt_H, cnt_L;

void main(void)
{
    EAXFR = 1;                                    //Enable access to XFR
    CKCON = 0x00;                                 //Set the external data bus speed to fastest
    WTST = 0x00;                                  //set the program code wait parameter.
                                                  //Assign a value of 0 to set the CPU to execute the
                                                  programme as fast as possible.

    p1m1 = 0x0f;
    p1m0 = 0x00.

    PWMA_ENO = 0x00; //Configure the TRGI pin to turn off the corresponding bit of ENO and assign it to input.
    //Configure the TRGI pin to switch off the corresponding bit of ENO and configure it as input.
    PWMA_PS = 0x00;                               //00:PWM at P1
```

```
    PWMA_PSCRH = 0x00.                          //pre-scaler register
```

```
    PWMA_PSCRL = 0x00;

    PWMA_CCMR1    = 0x21.                    //Channel mode configured as input to encoder, filter 4 clocks
    PWMA_CCMR2    = 0x21.                    //Channel mode configured as input to encoder, filter 4 clocks

    PWMA_SMCR     = 0x03.                    //Encoder mode 3

    PWMA_CCER1    = 0x55.                    //Configure channel enable and polarity
    PWMA_CCER2    = 0x55.                    //Configure channel enable and polarity

    PWMA_IER =  0x02.                        //Enable interrupt

    PWMA_CR1 |= 0x01; //Enable counter.      //Enable counter

    EA = 1;

    while (1);
}

/********************* PWM interrupt read encoder count value
 ************************/ void PWMA_ISR() interrupt 26
{
    if (PWMA_SR1 & 0X02)
    {
        P03 = ~P03.
        cnt_H = PWMA_CCR1H;
        cnt_L = PWMA_CCR1L;
        PWMA_SR1 &= ~0X02;
    }
}
```

23.8.5 Single pulse mode (trigger control pulse output)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          //see download software for header files
#include "intrins.h"

void main(void)
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

    p0m1 = 0x00;
    p0m0 = 0xff;
    p1m1 = 0x0c;
    p1m0 = 0xf3.

    PWMA_ENO = 0xF3.                         //IO output PWM
    PWMA_PS = 0x00;                          //00:PWM at P1

    /*------------------------------------------------------------*
```

```
  PWMA_duty = [CCRx/(ARR + 1)]*100
  *------------------------------------------------------------*/
```

```
    //Configure the TRGI pin to switch off the corresponding bit of ENO and configure it as input.
    PWMA_PSCRH = 0x00.                          //pre-scaler register
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x00.                            // Dead time configuration

    PWMA_CCMR1 = 0x68; //Channel mode configuration.   //Channel mode configuration
    PWMA_CCMR2 = 0x01;                          //configured as input channel
    PWMA_CCMR3 = 0x68.
    PWMA_CCMR4 = 0x68;

    PWMA_SMCR = 0x66.

    PWMA_ARRH = 0x08.                           //auto-reload register, counter overflow point
    PWMA_ARRL = 0x00.

    PWMA_CCR1H = 0x04;                          //counter
    compare value PWMA_CCR1L = 0x00;
    PWMA_CCR2H = 0x02;
    PWMA_CCR2L = 0x00;
    PWMA_CCR3H = 0x01;
    PWMA_CCR3L = 0x00;
    PWMA_CCR4H = 0x01;
    PWMA_CCR4L = 0x00.

    PWMA_CCER1 = 0x55.                          //configure channel output enable and polarity
    PWMA_CCER2 = 0x55.                          //configure channel output enable and polarity

    PWMA_BKR = 0x80.                            //Main output enable Equivalent to master switch
    PWMA_IER = 0x02.                            //Enable interrupt
    PWMA_CR1 = 0x08.                            //single pulse mode
    PWMA_CR1 |= 0x01; //Enable counter.         //Enable counter

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if (PWMA_SR1 & 0X02)
    {
        P03 = ~P03.
        PWMA_SR1 &= ~0X02.
    }
}
```

23.8.6 Gated mode (input level enables counter)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                             // see download software for header files
#include "intrins.h"

void main(void)
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
```

```
    WTST = 0x00;                                    //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.

    p0m1 = 0x00;
    p0m0 = 0xff;
    p1m1 = 0x00;
    p1m0 = 0xFF;
    p3m1 = 0x04;
    p3m0 = 0x00.

    PWMA_ENO = 0xFF.                                //IO output PWM
    PWMA_PS = 0x00;                                 //00:PWM at P1

/*********************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
*********************************************************/
//Configure the TRGI pin to switch off the corresponding bit of ENO and configure it as input.
    PWMA_PSCRH = 0x00.                              //pre-scaler register
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x00.                                //Dead time configuration

    PWMA_CCMR1 = 0x68; //Channel mode configuration.  //Channel mode configuration
    PWMA_CCMR2 = 0x68.                              //configured as input channel
    PWMA_CCMR3 = 0x68.
    PWMA_CCMR4 = 0x68.

    PWMA_SMCR = 0x75.                               //Gated Trigger Mode ETRF Input

    PWMA_ARRH = 0x08.                               //auto-reload register, counter overflow point
    PWMA_ARRL = 0x00.

    PWMA_CCR1H = 0x04.                              //counter comparison value
    PWMA_CCR1L = 0x00;                              //
    PWMA_CCR2H = 0x02.                              //
    PWMA_CCR2L = 0x00;                              //
    PWMA_CCR3H = 0x01.                              //
    PWMA_CCR3L = 0x00;                              //
    PWMA_CCR4H = 0x01.                              //
    PWMA_CCR4L = 0x00;                              //

    PWMA_CCER1 = 0x55.                              //configure channel output enable and polarity
    PWMA_CCER2 = 0x55.                              //configure channel output enable and polarity

    PWMA_BKR = 0x80.                                //Main output enable Equivalent to master switch
    PWMA_IER = 0x02.                                //Enable interrupt

    PWMA_CR1 |= 0x01; //Enable counter.             //Enable counter

    EA = 1;
    while (1) ;
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P03 = ~P03.
        PWMA_SR1 &= ~0X02.
    }
```

}

23.8.7 External Clock Mode

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     // see download software for header files
#include "intrins.h"

void main(void)
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

    p0m1 = 0x00;
    p0m0 = 0xff;
    p1m1 = 0x00;
    p1m0 = 0xFF;
    p3m1 = 0x04;
    p3m0 = 0x00.

    PWMA_ENO = 0xFF.                    //IO output PWM
    PWMA_PS = 0x00;                     //00:PWM at P1

/*********************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
*********************************************************/
//Configure the TRGI pin to switch off the corresponding bit of ENO and configure it as input.
    PWMA_PSCRH = 0x00.                  //pre-scaler register
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x00.                    // Dead time configuration

    PWMA_CCMR1 = 0x68; //Channel mode configuration.   //Channel mode configuration
    PWMA_CCMR2 = 0x68.                  //configured as input channel
    PWMA_CCMR3 = 0x68.
    PWMA_CCMR4 = 0x68.

    PWMA_SMCR = 0x77.                   //ETRF input

    PWMA_ARRH = 0x08.                   //auto-reload register, counter overflow point
    PWMA_ARRL = 0x00.

    PWMA_CCR1H = 0x04;                  //counter
    compare value PWMA_CCR1L = 0x00;
    PWMA_CCR2H = 0x02;
    PWMA_CCR2L = 0x00;
    PWMA_CCR3H = 0x01;
    PWMA_CCR3L = 0x00;
    PWMA_CCR4H = 0x01;
    PWMA_CCR4L = 0x00.

    PWMA_CCER1 = 0x55.                  //configure channel output enable and polarity
    PWMA_CCER2 = 0x55.                  //configure channel output enable and polarity
```

```
    PWMA_BKR = 0x80.                    //Main output enable Equivalent to master switch
    PWMA_IER = 0x02.                    //Enable interrupt
    PWMA_CR1 |= 0x01; //Enable counter.  //Enable counter

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P03 = ~P03.
        PWMA_SR1 &= ~0X02.
    }
}
```

23.8.8 Input capture mode to measure pulse period (capture rising edge to rising edge or falling edge to falling edge)

Principle: Using the capture module CCx of one of the channels inside the advanced PWM, the rising or falling edge of the external port is captured, and the time between the two rising edges or between the two falling edges is the period of the pulse, i.e. the difference between the two captured counts is the period value.

Example: Use the first capture module CC1 capture function of PWMA to capture the rising edge on the PWM1P (P1.0) pin, and subtract the capture values of the previous and previous captures in the interrupt to get the period.

Note: Only PWM1P, PWM2P, PWM3P, PWM4P, PWM5, PWM6, PWM7, PWM8 pins and the corresponding switching pins have the capture function.

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                   // see download software for header files
#include "intrins.h"

int         cap;
int         cap_new;
int         cap_old;

void main(void)
{
    EAXFR = 1;                                        //Enable access to XFR
    CKCON = 0x00;                                     //Set the external data bus speed to fastest
    WTST = 0x00;                                      //set the program code wait parameter.
                                                      //Assign a value of 0 to set the CPU to execute the
                                                      programme as fast as possible.

    p0m1 = 0x00;
    p0m0 = 0x00;
    p1m1 = 0x00;
    p1m0 = 0x00.

/*The pin configured as TRGI needs to switch off the corresponding bit of ENO and be configured as input*/.
    PWMA_ENO = 0x00; //IO Output PWM.                 //IO output PWM
    PWMA_PS = 0x00;                                   //00:PWM at P1

    PWMA_CCMR1 = 0x01;                                //configured as
    input channel PWMA_SMCR = 0x56;
    PWMA_CCER1 = 0x01; //Configure the channel output enable and polarity.    //configure channel output enable and
    polarity

    PWMA_IER = 0x02.                                  //Enable interrupt
    PWMA_CR1 |= 0x01; //Enable counter.               //Enable counter

    EA = 1;
    while (1);
```

```
}

/* Channel 1 input, capture data read via PWMA_CCR1H /
PWMA_CCR1L */ void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0x02)
    {
```

```
            cap_old = cap_new;
            cap_new = PWMA_CCR1H;                           //read
            CCR1H cap_new = (cap_new << 8) + PWMA_CCR1L;    //read
            CCR1L
            cap = cap_new - cap_old;
            PWMA_SR1 &= ~0x02.
        }
}
```

23.8.9 Input capture mode measures pulse high level width (captures rising edge to falling edge)

Principle: Use the capture module CCx and CCx+1 of the two channels inside the advanced PWM to capture the same external pin at the same time, CCx captures the rising edge of this pin, CCx+1 captures the falling edge of this pin, and then use the capture value of CCx+1 to subtract the capture value of CCx, the difference of which is the width of the pulse high level.

Example: Use the first capture module CC1 and the second capture module CC2 of PWMA to capture the PWM1P pin (P1.0) at the same time, in which CC1 captures the rising edge of PWM1P and CC2 captures the falling edge of PWM1P, and use the capture value of CC2 to subtract the capture value of CC1 in interrupt, and the difference of which is the width of the pulse high level.

Note: 1. The two capture modules inside the chip are used to capture the same external pin at the same time, so there is no need to connect multiple external pins.

2、Only CC1+CC2, CC3+CC4, CC5+CC6, CC7+CC8 can complete the above functions. CC1+CC2 combination can capture PWM1P pin and PWM2P pin at the same time; CC3+CC4 combination can capture PWM3P pin and PWM4P pin at the same time; CC5+CC6 combination can capture PWM5 pin and PWM6 pin at the same time; CC7+CC8 combination can capture PWM7 pin and PWM8 pin at the same time; CC7+CC8 combination can capture PWM8 pin at the same time; CC7+CC8 combination can capture PWM8 pin at the same time; CC7+CC8 combination can capture PWM7 pin at the same time. CC6 combination can capture PWM5 pin and PWM6 pin at the same time; CC7+CC8 combination can capture PWM7 pin and PWM8 pin at the same time.

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                  //see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                                       //Enable access to XFR
    CKCON = 0x00;                                    //Set the external data bus speed to fastest
    WTST = 0x00;                                     //set the program code wait parameter.
                                                     //Assign a value of 0 to set the CPU to execute the
                                                     programme as fast as possible.

    p1m0 = 0x00;
    p1m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

                                                     //(CC1 captures TI1 rising edge, CC2 captures TI1
    PWMA_CCER1 = 0x00;                               falling edge)

    PWMA_CCMR1 = 0x01.                               //CC1 is input mode and mapped to TI1FP1.
    PWMA_CCMR2 = 0x02.                               //CC2 is input mode and mapped to TI1FP2.
    PWMA_CCER1 = 0x11; //Enable capture function on CC1/CC2.      //Enable capture function on CC1/CC2.
    PWMA_CCER1 |= 0x00; //Set capture polarity to rising edge of CC1.  //Set capture polarity to rising edge of CC1.
    PWMA_CCER1 |= 0x20; //Set capture polarity to falling edge of CC2.  //Set capture polarity to falling edge of CC2.
```

```
    PWMA_CR1 = 0x01.

    PWMA_IER = 0x04.                                    //Enable CC2 capture interrupt
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;
    unsigned int cnt1.
```

```
    unsigned int cnt2.

    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04.

        cnt1 = (PWMA_CCR1H << 8) + PWMA_CCR1L;
        cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
        cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
        cnt = cnt2 - cnt1.                          //The difference is the width of the high level
    }
}
```

23.8.10 Input capture mode measures pulse low width (captures falling edge to rising edge)

Principle: Use the capture module CCx and CCx+1 of two channels inside the advanced PWM to capture the same pin externally at the same time, CCx captures the falling edge of this pin, CCx+1 captures the rising edge of this pin, and then use the capture value of CCx+1 to subtract the capture value of CCx, the difference of which is the width of the pulse low level.

Example: Use the first capture module CC1 and the second capture module CC2 of PWMA to capture the PWM1P pin (P1.0) at the same time, where CC1 captures the falling edge of PWM1P and CC2 captures the rising edge of PWM1P, and in the interrupt, use the capture value of CC2 to subtract the capture value of CC1, the difference of which is the width of the pulse low level.

Note: 1. The two capture modules inside the chip are used to capture the same external pin at the same time, so there is no need to connect multiple external pins.

2 、 Only CC1+CC2, CC3+CC4, CC5+CC6, CC7+CC8 can complete the above functions. CC1+CC2 combination can capture PWM1P pin and PWM2P pin at the same time; CC3+CC4 combination can capture PWM3P pin and PWM4P pin at the same time; CC5+CC6 combination can capture PWM5 pin and PWM6 pin at the same time; CC7+CC8 combination can capture PWM7 pin and PWM8 pin at the same time; CC7+CC8 combination can capture PWM8 pin at the same time; CC7+CC8 combination can capture PWM8 pin at the same time; CC7+CC8 combination can capture PWM7 pin at the same time. CC6 combination can capture PWM5 pin and PWM6 pin at the same time; CC7+CC8 combination can capture PWM7 pin and PWM8 pin at the same time.

//Tested operating frequency is *11.0592MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                                   // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                                        //Enable access to XFR
    CKCON = 0x00;                                     //Set external data bus speed to fastest
    WTST = 0x00;                                      //set the program code wait parameter.
                                                      //Assign a value of 0 to set the CPU to execute the
                                                      programme as fast as possible.

    p1m0 = 0x00;
    p1m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

                                                      //(CC1 captures TI1 rising edge, CC2 captures TI1
    PWMA_CCER1 = 0x00;                                falling edge)

    PWMA_CCMR1 = 0x01.                                //CC1 is input mode and mapped to TI1FP1.
    PWMA_CCMR2 = 0x02.                                //CC2 is input mode and mapped to TI1FP2.
    PWMA_CCER1 = 0x11; //Enable capture function on CC1/CC2.      //Enable capture function on CC1/CC2.
    PWMA_CCER1 |= 0x00; //Set capture polarity to rising edge of CC1.  //Set capture polarity to rising edge of CC1.
    PWMA_CCER1 |= 0x20; //Set capture polarity to falling edge of CC2.  //Set capture polarity to falling edge of CC2.
    PWMA_CR1 = 0x01.
```

```
    PWMA_IER = 0x02.                    //Enable CC1 capture interrupt
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;
    unsigned int cnt1;
    unsigned int cnt2.
```

```
    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02.

        cnt1 = (PWMA_CCR1H << 8) + PWMA_CCR1L;
        cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
        cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
        cnt = cnt1 - cnt2.                                  //The difference is the width of the low level
    }
}
```

23.8.11 Input capture mode for simultaneous measurement of pulse period and duty cycle

Principle: Use the two-channel capture module CCx and CCx+1 inside the advanced PWM to capture the same external pin at the same time, CCx captures the rising edge of this pin, CCx+1 captures the falling edge of this pin, and at the same time enables the rising edge signal of this pin as the reset trigger signal, the capture value of CCx is the period, and the capture value of CCx+1 is the duty cycle.

Example: Use the first capture module CC1 and the second capture module CC2 of PWMA to capture the PWM1P pin (P1.0) at the same time, in which CC1 captures the rising edge of PWM1P and CC2 captures the falling edge of PWM1P, and set the rising edge signal of PWM1P as the reset trigger signal, and set the capture value of CC1 to be the period and the capture value of CC2 to be the duty cycle.

Note: 1. The two capture modules inside the chip are used to capture the same external pin at the same time, so there is no need to connect multiple external pins.

2、Only CC1+CC2, CC5+CC6 combination can complete the above function. CC1+CC2 combination can capture PWM1P pin and PWM2P pin at the same time; CC5+CC6 combination can capture PWM5 pin and PWM6 pin at the same time.

3、Because the reset trigger signal is set, the capture value is the cycle value and duty cycle value, and there is no need to subtract the previous capture value.

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                  //see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                                       //Enable access to XFR
    CKCON = 0x00;                                    //Set the external data bus speed to fastest
    WTST = 0x00;                                     //set the program code wait parameter.
                                                     //Assign a value of 0 to set the CPU to execute the
                                                     programme as fast as possible.

    p1m0 = 0x00;
    p1m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

                                                     //(CC1 captures TI1 rising edge, CC2 captures TI1
                                                     falling edge)
    PWMA_CCER1 = 0x00;                               //CC1 captures the cycle width, CC2 captures the high
                                                     level width.
    PWMA_CCMR1 = 0x01.                               //CC1 is input mode and mapped to TI1FP1.
    PWMA_CCMR2 = 0x02.                               //CC2 is input mode and mapped to TI1FP2.
    PWMA_CCER1 = 0x11; //Enable capture function on CC1/CC2.      //Enable capture function on CC1/CC2.
    PWMA_CCER1 |= 0x00; //Set capture polarity to rising edge of CC1.  //Set capture polarity to rising edge of CC1.
    PWMA_CCER1 |= 0x20; //Set capture polarity to falling edge of CC2.  //Set capture polarity to falling edge of CC2.
```

```
    PWMA_SMCR = 0x54; //TS=TI1FP1,SMS=TI1 rising edge reset mode.        //TS=TI1FP1,SMS=TI1 rising edge reset mode
    PWMA_CR1 = 0x01.

    PWMA_IER = 0x06.                                                    // Enable CC1/CC2 capture interrupt.
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
```

```
    unsigned int cnt.

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02.

        cnt = (PWMA_CCR1H << 8) + PWMA_CCR1L; //CC1 capture cycle width.          //CC1 capture cycle width
    }
    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04.

        cnt = (PWMA_CCR2H << 8) + PWMA_CCR2L; //CC2 capture duty cycle (high level width).   //CC2 capture duty cycle
        (high level width)
    }
}
```

23.8.12 Simultaneous capture of period and duty cycle of 4 input signals

Principle: Use the capture module CCx and CCx+1 of two channels inside the advanced PWM to capture the same external pin at the same time, CCx captures the rising edge of this pin, and CCx+1 captures the falling edge of this pin, the difference between the two capture values of CCx is the period, and the difference between the capture value of CCx+1 and the previous capture value of CCx is the duty cycle.

Example: The first capture module CC1 and the second capture module CC2 of PWMA are used to capture the PWM1P pin (P1.0) at the same time, where CC1 captures the rising edge of PWM1P and CC2 captures the falling edge of PWM1P, the capture value of CC1 minus the previous capture value is the period, and the capture value of CC2 minus the previous capture value of CC1 is the duty cycle. CC5 and CC6 of PWMB capture PWM5 at the same time (P2.0), CC7 and CC8 of PWMB capture PWM7 at the same time (P2.2), and CC3 and CC4 of PWMA capture PWM3P at the same time (P1.4). In addition, Timer 0 is used to generate waveforms on P1.0, Timer 1 on P1.4, Timer 3 on P2.0, and Timer 4 on P2.2. The captured values are sent to the PC through the serial port.

Note: 1. The two capture modules inside the chip are used to capture the same external pin at the same time, so there is no need to connect multiple external pins.

2. Since the reset trigger signal is not set, the period value and duty cycle value need to be subtracted accordingly.

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    //see download software for header files
#include "intrins.h"
#include "stdio.h"

#define    FOSC        12000000UL
#define    BRT         (65536 - (FOSC / 115200+2) / 4)
bit f1, f2, f3, f4;  void

main()
{
#define    T10K        (65536 - FOSC / 10000)
#define    T11K        (65536 - FOSC / 11000)
#define    T12K        (65536 - FOSC / 12000)
#define    T13K        (65536 - FOSC / 13000)

unsigned int ccr1;
unsigned int ccr3;
unsigned int ccr5;
unsigned int ccr7.

unsigned int cycle1;
unsigned int  duty1;
unsigned int cycle2;
unsigned int  duty2;
unsigned int cycle3;
unsigned int  duty3;
unsigned int cycle4.
unsigned int duty4.
```

```
                                                    The // plus 2 operation is to allow the Keil compiler to
                                                    //Automatic implementation of rounding operations

    EAXFR = 1;                                      //Enable access to XFR
    CKCON = 0x00;                                   //Set the external data bus speed to fastest
    WTST = 0x00;                                    //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.
```

```
p0m0 = 0x00;
p0m1 = 0x00;
p1m0 = 0x00;
p1m1 = 0x00;
p2m0 = 0x00;
p2m1 = 0x00;
p3m0 = 0x00;
p3m1 = 0x00;
p5m0 = 0x00;
p5m1 = 0x00.

AUXR |= 0x80.                                   //Timer 0 uses 1T mode.
AUXR |= 0x40.                                   //Timer 1 uses 1T mode.
TMOD = 0x00.                                    //Timer 0/1 using 16-bit auto-reload mode
TL0 = T10K.                                     //Timer 0 period 10K
TH0 = T10K >> 8.
TL1 = T11K.                                     //Timer 1 cycle 11K
TH1 = T11K >> 8;
TR0 = 1; //Timer 0 starts counting.             //Timer 0 starts counting
TR1 = 1; //Timer 1 starts counting.             //Timer 1 starts counting
ET0 = 1; //Enable timer 0 interrupt.            //Enable timer 0 interrupt
ET1 = 1; //Enable timer 1 interrupt.            //Enable timer 1 interrupt

T3L = T12K.                                     //Timer 3 cycle 12K
T3H = T12K >> 8.
T4L = T13K.                                     //Timer 4 cycle 13K
T4H = T13K >> 8;
T4T3M = 0xaa.                                   //Timer 3/4 using 1T mode
IE2 |= 0x20.                                    IE2 |= 0x20; //Enable timer 3 interrupt
IE2 |= 0x40.                                    IE2 |= 0x40; //Enable timer 4 interrupt

scon = 0x52; t2l
= brt.
T2H = BRT >> 8;
AUXR |= 0x15.

printf("PWM Test .

\n"); PWMA_CCER1 =

0x00;
PWMA_CCMR1 = 0x01.                              //CC1 is input mode and mapped to TI1FP1.
PWMA_CCMR2 = 0x02.                              //CC2 is input mode and mapped to TI1FP2.
PWMA_CCER1 = 0x11.                              //Enable capture function on CC1, enable capture
function on CC2.
PWMA_CCER1 |= 0x00; //Set capture polarity to rising edge of CC1.   //Set capture polarity to rising edge of CC1.
PWMA_CCER1 |= 0x20; //Set capture polarity to falling edge of CC2.  //Set capture polarity to falling edge of CC2.

PWMA_CCER2 = 0x00;
PWMA_CCMR3 = 0x01.                              //CC3 is input mode and mapped to TI3FP3.
PWMA_CCMR4 = 0x02.                              //CC4 is input mode and mapped to TI3FP4.
PWMA_CCER2 = 0x11.                              //Enable capture function on CC3, enable capture
function on CC4.
PWMA_CCER2 |= 0x00; //Set capture polarity to rising edge of CC3.   //Set capture polarity to rising edge of CC3.
PWMA_CCER2 |= 0x20; //Set capture polarity to falling edge of CC4.  //Set capture polarity to falling edge of CC4.
PWMA_CR1 = 0x01.

PWMA_IER = 0x1e.                                // Enable CC1/CC2/CC3/CC4 capture interrupt
```

```
PWMB_CCER1 = 0x00;
PWMB_CCMR1 = 0x01.                  //CC5 is input mode and mapped to TI5FP5.
PWMB_CCMR2 = 0x02.                  //CC6 is input mode and mapped to TI5FP6.
```

```
    PWMB_CCER1 = 0x11.                          //Enable capture function on CC5, enable capture function on CC6.
    PWMB_CCER1 |= 0x00; //Set capture polarity to rising edge of CC5.   //Set capture polarity to rising edge of CC5.
    PWMB_CCER1 |= 0x20; //Set capture polarity to falling edge of CC6.  //Set capture polarity to falling edge of CC6.

    PWMB_CCER2 = 0x00;
    PWMB_CCMR3 = 0x01.                          //CC7 is input mode and mapped to TI7FP8.
    PWMB_CCMR4 = 0x02.                          //CC8 is input mode and mapped to TI7FP8.
    PWMB_CCER2 = 0x11.                          //Enable capture function on CC7, enable capture function on CC8.
    PWMB_CCER2 |= 0x00; //Set capture polarity to rising edge of CC7.   //Set capture polarity to rising edge of CC7.
    PWMB_CCER2 |= 0x20; //Set capture polarity to falling edge of CC8.  //Set capture polarity to falling edge of CC8.
    PWMB_CR1 = 0x01.

    PWMB_IER = 0x1e.                            //Enable CC5/CC6/CC7/CC8 capture interrupt.

    EA = 1;

    while (1)
    {
        if (f1)
        {
            f1 = 0;
            printf("cycle1 = %04x duty1 = %04x\n", cycle1, duty1);
        }
        if (f2)
        {
            f2 = 0;
            printf("cycle2 = %04x duty2 = %04x\n", cycle2, duty2);
        }
        if (f3)
        {
            f3 = 0;
            printf("cycle3 = %04x duty3 = %04x\n", cycle3, duty3);
        }
        if (f4)
        {
            f4 = 0;
            printf("cycle4 = %04x duty4 = %04x\n", cycle4, duty4);
        }
    }
}

void TMR0_ISR() interrupt TMR0_VECTOR           // Generate CC1 waveform to P1.0 port.
{
    static unsigned int cnt = 0;

    cnt++;
    if (cnt == 10)
    {
        P10 = 0;
    }
    else if (cnt == 30)
    {
        P10 = 1;
        cnt = 0;
    }
}
```

```
void TMR1_ISR() interrupt TMR1_VECTOR                   // Generate CC3 waveform to P1.4 port.
{
```

```
    static unsigned int cnt = 0;

    cnt++;
    if (cnt == 10)
    {
        P14 = 0;
    }
    else if (cnt == 30)
    {
        P14 = 1;
        cnt = 0;
    }
}

void TMR3_ISR() interrupt TMR3_VECTOR                   // Generate CC5 waveform to P2.0 port.
{
    static unsigned int cnt = 0;

    cnt++;
    if (cnt == 10)
    {
        P20 = 0;
    }
    else if (cnt == 30)
    {
        P20 = 1;
        cnt = 0;
    }
}

void TMR4_ISR() interrupt TMR4_VECTOR                   // Generate CC7 waveform to P2.2 port.
{
    static unsigned int cnt = 0;

    cnt++;
    if (cnt == 10)
    {
        P22 = 0;
    }
    else if (cnt == 30)
    {
        P22 = 1;
        cnt = 0;
    }
}

void PWMA_ISR() interrupt PWMA_VECTOR
{
    unsigned int ccr.

    if (PWMA_SR1 & 0x02)                                //CC1 capture interrupt
    {
        PWMA_SR1 &= ~0x02.

        ccr = (PWMA_CCR1H << 8) + PWMA_CCR1L;           //read
        capture value cycle1 = ccr - ccr1;
                                                        //calculat
        e cycle
        ccr1 = ccr;                                     // save the current capture value
        f1 = 1; //The period and duty cycle of waveform 1 is captured.      //Waveform 1 period and duty cycle capture is
```

complete, trigger serial port to send
}

```
    if (PWMA_SR1 & 0x04)                        //CC2 capture interrupt
    {
        PWMA_SR1 &= ~0x04.

        ccr = (PWMA_CCR2H << 8) + PWMA_CCR2L;       //read
        capture value duty1 = ccr - ccr1;  //read capture value duty1 =
        ccr - ccr1; //read capture value duty1 = ccr - ccr1; //calculate duty
        cycle                                       //calculate
        duty cycle
    }

    if (PWMA_SR1 & 0x08)                        //CC3 capture interrupt
    {
        PWMA_SR1 &= ~0x08.

        ccr = (PWMA_CCR3H << 8) + PWMA_CCR3L;       //read
        capture value cycle2 = ccr - ccr3;
                                                    //calculat
        e cycle
        ccr3 = ccr;                                 //save the current capture value
        f2 = 1; //Waveform 2 period and duty cycle capture completed.     //Period and duty cycle of waveform 2 are captured,
        trigger the serial port to send.
    }
    if (PWMA_SR1 & 0x10)                        //CC4 capture interrupt
    {
        PWMA_SR1 &= ~0x10.

        ccr = (PWMA_CCR4H << 8) + PWMA_CCR4L;       //read the
        capture value duty2 = ccr - ccr3.           //calculate
        duty cycle
    }
}

void PWMB_ISR() interrupt PWMB_VECTOR
{
    unsigned int ccr.

    if (PWMB_SR1 & 0x02)                        //CC5 capture interrupt
    {
        PWMB_SR1 &= ~0x02.

        ccr = (PWMB_CCR5H << 8) + PWMB_CCR5L;       //read
        capture value cycle3 = ccr - ccr5;
                                                    //calculat
        e cycle
        ccr5 = ccr;                                 //save the current capture value
        f3 = 1; //Waveform 3 period and duty cycle capture completed.     //Waveform 3 period and duty cycle capture is
        complete, trigger the serial port to transmit
    }
    if (PWMB_SR1 & 0x04)                        //CC6 capture interrupt
    {
        PWMB_SR1 &= ~0x04.

        ccr = (PWMB_CCR6H << 8) + PWMB_CCR6L;       //read
        capture value duty3 = ccr - ccr5; //calculate duty cycle.
        //calculate duty cycle
    }

    if (PWMB_SR1 & 0x08)                        //CC7 capture interrupt
    {
        PWMB_SR1 &= ~0x08.
```

```
        ccr = (PWMB_CCR7H << 8) + PWMB_CCR7L;           //read capture value
        cycle4 = ccr - ccr7;                            //calculate cycle
        ccr7 = ccr;                                     //save the current capture value
        f4 = 1; //Waveform 4 period and duty cycle capture completed.   //waveform 4 period and duty cycle capture is complete, trigger serial port to send
    }
    if (PWMB_SR1 & 0x10)                                //CC8 capture interrupt
    {
        PWMB_SR1 &= ~0x10.
```

```
            ccr = (PWMB_CCR8H << 8) + PWMB_CCR8L;           //read the
            capture value duty4 = ccr - ccr7.                //calculate
            duty cycle
        }
}
```

23.8.13 Method of outputting PWM waveforms with 100% and 0% duty cycles (PWM1P)

(as an example)

23.8.13.1 Method 1: Disable PWM output by setting PWMx_ENO

```
//Tested operating frequency is 11.0592MHz
//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    p1m0 = 0x00;
    p1m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    PWMA_ENO &= ~0x01.                       // Disable PWM1P port output.
                                             // At this time, the PWM1P port is GPIO and can be
                                             //Directly operate the I/O port registers to force the
                                             output to go high or low.

    while (1);
}
```

23.8.13.2 Method 2: Set PWMx_CCMRn register to force output valid/invalid level

C code

```
//Tested operating frequency is 11.0592MHz
//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void main()
{
```

```
    EAXFR = 1;          //Enable access to XFR
    CKCON = 0x00;       //Set the external data bus speed to fastest
    WTST = 0x00;        //set the program code wait parameter.
                        //Assign a value of 0 to set the CPU to execute the
                        programme as fast as possible.

    p1m0 = 0x00;
    p1m1 = 0x00.
```

```
    p3m0 = 0x00;
    p3m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 &= ~0x03;                    //CC1 is the output mode
    PWMA_CCMR1 |= 0x40;                     //CC1 Forces an invalid output level (duty cycle 0%)
//  PWMA_CCMR1 |= 0x50;                     //CC1 Forced output active level (100% duty cycle)
    PWMA_CCER1 |= 0x01;                     //Enable CC1 output and set high to active level
    PWMA_ENO = 0x01;                        // Enable PWM1P port outputs
    PWMA_BKR = 0x80;                        //Enable main outputs
    PWMA_CR1 = 0x01.                        //Start the clock

    while (1);
}
```

23.8.14 Complementary PWM outputs with deadband control

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                         // see download software for header files
#include "intrins.h"

void main(void)
{
    EAXFR = 1;                              //Enable access to XFR
    CKCON = 0x00;                           //Set the external data bus speed to fastest
    WTST = 0x00;                            //set the program code wait parameter.
                                            //Assign a value of 0 to set the CPU to execute the
                                            programme as fast as possible.

    p0m1 = 0x00;
    p0m0 = 0xff;
    p1m1 = 0x00;
    p1m0 = 0xFF.

    PWMA_ENO = 0xFF.                        //IO output PWM
    PWMA_PS = 0x00;                         //00:PWM at P1

/*********************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
*********************************************************/
    PWMA_PSCRH = 0x00;                      //pre-scaler
    register PWMA_PSCRL = 0x00.
    PWMA_DTR = 0x10.                        // Dead time configuration

    PWMA_CCMR1 = 0x68;                      //channel
    mode configuration PWMA_CCMR2 = 0x68;
    PWMA_CCMR3 = 0x68;
    PWMA_CCMR4 = 0x68.

    PWMA_ARRH = 0x08.                       //auto-reload register, counter overflow point
    PWMA_ARRL = 0x00.
```

```
    PWMA_CCR1H = 0x04.                          //counter comparison value
```

```
    PWMA_CCR1L = 0x00;
    PWMA_CCR2H = 0x02;
    PWMA_CCR2L = 0x00;
    PWMA_CCR3H = 0x01;
    PWMA_CCR3L = 0x00;
    PWMA_CCR4H = 0x01;
    PWMA_CCR4L = 0x00.

    PWMA_CCER1 = 0x55.                    //configure channel output enable and polarity
    PWMA_CCER2 = 0x55.                    //configure channel output enable and polarity

    PWMA_BKR = 0x80.                      //Main output enable Equivalent to master switch
    PWMA_IER = 0x02.                      //Enable interrupt
    PWMA_CR1 = 0x01.                      //Enable counter

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        PWMA_SR1 &=~0X02.
    }
}
```

23.8.15 PWM port does external interrupt (falling edge interrupt or rising edge interrupt)

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                       // see download software for header files
#include "intrins.h"

void main(void)
{
    EAXFR = 1;                            //Enable access to XFR
    CKCON = 0x00;                         //Set the external data bus speed to fastest
   WTST = 0x00;                           //set the program code wait
                                          parameter.
                                          //Assign a value of 0 to set the CPU to execute the
                                          programme as fast as possible.

    p1m1 = 0x00;
    p1m0 = 0x00;
    p3m1 = 0x00;
    p3m0 = 0x00.

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01.                    //CC1 is input mode and mapped to TI1FP1.
    PWMA_CCER1 = 0x01.                    //Enable capture on CC1.
    PWMA_CCER1 |= 0x00; //Set capture polarity to rising edge of CC1.   //Set capture polarity to rising edge of CC1.
  //PWMA_CCER1 |= 0x02;                   //PWMA_CCER1 |= 0x02; //Set capture
```

```
polarity to falling edge of CC1.                    //Set capture polarity to falling
                                                    edge of CC1
    PWMA_CR1 = 0x01.
    PWMA_IER = 0x02;
    EA = 1;
```

```
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P37 = ~P37;
        PWMA_SR1 &=~0X02.
    }
}
```

23.8.16 Output arbitrary period and arbitrary duty cycle waveforms

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    PWMA_CCER1 = 0x00; //Write CCMRx.       // CCERx must be cleared before writing CCMRx to
    close the channel.
    PWMA_CCMR1 = 0x60; //Set CC1 to PWMA output mode.   //Set CC1 to PWMA output mode.
    PWMA_CCER1 = 0x01; //Enable CC1 channel.           //Enable CC1 channel
    PWMA_CCR1H = 0x01.                       //set duty cycle time
    PWMA_CCR1L = 0x00;
    PWMA_ARRH = 0x02;                        //Set cycle
    time PWMA_ARRL = 0x00;
    PWMA_ENO = 0x01.                         //Enable PWM1P port outputs
    PWMA_BKR = 0x80.                         //Enable main output
    PWMA_CR1 = 0x01.                         //start the timer

    while (1);
```

23.8.17 PWMA Timer with PWM's CEN to trigger ADC in real time

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    p1m0 = 0x00;
    p1m1 = 0x01;
    p3m0 = 0x00;
    p3m1 = 0x00.

    ADC_CONTR = 0;                           //select P1.0 as ADC input channel
    ADC_POWER = 1;
    ADC_EPWMT = 1;
    delay();                                 //Wait for ADC power supply to stabilise
    EADC = 1;

    PWMA_CR2 = 0x10;                         //CEN signal is TRGO, which can
    be used to trigger ADC PWMA_ARRH = 0x13; PWMA_CR2 = 0x10     ; //CEN signal is
    TRGO.
    pwma_arrl = 0x38; pwma_ier =
    0x01;
    PWMA_CR1 = 0x01; //Set CEN to start PWMA timer.     //Set CEN to start PWMA timer to trigger ADC in real
    time
    EA = 1;

    while (1);
}

void ADC_ISR() interrupt 5
{
    ADC_ FLAG = 0;
}

void PWMA_ISR() interrupt 26
```

```
{
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01.
    }
}
```

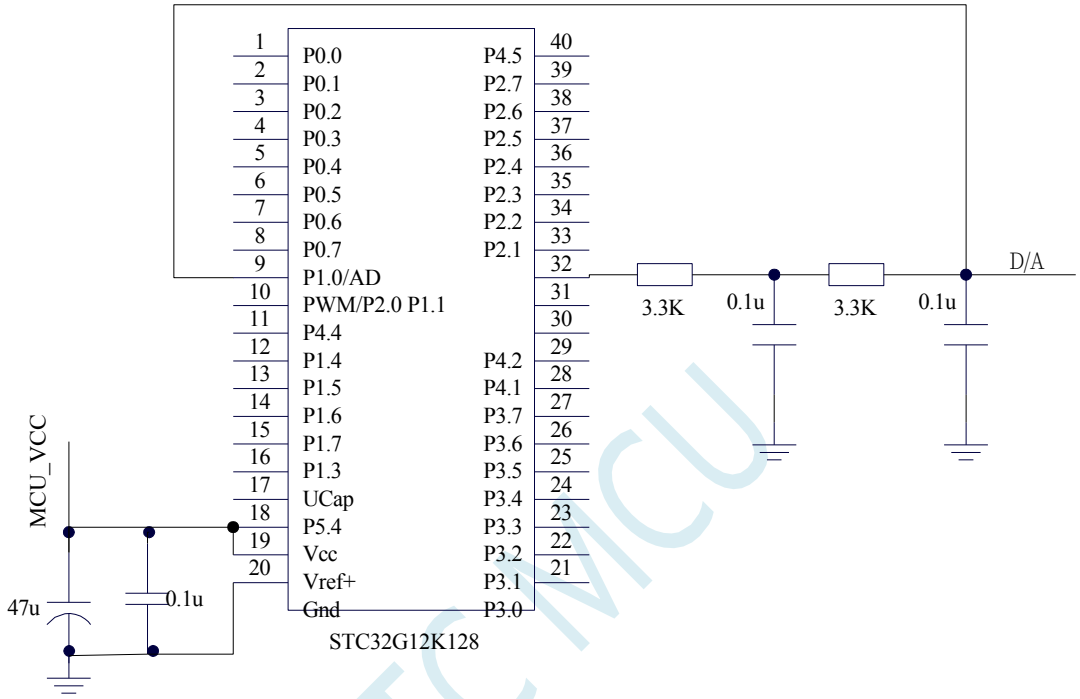
23.8.18 PWM Cycle Repeat Trigger ADC

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                          // see download software for header files
#include "intrins.h"

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    EAXFR = 1;                               //Enable access to XFR
    CKCON = 0x00;                            //Set the external data bus speed to fastest
    WTST = 0x00;                             //set the program code wait parameter.
                                             //Assign a value of 0 to set the CPU to execute the
                                             programme as fast as possible.

    p1m0 = 0x00;
    p1m1 = 0x01;
    p3m0 = 0x00;
    p3m1 = 0x00.

    ADC_CONTR = 0;                           //select P1.0 as ADC input channel
    ADC_POWER = 1;
    ADC_EPWMT = 1;
    delay();                                 //Wait for ADC power supply to stabilise
    EADC = 1;

    PWMA_CR2 = 0x20;                         //cycle update event for TRGO, used
    for cycle trigger ADC PWMA_ARRH = 0x13;
    pwma_arrl = 0x38; pwma_ier =
    0x01;
    PWMA_CR1 = 0x01; //Set CEN to start PWMA timer.   //Set CEN start PWMA timer.
    EA = 1;

    while (1);
}

void ADC_ISR() interrupt 5
{
    ADC_ FLAG = 0;
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01.
```

```
}
```

23.8.19 Reference Circuit Diagram for 16-bit DAC using PWM

The advanced PWM timer of STC32G series MCU can output a 16-bit PWM waveform, and then generate a 16-bit DAC signal after two stages of low-pass filtering, and the DAC signal can be changed by adjusting the duty cycle of the high level of the PWM waveform. The application circuit diagram is shown below, and the output DAC signal can be input to the ADC of MCU for feedback measurement.



23.8.20 Complementary SPWM with PWM

Advanced PWM Timers PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N per channel

The PWM1P and PWM1N can be used to generate complementary SPWM outputs. Demonstrate the use of PWM1P, PWM1N to generate complementary SPWM. 24MHZ is selected as the master clock, 1T is selected as the PWM clock, the PWM period is 2400, the deadband is 12 clocks (0.5us), and 200 points are used for the sinewave meter, and the output sinewave frequency is = 24000000 / 2400 / 200 = 50 HZ.

This program is only a demo program of SPWM, user can modify the points and amplitude of PWM period and sine wave by the above calculation method. The output frequency of this program is fixed, if you need frequency conversion, please design your own frequency conversion scheme.

```
//Tested operating
frequency is 11.0592MHz
```

```
//#include "stc8h.h"
#include "stc32g.h"                                    // See download software for header files
#include "intrins.h"

#define     MAIN_Fosc           24000000L              //Define master clock

            typedefunsign       u8;
ed char                         u16.
            typedefunsign
ed int
```

```
typedefunsigned long   u32;

/****************************** user-defined macros ***********************************/

#define   PWMA_1          0x00                //P:P1.0   N:P1.1
#define   PWMA_2          0x01                //P:P2.0   N:P2.1
#define   PWMA_3          0x02                //P:P6.0   N:P6.1

#define   PWMB_1          0x00                //P:P1.2   N:P1.3
#define   PWMB_2          0x04                //P:P2.2   N:P2.3
#define   PWMB_3          0x08                //P:P6.2   N:P6.3

#define   PWM3_1          0x00                //P:P1.4   N:P1.5
#define   PWM3_2          0x10                //P:P2.4   N:P2.5
#define   PWM3_3          0x20                //P:P6.4   N:P6.5

#define   PWM4_1          0x00                //P:P1.6   N:P1.7
#define   PWM4_2          0x40                //P:P2.6   N:P2.7
#define   PWM4_3          0x80                //P:P6.6   N:P6.7
#define   PWM4_4          0xC0                //P:P3.4   N:P3.3

#define   eno1p           0x01
#define   eno1n           0x02
#define   eno2p           0x04
#define   eno2n           0x08
#define   eno3p           0x10
#define   eno3n           0x20
#define   eno4p           0x40
#define   eno4n           0x80
#define
#define
#define

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

unsigned int code T_SinTable[]=
{
    1220, 1256, 1292, 1328, 1364, 1400, 1435, 1471.
    1506, 1541, 1575, 1610, 1643, 1677, 1710, 1742.
    1774, 1805, 1836, 1866, 1896, 1925, 1953, 1981.
    2007, 2033, 2058, 2083, 2106, 2129, 2150, 2171.
    2191, 2210, 2228, 2245, 2261, 2275, 2289, 2302.
    2314, 2324, 2334, 2342, 2350, 2356, 2361, 2365.
    2368, 2369, 2370, 2369, 2368, 2365, 2361, 2356.
    2350, 2342, 2334, 2324, 2314, 2302, 2289, 2275.
    2261, 2245, 2228, 2210, 2191, 2171, 2150, 2129.
    2106, 2083, 2058, 2033, 2007, 1981, 1953, 1925.
    1896, 1866, 1836, 1805, 1774, 1742, 1710, 1677.
    1643, 1610, 1575, 1541, 1506, 1471, 1435, 1400.
    1364, 1328, 1292, 1256, 1220, 1184, 1148, 1112.
    1076, 1040, 1005, 969, 934, 899, 865, 830.
      797.  763.  730.  698.  666,  635.  604,  574.
      544,  515.  487.  459,  433.  407,  382.  357.
      334,  311,  290,  269,  249.  230.  212.  195,
      179,  165,  151.  138,  126.  116.  106.   98.
       90.   84.   79.   75.   72.   71.   70.   71.
       72.   75.   79.   84.   90.   98.  106.  116.
      126.  138,  151.  165,  179,  195,  212.  230.
      249.  269,  290,  311.  334,  357.  382.  407,
```

435, 459, 487, 515, 544, 574, 604, 635,
666, 698, 730, 763, 797, 830, 865, 899,
934, 969, 1005, 1040, 1076, 1112, 1148, 1184.

```
};

u16 PWMA_Duty.
u8 PWM_Index.                                   //SPWM lookup table index

/********************* main function
**************************/ void main(void)
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign a value of 0 to set the CPU to execute the
                                                programme as fast as possible.

    p0m1 = 0;   p0m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p1m1 = 0;   p1m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p2m1 = 0;   p2m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p3m1 = 0;   p3m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p4m1 = 0;   p4m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p5m1 = 0;   p5m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p6m1 = 0;   p6m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p7m1 = 0;   p7m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port

    PWMA_Duty = 1220.

    PWMA_CCER1 = 0x00.                          //must clear CCxE before writing CCMRx to close
    channel
    PWMA_CCER2 = 0x00;
    PWMA_CCMR1 = 0x60; //Channel mode configuration.   //Channel mode configuration
    //PWMA_CCMR2 = 0x60;
    //PWMA_CCMR3 = 0x60.
    //PWMA_CCMR4 = 0x60.
    PWMA_CCER1 = 0x05; //Configure the channel output enable and polarity.   //configure channel output enable and
    polarity
    //PWMA_CCER2 = 0x55.

    PWMA_ARRH = 0x09.                           //Set the cycle time
    PWMA_ARRL = 0x60.

    PWMA_CCR1H = (u8)(PWMA_Duty >> 8);          //Set duty cycle
    time PWMA_CCR1L = (u8)(PWMA_Duty);

    PWMA_DTR = 0x0C.                            //set dead time

    PWMA_ENO = 0x00.
PWMA_ENO |= ENO1P.                              //Enable output
PWMA_ENO |= ENO1N.                              //Enable output
        //PWMA_ENO |= ENO2P.                    //Enable
                                                output
        //PWMA_ENO |= ENO2N.                    //Enable
                                                output
    //PWMA_ENO |= ENO3P.                        //PWMA_ENO
                                                |= ENO3P; //PWMA_ENO |= ENO3P
        //PWMA_ENO |= ENO3N.                    //Enable
                                                output
    //PWMA_ENO |= ENO4P.                        //PWMA_ENO
                                                |= ENO4P; //PWMA_ENO |= ENO4P
        //PWMA_ENO |= ENO4N.                    //Enable
```

output

```
PWMA_PS = 0x00;                          // Advanced PWM channel output pin select bit
 PWMA_PS |= PWMA_3; //Select PWMA_3 channel.   //Select PWMA_3
                                                channel
                          //PWMA_PS |= PWMB_3;//select PWMB_3 channel
                          //PWMA_PS |= PWM3_3;//select PWM3_3 channel
                          //PWMA_PS |= PWM4_3;//select PWM4_3 channel

PWMA_BKR = 0x80.                         //Enable main output
PWMA_IER = 0x01.                         //Enable interrupt
```

```
    PWMA_CR1 |= 0x01.                                   //start timing

    EA = 1;                                             //Turn on the general interrupt

    while (1)
    {
    }
}

/******************** interrupt function
*************************/ void PWMA_ISR() interrupt 26
{
    if (PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01.
        PWMA_Duty = T_SinTable[PWM_Index];
        if (++PWM_Index >= 200)
            PWM_Index = 0;

        PWMA_CCR1H = (u8)(PWMA_Duty >> 8);              //Set duty cycle
        time PWMA_CCR1L = (u8)(PWMA_Duty);
    }
    PWMA_SR1 = 0;
}
```

23.8.21 Advanced PWM Output - Adjustable Frequency - Pulse Counting (Software Method)

```
//Test operating frequency is 24MHz
/************* Function Description **************
Advanced PWM Timer for High-Speed PWM Pulse Output.
Adjustable period/duty cycle, pulse counting via compare/capture interrupt.
Demonstrate output through P6 port, output PWM every 10ms,
count 10 pulses and stop output. The timer adjusts the PWM
period every 1ms.
When downloading, select the clock 24MHZ (user can modify the frequency).
*****************************************/

//#include "stc8h.h"
#include "stc32g.h"                                     // See download
#include "intrins.h"                                    software for
                                                        header files

#define     MAIN_Fosc       24000000L

            typedefunsign   u8;
ed char                     u16;
            typedefunsign   u32.
ed int
            typedefunsign
ed long

/****************************** user-defined macros ***********************************/
```

```
#define    Timer0_Reload    (65536UL -(MAIN_Fosc / 1000))    //Timer0 interrupt frequency, 1000 times/sec.

/*****************************************************************************/


#define    PWM1_1           0x00                             //P:P1.0 N :P1.1
#define    PWM1_2           0x01                             //P:P2.0 N :P2.1
```

```
#define    PWM1_3          0x02                //P:P6.0   N:P6.1

#define    PWM2_1          0x00                //P:P1.2   N:P1.3
#define    PWM2_2          0x04                //P:P2.2   N:P2.3
#define    PWM2_3          0x08                //P:P6.2   N:P6.3

#define    PWM3_1          0x00                //P:P1.4   N:P1.5
#define    PWM3_2          0x10                //P:P2.4   N:P2.5
#define    PWM3_3          0x20                //P:P6.4   N:P6.5

#define    PWM4_1          0x00                //P:P1.6   N:P1.7
#define    PWM4_2          0x40                //P:P2.6   N:P2.7
#define    PWM4_3          0x80                //P:P6.6   N:P6.7
#define    PWM4_4          0xC0                //P:P3.4   N:P3.3

#define    eno1p           0x01
#define    eno1n           0x02
#define    eno2p           0x04
#define    eno2n           0x08
#define    eno3p           0x10
#define    eno3n           0x20
#define    eno4p           0x40
#define    eno4n           0x80
#define
#define
#define

/************ local variable declaration **************/
bit    B_1ms.                                  /1ms flag
bit    PWM1_Flag.

u16 Period;
u8 Counter;
u8 msSecond.

void UpdatePwm(void);
void TxPulse(void).

/******************** main function
*************************/ void main(void)
{
    eaxfr = 1; ckcon                           // Enable access to XFR
    = 0x00; wtst =                             //Set external data bus speed to fastest
    0x00.                                      //Set the parameter for waiting for the
                                               programme code.
                                               //Assign a value of 0 to set the CPU to
                                               execute the programme as fast as possible.

    P0M1 = 0x00.    P0M0 = 0x00.               //Set quasi-bidirectional port
    P1M1 = 0x00.    P1M0 = 0x00.               //Set quasi-bidirectional port
    P2M1 = 0x00.    P2M0 = 0x00.               //Set quasi-bidirectional port
    P3M1 = 0x00.    P3M0 = 0x00.               //Set quasi-bidirectional port
    P4M1 = 0x00.    P4M0 = 0x00.               //Set quasi-bidirectional port
    P5M1 = 0x00.    P5M0 = 0x00.               //Set quasi-bidirectional port
    P6M1 = 0x00.    P6M0 = 0x00.               //Set quasi-bidirectional port
    P7M1 = 0x00.    P7M0 = 0x00.               //Set quasi-bidirectional port

    PWM1_Flag = 0.
    Counter = 0;
    Period = 0x1000.
```

```
    //Timer0 initialisation
    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                    //Timer0 interrupt enable
    TR0 = 1; //Tiner0 run                                 //Tiner0 run

    PWMA_ENO = 0x00.
    PWMA_ENO |= ENO1P.                                   //Enable output

    PWMA_PS = 0x00;                                      //Advanced PWM channel output pin select bit
    PWMA_PS |= PWM1_3; //Select PWM1_3 channel.          //Select PWM1_3 channel

    UpdatePwm();
    PWMA_BKR = 0x80.                                     //Enable main output
    PWMA_CR1 |= 0x01.                                    //start timing

    P40 = 0; //Power up the LED.                         //Power LED
    EA = 1;                                              //Turn on the general interrupt

    while (1)
    {
        if(B_1ms)
        {
            B_1ms = 0;
            msSecond++;
            if(msSecond >= 10)
            {
                msSecond = 0;
                TxPulse().                               //10ms start PWM output once
            }
        }
    }
}

/************ send pulse function
**************/ void TxPulse(void)
{
    PWMA_CCER1 = 0x00.                                   //must clear CCxE before writing CCMRx to close
    channel
    PWMA_CCMR1 = 0x60; //Set PWM1 mode 1 output.        //Set PWM1 mode 1 output
    PWMA_CCER1 = 0x01.                                   //Enable CC1E channel, active high
    PWMA_SR1 = 0;                                        //clear flag bit
    PWMA_CNTR = 0; //Clear counter                       //clear counter
    PWMA_IER = 0x02.                                     //Enable Capture/Compare 1 interrupt.
}

                                    /**********************Timer0 1ms interrupt
function ************************/ void timer0(void) interrupt 1
{
    B_1ms = 1;
    if(PWM1_Flag)
    {
        Period++;                                        //Period increment
        if(Period >= 0x1000) PWM1_Flag = 0;
    }
    else
    {
        Period --;                                       //decreasing period
        if(Period <= 0x0100) PWM1_Flag = 1;
    }
    UpdatePwm();                                         //Set period, duty cycle
```

```
/****************** PWM interrupt function
*******************/ void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        PWMA_SR1 &=~0X02.                       //clear flag bit

        Counter++;
        if(Counter >= 10)                       //Count 10 pulses and then switch off the PWM counter.
        {
            Counter = 0;
            PWMA_CCER1 = 0x00.                  // must clear CCxE before writing CCMRx to close
            channel
            PWMA_CCMR1 = 0x40; //Set PWM1 to be forced to an invalid level.    //Set PWM1 to be forced to an
            invalid level.
            PWMA_CCER1 = 0x01.                  // Enable CC1E channel, active high
            PWMA_IER = 0x00.                    // turn off interrupt
        }
    }
}

//========================================================================
// Function: UpdatePwm(void)
// Description: Updates the PWM cycle duty cycle.
// Parameters: none.
// Returns: none.
// Version: V1.0, 2012-11-22
//========================================================================
void UpdatePwm(void)
{
    PWMA_ARR = Period.
    PWMA_CCR1 = (Period >> 1); //Set the duty cycle time: Period/2.    //set duty cycle time: Period/2
}
```

23.8.22 Advanced PWM Output - Adjustable Frequency - Pulse Counting (Hardware)

```
//Test operating frequency is 24MHz

/************* Function Description **************
This routine is written and tested based on the
STC32G as the main controller. Advanced PWM
Timer for High Speed PWM Pulse Output.
Adjustable period/duty cycle, pulse counting via compare/capture interrupt.
Demonstrate output through P6 port, output PWM every 10ms,
count 10 pulses and stop output. The number of pulses is
controlled in hardware by using single pulse mode with
repeat count register.
The timer adjusts the PWM period every 1ms.
When downloading, select the clock 24MHZ (user can modify the frequency).    Select the clock 24MHZ (user can modify the
frequency).
*****************************************/
```

```
#include "stc32g.h"
#include "intrins.h"

#define     MAIN_Fosc           24000000L           //Define master clock

typedef     unsigned char   u8;
typedef     unsigned int    u16;
typedef     unsigned long   u32.                    //1:printf use UART1; 2:printf use UART2
```

```
/****************************** user-defined macro
**********************************/                    #defineTimer0_Reload
(65536UL -(MAIN_Fosc / 1000))                          //Timer 0 Interrupt
Frequency , 1000 times/sec

#define   PWM1_0      0x00                             //P:P1.0    N:P1.1
#define   PWM1_1      0x01                             //P:P2.0    N:P2.1
#define   PWM1_2      0x02                             //P:P6.0    N:P6.1

#define   PWM2_0      0x00                             //P:P1.2/P5.4 N :P1.3
#define   PWM2_1      0x04                             //P:P2.2 N :P2.3
#define   PWM2_2      0x08                             //P:P6.2 N :P6.3

#define   PWM3_0      0x00                             //P:P1.4 N :P1.5
#define   PWM3_1      0x10                             //P:P2.4 N :P2.5
#define   PWM3_2      0x20                             //P:P6.4 N :P6.5

#define   PWM4_0      0x00                             //P:P1.6 N :P1.7
#define   PWM4_1      0x40                             //P:P2.6 N :P2.7
#define   PWM4_2      0x80                             //P:P6.6 N :P6.7
#define   PWM4_3      0xC0                             //P:P3.4 N :P3.3

#define   eno1p       0x01
#define   eno1n       0x02
#define   eno2p       0x04
#define   eno2n       0x08
#define   eno3p       0x10
#define   eno3n       0x20
#define   eno4p       0x40
#define   ENO4N       0x80
#define
#define
#define

/*******************************************************************************/

/************* local variable declaration **************/
bit B_1ms.                                             /1ms flag
bit PWM1_Flag.

u16 Period;
u8 msSecond.

void UpdatePwm(void);
void TxPulse(u8 rep).

/********************* main function
*****************************/ void main(void)
{
    WTST = 0;                                          //set the program instruction delay parameter.
                                                       //Assign 0 to set the CPU to execute instructions as fast as
                                                       possible.
    EAXFR = 1;                                         //Extended register (XFR) access enable
    CKCON = 0; //Increase the speed of accessing XRAM. //Increase the speed of accessing XRAM

    p0m1 = 0x00;    p0m0 = 0x00;                       //Set quasi-bidirectional port
    p1m1 = 0x00;    p1m0 = 0x00;                       //Set quasi-bidirectional port
```

```
    p2m1 = 0x00;    p2m0 = 0x00.                //Set quasi-bidirectional port
    p3m1 = 0x00;    p3m0 = 0x00;                //Set quasi-bidirectional port
    p4m1 = 0x00;    p4m0 = 0x00;                //Set quasi-bidirectional port
    p5m1 = 0x00;    p5m0 = 0x00.                //Set quasi-bidirectional port
    p6m1 = 0x00;    p6m0 = 0x00.                //Set quasi-bidirectional port
    p7m1 = 0x00;    p7m0 = 0x00.                //Set quasi-bidirectional port
```

```
    PWM1_Flag = 0;
    Period = 0x1000.

                                                    //Timer0 initialisation
    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              //Timer0 interrupt enable
    TR0 = 1; //Tiner0 run                          //Tiner0 run

    PWMA_ENO = 0x00.
    PWMA_ENO |= ENO1P.                              //Enable output

    PWMA_CCER1 = 0x00.                              // must clear CCxE before writing CCMRx to close
    channel
  PWMA_CCMR1 = 0x68; //Set PWM1 mode 1 output.     //Set PWM1 mode 1 output
      //PWMA_CCER1 = 0x01.                            //PWMA_CCER1 = 0x01.
    PWMA_CCER1 = 0x03.                              // Enable CC1E channel, active low

    PWMA_PS = 0x00;                                 // Advanced PWM channel output pin select bit
    PWMA_PS |= PWM1_2; //Select PWM1_2 channel.     //Select PWM1_2 channel

    UpdatePwm();
    PWMA_BKR = 0x80.                                //Enable main output
      //PWMA_CR1 |= 0x89; //Enable ARR preload, single pulse mode, start timing.      //enable ARR preload, single pulse
mode, start timing

    P40 = 0; //Power up the LED.                    //Power LED
    EA = 1;                                         //Turn on the general interrupt

    while (1)
    {
        if (B_1ms)
        {
            B_1ms = 0;
            msSecond++;
            if (msSecond >= 10)                     /10ms start PWM output once
            {
                msSecond = 0;
                TxPulse(10).                        //Output 10 pulses
            }
        }
    }
}

/************** send pulse function
***************/ void TxPulse(u8 rep)
{
    if (rep == 0) return;
    rep -= 1;

    PWMA_RCR = rep;                                 //Repeat count register, count rep pulses to generate
    update event
    PWMA_CR1 |= 0x89; //Enable ARR preload, single pulse mode, start timing.      //enable ARR preload, single pulse mode,
    start timing
}

/********************** timer0 1ms interrupt function
```

```
**********/ void timer0(void) interrupt 1
{
    B_1ms = 1;
    if (PWM1_Flag)
```

```
    {
        Period++;                                   //Period increment
        if (Period >= 0x1000) PWM1_Flag = 0;
    }
    else
    {
        Period --;                                  //decreasing period
        if (Period <= 0x0100) PWM1_Flag = 1;
    }
    UpdatePwm();                                    //Set period, duty cycle
}


//========================================================================
// Function: UpdatePwm(void)
// Description: Updates the PWM cycle duty cycle.
// Parameters: none.
// Returns: none.
// Version: V1.0, 2012-11-22
//========================================================================
void UpdatePwm(void)
{
    PWMA_ARRH = (u8)(Period >> 8);                  //Set the period time
    PWMA_ARRL = (u8)Period.
    PWMA_CCR1H = (u8)((Period >> 1) >> 8);          //Set duty cycle time:
    Period/2 PWMA_CCR1L = (u8)((Period >> 1));
}
```

23.8.23 Generate 3 complementary PWM waveforms with 120 degrees phase difference (courtesy of the Internet)

```
//Test operating frequency is 24MHz

/*************************************************
Main Functions: P2.0-P2.5 outputs complementary three-way PWM with 120 degrees phase difference.
The first P2.0/P2.1 is PWM output mode, the second P2.2/P2.3 and the third
P2.4/P2.5 are comparison output mode, the program is downloaded into the
target chip, and the output is 50hz SPWM with 25% duty cycle.
*************************************************/

#include "stc32g.h"

#define   FOSC              24000000UL

#define   PWM_PSC           (240-1)          //Define PWM clock prescaler factor
#define   PWM_PERIOD        2000             //Define PWM cycle value
                                             //(Frequency = FOSC/(PWM_PSC+1)/PWM_PERIOD=50Hz)
#define   PWM_DUTY          500              //Define the duty cycle value of PWM.
                                             //(Duty cycle = PWM_DUTY/PWM_PERIOD*100%=25%)

void SYS_Init();
void PWM_Init().

void main()
{
    SYS_Init();
    PWM_Init().

    EA = 1;                                  //Turn on the general interrupt

    while (1);
}

void SYS_Init()
{
    WTST = 0;                                //Set the program instruction delay parameter.
                                             //Assign 0 to set the CPU to execute instructions as fast as possible.
    EAXFR = 1;                               //Extended Register (XFR) Access Enable
    CKCON = 0;                               //Increase access speed to XRAM

    p0m1 = 0x00;   p0m0 = 0x00;
    p1m1 = 0x00;   p1m0 = 0x00;
    p2m1 = 0x00;   p2m0 = 0x00;
    p3m1 = 0x00;   p3m0 = 0x00;
    p4m1 = 0x00;   p4m0 = 0x00;
    p5m1 = 0x00;   p5m0 = 0x00;
    p6m1 = 0x00;   p6m0 = 0x00;
    p7m1 = 0x00.   p7m0 = 0x00.
}
```

```
void PWM_Init()
{
    PWMA_PSCRH = (char)(PWM_PSC >> 8);          //configure the prescaler coefficient
    PWMA_PSCRL = (char)(PWM_PSC).

    PWMA_CCER1 = 0x00; //Write CCMRx.          // CCxE must be cleared before writing CCMRx to close
    the channel.
```

```
    PWMA_CCER2 = 0x00;

    PWMA_CCMR1 = 0x60; //Channel Mode Configuration PWM Mode 1          //channel mode configuration PWM mode 1
    PWMA_CCMR2 = 0x30; //Channel mode configuration output compare mode.  //channel mode configuration output compare mode
    PWMA_CCMR3 = 0x30; //Channel mode configuration output compare mode.  //channel mode configuration output compare mode

    PWMA_CCER1 = 0x55.                                  //Configure channel 1,2,3 output enable and polarity.
    PWMA_CCER2 = 0x05.

    PWMA_ARRH = (char)(PWM_PERIOD >> 8);                //set cycle time
    PWMA_ARRL = (char)(PWM_PERIOD);

    PWMA_ENO = 0x3f.                                    //Enable PWM output
    PWMA_PS = 0x15;                                     //Advanced PWM channel output pin selection P2.0-P2.5

    PWMA_CCR1H = (char)(PWM_DUTY >> 8);                 //set duty cycle time
    PWMA_CCR1L = (char)(PWM_DUTY);
    PWMA_CCR2H = (char)(PWM_PERIOD/3 >> 8);             //Set OC2 start flip-flop bit
    PWMA_CCR2L = (char)(PWM_PERIOD/3);
    PWMA_CCR3H = (char)(PWM_PERIOD/3*2 >> 8);           //Set OC3 start flip-flop bit
    PWMA_CCR3L = (char)(PWM_PERIOD/3*2);

    PWMA_IER = 0x0d.                                    //Enable OC2/OC3 compare interrupt, update interrupt.

    PWMA_BKR = 0x80.                                    //Enable main output
    PWMA_CR1 |= 0x01.                                   //start timing
}

void PWMA_ISR()      interrupt 26
{
    if (PWMA_SR1 & 0x01)
    {
        PWMA_CCR2H = (char)(PWM_PERIOD/3 >> 8);         //set duty cycle time
        PWMA_CCR2L = (char)(PWM_PERIOD/3);
        PWMA_CCR3H = (char)(PWM_PERIOD/3*2 >> 8);       //set duty cycle time
        PWMA_CCR3L = (char)(PWM_PERIOD/3*2);
        PWMA_SR1 &= ~0x01.
    }
    else if (PWMA_SR1 &0x04)
    {
        PWMA_CCR2H = (char)((PWM_PERIOD/3+PWM_DUTY) >> 8);  //set 0C2 end flip-flop bit
        PWMA_CCR2L = (char)(PWM_PERIOD/3+PWM_DUTY);
        PWMA_SR1 &= ~0x04.
    }
    else if (PWMA_SR1 &0x08)
    {
          }


    }
    else
    {

    }
```

```
PWMA_C      WM_DUTY) >> 8);                                          //set 0C3 end flip-flop bit
CR3H =      PWMA_CCR3L = (char)(PWM_PERIOD/3*2+PWM_DUTY);
(char)((P   PWMA_SR1 &= ~0x08.
WM_PERI
OD/3*2+P
            PWMA_SR1 = 0;
```

24 High Speed Advanced PWM (HSPWM)

The STC32G series microcontrollers provide high-speed modes (HSPWMA and HSPWMB) for Advanced PWMA and Advanced PWMB. High-speed advanced PWM is based on advanced PWMA and advanced PWMB with the addition of high-speed mode.

When the system is running at a lower operating frequency, the high-speed advanced PWM can operate at up to 144M. This reduces core power consumption and improves peripheral performance.

24.1 Related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
HSPWMA_CFG	High-speed PWMA Configuration Register	7EFBF0H	-	-	-	-	-	INTEN	ASYNCEN	1	xxxx,0001
HSPWMA_ADR	High-speed PWMA address register	7EFBF1H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMA_DAT	High-speed PWMA data register	7EFBF2H	DATA[7:0]								0000,0000
HSPWMB_CFG	High-speed PWMB Configuration Register	7EFBF4H	-	-	-	-	-	INTEN	ASYNCEN	1	xxxx,0001
HSPWMB_ADR	High-speed PWMB address register	7EFBF5H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMB_DAT	High-speed PWMB data register	7EFBF6H	DATA[7:0]								0000,0000

24.1.1 HSPWM Configuration Register (HSPWMn_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_CFG	7EFBF0H	-	-	-	-	-	INTEN	ASYNCEN	1
HSPWMB_CFG	7EFBF4H	-	-	-	-	-	INTEN	ASYNCEN	1

ASYNCEN: Asynchronous
Control Mode Enable Bit 0:
Disables asynchronous
control.
1: Enable asynchronous control mode.
Note: When asynchronous control is turned off, advanced PWMA/advanced PWMB is in conventional mode, at this time, advanced PWM will automatically select the system operating frequency, and the PWM operating frequency is the same as the system operating frequency; if you need to work in high-speed mode, you need to enable the asynchronous control mode, and at this time, the PWM clock can be selected from the master clock (MCLK) or PLL output clock.

INTEN: Asynchronous mode
interrupt enable bit 0: Disable

PWM interrupt in asynchronous mode.

1: Enable PWM interrupt in asynchronous mode.

Note: In asynchronous mode, the INTEN bit must be enabled if you need to respond to an interrupt from the Advanced PWMA/Advanced PWMB.

24.1.2 HSPWM Address Register (HSPWMn_AD)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_ADR	7EFBF1H	RW/BUSY	ADDR[6:0]						
HSPWMB_ADR	7EFBF5H	RW/BUSY	ADDR[6:0]						

ADDR[6:0]: Advanced PWMA/PWMB Special Function Register Address Lower 7 Bits

0: Turn off asynchronous control.

1: Enable asynchronous control mode. RW/BUSY: read/write control bits, status bits

Write 0: **Write** PWMA/PWMB special function registers asynchronously. Write 1: Asynchronous **reading** of PWMA/PWMB special function registers. Read 0: Asynchronous read/write has been completed

Read 1: Asynchronous read/write in progress, busy state

24.1.3 HSPWM Data Register (HSPWMn_DAT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_DAT	7EFBF2H	DATA[7:0]							
HSPWMB_DAT	7EFBF6H	DATA[7:0]							

DATA[7:0]: special function register data of advanced PWMA/PWMB Write: Write data to special function register of advanced PWMA/PWMB.

Read: Read data from the special function registers of the advanced PWMA/PWMB.

Asynchronous Read PWMA Special Function Register Steps: (PWMB similar)

1. Read HSPWMA_ADR and wait for BUSY to be 0 to make sure the previous asynchronous read/write has been completed

2. Write the lower 7 bits of PWMA special function register to HSPWMA_ADR, and set "1" HSPWMA_ADR.7 at the same time.

3. Read HSPWMA_ADR and wait for BUSY to be 0.

4、Read HSPWMA_DAT

Asynchronous write PWMA special function register steps: (PWMB similar)

1. Read HSPWMA_ADR and wait for BUSY to be 0 to make sure the previous asynchronous read/write has been completed

2. Write the data to be written to the PWMA special function registers to HSPWMA_DAT.

3. Write the lower 7 bits of PWMA special function register to HSPWMA_ADR, and clear "0" HSPWMA_ADR.7.

4. Read HSPWMA_ADR and wait for BUSY to be 0. (You can skip this step and continue to execute other codes to improve system efficiency)

Special Note: The special function registers PWMA_PS and PWMB_PS belong to the port control registers and do not belong to the PWMA and PWMB register groups, so whether or not the asynchronous control mode of PWM is activated, the PWMA_PS and PWMB_PS registers can only be read and written using the normal synchronous mode.

24.2 sample procedure

24.2.1 Enable high-speed mode (asynchronous mode) for advanced PWMs

```
//Test operating frequency is 12MHz

//#include "stc8h.h"
#include "stc32g.h"                                     // see download software for header files
#include "intrins.h"

#define     FOSC        12000000UL

#define     HSCK_MCLK       0
#define     HSCK_PLL        1
#define     HSCK_SEL        HSCK_PLL

#define     PLL_96M         0
#define     PLL_144M        1
#define     PLL_SEL         PLL_144M

#define     CKMS            0x80
#define     HSIOCK          0x40
#define     MCK2SEL_MSK     0x0c
#define     MCK2SEL_SEL1    0x00
#define     MCK2SEL_PLL     0x04
#define     MCK2SEL_PLLD2   0x08
#define     MCK2SEL_IRC48   0x0c
#define     MCKSEL_MSK      0x03
#define     MCKSEL_HIRC     0x00
#define     MCKSEL_XOSC     0x01
#define     MCKSEL_X32K     0x02
#define     MCKSEL_IRC32K   0x03

#define     ENCKM           0x80
#define     PCKI_MSK        0x60
#define     PCKI_D1         0x00
#define     PCKI_D2         0x20
#define     PCKI_D4         0x40
#define     PCKI_D8         0x60

void delay()
{
    int     i;

    for ( i=0; i<100; i++);
}

char ReadPWMA(char addr)
{
    char dat.

    while (HSPWMA_ADR & 0x80).                          //Wait for the previous asynchronous read/write to
```

complete.

```
    HSPWMA_ADR = addr | 0x80.                   //Set the indirect access address, only need to set the
                                                lower 7 bits of the original XFR address.

                                                //HSPWMA_ADR register highest bit writes 1 to indicate
                                                read data
```

```
    while (HSPWMA_ADR & 0x80).                  //Wait for the current asynchronous read to complete
    dat =  HSPWMA_DAT.                          //read asynchronous data

    return dat.
}

void WritePWMA(char addr, char dat)
{
    while (HSPWMA_ADR & 0x80).                  //Wait for the previous asynchronous read/write to
    complete.
    HSPWMA_DAT = dat.                           //prepare the data to be written
    HSPWMA_ADR = addr & 0x7f.                   //Set the indirect access address, only need to set the
    lower 7 bits of the original XFR address.

                                                //Write 0 to the highest bit of the HSPWMA_ADR register
                                                to write data.
}

void main()
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00; //set external data bus speed to fastest
    WTST = 0x00.

    //Select PLL output clock
#if  (PLL_SEL == PLL_96M)
    CLKSEL &= ~CKMS.                            //Select 96M of PLL as PLL output clock.
#elif (PLL_SEL == PLL_144M)
    CLKSEL |= CKMS.                             //Select 144M of PLL as the output clock of PLL.
#else
    CLKSEL &= ~CKMS.                            //Default is to select 96M of PLL as the output clock of
    PLL.
#endif

    //Select PLL input clock divider to ensure
    input clock is 12M USBCLK &= ~PCKI_MSK.
#if  (FOSC == 12000000UL)
    USBCLK |= PCKI_D1.                          //PLL input clock 1 division
#elif (FOSC == 24000000UL)
    USBCLK |= PCKI_D2.                          //PLL input clock 2 divisions
#elif (FOSC == 48000000UL)
    USBCLK |= PCKI_D4.                          //PLL input clock 4 divisions
#elif (FOSC == 96000000UL)
    USBCLK |= PCKI_D8.                          //PLL input clock 8 divisions
#else
    USBCLK |= PCKI_D1.                          //Default PLL input clock 1 division
#endif

    //Start PLL
    USBCLK |= ENCKM.                            //Enable PLL multiplier

    delay();                                    //Wait for PLL to lock

    //Select HSPWM/HSSPI Clock
#if  (HSCK_SEL == HSCK_MCLK)
    CLKSEL &= ~HSIOCK.                          //HSPWM/HSSPI selects master clock as clock source
#elif (HSCK_SEL == HSCK_PLL)
    CLKSEL |= HSIOCK.                           //HSPWM/HSSPI Select PLL output clock as clock source
#else
    CLKSEL &= ~HSIOCK.                          // Default HSPWM/HSSPI Selects master clock as clock
```

source
#endif

HSCLKDIV = 0; //HSPWM/HSSPI clock sources are not divided. //HSPWM/HSSPI Clock source not divided by frequency

```
    HSPWMA_CFG = 0x03; //Enable PWMA related registers asynchronous access function. //Enable asynchronous access to
    PWMA related registers.

    //Set PWMA related registers asynchronously
    WritePWMA((char)&PWMA_CCER1, 0x00);
    WritePWMA((char)&PWMA_CCMR1, 0x00);                 //CC1 is output
    mode
    WritePWMA((char)&PWMA_CCMR1, 0x60);                 //OC1REF output PWM1 (active level 1 for CNT<CCR).
    WritePWMA((char)&PWMA_CCER1, 0x05);                 //Enable the output function on CC1/CC1N.
    WritePWMA((char)&PWMA_ENO, 0x03);                   //Enable the PWM signal output to the port.
    WritePWMA((char)&PWMA_BKR, 0x80);                   //Enable the main
    output WritePWMA((char)&PWMA_CCR1H, 200 >> 8);      //Set the duty cycle of
    output PWM
    WritePWMA((char)&PWMA_CCR1L, 200);
    WritePWMA((char)&PWMA_ARRH, 1000 >> 8);             //Set the period of the output PWM
    WritePWMA((char)&PWMA_ARRL, 1000);
    WritePWMA((char)&PWMA_DTR, 10);                     //Set the deadband of complementary symmetrical
    output PWM.
    WritePWMA((char)&PWMA_CR1, 0x01);                   //Start PWM counting

    P2M0 = 0.
    P2M1 = 0.
    P3M0 = 0.
    P3M1 = 0.

    P2 = ReadPWMA((char)&PWMA_ARRH);                    // Read registers
    asynchronously P3 = ReadPWMA((char)&PWMA_ARRL);

    while (1);
}
```

25 USB Universal Serial Bus

STC32G series microcontrollers have integrated USB2.0/USB1.1 compliant full-speed USB, 6 bi-directional endpoints supporting 4 endpoint transfer modes (control transfer, interrupt transfer, bulk transfer and synchronous transfer) with 64 bytes buffer for each endpoint.

The USB module has a total of 1280 bytes of FIFO with the following structure:

Control	Endp 0-IN	64 Bytes	64 Bytes	Endp 0-OUT	Control
INT BULK ISO	Endp 1-IN	64 Bytes	64 Bytes	Endp 1-OUT	INT BULK ISO
INT BULK ISO	Endp 2-IN	64 Bytes	64 Bytes	Endp 2-OUT	INT BULK ISO
INT BULK ISO	Endp 3-IN	64 Bytes	64 Bytes	Endp 3-OUT	INT BULK ISO
INT BULK ISO	Endp 4-IN	128 Bytes	256 Bytes	Endp 4-OUT	INT BULK ISO
INT BULK ISO	Endp 5-IN	128 Bytes	256 Bytes	Endp 5-OUT	INT BULK ISO

25.1 USB related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
USBCLK	USB Clock Control Register	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]		0010,0000
USBCON	USB Control Register	F4H	ENUSB	USBRST	PS2M	PUEN	PDEN	DFREC	DP	DM	0000,0000
USBADR	USB Address Register	ECH	BUSY	AUTORD	UADR[5:0]						0000,0000
USBDAT	USB Data Register	FCH									0000,0000

25.1.1 USB Control Register (USBCON)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USBCON	F4H	ENUSB	USBRST	PS2M	PUEN	PDEN	DFREC	DP	DM

ENUSB: USB Function and USB Clock Control Bits

0: Disable USB function and USB clock

1: Enabling USB Functions and the USB Clock

ENRST: USB reset setting control bit

0: Disable USB reset setting

1: Enable USB reset

PS2M: PS2 mode function control bit 0: disable PS2 mode function 1: enable PS2 mode function

PUEN: 1.5K pull-up resistor control bit on DP/DM port

0: Pull-up resistor disabled

1: Enable pull-up resistor

PDEN: 500K pull-down resistor control bit on DP/DM port

0: Pull-down resistor disabled

1: Enable pull-down resistor

DFREC: Differential receive status bit (read-only)

0: Current differential state of DP/DM is "0".

1: The current differential state of DP/DM is "1".

DP: D+ port status (read-only when PS2 is 0, read/write when PS2 is 1)

0: Current D+ is at logic 0 level

1: Currently D+ is logic 1 level

DM: D-Port status (read-only when PS2 is 0, read/write when PS2 is 1)

0: Current D-is logic 0 level

1: Currently, D is logic 1 level.

25.1.2 USB Clock Control Register (USBCLK)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USBCLK	DCH	ENCKM	PCKI[1:0]		CRE	TST_USB	TST_PHY	PHYTST[1:0]	

ENCKM: PLL multiplier control

0: Disable PLL frequency doubling

1: Enable PLL multiplier

PCKI[1:0]: PLL clock selection

PCKI[1:0]	PLL Clock Source
00	/1
01	/2
10	/4
11	/8

CRE: Clock chasing control bit 0: Clock chasing disabled

1: Enable clock chasing TST_USB: USB test mode

0: USB test mode disabled

1: Enable USB test mode

TST_PHY: PHY test mode

0: Disable PHY test mode

1: Enable PHY test mode

PHYTST[1:0]: USB PHY test

PHYTST[1:0]	way (of life)	DP	DM

00	Mode 0: Normal	x	x
01	Mode 1: Mandatory "1"	1	0
10	Mode 2: Forced "0"	0	1
11	Mode 3: Forcing single-ended "0's"	0	0

25.1.3 USB Interaddress Address Register (USBADR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USBADR	FCH	BUSY	AUTORD	UADR[5:0]					

BUSY: USB register read busy flag bit write 0: meaningless
Write 1: Initiates read operation of USB indirect register, address set by USBADR Read 0: Data in USBDAT register is valid
Read 1: Data in USBDAT register is invalid, USB is reading indirect registers

AUTORD: USB register auto-read flag, used for block read/write of USB FIFO 0: BUSY flag bit must be written every time the indirect USB register is read.
Write 1: When software reads USBDAT, the next USB indirect register read is automatically initiated (USBADR unchanged)

UADR[5:0]: Address of USB indirect registers

25.1.4 USB Interaddress Data Register (USBDAT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
USBDAT	ECH	UDAT[7:0]							

UDAT[7:0]: for indirect reading and writing of USB registers

25.2 USB Controller Register (SIE)

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
30H	UTRKCTL	UTRKSTS						
28H								
20H	FIFO0	FIFO1	FIFO2	FIFO3	FIFO4	FIFO5		
18H								
10H	INMAXP	CSR0 INCSR1	INCSR2	OUTMAXP	OUTCSR1	OUTCSR2	COUNT0 OUTCOUNT1	OUTCOUNT2
08H		INTROUT1E		INTRUSBE	FRAME1	FRAME2	INDEX	
00H	FADDR	POWER	INTRIN1	-	INTROUT1	-	INTRUSB	INTRIN1E

notation	descriptions	addr ess	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
FADDR	USB Function Address	00H	UPDATE	UADDR[6:0]							0000,0000

	Register										
POWER	USB Power Management Register	01H	ISOUD	-	-	-	USBRST	USBRSU	USBSUS	ENSUS	0xxx,0000
INTRIN1	USB endpoint IN interrupt flag bit	02H	-	-	EP5INIF	EP4INIF	EP3INIF	EP2INIF	EP1INIF	EP0IF	xx00,0000
INTROUT1	USB Endpoint OUT Interrupt Flag Bit	04H	-	-	EP5OUTIF	EP4OUTIF	EP3OUTIF	EP2OUTIF	EP1OUTIF	-	xx00,000x

INTRUSB	USB Power Interrupt Flag Bit	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF	xxxx,0000
INTRIN1E	USB Endpoint IN Interrupt Allow Bit	07H	-	-	EP5INIE	EP4INIE	EP3INIE	EP2INIE	EP1INIE	EP0IE	xx11,1111
INTROUT1E	USB Endpoint OUT Interrupt Allow Bit	09H	-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-	xx11,111x
INTRUSBE	USB Power Interrupt Allow Bit	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE	xxxx,0110
FRAME1	USB Data Frame Number Low Byte	0CH	FRAME[7:0]								0000,0000
FRAME2	USB Data Frame Number High Byte	0DH	-	-	-	-	-	FRAME[10:8]			xxxx,x000
INDEX	USB Endpoint Index Register	0EH	-	-	-	-	-	INDEX[2:0]			xxxx,x000
INMAXP	Maximum packet size for IN endpoints	10H	INMAXP[7:0]								0000,0000
CSR0	Endpoint 0 Control Status Register	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY	0000,0000
INCSR1	IN Endpoint control status register 1	11H	-	CLRDT	STSTL	SDSTL	FLUSH	UNDRUN	FIFONE	IPRDY	x000,0000
INCSR2	IN Endpoint control status register 2	12H	AUTOSET	ISO	MODE	ENDMA	FCDT	-	-	-	0010,0xxx
OUTMAXP	OUT Maximum packet size for the endpoint	13H	OUTMAXP [7:0]								0000,0000
OUTCSR1	OUT Endpoint control status register 1	14H	CLRDT	STSTL	SDSTL	FLUSH	DATERR	OVRRUN	FIFOFUL	OPRDY	0000,0000
OUTCSR2	OUT Endpoint control status register 2	15H	AUTOCLR	ISO	ENDMA	DMAMD	-	-	-	-	0000,xxxx
COUNT0	OUT length of endpoint 0	16H	-	OUTCNT0[6:0]							x000,0000
OUTCOUNT1	USB Endpoint OUT Length Low Byte	16H	OUTCNT[7:0]								0000,0000
OUTCOUNT2	USB Endpoint OUT Length High Byte	17H	-	-	-	-	-	OUTCNT[10:8]			xxxx,x000
FIFO0	FIFO Access Register for Endpoint 0	20H	FIFO0[7:0]								0000,0000
FIFO1	FIFO Access Register for Endpoint 1	21H	FIFO1[7:0]								0000,0000
FIFO2	FIFO Access Register for Endpoint 2	22H	FIFO2[7:0]								0000,0000
FIFO3	FIFO Access Register for Endpoint 3	23H	FIFO3[7:0]								0000,0000
FIFO4	FIFO Access Register for Endpoint 4	24H	FIFO4[7:0]								0000,0000
FIFO5	FIFO Access Register for Endpoint 5	25H	FIFO5[7:0]								0000,0000
UTRKCTL	USB Tracking Control Register	30H	FTM1	FTM0	INTV[1:0]		ENST5	RES[2:0]			1011, 1011
UTRKSTS	USB Tracking Status Register	31H	INTVCNT[3:0]				STS[1:0]		TST_UTRK	UTRK_RDY	1111,00x0

25.2.1 USB Function Address Register (FADDR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
FADDR	00H	UPDATE	UADDR[6:0]						

UPDATE: Update USB Function Address

0: Last UADDR address in effect

1: The last UADDR address is not yet active

UADDR[6:0]: store the 7-bit functional address of USB

25.2.2 USB Power Control Register (POWER)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
POWER	01H	ISOUD	-	-	-	USBRST	USBRSU	USBSUS	ENSUS

ISOUD (ISO Update): ISO Update

0: When software writes "1" to IPRDY, USB sends packets after receiving the next IN token.

1: When the software writes "1" to IPRDY, the USB sends the packet after receiving the SOF token, if it receives IN before the SOF token, the USB sends the packet after receiving the SOF token, if it receives IN before the SOF token, the USB sends the packet after receiving the SOF token.

token, then the USB sends a packet of length 0

USBRST (USB Reset): USB Reset Control Bit

Writing a "1" to this bit forces an asynchronous USB reset. Read this bit to get the current reset status information on the bus.

0: No reset signal detected on the bus

1: Reset signal detected on the bus

USBRSU (USB Resume): USB Resume Control bit

Forces a recovery signal on the bus in software to remotely wake up the USB device from suspend mode. When the USB is in suspend mode (USBSUS=1), writing a "1" to this bit will force a resume signal on the USB bus, and software should write a "0" to this bit after 10-15ms to end the resume signal. Software should write "0" to this bit after 10-15ms to end the recovery signal. After software writes "0" to USBRSU, it will generate a USB recovery interrupt, and then the hardware will automatically clear "0" to USBSUS.

USBSUS (USB Suspend): USB Suspend Control Bit

This bit is set to "1" by hardware when the USB is in suspend mode. This bit is automatically cleared to "0" by hardware when a recovery signal is forced on the bus by software or when a recovery signal is detected on the bus and the INTRUSB register is read.

ENSUS (Enable Suspend Detection): Enable USB suspend detection.

0: disable hang detection, USB will ignore the hang signal on the bus

1: Enable suspend detection, when suspend signal is detected on the bus, USB will enter suspend mode.

25.2.3 USB endpoint IN interrupt flag bit (INTRIN1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INTRIN1	02H	-	-	EP5INIF	EP4INIF	EP3INIF	EP2INIF	EP1INIF	EP0IF

EP5INIF: IN interrupt flag bit for endpoint 5

0: IN interrupt at endpoint 5 is invalid

1: IN interrupt valid at endpoint 5 EP4INIF: IN interrupt flag bit at endpoint 4 0: IN interrupt invalid at endpoint 4

1: IN interrupt valid at endpoint 4 EP3INIF: IN interrupt flag bit at endpoint 3 0: IN interrupt invalid at endpoint 3

1: IN interrupt valid at endpoint 3 EP2INIF: IN interrupt flag bit at endpoint 2 0: IN interrupt invalid at endpoint 2

1: IN interrupt valid for endpoint 2 EP1INIF: IN interrupt flag bit for endpoint 1

0: IN interrupt invalid for endpoint 1

1: IN interrupt valid at endpoint 1

EP0IF: IN/OUT interrupt flag bit for endpoint 0

0: Invalid IN/OUT interrupt for endpoint 0

1: IN/OUT interrupt valid for endpoint 0

After the software reads the INTRIN1 register, the hardware will automatically clear all interrupt flags in INTRIN1.

25.2.4 USB endpoint OUT interrupt flag bit (INTROUT1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INTROUT1	04H	-	-	EP5OUTIF	EP4OUTIF	EP3OUTIF	EP2OUTIF	EP1OUTIF	-

EP5OUTIF: OUT interrupt flag bit for endpoint 5

0: OUT interrupt for endpoint 5 invalidated

1: OUT interrupt valid at endpoint 5

EP4OUTIF: OUT interrupt flag bit for endpoint 4
0: OUT interrupt for endpoint 4 invalidated
1: OUT interrupt valid at endpoint 4
EP3OUTIF: OUT interrupt flag bit for endpoint 3
0: OUT interrupt for endpoint 3 invalidated
1: OUT interrupt valid at endpoint 3
EP2OUTIF: OUT interrupt flag bit for endpoint 2
0: OUT interrupt invalid for endpoint 2
1: OUT interrupt valid for endpoint 2
EP1OUTIF: OUT interrupt flag bit for endpoint 1
0: OUT interrupt invalid for endpoint 1
1: OUT interrupt valid for endpoint 1

After software reads the INTROUT1 register, hardware will automatically clear all interrupt flags in INTROUT1.

25.2.5 USB power interrupt flag (INTRUSB)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INTRUSB	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF

SOFIF: USB Start of Frame Interrupt Flag
0: USB frame start signal interrupt invalid
1: USB frame start signal interrupt active
RSTIF: USB reset signal interrupt flag
0: USB reset signal interrupt invalid
1: USB reset signal interrupt valid RSUIF: USB recovery signal interrupt flag
0: USB recovery signal interrupt invalid
1: USB recovery signal interrupt valid SUSIF: USB pending signal interrupt flag 0: USB pending signal interrupt invalid
1: USB hang signal interrupt active

The hardware will automatically clear all interrupt flags in the INTRUSB after the software reads the INTRUSB register.

25.2.6 USB Endpoint IN Interrupt Allow Register (INTRIN1E)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INTRIN1E	07H	-	-	EP5INIE	EP4INIE	EP3INIE	EP2INIE	EP1INIE	EP0IE

EP5INIE: IN interrupt control bit for endpoint 5
0: Disable IN interrupt at endpoint 5

1: Allow IN interrupt for endpoint 5 EP4INIE: IN interrupt control bit for endpoint 4 0: Disable IN interrupt for endpoint 4 1: Allow IN interrupt at endpoint 4 EP3INIE: IN interrupt control bit for endpoint 3 0: Disable IN interrupt at endpoint 3

1: Allow IN interrupt at endpoint 3

EP2INIE: IN interrupt control bit for endpoint 2
0: Disable IN interrupt at endpoint 2
1: Allow IN interrupt for endpoint 2 EP1INIE: IN interrupt control bit for endpoint 1 0: Disable IN interrupt for endpoint 1
1: Allow IN interrupt at endpoint 1
EP0IE: IN/OUT interrupt control bit for endpoint 0
0: Disable IN/OUT interrupt for endpoint 0
1: Allow IN/OUT interrupt for endpoint 0

25.2.7 USB Endpoint OUT Interrupt Allow Register (INTROUT1E)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INTROUT1E	09H	-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-

EP5OUTIE: OUT interrupt control bit for endpoint 5
0: disable OUT interrupt at endpoint 5
1: OUT interrupt allowed for endpoint 5
EP4OUTIE: OUT interrupt control bit for endpoint 4
0: disable OUT interrupt at endpoint 4
1: OUT interrupt allowed for endpoint 4
EP3OUTIE: OUT interrupt control bit for endpoint 3
0: Disable OUT interrupt at endpoint 3
1: OUT interrupt allowed for endpoint 3
EP2OUTIE: OUT interrupt control bit for endpoint 2
0: Disable OUT interrupt at endpoint 2
1: OUT interrupt allowed for endpoint 2
EP1OUTIE: OUT interrupt control bit for endpoint 1
0: Disable OUT interrupt at endpoint 1
1: OUT interrupt allowed for endpoint 1

25.2.8 USB Power Interrupt Allow Register (INTRUSB)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INTRUSB	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE

SOFIE: USB Start of Frame Interrupt Control Bit
0: Disable USB frame start signal interrupt
1: Allow USB frame start signal interrupt
RSTIE: USB reset signal interrupt control bit
0: Disable USB reset signal interrupt
1: Allow USB reset signal interrupt RSUIE: USB recovery signal interrupt control bit 0:

Disable USB recovery signal interrupt

1: Allow USB recovery signal interruption

SUSIE: USB Suspend Signal Interrupt Control Bit

0: Disable USB hang signal interrupt

1: Allow USB hang signal interrupt

25.2.9 USB Data Frame Number Register (FRAMEn)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
FRAME1	0CH	FRAME[7:0]							
FRAME2	0DH	-	-	-	-	-	FRAME[10:8]		

FRAME[10:0]: used to save the 11-bit frame number of the last received data frame

25.2.10 USB Endpoint Index Register (INDEX)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INDEX	0EH	-	-	-	-	-	INDEX[2:0]		

INDEX[2:0]: Select USB endpoints

INDEX[2:0]	target endpoint
000	Endpoint 0
001	Endpoint 1
010	Endpoint 2
011	Endpoint 3
100	Endpoint 4
101	Endpoint 5

25.2.11 Maximum packet size for IN endpoints (INMAXP)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INMAXP	10H	INMAXP[7:0]							

INMAXP[7:0]: Sets the maximum packet size of the IN endpoint of the USB (**Note: Packets are in 8 bytes. For example, if you need to set the packet size of the endpoint to 64 bytes, you need to set this register to 8**)

When you need to get/set this information, you must first use INDEX to select the target endpoint 0~5

25.2.12 USB Endpoint 0 Control Status Register (CSR0)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CSR0	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY

SSUEND (Serviced Setup End)

SETUP end event processing completion flag. When processing of the SETUP end event (SUEND) is complete, the software needs to set the SSUEND

Flag bit, hardware detects SSUEND is written to "1" and automatically clears the SUEND bit to "0".

SOPRDY (Serviced OPRDY)

OPRDY event processing completion flag. After the packet received from endpoint 0 is processed, the software needs to set the SOPRDY flag bit, and the hardware automatically clears the OPRDY bit to "0" when it detects that SOPRDY is written to "1".

SDSTL (Send Stall)

When an error condition or unsupported request is received, a "1" can be written to this bit

to end the current data transfer. When STALL
The hardware automatically clears this bit to "0" after the signal is sent.

SUEND (Setup End)

SETUP End-of-package flag. When a control transfer ends before software writes a "1" to the DATAEND bit, the hardware flags the end of the package only.

Read position "1". When software writes a '1' to SSUEND, hardware clears this bit to '0'.

DATEND (Data End)

End of data. The software should write "1" to this bit in the following cases:

1. When the firmware writes "1" to IPRDY after sending the last packet;
2. When the firmware writes "1" to IPRDY after sending a zero-length packet;
3、When the firmware writes "1" to SOPRDY after receiving the last data packet; this bit will be cleared "0" by hardware automatically.

STSTL (Sent Stall)

STALL signal transmission completion flag. This bit is set to "1" by hardware after the STALL signal has been sent. This bit must be cleared to "0" by software.

IPRDY (IN Packet Ready)

IN Packet Ready Flag. Software shall "1"this bit after loading a packet to be sent into the FIFO at endpoint 0.Hardware clears this bit to "0" when one of the following conditions occurs:

1. when the packet has been sent;
2. when the packet is overwritten by a SETUP packet;
3. when the packet is overwritten by an OUT packet;

OPRDY (OUT Packet Ready)

OUT packet ready flag. When an OUT packet is received, hardware sets this read-only bit to "1" and generates an interrupt. This bit is cleared to "0" only when software writes a "1" to the SOPRDY bit.

When you need to get/set this information, you must first use INDEX to select the target endpoint 0

25.2.13 IN Endpoint Control Status Register 1 (INCSR1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INCSR1	11H	-	CLRDT	STSTL	SDSTL	FLUSH	UNDRUN	FIFONE	IPRDY

CLRDT (Clear Data Toggle): resets the IN data toggle bit.

When the IN endpoint needs to reset the data toggle bit to "0" due to reconfiguration or STALL, the software needs to write "1" to this data bit.

STSTL (Sent Stall): STALL signal sending completion flag.

When the STALL signal has been sent, the hardware sets this bit to "1" (when the FIFO is cleared and the IPRDY bit is cleared to "0"). This flag must be cleared to "0" by software.

SDSTL (Send Stall): the STALL signal send request bit.

Software shall write a "1" to this bit to generate a STALL signal in response to an IN token.

Software shall write a "0" to this bit to terminate the STALL signal. This bit has no effect on the ISO method.

FLUSH (FIFO Flush): clears the next packet from the FIFO at the IN endpoint.

Writing a "1" to this bit will clear the next packet to be sent from the IN endpoint FIFO. the FIFO pointer is reset and the IPRDY bit is cleared to "0". If the FIFO contains more than one packet, software must write a "1" to FLUSH for each packet. When the FIFO is empty, hardware clears the FLUSH bit to "0".

UNDRUN (Data Underrun): insufficient data. The function of this bit depends on the IN endpoint method:

ISO mode: This bit is set to "1" when IPRDY is "0" and a zero-length packet is sent after an IN token is received. Interrupt/Bulk mode: This bit is set to "1" when NAK is used as an answer to an IN token.

This bit must be cleared to "0" by software.

FIFONE (FIFO Not Empty): FIFO Not Empty flag for IN endpoints

0: FIFO at IN endpoint is empty

1: FIFO at IN endpoint contains one or more packets

IPRDY (IN Packet Ready): IN packet ready completion flag.

Software shall "1" this bit after loading a packet to be sent into the endpoint's FIFO.

Hardware shall clear this bit to "0" upon the occurrence of one of the following conditions:

1. when the packet has been sent;

2. Auto setup is enabled (AUTOSET = "1") and the FIFO packets of endpoint IN reach the value set by INMAXP;

3、If the endpoint is in synchronous mode and ISOUD is "1", the readout value of IPRDY will always be 0 until the next SOF is received, when you need to get/set this information, you must first use INDEX to select the target endpoint 1~5.

25.2.14 IN Endpoint Control Status Register 2 (INCSR2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INCSR2	12H	AUTOSET	ISO	MODE	ENDMA	FCDT	-	-	-

AUTOSET: Automatically sets the IPRDY flag control bit.

0: Disable automatic setting of the IPRDY flag

1: Enable automatic IPRDY setting (data must be loaded into the IN FIFO up to the value set by INMAXP, otherwise IPRDY)

(Flag must be set manually)

ISO: Synchronous transmission enable.

0: Endpoint is configured for batch/interrupt transmission mode

1: The endpoint is configured for synchronous transmission mode MODE: Endpoint direction selection bit.

0: Select the endpoint direction as OUT

1: Select endpoint direction as IN ENDMA: DMA control of IN endpoints

0: Disable DMA requests for IN endpoints

1: Enable DMA requests for IN endpoints

FCDT: Forces the DATA0/DATA1 data toggle setting.

0: Endpoint data is switched only after a packet is sent and an ACK is received.

1: Endpoint data is forced to be switched after each packet is sent, regardless of whether an ACK is received or not. to get/set this information, you must first use INDEX to select the target endpoint 1~5.

25.2.15 OUT Maximum packet size at endpoint (OUTMAXP)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
OUTMAXP	13H	OUTMAXP [7:0]							

OUTMAXP[7:0]: Set the maximum packet size of the OUT endpoint of the USB (**Note: the packets are in 8 bytes unit. For example, if you need to set the packet size of the endpoint to 64 bytes, you need to set this register to 8**)

When you need to get/set this information, you must first use INDEX to select the target endpoints 1~5.

25.2.16 OUT Endpoint Control Status Register 1 (OUTCSR1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR1	14H	CLRDT	STSTL	SDSTL	FLUSH	DATERR	OVRRUN	FIFOFUL	OPRDY

CLRDT (Clear Data Toggle): resets the OUT data toggle bit.

When the OUT endpoint needs to reset the data toggle bit to "0" due to reconfiguration or STALL, the software needs to write "1" to this data bit.

STSTL (Sent Stall): STALL signal sending completion flag.

When the STALL signal has been sent, the hardware sets this position to "1". This flag must be cleared to "0" by software.

SDSTL (Send Stall): the STALL signal send request bit.

Software shall write a "1" to this bit to generate a STALL signal in response to an OUT token. Software should write a "0" to this bit to end the STALL signal. This bit has no effect on the ISO method.

FLUSH (FIFO Flush): clears the next packet from the FIFO at the OUT endpoint.

Writing a "1" to this bit will clear the next packet from the OUT endpoint FIFO. the FIFO pointer is reset and the OPRDY bit is cleared to "0". If the FIFO contains more than one packet, software must write a "1" to FLUSH for each packet. When the FIFO is empty, hardware clears the FLUSH bit to "0".

OVRRUN (Data Overrun): Data overrun.

This bit is set to "1" by hardware when an input packet cannot be loaded into the OUT endpoint FIFO. This bit is only valid in ISO mode. This bit must be cleared to "0" by software.

0: No data overflow

1: Packet loss due to FIFO full since this flag was last cleared

FIFOFUL (FIFO Full): FIFO data full flag of OUT endpoint.

0: FIFO at OUT endpoint not full

1: FIFO at OUT endpoint is full

OPRDY (OUT Packet Ready): OUT packet reception completion flag.

Hardware sets this bit to '1' when a packet is available. Software should clear this bit to '0' after each packet is unloaded from the OUT endpoint FIFO.

When you need to get/set this information, you must first use INDEX to select the target endpoints 1~5.

25.2.17 OUT Endpoint Control Status Register 2 (OUTCSR2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR2	15H	AUTOCLR	ISO	ENDMA	DMAMD	-	-	-	-

AUTOCLR: Automatically clears the OPRDY flag control bit.

0: Disable automatic clearing of the OPRDY flag

1: Enable automatic clearing of OPRDY (data must be downloaded from the OUT FIFO to reach the value set by OUTMAXP, otherwise OPRDY)

(Flag must be removed manually)

ISO: Synchronous transmission enable.

0: Endpoint is configured for batch/interrupt transmission mode

1: Endpoint is configured for synchronous transfer mode

ENDMA: DMA control of OUT endpoints

0: Disable DMA request for OUT endpoints

1: Enable DMA requests for OUT endpoints

DMAMD: Sets the DMA mode of the OUT end point

When you need to get/set this information, you must first use INDEX to select the target endpoints 1~5.

25.2.18 OUT length of USB endpoint 0 (COUNT0)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
COUNT0	16H	-	OUTCNT0[6:0]						

OUTCNT0[6:0]: OUT byte length for endpoint 0

COUNT0 is used to store the data length of the last OUT packet received at endpoint 0 (since endpoint 0 packets can only be at most

(64 bytes, so only 7 bits are needed). This length value is only valid if the OPRDY bit of endpoint 0 is "1". To obtain this length information, you must first use INDEX to select the target endpoint 0.

25.2.19 OUT length of USB endpoint (OUTCOUNTn)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
OUTCOUNT1	16H	OUTCNT[7:0]							
OUTCOUNT2	17H	-	-	-	-	-	OUTCNT[10:8]		

OUTCNT[10:0]: OUT byte length for endpoints

OUTCOUNT1 and OUTCOUNT2 combine to form an 11-bit number that holds the data length of the last OUT packet for endpoints 1 to 5. This length value is valid only when the OPRDY bit of endpoints 1 to 5 is "1".

When you need to get this length information, you must first use INDEX to select the target endpoints 1~5.

25.2.20 FIFO Data Access Register (FIFOn) for USB endpoints

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
FIFO0	20H	FIFO0[7:0]							
FIFO1	21H	FIFO1[7:0]							
FIFO2	22H	FIFO2[7:0]							
FIFO3	23H	FIFO3[7:0]							
FIFO4	24H	FIFO4[7:0]							
FIFO5	25H	FIFO5[7:0]							

FIFOn[7:0]: USB indirect access registers for IN/OUT data of each end point

25.2.21 USB Tracking Control Register (UTRKCTL)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UTRKCTL	30H	FTM1	FTM0	INTV[1:0]		ENST5	RES[2:0]		

FTM1: HFOSC fine-tuning control bit

0: Hardware uses coarse and fine tuning for adjusting the frequency

1: Hardware adjusts HFOSC

using trim bits only FTM0: valid

when FTM1 is 0

0: UTRK uses all 128 levels of trimming

1: UTRK prohibits the use of the maximum and minimum 12 levels of fine tuning.

INTV[1:0]: select UTRK update cycle

INTV[1:0]	cyclicality
00	2ms
01	4ms
10	8ms
11	16ms

ENST5: Enable plus 5th order and minus 5th order.

0: Upper and lower calibration limits of 10%

1: Calibration upper and

lower limits of 20% RES[2:0]: UTRK Auto Adjustment Resolution Setting

RES[2:0]	resolution (of a photo)	adjustment value

000	8	0.067 per cent
001	12	0.100%
010	16	0.133 per cent
011	**24**	**0.200 per cent**
100	28	0.233 per cent
101	32	0.267 per cent
110	48	0.4 per cent
111	64	0.5 per cent

25.2.22 USB Tracking Status Register (UTRKSTS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
UTRKSTS	31H	INTVCNT[3:0]				STS[1:0]		TST_UTRK	UTRK_RDY

INTVCNT[3:0]: internal count status STS[1:0]: UTRK status

STS[1:0]	state of affairs
00	**INC 5**
01	DEC 5
10	INC 1
11	DEC 1

TST_UTRK: UTRK test pattern control

0: Disable UTRK test mode

1: Enable UTRK test mode

UTRK_RDY: UTRK calibration completion status bit

25.3 USB Product Development Considerations

Each USB product must have its own unique VID&PID combination in order to be correctly recognised by the computer. If two different USB products have the same VID&PID combination, the USB product may be recognised by the computer abnormally and the USB product may not be used properly. To avoid this, VIDs and PIDs should be planned and assigned in a standardised way.

STC has now acquired the VID number 13503 (hex: 34BF) for STC's dedicated USB devices through the USB-IF organisation.Customers developing their own USB products using STC's USB chips can plan their own PIDs if you have already obtained your own VID by other means. If you need to use STC's official VID for your USB product, you must request the PID from STC.

25.4 sample procedure

25.4.1 Examples of HID Human Interface Devices

```
//Tested operating
frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                    // See download software for header files
#include "intrins.h"

        typedefunsigned char  BYTE;
        typedefunsigned int   WORD.
        typedefunsigned long  DWORD.

#define     FADDR           0
#define     POWER           1
#define     INTRIN1         2
#define     EP5INIF         0x20
#define     EP4INIF         0x10
#define     EP3INIF         0x08
#define     EP2INIF         0x04
#define     EP1INIF         0x02
#define     EP0IF           0x01
#define     INTROUT1        4
#define     EP5OUTIF        0x20
#define     EP4OUTIF        0x10
#define     EP3OUTIF        0x08
#define     EP2OUTIF        0x04
#define     EP1OUTIF        0x02
#define     INTRUSB         6
#define     SOFIF           0x08
#define     RSTIF           0x04
#define     RSUIF           0x02
#define     SUSIF           0x01
#define     INTRIN1E        7
#define     EP5INIE         0x20
#define     EP4INIE         0x10
#define     EP3INIE         0x08
#define     EP2INIE         0x04
#define     EP1INIE         0x02
#define     EP0IE           0x01
#define     INTROUT1E       9
#define     EP5OUTIE        0x20
#define     EP4OUTIE        0x10
#define     EP3OUTIE        0x08
#define     EP2OUTIE        0x04
#define     EP1OUTIE        0x02
#define     INTRUSBE        11
#define     SOFIE           0x08
#define     RSTIE           0x04
#define     RSUIE           0x02
#define     SUSIE           0x01
#define     FRAME1          12
#define     FRAME2          13
#define     INDEX           14
```

```
#define INMAXP 16
#define CSR0 17
#define SSUEND 0x80
#define SOPRDY 0x40
#define SDSTL 0x20
#define SUEND 0x10
#define DATEND 0x08
#define STSTL 0x04
#define IPRDY 0x02
#define OPRDY 0x01
#define INCSR1 17
#define INCLRDT 0x40
#define INSTSTL 0x20
#define INSDSTL 0x10
#define INFLUSH 0x08
#define INUNDRUN 0x04
#define INFIFONE 0x02
#define INIPRDY 0x01
#define INCSR2 18
#define INAUTOSET 0x80
#define INISO 0x40
#define INMODEIN 0x20
#define INMODEOUT 0x00
#define INENDMA 0x10
#define INFCDT 0x08
#define OUTMAXP 19
#define OUTCSR1 20
#define OUTCLRDT 0x80
#define OUTSTSTL 0x40
#define OUTSDSTL 0x20
#define OUTFLUSH 0x10
#define OUTDATERR 0x08
#define OUTOVRRUN 0x04
#define OUTFIFOFUL 0x02
#define OUTOPRDY 0x01
#define OUTCSR2 21
#define OUTAUTOCLR 0x80
#define OUTISO 0x40
#define OUTENDMA 0x20
#define OUTDMAMD 0x10
#define COUNT0 22
#define OUTCOUNT1 22
#define OUTCOUNT2 23
#define FIFO0 32
#define FIFO1 33
#define FIFO2 34
#define FIFO3 35
#define FIFO4 36
#define FIFO5 37
#define UTRKCTL 48
#define UTRKSTS 49

#define EPIDLE 0
#define EPSTATUS 1
#define EPDATAIN 2
#define EPDATAOUT 3
#define EPSTALL -1

#define GET_STATUS 0x00
```

```
#define   CLEAR_FEATURE          0x01
#define   SET_FEATURE            0x03
#define   SET_ADDRESS            0x05
#define   GET_DESCRIPTOR         0x06
#define   SET_DESCRIPTOR         0x07
#define   GET_CONFIG             0x08
#define   SET_CONFIG             0x09
#define   GET_INTERFACE          0x0A
#define   SET_INTERFACE          0x0B
#define   SYNCH_FRAME            0x0C

#define   GET_REPORT             0x01
#define   GET_IDLE               0x02
#define   GET_PROTOCOL           0x03
#define   SET_REPORT             0x09
#define   SET_IDLE               0x0A
#define   SET_PROTOCOL           0x0B

#define   DESC_DEVICE            0x01
#define   DESC_CONFIG            0x02
#define   DESC_STRING            0x03
#define   DESC_HIDREPORT         0x22

#define   STANDARD_REQUEST       0x00
#define   CLASS_REQUEST          0x20
#define   VENDOR_REQUEST         0x40
#define   REQUEST_MASK           0x60

typedef struct
{
    BYTE
    bmRequestType; BYTE
                BYTE
    bmRequestType; BYTE
    bRequest; bRequest.
    BYTE    wValueL.
    BYTE    wValueH.
    BYTE    wIndexL.
    BYTE    wIndexH.
    BYTE    wLengthL.
    BYTE    wLengthH.
}SETUP.

typedef struct
{
    BYTE    bStage.
    WORD
    wResidue.
    BYTE    *pData.
}EP0STAGE.

void UsbInit();
BYTE ReadReg(BYTE addr).
void WriteReg(BYTE addr, BYTE dat);
BYTE ReadFifo(BYTE fifo, BYTE *pdat).
void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt);;

char code DEVICEDESC[18];
char code CONFIGDESC[41];
char code
```

```
HDMIPORTDESC[27]; char
code LANGIDDESC[4].
char code MANUFACTDESC[8];
char code PRODUCTDESC[30].
```

```
SETUP Setup.
EP0STAGE Ep0Stage.
BYTE xdata
HidFreature[64]; BYTE
xdata HidInput[64]; BYTE
xdata HidOutput[64].

void main()
{
    EAXFR = 1;                  //Enable access to XFR
    CKCON = 0x00;               //Set the external data bus speed to fastest
    WTST = 0x00;                //set the program code wait parameter.
                                //Assign a value of 0 to set the CPU to execute the
                                programme as fast as possible.

    p0m0 = 0x00;
    p0m1 = 0x00;
    p1m0 = 0x00;
    p1m1 = 0x00;
    p2m0 = 0x00;
    p2m1 = 0x00;
    p3m0 = 0x00;
    p3m1 = 0x00;
    p4m0 = 0x00;
    p4m1 = 0x00;
    p5m0 = 0x00;
    p5m1 = 0x00.

    UsbInit();

    EA = 1;

    while (1);
}

BYTE ReadReg(BYTE addr)
{
    BYTE dat.

    while (USBADR & 0x80);
    USBADR = addr | 0x80;
    while (USBADR & 0x80);
    dat = USBDAT.

    return dat.
}

void WriteReg(BYTE addr, BYTE dat)
{
    while (USBADR & 0x80);
    USBADR = addr & 0x7f;
    USBDAT = dat.
}

BYTE ReadFifo(BYTE fifo, BYTE *pdat)
{
    BYTE cnt.
    BYTE ret.
```

```
    ret = cnt = ReadReg(COUNT0);
```

```
        while (cnt--)
        {
            *pdat++ = ReadReg(fifo).
        }

        return ret;
}

void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt)
{
        while (cnt--)
        {
            WriteReg(fifo, *pdat++).
        }
}

void DelayXns(WORD delayTime)
{
        while( delayTime-- );
}

void UsbInit()
{
        p3m0 = 0x00;
        p3m1 = 0x03.

        P_SW2 |= 0x80.
        PLLCR = (1<<7)|(0<<5)|(1<<3)|(1<<1); // enable PLLCR.        // enable PLL
        DelayXns(100).
        CLKSEL = 0x02;                                              //select system clock source as internal pll output
        CLKDIV = 0;
        P_SW2 &= ~0x80;
        USBCLK = 0x90;
        USBCON = 0x90;

        WriteReg(FADDR, 0x00);
        WriteReg(POWER, 0x08);
        WriteReg(INTRIN1E, 0x3f);
        WriteReg(INTROUT1E, 0x3f);
        WriteReg(INTRUSBE, 0x00).
        WriteReg(POWER, 0x01).

        Ep0Stage.bStage = EPIDLE;
}

void usb_isr() interrupt 22
{
        BYTE intrusb.
        BYTE intrin.
        BYTE introut.
        BYTE csr.
        BYTE cnt.
        WORD len.

        intrusb = ReadReg(INTRUSB);
        intrin = ReadReg(INTRIN1);
        introut = ReadReg(INTROUT1).

        if (intrusb & RSTIF)
```

```
    {
        WriteReg(INDEX, 1);
        WriteReg(INCSR1, INCLRDT).
        WriteReg(INDEX, 1);
        WriteReg(OUTCSR1, OUTCLRDT).
        Ep0Stage.bStage = EPIDLE;
    }

    if (intrin & EP0IF)
    {
        WriteReg(INDEX, 0);
        csr = ReadReg(CSR0);
        if (csr & STSTL)
        {
            WriteReg(CSR0, csr & ~STSTL);
            Ep0Stage.bStage = EPIDLE.
        }
        if (csr & SUEND)
        {
            WriteReg(CSR0, csr | SSUEND).
        }

        switch (Ep0Stage.bStage)
        {
        case EPIDLE.
            if (csr & OPRDY)
            {
                Ep0Stage.bStage = EPSTATUS;
                ReadFifo(FIFO0, (BYTE *)&Setup);
                ((BYTE *)&Ep0Stage.wResidue)[0] = Setup.wLengthH;
                ((BYTE *)&Ep0Stage.wResidue)[1] = Setup.wLengthL;
                switch (Setup.bmRequestType &amp ; REQUEST_MASK)
                {
                case STANDARD_REQUEST.
                    switch (Setup.bRequest)
                    {
                    case SET_ADDRESS:
                        WriteReg(FADDR, Setup.wValueL);
                        break;
                    case SET_CONFIG:
                        WriteReg(INDEX, 1);
                        WriteReg(INCSR2, INMODEIN).
                        WriteReg(INMAXP, 8).
                        WriteReg(INDEX, 1);
                        WriteReg(INCSR2, INMODEOUT);
                        WriteReg(OUTMAXP, 8).
                        WriteReg(INDEX, 0);
                        break;
                    case GET_DESCRIPTOR.
                        Ep0Stage.bStage = EPDATAIN;
                        switch (Setup.wValueH)
                        {
                        case DESC_DEVICE.
                            Ep0Stage.pData =
                            DEVICEDESC; len =
                            sizeof(DEVICEDESC); break;
                        case DESC_CONFIG.
                            Ep0Stage.pData =
                            CONFIGDESC; len =
                            sizeof(CONFIGDESC).
```

```
                    break;
                case DESC_STRING.
                    switch (Setup.wValueL)
                    {
                    case 0.
                        Ep0Stage.pData =
                        LANGIDDESC; len =
                        sizeof(LANGIDDESC); break;
                    case 1.
                        Ep0Stage.pData = MANUFACTDESC;
                        len = sizeof(MANUFACTDESC);
                        break;
                    case 2.
                        Ep0Stage.pData =
                        PRODUCTDESC; len =
                        sizeof(PRODUCTDESC); break;
                    default.
                        Ep0Stage.bStage = EPSTALL;
                        break;
                    }
                    break;
                case DESC_HIDREPORT.
                    Ep0Stage.pData = HIDREPORTDESC;
                    len = sizeof(HIDREPORTDESC);
                    break;
                default.
                    Ep0Stage.bStage = EPSTALL;
                    break;
                }
                if (len < Ep0Stage.wResidue)
                {
                    Ep0Stage.wResidue = len;
                }
                break;
            default.
                Ep0Stage.bStage = EPSTALL;
                break;
            }
            break;
        case CLASS_REQUEST.
            switch (Setup.bRequest)
            {
            case GET_REPORT:
                Ep0Stage.pData = HidFreature;
                Ep0Stage.bStage = EPDATAIN;
                break;
            case SET_REPORT.
                Ep0Stage.pData = HidFreature;
                Ep0Stage.bStage =
                EPDATAOUT; break;
            case SET_IDLE:
                break;
            case GET_IDLE.
            case GET_PROTOCOL.
            case SET_PROTOCOL.
            default.
                Ep0Stage.bStage = EPSTALL;
                break;
            }
```

```
                    break;
                default.
                    Ep0Stage.bStage = EPSTALL;
                    break;
                }

                switch (Ep0Stage.bStage)
                {
                case EPDATAIN.
                    WriteReg(CSR0, SOPRDY).
                    goto L_Ep0SendData;
                    break;
                case EPDATAOUT:
                    WriteReg(CSR0, SOPRDY);
                    break;
                case EPSTATUS.
                    WriteReg(CSR0, SOPRDY | DATEND).
                    Ep0Stage.bStage = EPIDLE;
                    break;
                case EPSTALL.
                    WriteReg(CSR0, SOPRDY | SDSTL).
                    Ep0Stage.bStage = EPIDLE;
                    break;
                }
            }
            break;
        case EPDATAIN.
            if (! (csr & IPRDY))
            {
L_Ep0SendData.
                cnt = Ep0Stage.wResidue > 64 ? 64 : Ep0Stage.wResidue;
                WriteFifo(FIFO0, Ep0Stage.pData, cnt);
                Ep0Stage.wResidue -= cnt;
                Ep0Stage.pData += cnt;
                if (Ep0Stage.wResidue == 0)
                {
                    WriteReg(CSR0, IPRDY | DATEND).
                    Ep0Stage.bStage = EPIDLE;
                }
                else
                {
                    WriteReg(CSR0, IPRDY).
                }
            }
            break;
        case EPDATAOUT.
            if (csr & OPRDY)
            {
                cnt = ReadFifo(FIFO0,
                Ep0Stage.pData); Ep0Stage.wResidue -=
                cnt; Ep0Stage.pData += cnt;
                if (Ep0Stage.wResidue == 0)
                {
                    WriteReg(CSR0, SOPRDY | DATEND).
                    Ep0Stage.bStage = EPIDLE;
                }
                else
                {
                    WriteReg(CSR0, SOPRDY).
```

```
                }
            }
            break;
        }
    }

    if (intrin & EP1INIF)
    {
        WriteReg(INDEX, 1).
        csr = ReadReg(INCSR1);
        if (csr & INSTSTL)
        {
            WriteReg(INCSR1, INCLRDT).
        }
        if (csr & INUNDRUN)
        {
            WriteReg(INCSR1, 0);
        }
    }

    if (introut & EP1OUTIF)
    {
        WriteReg(INDEX, 1).
        csr = ReadReg(OUTCSR1);
        if (csr & OUTSTSTL)
        {
            WriteReg(OUTCSR1, OUTCLRDT).
        }
        if (csr & OUTOPRDY)
        {
            ReadFifo(FIFO1, HidOutput);
            WriteReg(OUTCSR1, 0).

            WriteReg(INDEX, 1).
            WriteFifo(FIFO1, HidOutput,
            64); WriteReg(INCSR1,
            INIPRDY).
        }
    }
}

char code DEVICEDESC[18] =
{
    0x12.                                   //bLength(18);
    0x01.                                   //bDescriptorType(Device).
    0x00,0x02.                              //bcdUSB(2.00).
    0x00.                                   //bDeviceClass(0);
    0x00, //bDeviceSubClass0); //bDeviceSubClass0   //bDeviceSubClass0).
    0x00.                                   //bDeviceProtocol(0);
    0x40.                                   //bMaxPacketSize0(64);
    0xbf,0x34.                              //idVendor(STCUSB:34bf).
    0x80,0x43.                              //idProduct(4380);
    0x00,0x01.                              //bcdDevice(1.00).
    0x01.                                   //iManufacturer(1).
    0x02.                                   //iProduct(2).
    0x00, //iSerialNumber(0).               //iSerialNumber(0).
    0x01.                                   //bNumConfigurations(1).
};

char code CONFIGDESC[41] =
```

```
{
    0x09.                                   //bLength(9);
    0x02.                                   //bDescriptorType(Configuration).
    0x29,0x00,                              //wTotalLength(41).
    0x01.                                   //bNumInterfaces(1).
    0x01.                                   //bConfigurationValue(1);
    0x00.                                   //iConfiguration(0).
    0x80.                                   //bmAttributes(BUSPower).
    0x32,                                   //MaxPower(100mA).

    0x09.                                   //bLength(9);
    0x04.                                   //bDescriptorType(Interface).
    0x00.                                   //bInterfaceNumber(0).
    0x00.                                   //bAlternateSetting(0);
    0x02.                                   //bNumEndpoints(2).
    0x03.                                   //bInterfaceClass(HID).
    0x00.                                   //bInterfaceSubClass(0);
    0x00.                                   //bInterfaceProtocol(0).
    0x00.                                   //iInterface(0);

    0x09.                                   //bLength(9);
    0x21.                                   //bDescriptorType(HID).
    0x01,0x01,                              //bcdHID(1.01).
    0x00.                                   //bCountryCode(0).
    0x01.                                   //bNumDescriptors(1);
    0x22,                                   //bDescriptorType (HID Report).
    0x1b,0x00.                              //wDescriptorLength(27);

    0x07,                                   //bLength(7);
    0x05,                                   //bDescriptorType(Endpoint).
    0x81.                                   //bEndpointAddress(EndPoint1 as    IN).
    0x03.                                   //bmAttributes(Interrupt).
    0x40,0x00,                              //wMaxPacketSize(64);
    0x01.                                   //bInterval(10ms).

    0x07,                                   //bLength(7);
    0x05,                                   //bDescriptorType(Endpoint).
    0x01.                                   //bEndpointAddress(EndPoint1 as    OUT).
    0x03.                                   //bmAttributes(Interrupt).
    0x40,0x00,                              //wMaxPacketSize(64);
    0x01.                                   //bInterval(10ms).
};

char code HIDREPORTDESC[27] =
{
    0x05,0x0c.                              //USAGE_PAGE(Consumer).
    0x09,0x01.                              //USAGE(Consumer Control).
    0xa1,0x01.                              //COLLECTION(Application).
    0x15,0x00.                              //    LOGICAL_MINIMUM(0).
    0x25,0xff.                              //    LOGICAL_MAXIMUM(255).
    0x75,0x08.                              //    REPORT_SIZE(8).
    0x95,0x40.                              //    REPORT_COUNT(64);
    0x09,0x01.                                  //USAGE(Consumer Control).
    0xb1,0x02.                              //    FEATURE(Data,Variable).
    0x09,0x01.                                  //USAGE(Consumer Control).
    0x81,0x02.                              //    INPUT(Data,Variable).
    0x09,0x01.                                  //USAGE(Consumer Control).
    0x91,0x02.                              //    OUTPUT(Data,Variable).
    0xc0.                                   //END_COLLECTION.
```

```
};

char code LANGIDDESC[4] =
{
    0x04,0x03,
    0x09,0x04,
};

char code MANUFACTDESC[8] =
{
    0x08,0x03,
    'S',0,
    'T',0,
    'C',0,
};

char code PRODUCTDESC[30] =
{
    0x1e,0x03,
    's',0,
    'T',0,
    'C',0,
    ' ',0,
    'U',0,
    'S',0,
    'B',0,
    ' ',0,
    'D',0,
    'e',0,
    'v',0,
    'i',0,
    'c',0,
    'e',0,
};
```

25.4.2 HID (Human Interface Device) Protocol Example

After downloading the code to the test box, you can use the HID assistant in the latest STC-ISP download software to detect the test.
For detailed code, please refer to "69-HID (Human Interface Device) Protocol Example" in the "STC32G Demo Kit" package on the official website.

25.4.3 CDC (Communication Device Class) protocol example

WIN10 or below need to install the driver in the sys directory, WIN10 and WIN11 do not need to install the driver after downloading the code to the test box, the PC can be recognised as a USB to serial port device.
Use the DB9 interface on the test box to communicate with other serial ports The data bit of the serial port only supports 8 bits and the stop bit only supports 1 bit.
Check digit support: no parity, odd parity, even parity, 1-checksum and 0-checksum Baud rate support up to 460800, and support custom baud rate
For detailed code, please refer to "70-CDC(Communication Device Class)" in the "STC32G Demo Program" package on the official website.
Examples of protocols"

The CDC protocol is a generic protocol for communication device classes, and the CDC examples on the STC website simulate the communication interface classes of the CDC protocol. (02) and data interface class (0a), the user interface on the PC is the serial port and the device is mounted on the "usbser.sys" generic driver of windows. The device is mounted on the windows "usbser.sys" generic driver. Using the common serial port assistant, you can select the "COMx" port to transfer data to the CDC device.

If the CDC device is used as a Bridge (USB to Serial Bridge), the baud rate, parity bit, stop bit, and other parameters passed down from the host computer driver need to be processed, and then the data passed down from the host computer is passed down to the third-party device through the correct baud rate, parity bit, stop bit, and other formats of the serial port data.

If the CDC device is only used as an ordinary USB device to transfer data directly from PC, it can not deal with the baud rate, parity bit, stop bit and other parameters, and the data transfer between the CDC device and PC will not be affected by the baud rate of the serial port at all. The average bit rate of the actual test data transmission can reach 2~4MBPS.

25.4.4 Example of a USB keyboard based on the HID protocol

Download the code to the test box to implement the basic functions of the USB keyboard.
LED17 is the NumLock light, LED16 is the CapsLock light, LED15 is the ScrollLock light, and KEY0~KEY7 in the matrix keys are 1~8 in the keyboard respectively.
For detailed code, please refer to "71 - HID Protocol Based USB Keyboard Example" in the "STC32G

Demo Kit" package on the official website.

25.4.5 Example of a USB mouse based on the HID protocol

Download the code to the test box to achieve the basic functions of the USB mouse.
KEY0 is the left mouse button, KEY1 is the middle mouse button,
KEY2 is the right mouse button KEY4 is the left shift, KEY5 is the
right shift, KEY6 is the up shift, and KEY7 is the down shift in the
matrix key.

For detailed code, please refer to "72 - HID Protocol Based USB Mouse Example" in the "STC32G Demo Kit" package on the official website.

25.4.6 Example based on the WINUSB protocol

WIN10 and below need to install the driver in the sys directory, WIN10 and WIN11 can use the "STC_WINUSB.exe" in the exe directory to test the driver.

For detailed code, please refer to "73 - WINUSB Protocol Based Example" in the "STC32G Demo Kit" package on the official website.

25.4.7 MSC (Mass Storage Class) protocol example

After downloading the code to the experiment box, it can be recognised as a 512K USB stick on the PC. The USB memory is an external 512K serial FLASH on the experiment box.
In the case where there is no external FLASH, you can also use the STC32G12K128's internal EEPOM as memory by changing the memory type to MEMTYPE_INT in the config.h file.
Then when downloading from the ISP, set the EEPROM size to 64K, then you can realise a USB memory stick with 64K capacity.

For detailed code, please refer to "74-MSC (Mass Storage Class) Protocol Example" in the "STC32G Demo Kit" package on the official website.

26 RTC Real Time Clock

Some of the STC32G series microcontrollers have a real-time clock control circuit integrated internally, which has the following main features:

- Low power consumption: RTC module operating current as low as **2uA@VCC=3.3V, 3uA@VCC=5.0V (typical)**
- Long time span: supports 2000 to 2099, and automatically determines leap years.
- Alarm clock: support a set of alarm clock settings
- Multiple interrupts are supported: alarm interrupt, day interrupt, hour interrupt, minute interrupt, second interrupt, 1/2 second interrupt, 1/8 second interrupt, 1/32 second interrupt
- Wake on power down support

26.1 RTC-related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
RTCCR	RTC Control Register	7EFE60H	-	-	-	-	-	-	-	RUNRTC	xxxx,xxx0
RTCCFG	RTC Configuration Register	7EFE61H	-	-	-	-	-	-	RTCCKS	SETRTC	xxxx,xx00
RTCIEN	RTC Interrupt Enable Register	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I	0000,0000
RTCIF	RTC Interrupt Request Register	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF	0000,0000
ALAHOUR	RTC Hourly value of the alarm	7EFE64H	-	-	-						xxx0,0000
ALAMIN	RTC Minute value of the alarm	7EFE65H	-	-							xx00,0000
ALASEC	RTC Alarm Clock seconds value	7EFE66H	-	-							xx00,0000
ALASSEC	1/128 second value for RTC alarms	7EFE67H	-								x000,0000
INIYEAR	RTC Year Initialisation	7EFE68H	-								x000,0000
INIMONTH	RTC Month Initialisation	7EFE69H	-	-	-	-					xxxx,0000
INIDAY	RTC Day Initialisation	7EFE6AH	-	-	-						xxx0,0000
INIHOUR	RTC Hourly Initialisation	7EFE6BH	-	-	-						xxx0,0000
INIMIN	RTC Minute Initialisation	7EFE6CH	-	-							xx00,0000
INISEC	RTC second initialisation	7EFE6DH	-	-							xx00,0000
INISSEC	RTC1/128 seconds initialisation	7EFE6EH	-								x000,0000
YEAR	Annualised value of RTC	7EFE70H	-								x000,0000
MONTH	Monthly count of RTC	7EFE71H	-	-	-	-					xxxx,0000
DAY	Daily count of RTC	7EFE72H	-	-	-						xxx0,0000
HOUR	Hourly count of RTC	7EFE73H	-	-	-						xxx0,0000

MIN	Minute count value of RTC	7EFE74H	-	-		xx00,0000
SEC	Seconds value of RTC	7EFE75H	-	-		xx00,0000
SSEC	1/128 second count value for RTC	7EFE76H	-			x000,0000

26.1.1 RTC Control Register (RTCCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RTCCR	7EFE60H	-	-	-	-	-	-	-	RUNRTC

RUNRTC: RTC module control bits

0: RTC off, RTC stop counting

1: Enable RTC and start RTC counting

26.1.2 RTC Configuration Register (RTCCFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RTCCFG	7EFE61H	-	-	-	-	-	-	RTCCKS	SETRTC

RTCCKS: RTC Clock Source Selection

0: Select external 32.768KHz clock source (external 32K crystal needs to be software activated first)

1: Select internal 32K clock source (requires software to start internal 32K oscillator first)

SETRTC: Set RTC initial

value write 0:

meaningless

Write 1:TriggerRTC register initialisation. When SETRTC is set to 1, the hardware automatically copies the values in registers INIYEARJNIMONTH, INIDAY, INIHOUR, INIMIN, INISEC, INISSEC to registers YEAR, MONTH, DAY,

HOUR, MIN, SEC, SSEC. After initial completion, the hardware will automatically clear the SETRTC bit to 0. Read 0: Setting the RTC-related time registers is complete!

Read 1: Hardware is in the process of setting up the RTC and is not yet complete

Note: Waiting for the initialisation to complete, it needs to be judged after "RTC Enable". It takes 1 cycle time of 32768Hz to set the RTC time, about 30.5us, due to the synchronisation, the actual waiting time is 0~30.5us, if sleep without waiting for the completion of setup, the RTC will stop counting due to the incomplete setup, and will continue to complete the setup and counting only after waking up, and if the setup is to use the RTC interrupt to wake up at this time, then the MCU cannot be woken up. If you set to wake up with RTC interrupt, it will not be possible to wake up the MCU.

26.1.3 RTC Interrupt Enable Register (RTCIEN)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RTCIEN	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I

EALAI: Alarm clock

interrupt enable

bit 0: disable alarm

clock interrupt

1: Enable alarm clock interrupt

EDAYI: One-day (24-hour) interrupt enable bit

0: Closing of one-day interruptions

1: Enabling one-day interruptions

EHOURI: One hour (60 minutes) interrupt enable bit

0: Closing of hourly interruptions

1: Enable hourly interrupt

EMINI: One-minute (60 seconds)
interrupt enable bit

0: Minute interruptions closed

1: Enable minute interrupt ESECI: One second interrupt enable bit 0: Disable second interrupt 1: Enable second interrupt ESEC2I: 1/2 second interrupt enable bit 0: Disable 1/2 second interrupt 1: Enable 1/2 second interrupt ESEC8I: 1/8 second interrupt enable bit 0: Disable 1/8 second interrupt 1: Enable 1/8 second interrupt ESEC32I: 1/32 second interrupt enable bit 0: Disable 1/32 second interrupt 1: Enable 1/32 second interrupt

26.1.4 RTC Interrupt Request Register (RTCIF)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RTCIF	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF	SECIF	SEC2IF	SEC8IF	SEC32IF

ALAIF: Alarm clock interrupt request bit. Software clear 0 is required, software write 1 is invalid. DAYIF: One day (24 hours) interrupt request bit. It needs to be cleared by software, and writing 1 by software is invalid. HOURIF: One hour (60 minutes) interrupt request bit. A software clear of 0 is required and a software write of 1 is invalid. MINIF: One minute (60 seconds) interrupt request bit. A software clear of 0 is required and a software write of 1 is invalid. SECIF: One second interrupt request bit. SECIF: One-second interrupt request bit. SEC2IF: 1/2 second interrupt request bit. Software clear 0 is required, software write 1 is invalid. SEC8IF: 1/8 second interrupt request bit. SEC8IF: 1/8 second interrupt request bit. Software clear 0 is required, software write 1 is invalid. SEC32IF: 1/32 second interrupt request bit. SEC32IF: 1/32 second interrupt request bit.

26.1.5 RTC Alarm Setting Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
ALAHOUR	7EFE64H	-	-	-					
ALAMIN	7EFE65H	-	-						
ALASEC	7EFE66H	-	-						

ALASSEC	7EFE67H	-	

ALAHOUR: Sets the hourly value of the daily alarm.

Note: The value to be set is not the BCD code but the HEX code. For example, if you need to set the hour value 20 to ALAHOUR, you need to set it using the following code

```
MOV         WR6,#WORD0
ALAHOUR MOV         WR4,#WORD2
ALAHOUR MOV         A,#14H
MOV         @DR4,R11
```

ALAMIN: Sets the minute value for the daily alarm. The numeric code is the same as ALAHOUR. ALASEC: Sets the seconds value for the daily alarm. The numeric code is the same as ALAHOUR. ALASSEC: Sets the 1/128 second value for the daily alarm. The numeric code is the same as ALAHOUR.

26.1.6 RTC Real Time Clock Initial Value Setting Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
INIYEAR	7EFE68H	-							
INIMONTH	7EFE69H								
INIDAY	7EFE6AH								
INIHOUR	7EFE6BH	-	-	-					
INIMIN	7EFE6CH	-	-						
INISEC	7EFE6DH	-	-						
INISSEC	7EFE6EH	-							

INIYEAR: Set the year value of current real time. Valid values range from 00 to 99, corresponding to 2000 to 2099.

Note: The value to be set is not BCD code, but HEX code, for example, if you need to set 20 to INIYEAR, you need to use the following code to set it

```
MOVWR6,#WORD0 INIYEAR
MOVWR4,#WORD2 INIYEAR
MOV A,#14H
MOV @DR4,R11
```

INIMONTH: Set the month value of current real time. The valid value range is 1~12. The numeric code is the same as INIYEAR. INIDAY: Set the day value of current real time. Valid value range is 1~31. The numeric code is the same as INIYEAR. INIHOUR: Set the hour value of current real-time. Valid value range is 00~23. The numeric code is the same as INIYEAR. INIMIN: Set the minute value of current real-time. Valid value range is 00~59. The numeric code is the same as INIYEAR. INISEC: Set the second value of current real-time. The valid value range is 00~59. The numeric code is the same as INIYEAR. INISSEC: Sets the current real-time value of 1/128 seconds. Valid range is 00~127, the numeric code is the same as INIYEAR.

After the user has finished setting the initial value registers above, the user also needs to write a 1 to the SETRTC bit (RTCCFG.0) to trigger the hardware to load the initial value into the RTC real-time counter

Another note: the hardware will not check the validity of the initialisation data, which requires the user to set the initial value, must ensure the validity of the data, can not exceed its valid range.

26.1.7 RTC Real Time Clock Counter Register

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
YEAR	7EFE70H	-							
MONTH	7EFE71H								
DAY	7EFE72H								
HOUR	7EFE73H	-	-	-					
MIN	7EFE74H	-	-						
SEC	7EFE75H	-	-						
SSEC	7EFE76H	-							

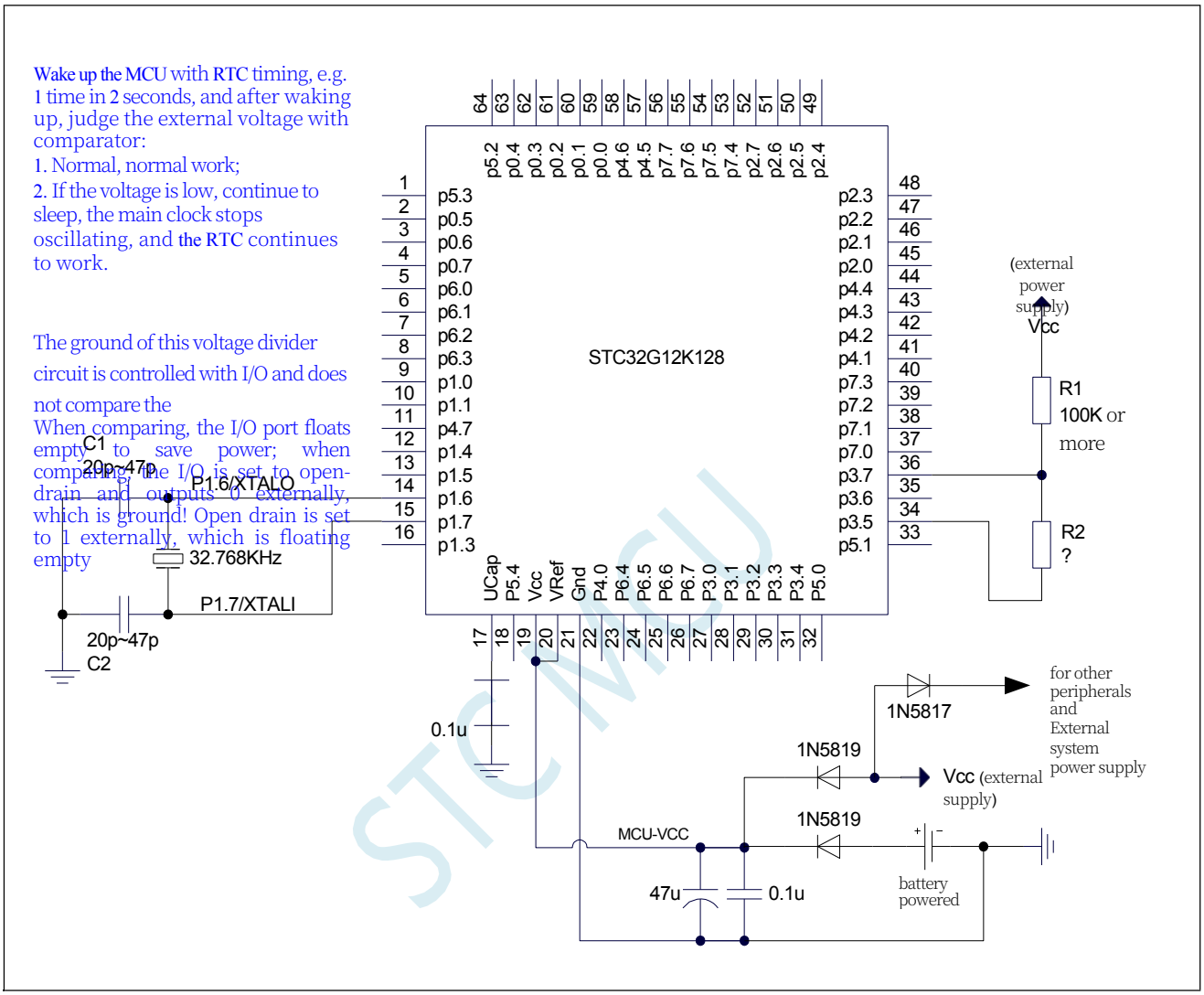
YEAR: the year value of the current real time. **Note: The value of the register is not BCD code, but HEX code.**

MONTH: Monthly value of the current real time. The numeric code is the same as YEAR. DAY: the day value of the current real-time. The numeric code is the same as YEAR. HOUR: Hour value of current real-time. The numeric code is the same as YEAR.

MIN: Minute value of the current real time. The numeric code is the same as YEAR. SEC: seconds value of current real-time. The numeric code is the same as YEAR. SSEC: 1/128 second value of the current real time. The numeric code is the same as YEAR.

Note: YEAR, MONTH, DAY, HOUR, MIN, SEC, and SSEC are read-only registers; if a write operation is required to be performed on these registers, it must be done through registers INIYEARĮNIMONTIHĮNIDATĮNIHOUĮNIMINĮNISECĮNISSEC, and SETRTC.

26.2 RTC real-world wiring diagram



26.3 sample procedure

26.3.1 Serial Print RTC Clock Example

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"
#include "stdio.h"

#define     MAIN_Fosc           22118400L
#define     Baudrate            115200L
#define     TM                  (65536 -(MAIN_Fosc/Baudrate+2)/4)
                                                       The // plus 2 operation is to allow the Keil compiler to
                                                       //Automatic implementation of rounding operations

bit         B1S_Flag.

void RTC_config(void);

void UartInit(void)
{
    SCON = (SCON & 0x3f) | 0x40;
    TL2 = TM.
    TH2 = TM>>8;
    S1BRT = 1;
    T2x12 = 1.
    T2R = 1.
}

void UartPutc(unsigned char dat)
{
    SBUF = dat;
    while(TI==0);
    TI = 0;
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

void RTC_Isr() interrupt 13
{
    if(RTCIF & 0x08)            //determine if second interrupt
    {
        rtcif &= ~0x08.                                // clear interrupt flag
        B1S_Flag = 1;
    }
}

void main(void)
{
    EAXFR = 1;                                         //Enable access to XFR
```

```
    CKCON = 0x00;                                   //Set the external data bus speed to fastest
    WTST = 0x00;                                    //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.

    P0M1 = 0.    P0M0 = 0.      //Set quasi-
                                bidirectional
                                port
    P1M1 = 0.    P1M0 = 0.      //Set quasi-
                                bidirectional
                                port
    P2M1 = 0.    P2M0 = 0.      //Set quasi-
                                bidirectional
                                port
    P3M1 = 0.    P3M0 = 0.      //Set quasi-
                                bidirectional
                                port
    P4M1 = 0.    P4M0 = 0.      //Set quasi-
                                bidirectional
                                port
    P5M1 = 0.    P5M0 = 0.      //Set quasi-
                                bidirectional
                                port

    UartInit();
    RTC_config();
    EA = 1;
    printf("RTC Test Programme!\r\n");              //UART sends a string

    while (1)
    {
        if(B1S_Flag)
        {
            B1S_Flag = 0;

            printf("Year=20%bd ", YEAR);
            printf("Month=%bd ", MONTH);
            printf("Day=%bd ", DAY);
            printf("Hour=%bd ", HOUR);
            printf("Minute=%bd ", MIN); printf(
            "Second=%bd ", SEC); printf("\r\n");
        }
    }
}

void RTC_config(void)
{
    //Select internal 32K
    IRC32KCR = 0x80.                                //start internal 32K oscillator
    while (! (IRC32KCR & 0x01));                    //Wait for the clock to stabilise
RTCCFG |= 0x02; //Select internal 32K as RTC clock source.  //Select internal 32K as RTC clock
                                                            source.

//  //Select external 32K
    //X32KCR = 0xc0.                                //Start external 32K crystal
    //while (! (X32KCR & 0x01));                    //wait for the clock to stabilise
        //RTCCFG &= ~0x02;                              //RTCCFG &= ~0x02; //Select
external 32K as RTC clock source.                   //Select external 32K as RTC
                                                    clock source

    INIYEAR = 21;                                   //Y:2021
```

```
INIMONTH = 12;                  //M:12
INIDAY = 31;                    //D:31
INIHOUR = 23;                   //H:23
INIMIN = 59;                    //M:59
INISEC = 50;                    //S:50
INISSEC = 0;                    //S/128:0
RTCCFG |= 0x01.                 //Trigger RTC register initialisation
while(RTCCFG & 0x01).           //Wait for initialisation to complete, need to be judged
after "RTC enable".

                                //Setting the RTC time requires 1 cycle time at 32768Hz.
                                //Approximately 30.5us. Due to synchronisation, the
                                actual wait time is 0~30.5us.
//If you sleep without waiting for the setting to complete, the RTC will sleep because
the setting is not completed.
                                //Complete, stop counting, wake up to finish setting and
                                continue counting.
```

```
    RTCIF = 0;                  //clear interrupt flag
    RTCIEN = 0x08;              //Enable RTC second interrupt
    RTCCR = 0x01.               //RTC enable
}
```

assembly code

Save the following code as an ASM file and load it into the project, e.g. *isr.asm.*

```
            CSEGAT 0123H
    JMP     006BH
    END
```

27 LCM Interface (8/16-bit Colour Module I8080/M6800) (Interface)

Some of the STC32G series microcontrollers have an integrated LCM interface controller, which can be used to drive popular LCD modules. It can drive I8080 interface and M6800 interface colour screens, supporting 8-bit and 16-bit data widths.

27.1 LCM Interface Function Pin Switching

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80
LCMIFCFG2	7EFE51H		LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

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

lcmifcps [1:0]	RS	I8080 read signal RD M6800 enable signal E	I8080 write signal WR Read/write signal RW of M6800
00	P4.5	P4.4	P4.2
01	P4.5	P3.7	P3.6
10	P4.0	P4.4	P4.2
11	P4.0	P3.7	P3.6

LCMIFDPS[1:0]: 8-bit data bits LCM interface data pin select bit

lcmifdps [1:0]	D16_D8	Data bytes 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 bits LCM interface data pin select bits

LCMIFDPS [1:0]	D16_D8	Data High Byte DAT[15:8]	Data Low Byte DAT[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	{P0[7:4], P4[7], P4[6], P4[3], P4[1]}
11	1	P6[7:0]	P7[7:0]

27.2 LCM Related Registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
LCMIFCFG	LCM Interface Configuration Register	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80	0x00,0000

LCMIFCFG2	LCM Interface Configuration Register 2	7EFE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]		x000,0000
LCMIFCR	LCM Interface Control Register	7EFE52H	ENLCMIF	-	-	-	-	CMD[2:0]			0xxx,x000
LCMIFSTA	LCM Interface Status Register	7EFE53H	-	-	-	-	-	-	-	LCMIFIF	xxxx,xxx0

LCMIFDATL	LCM Interface Low Byte Data	7EFE54H	LCMIFDAT[7:0]	0000,0000
LCMIFDATH	LCM Interface High Byte Data	7EFE55H	LCMIFDAT[15:8]	0000,0000

27.2.1 LCM Interface Configuration Register (LCMIFCFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFE50H	LCMIFIE	-	LCMIFIP[1:0]		LCMIFDPS[1:0]		D16_D8	M68_I80

LCMIFIE: LCM Interface Interrupt Enable Control Bit

0: Disable LCM interface interrupt

1: Allow LCM interface interrupt

LCMIFIP[1:0]: LCM interface interrupt priority control bits

LCMIFIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Highest (3)

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

LCMIFDPS [1:0]	D16_D8	Data High Byte DAT[15:8]	Data Low Byte DAT[7:0]
00	0	N/A	P2[7:0]
01	0	N/A	P6[7:0]
10	0	N/A	P2[7:0]
11	0	N/A	P6[7:0]
00	1	P2[7:0]	P0[7:0]
01	1	P6[7:0]	P2[7:0]
10	1	P2[7:0]	{P0[7:4], P4[7], P4[6], P4[3], P4[1]}
11	1	P6[7:0]	P7[7:0]

D16_D8: LCM interface data width control bits

0: 8-bit data width

1: 16-bit data width

M68_I80: LCM interface mode selection bit

0: I8080 mode

1: M6800 mode

27.2.2 LCM Interface Configuration Register 2 (LCMIFCFG2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG2	7EFE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			HOLDT[1:0]	

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

lcmifcps [1:0]	RS	I8080 read signal RD M6800 enable signal E	I8080 write signal WR Read/write signal RW of M6800
00	P4.5	P4.4	P4.2
01	P4.5	P3.7	P3.6
10	P4.0	P4.4	P4.2
11	P4.0	P3.7	P3.6

SETUPT[2:0]: data establishment time control bits for LCM interface communication (see timing diagrams in subsequent chapters for details)

HOLDT[1:0]: data hold time control bits for LCM interface communication (see timing diagrams in subsequent sections for details)

27.2.3 LCM Interface Control Register (LCMIFCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCR	7EFE52H	ENLCMIF	-	-	-	-	CMD[2:0]		

ELCMIF: LCM interface enable control bit

0: Disable LCM interface function

1: Allow LCM interface function

CMD[2:0]: LCM interface trigger command

CMD[2:0]	trigger command
100	write an order
101	write data
110	Read command/status
111	read data

27.2.4 LCM Interface Status Register (LCMIFSTA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFSTA	7EFE53H	-	-	-	-	-	-	-	LCMIFIF

LCMIFIF: LCM interface interrupt request flag, need to be cleared by software.

27.2.5 LCM Interface Data Registers (LCMIFDATL, LCMIFDATH)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFDATL	7EFE54H	LCMIFDAT[7:0]							
LCMIFDATH	7EFE55H	LCMIFDAT[15:8]							

LCMIFDAT: LCM interface data register.

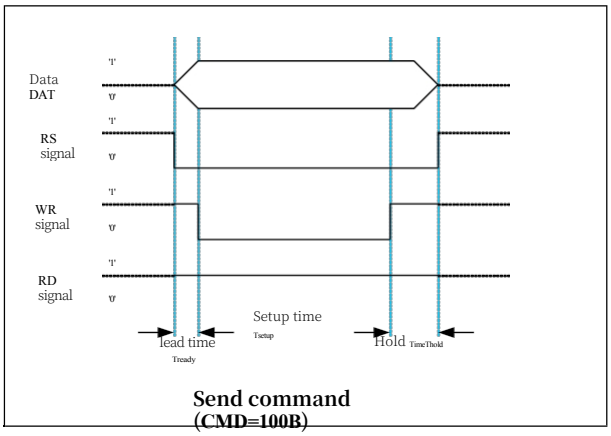
When the data width is 8-bit data, only LCMDATL data is valid;

When the data width is 16-bit data, LCMDATL and LCMDATH are combined to form 16-bit data.

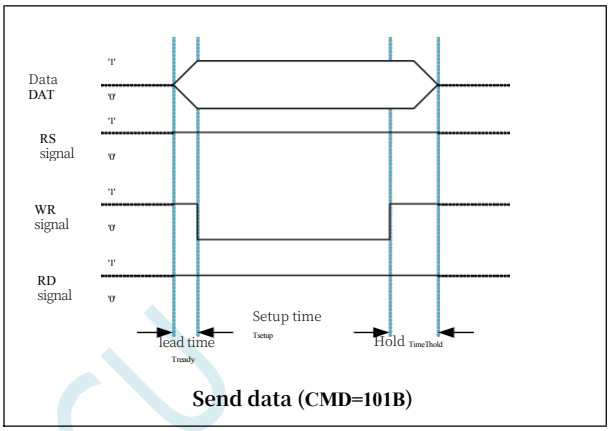
27.3 I8080/M6800 Mode LCM Interface Timing Chart

Note: T_{ready} = 1 system clock
T_{setup} = (SETUPT + 1) system clock
T_{hold} = (HOLDT + 1) system clock

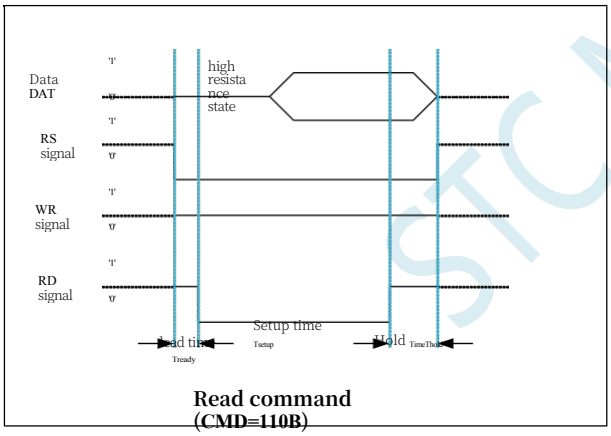
27.3.1 I8080 mode



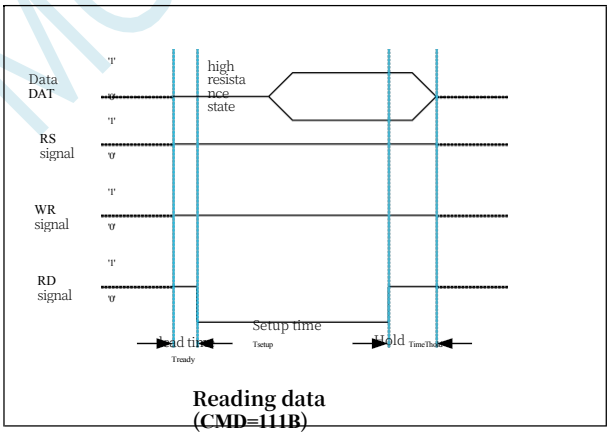
Send command (CMD=100B)



Send data (CMD=101B)

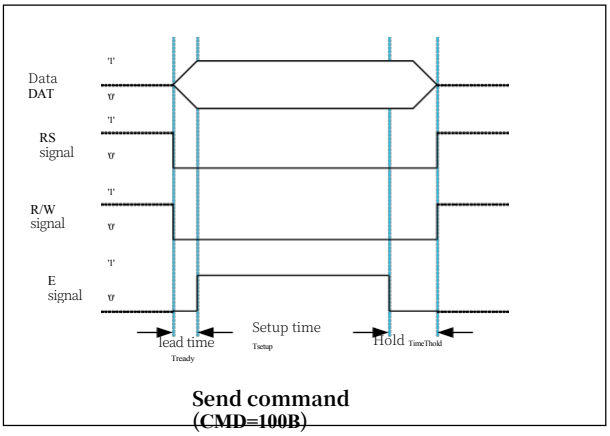


Read command (CMD=110B)

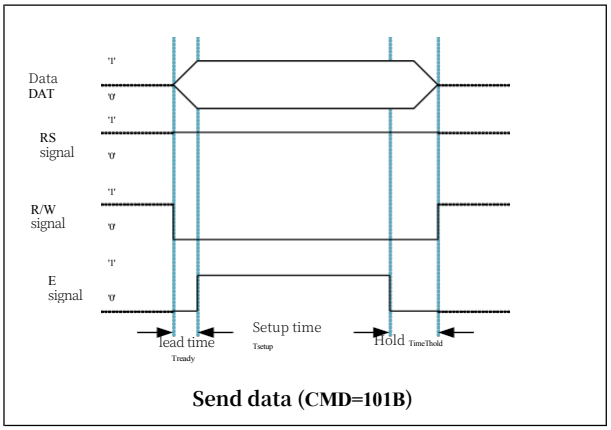


Reading data (CMD=111B)

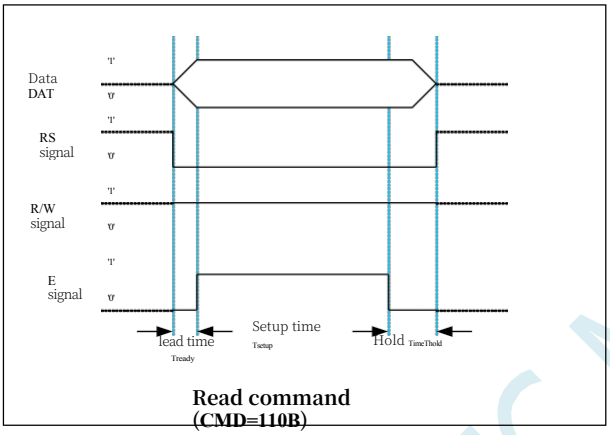
27.3.2 M6800 Mode



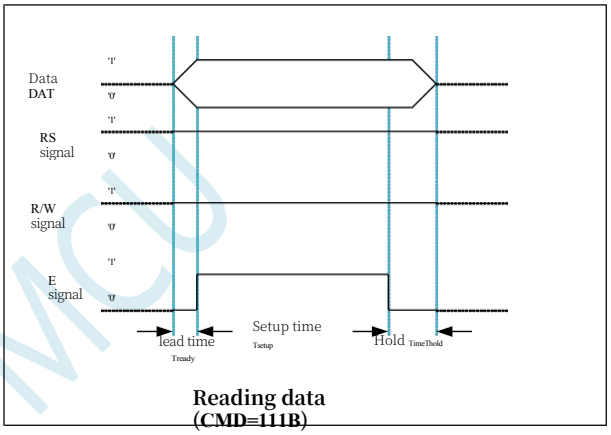
Send command (CMD=100B)



Send data (CMD=101B)



Read command (CMD=110B)



Reading data (CMD=111B)

28 DMA (bulk data transfer)

Some of the STC32G series microcontrollers support the bulk data storage function, i.e., traditional DMA. the following types of DMA operations are supported:

- M2M_DMA: XRAM memory to XRAM memory data read/write
- ADC_DMA: automatically scans the enabled ADC channel and stores the converted ADC data into XRAM automatically.
- SPI_DMA: Automatically exchanges data between data in XRAM and SPI peripherals.
- UR1T_DMA: Automatically sends data from XRAM out through serial port 1
- UR1R_DMA: Automatically stores data received by serial port 1 into XRAM
- UR2T_DMA: Automatically sends data from XRAM out via serial port 2
- UR2R_DMA: Automatically stores data received by serial port 2 into XRAM
- UR3T_DMA: Automatically sends data from XRAM out via serial port 3
- UR3R_DMA: Automatically stores data received by serial port 3 into XRAM
- UR4T_DMA: Automatically sends data from XRAM out via serial port 4
- UR4R_DMA: Automatically stores data received by serial port 4 into XRAM
- LCM_DMA: automatically exchanges data between data in XRAM and LCM devices
- I2CT_DMA: Automatically sends data from XRAM over the I2C interface
- I2CR_DMA: Automatically store data received from I2C into XRAM
- I2ST_DMA: Automatically sends data from XRAM out over I2S
- I2SR_DMA: Automatically stores I2S received data into XRAM

The maximum amount of data per DMA data transfer is 65536 bytes.

Each DMA can set 4 levels of access priority for reading and writing to XRAM, and the hardware will arbitrate the access to XRAM bus automatically, which will not affect the CPU's access to XRAM. 相同优先级下，不同 DMA 对 XRAM 的访问顺序如下：M2M_DMA， ADC_DMA， SPI_DMA，UR1R_DMA，UR1T_DMA，UR2R_DMA，UR2T_DMA，UR3R_DMA，UR3T_DMA，UR4R_DMA，UR4T_DMA，LCM_DMA，I2CR_DMA, I2CT_DMA, I2SR_DMA, I2ST_DMA

28.1 DMA-related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_M2M_CFG	M2M_DMA Configuration Registers	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]		0x00,0000
DMA_M2M_CR	M2M_DMA control registers	7EFA01H	ENM2M	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA Status Register	7EFA02H	-	-	-	-	-	-	-	M2MIF	xxxx,xxx0
DMA_M2M_AMT	M2M_DMA Total bytes transferred	7EFA03H									0000,0000
DMA_M2M_AMTH	M2M_DMA Total bytes transferred	7EFA80H									0000,0000
DMA_M2M_DONE	M2M_DMA Transmission	7EFA04H									0000,0000

	Completion Bytes			
DMA_M2M_DONEH	M2M_DMA Transmission Completion Bytes	7EFA81H		0000,0000
DMA_M2M_TXAH	M2M_DMA Send High Address	7EFA05H		0000,0000
DMA_M2M_TXAL	M2M_DMA Transmit Low Address	7EFA06H		0000,0000
DMA_M2M_RXAH	M2M_DMA Receive High Address	7EFA07H		0000,0000

DMA_M2M_RXAL	M2M_DMA Receive Low Address	7EFA08H									0000,0000
DMA_ADC_CFG	ADC_DMA Configuration Register	7EFA10H	ADCIE	-	-	-	ADCMIP[1:0]		ADCPTY[1:0]		0xxx,0000
DMA_ADC_CR	ADC_DMA Control Register	7EFA11H	ENADC	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_ADC_STA	ADC_DMA Status Register	7EFA12H	-	-	-	-	-	-	-	ADCIF	xxxx,xxx0
DMA_ADC_RXAH	ADC_DMA Receive High Address	7EFA17H									0000,0000
DMA_ADC_RXAL	ADC_DMA Receive Low Address	7EFA18H									0000,0000
DMA_ADC_CFG2	ADC_DMA Configuration Register 2	7EFA19H	-	-	-	-	CVTIMESEL [3:0]				xxxx,0000
DMA_ADC_CHSW0	ADC_DMA channel enable	7EFA1AH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0	0000,0001
DMA_ADC_CHSW1	ADC_DMA channel enable	7EFA1BH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	1000,0000
DMA_SPI_CFG	SPI_DMA Configuration Register	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]		000x,0000
DMA_SPI_CR	SPI_DMA Control Register	7EFA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO	000x,xxx0
DMA_SPI_STA	SPI_DMA Status Register	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF	xxxx,x000
DMA_SPI_AMT	SPI_DMA Total bytes transferred	7EFA23H									0000,0000
DMA_SPI_AMTH	SPI_DMA Total bytes transferred	7EFA84H									0000,0000
DMA_SPI_DONE	SPI_DMA transfer completion byte count	7EFA24H									0000,0000
DMA_SPI_DONEH	SPI_DMA transfer completion byte count	7EFA85H									0000,0000
DMA_SPI_TXAH	SPI_DMA Send High Address	7EFA25H									0000,0000
DMA_SPI_TXAL	SPI_DMA Send Low Address	7EFA26H									0000,0000
DMA_SPI_RXAH	SPI_DMA Receive High Address	7EFA27H									0000,0000
DMA_SPI_RXAL	SPI_DMA Receive Low Address	7EFA28H									0000,0000
DMA_SPI_CFG2	SPI_DMA Configuration Register 2	7EFA29H	-	-	-	-	-	WRPSS	SSS[1:0]		xxxx,x000
DMA_UR1T_CFG	UR1T_DMA Configuration Registers	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]		0xxx,0000
DMA_UR1T_CR	UR1T_DMA Control Registers	7EFA31H	ENUR1T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA Status Register	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF	xxxx,x0x0
DMA_UR1T_AMT	UR1T_DMA Total bytes transferred	7EFA33H									0000,0000
DMA_UR1T_AMTH	UR1T_DMA Total bytes transferred	7EFA88H									0000,0000
DMA_UR1T_DONE	UR1T_DMA Transmission Completion Bytes	7EFA34H									0000,0000
DMA_UR1T_DONEH	UR1T_DMA Transmission Completion Bytes	7EFA89H									0000,0000
DMA_UR1T_TXAH	UR1T_DMA Send High Address	7EFA35H									0000,0000
DMA_UR1T_TXAL	UR1T_DMA Send Low Address	7EFA36H									0000,0000
DMA_UR1R_CFG	UR1R_DMA Configuration	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]		0xxx,0000

	Registers										
DMA_UR1R_CR	UR1R_DMA Control Registers	7EFA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR1R_STA	UR1R_DMA Status Register	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF	xxxx,xx00
DMA_UR1R_AMT	UR1R_DMA Total bytes transferred	7EFA3BH									0000,0000
DMA_UR1R_AMTH	UR1R_DMA Total bytes transferred	7EFA8AH									0000,0000
DMA_UR1R_DONE	UR1R_DMA Transmission Completion Byte Count	7EFA3CH									0000,0000
DMA_UR1R_DONEH	UR1R_DMA Transmission Completion Byte Count	7EFA8BH									0000,0000
DMA_UR1R_RXAH	UR1R_DMA Receive High Address	7EFA3DH									0000,0000
DMA_UR1R_RXAL	UR1R_DMA Receive Low Address	7EFA3EH									0000,0000
DMA_UR2T_CFG	UR2T_DMA Configuration Registers	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]		0xxx,0000
DMA_UR2T_CR	UR2T_DMA Control Registers	7EFA41H	ENUR2T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA Status Register	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF	xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA Total bytes transferred	7EFA43H									0000,0000
DMA_UR2T_AMTH	UR2T_DMA Total bytes transferred	7EFA8CH									0000,0000

DMA_UR2T_DONE	UR2T_DMA Transfer Completion Bytes	7EFA44H								0000,0000
DMA_UR2T_DONEH	UR2T_DMA Transfer Completion Bytes	7EFA8DH								0000,0000
DMA_UR2T_TXAH	UR2T_DMA Send High Address	7EFA45H								0000,0000
DMA_UR2T_TXAL	UR2T_DMA Send Low Address	7EFA46H								0000,0000
DMA_UR2R_CFG	UR2R_DMA Configuration Registers	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	0xxx,0000
DMA_UR2R_CR	UR2R_DMA Control Registers	7EFA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR2R_STA	UR2R_DMA Status Register	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF	xxxx,xx00
DMA_UR2R_AMT	UR2R_DMA Total bytes transferred	7EFA4BH								0000,0000
DMA_UR2R_AMTH	UR2R_DMA Total bytes transferred	7EFA8EH								0000,0000
DMA_UR2R_DONE	UR2R_DMA Transmission Completion Bytes	7EFA4CH								0000,0000
dma_ur2r_doneh	UR2R_DMA Transmission Completion Bytes	7EFA8FH								0000,0000
DMA_UR2R_RXAH	UR2R_DMA Receive High Address	7EFA4DH								0000,0000
DMA_UR2R_RXAL	UR2R_DMA Receive Low Address	7EFA4EH								0000,0000
DMA_UR3T_CFG	UR3T_DMA Configuration Registers	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	0xxx,0000
DMA_UR3T_CR	UR3T_DMA Control Registers	7EFA51H	ENUR3T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA Status Register	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA Total bytes transferred	7EFA53H								0000,0000
DMA_UR3T_AMTH	UR3T_DMA Total bytes transferred	7EFA90H								0000,0000
DMA_UR3T_DONE	UR3T_DMA Transfer Completion Bytes	7EFA54H								0000,0000
DMA_UR3T_DONEH	UR3T_DMA Transfer Completion Bytes	7EFA91H								0000,0000
DMA_UR3T_TXAH	UR3T_DMA Send High Address	7EFA55H								0000,0000
DMA_UR3T_TXAL	UR3T_DMA Send Low Address	7EFA56H								0000,0000
DMA_UR3R_CFG	UR3R_DMA Configuration Registers	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	0xxx,0000
DMA_UR3R_CR	UR3R_DMA Control Registers	7EFA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR3R_STA	UR3R_DMA Status Register	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF	xxxx,xx00
DMA_UR3R_AMT	UR3R_DMA Total bytes transferred	7EFA5BH								0000,0000
DMA_UR3R_AMTH	UR3R_DMA Total bytes transferred	7EFA92H								0000,0000
DMA_UR3R_DONE	UR3R_DMA Transmission Completion Byte Count	7EFA5CH								0000,0000
dma_ur3r_doneh	UR3R_DMA Transmission Completion Byte Count	7EFA93H								0000,0000

DMA_UR3R_RXAH	UR3R_DMA Receive High Address	7EFA5DH									0000,0000
dma_ur3r_rxal	UR3R_DMA Receive Low Address	7EFA5EH									0000,0000
DMA_UR4T_CFG	UR4T_DMA Configuration Registers	7EFA60H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]		0xxx,0000
DMA_UR4T_CR	UR4T_DMA Control Registers	7EFA61H	ENUR4T	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA Status Register	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA Total bytes transferred	7EFA63H									0000,0000
DMA_UR4T_AMTH	UR4T_DMA Total bytes transferred	7EFA94H									0000,0000
DMA_UR4T_DONE	UR4T_DMA Transfer Completion Bytes	7EFA64H									0000,0000
DMA_UR4T_DONEH	UR4T_DMA Transfer Completion Bytes	7EFA95H									0000,0000
DMA_UR4T_TXAH	UR4T_DMA Send High Address	7EFA65H									0000,0000
DMA_UR4T_TXAL	UR4T_DMA Send Low Address	7EFA66H									0000,0000
DMA_UR4R_CFG	UR4R_DMA Configuration Registers	7EFA68H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]		0xxx,0000
DMA_UR4R_CR	UR4R_DMA Control Registers	7EFA69H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO	0x0x,xxx0
DMA_UR4R_STA	UR4R_DMA Status Register	7EFA6AH	-	-	-	-	-	-	RXLOSS	UR4RIF	xxxx,xx00
DMA_UR4R_AMT	UR4R_DMA Total bytes transferred	7EFA6BH									0000,0000

DMA_UR4R_AMTH	UR4R_DMA Total bytes transferred	7EFA96H								0000,0000
DMA_UR4R_DONE	UR4R_DMA Transmission Completion Bytes	7EFA6CH								0000,0000
DMA_UR4R_DONEH	UR4R_DMA Transmission Completion Bytes	7EFA97H								0000,0000
DMA_UR4R_RXAH	UR4R_DMA Receive High Address	7EFA6DH								0000,0000
DMA_UR4R_RXAL	UR4R_DMA Receive Low Address	7EFA6EH								0000,0000
DMA_LCM_CFG	LCM_DMA Configuration Register	7EFA70H	LCMIE	-	-	-	LCMIP[1:0]		LCMPTY[1:0]	0xxx,0000
DMA_LCM_CR	LCM_DMA Control Register	7EFA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	-	0000,0xxx
DMA_LCM_STA	LCM_DMA Status Register	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF	xxxx,xx00
DMA_LCM_AMT	LCM_DMA Total bytes transferred	7EFA73H									0000,0000
DMA_LCM_AMTH	LCM_DMA Total bytes transferred	7EFA86H									0000,0000
DMA_LCM_DONE	LCM_DMA transfer completion byte count	7EFA74H									0000,0000
DMA_LCM_DONEH	LCM_DMA transfer completion byte count	7EFA87H									0000,0000
DMA_LCM_TXAH	LCM_DMA Send High Address	7EFA75H									0000,0000
DMA_LCM_TXAL	LCM_DMA Send Low Address	7EFA76H									0000,0000
DMA_LCM_RXAH	LCM_DMA Receive High Address	7EFA77H									0000,0000
DMA_LCM_RXAL	LCM_DMA Receive Low Address	7EFA78H									0000,0000
DMA_I2CT_CFG	I2CT_DMA Configuration Registers	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]		0xxx,0000
DMA_I2CT_CR	I2CT_DMA Control Registers	7EFA99H	ENI2CT	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2CT_STA	I2CT_DMA Status Register	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF	xxxx,x0x0
DMA_I2CT_AMT	I2CT_DMA Total bytes transferred	7EFA9BH									0000,0000
DMA_I2CT_AMTH	I2CT_DMA Total bytes transferred	7EFAA8H									0000,0000
DMA_I2CT_DONE	I2CT_DMA Transfer Completion Bytes	7EFA9CH									0000,0000
DMA_I2CT_DONEH	I2CT_DMA Transfer Completion Bytes	7EFAA9H									0000,0000
DMA_I2CT_TXAH	I2CT_DMA Send High Address	7EFA9DH									0000,0000
DMA_I2CT_TXAL	I2CT_DMA Send Low Address	7EFA9EH									0000,0000
DMA_I2CR_CFG	I2CR_DMA Configuration Registers	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]		0xxx,0000
DMA_I2CR_CR	I2CR_DMA Control Registers	7EFAA1H	ENI2CR	TRIG	-	-	-	-	-	CLRFIFO	00xx,xxx0
DMA_I2CR_STA	I2CR_DMA Status Register	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF	xxxx,xx00
DMA_I2CR_AMT	I2CR_DMA Total bytes transferred	7EFAA3H									0000,0000

DMA_I2CR_AMTH	I2CR_DMA Total bytes transferred	7EFAAAH								0000,0000
DMA_I2CR_DONE	I2CR_DMA Transmission Completion Bytes	7EFAA4H								0000,0000
DMA_I2CR_DONEH	I2CR_DMA Transmission Completion Bytes	7EFAABH								0000,0000
DMA_I2CR_RXAH	I2CR_DMA Receive High Address	7EFAA5H								0000,0000
DMA_I2CR_RXAL	I2CR_DMA Receive Low Address	7EFAA6H								0000,0000
DMA_I2C_CR	I2C_DMA Control Register	7EFAADH	RDSEL	-	-	-	-	ACKERR	INTEN	BMMEN	0xxx,x000
DMA_I2C_ST1	I2C_DMA Status Register	7EFAAEH	COUNT[7:0]								0000,0000
DMA_I2C_ST2	I2C_DMA Status Register	7EFAAFH	COUNT[15:8]								0000,0000
DMA_I2ST_CFG	I2ST_DMA Configuration Registers	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]		0xxx,0000
DMA_I2ST_CR	I2ST_DMA Control Registers	7EFAB1H	ENI2ST	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2ST_STA	I2ST_DMA Status Register	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF	xxxx,x0x0
DMA_I2ST_AMT	I2ST_DMA Total bytes transferred	7EFAB3H									0000,0000
DMA_I2ST_AMTH	I2ST_DMA Total number of bytes transferred	7EFAC0H									0000,0000
DMA_I2ST_DONE	I2ST_DMA Transfer Completion Byte Count	7EFAB4H									0000,0000
DMA_I2ST_DONEH	I2ST_DMA Transfer Completion Byte Count	7EFAC1H									0000,0000

DMA_I2ST_TXAH	I2ST_DMA Send High Address	7EFAB5H								0000,0000
DMA_I2ST_TXAL	I2ST_DMA Send Low Address	7EFAB6H								0000,0000
DMA_I2SR_CFG	I2SR_DMA Configuration Registers	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]	0xxx,0000
DMA_I2SR_CR	I2SR_DMA Control Registers	7EFAB9H	ENI2SR	-	TRIG	-	-	-	-	CLRFIFO 0x0x,xxx0
DMA_I2SR_STA	I2SR_DMA Status Register	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF xxxx,xx00
DMA_I2SR_AMT	I2SR_DMA Total bytes transferred	7EFABBH								0000,0000
DMA_I2SR_AMTH	I2SR_DMA Total bytes transferred	7EFAC2H								0000,0000
DMA_I2SR_DONE	I2SR_DMA Transfer Completion Byte Count	7EFABCH								0000,0000
DMA_I2SR_DONEH	I2SR_DMA Transfer Completion Byte Count	7EFAC3H								0000,0000
DMA_I2SR_RXAH	I2SR_DMA Receive High Address	7EFABDH								0000,0000
DMA_I2SR_RXAL	I2SR_DMA Receive Low Address	7EFABEH								0000,0000
DMA_ARB_CFG	DMA President Configuration Register	7EFAF8H	WTRREN	-	-	-	STASEL[3:0]-			0xxx,0000
DMA_ARB_STA	DMA President Status Register	7EFAF9H								0000,0000

28.2 Memory-to-memory data read/write (M2M_DMA)

28.2.1 M2M_DMA Configuration Register (DMA_M2M_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CFG	7EFA00H	M2MIE	-	TXACO	RXACO	M2MIP[1:0]		M2MPTY[1:0]	

M2MIE: M2M_DMA interrupt enable control bit

0: M2M_DMA interrupt disabled

1: M2M_DMA interrupt allowed

TXACO: M2M_DMA source address (read address) change direction

0: Address is automatically incremented after data reading is complete

1: The address is automatically decremented

after data reading is complete RXACO: M2M_DMA

Destination address (write address) change

direction

0: The address is automatically incremented after the data is written.

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 level (1)
10	Higher (2)
11	Top level (3)

M2MPTY[1:0]: M2M_DMA data bus access priority control bits

M2MPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.2.2 M2M_DMA Control Register (DMA_M2M_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_CR	7EFA01H	ENM2M	TRIG	-	-	-	-	-	-

ENM2M: M2M_DMA function enable control bit

0: Disable M2M_DMA function

1: Allow M2M_DMA function

TRIG: M2M_DMA data read/write trigger control bit

0: Write 0 Invalid

1: Write 1 to start M2M_DMA operation.

28.2.3 M2M_DMA Status Register (DMA_M2M_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	7EFA02H	-	-	-	-	-	-	-	M2MIF

M2MIF: M2M_DMA interrupt request flag bit, when the M2M_DMA operation is completed, the hardware will automatically set M2MIF to 1. If enabled, M2MIF will be set to 1.

The M2M_DMA interrupt enters the interrupt service routine. Flag bits need to be cleared by software

28.2.4 M2M_DMA Transmit Total Byte Register (DMA_M2M_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_AMT	7EFA03H	AMT [7:0]							
DMA_M2M_AMTH	7EFA80H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.2.5 M2M_DMA transmission completion byte register (DMA_M2M_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_DONE	7EFA04H	DONE[7:0]							
DMA_M2M_DONEH	7EFA81H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been read and written.

28.2.6 M2M_DMA Transmit Address Register (DMA_M2M_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_TXAH	7EFA05H	ADDR[15:8]							
DMA_M2M_TXAL	7EFA06H	ADDR[7:0]							

DMA_M2M_TXA: Sets the source address for data reading and writing. M2M_DMA operation will start reading data from this address.

28.2.7 M2M_DMA Receive Address Register (DMA_M2M_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_RXAH	7EFA07H	ADDR[15:8]							
DMA_M2M_RXAL	7EFA08H	ADDR[7:0]							

DMA_M2M_RXA: Sets the destination address for data reading and writing. Data will be written from this address when M2M_DMA operation is performed.

28.3 ADC data auto-store (ADC_DMA)

28.3.1 ADC_DMA Configuration Register (DMA_ADC_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG	7EFA10H	ADCIE	-			ADCIP[1:0]		ADCPTY[1:0]	

ADCIE: ADC_DMA interrupt enable control bit

0: Disable ADC_DMA interrupt

1: Allow ADC_DMA interrupt

ADCIP[1:0]: ADC_DMA interrupt priority control bits

ADCIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

ADCPTY[1:0]: ADC_DMA data bus access priority control bits

ADCPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.3.2 ADC_DMA Control Register (DMA_ADC_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CR	7EFA11H	ENADC	TRIG	-	-	-	-	-	-

ENADC: ADC_DMA function enable control bit

0: Disable ADC_DMA function

1: Allow ADC_DMA function

TRIG: ADC_DMA operation trigger control bit

0: Write 0 Invalid

1: Write 1 to start ADC_DMA operation.

28.3.3 ADC_DMA Status Register (DMA_ADC_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_STA	7EFA12H	-	-	-	-	-	-	-	ADCIF

ADCIF: ADC_DMA interrupt request flag bit, when ADC_DMA finishes scanning all the enabled ADC channels, the hardware will automatically set the

ADCIF is set to 1. If ADC_DMA interrupt is enabled, it will enter the interrupt service routine.

Flags need to be cleared by software

28.3.4 ADC_DMA Receive Address Register (DMA_ADC_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_RXAH	7EFA17H	ADDR[15:8]							

DMA_ADC_RXAL	7EFA18H	ADDR[7:0]

DMA_ADC_RXA: Sets the storage address for ADC conversion data when ADC_DMA operation is performed.

28.3.5 ADC_DMA Configuration Register 2 (DMA_ADC_CFG2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG2	7EFA19H	-	-	-	-	CVTIMESEL [3:0]			

CVTIMESEL[3:0]: sets the number of ADC conversions for each ADC channel when performing

CVTIMESEL [3:0]	Number of conversions
0xxx	1 time
1000	2 times
1001	4 times
1010	8 times
1011	16 times
1100	32 times
1101	64 times
1110	128 times
1111	256 times

28.3.6 ADC_DMA channel enable register (DMA_ADC_CHSWx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CHSW0	7EFA1AH	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
DMA_ADC_CHSW1	7EFA1BH	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8

CHn: Sets the ADC channels that are automatically scanned during ADC_DMA operation. Channel scanning always starts with the channel with the smaller number.

28.3.7 ADC_DMA data storage format

注：ADC 转换速度和转换结果的对齐方式均由 ADC 相关寄存器进行设置

XRAM[DMA_ADC_RXA+0] = 使能的第 1 通道的第 1 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+1] = 使能的第 1 通道的第 1 次 ADC 转换结果的低字节;
XRAM[DMA_ ADC_RXA+2] = high byte of the 2nd ADC conversion result of the 1st channel enabled; XRAM[DMA_ADC_RXA+3] = low byte of the 2nd ADC conversion result of the 1st channel enabled.

...

XRAM[DMA_ADC_RXA+2n-2] = high byte of the nth ADC conversion result of the enabled channel 1; XRAM[DMA_ADC_RXA+2n-1] = low byte of the nth ADC conversion result of the enabled channel 1; XRAM[DMA_ADC_RXA+2n] = ADC channel number of channel 1.

XRAM[DMA_ADC_RXA+2n+1] = the remainder of the average of the n ADC conversion results of channel 1; XRAM[DMA_ADC_RXA+2n+2] = the high byte of the average of the n ADC conversion results of channel 1; XRAM[DMA_ADC_RXA+2n+3] = the low byte of the average of the n ADC conversion results of channel 1; XRAM[DMA_ADC_RXA+3] = the low byte of the average of the n ADC conversion results of channel 1; XRAM[DMA_ADC_RXA+3] = the low byte of the average of the n ADC conversion results of channel 1 XRAM[DMA_ADC_RXA+2n+3] = low byte of the average of the n ADC conversion results of channel 1

XRAM[DMA_ADC_RXA+(2n+3)+0] = high byte of 1st ADC conversion result for enabled channel 2; XRAM[DMA_ADC_RXA+(2n+3)+1] = low byte of 1st ADC conversion result for enabled channel 2; XRAM[DMA_ADC_RXA+(2n+3)+2] = high byte of 2nd ADC conversion result for enabled XRAM[DMA_ADC_RXA+(2n+3)+2] = high byte of 2nd ADC conversion result for enabled channel 2; XRAM[DMA_ADC_RXA+(2n+3)+3] = low byte of 2nd ADC conversion result for enabled channel 2.

...

XRAM[DMA_ADC_RXA+(2n+3)+2n-2] = high byte of the nth ADC conversion result for enabled channel 2; XRAM[DMA_ADC_RXA+(2n+3)+2n-1] = low byte of the nth ADC conversion result for enabled channel 2; XRAM[DMA_ADC_RXA+(2n+3)+2n] = ADC channel number of channel 2; XRAM[DMA_ADC_RXA+(2n+3)+2n] = ADC channel number of channel 2;
XRAM[DMA_ADC_RXA+(2n+3)+2n 2n] = 第 2 通道的 ADC 通道号;
XRAM[DMA_ADC_RXA+(2n+3)+2n+1] = the remainder of the average of the n ADC conversion results of channel 2; XRAM[DMA_ADC_RXA+(2n+3)+2n+2] = the high byte of the average of the n ADC conversion results of channel 2; XRAM[DMA_ADC_RXA+(2n+3) +2n+3] = the low byte of the average of the n ADC conversion results of channel 2; XRAM[DMA_ADC_RXA+(2n+3) +2n+3] = low byte of the average of the n ADC conversion results for channel 2.

...

XRAM[DMA_ADC_RXA+(m-1)(2n+3)+0] = 使能的第 m 通道的第 1 次 ADC 转换结果的高字节;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+1] = 使能的第 m 通道的第 1 次 ADC 转换结果的低字节;
XRAM[DMA_ADC_RXA+(m- 1)(2n+3)+2] = high byte of the 2nd ADC conversion result of the mth channel enabled; XRAM[DMA_ADC_RXA+(m-1)(2n+3)+3] = low byte of the 2nd ADC conversion result of the mth channel enabled.

...

XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n-2] = high byte of the nth ADC conversion result of the mth channel enabled; XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n-1] = low byte of the nth

ADC conversion result of the mth channel enabled; XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n] = the ADC channel number of the mth channel; XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n RXA+(m-1)(2n+3)+2n] = ADC channel number of the mth channel.
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n+1] = 第 m 通道的 n 次 ADC 转换结果取完平均值之后的余数;
XRAM[DMA_ADC_RXA+(m-1)(2n+3)+2n+2] =第 m 通道的 n 次 ADC 转换结果平均值的高字节;
XRAM[DMA_ADC_ RXA+(m-1)(2n+3)+2n+3] = low byte of the average of the nth ADC conversion result of the mth channel; XRAM[DMA_ADC_

The form is shown below:

ADC Channel	offset address	numbers
Channel 1	0	High byte of the first ADC conversion result of the enabled channel 1
	1	Low byte of the first ADC conversion result of the enabled channel 1
	2	High byte of the second ADC conversion result of the enabled channel 1
	3	Low byte of the 2nd ADC conversion result of the enabled channel 1
	...	...
	2n-2	High byte of the nth ADC conversion result of the 1st channel of the enabled channel.
	2n-1	Low byte of the nth ADC conversion result of the enabled channel 1
	2n	ADC channel number of channel 1
	2n+1	Remainder of the average of the n ADC conversion results of the 1st channel
	2n+2	High byte of the average of the n ADC conversion results of the 1st channel
	2n+3	Low byte of the average of the n ADC conversion results for channel 1
Channel 2	(2n+3) + 0	High byte of the first ADC conversion result of enabled channel 2
	(2n+3) + 1	Low byte of the first ADC conversion result of the enabled channel 2
	(2n+3) + 2	High byte of the 2nd ADC conversion result of the enabled channel 2
	(2n+3) + 3	Low byte of the 2nd ADC conversion result of the enabled channel 2
	...	...
	(2n+3) + 2n-2	High byte of the nth ADC conversion result of enabled channel 2
	(2n+3) + 2n-1	Low byte of the nth ADC conversion result of the enabled channel 2
	(2n+3) + 2n	ADC channel number of the 2nd channel
	(2n+3) + 2n+1	Remainder of the average of n ADC conversions for channel 2
	(2n+3) + 2n+2	High byte of the average of the n ADC conversion results for channel 2
	(2n+3) + 2n+3	Low byte of the average of n ADC conversion results for channel 2
	...	...
Channel m	(m-1)(2n+3) + 0	High byte of the 1st ADC conversion result of the mth channel of the enabled channel
	(m-1)(2n+3) + 1	Low byte of the 1st ADC conversion result of the mth channel of the enabled channel
	(m-1)(2n+3) + 2	High byte of the 2nd ADC conversion result of the mth channel enabled.
	(m-1)(2n+3) + 3	Low byte of the 2nd ADC conversion result of the mth

28.4 Data exchange between SPI and memory (SPI_DMA)

28.4.1 SPI_DMA Configuration Register (DMA_SPI_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG	7EFA20H	SPIIE	ACT_TX	ACT_RX	-	SPIIP[1:0]		SPIPTY[1:0]	

SPIIE: SPI_DMA interrupt enable control bit

0: Disable SPI_DMA interrupt

1: Allow SPI_DMA interrupt

ACT_TX: SPI_DMA transmit data control bit

0: Disable SPI_DMA to send data. In host mode, SPI only sends clock to SCLK port, but does not read data from XRAM or send data to MOSI port; in slave mode, SPI does not read data from XRAM or send data to MISO port.

1: Allow SPI_DMA to send data. In host 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 host mode, SPI only sends clock to SCLK port, but does not read data from MISO port or write data to XRAM; in slave mode, SPI does not read data from MOSI port or write data to XRAM.

1: Allow SPI_DMA to receive data. In host mode, SPI sends clock to SCLK port and also reads data from MISO port and writes data to XRAM; in slave mode, SPI reads data from MOSI port and writes data to XRAM.

SPIIP[1:0]: SPI_DMA interrupt priority control bits

SPIIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

SPIPTY[1:0]: SPI_DMA data bus access priority control bits

SPIPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.4.2 SPI_DMA Control Register (DMA_SPI_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CR	7EFA21H	ENSPI	TRIG_M	TRIG_S	-	-	-	-	CLRFIFO

ENSPI: SPI_DMA function enable control bit

0: Disable SPI_DMA function

1: Allow SPI_DMA function

TRIG_M: SPI_DMA host mode trigger control bit

0: Write 0 Invalid

1: Write 1 to start SPI_DMA host mode operation.

TRIG_S: SPI_DMA slave mode trigger control bit

0: Write 0 Invalid

1: Write 1 to start SPI_DMA slave mode operation.

CLRFIFO: clear SPI_DMA receive FIFO control bit

0: Write 0 Invalid

1: Clear the SPI_DMA built-in FIFO before starting SPI_DMA operation.

28.4.3 SPI_DMA Status Register (DMA_SPI_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_STA	7EFA22H	-	-	-	-	-	TXOVW	RXLOSS	SPIIF

SPIIFSPI_DMA interrupt request flag bit, when the SPI_DMA data exchange is completed, the hardware will automatically set SPIIF to 1,ifenableSPI_DMA

The interrupt will enter the interrupt service routine. Flag bits need to be cleared by software

RXLOSS: SPI_DMA receive data discard flag bit, during SPI_DMA operation, when the XRAM bus is too busy to empty the SPI_DMA receive FIFO, resulting in the SPI_DMA receive data automatically discarded, the hardware hardware automatically sets RXLOSS to 1. The flag bit needs to be cleared by software.

TXOVW: SPI_DMA Data Overwrite Flag Bit. when SPI_DMA is in the process of data transfer, and SPI data transfer is triggered again by SPI Write SPDAT Register in Host Mode, it will result in data transfer failure, and then the hardware hardware will automatically set TXOVW to 1. Flag Bit needs to be cleared by software

28.4.4 SPI_DMA Transmit Total Byte Register (DMA_SPI_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_AMT	7EFA23H	AMT [7:0]							
DMA_SPI_AMTH	7EFA84H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.4.5 SPI_DMA transfer completion byte register (DMA_SPI_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_DONE	7EFA24H	DONE[7:0]							
DMA_SPI_DONEH	7EFA85H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been transferred.

28.4.6 SPI_DMA Transmit Address Register (DMA_SPI_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_TXAH	7EFA25H	ADDR[15:8]							
DMA_SPI_TXAL	7EFA26H	ADDR[7:0]							

DMA_SPI_TXA: Sets the source address for data transfer. The data will be read from this address

when SPI_DMA operation is performed.

28.4.7 SPI_DMA Receive Address Register (DMA_SPI_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_RXAH	7EFA27H	ADDR[15:8]							
DMA_SPI_RXAL	7EFA28H	ADDR[7:0]							

DMA_SPI_RXA: Sets the destination address for data transfer. Data will be written from this address when SPI_DMA operation is performed.

28.4.8 SPI_DMA Configuration Register 2 (DMA_SPI_CFG2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG2	7EFA29H	-	-	-	-	-	WRPSS	SSS[1:0]	

WRPSS: Enable SS pin control bit during SPI_DMA process

0: No automatic control of SS pin during SPI_DMA transfer

1: SPI_DMA automatically pulls down the SS pin during transmission, and automatically restores the original state after the transmission is complete

SSS[1:0]: automatic control of the SS selection bit during SPI_DMA process

SSS[1:0]	SS Leg
00	P1.2/P5.4[1]
01	P2.2
10	P7.4
11	P3.5

[1] ：有 P1.2 port, this function is on theP1.2 port, and for models without theP1.2 port, this function is on the P5.4 口上

28.5 serial port 1 Data exchange with the memory (UR1T_DMA. UR1R_DMA)

28.5.1 UR1T_DMA Configuration Register (DMA_UR1T_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CFG	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]		UR1TPTY[1:0]	

UR1TIE: UR1T_DMA interrupt enable control bit

0: Disable UR1T_DMA interrupt

1: Allow UR1T_DMA interrupts

UR1TIP[1:0]: UR1T_DMA interrupt priority control bits

UR1TIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

UR1TPTY[1:0]: UR1T_DMA data bus access priority control bits

UR1TPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.5.2 UR1T_DMA Control Register (DMA_UR1T_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CR	7EFA31H	ENUR1T	TRIG	-	-	-	-	-	-

ENUR1T: UR1T_DMA function enable control bit

0: Disable UR1T_DMA function

1: Allow UR1T_DMA function

TRIG: UR1T_DMA Serial port 1 transmit trigger control bit

0: Write 0 Invalid

1: Write 1 to start UR1T_DMA auto send data

28.5.3 UR1T_DMA Status Register (DMA_UR1T_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_STA	7EFA32H	-	-	-	-	-	TXOVW	-	UR1TIF

UR1TIF: UR1T_DMA interrupt request flag bit, when the UR1T_DMA data transmission is completed, the hardware will automatically set UR1TIF to 1, if the UR1T_DMA interrupt is enabled, then enter the interrupt service programme. If UR1T_DMA interrupt is enabled,

the interrupt service procedure will be started.

TXOVW: UR1T_DMA data overwrite flag bit. when UR1T_DMA is in the process of data transmission and the serial port write SBUF register triggers the serial port to send data again, it will lead to the data transmission failure, at this time, the hardware hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

28.5.4 UR1T_DMA Transmit Total Byte Register (DMA_UR1T_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_AMT	7EFA33H	AMT [7:0]							
DMA_UR1T_AMTH	7EFA88H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.5.5 UR1T_DMA transfer completion byte register (DMA_UR1T_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_DONE	7EFA34H	DONE[7:0]							
DMA_UR1T_DONEH	7EFA89H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been sent.

28.5.6 UR1T_DMA Transmit Address Register (DMA_UR1T_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_TXAH	7EFA35H	ADDR[15:8]							
DMA_UR1T_TXAL	7EFA36H	ADDR[7:0]							

DMA_UR1T_TXA: Sets the source address from which data is automatically sent. The data will be read from this address when UR1T_DMA operation is performed.

28.5.7 UR1R_DMA Configuration Register (DMA_UR1R_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CFG	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]		UR1RPTY[1:0]	

UR1RIE: UR1R_DMA interrupt enable control bit

0: Disable UR1R_DMA interrupt

1: UR1R_DMA interrupt allowed

UR1RIP[1:0]: UR1R_DMA interrupt priority control bits

UR1RIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)

UR1RPTY[1:0]: UR1R_DMA data bus access priority control bits

UR1RPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.5.8 UR1R_DMA Control Register (DMA_UR1R_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_CR	7EFA39H	ENUR1R	-	TRIG	-	-	-	-	CLRFIFO

ENUR1R: UR1R_DMA function enable control bit

0: Disable UR1R_DMA function

1: Allow UR1R_DMA function

TRIG: UR1R_DMA Serial port 1 receive trigger control bit

0: Write 0 Invalid

1: Write 1 Start UR1R_DMA to receive data automatically

CLRFIFO: clear UR1R_DMA receive FIFO control bit

0: Write 0 Invalid

1: Clear the UR1R_DMA built-in FIFO before starting UR1R_DMA operation.

28.5.9 UR1R_DMA Status Register (DMA_UR1R_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_STA	7EFA3AH	-	-	-	-	-	-	RXLOSS	UR1RIF

UR1RIF: UR1R_DMA interrupt request flag bit, when UR1R_DMA receive data is completed, the hardware will automatically set UR1RIF to 1, if enable UR1R_DMA interrupt will enter the interrupt service programme. If UR1R_DMA interrupt is enabled, the interrupt service procedure will be started.

RXLOSS: UR1R_DMA receive data discard flag bit, during UR1R_DMA operation, when the XRAM bus is too busy to empty the UR1R_DMA receive FIFO, resulting in the data received by UR1R_DMA being automatically discarded, the hardware will automatically set RXLOSS to 1. Flag bit needs to be cleared by software

28.5.10 UR1R_DMA Transmit Total Byte Register (DMA_UR1R_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_AMT	7EFA3BH	AMT [7:0]							
DMA_UR1R_AMTH	7EFA8AH	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.5.11 UR1R_DMA Transfer completion byte register (DMA_UR1R_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_DONE	7EFA3CH	DONE[7:0]							
DMA_UR1R_DONEH	7EFA8BH	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been received.

28.5.12 UR1R_DMA Receive Address Register (DMA_UR1R_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_RXAH	7EFA3DH	ADDR[15:8]							
DMA_UR1R_RXAL	7EFA3EH	ADDR[7:0]							

DMA_UR1R_RXA: Sets the destination address for receiving data automatically. Data will be written from this address when UR1R_DMA operation is performed.

28.6 serial port 2 Data exchange with the memory (UR2T_DMA. UR2R_DMA)

28.6.1 UR2T_DMA Configuration Register (DMA_UR2T_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CFG	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]		UR2TPTY[1:0]	

UR2TIE: UR2T_DMA interrupt enable control bit

0: Disable UR2T_DMA interrupt

1: Allow UR2T_DMA interrupts

UR2TIP[1:0]: UR2T_DMA interrupt priority control bits

UR2TIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

UR2TPTY[1:0]: UR2T_DMA data bus access priority control bits

UR2TPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.6.2 UR2T_DMA Control Register (DMA_UR2T_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CR	7EFA41H	ENUR2T	TRIG	-	-	-	-	-	-

ENUR2T: UR2T_DMA function enable control bit

0: Disable UR2T_DMA function

1: Allow UR2T_DMA function

TRIG: UR2T_DMA Serial port 1 transmit trigger control bit

0: Write 0 Invalid

1: Write 1 to start UR2T_DMA sending data automatically

28.6.3 UR2T_DMA Status Register (DMA_UR2T_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_STA	7EFA42H	-	-	-	-	-	TXOVW	-	UR2TIF

UR2TIF: UR2T_DMA interrupt request flag bit, when the UR2T_DMA data transmission is completed, the hardware will automatically set UR2TIF to 1, if the UR2T_DMA interrupt is enabled, then enter the interrupt service programme. If UR2T_DMA interrupt is enabled,

the interrupt service procedure will be started.

TXOVW: UR2T_DMA data overwrite flag bit. when UR2T_DMA is in the process of data transmission and the serial port writes the S2BUF register to trigger the serial port to send data again, it will lead to the data transmission failure, and at this time, the hardware hardware will automatically set the TXOVW to 1. The flag bit needs to be cleared by the software.

28.6.4 UR2T_DMA Transmit Total Byte Register (DMA_UR2T_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_AMT	7EFA43H	AMT [7:0]							
DMA_UR2T_AMTH	7EFA8CH	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written to the data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.6.5 UR2T_DMA transfer completion byte register (DMA_UR2T_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_DONE	7EFA44H	DONE[7:0]							
DMA_UR2T_DONEH	7EFA8DH	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been sent.

28.6.6 UR2T_DMA Transmit Address Register (DMA_UR2T_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_TXAH	7EFA45H	ADDR[15:8]							
DMA_UR2T_TXAL	7EFA46H	ADDR[7:0]							

DMA_UR2T_TXA: Sets the source address from which data is automatically sent. The data will be read from this address when UR2T_DMA operation is performed.

28.6.7 UR2R_DMA Configuration Register (DMA_UR2R_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CFG	7EFA48H	UR2RIE	-	-	-	UR2RIP[1:0]		UR2RPTY[1:0]	

UR2RIE: UR2R_DMA interrupt enable control bit

0: Disable UR2R_DMA interrupt

1: UR2R_DMA interrupt allowed

UR2RIP[1:0]: UR2R_DMA interrupt priority control bits

UR2RIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)

UR2RPTY[1:0]: UR2R_DMA data bus access priority control bits

UR2RPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.6.8 UR2R_DMA Control Register (DMA_UR2R_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_CR	7EFA49H	ENUR2R	-	TRIG	-	-	-	-	CLRFIFO

ENUR2R: UR2R_DMA function enable control bit

0: Disable UR2R_DMA function

1: Allow UR2R_DMA function

TRIG: UR2R_DMA Serial port 1 receive trigger control bit

0: Write 0 Invalid

1: Write 1 to start UR2R_DMA receiving data automatically

CLRFIFO: clear UR2R_DMA receive FIFO control bit

0: Write 0 Invalid

1: Clear the UR2R_DMA built-in FIFO before starting UR2R_DMA operation.

28.6.9 UR2R_DMA Status Register (DMA_UR2R_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_STA	7EFA4AH	-	-	-	-	-	-	RXLOSS	UR2RIF

UR2RIF: UR2R_DMA interrupt request flag bit, when UR2R_DMA receive data is completed, the hardware will automatically set UR2RIF to 1, if enable UR2R_DMA interrupt, then enter the interrupt service programme. If UR2R_DMA interrupt is enabled, it will enter interrupt service procedure.

RXLOSS: UR2R_DMA receive data discard flag bit, during UR2R_DMA operation, when the XRAM bus is too busy to empty the UR2R_DMA receive FIFO, resulting in the data received by UR2R_DMA being automatically discarded, the hardware will automatically set RXLOSS to 1. Flag needs to be cleared by software

28.6.10 UR2R_DMA Transmit Total Byte Register (DMA_UR2R_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_AMT	7EFA4BH	AMT [7:0]							
DMA_UR2R_AMTH	7EFA8EH	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.6.11 UR2R_DMA Transmission completion byte register (DMA_UR2R_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_DONE	7EFA4CH	DONE[7:0]							
dma_ur2r_doneh	7EFA8FH	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been received.

28.6.12 UR2R_DMA Receive Address Register (DMA_UR2R_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_RXAH	7EFA4DH	ADDR[15:8]							
DMA_UR2R_RXAL	7EFA4EH	ADDR[7:0]							

DMA_UR2R_RXA: Sets the destination address for receiving data automatically. Data will be written from this address when UR2R_DMA operation is performed.

28.7 serial port 3 Data exchange with the memory (UR3T_DMA, UR3T_DMA, UR3T_DMA, UR3T_DMA, UR3T_DMA). UR3R_DMA)

28.7.1 UR3T_DMA Configuration Register (DMA_UR3T_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CFG	7EFA50H	UR3TIE	-	-	-	UR3TIP[1:0]		UR3TPTY[1:0]	

UR3TIE: UR3T_DMA interrupt enable control bit

0: Disable UR3T_DMA interrupt

1: Allow UR3T_DMA interrupts

UR3TIP[1:0]: UR3T_DMA interrupt priority control bits

UR3TIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

UR3TPTY[1:0]: UR3T_DMA data bus access priority control bits

UR3TPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.7.2 UR3T_DMA Control Register (DMA_UR3T_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_CR	7EFA51H	ENUR3T	TRIG	-	-	-	-	-	-

ENUR3T: UR3T_DMA function enable control bit

0: Disable UR3T_DMA function

1: Allow UR3T_DMA function

TRIG: UR3T_DMA Serial port 1 transmit trigger control bit

0: Write 0 Invalid

1: Write 1 to start UR3T_DMA auto send data

28.7.3 UR3T_DMA Status Register (DMA_UR3T_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_STA	7EFA52H	-	-	-	-	-	TXOVW	-	UR3TIF

UR3TIF: UR3T_DMA interrupt request flag bit, when the UR3T_DMA data transmission is

completed, the hardware will automatically set UR3TIF to 1, if the UR3T_DMA interrupt is enabled, then enter the interrupt service programme. If UR3T_DMA interrupt is enabled, the interrupt service procedure will be started.

TXOVW: UR3T_DMA data overwrite flag bit. when UR3T_DMA is in the process of data transmission and the serial port writes the S3BUF register to trigger the serial port to send data again, it will lead to data transmission failure, and then the hardware hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software.

28.7.4 UR3T_DMA Transmit Total Byte Register (DMA_UR3T_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_AMT	7EFA53H	AMT [7:0]							
DMA_UR3T_AMTH	7EFA90H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.7.5 UR3T_DMA transfer completion byte register (DMA_UR3T_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_DONE	7EFA54H	DONE[7:0]							
DMA_UR3T_DONEH	7EFA91H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been sent.

28.7.6 UR3T_DMA Transmit Address Register (DMA_UR3T_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_TXAH	7EFA55H	ADDR[15:8]							
DMA_UR3T_TXAL	7EFA56H	ADDR[7:0]							

DMA_UR3T_TXA: Sets the source address from which data is automatically sent. The data will be read from this address when UR3T_DMA operation is performed.

28.7.7 UR3R_DMA Configuration Register (DMA_UR3R_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CFG	7EFA58H	UR3RIE	-	-	-	UR3RIP[1:0]		UR3RPTY[1:0]	

UR3RIE: UR3R_DMA interrupt enable control bit

0: Disable UR3R_DMA interrupt

1: UR3R_DMA interrupt allowed

UR3RIP[1:0]: UR3R_DMA interrupt priority control bits

UR3RIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)

UR3RPTY[1:0]: UR3R_DMA data bus access priority control bits

UR3RPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.7.8 UR3R_DMA Control Register (DMA_UR3R_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_CR	7EFA59H	ENUR3R	-	TRIG	-	-	-	-	CLRFIFO

ENUR3R: UR3R_DMA function enable control bit

0: Disable UR3R_DMA function

1: Allow UR3R_DMA function

TRIG: UR3R_DMA Serial port 1 receive trigger control bit

0: Write 0 Invalid

1: Write 1 to start UR3R_DMA receiving data automatically

CLRFIFO: clear UR3R_DMA receive FIFO control bit

0: Write 0 Invalid

1: Clear the UR3R_DMA built-in FIFO before starting UR3R_DMA operation.

28.7.9 UR3R_DMA Status Register (DMA_UR3R_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_STA	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR3RIF

UR3RIF: UR3R_DMA interrupt request flag bit, when UR3R_DMA receive data is completed, the hardware will automatically set UR3RIF to 1, if enable UR3R_DMA interrupt, then enter the interrupt service programme. If UR3R_DMA interrupt is enabled, it will enter interrupt service procedure. The flag bit needs to be cleared by software

RXLOSS: UR3R_DMA receive data discard flag bit, during UR3R_DMA operation, when the XRAM bus is too busy to empty the UR3R_DMA receive FIFO, resulting in the data received by UR3R_DMA being automatically discarded, the hardware will automatically set RXLOSS to 1. Flag needs to be cleared by software

28.7.10 UR3R_DMA Transmit Total Byte Register (DMA_UR3R_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_AMT	7EFA5BH	AMT [7:0]							
DMA_UR3R_AMTH	7EFA92H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.7.11 UR3R_DMA Transfer completion byte register (DMA_UR3R_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_DONE	7EFA5CH	DONE[7:0]							
dma_ur3r_doneh	7EFA93H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been received.

28.7.12 UR3R_DMA Receive Address Register (DMA_UR3R_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_RXAH	7EFA5DH	ADDR[15:8]							
dma_ur3r_rxal	7EFA5EH	ADDR[7:0]							

DMA_UR3R_RXA: Sets the destination address for receiving data automatically. Data will be written from this address when UR3R_DMA operation is performed.

28.8 serial port 4 Data exchange with the memory (UR4T_DMA. UR4R_DMA)

28.8.1 UR4T_DMA Configuration Register (DMA_UR4T_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CFG	7EFA50H	UR4TIE	-	-	-	UR4TIP[1:0]		UR4TPTY[1:0]	

UR4TIE: UR4T_DMA interrupt enable control bit

0: Disable UR4T_DMA interrupt

1: Allow UR4T_DMA interrupts

UR4TIP[1:0]: UR4T_DMA interrupt priority control bits

UR4TIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

UR4TPTY[1:0]: UR4T_DMA data bus access priority control bits

UR4TPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.8.2 UR4T_DMA Control Register (DMA_UR4T_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_CR	7EFA51H	ENUR4T	TRIG	-	-	-	-	-	-

ENUR4T: UR4T_DMA function enable control bit

0: Disable UR4T_DMA function

1: Allow UR4T_DMA function

TRIG: UR4T_DMA Serial port 1 transmit trigger control bit

0: Write 0 Invalid

1: Write 1 to start UR4T_DMA auto send data

28.8.3 UR4T_DMA Status Register (DMA_UR4T_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_STA	7EFA62H	-	-	-	-	-	TXOVW	-	UR4TIF

UR4TIF: UR4T_DMA interrupt request flag bit, when the UR4T_DMA data transmission is completed, the hardware will automatically set UR4TIF to 1, if the UR4T_DMA interrupt is enabled, then enter the interrupt service programme. If UR4T_DMA interrupt is enabled,

the interrupt service procedure will be started.

TXOVW: UR4T_DMA data overwrite flag bit. when UR4T_DMA is in the process of data transmission and the serial port writes the S4BUF register to trigger the serial port to send data again, it will lead to data transmission failure, and at this time, the hardware hardware will automatically set the TXOVW to 1. The flag bit needs to be cleared by software.

28.8.4 UR4T_DMA Transmit Total Byte Register (DMA_UR4T_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_AMT	7EFA53H	AMT [7:0]							
DMA_UR4T_AMTH	7EFA94H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.8.5 UR4T_DMA transfer completion byte register (DMA_UR4T_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_DONE	7EFA54H	DONE[7:0]							
DMA_UR4T_DONEH	7EFA95H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been sent.

28.8.6 UR4T_DMA Transmit Address Register (DMA_UR4T_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_TXAH	7EFA55H	ADDR[15:8]							
DMA_UR4T_TXAL	7EFA56H	ADDR[7:0]							

DMA_UR4T_TXA: Sets the source address from which data is automatically sent. The data will be read from this address when UR4T_DMA operation is performed.

28.8.7 UR4R_DMA Configuration Register (DMA_UR4R_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CFG	7EFA58H	UR4RIE	-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	

UR4RIE: UR4R_DMA interrupt enable control bit

0: Disable UR4R_DMA interrupt

1: UR4R_DMA interrupt allowed

UR4RIP[1:0]: UR4R_DMA interrupt priority control bits

UR4RIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)

UR4RPTY[1:0]: UR4R_DMA data bus access priority control bits

UR4RPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.8.8 UR4R_DMA Control Register (DMA_UR4R_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_CR	7EFA59H	ENUR4R	-	TRIG	-	-	-	-	CLRFIFO

ENUR4R: UR4R_DMA function enable control bit

0: Disable UR4R_DMA function

1: Allow UR4R_DMA function

TRIG: UR4R_DMA Serial port 1 receive trigger control bit

0: Write 0 Invalid

1: Write 1 to start UR4R_DMA auto receive data

CLRFIFO: clear UR4R_DMA receive FIFO control bit

0: Write 0 Invalid

1: Clear the UR4R_DMA built-in FIFO before starting UR4R_DMA operation.

28.8.9 UR4R_DMA Status Register (DMA_UR4R_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_STA	7EFA5AH	-	-	-	-	-	-	RXLOSS	UR4RIF

UR4RIF: UR4R_DMA interrupt request flag bit, when UR4R_DMA receive data is completed, the hardware will automatically set UR4RIF to 1, if enable UR4R_DMA interrupt will enter the interrupt service programme. If UR4R_DMA interrupt is enabled, the interrupt service procedure will be started.

RXLOSS: UR4R_DMA receive data discard flag bit, during UR4R_DMA operation, when the XRAM bus is too busy to empty the UR4R_DMA receive FIFO, resulting in the data received by UR4R_DMA being automatically discarded, the hardware will automatically set RXLOSS to 1. Flag needs to be cleared by software

28.8.10 UR4R_DMA Transmit Total Byte Register (DMA_UR4R_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_AMT	7EFA5BH	AMT [7:0]							
DMA_UR4R_AMTH	7EFA96H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.8.11 UR4R_DMA transfer completion byte register (DMA_UR4R_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_DONE	7EFA5CH	DONE[7:0]							
DMA_UR4R_DONEH	7EFA97H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been received.

28.8.12 UR4R_DMA Receive Address Register (DMA_UR4R_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_RXAH	7EFA5DH	ADDR[15:8]							
DMA_UR4R_RXAL	7EFA5EH	ADDR[7:0]							

DMA_UR4R_RXA: Sets the destination address for receiving data automatically. Data will be written from this address when UR4R_DMA operation is performed.

28.9 Data read/write between LCM and memory (LCM_DMA)

28.9.1 LCM_DMA Configuration Register (DMA_LCM_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CFG	7EFA70H	LCMIE	ACT_TX	ACT_RX	-	LCMIP[1:0]		LCMPTY[1:0]	

LCMIE: LCM_DMA interrupt enable control bit

0: Disable LCM_DMA interrupt

1: Allow LCM_DMA interrupt

LCMIP[1:0]: LCM_DMA interrupt priority control bits

LCMIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

LCMPTY[1:0]: LCM_DMA data bus access priority control bits

LCMPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.9.2 LCM_DMA Control Register (DMA_LCM_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_CR	7EFA71H	ENLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	CLRFIFO

ENLCM: LCM_DMA function enable control bit

0: Disable LCM_DMA function

1: Allow LCM_DMA function

TRIGWC: LCM_DMA transmit command mode trigger control bit

0: Write 0 Invalid

1: Write 1 to start the LCM_DMA transmit command mode operation.

TRIGWD: LCM_DMA transmit data mode trigger control bit

0: Write 0 Invalid

1: Write 1 to start the LCM_DMA transmit data mode operation.

TRIGRC: LCM_DMA read command mode trigger control bit

0: Write 0 Invalid

1: Write 1 Start LCM_DMA Read Command Mode Operation

TRIGRD: LCM_DMA read data mode trigger control bit

0: Write 0 Invalid

1: Write 1 to start LCM_DMA read data mode operation

28.9.3 LCM_DMA Status Register (DMA_LCM_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_STA	7EFA72H	-	-	-	-	-	-	TXOVW	LCMIF

LCMIF: LCM_DMA interrupt request flag bit, when the LCM_DMA data exchange is completed, the hardware will automatically set LCMIF to 1, if the LCM_DMA interrupt is enabled, then enter the interrupt service procedure. If LCM_DMA interrupt is enabled, the interrupt service procedure will start. The flag bit needs to be cleared by software.

TXOVW: LCM_DMA Data Overwrite Flag Bit LCM_DMA is in the process of data transfer, when LCMIF writes the LCMIFDATL and LCMIFDATH registers, it will cause the data transfer to fail, and then the hardware hardware will automatically set TXOVW to 1. Flag Bit Needs to be Cleared by Software

28.9.4 LCM_DMA Transmit Total Byte Register (DMA_LCM_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_AMT	7EFA73H	AMT [7:0]							
DMA_LCM_AMTH	7EFA86H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.9.5 LCM_DMA Transmit Complete Byte Register (DMA_LCM_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_DONE	7EFA74H	DONE[7:0]							
DMA_LCM_DONEH	7EFA87H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been transferred.

28.9.6 LCM_DMA Transmit Address Register (DMA_LCM_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_TXAH	7EFA75H	ADDR[15:8]							
DMA_LCM_TXAL	7EFA76H	ADDR[7:0]							

DMA_LCM_TXA: Sets the source address for data transfer. The data will be read from this address when LCM_DMA operation is performed.

28.9.7 LCM_DMA Receive Address Register (DMA_LCM_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_LCM_RXAH	7EFA77H	ADDR[15:8]							

DMA_LCM_RXAL	7EFA78H	ADDR[7:0]

DMA_LCM_RXA: Sets the destination address for data transfer. Data will be written from this address when LCM_DMA operation is performed.

28.10 Data exchange between I2C and memory (I2CT_DMA, I2CR_DMA)

28.10.1 I2CT_DMA Configuration Register (DMA_I2CT_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_CFG	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0]	

I2CTIE: I2CT_DMA interrupt enable control bit

0: Disable I2CT_DMA interrupts

1: I2CT_DMA interrupt allowed

I2CTIP[1:0]: I2CT_DMA interrupt priority control bits

I2CTIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

I2CTPTY[1:0]: I2CT_DMA data bus access priority control bits

I2CTPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.10.2 I2CT_DMA Control Register (DMA_I2CT_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_CR	7EFA99H	ENI2CT	TRIG	-	-	-	-	-	-

ENI2CT: I2CT_DMA function enable control bit

0: Disable I2CT_DMA function

1: Allow I2CT_DMA function

TRIG: I2CT_DMA Serial port 1 transmit trigger control bit

0: Write 0 Invalid

1: Write 1 to start I2CT_DMA auto send data

28.10.3 I2CT_DMA Status Register (DMA_I2CT_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_STA	7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF

I2CTIF: I2CT_DMA interrupt request flag bit, when the I2CT_DMA data transmission is completed, the hardware will automatically set I2CTIF to 1, if enable I2CT_DMA interrupt, then enter the interrupt service procedure. If I2CT_DMA interrupt is enabled, the interrupt service procedure will be started.

TXOVW: I2CT_DMA Data Overwrite Flag Bit. I2CT_DMA is in the process of data transfer, write I2C

data register I2CTXD

If the data transfer fails, the hardware automatically sets TXOVW to 1. Flags need to be cleared by software.

28.10.4 I2CT_DMA Transmit Total Byte Register (DMA_I2CT_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_AMT	7EFA9BH	AMT [7:0]							
DMA_I2CT_AMTH	7EFAA8H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.10.5 I2CT_DMA Transfer Completion Byte Register (DMA_I2CT_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_DONE	7EFA9CH	DONE[7:0]							
DMA_I2CT_DONEH	7EFAA9H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been sent.

28.10.6 I2CT_DMA Transmit Address Register (DMA_I2CT_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_TXAH	7EFA9DH	ADDR[15:8]							
DMA_I2CT_TXAL	7EFA9EH	ADDR[7:0]							

DMA_I2CT_TXA: Sets the source address from which data is automatically sent. The data will be read from this address when I2CT_DMA operation is performed.

28.10.7 I2CR_DMA Configuration Register (DMA_I2CR_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_CFG	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]	

I2CRIE: I2CR_DMA Interrupt Enable Control Bit

0: Disable I2CR_DMA interrupt

1: I2CR_DMA interrupt allowed

I2CRIP[1:0]: I2CR_DMA interrupt priority control bits

I2CRIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

I2CRPTY[1:0]: I2CR_DMA data bus access priority control bits

I2CRPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.10.8 I2CR_DMA Control Register (DMA_I2CR_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_CR	7EFAA1H	ENI2CR	TRIG	-	-	-	-	-	CLRFIFO

ENI2CR: I2CR_DMA function enable control bit

0: Disable I2CR_DMA function

1: Allow I2CR_DMA function

TRIG: I2CR_DMA Serial Port 1 Receive Trigger Control Bit

0: Write 0 Invalid

1: Write 1 to start I2CR_DMA auto receive data

CLRFIFO: Clears the I2CR_DMA receive FIFO control bit.

0: Write 0 Invalid

1: Clear the I2CR_DMA built-in FIFO before starting I2CR_DMA operation.

28.10.9 I2CR_DMA Status Register (DMA_I2CR_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_STA	7EFAA2H	-	-	-	-	-	-	RXLOSS	I2CRIF

I2CRIF: I2CR_DMA interrupt request flag bit, when the I2CR_DMA receive data is completed, the hardware will automatically set I2CRIF to 1, if enable I2CR_DMA interrupt, then enter the interrupt service procedure. If I2CR_DMA interrupt is enabled, the interrupt service procedure will be started.

RXLOSS: I2CR_DMA receive data discard flag bit, during I2CR_DMA operation, when the XRAM bus is too busy to empty the I2CR_DMA receive FIFO, resulting in the I2CR_DMA receive data automatically discarded, the hardware hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software

28.10.10 I2CR_DMA Transmit Total Byte Register (DMA_I2CR_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_AMT	7EFAA3H	AMT [7:0]							
DMA_I2CR_AMTH	7EFAAAH	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.10.11 I2CR_DMA Transfer Completion Byte Register (DMA_I2CR_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_DONE	7EFAA4H	DONE[7:0]							
DMA_I2CR_DONEH	7EFAABH	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been received.

28.10.12 I2CR_DMA Receive Address Register (DMA_I2CR_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_RXAH	7EFAA5H	ADDR[15:8]							
DMA_I2CR_RXAL	7EFAA6H	ADDR[7:0]							

DMA_I2CR_RXA: Sets the destination address for receiving data automatically. Data will be written from this address when I2CR_DMA operation is performed.

28.10.13 I2C_DMA Control Register (DMA_I2C_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2C_CR	7EFAADH	RDSEL	-	-	-	-	ACKERR	ERRIE	BMMEN

RDSEL: I2C_DMA_ST register read function selection

ACKERR: ACK error

0: The answer received after sending data is ACK

1: The answer received after sending data is NAK (software clearing required)

ERRIE: ACKERR Interrupt enable control bit

0: Disable ACKERR interrupt

1: ACKERR interrupt allowed (ACKERR interrupt entry address is I2C interrupt entry address FF:00C3H)

BMMEN: DMA function enable bit for I2C

0: Disable I2C_DMA function

1: Allow I2C_DMA function

28.10.14 I2C_DMA Status Register (DMA_I2C_ST)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2C_ST1	7EFAAEH	COUNT[7:0]							
DMA_I2C_ST2	7EFAAFH	COUNT[15:8]							

COUNT: I2C_DMA transfer byte control

Write register: set the number of bytes to be transferred by I2C_DMA.

Read register: COUNT is the number of bytes to be transferred when RDSEL = 0.

When RDSEL = 1, COUNT is the number of bytes that have been transferred.

28.11 Data exchange between I2S and memory (I2ST_DMA, I2SR_DMA)

28.11.1 I2ST_DMA Configuration Register (DMA_I2ST_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_CFG	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0]	

I2STIE: I2ST_DMA interrupt enable control bit

0: Disable I2ST_DMA interrupt

1: I2ST_DMA interrupt allowed

I2STIP[1:0]: I2ST_DMA interrupt priority control bits

I2STIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

I2STPTY[1:0]: I2S_DMA data bus access priority control bits

I2STPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.11.2 I2ST_DMA Control Register (DMA_I2ST_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_CR	7EFAB1H	ENI2ST	TRIG	-	-	-	-	-	-

ENI2ST: I2ST_DMA function enable control bit

0: Disable I2ST_DMA function

1: Allow I2ST_DMA function

TRIG: I2ST_DMA Serial port 1 transmit trigger control bit

0: Write 0 Invalid

1: Write 1 to start I2ST_DMA auto send data

28.11.3 I2ST_DMA Status Register (DMA_I2ST_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_STA	7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF

I2STIF: I2ST_DMA interrupt request flag bit, when the I2ST_DMA data transmission is completed, the hardware will automatically set I2STIF to 1, if enable I2ST_DMA interrupt, then enter the interrupt service procedure. If I2ST_DMA interrupt is enabled, the interrupt service procedure will be started.

TXOVW: I2ST_DMA Data Overwrite Flag Bit. I2ST_DMA is in the process of data transfer, write I2S

Data Register I2S_DRH

and I2S_DRL will result in a data transfer failure, in which case the hardware hardware automatically sets TXOVW to 1. Flags need to be cleared by software

28.11.4 I2ST_DMA Transmit Total Byte Register (DMA_I2ST_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_AMT	7EFAB3H	AMT [7:0]							
DMA_I2ST_AMTH	7EFAC0H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.11.5 I2ST_DMA transfer completion byte register (DMA_I2ST_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_DONE	7EFAB4H	DONE[7:0]							
DMA_I2ST_DONEH	7EFAC1H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been sent.

28.11.6 I2ST_DMA Transmit Address Register (DMA_I2ST_TXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2ST_TXAH	7EFAB5H	ADDR[15:8]							
DMA_I2ST_TXAL	7EFAB6H	ADDR[7:0]							

DMA_I2ST_TXA: Sets the source address from which data is automatically sent. The data will be read from this address when I2ST_DMA operation is performed.

28.11.7 I2SR_DMA Configuration Register (DMA_I2SR_CFG)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_CFG	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]	

I2SRIE: I2SR_DMA interrupt enable control bit

0: Disable I2SR_DMA interrupts

1: I2SR_DMA interrupt allowed

I2SRIP[1:0]: I2SR_DMA interrupt priority control bits

I2SRIP[1:0]	interrupt priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

I2SRPTY[1:0]: I2SR_DMA data bus access priority control bits

I2SRPTY [1:0]	bus priority
00	Lowest (0)
01	Lower level (1)
10	Higher (2)
11	Top level (3)

28.11.8 I2SR_DMA Control Register (DMA_I2SR_CR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_CR	7EFAB9H	ENI2SR	-	TRIG	-	-	-	-	CLRFIFO

ENI2SR: I2SR_DMA function enable control bit

0: Disable I2SR_DMA function

1: Allow I2SR_DMA function

TRIG: I2SR_DMA Serial Port 1 Receive Trigger Control Bit

0: Write 0 Invalid

1: Write 1 to start I2SR_DMA auto receive data

CLRFIFO: Clears the I2SR_DMA receive FIFO control bit.

0: Write 0 Invalid

1: Clear the I2SR_DMA built-in FIFO before starting I2SR_DMA operation.

28.11.9 I2SR_DMA Status Register (DMA_I2SR_STA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_STA	7EFABAH	-	-	-	-	-	-	RXLOSS	I2SRIF

I2SRIF: I2SR_DMA interrupt request flag bit, when I2SR_DMA receives data, the hardware will set I2SRIF to 1 automatically, and if I2SR_DMA interrupt is enabled, it will enter the interrupt service programme. If I2SR_DMA interrupt is enabled, the interrupt service procedure will be started.

RXLOSS: I2SR_DMA receive data discard flag bit, during I2SR_DMA operation, when the XRAM bus is too busy to empty the I2SR_DMA receive FIFO, resulting in the data received by I2SR_DMA being automatically discarded, the hardware will automatically set RXLOSS to 1. The flag bit needs to be cleared by software.

28.11.10 I2SR_DMA Transmit Total Byte Register (DMA_I2SR_AMT)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_AMT	7EFABBH	AMT [7:0]							
DMA_I2SR_AMTH	7EFAC2H	AMT [15:8]							

AMT[15:0]: set the number of bytes to be read and written for data.

Note: The actual number of bytes read and written is (AMT+1), i.e. when AMT is set to 0, 1 byte is read and written, when AMT is set to 255, 256 bytes are read and written

28.11.11 I2SR_DMA Transmit Complete Byte Register (DMA_I2SR_DONE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_DONE	7EFABCH	DONE[7:0]							
DMA_I2SR_DONEH	7EFAC3H	DONE[15:8]							

DONE[15:0]: the current number of bytes that have been received.

28.11.12 I2SR_DMA Receive Address Register (DMA_I2SR_RXAx)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2SR_RXAH	7EFABDH	ADDR[15:8]							
DMA_I2SR_RXAL	7EFABEH	ADDR[7:0]							

DMA_I2SR_RXA: Sets the destination address for receiving data automatically. Data will be written from this address when I2SR_DMA operation is performed.

28.12 sample procedure

28.12.1 Serial 1 Interrupt Mode and PC Transceiver Test - DMA Receive Timeout Interrupt

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files

/*************  Function Description  **************
Serial 1 full-duplex interrupt mode send/receive communication programme. The PC sends data to the MCU, and the MCU automatically stores the received data in the DMA space. When the data received at one time fills up the set DMA space, the data in the storage space will be output through the DMA auto-send function of serial port 1. When no new data is received after the timeout, it means that a string of data has been received, the received content will be output and the DMA space will be cleared. When using a timer as a baud rate generator, it is recommended to use 1T mode (unless 12T is used for low baud rate) and select a clock frequency that can be divisible by the baud rate to improve accuracy.
When downloading, select the clock 22.1184MHz (user can modify the frequency).
*****************************************/

#include "stdio.h"

#define   MAIN_Fosc       22118400L                    //Define master clock (precisely calculated
#define   Baudrate1       115200L                      115200 baud rate)
#define   Timer0_Reload   (65536UL - (MAIN_Fosc / 1000))

#define   DMA_AMT_LEN     255                          //Set the total number of bytes transferred
                                                       (0~255): DMA_AMT_LEN+1

bit       B_1ms;                                       //1ms flag
bit       DMATxFlag;
bit       DMARxFlag;
bit       BusyFlag;
u8        Rx_cnt;
u8        RX1_TimeOut.

          u8xdata

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;

    ckcon = 0x00;                                   //Set external data bus speed to fastest
    wtst = 0x00.                                    //Set the parameter for waiting for the
                                                    programme code.
                                                    //Assign a value of 0 to set the CPU to
                                                    execute the programme as fast as possible.

    P0M1 = 0x00.    P0M0 = 0x00.                    //Set quasi-bidirectional port
    P1M1 = 0x00.    P1M0 = 0x00.                    //Set quasi-bidirectional port
    P2M1 = 0x00.    P2M0 = 0x00.                    //Set quasi-bidirectional port
    P3M1 = 0x00.    P3M0 = 0x00.                    //Set quasi-bidirectional port
    P4M1 = 0x00.    P4M0 = 0x00.                    //Set quasi-bidirectional port
    P5M1 = 0x00.    P5M0 = 0x00.                    //Set quasi-bidirectional port
    P6M1 = 0x00.    P6M0 = 0x00.                    //Set quasi-bidirectional port
    P7M1 = 0x00.    P7M0 = 0x00.                    //Set 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              //Timer0 interrupt enable
    TR0 = 1; //Tiner0 run                           //Tiner0 run

    UART1_config(1).                                //Use Timer1 for
    baud rate. DMA_Config().
    EA = 1;                     // Allow total interrupt

    printf("UART1 DMA Timeout Programme!\r\n");     //UART1 sends a string
    DMATxFlag = 0;
    DMARxFlag = 0;

    while (1)
    {
        if((DMATxFlag) && (DMARxFlag))              //Determine the transmit completion flag and receive
        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, //bit7 1: enable UART1_DMA.
                                                    //bit6 1: start UART1_DMA auto transmit

            DMARxFlag = 0;
            DMA_UR1R_CR = 0xa1;                     //bit7 1:使能 UART1_DMA,
                                                    //bit5 1:开始 UART1_DMA 自动接收,
                                                    //bit0 1: clear FIFOs
        }

        if(B_1ms)                                   /1ms to
        {
            B_1ms = 0;
            if(RX1_TimeOut > 0)                     //Timeout count
            {
```

```
if(--RX1_TimeOut == 0)
{
    DMA_UR1R_CR = 0x00;                 //turn off 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:使能 UART1_DMA,
                                                          //bit5 1:开始 UART1_DMA 自动接收,
                                                          //bit0 1: clear FIFOs
                }
            }
        }
    }
}

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 the total number of bytes transferred: n+1
    DMA_UR1T_TXA = DMABuffer.
    DMA_UR1T_CR = 0xc0.                                   //bit7 1: enable UART1_DMA, //bit7 1: enable UART1_DMA.
                                                          //bit6 1: start UART1_DMA auto transmit

    DMA_UR1R_CFG = 0x80;                                  //bit7 1:Enable Interrupt
    DMA_UR1R_STA = 0x00;
    DMA_UR1R_AMT = DMA_AMT_LEN;                           //Set the total number of bytes transferred: n+1
    DMA_UR1R_RXA = DMABuffer.
    DMA_UR1R_CR = 0xa1;                                   //bit7 1:使能 UART1_DMA,
                                                          //bit5 1:开始 UART1_DMA 自动接收,
                                                          //bit0 1: clear FIFOs
}

void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                                      //Timer stop
    AUXR &= ~(1<<3); //Timer2 Set As Timer                //Timer2 set As Timer
    AUXR |= (1<<2); //Timer2 set as 1T mode.              //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256.
    IE2 &= ~(1<<2).                                       //disable interrupt
    AUXR |= (1<<4); //Timer run enable.                   //Timer run enable
}

void UART1_config(u8 brt)                                 //Select baud rate.
                                                          //2: Use Timer2 for baud rate.
                                                          //Other values: Use Timer1 for baud rate.
{
    /********** baud rate using timer 2
    **********/ if(brt == 2)
    {
        AUXR |= 0x01.                                     //S1 BRT Use Timer2.
        SetTimer2Baudraye(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /********** baud rate usage timer 1
    **********/ else
    {
```

```
        TR1 = 0;
        AUXR &= ~0x01.                          //S1 BRT Use Timer1.
        AUXR |= (1<<6); //Timer1 set as 1T mode.  //Timer1 set as 1T mode
        TMOD &= ~(1<<6); //Timer1 Set As Timer    //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 interrupt
        INTCLKO &= ~0x02.                       //no clock output
        TR1 = 1;
    }
    /*********************************************/

    SCON = (SCON & 0x3f) | 0x40.                //UART1 mode.
                                                //0x00: Synchronous shift output, //0x00: Synchronous shift output, //0x00: Synchronous shift output, //0x00: Synchronous shift output, //0x00: Synchronous shift output, //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;                                     // Allow interrupt
    REN = 1;                                    // Allow to receive
    P_SW1 &= 0x3f.
    P_SW1 |= 0x00; //UART1 switch to.           //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 the new data is not received for 5ms, it is decided that a string of data has been received.   //If the new data is not received in 5ms, it will be decided that the 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)                    //Transmission complete
```

```
    {
        DMA_UR1T_STA &= ~0x01.
        DMATxFlag = 1;
```

```
    }
    if (DMA_UR1T_STA & 0x04)                    //Data overwrite
    {
        DMA_UR1T_STA &= ~0x04;
    }

    if (DMA_UR1R_STA & 0x01)                    //Reception complete
    {
        DMA_UR1R_STA &= ~0x01.
        DMARxFlag = 1;
    }
    if (DMA_UR1R_STA & 0x02)                    //Data discard
    {
        DMA_UR1R_STA &= ~0x02;
    }
}
```

// Document: ISR.ASM

//Interrupt number greater than 31 requires interrupt entry address remapping.

```
            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
```

28.12.2 Serial 1 Interrupt Mode and PC Transceiver Test - DMA Data Verification

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                     // see download software for header files

/*************  Function Description    **************
Serial 1 full-duplex interrupt mode send/receive communication programme. The PC sends data to the MCU, and the MCU automatically stores the received data into the DMA space. The last two bytes of the packet are used as parity bits, and the routine uses the crc16_ccitt algorithm to do the parity check. When the DMA space is full of the set size, the valid data is checked and compared with the last two parity bits. The data in the memory space is output through the DMA auto-send function of serial port 1. When using a timer as a 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 (the frequency can be changed by the user).
***/

```
#include "stdio.h"
```

```
#include "crc16.h"

#define   MAIN_Fosc     22118400L
#define   Baudrate1     115200L

#define   DMA_AMT_LEN   255

bit       DMATxFlag;
bit       DMARxFlag.

          u8xdata

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,
```

//Define master clock (precisely calculated *115200* baud rate)

//Set the total number of bytes transferred *(0~255)*: *DMA_AMT_LEN+1*

0x17DB, 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.

    ckcon = 0x00;                                   //Set external data bus speed to fastest
    wtst = 0x00.                                    //Set the parameters for the programme
                                                    code wait.
                                                    //Assign a value of 0 to set the CPU to
                                                    execute the programme as fast as possible.
    P0M1 = 0x00.    P0M0 = 0x00.                    //Set quasi-bidirectional port
    P1M1 = 0x00.    P1M0 = 0x00.                    //Set quasi-bidirectional port
    P2M1 = 0x00.    P2M0 = 0x00.                    //Set quasi-bidirectional port
    P3M1 = 0x00.    P3M0 = 0x00.                    //Set quasi-bidirectional port
    P4M1 = 0x00.    P4M0 = 0x00.                    //Set quasi-bidirectional port
    P5M1 = 0x00.    P5M0 = 0x00.                    //Set quasi-bidirectional port
    P6M1 = 0x00.    P6M0 = 0x00.                    //Set quasi-bidirectional port
    P7M1 = 0x00.    P7M0 = 0x00.                    //Set 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;                                         // Allow total 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, //bit7 1: enable UART1_DMA.
                                                    //bit6 1: start UART1_DMA auto transmit
                DMARxFlag = 0;
                DMA_UR1R_CR = 0xa1;                 //bit7 1:使能 UART1_DMA,
                                                    //bit5 1:开始 UART1_DMA 自动接收,
                                                    //bit0 1: clear FIFOs
            }
        }
}

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 the total number of bytes transferred: n+1
    DMA_UR1T_TXA = DMABuffer.
    DMA_UR1T_CR = 0xc0.                             //bit7 1: enable UART1_DMA, //bit7 1: enable UART1_DMA.
                                                    //bit6 1: start UART1_DMA auto transmit

    DMA_UR1R_CFG = 0x80;                            //bit7 1:Enable Interrupt
    DMA_UR1R_STA = 0x00;
    DMA_UR1R_AMT = DMA_AMT_LEN;                     //Set the total number of bytes transferred: n+1
    DMA_UR1R_RXA = DMABuffer.
    DMA_UR1R_CR = 0xa1;                             //bit7 1:使能 UART1_DMA,
                                                    //bit5 1: start UART1_DMA auto-reception, bit0 1: clear
                                                    FIFOs
}

void SetTimer2Baudraye(u16 dat)                     //Select baud rate.
                                                    //2: Use Timer2 for baud rate.
                                                    // Other values: Use Timer1 for baud rate.
{
    AUXR &= ~(1<<4);                                //Timer stop
    AUXR &= ~(1<<3); //Timer2 Set As Timer          //Timer2 set As Timer
    AUXR |= (1<<2); //Timer2 set as 1T mode.        //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256.
    IE2 &= ~(1<<2).                                 // disable interrupt
    AUXR |= (1<<4); //Timer run enable.             //Timer run enable
}

void UART1_config(u8 brt)                           //Select baud rate.
                                                    //2: Using Timer2 for baud rate
                                                    // Other values: Use Timer1 for baud rate.
{
```

```
/********** baud rate usage timer 2 ****************/
```

```
    if(brt == 2)
    {
        AUXR |= 0x01.                                   //S1 BRT Use Timer2.
        SetTimer2Baudraye(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /********** baud rate usage timer 1
    ****************/ else
    {
        TR1 = 0;
        AUXR &= ~0x01.                                  //S1 BRT Use Timer1.
        AUXR |= (1<<6); //Timer1 set as 1T mode.        //Timer1 set as 1T mode
        TMOD &= ~(1<<6); //Timer1 Set As Timer          //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 interrupt
        INTCLKO &= ~0x02.                               //no clock output
        TR1 = 1;
    }
    /*************************************************/

    SCON = (SCON & 0x3f) | 0x40.                        //UART1 mode, SCON = (SCON & 0x3f | 0x40)
                                                        //0x00: Synchronous shift output, //0x00: Synchronous
                                                        shift output, //0x00: Synchronous shift output, //0x00:
                                                        Synchronous shift output, //0x00: Synchronous shift
                                                        output, //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;                                           //Allow interrupts
    REN = 1;                                            // Allow to receive
    P_SW1 &= 0x3f.
    P_SW1 |= 0x00;
}

void UART1_DMA_Interrupt(void) interrupt 13
{
    if (DMA_UR1T_STA & 0x01)                            //Transmission complete
    {
        DMA_UR1T_STA &= ~0x01.
        DMATxFlag = 1;
    }
    if (DMA_UR1T_STA & 0x04)                            //Data overwrite
    {
        DMA_UR1T_STA &= ~0x04;
    }

    if (DMA_UR1R_STA & 0x01)                            //Reception complete
    {
        DMA_UR1R_STA &= ~0x01.
        DMARxFlag = 1;
    }
    if (DMA_UR1R_STA & 0x02)                            //Data discard
    {
        DMA_UR1R_STA &= ~0x02;
    }
}
```

// Document: ISR.ASM

```
//Interrupt number greater than 31 requires interrupt entry address remapping.
                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 Methodology

Based on the predefined DMA packet length (e.g. 256 bytes), a packet of data (254 bytes) is sent via the serial tool with a 2-byte CCITT-CRC16 checksum at the end:

After MCU receives the whole packet data (256 bytes), it performs CRC16 checksum on the first 254 bytes of data, compares the resulting checksum with the last two bytes, and if the values are equal, it prints "OK!" and the calculated checksum, and then outputs the contents received by the DMA space.

If the checksum values are not equal, print "ERROR!" and the calculated checksum.

28.12.3 Use SPI_DMA+LCM_DMA to double buffer the TFT screen.

```
//Test operating frequency is 35MHz

/*************  Function Description  **************
Use SPI DMA to read data from the external serial FLASH, store the data in the XDATA buffer, and then use LCM DMA to write the data in the buffer to the TFT screen.
The whole process is sampled in double-buffered Ping-Pang mode:
1、SPI_DMA Read data from FLASH to buffer 1
2、After the completion of SPI_DMA in the previous step, start LCM_DMA to send the data in buffer 1 to the colour screen, meanwhile, SPI_DMA will send the data from FLASH to the colour screen, and SPI_DMA will send the data from FLASH to the colour screen.
Read data into buffer 2
3、After the completion of SPI_DMA in the previous step, start LCM_DMA to send the data in buffer 2 to the colour screen, and at the same time, SPI_DMA sends the data from FLASH to the colour screen, and SPI_DMA sends the data from FLASH to the colour screen.
Read data into buffer
4. Repeat steps 2 and 3
This test code has been tested on the Experiment Box 9.4B. The use of DMA interrupts and double buffering can greatly improve CPU efficiency, and the measured data are as follows:
```

DMA buffer size	interrupt cycle	Interrupt code execution time	CPU Occupancy of DMA Interrupt Handlers
3K	7.5ms	1.6us	0.021 per cent
2K	5.0ms	1.6us	0.032 per cent
1K	2.49ms	1.6us	0.064 per cent
512	1.25ms	1.6us	0.128 per cent
256	610us	1.6us	0.262 per cent
128	310us	1.6us	0.516 per cent

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

//#include "stc8h.h"
#include "stc32g.h"                                  // see download software for header files
#include "stdio.h"

#define    FOSC            35000000UL                // System operating frequency
#define    BAUD            (65536 - (FOSC/115200+2)/4)

                                                     // The // plus 2 operation is to allow the Keil compiler to //Automatic implementation of rounding operations
#define    T2S             (65536 - FOSC*2/12/128)

#define    SCREENCX        320                       // Width of the //TFT colour screen (horizontal pixels)
#define    SCREENCY        240                       // Height of the //TFT colour screen (vertical pixels)
```

```
#define   IMG1_ADDR       0x00000000      //Starting address of the first image in FLASH.
#define   IMG2_ADDR       0x00025800      //Starting address of the 2nd image in FLASH.
#define   IMG3_ADDR       0x0004b000      //Starting address of the 3rd image in FLASH.

#define   HIBYTE(w)       (BYTE)(((WORD)(w)) >> 8)
#define   LOBYTE(w)       (BYTE)(w)

#defineRGB565(r, g, b)((((r) & 0x1f) << 11) | (((g) & 0x3f) << 5) | ((b) & 0x1f))
```

```
#define     WHITE               RGB565(31, 63, 31)      //White
#define     BLACK               RGB565(0, 0, 0)         //Black
#define     GRAY                RGB565(16, 32, 16)      //Grey
#define     RED                 RGB565(31, 0, 0)        //Red
#define     GREEN               RGB565(0, 63, 0)        //Green
#define     BLUE                RGB565(0, 0, 31)        //Blue
#define     CYAN                RGB565(0, 63, 31)       //Cyan
#define     MAGENTA             RGB565(31, 0, 31)       //Purple
#define     YELLOW              RGB565(31, 63, 0)       //Yellow

#define                         (3*1024)                //DMA buffer size
DMA_BUFSIZE                                             //320*240 colour screen has 150KB of image data in one screen.

                                BOOL.                   /DMA buffer setting of 3K is better.
typedef     bit
            typedefunsign       byte;
ed char                         word;
            typedefunsign       dword.
ed int
            typedefunsign
ed long

sbit SPI_SS             =   p2^2;
sbit SPI_MOSI           =   p2^3;
sbit SPI_MISO           =   p2^4;
sbit SPI_SCLK           =   p2^5.

sbit LCM_CS             =   p3^4;
sbit LCM_RST            =   p4^3;
sbit LCM_RS             =   p4^5;
sbit LCM_RD             =   p4^4;
sbit                        p4^2;
                    L       p6
CM_WR=#define
LCM_DB

#define     LCM_WRDB(d)     LCM_WR = 0;\
                            LCM_DB = (d);\
                            _nop_();\
                            LCM_WR = 1

void delay_ms(WORD n);
void  delay_us(WORD  n);
void sys_init().
void lcm_init();
void lcm_write_cmd(BYTE   cmd).
void lcm_write_dat(BYTE    dat).
void lcm_write_dat2(WORD dat).
void lcm_set_window(int x0, int x1, int y0, int y1);
void lcm_clear_screen().
void lcm_show_next_image().

BYTE xdata buffer1[DMA_BUFSIZE];                        //define the buffer
BYTE xdata buffer2[DMA_BUFSIZE];; BYTE xdata buffer2[DMA_BUFSIZE].      //Note: If you need to use DMA to send data, the
                                                        //then the buffer must be defined in the xdata area.
BYTE image;
BYTE stage;
BYTE

BOOL f2s.                                               /2s flag bit
```

```
void main()
{
    sys_init().                    //System initialisation
    lcm_init();
```

```
    f2s = 1;
    while (1)
    {
        if (f2s)
        {
            f2s = 0;
            lcm_show_next_image().                    //Automatically show the next image every 2 seconds.
        }
    }
}

void tm0_isr() interrupt 1
{
    f2s =  1; //Set the 2 seconds flag.                //Set the 2 seconds flag
}

void common_isr(void) interrupt 13
{
    if (DMA_LCM_STA & 0x01)
    {
        DMA_LCM_STA = 0;
        if (stage >= 2L*SCREENCY*SCREENCX/DMA_BUFSIZE)
        {
            LCM_CS = 1;
        }
    }

    if (DMA_SPI_STA & 0x01)
    {
        DMA_SPI_STA = 0;

        if (! (stage & 1))
        {
            DMA_LCM_TXAL = (BYTE)&buffer1; //Set the DMA buffer start address.    //Set the DMA buffer start address.
            DMA_LCM_TXAH = (WORD)&buffer1 >> 8;
            DMA_LCM_CR = 0xa1; //Start DMA to start sending data.          //Start DMA to start sending data
        }
        else
        {
            DMA_LCM_TXAL = (BYTE)&buffer2; //Set the DMA buffer start address.    //Set the DMA buffer start address.
            DMA_LCM_TXAH = (WORD)&buffer2 >> 8;
            DMA_LCM_CR = 0xa1; //Start DMA to start sending data.          //Start DMA to start sending data
        }

        if (stage < 2L*SCREENCY*SCREENCX/DMA_BUFSIZE)
        {
            if (! (stage & 1))
            {
                DMA_SPI_RXAL = (BYTE)&buffer2; //Set the DMA buffer start address.      //Set the DMA buffer start
                address.
                DMA_SPI_RXAH = (WORD)&buffer2 >> 8;
            }       DMA_SPI_CR = 0xc1; //Start DMA to start receiving data.          //Start DMA to start receiving data
            else
            {

                DMA_SPI_RXAL = (BYTE)&buffer1; //Set the DMA buffer start address.      //set DMA buffer start address
                DMA_SPI_RXAH = (WORD)&buffer1 >> 8;
            }       DMA_SPI_CR = 0xc1; //Start DMA to start receiving data.          //Start DMA to start receiving data
            stage++;
```

```
            }
            else
            {
                SPI_SS = 1;
            }
        }
}

void delay_ms(WORD n)
{
        while (n--)
            delay_us(1000);
}

void delay_us(WORD n)
{
        while (n--)                                     // 24 clocks per cycle
        {
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().

            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
            _nop_().
        }
}

void sys_init()
{
        wtst = 0x00;
        ckcon = 0x00;
        eaxfr = 1.

        P0M0 = 0x00; P0M1 = 0x00;
        P1M0 = 0x00; P1M1 = 0x00;
        P2M0 = 0x00; P2M1 = 0x00;
        P3M0 = 0x10; P3M1 = 0x00;
        P4M0 = 0x3c; P4M1 = 0x00;
        P5M0 = 0x00; P5M1 = 0x00;
        P6M0 = 0xff. P6M1 = 0x00;
        P7M0 = 0x00; P7M1 = 0x00.

        TM0PS = 127;                                    //set timer 0 clock prescaler system to
        128(127+1) TMOD = 0x00;
        T0x12 = 0.
        TL0 = T2S.                                      //Set the timing period of Timer 0 to 2 seconds.
```

```
    th0 = t2s >> 8;
    tr0 = 1.
    ET0 = 1;
    EA = 1;

    SPI_S0 = 1;                                   //P2.2(SS_2)/P2.3(MOSI_2)/P2.4(MISO_2)/P2.5(SCLK_2)
    SPI_S1 = 0;
    SPI_SS = 1;
    SPCTL = 0xd0;                                 //initialise SPI module (master
    mode,CPHA=CPOL=0) SPIF = 1;

    f3s = 0;
    image = 0;
}

void lcm_init()
{
    static BYTE code INIT[]     INIT[] =          //TFT screen initialisation command
    {
        4.  0xcf,   0x00, 0xc1, 0x30.
        5.  0xed.   0x64, 0x03, 0x12, 0x81.
        4.  0xe8,   0x85, 0x10, 0x7a.
        6.  0xcb.   0x39, 0x2c, 0x00, 0x34, 0x02.
        2.  0xf7,   0x20.
        3.  0xea.   0x00, 0x00.
        2.  0xc0,   0x1b.
        2.  0xc1,   0x01.
        3.  0xc5,   0x30, 0x30,
        2.  0xc7,   0xb7,
        2.  0x36.   0x28.
        2.  0x3a,   0x55.
        3.  0xb1,   0x00, 0x1a.
        3.  0xb6,   0x0a, 0xa2.
        2.  0xf2,   0x00.
        2.  0x26.   0x01.
        16. 0xe0,   0x0f, 0x2a, 0x28, 0x08, 0x0e, 0x08, 0x54.
            0xa9,   0x43, 0x0a, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00.
        16. 0xe1,   0x00, 0x15, 0x17, 0x07, 0x11, 0x06, 0x2b.
            0x56.   0x3c, 0x05, 0x10, 0x0f, 0x3f, 0x3f, 0x0f.
        5.  0x2b.   0x00, 0x00, HIBYTE(SCREENCY-1), LOBYTE(SCREENCY-1),
        5.  0x2a.   0x00, 0x00, HIBYTE(SCREENCX-1), LOBYTE(SCREENCX-1),
        1.  0x11.
        0
    };
    BYTE i, j.

    LCM_DB = 0xff;
    LCM_RS = 1;
    LCM_CS = 1;
    LCM_WR = 1;
    LCM_RD = 1;
    LCM_RST = 0; //Reset TFT colour screen          //Reset TFT colour screen
    delay_ms(50);
    LCM_RST = 1;
    delay_ms(50);

    i = 0;
    while (INIT[i])
    {
```

```
            j = INIT[i++] - 1;
            lcm_write_cmd(INIT[i++]);
            while (j--)
            {
                lcm_write_dat(INIT[i++]).
            }
        }

        lcm_clear_screen().
        lcm_write_cmd(0x29).                    //Turn on the display
}

void lcm_write_cmd(BYTE cmd)                    //write command to colour screen
{
        LCM_RS = 0;
        LCM_CS = 0;
        LCM_WRDB(cmd);
        LCM_CS = 1;
}

void lcm_write_dat(BYTE dat)                    //Write 8-bit data to the colour screen
{
        LCM_RS = 1;
        LCM_CS = 0;
        LCM_WRDB(dat);
        LCM_CS = 1;
}

void lcm_write_dat2(WORD dat)                   //Write 16-bit data to the colour screen
{
        LCM_RS = 1;
        LCM_CS = 0;
        LCM_WRDB(dat >> 8);
        LCM_WRDB(dat);
        LCM_CS = 1;
}

BYTE spi_shift(BYTE dat)                        //Use SPI to read or write FLASH data.
{
        SPIF = 1;
        SPDAT = dat;
        while (!SPIF);

        return SPDAT.
}

void lcm_set_window(int x0, int x1, int y0, int y1)   //Define the window size in TFT colour screen.
{
        lcm_write_cmd(0x2a);
        lcm_write_dat2(x0);
        lcm_write_dat2(x1);
        lcm_write_cmd(0x2b);
        lcm_write_dat2(y0);
        lcm_write_dat2(y1).
}

void lcm_clear_screen()                         //clear screen (use white colour to fill full screen)
{
        int i, j.
```

```
    lcm_set_window(0, SCREENCX - 1, 0, SCREENCY - 1);

    lcm_write_cmd(0x2c);                                    //Set the command to write display data
    LCM_RS = 1;
    LCM_CS = 0;
    for (i=SCREENCY; i; i--)
    {
        for (j=SCREENCX; j; j--)
        {
            LCM_WRDB(WHITE >> 8);
            LCM_WRDB(WHITE).
        }
    }
    LCM_CS = 1;
}

void lcm_show_next_image()
{
    DWORD addr.

    switch (image++)                                        //Get the starting address of the image in Flash.
    {
    default:    image = 1;
    case  0:  addr  =  IMG1_ADDR;  break;
    case  1:  addr  =  IMG2_ADDR;  break;
    case 2: addr = IMG3_ADDR; break;
    }

    SPI_SS = 0;

    spi_shift(0x03).                                        //Send command to read FLASH data.
    spi_shift((BYTE)(addr >> 16));                          //Set the destination address
    spi_shift((BYTE)(addr >> 8));
    spi_shift((BYTE)(addr));

    lcm_set_window(0, SCREENCX - 1, 0, SCREENCY - 1);

    lcm_write_cmd(0x2c);                                    //Set the command to write display data
    LCM_RS = 1;
    LCM_CS = 0;

    LCMIFCFG = 0x04; //Set the LCM interface to 8-bit data bit, I8080 interface, data port to P6.   //Set LCM interface to 8-bit data bit, I8080 interface, data port is P6
    lcmifcfg2 = 0x09; //Configure LCM timing.              //configure LCM timing
    LCMIFSTA = 0x00; //Clear LCM status.                   //clear LCM status
    LCMIFCR = 0x80; //Enable the LCM interface.            //Enable LCM interface
    DMA_LCM_CFG = 0xc0.                                     //Enable LCM send DMA function, enable DMA interrupt.
    DMA_LCM_STA = 0x00; //Clear DMA status.                //clear DMA status
    DMA_LCM_AMT = (DMA_BUFSIZE - 1); //Set the DMA transfer data length.   //Set the DMA transfer data length
    DMA_LCM_AMTH = (DMA_BUFSIZE - 1) >> 8;
    DMA_SPI_CFG = 0xa0; //Enable SPI receive DMA function, enable DMA interrupt.   //Enable SPI receive DMA function, enable DMA interrupt.
    DMA_SPI_STA = 0x00; //Clear DMA status.                //clear DMA status
    DMA_SPI_AMT = (DMA_BUFSIZE - 1); //Set the DMA transfer data length.   //Set the DMA transfer data length
    DMA_SPI_AMTH = (DMA_BUFSIZE - 1) >> 8;
    DMA_SPI_CFG2 = 0x00; /DMA_SPI_CFG2 = 0x00               //DMA transfer does not control SS pin during
```

transmission

```
DMA_SPI_RXAL = (BYTE)&buffer1; //Set the DMA buffer start address.        //Set the DMA buffer start address.
DMA_SPI_RXAH = (WORD)&buffer1 >> 8;

DMA_SPI_CR = 0xc1; //Start DMA to start receiving data.        //Start DMA to start receiving data
```

```
    stage = 0;
}
```

// Document: ISR.ASM

//Interrupt number greater than 31 requires interrupt entry address remapping.

```
            CSEG    AT      018BH                   SPIDMA_VECTOR
            JMP             SPIDMA_ISR
            CSEG    AT      01D3H                   LCMDMA_VECTOR
            JMP             LCMDMA_ISR

SPIDMA_ISR:
LCMDMA_ISR.
            JMP             006BH

            END
```

29 CAN bus

The STC32G series microcontrollers integrate two independent CAN bus function units and support the CAN 2.0 protocol.

The main functions are as follows:

- Receiving and transmitting standard and extended frame messages
- 64-byte receive FIFO
- Single/double acceptance filters are available in both standard and extended formats
- Error counters for sending and receiving
- Bus Error Analysis

29.1 CAN Function Pin Switching

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		**CAN_S[1:0]**		SPI_S[1:0]		LIN_S[1:0]	

CAN_S[1:0]: CAN function pin select bits

CAN_S[1:0]	CAN_RX	CAN_TX
00	P0.0	P0.1
01	P5.0	P5.1
10	P4.2	P4.5
11	P7.0	P7.1

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW3	BBH	I2S_S		S2SPI_S[1:0]		S1SPI_S[1:0]		CAN2_S[1:0]	

CAN2_S[1:0]: CAN2 function pin select bits

CAN2_S[1:0]	CAN2_RX	CAN2_TX
00	P0.2	P0.3
01	P5.2	P5.3
10	P4.6	P4.7
11	P7.2	P7.3

29.2 CAN-related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CANICR	CANBUS Interrupt Control Register	F1H	PCAN2H	CAN2IF	CAN2IE	PCAN2L	PCANH	CANIF	CANIE	PCANL	0000,0000
CANAR	CANBUS Address Register	7EFEBBH									0000,0000
CANDR	CANBUS Data Register	7EFEBCH									0000,0000
AUXR2	Auxiliary Register 2	97H	-	-	-	-	CANSEL	CAN2EN	CANEN	LINEN	xxxx,0000

29.2.1 Auxiliary Register 2 (AUXR2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR2	97H	-	-	-	-	CANSEL	CAN2EN	CANEN	LINEN

CANEN: CAN bus enable control bit

0: Disable CAN function

1: Enable CAN function

CAN2EN: CAN2 bus enable control bit

0: Disable CAN2 function

1: Enable CAN2 function

CANSEL: CAN bus selection

0: Select the first CAN group

1: Select Group II CAN

29.2.2 CAN Bus Interrupt Control Register (CANICR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CANICR	F1H	PCAN2H	CAN2IF	CAN2IE	PCAN2L	PCANH	CANIF	CANIE	PCANL

CANIE: CAN bus interrupt enable control bit

0: Disable CAN interrupt

1: Enable CAN interrupt

CANIF: CAN bus interrupt request flag bit, need to be cleared by software.

PCANH, PCANL: CAN interrupt priority control bits

PCANH	PCANL	priority
0	**0**	**0 (minimum)**
0	1	1
1	0	2
1	1	3 (maximum)

CAN2IE: CAN2 bus interrupt enable control bit

0: Disable CAN2 interrupt

1: Enable CAN2 interrupt

CAN2IF: CAN2 bus interrupt request flag bit, need to be cleared by software.

PCAN2H, PCAN2L: CAN2 interrupt priority control bits

PCAN2H	PCAN2L	priority
0	**0**	**0 (minimum)**
0	1	1
1	0	2
1	1	3 (maximum)

29.2.3 CAN Bus Address Register (CANAR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CANAR	7EFEBBH								

29.2.4 CAN Bus Data Register (CANDR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CANDR	7EFEBCH								

Read and write to the CAN internal function registers require indirect

access to **the CAN internal function registers** via CANAR and CANDR:

1. Write the address of the CAN internal function register into the CAN bus address register CANAR.
2. Read CAN bus data register CANDR

For example, the CAN internal function register ISR needs to be read.

```
CANAR = 0x03;          //write the address of the ISR
toCANAR dat = CANDR;   //read CANDR to get
the value of the ISR
```

Method of writing the CAN internal function registers:

1. Write the address of the CAN internal function register into the CAN bus address register CANAR.
2. Write the value to be written to CAN bus data register CANDR.

Example: Data 0x5a needs to be written to the CAN

```
internal function registerTXBUF0 CANAR = 0x08;
                          //Write the address of TXBUF0
to CANAR CANDR = 0x5a;    // Write the value to be
written, 0x5a, to CANDR
```

29.3 CAN Internal Function Register

Note: The two CAN groups are completely independent in hardware, and the internal registers of the two CAN groups are also independent, only the access address is the same. When you need to access the internal registers of different CAN groups, you need to select them through AUXR2.CANSEL first and then you can access them correctly.

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
18H	ECC	RXERR	TXERR	ALC				
10H	ACR0	ACR1	ACR2	ACR3	AMR0	AMR1	AMR2	AMR3
08H	TXBUF0	TXBUF1	TXBUF2	TXBUF	RXBUF0	RXBUF1	RXBUF2	RXBUF3
00H	MR	CMR	SR	ISR	IMR	RMC	BTR0	BTR1

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
MR	0x00						RM	LOM	AFM
CMR	0x01						TR	AT	
SR	0x02	RBS	DSO	TBS		RS	TS	ES	BS
ISR/IACK	0x03		ALI	EWI	EPI	RI	TI	BEI	DOI
IMR	0x04		ALIM	EWIM	EPIM	RIM	TIM	BEIM	DOIM
RMC	0x05				RMC4	RMC3	RMC2	RMC1	RMC0
BTR0	0x06	SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0
BTR1	0x07	SAM	TSG2.2	TSG2.1	TSG2.0	TSG1.3	TSG1.2	TSG1.1	TSG1.0

TXBUF0	0x08	frame byte n
TXBUF1	0x09	frame byte n+1
TXBUF2	0x0A	frame byte n+2
TXBUF3	0x0B	frame byte n+3

RXBUF0	0x0C	frame byte n							
RXBUF1	0x0D	frame byte n+1							
RXBUF2	0x0E	frame byte n+2							
RXBUF3	0x0F	frame byte n+3							
ACR0	0x10	ACR0							
ACR1	0x11	ACR1							
ACR2	0x12	ACR2							
ACR3	0x13	ACR3							
AMR0	0x14	AMR0							
AMR1	0x15	AMR1							
AMR2	0x16	AMR2							
AMR3	0x17	AMR3							
ECC	0x18	RXWRN	TXWRN	EDIR	ACKER	FRMER	CRCER	STFER	BER
RXERR	0x19	RXERR							
TXERR	0x1A	TXERR							
ALC	0x1B				ALC.4	ALC.3	ALC.2	ALC.1	ALC.0

29.3.1 CAN Mode Register (MR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
MR	0x00	-	-	-	-	-	RM	LOM	AFM

RM:RESET MODE of CAN module 0: disable RESET MODE
1: Enable RESET MODE

LOM:LISTEN ONLY MODE of CAN module 0: turn off LISTEN ONLY MODE
1: Enable LISTEN ONLY MODE

AFM: Receive filter selection for CAN module (see ACR register description)
0: Accepts filter with double filter setting
1: Acceptance filter with single filter setting

29.3.2 CAN Command Register (CMR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
CMR	0x01	-	-	-	-	-	TR	AT	-

TR:CAN module sends request
0:
1: Initiate a frame transmission AT:CAN module transmission termination
0:
1: Terminate the current frame transmission

29.3.3 CAN Status Register (SR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
SR	0x02	RBS	DSO	TBS	-	RS	TS	ES	BS

RBS: Receive BUFFER Status

0: Receive BUFFER no data frame

1: Receive BUFFER with data frame

DSO: Receive FIFO overflow loop flag

0: No overflow cycle generated in the receive FIFO

1: Receive FIFO with overflow cycle generated

TBS: CAN module sends BUFFER status.

0: Send BUFFER disable, CPU cannot write to BUFFER.

1: Send BUFFER idle, CPU can write to BUFFER.

RS: CAN module receive status

0: CAN module receive idle

1: CAN module is receiving a data frame

TS:CAN module transmission status

0: CAN module transmits idle

1: CAN module is sending data frames

ES:CAN module error status

0: CAN module error register value not reached 96

1: The CAN module has at least one error register with a value of 96 or more BS:CAN module BUS-OFF status

0: CAN module is not in BUS-OFF state

1: CAN module in BUS-OFF state A BUS-OFF error is triggered when the number of CAN controller errors exceeds 255. Generally, the condition of BUS-OFF is that the CAN bus is interfered by the surrounding environment, which causes the data sent to the bus by the CAN transmitter to be judged as abnormal by the BUS bus. However, if the number of abnormalities exceeds 255 times, the BUS bus will automatically be set to the BUS-OFF state, and at this time the bus is in the busy state, and the data can not be sent or received.

29.3.4 CAN interrupt/answer register (ISR/IACK)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
ISR/IACK	0x03	-	ALI	EWI	EPI	RI	TI	BEI	DOI

ALI:Arbitration Loss Interrupt 0:

1: Arbitration lost, write 1 to clear

EWI:Error Warning Interrupt 0:

1: This position bit when the ES or BS value in the SR register is 1. Write 1 to clear.

EPI: CAN module passive error interrupt

0:

1: Bit in this position when the CAN error register operates the passive error count value. Write 1 to clear.

RI: CAN module receive interrupt

0:

1: CAN module receives data frames present in the BUFFER, the user needs to write 1 to RI to reduce the Receive Message Counter (RMC)

Value.

TI:CAN Module Transmit Interrupt

0:

1: CAN module data frame transmission is completed. User needs to write 1 to TI to reset the write pointer of transmit BUFFER.

BEI:CAN Module Bus Error Interrupt

0:

1: The CAN module has generated a bus error during reception or transmission.

DOI: CAN module receives overflow interrupt

0:

1: CAN module receive FIFO overflow

29.3.5 CAN Interrupt Register (IMR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
IMR	0x04	-	ALIM	EWIM	EPIM	RIM	TIM	BEIM	DOIM

ALIM:Arbitration Loss

Interrupt 0:

Arbitration Loss

Interrupt Masking

1: Arbitration Loss

Interrupt On

EWIM:Error Warning

Interrupt

0: Error warning interrupt mask

1: Error warning

interrupt on EPIM: CAN

module passive error

interrupt

0: CAN module passive error interrupt blocking

1: CAN module passive error interrupt on

RI: CAN module receive interrupt

0: CAN module receive interrupt mask

1: CAN module receive interrupt on

TI:CAN Module Transmit Interrupt

0: CAN module transmit interrupt mask

1: CAN module transmit interrupt on

BEI:CAN Module Bus Error Interrupt

0: CAN module bus error interrupt blocking

1: CAN module bus error interrupt on

DOI: CAN module receives overflow interrupt

0: CAN module receive overflow interrupt blocking

1: CAN module receive overflow interrupt on

29.3.6 CAN Data Frame Receive Counter (RMC)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RMC	0x05	-	-	-	RMC[4;0]				

RMC: Data Frame Receive Counter

29.3.7 CAN bus clock register 0 (BTR0)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
BTR0	0x06	SJW[1:0]		BRP[5:0]					

BRP :CAN Baud Rate Partition Factor

BRP= BTR0[5:0]+1; CAN module internal clock tq= tCLK*BRP; tCLK = 1/ fXTAL (mains 2 division)

SJW: :resynchronise jump widths

SJW = SJW.1*2 + SJW.0

29.3.8 CAN bus clock register 1 (BTR1)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
BTR1	0x07	SAM	TSG2[2:0]			TSG1[3:0]			

TSG1: Synchronised sampling segment 1

TSG2: Synchronous

Sampling Segment 2

SAM: Number of Bus

Level Samples

0: Bus level sampled 1 time

1: Bus level sampling 3 times

CAN baud rate = 1/normal time bit

Normal time bit = (1 + (TSG1+1) + (TSG2+1))*tq

29.3.9 CAN bus data frame transmit buffer (TXBUFn)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TXBUF0	0x08	Frame byte n							
TXBUF1	0x09	Frame byte n+1							
TXBUF2	0x0A	Frame byte n+2							
TXBUF3	0x0B	Frame byte n+3							

The transmit BUFFER contains 4 registers: TXBUF0, TXBUF1, TXBUF2, TXBUF3.

Whenever the TXBUF3 register is written, the BUFFER pointer is automatically incremented by 1, and TXBUF0, TXBUF1, TXBUF2, and TXBUF3, are written to BUFFER.

29.3.10 CAN bus data frame receive buffer (RXBUFn)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RXBUF0	0x0C	Frame byte n							
RXBUF1	0x0D	Frame byte n+1							
RXBUF2	0x0E	Frame byte n+2							
RXBUF3	0x0F	Frame byte n+3							

The receive BUFFER contains 4 registers: RXBUF0, RXBUF1, RXBUF2, RXBUF3.

Whenever RXBUF3 register is written, the BUFFER pointer is automatically incremented by 1,

and RXBUF0, RXBUF1, RXBUF2, RXBUF3, are written to the BUFFER. the maximum length of a CAN frame is 16 BYTE, so it is necessary to read the RXBUF four times in a cycle for receiving a frame of data. the RXFIFO of CAN module is a 64 BYTE FIFO, which can store at most 1 BYTE of data. The RXFIFO of CAN module is a 64BYTE FIFO, which can store up to 1 BYTE of data.

21 frames of data. Up to 5 frames of data can be stored when the data is 8 BYTE. The number of received frames can be obtained by reading the RMC (RECEIVE MESSAGE COUNTER) register.

CAN bus acceptance filter

With the help of acceptance filters, the CAN controller can allow the RXFIFO to receive only those messages that correspond to the identification code and the preset values in the acceptance filters. The acceptance filters are defined in the Acceptance Code Register ACR and the Acceptance Mask Register AMR.

29.3.11 CAN Bus Acceptance Code Register (ACRn)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
ACR0	0x10	ACR0							
ACR1	0x11	ACR1							
ACR2	0x12	ACR2							
ACR3	0x13	ACR3							

29.3.12 CAN Bus Acceptance Mask Register (AMRn)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
AMR0	0x14	AMR0							
AMR1	0x15	AMR1							
AMR2	0x16	AMR2							
AMR3	0x17	AMR3							

There are two types of filtering, selected by the AFM (MR.0) bit in the Mode Register: single filter mode (AFM bit is 1), and dual filter mode (AFM bit is 0).

The filtering rule is: each acceptance mask corresponds to each acceptance code, when the acceptance mask bit is "1" (i.e., set to irrelevant), the corresponding frame ID bit of the received frame is indicated as received regardless of whether it is the same as the corresponding acceptance code bit; when the acceptance mask bit is "0 When the Acceptance Mask bit is "0 (i.e. set to Relevant), only if the corresponding Frame ID bit and the corresponding Acceptance Code bit have the same value is the message indicated as received. The CAN controller will only accept the message if all bits are received.

(1) Single filter configuration

This filter configuration defines a long filter (4 bytes, 32 bits), an acceptance filter consisting of 4 Acceptance Code Registers and 4 Acceptance Mask Registers, and the bit correspondence between the filter byte and the message byte depends on the current receive frame format. Receive CAN standard frame single filter configuration: for standard frame, 11-bit identifier, RTR bit, and the first two bytes of the data field are involved in the filtering; for the data involved in the filtering, all the ACR bits corresponding to the bits of AMR 0 and the corresponding bits of the data involved in the filtering must be the same in order to be considered as passing the acceptance; if there is no data byte due to the setting of the RTR=1 bit, or there is no or only one data byte of information because of the setting of the corresponding data length code, the acceptance filter is not available or there is only one data byte of information. If there is no data byte due to setting the RTR=1 bit, or if there is no or only one data byte of information due to setting the corresponding data length code, the message is also received. For a successfully received message, all individual bits must

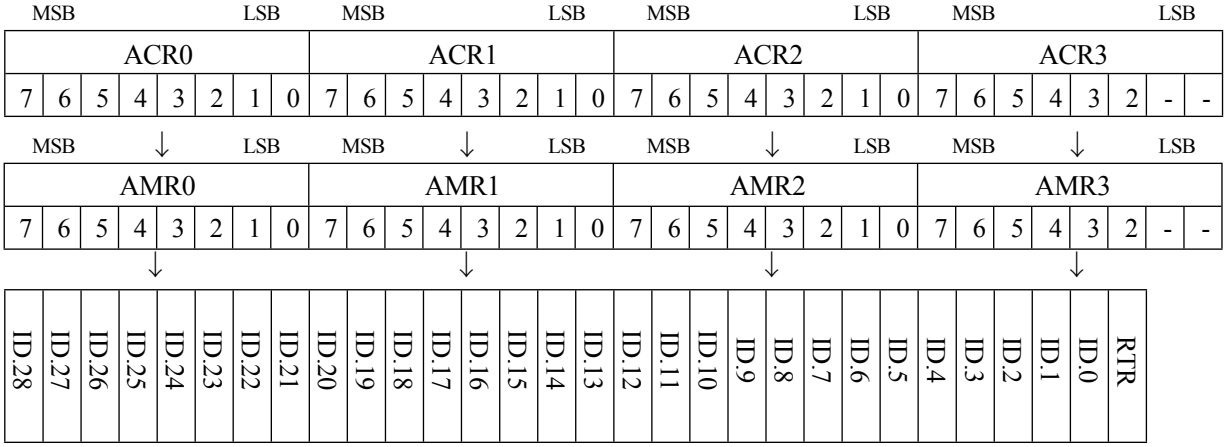
be "accepted" for comparison in the filter; note that the lower four bits of AMR1 and ACR1 are not used, and for compatibility with future products these bits can be set to "do not affect" by setting AMR1.3, AMR1.2, AMR1.1, and AMR1.0 to one; these bits can be set to "do not affect" by setting RTR=1 to one. and AMR1.0 to 1.

Single filter configuration for receiving CAN standard frames:

RTR
ID.0
ID.1
ID.2
ID.3
ID.4
ID.5
ID.6
ID.7
ID.8
ID.9
ID.10

DB2.0
DB2.1
DB2.2
DB2.3
DB2.4
DB2.5
DB2.6
DB2.7
DB1.0
DB1.1
DB1.2
DB1.3
DB1.4
DB1.5
DB1.6
DB1.7

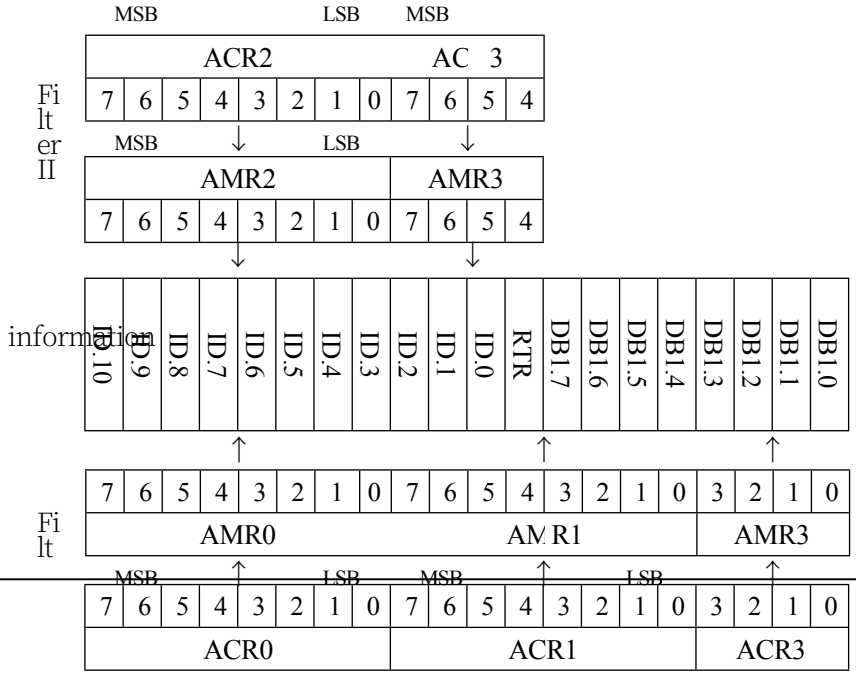
Single filter configuration when receiving CAN extended frames:



(2) Dual Filter Configuration

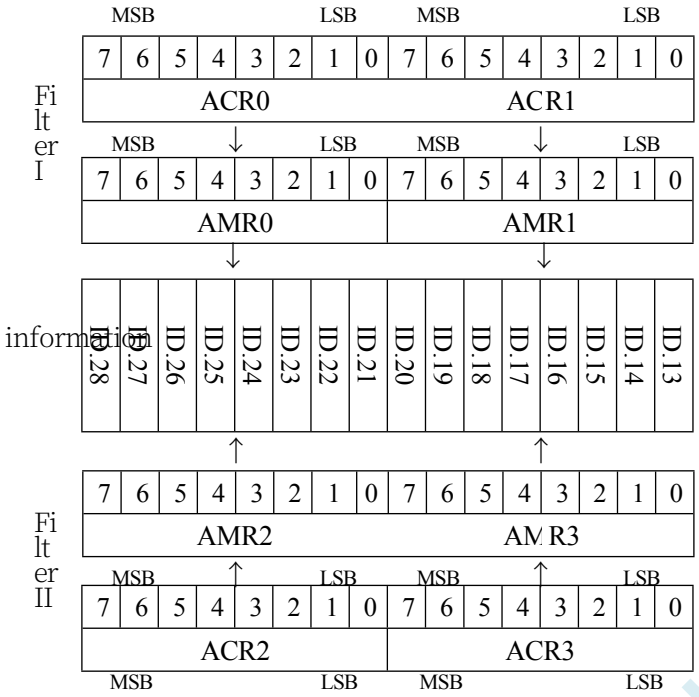
This configuration allows the definition of two short filters, consisting of 4 ACRs and 4 AMRs forming two short filters. Information on the bus is received as soon as it passes through either filter. The bit correspondence between the filter byte and the message byte depends on the currently received frame format. Configuration of the two filters for receiving CAN standard frames: If a standard frame message is received, the two filters defined are different. The first filter consists of ACR0, ACR1, AMR0, AMR1 and the lower 4 bits of ACR3, AMR3, with the 11-bit identifier, the RTR bit and byte 1 of the data field participating in the filtering; the second filter consists of ACR2, AMR2 and the upper 4 bits of ACR3, AMR3, with the 11-bit identifier and the RTR bit participating in the filtering. In order to successfully receive a message, at least one filter should indicate acceptance at all individual bit comparisons; an RTR position of "1" or a data length code of "0" indicates that no data byte is present; as long as the portion from the beginning to the RTR bit is indicated as acceptance, the message can pass through the filter. As long as the portion of the RTR bit from the beginning to the RTR bit is accepted, the message can pass through filter 1. If no data bytes are requesting filtering from the filters, the lower 4 bits of AMR1 and AMR3 must be set to "1", i.e., "do not affect". In this case, the recognition of both filters is to verify the entire standard identification code including the RTR bit.

Dual filter configuration for receiving CAN standard frames:



MSB LSB MSB LSB LSB

Dual filter configuration when receiving CAN extended frames:



29.3.13 CAN bus error message register (ECC)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
ECC	0x18	RXWRN	TXWRN	EDIR	ACKER	FRMER	CRCER	STFER	BER

RXWRN: This bit is present when RXERR is greater than or equal to 96.
TXWRN: This bit is present when TXERR is greater than or equal to 96. EDIR: Transmission Error Direction. 0: Error occurred while transmitting. 1: Error occurred while receiving. ACKER: ACK error.
FRMER: Frame format error. CRCER: CRC error. STFER: Bit stuffing error. BER: Bit error.

29.3.14 CAN bus receive error counter (RXERR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
RXERR	0x19	RXERR							

RXERR: The counter value represents the current receive error count value. This register is read-only. This counter value is cleared by hardware when a BUS-OFF event occurs.

29.3.15 CAN bus transmit error counter (TXERR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TXERR	0x1A	TXERR							

TXERR: Transmit Error Counter Register reflects the current value of the transmit error counter, which is read-only in the operating mode, hardware reset

The post register is initialised to 0. When the BUS-OFF time occurs the error counter is initialised to 127 to calculate the minimum time defined for the bus (128 bus idle signals). Reading the transmit error counter during this time will reflect the status information of the bus-off recovery.

29.3.16 CAN Bus Arbitration Loss Register (ALC)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
ALC	0x1B	-	-	-	ALC[4:0]				

ALC: This register contains information about the location of the arbitration loss. After the current Arbitration Loss interrupt is processed (answered), this value is updated for the next Arbitration Loss. The specific values correspond to the table below:

ALC[4:0]	numerical value	descriptions
00000	0	Arbit lost in ID28/11
00001	1	Arbit lost in ID27/10
00010	2	Arbit lost in ID26/9
00011	3	Arbit lost in ID25/8
00100	4	Arbit lost in ID24/7
00101	5	Arbit lost in ID23/6
00110	6	Arbit lost in ID22/5
00111	7	Arbit lost in ID21/4
01000	8	Arbit lost in ID20/3
01001	9	Arbit lost in ID19/2
01010	10	Arbit lost in ID18/1
01011	11	Arbit lost in ID17/0
01100	12	Arbit lost in SRTR/RTR
01101	13	Arbit lost in IDE bit
01110	14	Arbit lost in ID16*
01111	15	Arbit lost in ID15*
10000	16	Arbit lost in ID14*
10001	17	Arbit lost in ID13*
10010	18	Arbit lost in ID12*
10011	19	Arbit lost in ID11*
10100	20	Arbit lost in ID10*
10101	21	Arbit lost in ID9*
10110	22	Arbit lost in ID8*
10111	23	Arbit lost in ID7*
11000	24	Arbit lost in ID6*
11001	25	Arbit lost in ID5*
11010	26	Arbit lost in ID4*
11011	27	Arbit lost in ID3*
11100	28	Arbit lost in ID2*
11101	29	Arbit lost in ID1*
11110	30	Arbit lost in ID0*

11111	31	Arbit lost in RTR

29.4 sample procedure

29.4.1 CAN bus frame format

CAN2.0B standard frame

The CAN standard frame message is 11 bytes and consists of two parts: the information and the data part. The first 3 bytes are the information part.

placement	B7	B6	B5	B4	B3	B2	B1	B0
Byte 01	FF	RTR	-	-	DLC (data length)			
Byte 02	CAN ID[10:3]							
Bytes 03	CAN ID[2:0]			-	-	-	-	-
Bytes 04	Data 1							
Byte 05	Data 2							
Byte 06	Data 3							
Bytes 07	Data 4							
Bytes 08	Data 5							
Bytes 09	Data 6							
Bytes 10	Data 7							
Bytes 11	Data 8							

Byte 1 is the frame information. Bit 7 (FF) indicates the frame format, in standard frames FF = 0; bit 6 (RTR) indicates the type of frame.

RTR=0 means data frame, RTR=1 means remote frame; DLC means actual data length in data frame. Byte 2 and 3 are message identification code, 11 bits are valid.

Bytes 4 to 11 are the actual data of the data frame, invalid for remote frames.

CAN2.0B Extended Frame

The CAN extended frame information is 13 bytes and consists of two parts, the information and the data part. The first 5 bytes are the information part.

placement	B7	B6	B5	B4	B3	B2	B1	B0
Byte 01	FF	RTR	-	-	DLC (data length)			
Byte 02	CAN ID [28:21]							
Bytes 03	CAN ID [20:13]							
Byte 04	CAN ID[12:5]							
Byte 05	CAN ID[4:0]					-	-	-
Byte 06	Data 1							
Bytes 07	Data 2							
Bytes 08	Data 3							
Bytes 09	Data 4							
Bytes 10	Data 5							
Bytes 11	Data 6							
Bytes 12	Data 7							
Bytes 13	Data 8							

Byte 1 is the frame information. Bit 7 (FF) indicates the frame format, in extended frames FF = 1; bit 6 (RTR) indicates the type of frame.

RTR=0 means data frame, RTR=1 means remote frame; DLC means actual data length in data frame. Bytes 2 to 5 are the message identification code, and its 29 bits are valid.

Bytes 6 to 13 Actual data of the data frame, invalid for remote frames.

29.4.2 CAN bus standard frame sending and receiving example

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

typedef     unsigned char    u8.
typedef     unsigned int     u16;
typedef     unsigned long    u32.

#define MAIN_Fosc              24000000UL

/***************************** user-defined macros ***********************************/
//CAN bus baud rate = Fclk/((1+(TSG1+1)+(TSG2+1))*(BRP+1)*2)
#define TSG1      2                //0~15
#define TSG2      1                //0~7
#define BRP       3                //0~63
//24000000/((1+3+2)*4*2)=500KHz

#define SJW       1                //Resynchronise the jump width

/*****************************************************************************/

u16 CAN_ID;
u8 RX_BUF[8];
u8 TX_BUF[8].

void CANInit().
void CanSendMsg(u16 canid, u8 *pdat);

void main(void)
{
    EAXFR = 1;                                         //Enable access to XFR
    CKCON = 0x00;                                      //Set the external data bus speed to fastest
    WTST = 0x00;                                       //set the program code wait parameter.
                                                       //Assign a value of 0 to set the CPU to execute the
                                                       programme as fast as possible.

    p0m1 = 0;    p0m0 = 0; //Set quasi-bidirectional port.     //Set quasi-bidirectional port
    p1m1 = 0;    p1m0 = 0; //Set quasi-bidirectional port.     //Set quasi-bidirectional port
    p2m1 = 0;    p2m0 = 0; //Set quasi-bidirectional port.     //Set quasi-bidirectional port
    p3m1 = 0;    p3m0 = 0; //set the quasi-bidirectional port.  //Set quasi-bidirectional port
    p4m1 = 0;    p4m0 = 0; //Set the quasi-bidirectional port.  //Set quasi-bidirectional port
    p5m1 = 0;    p5m0 = 0; //Set quasi-bidirectional port.     //Set quasi-bidirectional port
    p6m1 = 0;    p6m0 = 0; //Set quasi-bidirectional port.     //Set quasi-bidirectional port
    p7m1 = 0;    p7m0 = 0; //Set quasi-bidirectional port.     //Set quasi-bidirectional port

    CANInit();
```

```
    EA = 1;                                 //Turn on the general interrupt

    CAN_ID = 0x0345.
```

```
        TX_BUF[0] = 0x11;
        TX_BUF[1] = 0x22;
        TX_BUF[2] = 0x33;
        TX_BUF[3] = 0x44;
        TX_BUF[4] = 0x55;
        TX_BUF[5] = 0x66;
        TX_BUF[6] = 0x77;
        TX_BUF[7] = 0x88.

        while(1)
        {
            if(!P32)                                    //Press the P32 key to send a fixed data frame.
            {
                CanSendMsg(CAN_ID,TX_BUF).
                while(!P32);                            //prevent retransmission
            }
        }
}

u8 CanReadReg(u8 addr)
{
        u8 dat.
        CANAR = addr;
        dat = CANDR;
        return dat;
}

void CanWriteReg(u8 addr, u8 dat)
{
        CANAR = addr;
        CANDR = dat.
}

void CanReadFifo(u8 *pdat)
{
        pdat[0]  =  CanReadReg(RX_BUF0);
        pdat[1]  =  CanReadReg(RX_BUF1);
        pdat[2]  =  CanReadReg(RX_BUF2);
        pdat[3] = CanReadReg(RX_BUF3).

        pdat[4]  =  CanReadReg(RX_BUF0);
        pdat[5]  =  CanReadReg(RX_BUF1);
        pdat[6]  =  CanReadReg(RX_BUF2);
        pdat[7] = CanReadReg(RX_BUF3).

        pdat[8]  =  CanReadReg(RX_BUF0);
        pdat[9]  =  CanReadReg(RX_BUF1);
        pdat[10] = CanReadReg(RX_BUF2);
        pdat[11] = CanReadReg(RX_BUF3).

        pdat[12]  =  CanReadReg(RX_BUF0);
        pdat[13]  =  CanReadReg(RX_BUF1);
        pdat[14]  =  CanReadReg(RX_BUF2);
        pdat[15] = CanReadReg(RX_BUF3).
}

u16 CanReadMsg(u8 *pdat)
{
        u8 i;
```

```
    u16 CanID;
    u8 buffer[16];

    CanReadFifo(buffer).
    CanID = ((buffer[1] << 8) + buffer[2]) >> 5;
    for(i=0;i<8;i++)
    {
        pdat[i] = buffer[i+3];
    }
    return CanID.
}

void CanSendMsg(u16 canid, u8 *pdat)
{
    u16 CanID.

    CanID = canid << 5;
    CanWriteReg(TX_BUF0,0x08).                          //Standard frame, data frame,
    bit3~bit0: data length (DLC) CanWriteReg(TX_BUF1,(u8)(CanID>>8));
    CanWriteReg(TX_BUF2,(u8)CanID);
    CanWriteReg(TX_BUF3,pdat[0]).

    CanWriteReg(TX_BUF0,pdat[1]);
    CanWriteReg(TX_BUF1,pdat[2]);
    CanWriteReg(TX_BUF2,pdat[3]);
    CanWriteReg(TX_BUF3,pdat[4]).

    CanWriteReg(TX_BUF0,pdat[5]);
    CanWriteReg(TX_BUF1,pdat[6]);
    CanWriteReg(TX_BUF2,pdat[7]).

    CanWriteReg(TX_BUF3,0x00).
    CanWriteReg(CMR,0x04).                              //initiate a frame transfer
}

void CANSetBaudrate()
{
    CanWriteReg(BTR0,(SJW << 6) + BRP);
    CanWriteReg(BTR1,(TSG2 << 4) + TSG1).
}

void CANInit()
{
    CANSetBaudrate();                                   //Set the baud rate

    CanWriteReg(ACR0,0x00).                             //Bus Acceptance
    Code Register CanWriteReg(ACR1,0x00);
    CanWriteReg(ACR2,0x00);
    CanWriteReg(ACR3,0x00).
    CanWriteReg(AMR0,0xFF).                             //Bus acceptance mask register.
    CanWriteReg(AMR1,0xFF).
    CanWriteReg(AMR2,0xFF);
    CanWriteReg(AMR3,0xFF).

    CanWriteReg(IMR,0xff).                              //Interrupt Register
    CanWriteReg(MR,0x00).

    P_SW1 = 0;
    CANICR = 0x02.                                      //CAN interrupt enable
```

```
    AUXR2 |= 0x02.                                  //CAN module is enabled
}

void CANBUS_Interrupt(void) interrupt 28
{
    u8 isr.

    isr = CanReadReg(ISR);
    if((isr & 0x04) == 0x04)
    {
        CANAR = 0x03.
        CANDR = 0x04;                               //CLR FLAG

  }
    if((isr & 0x08) == 0x08)
    {
        CANAR = 0x03.
        CANDR = 0x08;                               //CLR FLAG

        CAN_ID = CanReadMsg(RX_BUF); //Receive CAN bus data.    //Receive CAN bus data

        CanSendMsg(CAN_ID+1,RX_BUF); //CANID plus 1, send CAN bus data as is.    //CANID plus 1, send CAN bus
        data as is.
    }
}
```

29.4.3 CAN bus extended frame transceiver example

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                 // see download software for header files
#include "intrins.h"

typedef     unsigned char   u8;
typedef     unsigned int    u16;
typedef     unsigned long   u32.

#define MAIN_Fosc           24000000UL

/***************************** user-defined macros ***********************************/
//CAN bus baud rate = Fclk/((1+(TSG1+1)+(TSG2+1))*(BRP+1)*2)
#define TSG1    2               //0~15
#define TSG2    1               //0~7
#define BRP     3               //0~63
//24000000/((1+3+2)*4*2)=500KHz

#define SJW     1               //Resynchronise jump width

/*****************************************************************************/

u32 CAN_ID;
u8 RX_BUF[8];
u8 TX_BUF[8].
```

```
void CanSendMsg(u32 canid, u8 *pdat);

void main(void)
{
    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign a value of 0 to set the CPU to execute the
                                                programme as fast as possible.

    p0m1 = 0;   p0m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p1m1 = 0;   p1m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p2m1 = 0;   p2m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p3m1 = 0;   p3m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p4m1 = 0;   p4m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p5m1 = 0;   p5m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p6m1 = 0;   p6m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port
    p7m1 = 0;   p7m0 = 0; //Set quasi-bidirectional port.   //Set quasi-bidirectional port

    CANInit();

    EA = 1;                                     //Turn on the general interrupt

    CAN_ID = 0x01234567;
    TX_BUF[0] = 0x11;
    TX_BUF[1] = 0x22;
    TX_BUF[2] = 0x33;
    TX_BUF[3] = 0x44;
    TX_BUF[4] = 0x55;
    TX_BUF[5] = 0x66;
    TX_BUF[6] = 0x77;
    TX_BUF[ 7] = 0x88.

    while(1)
    {
        if(!P32)                                //Press the P32 key to send a fixed data frame.
        {
            CanSendMsg(CAN_ID,TX_BUF).
            while(!P32);                        //prevent retransmission
        }
    }
}

u8 CanReadReg(u8 addr)
{
    u8 dat.
    CANAR = addr;
    dat = CANDR;
    return dat;
}

void CanWriteReg(u8 addr, u8 dat)
{
    CANAR = addr;
    CANDR = dat.
}

void CanReadFifo(u8 *pdat)
{
    pdat[0] = CanReadReg(RX_BUF0);
```

```
    pdat[1]  =  CanReadReg(RX_BUF1);
    pdat[2]  =  CanReadReg(RX_BUF2);
    pdat[3] = CanReadReg(RX_BUF3).

    pdat[4]  =  CanReadReg(RX_BUF0);
    pdat[5]  =  CanReadReg(RX_BUF1);
    pdat[6]  =  CanReadReg(RX_BUF2);
    pdat[7] = CanReadReg(RX_BUF3).

    pdat[8]  =  CanReadReg(RX_BUF0);
    pdat[9]  =  CanReadReg(RX_BUF1);
    pdat[10] =  CanReadReg(RX_BUF2);
    pdat[11] = CanReadReg(RX_BUF3).

    pdat[12]  =  CanReadReg(RX_BUF0);
    pdat[13]  =  CanReadReg(RX_BUF1);
    pdat[14]  =  CanReadReg(RX_BUF2);
    pdat[15] = CanReadReg(RX_BUF3).
}

u32 CanReadMsg(u8 *pdat)
{
    u8 i;
    u32 CanID;
    u8 buffer[16];

    CanReadFifo(buffer).
    CanID = (((u32)buffer[1] << 24) + ((u32)buffer[2] << 16) + ((u32)buffer[3] << 8) + buffer[4]) >> 3;
    for(i=0;i<8;i++)
    {
        pdat[i] = buffer[i+5];
    }
    return CanID.
}

void CanSendMsg(u32 canid, u8 *pdat)
{
    u32 CanID.

    CanID = canid << 3;
    CanWriteReg(TX_BUF0,0x88).                                  //Extended frame, data frame,
    bit3~bit0: data length (DLC) CanWriteReg(TX_BUF1,(u8)(CanID>>24));
    CanWriteReg(TX_BUF2,(u8)(CanID>>16));
    CanWriteReg(TX_BUF3,(u8)(CanID>>8)).

    CanWriteReg(TX_BUF0,(u8)CanID);
    CanWriteReg(TX_BUF1,pdat[0]);
    CanWriteReg(TX_BUF2,pdat[1]);
    CanWriteReg(TX_BUF3,pdat[2]).

    CanWriteReg(TX_BUF0,pdat[3]);
    CanWriteReg(TX_BUF1,pdat[4]);
    CanWriteReg(TX_BUF2,pdat[5]);
    CanWriteReg(TX_BUF3,pdat[6]).

    CanWriteReg(TX_BUF0,pdat[7]);
    CanWriteReg(TX_BUF1,0x00);
    CanWriteReg(TX_BUF2,0x00);
    CanWriteReg(TX_BUF3,0x00).
```

```
    CanWriteReg(CMR,0x04).                          //initiate a frame transfer
}

void CANSetBaudrate()
{
    CanWriteReg(BTR0,(SJW << 6) + BRP);
    CanWriteReg(BTR1,(TSG2 << 4) + TSG1).
}

void CANInit()
{
    CANSetBaudrate();                               //Set the baud rate

    CanWriteReg(ACR0,0x00).                         //Bus Acceptance
    Code Register CanWriteReg(ACR1,0x00);
    CanWriteReg(ACR2,0x00);
    CanWriteReg(ACR3,0x00).
    CanWriteReg(AMR0,0xFF).                         //Bus acceptance mask register.
    CanWriteReg(AMR1,0xFF).
    CanWriteReg(AMR2,0xFF);
    CanWriteReg(AMR3,0xFF).

    CanWriteReg(IMR,0xFF).                          //Interrupt
    Register CanWriteReg(MR,0x00).

    P_SW1 = 0;
    CANICR = 0x02.                                  //CAN interrupt enable
    AUXR2 |= 0x02.                                  //CAN module is enabled
}

void CANBUS_Interrupt(void) interrupt 28
{
    u8 isr.

    isr = CanReadReg(ISR);
    if((isr & 0x04) == 0x04)
    {
        CANAR = 0x03.
        CANDR = 0x04;                               //CLR FLAG

  }
    if((isr & 0x08) == 0x08)
    {
        CANAR = 0x03.
        CANDR = 0x08;                               //CLR FLAG

        CAN_ID = CanReadMsg(RX_BUF); //Receive CAN bus data.    //Receive CAN bus data

        CanSendMsg(CAN_ID+1,RX_BUF); //CANID plus 1, send CAN bus data as is.    //CANID plus 1, send CAN bus
        data as is.
    }
}
```

29.4.4 CAN Bus Standard Frame Transceiver Test (Compilation)

assembly code

Test operating frequency is *11.0592MHz*.

```
;************* Functional description **************
```

This routine is written and tested based on the *STC32G* as the main control chip.

Using *Keil C251* compiler, *Memory Model* is recommended to set *XSmall* mode, which defines variables in *edata* by default, and provides fast access with single clock access.

It is recommended to reserve *1K* for the stack. If there is not enough space, you can define large arrays and infrequently used variables with the *xdata* keyword into the *xdata* space.

CAN bus send/receive test cases.

The ;DCAN is a functional unit that supports the *CAN2.0B* protocol.

Sends one frame of standard *CAN* bus data every second.

When it receives a standard frame, *it* replaces the original *CAN ID* with the transmitted data.

Default baud rate *500KHz*, can be changed by user.

When downloading, select the clock *24MHZ*. *(*user can modify the frequency*)*.

```
;*****************************************/

$include (STC32G.INC)

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

Fosc_KHZ        EQU     24000           ;24000KHZ

STACK_POIRTER   EQU     0D0H            Start of stack address

Timer0_Reload EQU       (65536 - Fosc_KHZ)      Timer0 interrupt frequency, 1000 times/sec.

CAN bus baud rate = Fclk/((1+(TSG1+1)+(TSG2+1))*(BRP+1)*2)
TSG1            EQU     2               ;0~15
TSG2            EQU     1               ;1~7 (TSG2 cannot be
                                        set to 0)
BRP             EQU     3               ;0~63
;24000000/((1+3+2)*4*2)=500KHz

SJW             EQU     00H             ;Resynchronise the jump width:
00h/040h/080h/0c0h

Bus baud rate above 100KHz is set to 0; below 100KHz is set to 1    0 for 100KHz or above; 1 for 100KHz or below
SAM             EQU     00H             SAM EQU 00H; bus level sampling times:
00H:Sample 1 time; 080H:Sample 3 times

;*********************************************************************
;*********************************************************************
```

```
;**************** IO Port Definitions ***************/

;**************** Local variable declarations ***************/
Flag0           DATA    20H
B_1ms           BIT     Flag0.0         ; 1ms flag
```

```
B_CanRead       BIT         Flag0.1             CAN Received Data Flag

msecond         DATA        21H                 ;
CAN_ID          DATA        23H                 ;

RX_BUF          DATA        30H
TX_BUF          DATA        38H

TMP_BUF         DATA        40H


;/*************************************************************************
;*************************************************************************
                ORG         0000H               ; program reset entry, compiler
                LJMP        F_Main              automatically defined to 0FF0000H address


                ORG         000BH               ;1 Timer0 interrupt
                LJMP        F_Timer0_Interrupt

                ORG         00E3H               ;28 CAN interrupt
                LJMP        F_CAN_Interrupt


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


;******************** Main Programme **************************/
    ORG             0200H                   ; compiler automatically defines the address 0FF0200H.
F_Main.
                MOV             WTST, #00H          ;Set the program instruction delay parameter.
                                                    ; Assigning 0 sets the CPU to execute instructions as
                                                    fast as possible.
                                ORLP_SW2, #080H     ;Enable access to XFR
                                MOVCKCON, #00H      ;Set external data bus speed to fastest

                                MOVP0M1, #00H       ;Set quasi-
                bidirectional port                  MOVP0M0,
                #00H
                                MOVP1M1, #00H       ;Set quasi-bidirectional port
                                MOVP1M0, #00H
                                MOVP2M1, #00H       ;Set quasi-
                bidirectional port                  MOVP2M0,
                #00H
                                MOVP3M1, #00H       ;Set quasi-bidirectional port
                                MOVP3M0, #00H
                                MOVP4M1, #00H       ;Set quasi-
                bidirectional port                  MOVP4M0,
                #00H
                                MOVP5M1, #00H       ;Set quasi-bidirectional port
                                MOVP5M0, #00H
                                MOVP6M1, #00H       ;Set quasi-
                bidirectional port                  MOVP6M0,
                #00H
                                MOVP7M1, #00H       ;Set quasi-bidirectional port
                                MOVP7M0, #00H

                MOV             SP, #STACK_POIRTER
```

```
MOV     PSW,#0          USING   Select group 0, R0~R7.
USING   #0              Select group 0, R0~R7.

;================== User Initialisation Procedure ===================================
```

```
                CLR         TR0
                            ORLAUXR, #(1 SHL 7)          ; Timer0_1T().
                            ANLTMOD, #NOT 04H            ; Timer0_AsTimer().
                            ANLTMOD, #NOT 03H            ; Timer0_16bitAutoReload();
                            MOVTH0, #Timer0_Reload / 256 ; Timer0_Load(Timer0_Reload);
                            MOVTL0, #Timer0_Reload MOD256
                SETB        ET0                          ;Timer0_InterruptEnable(); ;Timer0_InterruptEnable();
                ;Timer0_InterruptEnable()
                SETB        TR0                          ;Timer0_Run().

                LCALL       F_CAN_Init                   CAN controller initialisation
                SETB        EA                           ;Turn on general interrupt

;=====================================================

                MOV         WR4, #0345H
                MOV         CAN_ID, WR4                  ;Setting the Default
                                                         CAN ID
                MOV         TX_BUF+0, #11H               Setting the default CAN
                                                         data to be sent
                MOV         TX_BUF+1, #22H
                MOV         TX_BUF+2, #33H
                MOV         TX_BUF+3, #44H
                MOV         TX_BUF+4, #55H
                MOV         TX_BUF+5, #66H
                MOV         TX_BUF+6, #77H
                MOV         TX_BUF+7, #88H

;==================== Main loop ================================
L_Main_Loop.
                JNB         B_1ms,L_Main_Loop            ;1ms not arrived
                CLR         B_1ms

;==================== detects if 1000ms to ================================
                MOV         WR6, msecond
                INC         WR6, #1                      ;msecond + 1
                MOV         msecond, WR6
                CMP         WR6, #1000
                JC          L_Quit_Check_1000ms          ;if(msecond < 1000), jmp
                MOV         WR6, #0
                MOV         msecond, WR6                 ;msecond = 0

;================= 1000ms to, process RTC ================================
                MOV         WR6, #0
                MOV         msecond, WR6                 ;msecond = 0

                MOV         A, #SR
                LCALL       F_CAN_ReadReg
                JB          ACC.0,L_CAN_BUSOFF
                LCALL       F_CAN_SendMsg
                LJMP        L_Quit_Check_1000ms
L_CAN_BUSOFF.
                MOV         A, #MR
                MOV         WR6, #WORD0 CANAR
                MOV         WR4, #WORD2 CANAR
                MOV         @DR4, R11
                MOV

                MOV         WR6, #WORD0 CANDR
                MOV         WR4, #WORD2 CANDR
                MOV         R11, @DR4
```

```
ANL     A,#NOT 04H
MOV     @DR4, R11
```

Clear *Reset Mode*, exit from *BUS-OFF* state.

```
L_Quit_Check_1000ms.

                JNBB_CanRead, L_Main_Loop   No CAN bus data received.
        CLR     B_CanRead
        LCALL   F_CAN_ReadMsg               Receive CAN data, replace the original CAN_ID
        with the sent data.
;=============================================
        LJMP    L_Main_Loop

;*********************************************/

;=============================================================================
; Function.    F_CAN_ReadReg
Description.  CAN function register read function.
Parameter:    A: Addr.
Return:       A: Data.
; Version:    V1.0, 2022-04-06
;=============================================================================
F_CAN_ReadReg.
                MOVWR6, #WORD0 CANAR
                MOVWR4, #WORD2 CANAR
                MOV@DR4, R11

                MOVWR6, #WORD0 CANDR
                MOVWR4, #WORD2 CANDR
        MOV
        R11,@DR4 RET

;=============================================================================
; Function.    F_CAN_WriteReg
Description.  CAN function register configuration function.
Parameters:   A: Addr, B: Data.   Data.
Returns.      none.
; Version:    V1.0, 2022-04-06
;=============================================================================
F_CAN_WriteReg.
                MOVWR6, #WORD0 CANAR
                MOVWR4, #WORD2 CANAR
                MOV@DR4, R11

                MOVWR6, #WORD0 CANDR
                MOVWR4, #WORD2 CANDR
                MOV@DR
        4, R10 RET

;=============================================================================
; Function.    F_CAN_ReadMsg
Description.  CAN receive data function.
Parameters:   none.       none.
Returns.      none.
; Version:    V1.0, 2022-04-06
;=============================================================================
F_CAN_ReadMsg.
                MOVA, #RX_BUF0
        CALL    F_CAN_ReadReg
                MOVTMP_BUF+0, A

                MOVA, #RX_BUF1
        CALL    F_CAN_ReadReg
```

```
                MOV     TMP_BUF+1, A

                MOV     A, #RX_BUF2
                CALL    F_CAN_ReadReg
                MOV     TMP_BUF+2, A

                MOV     A, #RX_BUF3
                CALL    F_CAN_ReadReg
                MOV     TX_BUF+0, A             ;RX_BUF+0

                MOV     A, #RX_BUF0
                CALL    F_CAN_ReadReg
                MOV     TX_BUF+1, A             ;RX_BUF+1

                MOV     A, #RX_BUF1
                CALL    F_CAN_ReadReg
                MOV     TX_BUF+2, A             ;RX_BUF+2

                MOV     A, #RX_BUF2
                CALL    F_CAN_ReadReg
                MOV     TX_BUF+3, A             ;RX_BUF+3

                MOV     A, #RX_BUF3
                CALL    F_CAN_ReadReg
                MOV     TX_BUF+4, A             ;RX_BUF+4

                MOV     A, #RX_BUF0
                CALL    F_CAN_ReadReg
                MOV     TX_BUF+5, A             ;RX_BUF+5

                MOV     A, #RX_BUF1
                CALL    F_CAN_ReadReg
                MOV     TX_BUF+6, A             ;RX_BUF+6

                MOV     A, #RX_BUF2
                CALL    F_CAN_ReadReg
                MOV     TX_BUF+7, A             ;RX_BUF+7

                MOV     A, #RX_BUF3
                CALL    F_CAN_ReadReg

                MOV     A, #RX_BUF0
                CALL    F_CAN_ReadReg

                MOV     A, #RX_BUF1
                CALL    F_CAN_ReadReg

                MOV     A, #RX_BUF2
                CALL    F_CAN_ReadReg

                MOV     A, #RX_BUF3
                CALL    F_CAN_ReadReg

;               CanID = ((buffer[1] << 8) + buffer[2]) >> 5;
                        MOVR6, TMP_BUF+1
                        MOVR7, TMP_BUF+2
                SRL      WR6
                SRL      WR6
                SRL      WR6
```

```
        SRL     WR6
        SRL     WR6
        ANL     WR6, #07FFH
        MOV     CAN_ID, WR6             ;Resolve CAN ID
        RET

;=========================================================================
F_CAN_SendMsg   F_CAN_SendMsg
Description:  CAN send data function.
Parameters: none.   none.
Returns:  none.
; Version:  V1.0, 2022-04-06
;=========================================================================
F_CAN_SendMsg.
        MOV     A, #TX_BUF0
        MOV     B, #08H                 ;bit7: Standard frame (0)/Extended frame (1)
                                        ;bit6: Data frame (0)/Remote frame (1)
        CALL    F_CAN_WriteReg          ;bit3~bit0: Data Length (DLC)

        MOV     WR2, CAN_ID
        SLL     WR2
        SLL     WR2
        SLL     WR2
        SLL     WR2
        SLL     WR2
        MOV     A, #TX_BUF1
        MOV     B, R2
        CALL    F_CAN_WriteReg
        MOV     A, #TX_BUF2
        MOV     B, R3
        CALL    F_CAN_WriteReg
        MOV     A, #TX_BUF3
        MOV     B, TX_BUF+0
        CALL    F_CAN_WriteReg

        MOV     A, #TX_BUF0
        MOV     B, TX_BUF+1
        CALL    F_CAN_WriteReg
        MOV     A, #TX_BUF1
        MOV     B, TX_BUF+2
        CALL    F_CAN_WriteReg
        MOV     A, #TX_BUF2
        MOV     B, TX_BUF+3
        CALL    F_CAN_WriteReg
        MOV     A, #TX_BUF3
        MOV     B, TX_BUF+4
        CALL    F_CAN_WriteReg

        MOV     A, #TX_BUF0
        MOV     B, TX_BUF+5
        CALL    F_CAN_WriteReg
        MOV     A, #TX_BUF1
        MOV     B, TX_BUF+6
        CALL    F_CAN_WriteReg
        MOV     A, #TX_BUF2
        MOV     B, TX_BUF+7
```

CALL F_CAN_WriteReg

```
                MOV     A, #TX_BUF3
                MOV     B, #0
                CALL    F_CAN_WriteReg

                MOV     A, #CMR
                MOV     B, #04H                 ; Initiate a
                                                frame transfer
                CALL    F_CAN_WriteReg
                RET

;===========================================================================
; Functions.    F_CAN_Init
Description.    CAN initialisation procedure.
Parameters:     none        none
Returns.        none.
; Version:      V1.0, 2022-04-06
;===========================================================================
F_CAN_Init.
                ORL     auxr2, #02h auxr2,      CAN1 module
                ANL     #not 08h p_sw1, #0      enable
                MOV                             ;Select CAN1
                                                Module

                MOV     A, #MR
                MOV     B, #04H                 Enable Reset
                                                Mode
                CALL    F_CAN_WriteReg

                ;Set baud
                rate MOV        A, #SJW
                ADD     A, #BRP
                MOV     B, A
                MOV     A, #BTR0
                CALL    F_CAN_WriteReg

                MOV     A, #TSG2
                SWAP    A
                ADD     A, #TSG1
                ADD     A, #SAM
                MOV     B, A
                MOV     A, #BTR1
                CALL    F_CAN_WriteReg

                ; Bus Acceptance
                Code Register MOVA,
                #ACR0
                        MOVB, #00H
                CALL     F_CAN_WriteReg
                        MOVA, #ACR1
                        MOVB, #00H
                CALL     F_CAN_WriteReg
                        MOVA, #ACR2
                        MOVB, #00H
                CALL     F_CAN_WriteReg
                        MOVA, #ACR3
                        MOVB, #00H
                CALL     F_CAN_WriteReg

                Bus Acceptance
```

Mask Register

```
                MOVA,
#AMR0
                MOVB, #0FFH
CALL             F_CAN_WriteReg
                MOVA, #AMR1
```

```
        MOV     B, #0FFH
        CALL    F_CAN_WriteReg
        MOV     A, #AMR2
        MOV     B, #0FFH
        CALL    F_CAN_WriteReg
        MOV     A, #AMR3
        MOV     B, #0FFH
        CALL    F_CAN_WriteReg

        MOV     A, #IMR
        MOV     B, #0FFH            ;Interrupt register
        CALL    F_CAN_WriteReg
        MOV     A, #ISR
        MOV     B, #0FFH            ;Clear interrupt flag
        CALL    F_CAN_WriteReg
        MOV     A, #MR
        MOV     B, #00H             ;Exit Reset Mode
        CALL    F_CAN_WriteReg

        MOV     CANICR, #02H        CAN interrupt enable
        RET

;**************** Interrupt function *********************************************

F_Timer0_Interrupt.                 Timer0 1ms interrupt function
        PUSH    PSW                 ;PSW into the stack
        PUSH    ACC                 ;ACC Into the stack

        SETB    B_1ms               ; 1ms flag

        POP     ACC                 ;ACC out of the stack
        POP     PSW                 ;PSW out of the stack
        RETI

;=========================================================================
F_CAN_Interrupt.                    CAN interrupt function
        PUSH    PSW                 ;PSW into the stack
        PUSH    ACC                 ;ACC Into the stack
        PUSH    R4                  ;R4 Into the stack
        PUSH    R5                  ;R5 Into the stack
        PUSH    R6                  ;R6 Into the stack
        PUSH    R7                  ;R7 Into the stack

        MOV     A, #ISR
        MOV     WR6, #WORD0 CANAR
        MOV     WR4, #WORD2 CANAR
        MOV     @DR4, R11

        MOV     WR6, #WORD0 CANDR
        MOV     WR4, #WORD2 CANDR
        MOV     R11,@DR4
        MOV     @DR4, R11           Clear Flag Bit, Write 1 Clear
```

```
                JB          ACC.0,CAN_DOIIF
ISRTEST1:
                JB          ACC.1, CAN_BEIIF
ISRTEST2.
                JB          ACC.2, CAN_TIIF
```

```
isrtest3:
        JB      ACC.3,CAN_RIIF
isrtest4:
        JB      ACC.4,CAN_EPIIF
isrtest5:
        JB      ACC.5,CAN_EWIIF
isrtest6.
        JB      ACC.6, CAN_ALIIF

ISREXIT.
        POP     R7          ;R7 Out of the stack
        POP     R6          ;R6 Out of the stack
        POP     R5          ;R5 Out of the stack
        POP     R4          ;R4 Out of stack
        POP     ACC         ;ACC out of the stack
        POP     PSW         ;PSW out of the stack
        RETI

CAN_DOIIF.
;       ORL     A,#01H      Clear Flag Bit, Write 1 Clear
;       MOV     @DR4, R11
        JMP     ISRTEST1

CAN_BEIIF.
;       ORL     A,#02H      Clear Flag Bit, Write 1 Clear
;       MOV     @DR4, R11
        JMP     ISRTEST2

CAN_TIIF.
;       ORL     A,#04H      Clear Flag Bit, Write 1 Clear
;       MOV     @DR4, R11
        JMP     ISRTEST3

CAN_RIIF.
;       ORL     A,#08H      Clear Flag Bit, Write 1 Clear
;       MOV     @DR4, R11
        SETB    B_CanRead
        JMP     ISRTEST4

CAN_EPIIF.
;       ORL     A,#010H ;   Clear Flag Bit, Write 1 Clear
;       MOV     @DR4, R11
        JMP     ISRTEST5

CAN_EWIIF.
;       ORL     A,#020H     Clear Flag Bit, Write 1 Clear
;       MOV     @DR4, R11
        JMP     ISRTEST6

CAN_ALIIF.
;       ORL     A,#040H     Clear Flag Bit,
```

Write *1* Clear

```
;                   MOV         @DR4, R11
                    JMP         ISREXIT

;=======================================================================
                    END
```

30 LIN bus

The STC32G series microcontrollers have an integrated LIN bus function unit that supports LIN2.1 and LIN1.3 protocols.

The main functions are as follows:

- Automatic frame header processing
- Can be switched between master and slave modes
- timeout detection
- error analysis

30.1 LIN Function Pin Switching

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
P_SW1	A2H	S1_S[1:0]		CAN_S[1:0]		SPI_S[1:0]		**LIN_S[1:0]**	

LIN_S[1:0]: LIN function pin select bits

LIN_S[1:0]	LIN_RX	LIN_TX
00	P0.2	P0.3
01	P5.2	P5.3
10	P4.6	P4.7
11	P7.2	P7.3

30.2 LIN-related registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
LINICR	LINBUS Interrupt Control Register	F9H	-	-	-	-	PLINH	LINIF	LINIE	PLINL	0000,0000
LINAR	LINBUS Address Register	FAH									0000,0000
LINDR	LINBUS Data Register	FBH									0000,0000
AUXR2	Auxiliary Register 2	97H	-	-	-	-	CANSEL	CAN2EN	CANEN	LINEN	xxxx,0000

30.2.1 Auxiliary Register 2 (AUXR2)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
AUXR2	97H	-	-	-	-	CANSEL	CAN2EN	CANEN	LINEN

LINEN: LIN bus enable control bit

0: Disable LIN function

1: Enable LIN function

30.2.2 LIN Bus Interrupt Control Register (LINICR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LINICR	F9H	-	-	-	-	PLINH	LINIF	LINIE	PLINL

LINIE: LIN bus interrupt enable control bit

0: Disable LIN interrupt

1: Enable LIN interrupt

LINIF: LIN bus interrupt request flag bit, hardware clear (read LSR clear).

PLINH, PLINL: LIN interrupt priority control bits

PLINH	PLINL	priority
0	**0**	**0 (minimum)**
0	1	1
1	0	2
1	1	3 (maximum)

30.2.3 LIN Bus Address Register (LINAR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LINAR	FAH								

30.2.4 LIN Bus Data Register (LINDR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LINDR	FBH								

Reading and writing to the LIN internal function registers requires

indirect access to the **LIN internal function registers** via LINAR and

LINDR:

1. Write the address of the LIN internal function register to the LIN bus address register LINAR.
2、Read LIN bus data register LINDR

For example, you need to read the value of the LIN internal function register LSR.

```
LINAR = 0x05;//write the address of the LSR to LINAR
dat = LINDR;              //Read CANDR to get the
```

value of LSR **Write LIN internal function register method:**

1. Write the address of the LIN internal function register to the LIN bus address register LINAR.
2. Write the value to be written to the LIN bus data register LINDR.

Example: Data 0x5a needs to be written to the LIN

```
internal function registerLBUF LINAR = 0x00;
                          //write the address of LBUF
toLINAR LINDR = 0x5a;     //write the value to be
written 0x5a to LINDR
```

30.3 LIN Internal Function Register

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
08H	HDRL	HDRH	HDP					
00H	LBUF	LSEL	LID	LER	LIE	LSR	DLL	DLH

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LBUF	0x00	LBUF							
LSEL	0x01	ANIC				INDEX			
LID	0x02			ID					

LER	0x03		OVER	SYNCER	TOVER	CHKER	PER	BITER	FER
LIE	0x04					ABORTE	ERRE	RDYE	LIDE
LSR	0x05	LOG SIZE				ABORT	ERR	RDY	LID
LCR	0x05	MODE	LIN13	SIZE				CMD	
DLL	0x06	DLL							
DLH	0x07	SYNC	DLH						
HDRL	0x08	HDRL							
HDRH	0x09	HDRH							
HDP	0x0A			HDP					

30.3.1 LIN Data Register (LBUF)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LBUF	0x00								

LBUF is the data interface to the internal data FIFO of the LIN module.

30.3.2 LIN Data Address Register (LSEL)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LSEL	0x01	AINC	-	-	-	INDEX[3:0]			

AINC Address Increment

0: The address of the data written to the LIN internal FIFO is determined by INDEX.

1: The data address is self-incremented, every time LBUF is operated, the data address of LIN internal FIFO is automatically incremented by one.

30.3.3 LIN Frame ID Register (LID)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LID	0x02			ID[5:0]					

LID Frame ID, if LCR[5:2]=4'b1111, then the combination of LID[5:4] represents the amount of data in the data frame: 00: 2 bytes

01: 2 bytes

10:4 bytes

11:8 bytes

30.3.4 LIN Error Register (LER)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LER	0x03	-	OV	SYN	TOV	CHK	PER	BIT	FER

OV: over run error. The user sends a new command to the LIN while the LIN is running a command (RDY is low). SYN: Only available in slave mode, an error is generated when the LIN receives a frame synchronisation.

TOV: timeout error. The LIN did not receive a complete data frame within the maximum allowed time.

CHK: checksum error.

PER: Parity error, the protected ID segment was not received correctly.

BIT: Bit error, signalling that the data bits sent by the LIN do not correspond to the status monitored by the bus.

FER: Frame Error, received data did not contain a valid stop bit. Read the clear error register.

30.3.5 LIN Interrupt Enable Register (LIE)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LIE	0x04	-	-	-	-	ABORTE	ERRE	RDYE	LIDE

ABORTE: Enable termination interrupt.

ERRE: Enable the Error interrupt. RDYE: enable Ready interrupt. LIDE: Enable head interrupt.

30.3.6 LIN Status Register (Read-Only Register) (LSR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LSR	0x05	LOG SIZE[3:0]				ABORT	ERR	RDY	LID

LOG SIZE: The length of the detected data in LOG mode.

ABORT: Abort command is in progress. ERR: Error status.

RDY: Ready status, can execute new command. LID: Correct header is received. Bit0~Bit3 Clear after reading.

30.3.7 LIN Control Register (Write-Only Register) (LCR)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
LCR	0x05	MODE	LIN13	SIZE[3:0]				CMD[1:0]	

MODE: LIN mode selection

0: Slave mode

1: Master mode.

LIN13: Checksum selection.

0: Enhanced checksum, LIN2.1 protocol.

1: Common checksum, LIN 1.3 protocol.

SIZE[3:0]: data length

set up	lengths	set up	lengths
0000	0 bytes	1000	8 bytes
0001	1 byte	1001	reservation
0010	2 bytes	1010	reservation
0011	3 bytes	1011	reservation
0100	4 bytes	1100	reservation
0101	5 bytes	1101	reservation
0110	6 bytes	1110	LOG MODE
0111	7 bytes	1111	Data length is determined by LID[5:4].

CMD[1:0]: LIN command.

00: Abort command.

01: Master Mode Send header command.

10: TX response.

11: RX response.

30.3.8 LIN Baud Rate Register (DLL/DLH)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DLL	0x06	DLL							
DLH	0x07	SYNC	DLH						

The DLL and DLH determine the baud rate of the LIN module. Baud rate = SYSclk/16/DL. SYNC: Synchronisation mode, only useful in slave mode.

30.3.9 LIN Header Delay Counter Register (HDRL/HDRH)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HDRL	0x08	HDRL							
HDRH	0x09	HDRH							

Delay (ms) = (HDR*1000)/ SYSclk.

30.3.10 LIN Header Delay Division Register (HDP)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
HDP	0x0A			HDP					

HDR and HDP together define a header delay time, which is used to define the time delay between when the software is configured to send the header and when the LIN module actually sends the header.

30.4 sample procedure

30.4.1 LIN bus host transceiver example

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                   // see download software for header files
#include "intrins.h"

typedef     unsigned char   u8;
typedef     unsigned int    u16.
typedef     unsigned long   u32.

#define MAIN_Fosc           24000000UL

sbit SLP_N = P5^3;                                    /LIN transceiver control pin 0:
                                                      sleep; 1: active

#define LIN_MODE            1                         //0: LIN2.1; 1 : LIN1.3
#define FRAME_LEN           8                         //Data length: 8 bytes

u8 Lin_ID.
u8 TX_BUF[8].

u8 Key1_cnt;
u8 Key2_cnt;
bit Key1_Flag.
bit Key2_Flag.

void LinInit().
void LinSendMsg(u8 lid, u8 *pdat);
void LinSendHeader(u8 lid).
void delay_ms(u8 ms).

void main(void)
{
        EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                                     //Set the external data bus speed to fastest
    WTST = 0x00;                                      //set the program code wait parameter.
                                                      //Assign a value of 0 to set the CPU to execute the
                                                      programme as fast as possible.

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

    P_SW2 |= 0x80.
    P0PU = 0x0c.                                      //LIN_TX, LIN_RX pin enable internal pull-ups
    P_SW2 &= ~0x80.
```

```
    Lin_ID = 0x32;
```

```
    LinInit().
    EA = 1;                                          //Turn on the general interrupt

    SLP_N = 1;
    TX_BUF[0] = 0x11;
    TX_BUF[1] = 0x22;
    TX_BUF[2] = 0x33;
    TX_BUF[3] = 0x44;
    TX_BUF[4] = 0x55;
    TX_BUF[5] = 0x66;
    TX_BUF[6] = 0x77;
    TX_BUF[7] = 0x88.

    while(1)
    {
        delay_ms(1);
        if(!P32)
        {
            if(!Key1_Flag)
            {
                Key1_cnt++;
                if(Key1_cnt > 50)                    //50ms anti-dithering
                {
                    P46 = ~P46.
                    Key1_Flag = 1;
                    LinSendMsg(Lin_ID, TX_BUF).      //Host sends complete frame
                }
            }
        }
        else
        {
            Key1_cnt = 0;
            Key1_Flag = 0;
        }

        if(!P33)
        {
            if(!Key2_Flag)
            {
                Key2_cnt++;
                if(Key2_cnt > 50)                    //50ms stabilisation
                {
                    P47 = ~P47.
                    Key2_Flag = 1;
                    LinSendHeader(0x13); //Host sends Header, answer data sent by slave with specific identifier,
                                         put together into a complete frame
            }   }
        }
        else
        {
            Key2_cnt = 0;
            Key2_Flag = 0;
        }
    }
}

void delay_ms(u8 ms)
{
     u16 i;
```

```
    do{
        i = MAIN_Fosc / 6000;
        while(--i).
    }while(---ms).
}

u8 LinReadReg(u8 addr)
{
    u8 dat.
    LINAR = addr;
    dat = LINDR;
    return dat;
}

void LinWriteReg(u8 addr, u8 dat)
{
    LINAR = addr;
    LINDR = dat.
}

void LinReadMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL,0x80).                          //address increment, start from 0
    for(i=0;i<FRAME_LEN;i++)
    {
        pdat[i] = LinReadReg(LBUF).
    }
}

void LinSetMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL,0x80).                          //address increment, start from 0
    for(i=0;i<FRAME_LEN;i++)
    {
        LinWriteReg(LBUF,pdat[i]).
    }
}

void LinSetID(u8 lid)
{
    LinWriteReg(LID,lid).                            //Set the bus ID
}

u8 GetLinError(void)
{
    u8 sta.
    sta = LinReadReg(LER);                           //read clear error
    register return sta;
}

u8 WaitLinReady(void)
{
    u8 lsr;
    do{
        lsr = LinReadReg(LSR);
```

```
    }while(! (lsr & 0x02)); //Judge the ready state.          // Determine the ready state
    return lsr;
}

void SendAbortCmd(void)
{
    LinWriteReg(LCR,0x80).                                    //Main Mode Send Abort
}

void SendHeadCmd(void)
{
    LinWriteReg(LCR,0x81).                                    //Main Mode Send Header
}

void SendDatCmd(void)
{
    u8 lcr_val.
    lcr_val = 0x82+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val).
}

void ResponseTxCmd(void)
{
    u8 lcr_val.
    lcr_val = 0x02+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val).
}

void ResponseRxCmd(void)
{
    u8 lcr_val.
    lcr_val = 0x03+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val).
}

void LinSendMsg(u8 lid, u8 *pdat)
{
    LinSetID(lid).                                            //Set the bus ID
    LinSetMsg(pdat).

    SendHeadCmd();                                            //Main Mode Send Seader
    WaitLinReady();                                           //Wait for ready state
    GetLinError();                                            //Read and clear the error register

    SendDatCmd();                                             //Send Data
    WaitLinReady();                                           //Wait for ready state
    GetLinError();                                            //Read and clear the error register
}

void LinSendHeader(u8 lid)
{
    LinSetID(lid);                                            //Set the ID of the slave bus that sends the response.

    SendHeadCmd();                                            //main mode send header
    WaitLinReady();                                           //Wait for ready state
    GetLinError();                                            //Read and clear the error register

    ResponseRxCmd().                                          //RX response
    WaitLinReady();                                           //Wait for ready state
```

```
    GetLinError();                                  //Read and clear the error register

    LinReadMsg(TX_BUF); //Receive answer data from Lin bus slave.    //Receive the answer data sent by the slave of
    Lin bus.
}

void LinSetBaudrate(u16 brt)
{
    u16 tmp.
    tmp = (MAIN_Fosc >> 4) / brt;
    LinWriteReg(DLH,(u8)(tmp>>8)); //set baud rate          //set baud
    rate LinWriteReg(DLL,(u8)tmp);
}

void LinSetHeadDelay(u8 base_ms, u8 prescaler)
{
    u16 tmp.
    tmp = (MAIN_Fosc * base_ms) / 1000; // Note the range of base_ms.  //Note the range of base_ms, do not exceed the
    16-bit limit.
    LinWriteReg(HDRH,(u8)(tmp>>8));
    LinWriteReg(HDRL,(u8)tmp).                      //Set the frame header delay count

    LinWriteReg(HDP,prescaler).                     //Set the frame header delay frequency division
}

void LinInit()
{
    P_SW1 = 0x00.                                   //select P0.2,P0.3
    LINICR = 0x02.                                  //LIN module interrupt enable
    AUXR2 |= 0x01.                                  //LIN module is enabled

    GetLinError();                                  //Read and clear the error register
    LinWriteReg(LIE,0x00).                          //LIE Interrupt Enable Register
    LinSetBaudrate(9600).                           //Set the baud rate
    LinSetHeadDelay(0x01,0x00);                     //Set the frame head delay
}
```

30.4.2 LIN bus slave transceiver example

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                 //see download software for header files
#include "intrins.h"

typedef     unsigned char   u8;
typedef     unsigned int    u16.
typedef     unsigned long   u32.

#define MAIN_Fosc           24000000UL

sbit SLP_N = P5^3;                                  /LIN transceiver control pin 0:
                                                    sleep; 1: active

#define LIN_MODE            1                       //0: LIN2.1; 1 : LIN1.3
```

```
#define FRAME_LEN       8                   //Data length: 8 bytes

u8 Lin_ID.
u8 TX_BUF[8].
```

```
bit RxFlag.

void LinInit().
void LinReadMsg(u8 *pdat);
void  ResponseRxCmd(void);
u8 WaitLinReady(void).
u8 GetLinError(void).

void main(void)
{
        EAXFR = 1;      //Enable access to XFR
    CKCON = 0x00;                                   //Set the external data bus speed to fastest
    WTST = 0x00;                                    //set the program code wait parameter.
                                                    //Assign a value of 0 to set the CPU to execute the
                                                    programme as fast as possible.

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

    P_SW2 |= 0x80.
    P0PU = 0x0c.                                    //LIN_TX, LIN_RX pin enable internal pull-ups
    P_SW2 &= ~0x80.

    Lin_ID = 0x32;
    LinInit().
    EA = 1;                                         //Turn on the general interrupt

    SLP_N = 1;
    TX_BUF[0] = 0x11;
    TX_BUF[1] = 0x22;
    TX_BUF[2] = 0x33;
    TX_BUF[3] = 0x44;
    TX_BUF[4] = 0x55;
    TX_BUF[5] = 0x66;
    TX_BUF[6] = 0x77;
    TX_BUF[7] = 0x88.

    while(1)
    {
        if(RxFlag == 1)
        {
            ResponseRxCmd().                        //RX response
            WaitLinReady();                         //Wait for ready state
            GetLinError();                          //Read and clear the error register

            LinReadMsg(TX_BUF); //Receive Lin bus data.     //Receive Lin bus data
            RxFlag = 0;
        }
    }
}

u8 LinReadReg(u8 addr)
{
    u8 dat.
```

```
    LINAR = addr;
    dat = LINDR;
    return dat;
}

void LinWriteReg(u8 addr, u8 dat)
{
    LINAR = addr;
    LINDR = dat.
}

void LinReadMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL,0x80).                          //address increment, start from 0

    for(i=0;i<FRAME_LEN;i++)
    {
        pdat[i] = LinReadReg(LBUF).
    }
}

void LinSetMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL,0x80).                          //address increment, start from 0
    for(i=0;i<FRAME_LEN;i++)
    {
        LinWriteReg(LBUF,pdat[i]).
    }
}

void LinSetID(u8 lid)
{
    LinWriteReg(LID,lid).                            //Set the bus ID
}

u8 GetLinError(void)
{
    u8 sta.
    sta = LinReadReg(LER);                           //read clear error
    register return sta;
}

u8 WaitLinReady(void)
{
    u8 lsr;
    do{
        lsr = LinReadReg(LSR);
    }while(! (lsr & 0x02)); // determine ready state  // Determine the ready state
    return lsr;
}

void SendAbortCmd(void)
{
    LinWriteReg(LCR,0x80).                           //Main Mode Send Abort
}
```

```
void SendHeadCmd(void)
{
    LinWriteReg(LCR,0x81).                          //Main Mode Send Header
}

void SendDatCmd(void)
{
    u8 lcr_val.
    lcr_val = 0x82+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val).
}

void ResponseTxCmd(void)
{
    u8 lcr_val.
    lcr_val = 0x02+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val).
}

void ResponseRxCmd(void)
{
    u8 lcr_val.
    lcr_val = 0x03+(LIN_MODE<<6)+(FRAME_LEN<<2);
    LinWriteReg(LCR,lcr_val).
}

void LinTxResponse(u8 *pdat)
{
    LinSetMsg(pdat).
    ResponseTxCmd().                                //TX response
    WaitLinReady();                                 //Wait for ready state
    GetLinError();                                  //Read and clear the error register
}

void LinSetBaudrate(u16 brt)
{
    u16 tmp.
    tmp = (MAIN_Fosc >> 4) / brt;
    LinWriteReg(DLH,(u8)(tmp>>8));
    LinWriteReg(DLL,(u8)tmp).
}

void LinSetHeadDelay(u8 base_ms, u8 prescaler)
{
    u16 tmp.
    tmp = (MAIN_Fosc * base_ms) / 1000; // Note the range of base_ms.  //Note the range of base_ms, do not exceed the
    16-bit limit.
    LinWriteReg(HDRH,(u8)(tmp>>8));
    LinWriteReg(HDRL,(u8)tmp).                      //Set the frame header delay count

    LinWriteReg(HDP,prescaler).                     //Set the frame header delay frequency division
}

void LinInit()
{
    P_SW1 = 0x00.                                   //select P0.2,P0.3
    LINICR = 0x02.                                  //LIN module interrupt enable
    AUXR2 |= 0x01.                                  //LIN module is enabled
```

```
    GetLinError();                                  //Read and clear the error register
    LinWriteReg(LIE,0x0F).                          //LIR Interrupt Enable Register
    LinSetBaudrate(9600).                           //Set the baud rate
    LinSetHeadDelay(0x01,0x00);                     //Set the frame head delay
}

void LinBUS_Interrupt(void) interrupt LIN_VECTOR
{
    u8 isr.

    isr = LinReadReg(LSR);
    if((isr & 0x03) == 0x03)                        //Receive Header, in Ready state.
    {
        isr = LinReadReg(LER);
        if(isr == 0x00)                             //no error generated
        {
            P46 = ~P46.

            isr = LinReadReg(LID);
            if(isr == 0x12)                         //determine if slave responds with ID
            {
                LinTxResponse(TX_BUF).              //return response data
            }
            else
            {
                RxFlag = 1;
            }
        }
    }
    else
    {
        isr = LinReadReg(LER).                      //read clear error register
    }
}
```


30.4.3 Example of LIN bus emulation using a serial port

LIN (Local Interconnect Network) bus is a low-cost serial communication protocol based on UART/SCI (Universal Asynchronous Transceiver/Serial Interface).The byte field of LIN is in 8N1 format like UART, but the interval field of LIN is 13 consecutive significant levels, which is inconsistent with the 8 bits of the serial port. Here, we output 13 significant level width signals as the interval field by switching the baud rate.

Take 9600 baud rate as an example:
13 dominant signal times = 13/9600 = 1354us
Converts to sending a byte 0x00, 1 start bit + 8 data bits + 1 stop bit, which contains 9 significant levels. Conversion baud rate = 9/1354us = 6647 baud rate

Test Methods of this routine: UART1 connects to the computer through the serial tool; UART2 connects to the LIN bus with an external LIN transceiver. Forward the data sent from the computer serial port to the LIN bus; forward the data received from the LIN bus to the computer serial port.

//Tested operating frequency is *11.0592MHz*

```
//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files
#include "intrins.h"

typedef     unsigned char    u8.
typedef     unsigned int     u16;
typedef     unsigned long    u32.

#define MAIN_Fosc              24000000UL

#define Baudrate1              (65536UL - (MAIN_Fosc / 4) / 9600UL)
#define Baudrate2              (65536UL - (MAIN_Fosc / 4) / 9600UL) //Send Data Transfer Baud Rate

#define Baudrate_Break         (65536UL- (MAIN_Fosc / 4) / 6647UL) // Transmit Significant
Interval Signal Baud Rate #define       UART1_BUF_LENGTH32
#define UART2_BUF_LENGTH       32

#define LIN_ID       0x31

sbit SLP_N =  P2^4.                                    //LIN transceiver control pin 0: sleep; 1: active

u8 TX1_Cnt.                                            //Transmit count
u8 RX1_Cnt.                                            //Receive count
u8 TX2_Cnt.                                            //Transmit count
u8 RX2_Cnt.                                            //Receive count
bit B_TX1_Busy.                                        //transmit busy flag
bit B_TX2_Busy.                                        //transmit busy flag
u8 RX1_TimeOut.
u8 RX2_TimeOut.

u8 xdata RX1_Buffer[UART1_BUF_LENGTH];                 //Receive buffer
u8 xdata RX2_Buffer[UART2_BUF_LENGTH];                 //Receive buffer

void UART1_config(void);
void UART2_config(void);
void delay_ms(u8 ms).
void UART1_TxByte(u8 dat);
void UART2_TxByte(u8 dat);
void Lin_Send(u8 *puts).
void SetTimer2Baudraye(u16 dat).

void main(void)
{
    u8 i;

    CKCON = 0x00;                                      //Set the external data bus speed to fastest
    WTST = 0x00;                                       //set the program code wait parameter.
                                                       //Assign a value of 0 to set the CPU to execute the
                                                       programme as fast as possible.

    p0m1 = 0;    p0m0 = 0; //Set quasi-bidirectional port.    //Set quasi-bidirectional port
    p1m1 = 0;    p1m0 = 0; //Set quasi-bidirectional port.    //Set quasi-bidirectional port
    p2m1 = 0;    p2m0 = 0; //Set quasi-bidirectional port.    //Set quasi-bidirectional port
    p3m1 = 0;    p3m0 = 0; //Set quasi-bidirectional port.    //Set quasi-bidirectional port
    p4m1 = 0;    p4m0 = 0; //Set the quasi-bidirectional port.//Set quasi-bidirectional port
    p5m1 = 0;    p5m0 = 0; //Set quasi-bidirectional port.    //Set quasi-bidirectional port
    p6m1 = 0;    p6m0 = 0; //Set quasi-bidirectional port.    //Set quasi-bidirectional port
    p7m1 = 0;    p7m0 = 0; //Set quasi-bidirectional port.    //Set quasi-bidirectional port
```

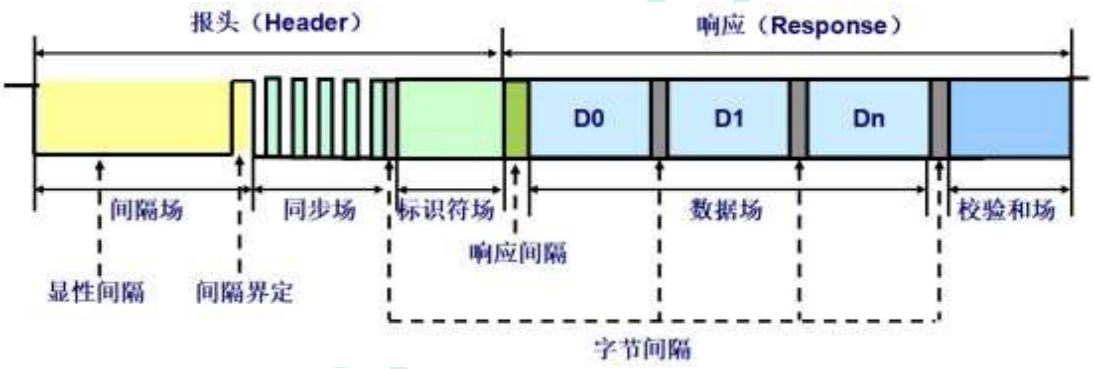
```
    UART1_config();
    UART2_config().
    EA = 1;                                         // Allow global interrupt
    SLP_N = 1;

    while(1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)
        {
            if(--RX1_TimeOut == 0)                  //Timeout, serial port reception is finished.
            {
                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)                  //Timeout, serial port reception is finished.
            {
                if(RX2_Cnt > 0)
                {
                    for (i=0; i < RX2_Cnt; i++)     // encounter stopper 0 end
                    {
                        UART1_TxByte(RX2_Buffer[i]); //The data received from the LIN bus is sent to UART1.   //The
                        data received from the LIN bus is sent to UART1.
                    }
                }
                RX2_Cnt = 0;
            }
        }
    }
}

void delay_ms(u8 ms)
{
    u16 i;
    do{
        i = MAIN_Fosc / 6000;
        while(--i).
    }while(---ms).
}

u8 Lin_CheckPID(u8 id)                              //ID to PID
{
    u8 pid;
    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 ;

    pid= id|P0|P1 ;

    return pid.
}
```

```
static u8 LINCalcChecksum(u8 *dat)                  // Calculate the LIN1.3 classical checksum.
{
    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;
}

void Lin_SendBreak(void)
{
    SetTimer2Baudraye((u16)Baudrate_Break);
    UART2_TxByte(0);
    SetTimer2Baudraye((u16)Baudrate2).
}

void Lin_Send(u8 *puts)
{
    u8 i;

    Lin_SendBreak().                                //Send frame interval field
    UART2_TxByte(0x55).                             //Send the synchronisation field
    UART2_TxByte(Lin_CheckPID(LIN_ID));             //transmit identifier
    for(i=0;i<8;i++)
    {
        UART2_TxByte(puts[i]).
    }
    UART2_TxByte(LINCalcChecksum(puts)).
}

void UART1_TxByte(u8 dat)
{
    SBUF = dat;
    B_TX1_Busy = 1;
    while(B_TX1_Busy).
}

void UART2_TxByte(u8 dat)
{
    S2BUF = dat;
    B_TX2_Busy = 1;
    while(B_TX2_Busy).
}

void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                                //Timer stop
    AUXR &= ~(1<<3); //Timer2 Set As Timer          //Timer2 set As Timer
    AUXR |= (1<<2); //Timer2 set as 1T mode.        //Timer2 set as 1T mode
    T2H = dat / 256.
```

```
    T2L = dat % 256.
    IE2 &= ~(1<<2).                                 // disable interrupt
    AUXR |= (1<<4); //Timer run enable.             //Timer run enable
}

void UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01.                                  //S1 BRT Use Timer1.
    AUXR |= (1<<6); //Timer1 set as 1T mode.        //Timer1 set as 1T mode
    TMOD &= ~(1<<6); //Timer1 Set As Timer          //Timer1 set As Timer
    TMOD &= ~0x30;                                  //Timer1_16bitAutoReload;
    TH1 = (u8)(Baudrate1 / 256);
    TL1 = (u8)(Baudrate1 % 256).
    ET1 = 0;                                        // disable interrupt
    INTCLKO &= ~0x02.                               //no clock output
    TR1 = 1;

    SCON = (SCON & 0x3f) | 0x40; //UART1 mode 8-bit data, variable baud rate.     //UART1 mode 8-bit data, variable baud rate.
    ES =  1;                                        // Allow interrupt
    REN = 1;                                        // Allow to receive
    P_SW1 &= 0x3f.                                  //UART1 switches to P3.0 P3.1.

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

void UART2_config(void)
{
    SetTimer2Baudraye((u16)Baudrate2);
    S2CON &= ~(1<<7); // 8-bit data, 1 start bit, 1 stop bit, no parity.       // 8-bit data, 1 start bit, 1 stop bit, no parity
    IE2 |= 1;                                       //Allow interrupt
    S2CON |= (1<<4).                                // Allow to receive
    P_SW2 &= ~0x01.                                 //UART2 switch to: 0: P1.0 P1.1, 1 : P4.6 P4.7

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

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;
    }
}
```

```
void UART2_int (void) interrupt 8
{
    if((S2CON & 1) ! = 0)
    {
        S2CON &= ~1;                                    //clear flag bit
        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 flag bit
        B_TX2_Busy = 0;
    }
}
```

30.4.4 LIN Bus Timing Introduction



1. Byte field

1) UART/SCI based communication format;
2) It takes 10 bit times (TBIT) to send a byte

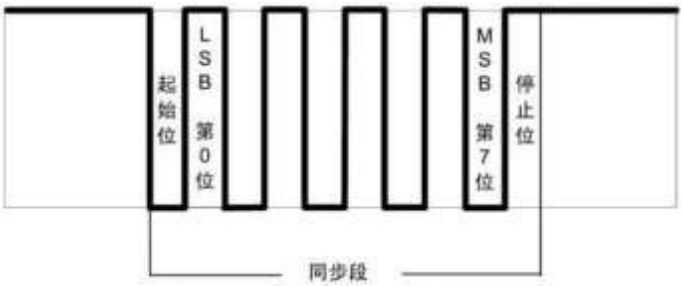
2. Interval field

1) Indicates the start of a frame message, sent by the master node;
2) The spacer signal consists of at least 13 dominant bits;
3) The interval definer consists of at least 1 invisible bit;
4) The spacer field is the only field that does not conform to the byte field format;
5) The slave node needs to detect at least 11 consecutive dominant bits to be considered an interval signal;

3. Synchronised field

1) Ensure that all slave nodes send and receive data using the same baud rate as the node;

2) One byte with fixed structure: 0x55;



4. Identifier field

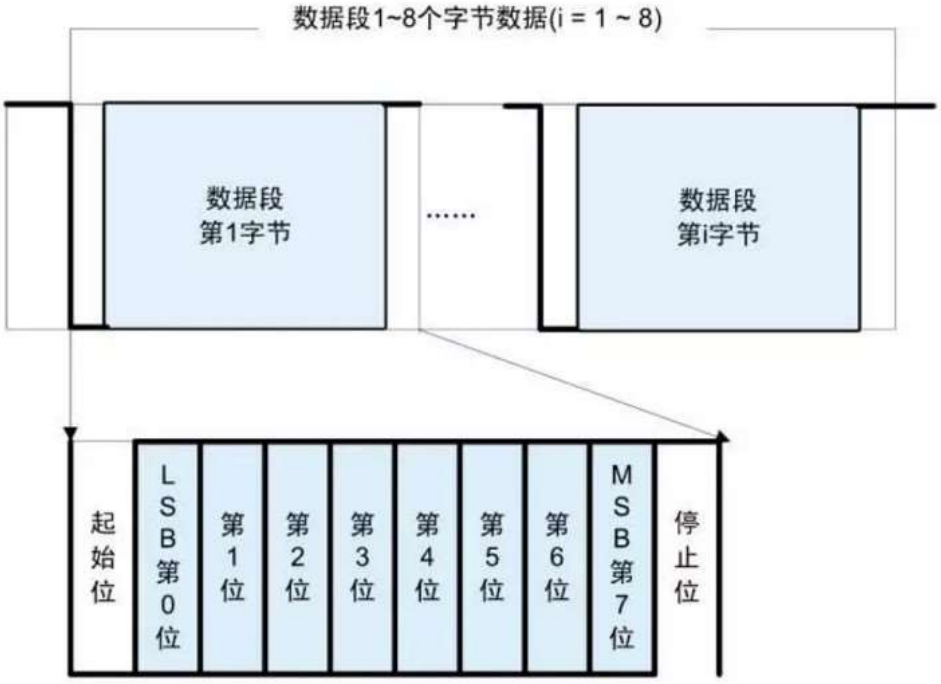
1) The ID ranges from 0 to 63 (0x3f);
2) Parity Character (Parity) P0,P1

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$$

$$P1 = \neg (ID1 \oplus ID3 \oplus ID4 \oplus ID5)$$

5. Data field

1) Data field length 1 to 8 bytes;
2) Low byte first, low first;
3) If a signal is more than 1 byte long it is sent with the lower bit first (little end);



6. Calibration and field

Used to verify that the received data is correct

1) Classic Checksum (Classic Checksum) Checks the data field only (LIN1.3) 2) Enhance Checksum (Enhance Checksum) Checks the identifier field against the content of the data field (LIN2.0, LIN2.1) Frames with identifiers 0x3C and 0x3D can only use Classic Checksum.

Calculation method: Invert 8 bits and sum

31 32-bit hardware multiplication and division unit (MDU32)

The Multiply and Divide Unit (known as the MDU32) provides fast 32-bit arithmetic operations.The MDU32 supports unsigned and complementary signed integer operands.The MDU32 is controlled by a dedicated Direct Memory Access Module (known as the DMA).All MDU32 arithmetic operations are initiated by writing DMA instructions to register DMAIR. All MDU32 arithmetic operations are initiated by writing DMA instructions to the DMA control in register DMAIR. The operands and results of all arithmetic operations performed by the MDU32 module are located in registers R0-R7.

Attention:

1. the execution time required by the DMA module to perform arithmetic operations, including:

- Operands are loaded from DR0-DR4 registers into the MDU32 module
- MDU32 Arithmetic Operations
- Result storage from MDU32 module to R0-R7 registers

2. the execution time required by the processor to execute C compiled arithmetic functions, including:

- DMA instructions are written to the DMAIR register.
- Operands are loaded from DR0-DR4 registers into the MDU32 module
- MDU32 Arithmetic Operations
- Result storage from MDU32 module to R0-R7 registers
- Return from function (RET instruction)

When MDU32 performs multiplication and division operation, the MCU will automatically switch to IDLE mode, i.e., the CPU stops the clock instruction, while other peripherals continue to work. After the operation is completed, the microcontroller automatically switches to normal operation mode.

31.1 Related Special Function Registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMAIR	DMA Instruction Register	EDH	DMAIR[7:0]								0000,0000

Note: To write instruction code to DMAIR register, you can only use the instruction "MOV DMAIR,#N" in immediate addressing mode, using other instructions will not trigger the calculation properly.

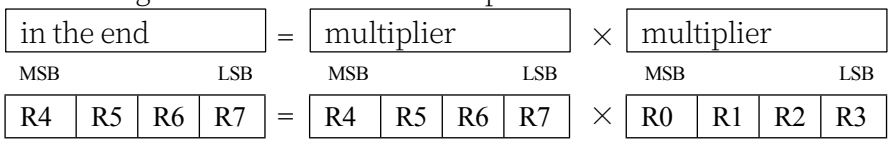
31.2 Operational implementation schedule

MDU operation	DMA command code		Number of execution clocks
	HEX	DEC	
32-bit multiplication	0x02	2	3
32-bit unsigned division	0x04	4	19
32-bit signed division	0x06	6	21

31.3 MDU32 Arithmetic Operations

31.3.1 32-bit multiplication

32-bit multiplication operations are performed on two unsigned or signed complementary integer arguments. The first parameter is located in the R4-R7 registers and the second parameter is located in the R0-R3 registers. The result of the operation is stored into the R4-R7 registers.

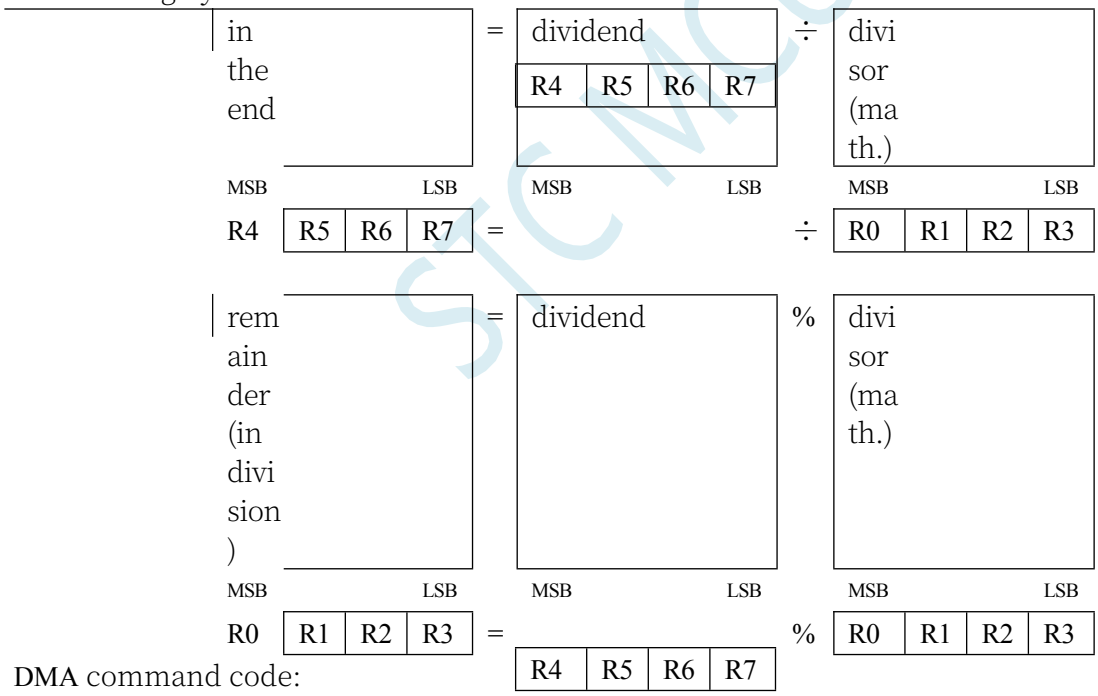


DMA command code: 0x02

Execution time: 3clk

31.3.2 32-bit unsigned division

Performs 32-bit unsigned division on two unsigned integer parameters. The first parameter "Dividend" is located in R4-R7 registers and the second parameter "Divisor" is located in R0-R3 registers. The result is stored in R4-R7 registers. The remainder is returned in R0-R3. Dividing by zero returns 0xFFFFFFFF.



DMA command code: 0x04

Execution time: 19clk

31.3.3 32-bit signed division

Performs 32-bit signed division on two signed complement parameters. The first parameter "Dividend" is located in R4-R7 registers and the second parameter "Divisor" is located in R0-R3 registers. The result is stored in R4-R7 registers. The remainder is returned in R0-R3. Dividing by zero returns 0xFFFFFFFF.

R4 R5 R6 R7

in the end				=	dividend		÷	divisor (math.)				
MSB			LSB		MSB	LSB		MSB			LSB	
R4	R5	R6	R7	=			÷	R0	R1	R2	R3	

remainder (in division)				=	dividend		%	divisor (math.)				
MSB			LSB		MSB	LSB		MSB			LSB	
R0	R1	R2	R3	=			%	R0	R1	R2	R3	

DMA command code:

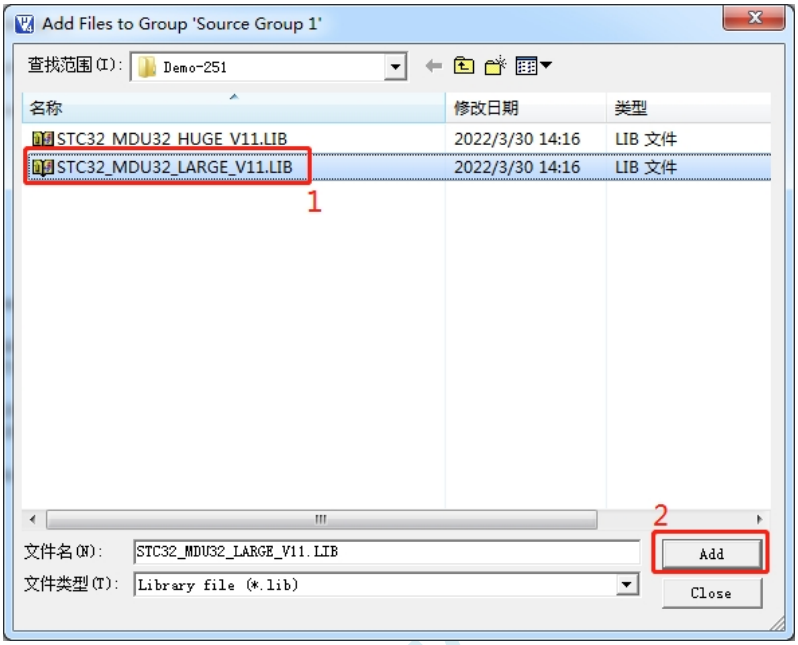
0x06

Execution time: 21clk

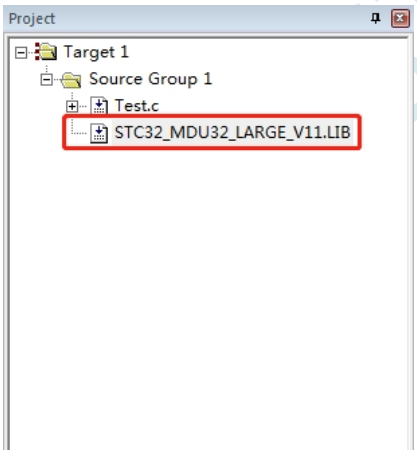
31.4 sample procedure

To use a 32-bit hardware multiply-divide unit, simply add to your keil project the The library file "STC32_MDU32_LARGE_Vxx.LIB" or "STC32_MDU32_HUGE_Vxx.LIB" can be used.

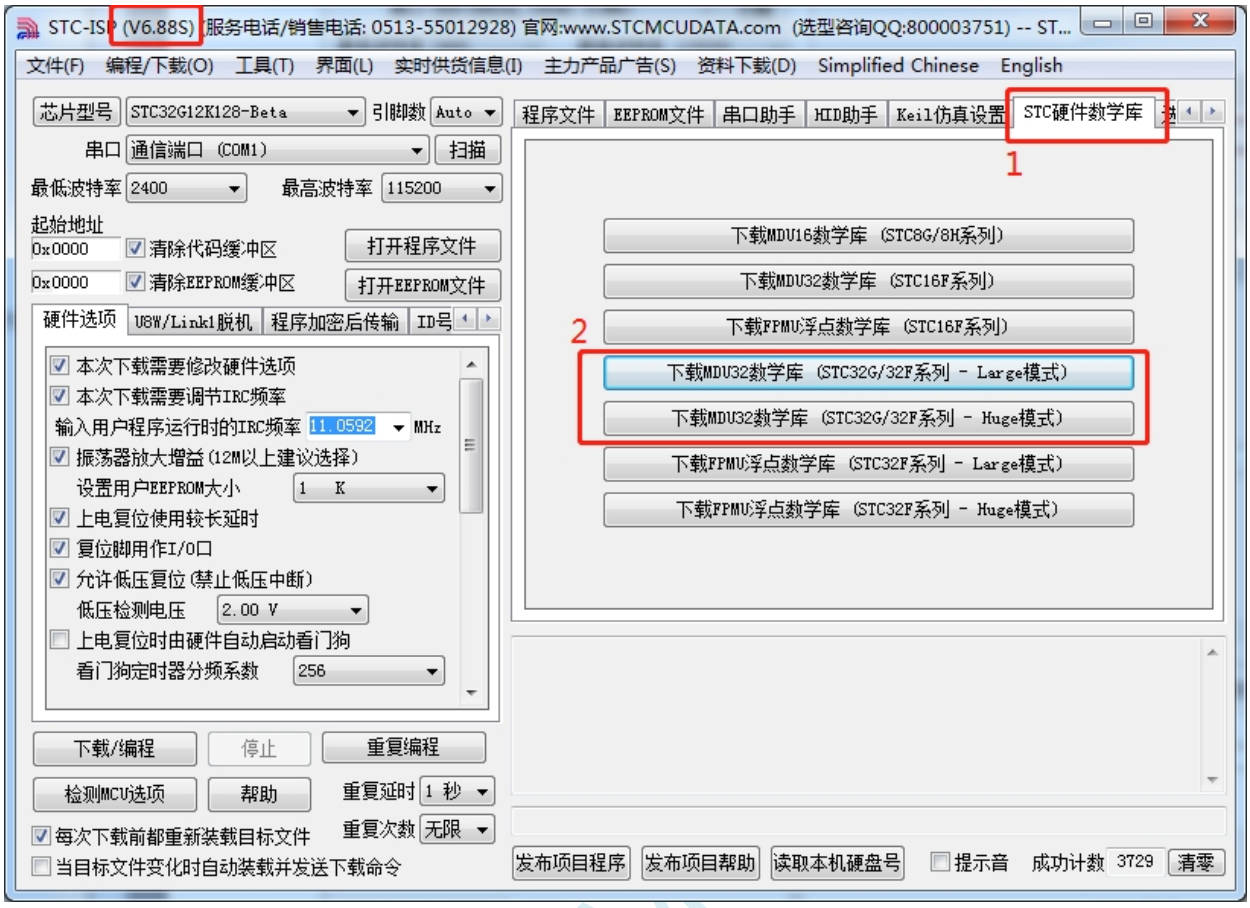
(Note: Whether you need to add the LAGRE version or the HUGE version depends on the code size mode of your project. If the code size mode is Huge, then you need to add the HUGE version of the library, while other modes need to add the LARGE version).



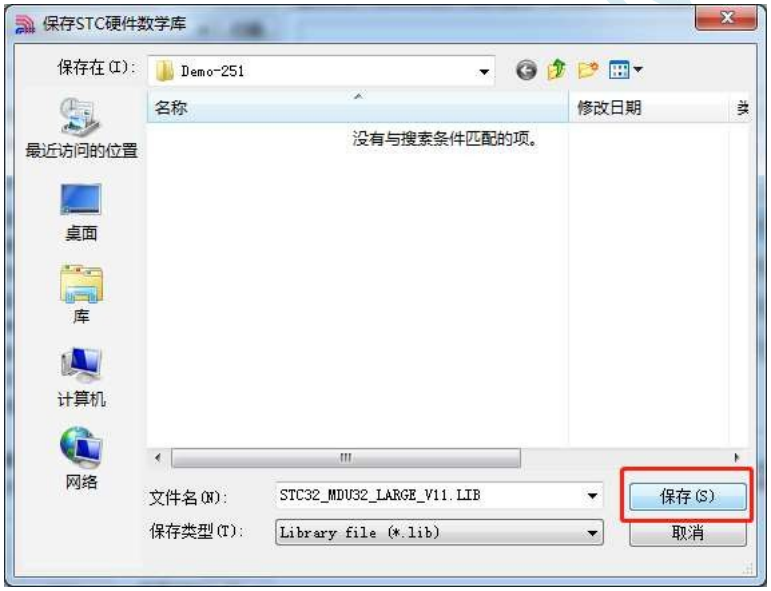
Add library files to the project:



How to get the library files: STC-ISP software V6.88S and later versions, click the "STC Hardware Maths Library" tab to get the library files.



Click on the button of the MDU32 Maths Library you want to download, choose the location




```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    //see download software for header files
#include "intrins.h"

volatile unsigned long int near uint1, uint2, xuint;\
volatile long int sint1, sint2, xsint; \ volatile long int
sint1, sint2, xsint; \ volatile unsigned long int near uint1,
```

```
void main(void)
{
    EAXFR = 1;                          //Enable access to XFR
    CKCON = 0x00;                       //Set the external data bus speed to fastest
    WTST = 0x00;                        //set the program code wait parameter.
                                        //Assign a value of 0 to set the CPU to execute the
                                        programme as fast as possible.

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

    P10 = 0;
    sint1 = 0x31030F05;
    sint2 = 0x00401350.
    xsint = sint1 * sint2.

    uint1 = 5 ;
    uint2 = 50.
    xuint = uint1 * uint2.

    uint1 = 528745.
    uint2 = 654689;
    xuint = uint1 / uint2.

    sint1 = 2000000000.
    sint2 = 2134135177.
    xsint = sint1 / sint2.

    sint1 = -2000000000;
    sint2 = -2134135177.
    xsint = sint1 / sint2.

    sint1 = -2000000000;
    sint2 = 2134135177.
    xsint = sint1 / sint2.
    P10 = 1;

    while(1);
}
```

32 Single Precision Floating Point Unit (FPMU)

32.1 Introduction to FPMU Floating Point Operators

The single-precision floating-point operator (FPMU) provides fast single-precision floating-point arithmetic operations.FPMU supports single-precision floating-point addition, subtraction, multiplication, division, squaring, comparison, and trigonometric functions (sine, cosine, tangent, and secant). It also supports conversion between integer types and single precision floating point numbers. The input floating-point number format complies with the IEEE-754 standard.The FPMU is controlled by a dedicated direct memory access DMA. All arithmetic operations are initiated by writing arithmetic instructions to the control registers called DMAIR. The operands (or pointers to them) and the results (or pointers to them) of all arithmetic operations performed by the FPMU module are located in the current group of registers R0-R7.

32.2 Related Special Function Registers

notation	descriptions	addr-ess	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMAIR	DMA Instruction Register	EDH	DMAIR[7:0]								0000,0000

Note: To write instruction code to DMAIR register, you can only use the instruction "MOV DMAIR,#N" in immediate addressing mode, using other instructions will not trigger the calculation properly.

32.3 Operational implementation schedule

FPMU Operations	DMA command code		Number of execution clocks
	HEX	DEC	
floating point addition	0x1C	28	31 to 40
floating-point subtraction	0x1D	29	31 to 40
floating point multiplication	0x1E	30	26 to 34
floating point division	0x1F	31	58 to 67
floating point number square (math.)	0x20	32	50 to 54
floating point comparison	0x21	33	18
Floating Point Detection	0x22	34	15
sine function	0x2D	45	32 to 270
cosine function	0x2E	46	32 to 270
tangent function (math.)	0x2F	47	58 to 258
tangent function (math.)	0x30	48	62 to 175

Floating point to 8-bit integer	0x23	35	19 - 30
Floating point to 16-bit integer	0x24	36	19 - 30
Floating point to 32-bit integer	0x25	37	23 - 39
8-bit Integer to Floating Point	0x27	39	23 to 33
16-bit Integer to Floating Point	0x28	40	23 to 33
32-bit Integer to Floating Point	0x29	41	24 to 33
Initialising the coprocessor	0x31	49	2
Clear anomalies	0x32	50	4

Read Status Register	0x33	51	4
Write Status Register	0x34	52	4
Read Control Register	0x35	53	4
Write Control Register	0x36	54	4

32.4 FPMU Basic Arithmetic Operations

This subsection describes all of the arithmetic operations that the FPMU module can perform using the DMA controller. All operands must be located in data memory. The results of the operations are also stored in the data memory space of the current group of R0-R7 selected by the PSW (0xD0) bit.

32.4.1 Floating point addition (+)

Performs addition operation on two floating point numbers. The adding number BR is located in R0 to R3 registers, the added number AR is located in R4 to R7 registers, and the result of the calculation and saved to R4 to R7 registers

respond in singing	=	Additive AR	+	Additive BR
MSB LSB		MSB LSB		MSB LSB
R4 R5 R6 R7	=	R4 R5 R6 R7	+	R0 R1 R2 R3

Command Code 0x1C(28)

Execution time (number of clocks) 31 to 40

32.4.2 Floating-point subtraction (-)

Subtraction operation is performed on two floating point numbers. The subtractor BR is located in the R0 to R3 registers, the subtracted number AR is located in the R4 to R7 registers, and the difference of the calculation results is saved in the R4 to R7 registers.

differ from	=	Subtracted number AR	-	Decrease BR
MSB LSB		MSB LSB		MSB LSB
R4 R5 R6 R7	=	R4 R5 R6 R7	-	R0 R1 R2 R3

Command Code 0x1D(29)

Execution time (number of clocks) 31 to 40

32.4.3 Floating-point multiplication (x)

The multiplication operation is performed on two floating point numbers. The multiplier BR is located in the R0 to R3 registers, the multiplied number AR is located in the R4 to R7 registers, and the product of the calculation results is stored in the R4 to R7 registers.

measured quantity (such as area of rectangle)	=	Multiplier AR	×	Multiplier BR
MSB LSB		MSB LSB		MSB LSB
R4 R5 R6 R7	=	R4 R5 R6 R7	×	R0 R1 R2 R3

Command Code 0x1E(30)

Execution time (number of clocks) 26 to 34,

32.4.4 Floating-point division (÷)

The division operation is performed on two floating point numbers. The divisor BR is located in R0 to R3 registers, the divisor AR is located in R4 to R7 registers, and the resultant quotient is stored in R4 to R7 registers.

2nd note in pentatonic scale				=	Dividend AR				÷	Divisor BR			
MSB			LSB		MSB			LSB		MSB			LSB
R4	R5	R6	R7	=	R4	R5	R6	R7	÷	R0	R1	R2	R3

Command Code 0x1F(31)

Execution time (number of clocks) 58 to 67

32.4.5 Floating-point number square/square root (sqrt)

The square root of the result is stored in the R4 to R7 registers. The open square number AR is located in R4 to R7 registers, and the square root of the calculation result is stored in R4 to R7 registers.

square root				= sqrt	Floating point AR			
MSB			LSB		MSB			LSB
R4	R5	R6	R7	= sqrt	R4	R5	R6	R7

Command Code 0x20(32)

Execution time (number of clocks) 50 - 54

32.4.6 Floating Point Comparison (comp)

The arithmetic comparison operation is performed on two floating point numbers. The comparison number BR is located in R0 to R3 registers, the compared number AR is located in R4 to R7 registers, and the comparison result is saved to R7 register.

R7.3	R7.2	R7.1	R7.0	Comparative results	Description of results
0	0	0	0	0x0001	AR > BR
1	0	0	0	0x0000	AR = BR
0	0	0	1	0xFFFF	AR < BR
1	1	0	1	unchanged	Unordered

Comparative results				=	Comparator AR				?	Comparator BR			
			LSB		MSB			LSB		MSB			LSB
-	-	-	R7	=	R0	R1	R2	R3	?	R4	R5	R6	R7

Command Code 0x21(33)

Execution time (number of clocks) 18

32.4.7 Floating point detection (check)

Detection is performed on one floating point number. The detected number AR is located in R4 to R7 registers, and the detection result is saved in R7 register.

R7[7:4]	R7.3	R7.2	R7.1	R7.0	Description of results
0000	0	0	0	0	Positive non-floating point number (+NaN)
0000	0	0	1	1	Negative non-floating point number (-NaN)
0000	0	1	0	0	positive-normed floating point number (PNF)
0000	0	1	1	0	negative-normed floating point number (NNF)
0000	0	1	0	1	Positive infinity (+INF)
0000	0	1	1	1	Negative infinity (-INF)

0000	1	0	0	0	positive zero
0000	1	0	1	0	minus zero
0000	1	1	0	0	positive unregulated floating point number (PUN)
0000	1	1	1	0	negative unregulated floating-point number

Test results	=	Number of Detected AR
LSB		MSB LSB

-	-	-	R7	=	R4	R5	R6	R7

Command Code 0x22(34)

Execution time (number of clocks) 15

32.5 FPMU trigonometric function

Note: All trigonometric functions have an angle argument type of radians. The formula for converting radians to angles:

$$\text{Angle} = \frac{180^\circ}{\pi} \times \text{radian} \qquad \text{Radian} = \frac{\pi}{180^\circ} \times \text{angle}$$

32.5.1 Sine function (sin)

Finds the sine of a single precision radian floating point number. The radian number AR is located in the R4 to R7 registers, and the result of the calculation is saved to R4 to R7

in the end	Arc AR

= sin

MSB			LSB		MSB			LSB
R4	R5	R6	R7	= sin	R4	R5	R6	R7

Command Code 0x2D(45)

Execution time (number of clocks) 32 - 270 clk

32.5.2 Cosine function (cos)

Finds the cosine of a single precision radian floating point number. The radian number AR is located in the R4 to R7 registers, and the result of the calculation is saved to R4 to R7

in the end	Arc AR

= cos

MSB			LSB		MSB			LSB
R4	R5	R6	R7	= cos	R4	R5	R6	R7

Command Code 0x2E(46)

Execution time (number of clocks) 32 - 270

32.5.3 Tangent function (tan)

Finds the tangent of a single precision radian floating point number. The radian number AR is located in the R4 to R7 registers, and the result of the calculation is saved to R4 to R7

in the end	Arc AR

= tan

MSB			LSB		MSB			LSB
R4	R5	R6	R7	= tan	R4	R5	R6	R7

Command Code 0x2F(47)

Execution time (number of clocks) 58 - 258

32.5.4 Arctangent function (arctan)

Finds the arc tangent of a single precision radian floating point number. The radian number AR is located in the R4 to R7 registers, and the result of the calculation is saved to R4 to R7

result	Arc AR

= arctan

MSB			LSB		MSB			LSB
R4	R5	R6	R7	= arctan	R4	R5	R6	R7

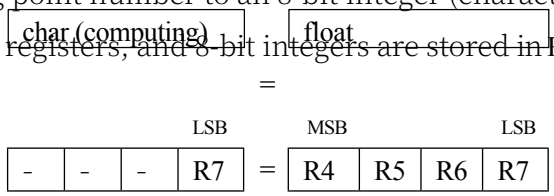
Command Code 0x30(48)

Execution time (number of clocks) 62 - 175

32.6 FPMU Data Conversion Operation

32.6.1 Float to 8-bit integer (float → char)

Converts a floating point number to an 8-bit integer (character char). Floating point numbers are located in R4 to R7 registers, and 8-bit integers are stored in R7



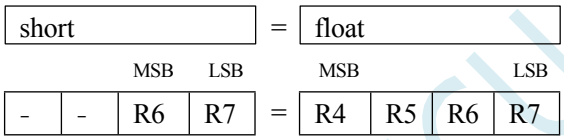
Command Code 0x23(35)

Execution time (number of clocks) 19 - 30

32.6.2 Float to 16-bit integer (float → short)

Converts a floating point number to a 16-bit integer (short integer short). Floating point numbers are located in the R4 to R7 registers, and 16-bit integers are stored in the R6 to R7 registers.

middle

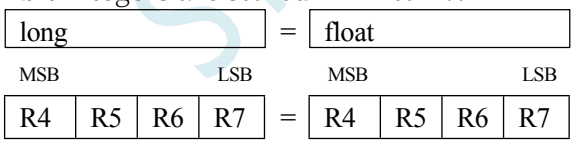


Command Code 0x24(36)

Execution time (number of clocks) 19 - 30

32.6.3 Float to 32-bit integer (float → long)

Converts a floating point number to a 32-bit integer (long). Floating point AR is located in R4 to R7 registers, and 32-bit integers are stored in R4 to R7.

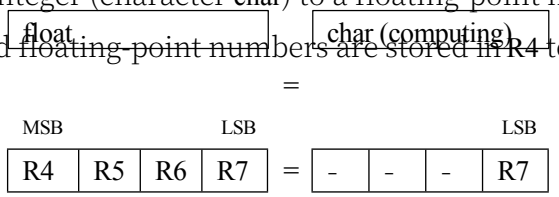


Command Code 0x25(37)

Execution time (number of clocks) 23 - 39

32.6.4 8-bit integer to float (char → float)

Converts an 8-bit integer (character char) to a floating-point number. 8-bit integers are located in the R7 register, and floating-point numbers are stored in R4 to R7.



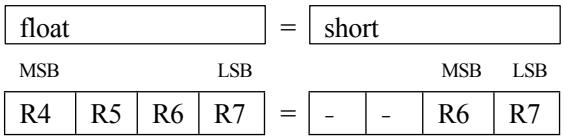
Command Code 0x27(39)

Execution time (number of clocks) 23 - 33

32.6.5 16-bit integer to float (short → float)

Converts a 16-bit integer (short integer) to a floating-point number. 16-bit integers are located in the R6 to R7 registers, and floating-point numbers are stored in the R4 to R7 registers.

middle



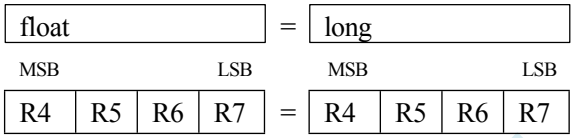
Command Code 0x28(40)

Execution time (number of clocks) 23 - 33

32.6.6 32-bit integer to float (long → float)

Converts a 32-bit integer (long integer) to a floating-point number. 32-bit integers are located in the R4 to R7 registers, and floating-point numbers are stored in the R4 to R7 registers.

middle



Command Code 0x29(41)

Execution time (number of clocks) 24 - 33

32.7 FPMU coprocessor controlled operation

32.7.1 Initialising the coprocessor

Initialise the FPMU coprocessor, after initialisation an exception is generated which needs to be cleared by software.

Command Code 0x31(49)

Execution time (number of clocks) 2

32.7.2 Clear anomalies

Remove all anomalies

Command Code 0x32(50)

Execution time (number of clocks) 4

32.7.3 Read Status Register

Reads the status register of the coprocessor. The result is saved to R7

Command Code 0x33(51)

Execution time (number of clocks) 4

32.7.4 Write Status Register

Write the coprocessor's status register. The value to be written is located in the R0 register, and the new status register value is saved to R7 when finished

Command Code 0x34(52)

Execution time (number of clocks) 4

32.7.5 Read Control Register

Reads the control register of the coprocessor. The result is saved to R0

Command Code 0x35(53)

Execution time (number of clocks) 4

32.7.6 Write Control Register

Write the coprocessor's control register. The value to be written is located in the R0 register, and the new control register value is saved to R7 when finished

Command Code 0x36(54)

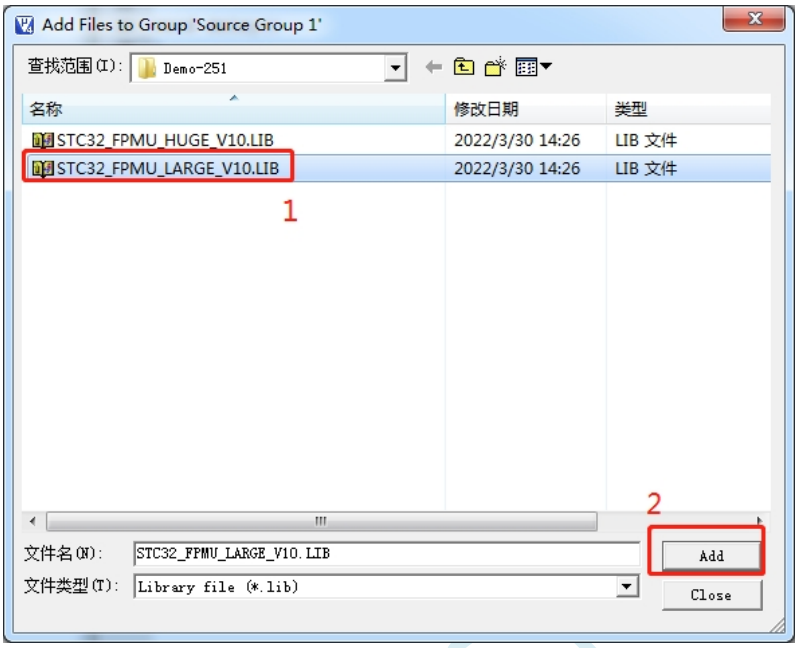
Execution time (number of clocks) 4

32.8 sample procedure

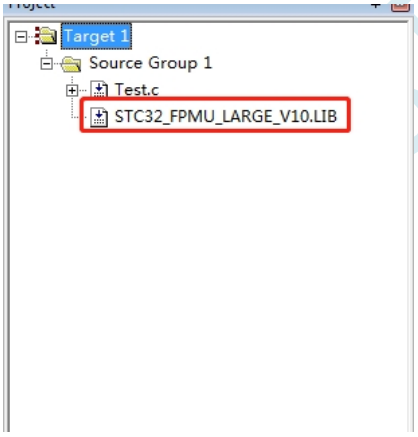
When you want to use hardware floating point, just add the keil project

The library file "STC32_FPMU_LARGE_Vxx.LIB" or "STC32_FPMU_HUGE_Vxx.LIB" can be used:

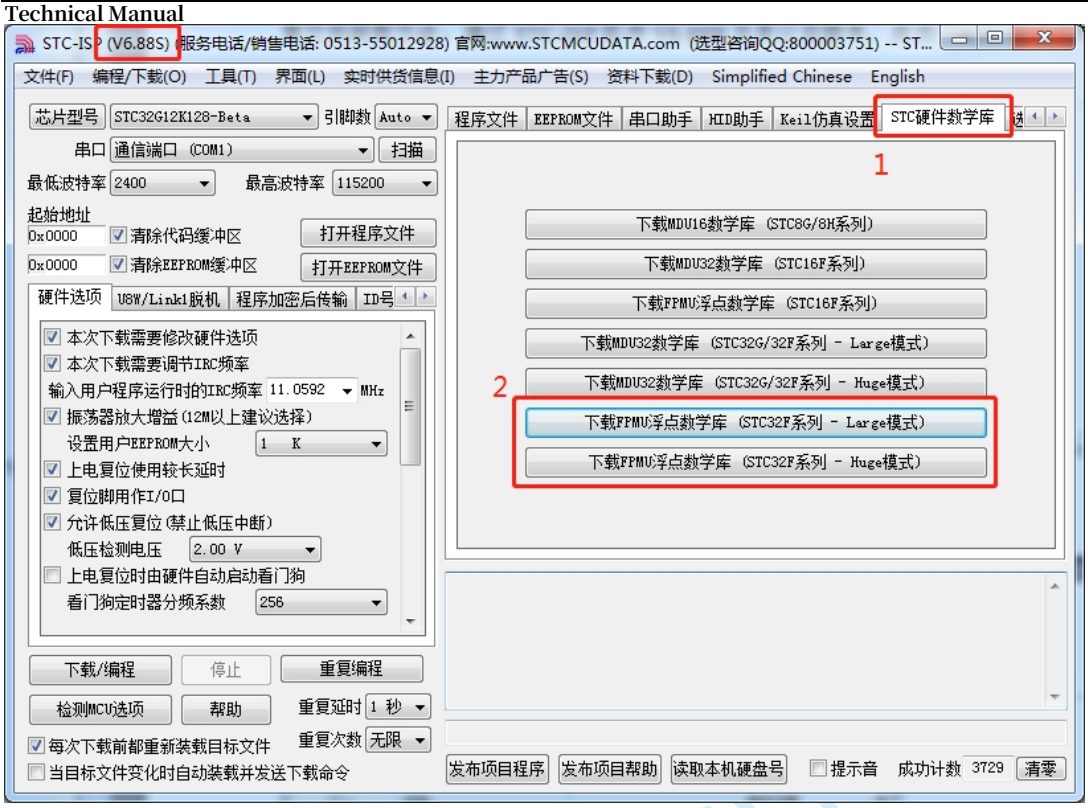
(Note: Whether you need to add the LAGRE version or the HUGE version depends on the code size mode of your project. If the code size mode is Huge, then you need to add the HUGE version of the library, while other modes need to add the LARGE version).



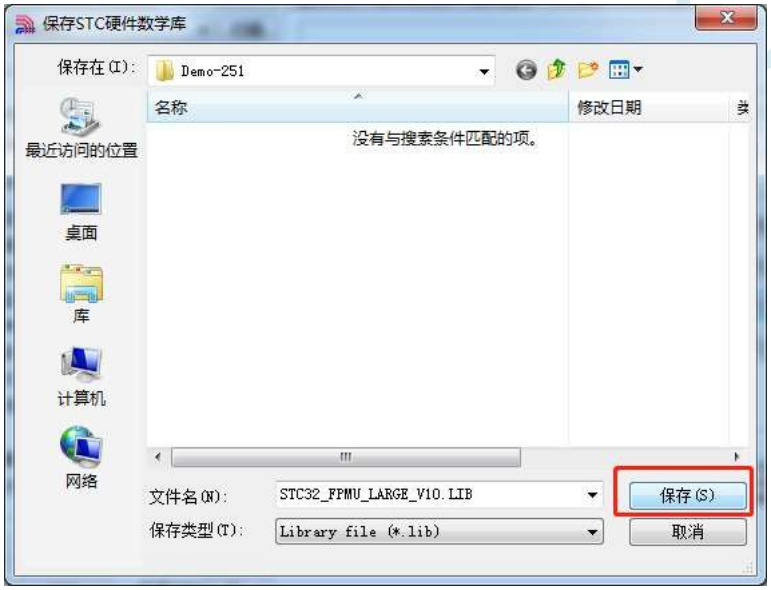
Add library files to the project:



How to get the library files: STC-ISP software V6.88S and later versions, click the "STC Hardware Maths Library" tab to get the library files.



Click on the button of the FPMU Maths Library you want to download, select the location where




```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    //see download software for header files
#include "intrins.h"
#include <math.h>

float data cfl1=3.9;
float data cfl2=5.1;
float data cfl3.

void main(void)
{
    EAXFR = 1;                                         //Enable access to XFR
```

```
    CKCON = 0x00;                   //Set the external data bus speed to fastest
    WTST = 0x00;                    //set the program code wait parameter.
                                    //Assign a value of 0 to set the CPU to execute the
                                    programme as fast as possible.

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

    P10 = 0;
    cfl3 = cfl1*cfl2;
    cfl3 = cfl1/cfl2-cfl3.
    cfl3 = cfl1*cfl2+cfl3;
    cfl3 = cfl1/cfl2*sin(cfl3);
    cfl3 = cfl1/cfl2*cos(cfl3);
    cfl3 = cfl1/cfl2*tan(cfl3).
    cfl3  =  cfl1/cfl2*sqrt(cfl3);
    cfl3 = cfl1/cfl2*atan(cfl3);
    P10 = 1;

    while(1);
}
```

33 Enhanced Double Data Pointer

Two 24-bit data pointers are integrated in the STC32G series microcontrollers. The automatic increment or decrement function of the data pointer and the automatic switching function of the two sets of data pointers can be realised by program control.

33.1 Related Special Function Registers

notation	descriptions	address	bit address and symbol								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	Data pointer (low byte)	82H									0000,0000
DPH	Data pointer (high byte)	83H									0000,0000
DPXL	Second set of data pointers (low byte)	84H									0000,0000
DPS	DPTR Pointer Selector	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR Timing Control Register	AEH									0000,0000

33.1.1 Group 1 16-bit Data Pointer Register (DPTR0)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								
DPXL	84H								

DPL for B7~B0 data bits DPH for B15~B8 data bits DPXL for B23~B16 data bits

The combination of DPL, DPH and DPXL is the 24-bit data pointer register DPTR0 or DPTR1.

33.1.2 Data Pointer Control Register (DPS)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1: Controls DPTR1 auto increment method 0: DPTR1 auto increment
1: DPTR1 Automatic Decrement

ID0: Controls DPTR0 auto increment mode 0: DPTR0 auto increment
1: DPTR0 Auto Decrement

TSL: DPTR0/DPTR1 auto-switching control (automatically inverts SEL) 0: Disable auto-

switching function

1: Enable automatic switching function

When the TSL bit is set to 1, the system automatically reverses the SEL bit whenever the relevant instruction is executed. The instructions related to TSL include the following instructions:

```
MOV     DPTR,#data16
INC     DPTR
MOVC
A,@A+DPTR MOVX
        A,@DPTR
MOVX    @DPTR,A
MOV     WRj,@DR56
MOV     @DR56,WRj
MOV     Rm,@DR56
MOV     @DR56,Rm
CMP     Rm,@DR56
SUB     Rm,@DR56
ADD     Rm,@DR56
ORL     Rm,@DR56
ANL     Rm,@DR56
XRL     Rm,@DR56
PUSH    DR56
POP     DR56
MOV     DR56,DRks
ADD     DR56,DRks
SUB     DR56,DRks
MOVH    DR56,#i16
MOV     DR56,#0_i16
MOV     DR56,#1_i16
MOV     DR56,d16
```

AU1/AU0: Enable DPTR1/DPTR0 to use ID1/ID0 control bits for auto increment/decrement control 0: Disable auto increment/decrement function

1: Enable auto increment/decrement function

Note: In write-protect mode, the AU0 and AU1 bits cannot be directly enabled individually; if the AU1 bit is enabled individually, the AU0 bit will also be enabled automatically, and if AU0 is enabled individually, it has no effect. If you need to enable AU1 or AU0 individually, you must use TA register to trigger the protection mechanism of DPS (refer to the description of TA register). In addition, the auto increment/decrement operation of DPTR0/DPTR1 will be performed only after the following 3 instructions are executed. 3 related instructions are as follows:

```
MOVC
A,@A+DPTR MOVX
        A,@DPTR
MOVX    @DPTR,A
MOV     WRj,@DR56
MOV     @DR56,WRj
MOV     Rm,@DR56
MOV     @DR56,Rm
CMP     Rm,@DR56
SUB     Rm,@DR56
```

```
ADD     Rm,@DR56
ORL     Rm,@DR56
ANL     Rm,@DR56
XRL     Rm,@DR56
```

SEL: Select DPTR0/DPTR1 as the current target DPTR

0: Selects DPTR0 as the target DPTR.

1: Select DPTR1 as the target DPTR

SEL Select Target DPTR is valid for the following commands:

```
MOV     DPTR,#data16
INC     DPTR
MOVC
A,@A+DPTR MOVX
        A,@DPTR
MOVX    @DPTR,A
JMP     @A+DPTR
MOV     WRj,@DR56
MOV     @DR56,WRj
MOV     Rm,@DR56
MOV     @DR56,Rm
CMP     Rm,@DR56
SUB     Rm,@DR56
ADD     Rm,@DR56
ORL     Rm,@DR56
ANL     Rm,@DR56
XRL     Rm,@DR56
PUSH    DR56
POP     DR56
MOV     DR56,DRks
ADD     DR56,DRks
SUB     DR56,DRks
MOVH    DR56,#i16
MOV     DR56,#0_i16
MOV     DR56,#1_i16
MOV     DR56,d16
```


33.1.3 Data Pointer Control Register (TA)

notation	address	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

The TA register is write-protected for AU1 and AU0 in the DPS registers. Since the programme cannot write individually to AU1 and AU0 in the DPS, when it is necessary to enable AU1 or AU0 individually, it must be triggered using the TA register, which is a write-only register. The TA register is a write-only register. When it is necessary to enable AU1 or AU0 individually, the following steps must be followed:

```
CLR     EA          ;Turn off interrupts (required)
MOV     TA,#0AAH    Write Trigger Command Sequence 1
                    No other instructions can be given
                    here.
MOV     TA,#55H     Write Trigger Command Sequence 2
                    No other instructions can be given
```

```
                                here.
        MOV     DPS,#xxH        Write protection is temporarily
                                disabled and any value can be
                                written to the DPS.
                                ;DSP write-protect state again
        SETB    EA              ;Turn on interrupts (if necessary)
```

33.2 sample procedure

33.2.1 Sample Code 1

Copy 4 bytes of data from FF:1000H to FF:1003H into 1:0100H to 1:0103H of extended RAM, i.e.
C:FF1000H -> X:10103H.
C:FF1001H -> X:10102H
C:FF1002H -> X:10101H
C:FF1003H -> X:10100H

assembly code

Tested operating frequency is *11.0592MHz.*

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN.
            MOV     SP, #5FH
            ORL     P_SW2,#80H          ;Enable access to XFR

            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            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 FF1000H to DPTR0 and then select
                                        DPTR1 as DPTR.
            MOV     DPTR,#0103H         Write 10103H to DPTR1.
            MOV     dps,#10111000b      Set DPTR1 to decrement mode, DPTR0 to
                                        increment mode, and enable TSL.
            MOV     r7,#4               AU0 and AU1, and select DPTR0 as the
                                        current DPTR.
COPY_NEXT.
            CLR     A                   Setting the number of data replicas
```

```
                                                    ;

        MOVC        A,@A+DPTR                       ;read data from the programme space indicated
                                                    by DPTR0, ;read data from the programme space
                                                    indicated by DPTR0.
```

```
                                        DPTR0 is automatically incremented by 1
                                        after completion and DPTR1 is set to
                                        DPTR.
        MOVX    @DPTR,A                 Write the ACC data to XDATA indicated by
                                        DPTR1.
                                        When finished, DPTR1 is automatically
                                        decremented by 1 and DPTR0 is set to DPTR.
        DJNZ    R7,COPY_NEXT            ;
        MOV     DPS,#00000000B          ;Restore DPS defaults

        SJMP    $

        END
```

33.2.2 Sample Code 2

Send the data from 0100H to 0103H of the extended RAM to the P0 port in sequence.

Test operating frequency is *11.0592MHz*.

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

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN.
        MOV     SP, #5FH
        ORL     P_SW2,#80H              ;Enable access to XFR

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

        CLR     EA                      ;Turn off interrupts
        MOV     TA,#0AAH                ;Write DPS Write Protect Trigger Command 1
        MOV     TA,#55H                 Write DPS Write Protect Trigger Command 2
        MOV     DPS,#00001000B          DPTR0 increment, enable AU0 alone, and
                                        select DPTR0.
        SETB    EA                      ;Turn on interrupts
```

```
MOV     DPTR,#0100H     ;Write 0100H to DPTR0.
MOVX    A,@DPTR         ;DPTR0 is automatically incremented by 1
                        ;when data is read from the XRAM indicated
                        ;by DPTR0.
MOV     P0,A            ;Data output to P0 port
```

```
        MOVX    A,@DPTR         ;DPTR0 is automatically incremented by 1 when data is read from the XRAM indicated by DPTR0.
        MOV     P0,A            ;Data output to P0 port
        MOVX    A,@DPTR         ;DPTR0 is automatically incremented by 1 when data is read from the XRAM indicated by DPTR0.
        MOV     P0,A            ;Data output to P0 port
        MOVX    A,@DPTR         ;DPTR0 is automatically incremented by 1 when data is read from the XRAM indicated by DPTR0.
        MOV     P0,A            ;Data output to P0 port
        MOV     DPS,#00000000B  ;Restore DPS defaults

        SJMP    $

        END
```

Appendix A Instruction Set

A.1 Instruction Set Introduction

A.1.1 BINARY mode and SOURCE mode

Binary mode and Source mode refer to the two ways of providing opcodes for the STC32G architecture instruction set. Depending on the user programme, either Binary mode or Source mode may produce more efficient code.Binary mode refers to the standard opcodes of the MCU51.Source mode refers to the MCU251-specific set of opcodes that extends the instruction set with additional operations and addressing modes. The special mnemonic 0xA5 is used to distinguish the specific instructions in each mode. All unused opcodes are correctly decoded and executed as NOPs.

A.1.2 instruction set tag

Instructions have five different addressing modes: immediate addressing, direct addressing, register addressing, indirect addressing addressing and relative addressing. In immediate addressing mode, the operand is contained in the opcode. For direct addressing, an 8-bit address or 16-bit address is part of the opcode; for register addressing, a register is selected in the opcode for operation. In indirect addressing mode, a register is selected in the opcode to point to the address used for the operation. Relative addressing mode is used for jump instructions. The following table provides the number of cycles in the STC32G microcontroller core instruction set. One cycle number is equal to one system clock. Tables 1 and 2 contain descriptions of the mnemonics used in the instruction set tables. Tables 3 through 7 identify the number of hex codes, bytes, and system clocks required for each instruction to execute.

Rn N rrr	Current working byte register R0, R1, ..., R7 byte register index 0~7 Binary representation of n
Rm Rmd Rms m, md, ms ssss SSSS	Byte registers R0, R1, ..., R15 Target registers Source Number Register Byte register index: m, md, ms = 0, 1, ..., 15 binary representation of m or md Binary representation of ms

WR	Word registers WR0, WR2, ...,
WRjd	WR30 Target registers
WRjs	Source Number Register
@WRj	Indirect memory locations addressed by word registers (0x0000~0xFFFF)
@WRj+dis	Indirect memory location addressed by word register +0 to 64KB
j, jd, js	offset value (0x0000~0xFFFF) Word register index: j, jd, js = 0, 2,
tttt	..., 30
TTTT	Binary representation of j or jd
	Binary representation of js
DRk	Double word registers DR0, DR4, ..., DR28, DR56, DR60

DRkd DRks @DRk @DRk+dis k, kd, ks uuuuu UUUUU	Target Register Source Number Register Indirect memory locations addressed by double word registers (0x000000~0xFFFFFF) Indirect memory location from double-word register addressing +0 to 64KB offset value (0x000000~0xFFFFFF) Double-word register index: k, kd, ks = 0, 4, …, 28, 56, 60 Binary representation of k or kd Binary representation of ks
dir8	128 internal memory locations, various special function registers
dir16	16-bit memory address (0x000000~0x00FFFF)
@Ri	Indirect memory location addressed by register R0 or R1 (0x00~0xFF)
#data	8-bit immediate number included in the instruction
#data16	The 16-bit immediate number is contained in bytes 2 and 3 of the instruction.
#0data16	32-bit immediate number; high word filled with zeros, low word contained in bytes 2 and 3 of the instruction
#1data16	32-bit immediate number; high word filled with 1, low word contained in bytes 2 and 3 of the instruction
#short vv	Constants equal to 1, 2 or 4 are included in the instruction Binary representation of #short
bit	Memory location (0x20~0x7F) or any defined direct addressing bit in the SFR
bit51	Directly addressed bits (bit number = 0x00~0xFF) in memory or SFR. Bits 0x00 to 0x7F are the 128 bits in internal memory with byte locations 0x20 to 0x2F. Bits 0x80~0xFF are the 128 bits in the 16 SFRs whose addresses are bounded by 0 or 8 Tail: 0x80, 0x88, 0x90, …, 0xF0, 0xF8
A	accumulator (computing)

Table 1, Data Addressing Mode Comments

addr24	The 24-bit destination address can be located anywhere in the 16MB address space. It is used for ECALL and EJMP instructions.
addr16	The destination address of LCALL and LJMP can be anywhere within the 64KB program memory address space.
addr11	The destination address of ACALL and AJMP will be located in the same 2 KB page of program memory as the first byte of the next instruction Inside the face.
rel	SJMP and all conditional jumps contain an 8-bit offset byte. The range is relative to the first byte of the next instruction offset byte Move +127 to -128 bytes.

Table 2, Program Addressing Mode Comments

-	This instruction does not modify the flag bits.
√	This instruction sets or clears the flag bits as required.
1	This instruction sets the flag bit.
0	This instruction clears the zero flag bit.

Table 3, Flag Bit Description Comments

A.1.3 Command List (Functional Ordering)

The total instruction execution time depends on the value of WTST (0xe9). Each table below labels the number of clocks when WTST=0. Calculate for each instruction the

The general formula for the total number of clocks is:

Instruction Clock Count = Clock Count + nrPRGACS*WTST (nrPRGACS = 0 or 1)

If the instruction accesses XDM memory, the CKCON[2:0] value should be multiplied by nrXDMACS (1 or 2) to calculate the correct number of cycles.

Instruction Clock Count = Clock Count + nrPRGACS*WTST + nrXDMACS*CKCON

Instructions dedicated to binary mode are marked in red. These instructions require the prefix 0xA5 before the opcode when executed in source mode (ESC). Instructions dedicated to source code mode are marked in blue. When executed in binary mode, these instructions require the 0xA5 prefix (ESC) before the opcode. All other instructions are always available regardless of CPU mode.

arithmetic operation

mnemonic sign	descriptions	encodings	nibbles	Number of clocks
ADD A,Rn	Add registers to accumulator	0x28-0x2F	1	1
ADD A,dir8	Add direct byte to accumulator	0x25	2	1
ADD A,@Ri	Add indirect memory to the accumulator	0x26-0x27	1	1
ADD A,#data	Add the immediate number to the accumulator	0x24	2	1
ADD Rm,Rm	Add byte registers to byte registers	0x2C	2	1
ADD WRj,WRj	Add word registers to word registers	0x2D	2	1
ADD reg,op2(3)	Add operand to Rm, WRj or DRk	0x2E	Note 1	Note 2
ADD DRk,DRk	Adding a double word register to a double word register	0x2F	2	1
ADDC A,Rn	Add registers to accumulator with rounding flag bits	0x38-0x3F	1	1
ADDC A,dir8	Add direct byte to accumulator with rounding flag A	0x35	2	1
ADDC A,@Ri	Add indirect memory to accumulator with rounding flag A	0x36-0x37	1	1
ADDC A,#data	Adds the immediate number to the accumulator with the rounding flag A	0x34	2	1
SUBB A,Rn	Borrow from A and subtract from register.	0x98-0x9F	1	1
SUBB A,dir8	Subtract the direct byte from the A debit	0x95	2	1
SUBB A,@Ri	Borrow from A and subtract indirect memory	0x96-0x97	1	1
SUBB A,#data	Borrow from A and subtract immediately	0x94	2	1
SUB Rm,Rm	Subtract byte registers from byte registers	0x9C	2	1
SUB WRj,WRj	Subtract word registers from word registers	0x9D	2	1
SUB reg,op2(3)	Subtract operands from Rm, WRj or DRk	0x9E	Note 1	Note 2
SUB DRk,DRk	Subtract double word registers from double word registers	0x9F	2	1

CMP Rm,Rm	Compare two byte registers	0xBC	2	1
CMP WRj,WRj	Compare two word registers	0xBD	2	1
CMP reg,op2(3)	Compare Rm, WRj or DRk to the operand.	0xBE	Note 1	Note 2
CMP DRk,DRk	Compare two double word registers	0xBF	2	1
INC A	incremental accumulator	0x04	1	1
INC Rn	incremental register	0x08-0x0F	1	1
INC dir8	Incremental Direct Bytes	0x05	2	1
INC @Ri	Incremental indirect memory	0x06-0x07	1	1
INC reg,#short(3)	Increment Rm, WRj or DRk.	0x0B	2	Note 2

DEC A	decremental accumulator	0x14	1	1
DEC Rn	decrement register	0x18-0x1F	1	1
DEC dir8	Decreasing direct byte	0x15	1	1
DEC @Ri	Dwindling indirect memory	0x16-0x17	2	1
DEC reg,#short(3)	Decreasing Rm, WRj or DRk	0x1B	2	Note 2
INC DPTR	Incremental data pointer	0xA3	1	1
MUL A,B	Multiply A by B	0xA4	1	1
MUL Rm,Rm	byte register multiplication	0xAC	2	1
MUL WRj,WRj	word register multiplication	0xAD	2	1
DIV A,B	Divide A by B	0x84	1	6
DIV Rm,Rm	byte register division	0x8C	1	6
DIV WRj,WRj	word register division	0x8D	1	10
DA A	Decimal Adjustment Accumulator	0xD4	1	3

Note 1: The number of bytes required for an instruction depends on the addressing mode determined by the immediately preceding byte. Refer to Instruction Set Details. Note 2: The number of cycles required for an instruction depends on the addressing mode determined by the immediately following byte. Refer to the instruction set for details. Note 3: The operands and addressing modes depend on the immediately preceding byte. All options are described in the Instruction Set Explanation.

logical operation

mnemonic sign	descriptions	encodings	nibbles	Number of clocks
ANL A,Rn	Register Logic and Accumulators	0x58-0x5F	1	1
ANL A,dir8	Direct Byte Logic and Accumulators	0x55	2	1
ANL A,@Ri	Indirect Memory Logic and Accumulators	0x56-0x57	1	1
ANL A,#data	Immediate Logic and Accumulators	0x54	2	1
ANL dir8,A	Direct Byte Logic and Accumulators	0x52	2	1
ANL dir8,#data	Direct Data Logic and Direct Bytes	0x53	3	1
ANL Rm,Rm	Two byte register logical sum	0x5C	2	1
ANL WRj,WRj	Two word register logical sums	0x5D	2	1
ANL reg,op2(3)	Operand Logic with Rm, WRj or DRk	0x5E	Note 1	Note 2
ORL A,Rn	Register Logic or Accumulator	0x48-0x4F	1	1
ORL A, dir8	Direct Byte Logic or Accumulator	0x45	2	1
ORL A,@Ri	Indirect Memory Logic or Accumulator	0x46-0x47	1	1
ORL A,#data	Immediate logic or accumulator	0x44	2	1
ORL dir8,A	accumulator direct byte	0x42	2	1
ORL dir8,#data	Immediate logic or direct byte	0x43	3	1
ORL Rm,Rm	Two byte register logical or	0x4C	2	1
ORL WRj,WRj	Two word register logical or	0x4D	2	1

ORL reg,op2[3]	Operand Logic or Rm, WRj or DRk	0x4E	Note 1	Note 2
XRL A,Rn	register different-or accumulator	0x68-0x6F	1	1
XRL A, dir8	direct byte allosteric accumulator	0x65	2	1
XRL A,@Ri	Indirect Memory Iso-accumulator	0x66-0x67	1	1

XRL A,#data	Immediate Differential or Accumulator	0x64	2	1
XRL dir8,A	direct byte allosteric accumulator	0x62	2	1
XRL dir8,#data	Immediate to Iso-Or-Direct Byte	0x63	3	1
XRL Rm,Rm	Two byte registers are different or	0x6C	2	1
XRL WRj,WRj	Dissimilarity of two word registers	0x6D	2	1
XRL reg,op2(3)	Operand dissimilarity Rm, WRj or DRk	0x6E	Note 1	Note 2
CLR A	Zeroing the accumulator	0xE4	1	1
CPL A	accumulator inversion	0xF4	1	1
RL A	Accumulator cyclically moves to the left	0x23	1	1
RLC A	Accumulator with rounding to the left in a circular motion	0x33	1	1
RR A	The accumulator moves circularly to the right	0x03	1	1
RRC A	Accumulator with rounding to the right cyclic movement	0x13	1	1
SRA reg(3)	Shift right by MSB Rm or WRj	0x0E	2	1
SRL reg(3)	Shift right Rm or WRj	0x1E	2	1
SLL reg(3)	Shift left Rm or WRj	0x3E	2	1
SWAP A	Swap half bytes within the accumulator	0xC4	1	1

Note 1: The number of bytes required for an instruction depends on the addressing mode determined by the immediately preceding byte. Refer to Instruction Set Details. Note 2: The number of cycles required for an instruction depends on the addressing mode determined by the immediately following byte. Refer to the instruction set for details. Note 3: The operands and addressing modes depend on the immediately preceding byte. All options are described in the Instruction Set Explanation.

boolean operation

mnemonic sign	descriptions	encodings	nibbles	Number of clocks
CLR C	Bit Flag Bit Clear	0xC3	1	1
CLR bit	Direct Bit Clear	0xC2	2	1
SETB C	Bit Flag Position Bit	0xD3	1	1
SETB bit	direct location	0xD2	2	1
CPL C	Bitwise Flag Bit Inversion	0xB3	1	1
CPL bit	direct bitwise inversion	0xB2	2	1
ANL C,bit	Direct Bit Logic and Progressive Flag Bits	0x82	2	1
ANL C,/bit	Non-logical and incoming flag bits for direct bits	0xB0	2	1
ORL C, bit	Direct Bit Logic or Progressive Flag Bits	0x72	2	1
ORL C,/bit	Non-logical or progressive flag bits for direct bits	0xA0	2	1
MOV C,bit	Direct Bit Carry to Feed Flag Bit	0xA2	2	1
MOV bit,C	Progressive flag bit carry to direct bit	0x92	2	1

Bit instr[1]	Bit instruction set (MCU251 specific)	0xA9	3	1

data transmission

mnemonic sign	descriptions	encodings	nibbles	Number of clocks
MOV A,Rn	Carrying registers to accumulators	0xE8-0xEF	1	1
MOV A,dir8	Carrying direct bytes to the accumulator	0xE5	2	1

MOV A,@Ri	Carrying indirect memory to the accumulator	0xE6-0xE7	1	1
MOV A,#data	Carrying immediate numbers to the accumulator	0x74	2	1
MOV Rn,A	Carrying accumulators to registers	0xF8-0xFF	1	1
MOV Rn,dir8	Carry direct bytes to registers	0xA8-0xAF	2	1
MOV Rn,#data	Carrying immediate numbers to registers	0x78-0x7F	2	1
MOV dir8,A	Carrying accumulators to direct bytes	0xF5	2	1
MOV dir8,Rn	Carrying registers to direct bytes	0x88-0x8F	2	1
MOV dir8,dir8	Carrying direct bytes to direct bytes	0x85	3	1
MOV dir8,@Ri	Carrying indirect memory to direct bytes	0x86-0x87	2	1
MOV dir8,#data	Carrying immediate numbers to direct bytes	0x75	3	1
MOV @Ri,A	Migrating the accumulator to indirect memory	0xF6-0xF7	1	1
MOV @Ri,dir8	Carrying direct bytes to indirect memory	0xA6-0xA7	2	1
MOV @Ri,#data	Moving immediate numbers to indirect memory	0x76-0x77	2	1
MOV Rm,Rm	Carrying byte registers to byte registers	0x7C	2	1
MOV WRj,WRj	Carrying word registers to word registers	0x7D	2	1
MOV reg,op2(3)	Carry operands to Rm, WRj, or DRk.	0x7E	Note 1	Note 2
MOV DRk,DRk	Carrying double word registers to double word registers	0x7F	2	1
MOV WRj,@DRk	Carry indirect (24-bit) memory to WRj	0x0B	3	4
MOV @DRk,WRj	Moving WRj to indirect (24-bit) memory	0x1B	3	6
MOV Rm, @WRj+dis	Carry indirect (16-bit) memory with a 16-bit offset to Rm	0x09	4	1
MOV @WRj+dis,Rm	Moving Rm to Indirect (16-bit) Memory with 16-bit Offset	0x19	4	1
MOV Rm, @DRk+dis	Carry indirect (24-bit) memory with 16-bit offset to Rm	0x29	4	3
MOV @DRk+dis, Rm	Moving Rm to 16-bit offset indirect (24-bit) memory	0x39	4	4
MOV WRj,@WRj+dis	Carry indirect (16-bit) memory with 16-bit offset to WRj	0x49	4	1
MOV @WRj+dis,WRj	Moving WRj to 16-bit offset indirect (16-bit) memory	0x59	4	3
MOV WRj,@DRk+dis	Carry indirect (24-bit) memory with 16-bit offset to WRj	0x69	4	4
MOV @DRk+dis,WRj	Moving WRj to 16-bit offset indirect (24-bit) memory	0x79	4	6
MOV op1,reg(3)	Carry Rm, WRj, or DRk to operand	0x7A	Note 1	Note 2
MOVH DRk,#data16(4)	Carry 16-bit immediate numbers to the high word of a double-word register.	0x7A	4	1
MOVZ WRj,Rm	Carrying byte registers to zero-extended word registers	0x0A	2	1
MOVS WRj,Rm	Carrying byte registers to word registers with symbolic extensions	0x1A	2	1
MOV DPTR,#data16	Load 16-bit constants into the active DPTR	0x90	3	1

MOVC A,@A+DPTR	Accumulator to carry code bytes to DPTR offset	0x93	1	4
MOVC A,@A+PC	Accumulator to carry code bytes to PC offset	0x83	1	3
MOVX A,@Ri	Carry external memory (8-bit address) to A	0xE2-0xE3	1	2*
MOVX A,@DPTR	Carry external memory (16-bit address) to A	0xE0	1	2*
MOVX @Ri,A	Carry A to external memory (8-bit address)	0xF2-0xF3	1	2*
MOVX @DPTR,A	Carry A to external memory (16-bit address)	0xF0	1	2*
PUSH dir8	Press direct bytes into the IDM stack	0xC0	2	1

POP dir8	Popping direct bytes from the IDM stack	0xD0	2	1
PUSH op1(3)	Press operands into the IDM stack	0xCA	Note 1	Note 2
POP op1(3)	Popping operands from the IDM stack	0xDA	Note 1	Note 2
XCH A,Rn	Registers are swapped with accumulators	0xC8-0xCF	1	1
XCH A,dir8	Direct byte exchange with accumulator	0xC5	2	1
XCH A,@Ri	Indirect memory swap with accumulator	0xC6-0xC7	1	1
XCHD A,@Ri	The lower half byte of indirect memory is exchanged with A	0xD6-0xD7	1	3

Note 1: The number of bytes required for an instruction depends on the addressing mode determined by the immediately preceding byte. Refer to Instruction Set Details. Note 2: The number of cycles required for an instruction depends on the addressing mode determined by the immediately following byte. Refer to the instruction set for details. Note 3: The operands and addressing modes depend on the immediately preceding byte. All options are described in the Instruction Set Explanation.

Note 4: The first byte of the opcode of the MOVH instruction is the same as that of the MOV op1,reg group. The instruction is distinguished by the value of the second byte.

program jump

mnemonic sign	descriptions	encodings	nibbles	Number of clocks
ACALL addr11	absolute subroutine call	0x11-0xF1	2	3
LCALL addr16	The long subroutine is called directly	0x12	3	3
ECALL addr24	Extension subroutines call directly	0x9A	4	3
ECALL @DRk	Extended Subroutine Indirect Calls	0x99	2	3
LCALL @WRj	Indirect calls to long subroutines	0x99	2	3
RET	The subroutine returns	0x22	1	3
ERET	The extension subroutine returns	0xAA	1	3
RETI	Interrupt return	0x32	1	3
AJMP addr11	Absolute Jump	0x01-0xE1	2	3
LJMP addr16	Direct long jump	0x02	3	3
EJMP addr24	Direct Extension Jump	0x8A	4	3
LJMP @WRj	long indirect jump	0x89	2	3
EJMP @DRk	Indirect Extension Jump	0x89	2	3
SJMP rel	Short jumps (relative addresses)	0x80	2	3
JMP @A+DPTR	Indirect Jump for DPTR Offset	0x73	1	3
JZ rel	Jump if accumulator is zero	0x60	2	1/3
JNZ rel	Jump if accumulator is not zero	0x70	2	1/3
JC rel	Jump if feed flag bit	0x40	2	1/3
JNC rel	Jump if the feed flag bit is not set	0x50	2	1/3
JB bit,rel	Jump if direct position bit	0x20	3	1/3

JNB bit, rel	Jump if direct bit is not set	0x30	3	1/3
JBC bit,rel	If the direct position bit is jumped and the zero bit is cleared	0x10	3	1/3
JSLE rel	Jump if less than or equal to (signed)	0x08	2	1/3
JSG rel	Jump if greater than (signed)	0x18	2	1/3
JLE rel	Jump if less than or equal to	0x28	2	1/3
JG rel	If greater then jump	0x38	2	1/3

JSL rel	Jump if less than (signed)	0x48	2	1/3
JSGE rel	Jump if greater than or equal to (signed)	0x58	2	1/3
JE rel	If equal then jump	0x68	2	1/3
JNE rel	If not equal then jump	0x78	2	1/3
cjne a,dir8,rel	Compare direct byte with A, if not equal then jump	0xB5	3	1/3
CJNE A,#data,rel	Compare the immediate number with A. If they are not equal, skip.	0xB4	3	1/3
CJNE Rn,#data,rel	Compare the immediate number with the register. If not equal then jump	0xB8-0xBF	3	1/3
CJNE @Ri,#data,rel	Compare immediate number with indirect memory. If not equal then jump	0xB6-0xB7	3	1/3
DJNZ Rn,rel	Register decrement, jump if not zero	0xD8-0xDF	2	1/3
DJNZ dir8,rel	Direct byte decrement, jump if not zero	0xD5	3	1/3
NOP	no operation	0x00	1	1

Note: Unconditional jumps are executed within 1 clock cycle.

special instruction

mnemonic sign	descriptions	encodings	nibbles	Number of clocks
TRAP	Trap Interrupt - Performed as NOP	0xB9	1	1
ESC	fig. appear suddenly	0xA5	1	1

A.1.4 Instruction list (machine code ordering)

Note: The STC32G uses SOURCE mode.

BINARY mode

opcode	mnemonic sign	opcode	mnemonic sign	opcode	mnemonic sign	opcode	mnemonic sign
00 H	NOP	40 H	JC rel	80 H	SJMP rel	C0 H	PUSH direct
01 H	AJMP addr11	41 H	AJMP addr11	81 H	AJMP addr11	C1 H	AJMP addr11
02 H	LJMP addr16	42 H	ORL direct, A	82 H	ANL C,bit	C2 H	CLR bit
03 H	RR A	43 H	ORL direct,#data	83 H	MOVC A,@A+PC	C3 H	CLR C
04 H	INC A	44 H	ORL A,#data	84 H	DIV AB	C4 H	SWAP A
05 H	INC direct	45 H	ORL A, direct	85 H	MOV direct,direct	C5 H	XCH A, direct
06 H	INC @R0	46 H	ORL A,@R0	86 H	MOV direct,@R0	C6 H	XCH A,@R0
07 H	INC @R1	47 H	ORL A,@R1	87 H	MOV direct,@R1	C7 H	XCH A,@R1
08 H	INC R0	48 H	ORL A,R0	88 H	MOV direct,R0	C8 H	XCH A,R0
09 H	INC R1	49 H	ORL A,R1	89 H	MOV direct,R1	C9 H	XCH A,R1
0A H	INC R2	4A H	ORL A,R2	8A H	MOV direct,R2	CA H	XCH A,R2
0B H	INC R3	4B H	ORL A,R3	8B H	MOV direct,R3	CB H	XCH A,R3
0C H	INC R4	4C H	ORL A,R4	8C H	MOV direct,R4	CC H	XCH A,R4
0D H	INC R5	4D H	ORL A,R5	8D H	MOV direct,R5	CD H	XCH A,R5
0E H	INC R6	4E H	ORL A,R6	8E H	MOV direct,R6	CE H	XCH A,R6
0F H	INC R7	4F H	ORL A,R7	8F H	MOV direct,R7	CF H	XCH A,R7
10 H	JBC bit,rel	50 H	JNC rel	90 H	MOV DPTR,#data16	D0 H	POP direct
11 H	ACALL addr11	51 H	ACALL addr11	91 H	ACALL addr11	D1 H	ACALL addr11
12 H	LCALL addr16	52 H	ANL direct, A	92 H	MOV bit,C	D2 H	SETB bit
13 H	RRC A	53 H	ANL direct,#data	93 H	MOVC A,@A+DPTR	D3 H	SETB C
14 H	DEC A	54 H	ANL A,#data	94 H	SUBB A,#data	D4 H	DA A
15 H	DEC direct	55 H	ANL A, direct	95 H	SUBB A,direct	D5 H	DJNZ direct, rel
16 H	DEC @R0	56 H	ANL A,@R0	96 H	SUBB A,@R0	D6 H	XCHD A,@R0
17 H	DEC @R1	57 H	ANL A,@R1	97 H	SUBB A,@R1	D7 H	XCHD A,@R1
18 H	DEC R0	58 H	ANL A,R0	98 H	SUBB A,R0	D8 H	DJNZ R0,rel
19 H	DEC R1	59 H	ANL A,R1	99 H	SUBB A,R1	D9 H	DJNZ R1,rel
1A H	DEC R2	5A H	ANL A,R2	9A H	SUBB A,R2	DA H	DJNZ R2,rel
1B H	DEC R3	5B H	ANL A,R3	9B H	SUBB A,R3	DB H	DJNZ R3,rel
1C H	DEC R4	5C H	ANL A,R4	9C H	SUBB A,R4	DC H	DJNZ R4,rel
1D H	DEC R5	5D H	ANL A,R5	9D H	SUBB A,R5	DD H	DJNZ R5,rel
1E H	DEC R6	5E H	ANL A,R6	9E H	SUBB A,R6	DE H	DJNZ R6,rel

1F H	DEC R7	5F H	ANL A,R7	9F H	SUBB A,R7	DF H	DJNZ R7,rel
20 H	JB bit,rel	60 H	JZ rel	A0 H	ORL C, bit	E0 H	MOVX A,@DPTR
21 H	AJMP addr11	61 H	AJMP addr11	A1 H	AJMP addr11	E1 H	AJMP addr11
22 H	RET	62 H	XRL direct, A	A2 H	MOV C,bit	E2 H	MOVX A,@R0
23 H	RL A	63 H	XRL direct,#data	A3 H	INC DPTR	E3 H	MOVX A,@R1
24 H	ADD A,#data	64 H	XRL A,#data	A4 H	MUL AB	E4 H	CLR A
25 H	ADD A,direct	65 H	XRL A, direct	A5 H	ESC	E5 H	MOV A, direct
26 H	ADD A,@R0	66 H	XRL A,@R0	A6 H	MOV @R0,direct	E6 H	MOV A,@R0
27 H	ADD A,@R1	67 H	XRL A,@R1	A7 H	MOV @R1,direct	E7 H	MOV A,@R1
28 H	ADD A,R0	68 H	XRL A,R0	A8 H	MOV R0,direct	E8 H	MOV A,R0
29 H	ADD A,R1	69 H	XRL A,R1	A9 H	MOV R1,direct	E9 H	MOV A,R1
2A H	ADD A,R2	6A H	XRL A,R2	AA H	MOV R2,direct	EA H	MOV A,R2
2B H	ADD A,R3	6B H	XRL A,R3	AB H	MOV R3,direct	EB H	MOV A,R3
2C H	ADD A,R4	6C H	XRL A,R4	AC H	MOV R4,direct	EC H	MOV A,R4
2D H	ADD A,R5	6D H	XRL A,R5	AD H	MOV R5,direct	ED H	MOV A,R5
2E H	ADD A,R6	6E H	XRL A,R6	AE H	MOV R6,direct	EE H	MOV A,R6
2F H	ADD A,R7	6F H	XRL A,R7	AF H	MOV R7,direct	EF H	MOV A,R7
30 H	JNB bit, rel	70 H	JNZ rel	B0 H	ANL C,bit	F0 H	MOVX @DPTR,A
31 H	ACALL addr11	71 H	ACALL addr11	B1 H	ACALL addr11	F1 H	ACALL addr11
32 H	RETI	72 H	ORL C, direct	B2 H	CPL bit	F2 H	MOVX @R0,A
33 H	RLC A	73 H	JMP @A+DPTR	B3 H	CPL C	F3 H	MOVX @R1,A
34 H	ADDC A,#data	74 H	MOV A,#data	B4 H	CJNE A,#data,rel	F4 H	CPL A
35 H	ADDC A, direct	75 H	MOV direct,#data	B5 H	CJNE A, direct, rel	F5 H	MOV direct, A
36 H	ADDC A,@R0	76 H	MOV @R0,#data	B6 H	cjne @r0,#data,rel	F6 H	MOV @R0,A
37 H	ADDC A,@R1	77 H	MOV @R1,#data	B7 H	cjne @r1,#data,rel	F7 H	MOV @R1,A
38 H	ADDC A,R0	78 H	MOV R0,#data	B8 H	cjne r0,#data,rel	F8 H	MOV R0,A
39 H	ADDC A,R1	79 H	MOV R1,#data	B9 H	cjne r1,#data,rel	F9 H	MOV R1,A
3A H	ADDC A,R2	7A H	MOV R2,#data	BA H	cjne r2,#data,rel	FA H	MOV R2,A
3B H	ADDC A,R3	7B H	MOV R3,#data	BB H	cjne r3,#data,rel	FB H	MOV R3,A
3C H	ADDC A,R4	7C H	MOV R4,#data	BC H	cjne r4,#data,rel	FC H	MOV R4,A
3D H	ADDC A,R5	7D H	MOV R5,#data	BD H	cjne r5,#data,rel	FD H	MOV R5,A
3E H	ADDC A,R6	7E H	MOV R6,#data	BE H	cjne r6,#data,rel	FE H	MOV R6,A
3F H	ADDC A,R7	7F H	MOV R7,#data	BF H	cjne r7,#data,rel	FF H	MOV R7,A

SOURCE mode

opcode	mnemonic sign	opcode	mnemonic sign	opcode	mnemonic sign	opcode	mnemonic sign
00 H	NOP	40 H	JC rel	80 H	SJMP rel	C0 H	PUSH direct
01 H	AJMP addr11	41 H	AJMP addr11	81 H	AJMP addr11	C1 H	AJMP addr11
02 H	LJMP addr16	42 H	ORL direct, A	82 H	ANL C,bit	C2 H	CLR bit
03 H	RR A	43 H	ORL direct,#data	83 H	MOVC A,@A+PC	C3 H	CLR C
04 H	INC A	44 H	ORL A,#data	84 H	DIV AB	C4 H	SWAP A
05 H	INC direct	45 H	ORL A, direct	85 H	MOV direct,direct	C5 H	XCH A, direct
06 H	-	46 H	-	86 H	-	C6 H	-
07 H	-	47 H	-	87 H	-	C7 H	-
08 H	JSLE rel	48 H	JSL rel	88 H	-	C8 H	-
09 H	MOV Rm,@WRj+dis	49 H	MOV WRj,@WRj+dis	89 H	LJMP @WRj EJMP @DRk	C9 H	-
0A H	MOVZ WRj,Rm	4A H	-	8A H	EJMP addr24	CA H	PUSH op1
0B H	INC R,#short MOV WRj,@DRk	4B H	-	8B H	-	CB H	-
0C H	-	4C H	ORL Rm,Rm	8C H	DIV Rm,Rm	CC H	-
0D H	-	4D H	ORL WRj,WRj	8D H	DIV WRj,WRj	CD H	-
0E H	SRA reg	4E H	ORL reg,op2	8E H	-	CE H	-
0F H	-	4F H	-	8F H	-	CF H	-
10 H	JBC bit,rel	50 H	JNC rel	90 H	MOV DPTR,#data16	D0 H	POP direct
11 H	ACALL addr11	51 H	ACALL addr11	91 H	ACALL addr11	D1 H	ACALL addr11
12 H	LCALL addr16	52 H	ANL direct, A	92 H	MOV bit,C	D2 H	SETB bit
13 H	RRC A	53 H	ANL direct,#data	93 H	MOVC A,@A+DPTR	D3 H	SETB C
14 H	DEC A	54 H	ANL A,#data	94 H	SUBB A,#data	D4 H	DA A
15 H	DEC direct	55 H	ANL A, direct	95 H	SUBB A,direct	D5 H	DJNZ direct, rel
16 H	-	56 H	-	96 H	-	D6 H	-
17 H	-	57 H	-	97 H	-	D7 H	-
18 H	JSG rel	58 H	JSGE rel	98 H	-	D8 H	-
19 H	MOV @WRj+dis,Rm	59 H	MOV @WRj+dis,WRj	99 H	LCALL @WRj ECALL @DRk	D9 H	-
1A H	MOVS WRj,Rm	5A H	-	9A H	ECALL addr24	DA H	POP op1
1B H	DEC R,#short MOV @DRk, WRj	5B H	-	9B H	-	DB H	-
1C H	-	5C H	ANL Rm,Rm	9C H	SUB Rm,Rm	DC H	-
1D H	-	5D H	ANL WRj,WRj	9D H	SUB WRj,WRj	DD H	-
1E H	SRL reg	5E H	ANL reg,op2	9E H	SUB reg,op2	DE H	-

1F H	-	5F H	-	9F H	SUB DRk,DRk	DF H	-
20 H	JB bit,rel	60 H	JZ rel	A0 H	ORL C, bit	E0 H	MOVX A,@DPTR
21 H	AJMP addr11	61 H	AJMP addr11	A1 H	AJMP addr11	E1 H	AJMP addr11
22 H	RET	62 H	XRL direct, A	A2 H	MOV C,bit	E2 H	MOVX A,@R0
23 H	RL A	63 H	XRL direct,#data	A3 H	INC DPTR	E3 H	MOVX A,@R1
24 H	ADD A,#data	64 H	XRL A,#data	A4 H	MUL AB	E4 H	CLR A
25 H	ADD A,direct	65 H	XRL A, direct	A5 H	ESC	E5 H	MOV A, direct
26 H	-	66 H	-	A6 H	-	E6 H	-
27 H	-	67 H	-	A7 H	-	E7 H	-
28 H	JLE rel	68 H	JE rel	A8 H	-	E8 H	-
29 H	MOV Rm,@DRk+dis	69 H	MOV WRj,@DRk+dis	A9 H	Bit instructions	E9 H	-
2A H	-	6A H	-	AA H	ERET	EA H	-
2B H	-	6B H	-	AB H	-	EB H	-
2C H	ADD Rm,Rm	6C H	XRL Rm,Rm	AC H	MUL Rm,Rm	EC H	-
2D H	ADD WRj,WRj	6D H	XRL WRj,WRj	AD H	MUL WRj,WRj	ED H	-
2E H	ADD reg,op2	6E H	XRL reg,op2	AE H	-	EE H	-
2F H	ADD DRk,DRk	6F H	-	AF H	-	EF H	-
30 H	JNB bit, rel	70 H	JNZ rel	B0 H	ANL C,bit	F0 H	MOVX @DPTR,A
31 H	ACALL addr11	71 H	ACALL addr11	B1 H	ACALL addr11	F1 H	ACALL addr11
32 H	RETI	72 H	ORL C, direct	B2 H	CPL bit	F2 H	MOVX @R0,A
33 H	RLC A	73 H	JMP @A+DPTR	B3 H	CPL C	F3 H	MOVX @R1,A
34 H	ADDC A,#data	74 H	MOV A,#data	B4 H	CJNE A,#data,rel	F4 H	CPL A
35 H	ADDC A, direct	75 H	MOV direct,#data	B5 H	CJNE A, direct, rel	F5 H	MOV direct, A
36 H	-	76 H	-	B6 H	-	F6 H	-
37 H	-	77 H	-	B7 H	-	F7 H	-
38 H	JG rel	78 H	JNE rel	B8 H	-	F8 H	-
39 H	MOV @DRk+dis,Rm	79 H	MOV @DRk+dis,WRj	B9 H	TRAP	F9 H	-
3A H	-	7A H	MOVH DRk,#data16 MOV op1,reg	BA H	-	FA H	-
3B H	-	7B H	-	BB H	-	FB H	-
3C H	-	7C H	MOV Rm,Rm	BC H	CMP Rm,Rm	FC H	-
3D H	-	7D H	MOV WRj,WRj	BD H	CMP WRj,WRj	FD H	-
3E H	SLL reg	7E H	MOV reg,op2	BE H	CMP reg,op2	FE H	-
3F H	-	7F H	MOV DRk,DRk	BF H	CMP DRk,DRk	FF H	-

A.2 Command Details

ACALL <addr11>

Function: Absolute call

Description: ACALL unconditionally calls the subroutine at the specified address. This instruction increments PC twice to obtain the address of the next instruction, then presses the 16-bit result onto the stack (low byte first) and increments the stack pointer twice. The destination address is obtained by concatenating the 5 high bits of the PC increment, bits 7 through 5 of the opcode, and the second byte of the instruction. Therefore, the calling subroutine must start in the same 2K block of program memory as the first byte of the instruction following ACALL. Flag bits are not affected.

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

Binary Mode **Source mode**

Instruction length: 2 2 2

Number of clocks. 3 3

Hex. Command Code Command Code

[Instruction Code] 11h, 31h, 51h, 71h, 91h, b1h, d1h, f1h

A10	A9	A8	1	0	0	0	1	A7	A6	A5	A4	A3	A2	A1	A0

Command Operation: ACALL ACALL

(PC) ← (PC) + 2

(SP) ← (SP) + 1

((SP)) ← (PC.7:0)

(SP) ← (SP) + 1

((SP)) ← (PC.15:8)

(PC.10:0) ← Page address

ADD <dest>,<src>

Function. The source number is added to the target operand.

Description. Adds the source operand to the destination operand, which can be a register or accumulator, leaving the result of the calculation in the register or accumulator. If bit 7 (CY) has a feed, the CY flag position bit.

If a byte variable is added and if there is an advance (AC) in bit 3, the AC flag bit is set. For unsigned integer addition, the CY flag bit indicates that an overflow has occurred. If bit 6 is rounded but bit 7 is not rounded, or if bit 7 is rounded but bit 6 is not rounded, the OV flag bit is in position. When signed integers are added, the OV flag bit indicates that the sum of two positive operands is negative or the sum of two negative operands is positive. Bits 6 and 7 in this description refer to the highest significant byte of the operand (8, 16, or 32 bits). The result affects the N and Z flag bits. The allowed addressing modes for source operands are register, direct, register indirect, and immediate number addressing.

ADD A,Rn

Command Operation. (PC) ←(PC) + 1

(A) ←(A) + (Rn)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 28H - 2FH

0	0	1	0	1	r	r	r

Binary Mode **Source** mode

Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

ADD A,Direct

Command Action. (PC) ←(PC) + 2
(A) ←(A) + (direct)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 25H

0 0 1 0	0 1 0 1	direct address
Binary mode	**Source mode**	

Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

ADD A,@Ri

Command Action. (PC) ←(PC) + 1
(A) ←(A) + ((Ri))

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 26H, 27H

0 0 1 0	0 1 1 i
Binary mode	**Source mode**

Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

ADD A,#DATA

Command Action. (PC) ←(PC) + 2
(A) ←(A) + #data

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 24H

0	0	1	0	0	1	0	0	immediate number

	Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

ADD Rmd,Rms

Command Operation. (PC) ←(PC) + 2

(Rmd) ←(Rms) + (Rmd)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 2CH

0 0 1 0	1 1 0 0	s s s s	S S S S

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ADD WRjd,WRjs

Command Action.	(PC)	←(PC) + 2
	(WRjd)	← (WRjs) + (WRjd)

Impact Flags.	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Command Code] 2DH

0 0 1 0	1 1 0 1	t t t t	T T T T

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ADD DRkd,DRks

Command Operation.	(PC)	←(PC) + 2
	(DRkd)	← (DRks) + (DRkd)

Impact Flags.	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Command Code] 2FH

0 0 1 0	1 1 1 1	u u u u	U U U U

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ADD Rm,#DATA

Command Operation.	(PC)	←(PC) + 3
	(Rm)	←(Rm) + #data

Impact Flags.	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Command Code] 2E(s)0H

0 0 1 0	1 1 1 0	s s s s	0 0 0 0	immediate number

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ADD WRj,#DATA16

Command Operation.		
	(PC)	←(PC) + 4
	(WRj)	←(WRj) + #data16

Impact Flags.	CY	AC	OV	N	Z

✓	-	✓	✓	✓

[Command Code] 2E(t)4H

0 0 1 0	1 1 1 0	t t t t	0 1 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ADD DRk,#0DATA16

Command Operation. (PC) ←(PC) + 4
(DRk) ←(DRk) + #data16

Impact Flags.

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Command Code] 2E(u)8H

0 0 1 0	1 1 1 0	u u u u	1 0 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ADD Rm,DIR8

Command operation. (PC) ←(PC) + 3
(Rm) ←(Rm) + (dir8)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 2E(s)1H

0 0 1 0	1 1 1 0	s s s s	0 0 0 1	direct address

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ADD WRj,DIR8

Command Operation. (PC) ←(PC) + 3
(WRj) ←(WRj) + (dir8)

Impact Flags.

CY	AC	OV	N	Z

✓	-	✓	✓	✓

[Command Code] 2E(t)5H

0 0 1 0	1 1 1 0	t t t t	0 1 0 1	direct address

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1 (2 for SFR) [A5] Instruction Code	1 (2 for SFR) command code
Hex.		

ADD Rm,DIR16

Command Operation. (PC) ←(PC) + 4

(Rm) ←(Rm) + (dir16)

Impact Flags.	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Command Code] 2E(s)3H

0 0 1 0	1 1 1 0	s s s s	0 0 1 1	Address High Byte	Address Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ADD WRj,DIR16

Command Operation. (PC) ←(PC) + 4

(WRj) ←(WRj) + (dir16)

Impact Flags.	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Command Code] 2E(t)7H

0 0 1 0	1 1 1 0	t t t t	0 1 1 1	Address High Byte	Address Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	2	2
Hex.	[A5] Instruction code	command code

ADD Rm,@DRk

Command Operation. (PC) ←(PC) + 3

(Rm) ←(Rm) + ((DRk))

Impact Flags.	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Command Code] 2E(u)B(s)0H

0 0 1 0	1 1 1 0	u u u u	1 0 1 1	s s s s	0 0 0 0

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	3	3
Hex.	[A5] Instruction	command code

code

ADDC A,<src-byte>

Function: Add A and the source operand, plus one (1) if CY is set, and put the result into A.

Description: The ADDC simultaneously adds the specified byte variable, the rounding flag bit, and the contents of the accumulator, leaving the result in the accumulator. If the 7th

If there is a rounding in bit 1 or bit 3, the rounding and auxiliary rounding flags are set to bits respectively, otherwise they are cleared to zero. When adding an unsigned integer, the The feed flag bit indicates that an overflow has occurred. If bit 6 is rounded but bit 7 is not, or if bit 7 is rounded but bit 6 is not, OV is set; otherwise, OV is cleared. When signed integers are added, OV indicates a negative sum of two positive operands or a positive sum of two negative operands, and the N and Z flag bits are affected by the result. The source operands allow four addressing modes: register, direct, register indirect, or immediate number addressing.

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

ADDC A,Rn

Command Operation.	(PC)	←(PC) + 1
	(A)	←(A) + (C) + (Rn)
[Command Code]	38H - 3FH	

0	0	1	1	1	r	r	r

	Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

ADDC A,DIRECT

Command Operation.	(PC)	←(PC) + 2
	(A)	←(A) + (C) + (direct)
[Command Code]	35H	

	Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

ADDC A,@Ri

Command Operation.	(PC)	←(PC) + 1
	(A)	←(A) + (C) + ((Ri))
[Command Code]	36H, 37H	

0	0	1	1	0	1	1	i

	Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1

Hex.	command code	[A5] Instruction code

ADDC A,#DATA

Command Action.	(PC)	←(PC) + 2
	(A)	←(A) + (C) + #data
[Command Code]	34H	

	Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1	1
Hex.	0 0 1 1 command code	0 1 0 0 immediate number 0 1 0 1 direct address

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the specified address, which is formed at runtime by concatenating the high 5 bits of PC (after PC has been incremented twice), bits 7 through 5 of the opcode, and the second byte of the instruction. Therefore, the target must be located in the same 2K block of programme memory as the first byte of the instruction following AJMP.

Impact Flags.	CY	AC	OV	N	Z

-	-	-	-	-

Command Operation. (PC) ←(PC) + 2

(PC10-0) ←Page address

[Instruction Code] 01h, 21h, 41h, 61h, 81h, a1h, c1h, e1h

a10	a9	a8	0	0	0	0	1	a7	a6	a5	a4	a3	a2	a1	a0

	Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	3	3
Hex.	command code	command code

ANL <dest>,<src>

Function: Logical sum of byte operands

Description: Performs a bitwise logical sum operation between the specified variables and stores the result in the target variable. These two operands allow 10 combinations of addressing modes. When the target is a register or accumulator, the source can be addressed using register, direct, register indirect, or immediate; when the target is a direct address, the source can be an accumulator or an immediate; the N and Z flag bits are affected depending on the result.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pin.

ANL A,Rn

Command Operation. (PC) ←(PC) + 1

(A) ←(A) and (Rn)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 58H - 5FH

0	1	0	1	1	r	r	r

	Binary Mode	**Source mode**
Instruction length: 1	1	2
Number of clocks: 1	1	1
Hex.	Command Code	[A5] Command Code

ANL A, Direct

Command Action. (PC) ←(PC) + 2

(A) ←(A) and (direct)

Impact Flags.

ANL A,@Ri

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 55H

	Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1	1
	command	command code

0 1 0 1	0 1 0 1	direct address

Command

Operation.

(PC)
←(PC) + 1

(A) ←(A) and ((Ri))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 56H, 57H

0 1 0 1	0 1 1 i
Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

ANL A,#DATA

Command Operation. (PC) ←(PC) + 2
(A) ←(A) and #data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 54H

0 1 0 1	0 1 0 0	immediate number
Binary mode	**Source mode**	
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

0 1 0 1	0 0 1 0	direct address

ANL Direct, A

Command Action. (PC) ←(PC) + 2
(direct) ←(direct) and (A)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 52H

0 1 0 1	0 0 1 1	direct address	immediate number
Binary mode	**Source mode**		
Instruction length.	2	2	
Number of clocks.	1 command	1 command code	

Hex.	code

ANL Direct,#DATA

Command Operation.	(PC)	←(PC) + 3
	(direct)	←(direct) and #data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 53H

	Binary mode	**Source mode**
Instruction length.	3	3
Number of clocks.	1	1
Hex.	command code	command code

ANL Rmd,Rms

Command Operation. (PC) ←(PC) + 2
(Rmd) ←(Rms) and (Rmd)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 5CH

0 1 0 1	1 1 0 0	s s s s	S S S S

Binary mode **Source mode**

Instruction length. 3 2

Number of clocks. 1 1

Hex. [A5] command code
Instruction code

ANL WRjd,WRjs

Command Operation. (PC) ←(PC) + 2
(WRjd) ←(WRjs) and (WRjd)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 2DH

0 0 1 0	1 1 0 1	t t t t	T T T T

Binary mode **Source mode**

Instruction length. 3 2

Number of clocks. 1 1

Hex. [A5] command code
Instruction code

ANL Rm,#DATA

Command Operation. (PC) ←(PC) + 3
(Rm) ←(Rm) and #data

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 5E(s)0H

0 1 0 1	1 1 1 0	s s s s	0 0 0 0	immediate number

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ANL WRj,#DATA16

Command Operation.	(PC)	←(PC) + 4
	(WRj)	←(WRj) and #data16

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 5E(t)4H

0 1 0 1	1 1 1 0	t t t t	0 1 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1

Hex. [A5] Instruction Code Command Code

ANL Rm,DIR8

Command operation. (PC) ←(PC) + 3
(Rm) ←(Rm) and (dir8)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 5E(s)1H

0 1 0 1	1 1 1 0	s s s s	0 0 0 1	direct address

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks. 1 1

Hex. [A5] Instruction code command code

ANL WRj,DIR8

Command Action. (PC) ←(PC) + 3
(WRj) ←(WRj) and (dir8)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 5E(t)5H

0 1 0 1	1 1 1 0	t t t t	0 1 0 1	direct address

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks. 1 (2 for SFR) 1 (2 for SFR)

Hex. [A5] Instructio n Code command code

ANL Rm,DIR16

Command Operation. (PC) ←(PC) + 4
(Rm) ←(Rm) and (dir16)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 5E(s)3H

0 1 0 1	1 1 1 0	s s s s	0 0 1 1	Address High Byte	Address Low Byte

Binary mode **Source mode**

Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ANL WRj,DIR16

Command Action.	(PC)	←(PC) + 4
	(WRj)	←(WRj) and (dir16)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 5E(t)7H

0 1 0 1	1 1 1 0	t t t t	0 1 1 1	high character address	classifier for segments, e.g. lessons, train wagons, biblical verses	Address Low Byte

Binary mode **Source mode**

Instruction length.	5	4
Number of clocks.	2	2
Hex.	[A5] Instruction code	comm and code

ANL Rm,@WRj

Command operation. (PC) ←(PC) + 3
(Rm) ←(Rm) and ((WRj))

CY	AC	OV	N	Z
-	-	-	✓	✓

Impact Flags.

[Command Code] 5E(t)9(s)0H

0	1	0	1	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0

Binary mode **Source mode**

Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ANL Rm,@DRk

Command Action. (PC) ←(PC) + 3
(Rm) ←(Rm) and ((DRk))

CY	AC	OV	N	Z
-	-	-	✓	✓

Impact Flags.

[Command Code] 5E(u)B(s)0H

0	1	0	1	1	1	1	0	u	u	u	u	1	0	1	1	s	s	s	s	0	0	0	0

Binary mode **Source mode**

Instruction length.	4	3
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

ANL C,<src-bit>

Function: Logical sum of bit operands

Description: If the Boolean value of the source bit is a logic 0, the rounding flag bit is cleared to zero; otherwise, the rounding flag bit remains in its current state. A slash ("/") before an operand in assembly language means that the logical complement of the addressed bit is used as the value of the source, but the source bit itself is not affected. No other flag

bits are affected. Only direct bit addressing is allowed as a source operand.

ANL C,bit51

Command Operation. (PC) ←(PC) + 2

(A) ←(C) and (bit51)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] 82H

1	0	0	0	0	0	1	0	bit address

Binary Mode **Source mode**

Instruction length: 2 2 2

Number of clocks: 1 1 1

Hex. Command Code Command Code

ANL C,/bit51

Command Action. (PC) ←(PC) + 2
(B) ←(C) and /(bit51)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] B0H

1 0 1 1	0 0 0 0	bit address

Binary mode	**Source mode**
Instruction length. 2	2
Number of clocks. 1	1
Hex. command code	command code

ANL C,bit

Command Action. (PC) ←(PC) + 3
(A) ←(C) and (bit)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] A98(y)H

1 0 1 0	1 0 0 1	1 0 0 0	0 y y y	direct addres

Binary mode	**Source mode**
Instruction length. 4	3
Number of clocks. 1	1
Hex. [A5] Instruction code	command code

ANL C,/bit

Command Action. (PC) ←(PC) + 3
(A) ←(C) and /(bit)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] A9F(y)H

1 0 1 0	1 0 0 1	1 1 1 1	0 y y y	direct addres

Binary mode	**Source mode**
Instruction length. 4	3

Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

CJNE <dest-byte>,<src-byte>,rel

Function: Compare, jump if not equal.

Description: CJNE compares the size of the first two operands and jumps if their values are not equal. The jump target is calculated by adding PC to the signed relative offset in the last byte of the instruction, and then incrementing PC to the beginning of the next instruction. If the unsigned integer

The value of <dest-byte> is less than the value of the unsigned integer <src-byte>, then the flag position bit is advanced; otherwise the advance is cleared. Neither operand is affected. The first two operands allow four combinations of addressing modes: the accumulator can be compared to any directly addressed byte or immediate number, and any indirect memory location or working register can be compared to an immediate number. Affects the C, N, and Z flag bits.

CJNE A, Direct, rel

Command Operation.

(PC) ←(PC) + 3
if (A) ≠ (direct)
then (PC) ←(PC) + relative offset
if (A) < (direct)
then (C) ←1
else (C) ←0

Impact Flags.	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Command Code]

B5H

1 0 1 1	0 1 0 1	direct address	relative address (computing)
Binary mode	**Source mode**		

Instruction length.	3	3
Number of clocks.	1/3 command code	1/3 command code
Hex.		

CJNE A,#DATA,rel

Command operation.

(PC) ←(PC) + 3
if (A) ≠ data
then (PC) ←(PC) + relative offset
if (A) < data
then (C) ←1
else (C) ←0

Impact Flags.	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Command Code]

B4H

1 0 1 1	0 1 0 0	immediate number	relative address (computing)
Binary mode	**Source mode**		

Instruction length.	3	3
Number of clocks.	1/3 command code	1/3 command code
Hex.		

CJNE Rn,#DATA,rel

Command Operation.

(PC) ←(PC) + 3
if (Rn) ≠ data
then (PC) ←(PC) + relative offset
if (Rn) < data
then (C) ←1
else (C) ←0

Impact Flags.	CY	AC	OV	N	Z

	✓	-	-	✓	✓

[Command Code] B8H - BFH

1	0	1	1	1	r	r	r	immediate number	relative address (computing)

	Binary mode	**Source mode**
Instruction length.	3	4
Number of clocks.	1/3 command	1/3 [A5]
Hex.	code	Instruction code

CJNE @Ri,#DATA,rel

Command Operation. (PC) ←(PC) + 3
if ((Ri)) ≠ data

then (PC) ←(PC) + relative offset
if ((Ri)) < data
then (C) ←1
else (C) ←0

Impact Flags.	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Command Code] B6H, B7H

	Binary mode	Source mode
Instruction length.	3	4
Number of clocks.	1/3	1/3
Hex.	1 0 1 1 command code	0 1 1 i [A5] Instruction code
		immediate number
		relative address (computing)

CLR A

Function. Zeroing the totaliser

Description. The accumulator is cleared to zero (all position bits are zero). Affects the N and Z flag bits.

Command Operation. (PC) ←(PC) + 1
(A) ←0

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] E4H

1 1 1 0	0 1 0 0

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks. 1 1

Hex. Command Code Command Code

CLR bit51

Function Bit Zero

Description. The specified bit is cleared (reset to zero). No other flag bits are affected.

Command Operation. (PC) ←(PC) + 2
bit ←0

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] C2H

1 1 0 0	0 0 1 0	bit address

Binary Mode **Source mode**

Instruction length: 2 2 2

Number of clocks: 1 1 1

Hex. Command Code Command Code

CLR C

Function. Progressive Zeroing

Description. The rounding flag bit is cleared (reset to zero). No other flag bits are affected.

Command Operation. (PC) ←(PC) + 1

(C) ←0

Impact Flags.

CY	AC	OV	N	Z

✓	-	-	-	-

[Command Code] C3H

1	1	0	0	0	0	1	1

Binary Mode **Source mode**

Instruction length. 1 1

Number of clocks. 1 1

Hex. Command Code command code

CLR bit

Function. Bit Zero

Description. The specified bit is cleared (reset to zero). No other flag bits are affected.

CY	AC	OV	N	Z
-	-	-	-	-

Command Operation. (PC) ←(PC) + 3

(bit) ←0

Impact Flags.

[Command Code] A9C(y)H

Binary Mode **Source mode**

Instruction length.

4 1	1	0	0	1 3	0	0	1	1	1	0	0	0	y	y	y	direct address

Number of clocks. 1 1

Hex. [A5] Instruction Code Instruction code

CMP <dest>,<src>

Function. Compare

Description. Subtracts the source operand from the target operand. The result is not stored in the destination operand. The CY (borrow) flag bit is set if bit 7 is to be borrowed; otherwise, it is cleared. When subtracting signed integers, the OV flag bit indicates a negative result for a positive number subtracted from a negative number, or a positive result for a negative number subtracted from a positive number. Bit 7 in this description refers to the highest significant byte of the operand (8, 16, or 32 bits). ac is affected only by the byte operand. the N and Z flag bits are affected depending on the result. The source operand allows four addressing modes: register, direct, immediate, and indirect.

CMP Rmd,Rms

Command Operation. (PC) ←(PC) + 2

(Rmd) -(Rms)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] BCH

1 0 1 1	1 1 0 0	s s s s	S S S S

Binary Mode **Source mode**

Instruction length: 3 3 2

Number of clocks: 1 1 1

Hex. [A5] Instruction Code Command Code

CMP WRjd,WRjs

Command Operation. (PC) ←(PC) + 2

(WRjd) -(WRjs)

Impact Flags.

CY	AC	OV	N	Z

✓	-	✓	✓	✓

[Command Code] BDH

1	0	1	1	1	1	0	1	t	t	t	t	T	T	T	T

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

CMP DRkd,DRks

Command Operation. (PC) ←(PC) + 2
(DRkd) -(DRks)

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Command Code] BFH

1	0	1	1	1	1	1	1	u	u	u	u	U	U	U	U

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

CMP Rm,#DATA

Command Operation. (PC) ←(PC) + 3
(Rm) -#data

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] BE(s)0H

1	0	1	1	1	1	1	0	s	s	s	s	0	0	0	0	immediate number

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction	command code

code

CMP WRj,#DATA16

Command Operation.	(PC)	←(PC) + 4
	(WRj)	-#data16

Impact Flags.

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Command Code] BE(t)4H

1 0 1 1	1 1 1 0	t t t t	0 1 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

CMP DRk,#0DATA16

Command Action.	(PC)	←(PC) + 4

(DRk) -#0data16

Impact Flags.

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Command Code] BE(u)8H

1 0 1 1	1 1 1 0	u u u u	1 0 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

CMP DRk,#1DATA16

Command Operation. (PC) ← (PC) + 4

(DRk) -#1data16

Impact Flags.

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Command Code] BE(u)CH

1 0 1 1	1 1 1 0	u u u u	1 1 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

CMP Rm,Dir8

Command Operation. (PC) ←(PC) + 3

(Rm) -(dir8)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] BE(s)1H

1 0 1 1	1 1 1 0	s s s s	0 0 0 1	direct address

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

Command Operation.

(PC)	←(PC) + 3
(WRj)	-(dir8)

Impact Flags.

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Command Code]

BE(t)5H

1	0	1	1	1	1	1	0	t	t	t	t	0	1	0	1	direct address

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1(2 for SFR)	1(2 for SFR)
Hex.	[A5] Instruction code	command code

CMP Rm,Dir16

Command Operation.	(PC)	←(PC) + 4
	(Rm)	-(dir16)

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] BE(s)3H

1 0 1 1	1 1 1 0	s s s s	0 0 1 1	Address High Byte	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

CMP WRj,Dir16

Command Operation.	(PC)	←(PC) + 4
	(WRj)	-(dir16)

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Command Code] BE(t)7H

1 0 1 1	1 1 1 0	t t t t	0 1 1 1	Address High Byte	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	4	3
	1(2 for SFR)	1(2 for SFR)
Number of clocks.	[A5]	command
Hex.	Instruction code	code

CMP Rm,@WRj

Command Operation.	(PC)	←(PC) + 3
	(Rm)	-((WRj))

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] BE(t)9(s)0H

1 0 1 1	1 1 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0		

BE(WRj)9(Rm)0

	Binary mode	**Source mode**
Instruction	4	3

length.

Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

CMP Rm,@DRk

Command Operation.	(PC)	←(PC) + 3
	(Rm)	-((DRk))

Impact Flags.	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Command Code] BE(u)B(s)0H

1	0	1	1	1	1	1	0	u	u	u	u	1	0	1	1	s	s	s	s	0	0	0	0

BE(DRk)B(Rm)0

Binary mode **Source mode**

Instruction length.	4	3
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

CPL A

Function. Accumulator inverse

Description. Each bit of the accumulator is logically inverted (one by one). Bits that were previously 1 will change to 0 and vice versa. Only affects N and Z Flag bits.

Command Operation. (PC) ←(PC) + 1
(A) ←/(A)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] F4H

1	1	1	1	0	1	0	0

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks: 1 1 1

Hex. Command Code Command Code

CPL Bit51

Function: Bit Inversion Bit Inversion

Description. Inverts the specified bit variable. A bit that was previously a 1 is changed to a zero, and vice versa. No other flags are affected.

Note: When this instruction is used to modify an output pin, the value used as raw data will be read from the output data latch, not the input pin.

Command Operation. (PC) ←(PC) + 2
(bit51) ←/(bit51)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] B2H

1	0	1	1	0	0	1	0	bit address

Binary Mode **Source mode**

Instruction length: 2 2 2

Number of clocks. 1 1

Hex. Command Code Command Code

CPL C

Function. Bitwise inverse

Description. Inverts the rounding flag bit. A bit that was previously a 1 is changed to a zero, and vice versa.

Command Operation. (PC) ←(PC) + 1

C) ←/(C)

Impact Flags.	CY	AC	OV	N	Z
	✓	-	-	-	-

[Command Code] B3H

1	0	1	1	0	0	1	1

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks: 1 1 1

Hex. Command Code Command Code

CPL Bit

Function: Bit Inversion Bit Inversion

Description. Inverts the specified bit.

Command Operation. (PC) ←(PC) + 3

(bit) ←/(bit)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] A9B(y)H

1	0	1	0	1	0	0	1	1	0	1	1	0	y	y	y	direct address

Binary Mode **Source mode**

Instruction length: 4 4 3

Number of clocks. 1 1

Hex. [A5] Instruction Code Command Code

DA A

Function: Adjust accumulator after decimal addition

Description: DA A adjusts the 8-bit value in the accumulator that is generated by adding the two previous variables (each in compressed BCD format) to produce two 4-bit numbers. Any ADD or ADDC instruction may be used to perform the addition. If bits 3 through 0 of the accumulator are greater than 9 (xxxx1010-xxxx1111), or if the AC flag bit is 1, the accumulator is added by 6 to generate the correct BCD number in the lower half-byte. This internal addition will set the rounding flag bit if the rounding in the lower four fields is transmitted through all the higher bits, otherwise it will not clear the rounding flag bit. If the rounding flag bit is now set, or if the four high bits now exceed nine
(1010xxxx-1111xxxx), these high bits will be added six to generate the correct BCD number in the high half byte. Similarly, if the high bits are rounded, the rounding flag bit will be set, but the rounding bit will not be cleared. Thus, the rounding flag bit indicates whether the sum of the original two BCD variables is greater than 100, allowing for a wide variety of accurate decimal additions. ov is unaffected. All of this occurs in a single instruction cycle. Essentially, the instruction performs a decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on the initial accumulator and PSW conditions. Affects the C, N, and Z flag bits.

Note: DA A cannot simply convert a hexadecimal number in the accumulator to BCD representation, nor does DA A apply to decimal subtraction.

Command Operation. **Flags.**

Impact

(PC) ← (PC) + 3
if [[(A3-0) > 9] ^ [(AC) = 1]]
then (A3-0) ← (A3-0) + 6
next if [[(A7-4) > 9] ^ [(C) = 1]]
then (A7-4) ← (A7-4) + 6

CY	AC	OV	N	Z
✓	-	-	✓	✓

[Command Code] B4H

1	0	1	1	0	1	0	0

	Binary mode	**Source mode**
Instruction length.	1	1
Number of clocks.	3	3
Hex.	command code	command code

DEC byte

Function. Decrement

Description. The specified variable minus 1. The original value of 00H will underflow to 0FFH. affects only the N and Z flag bits. The following addressing modes are allowed: accumulator, register, direct, or register indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pin.

DEC A

Command Operation.

(PC) ←(PC) + 1

(A) ←(A) - 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 14H

0 0 0 1	0 1 0 0

Binary mode	**Source mode**
Instruction length.	1	1
Number of clocks.	1	1
Hex.	command code	command code

DEC Rn

Command Operation.

(PC) ←(PC) + 1

(Rn) ←(Rn) - 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 18H - 1FH

0 0 0 1	1 r r r

Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

DEC Direct

Command Action.

(PC) ←(PC) + 2

(direct) ←(direct) - 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code]	15H		
	0 0 0 1	0 1 0 1	direct address
	Binary mode	**Source mode**	
Instruction length.	2	2	
Number of clocks.	1	1	
Hex.	command code	command code	

DEC @Ri

Command Operation. (PC) ←(PC) + 1

((Ri)) ← ((Ri)) - 1

Impact Flags.	CY	AC	OV	N	Z

-	-	-	✓	✓

[Command Code] 16H, 17H

0	0	0	1	0	1	1	i

Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

DEC <dest>,<src>

Function. Decreasing target value

Description. Decrements the variable specified by the destination operand by 1, 2, or 4. The original value 00H overflows to 0FFH. affects the N and Z flag bits. The decremented values are coded as follows:

vv	(be) worth
00	1
01	2
10	3

DEC Rm,#short

Command Operation.
(PC) ←(PC) + 2
(Rm) ←(Rm) - #short

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 1B(s)(00v)H

0	0	0	1	1	0	1	1	s	s	s	s	0	0	v	v

Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

DEC WRj,#short

Command Action.
(PC) ←(PC) + 2
(WRj) ←(WRj) - #short

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command 1B(t)(01v)H

Code]

0	0	0	1	1	0	1	1	t	t	t	t	0	1	v	v

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

DEC DRk,#short

Command operation.	(PC)	←(PC) + 2
	(DRk)	←(DRk) - #short

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 1B(u)(11v)H

0 0 0 1	1 0 1 1	u u u u	1 1 v v
Binary mode	**Source mode**		

Instruction length.	3	2
Number	1	1
of clocks.	[A5] Instruction	command
Hex.	code	code

DIV AB

DESCRIPTION: DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The integer remainder is the remainder of the integer. The accumulator holds the integer portion of the quotient, and register B holds the integer remainder. The rounding and OV flags are cleared to zero, and the N and Z flags are also affected by the result.

EXCEPTION: If B was originally placed into 00H, the values returned to the accumulator and B registers are undefined. In addition, the overflow flag bit. The carry flag bit is cleared in all cases. The other flag bits are undefined.

Command Operation. (PC) ←(PC) + 1

(A15-8) ← (A) / (B) - bit of result 15..8

(A7-0) ← (A) / (B) - bit of result 7..0

Impact Flags.

CY	AC	OV	N	Z
0	-	√*	√*	√*

*) - If B was originally placed in 00H, OV=1, N and Z undefined.

[Command Code] 84H

1 0 0 0	0 1 0 0
Binary Mode	**Source mode**

Instruction length:	1	1
Number of clocks.	6	6
Hex.	Command Code	Command Code

DIV <dest>,<src>

Description: Divide the unsigned integer in the register by the unsigned integer operand in register addressing mode, and clear the CY and OV flag bits. For byte operands (<dest>, <src> = Rmd,Rms), the result is 16-bit. 8-bit quotient is stored in the high byte of the word where Rmd is located, and 8-bit remainder is stored in the low byte of the word where Rmd is located. For example, register 1 is put into 251 (0FBH) and register 5 is put into 18 (12H). After executing DIV R1 instruction, R5 register 1 is put into 13 (0DH or 00001101B); register 0 is put into 17 (11H or 00010001B), because 251= (13×18) + 17; at the same time, CY and OV bits are cleared to zero. The CY flag bit is cleared. If the highest valid bit of the quotient is set, the N flag bit is set. If the quotient is zero, the Z flag bit is set.

Exception: if <src> is put into 00H, the values returned in both operands are undefined; the CY flag bit is cleared, OV

Flag position bits, the rest of the flag bits are undefined.

DIV Rmd,Rms

Command Operation.	(PC) ←(PC) + 2	
(Rmd)	Remainder (Rmd) / (Rms)	if <dest> md = 0,2,4, ,14
(Rmd+1)	Quotient (Rmd) / (Rms)	

(Rmd-1)	Remainder (Rmd) / (Rms)	if <dest> md = 1,3,5, ,15
(Rmd)	Quotient (Rmd) / (Rms)	

Impact Flags. *) - OV=1 if the source was originally placed in 0, N and Z are undefined.

CY	AC	OV	N	Z
0	-	✓*	✓*	✓*

[Command Code] 8CH

	Binary mode	Source mode
Instruction length.	3	2
	1 0 0 0 | 1 1 0 0 | s s s s | S S S S	
Number of clocks.	6	6
Hex.	[A5] Instruction code	command code

DIV WRjd,WRjs

Command Operation. (PC) ←(PC) + 2

(WRjd) Remainder(WRjd) / (WRjs) if<dest> jd = 0,4,8,...

,28 (WRjd+1) quotient (WRjd) / (WRjs)

(WRjd-1) Remainder(WRjd) / (WRjs) if<dest> jd = 2,6,10,...

,30 (WRjd) quotient (WRjd) / (WRjs)

For word operands (<dest>,<src> = WRjd,WRjs), the 16-bit quotient is in WR(jd+2) and the 16-bit remainder is in WRjd. For example, for target register WR4, assume that the quotient is 1122H and the remainder is 3344H. The result is then stored in these register file locations: (4)->0x33, (5)->0x44, (6)->0x11, (7)->0x22.

Impact Flags.

CY	AC	OV	N	Z
0	-	√*	√*	√*

*) - OV=1 if the source was originally placed in 0, N and Z are undefined.

[Command Code] 8DH

1 0 0 0	1 1 0 1	t t t t	T T T T

Binary Mode	Source mode
Instruction length: 3	3 2
Number of clocks.	10 10
Hex: [A5] Command Code	Command Code

DJNZ <byte>,<rel-addr>

Function: Decrement, jump if not zero

Description: DJNZ decrements the value at the specified location by one, and if the result is not zero, jumps to the address specified by the second operand. The original value of 00H will underflow to 0FFH. only the N and Z flag bits are affected. Calculate the jump target by adding PC to the signed relative offset in the last byte of the instruction, then increment PC to the first byte of the next instruction. The decremented location can be a register or a directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pin.

DJNZ Rn,rel

Command Operation. (PC)←(PC) + 2

(Rn) ← (Rn) - 1

if (Rn) ≠ 0

then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] D8H - DFH

1	1	0	1	1	r	r	r	relative address (computing)

Binary Mode **Source mode**

Instruction length: 2 2 3

Number of clocks.	1/3	1/3
Hex.	command code	[A5] Instruction code

DJNZ Direct, rel

Command Action.

(PC) ← (PC) + 3

CY	AC	OV	N	Z
-	-	-	✓	✓

if (direct) ← (direct) - 1

then (direct) ≠ 0

(PC) ← (PC) + rel

Impact Flags.

[Command Code] D5H

	Binary mode	**Source mode**		
Instruction length.	3	4		
Number of clocks.	3	3		
Hex.	1 1 0 1 command code	0 1 0 1 [A5] Instruction code	direct address	relative address (computing)

ECALL <dest>

Function: Extended Call

Description: Calls the subroutine at the specified address. This instruction adds 4 to the program counter to generate the address of the next instruction, then presses the 24-bit result onto the stack (high byte first) and increments the stack pointer by 3. The 8-bit high word and the 16-bit low word of the PC are then loaded into the second, third, and fourth bytes of the ECALL instruction, respectively. The programme continues to execute the instruction at that address. Thus, a subroutine can start anywhere in the entire 16MB of memory. The flag bits are not affected.

ECALL addr24

Command Operation.

(PC)	←(PC) + 4
(SP)	←(SP) + 1
((SP))	←(PC.23:16)
(SP)	←(SP) + 1
((SP))	←(PC.15:8)
(SP)	←(SP) + 1
((SP))	←(PC.7:0)
(PC)	←(addr.23:0)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 9AH

1 0 0 1	1 0 1 0	addr 23-16	addr 15-8	addr 7-0

	Binary Mode	Source mode
Instruction length: 5	5	4
Number of clocks.	3	3
Hex.	[A5] Instruction Code	Command Code

ECALL @DRk

Command Operation.	(PC)	←(PC) + 4
	(SP)	←(SP) + 1
	((SP))	←(PC.23:16)

	(SP)	←(SP) + 1
	((SP))	← (PC.15:8)
	(SP)	←(SP) + 1
	((SP))	←(PC.7:0)
	(PC)	←((DRk))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 99(u)8H

1 0 0 1	1 0 0 1	u u u u	1 0 0 0

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

EJMP <dest>

Function. Extended Jump

Description. An unconditional jump to the specified address is performed by loading the second, third, and fourth bytes of the instruction into the PC's 8-bit high word and 16-bit low word. Thus, the target can be anywhere in the entire 16MB memory space. Flag bits are not affected.

EJMP addr24

Command Operation.

CY	AC	OV	N	Z
(PC) -	←(addr.23:0) -	-	-	-

Impact Flags.

1 0 0 0	1 0 1 0	addr 23-16	addr 15-8	addr 7-0

[Command Code] 8AH

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

EJMP @DRk

Command Operation.

	(PC)	←((DRk))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command 89(u)8H

Code]

1 0 0 0	1 0 0 1	u u u u	1 0 0 0

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	3	3
Hex. [A5]	Instruction code	command code

ESC

Function. Switch to the opposite mode

Description. Execution of the immediately following instruction continues in the reverse mode. No registers or flags are affected except for PC.

Command Operation. (PC) ←(PC) + 1

Impact Flags.	CY	AC	OV	N	Z

-	-	-	-	-

[Command Code] A5H

1	0	1	0	0	1	0	1

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks. 1 1

Hex. Command Code Command Code

ERET

Function. Extended Returns

Description. Pop byte 3, byte 2, byte 1, and byte 0 of the PC from the stack in sequence and decrement the stack pointer by 3. Program execution continues at the result address, usually immediately following the instruction immediately following ECALL. Flag bits are not affected.

Command Operation. (PC.7:0) ¨((SP))

(SP) ¨(SP) - 1

(PC.15:8) (PC.15:8)

(SP) ¨(SP) - 1

(PC.23:16) (PC.23:16) ←((SP))

(SP) ←(SP) - 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] AAH

1	0	1	0	1	0	1	0

Binary Mode **Source mode**

Instruction length: 2 2 1

Number of clocks. 3 3

Hex. [A5] Instruction Code Command Code

INC byte

Functions: Incremental

Description: INC adds 1 to the specified variable. INC adds 1 to the specified variable; the original value of 0FFh overflows to 00h. affects only the N and Z flag bits. Allows three addressing modes: register, direct, or register indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pin.

INC A

Command Operation. (PC) ←(PC) + 1

(A) ←(A) + 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 04H

0	0	0	0	0	1	0	0

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks: 1 1 1

Hex. Command Code Command Code

INC Rn

Command Action.

(PC) ←(PC) + 1

(Rn) ←(Rn) + 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 08H - 0FH

0 0 0 0	1 r r r

Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

INC Direct

Command Operation.

(PC) ←(PC) + 1

(direct) ←(direct) + 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 05H

0 0 0 0	0 1 0 1	direct address

Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

INC @Ri

Command operation.

(PC) ←(PC) + 1

((Ri)) ←((Ri)) + 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 06H, 07H

0 0 0 0	0 1 1 i

Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

INC <dest>,<src>

Functions: Incremental

Description. Increments the specified variable by 1, 2, or 4. The original value of 0FFH overflows to 00H. Only the N and Z flags are affected. The incremental value is coded as follows:

vv	(be) worth
00	1
01	2
10	3

INC Rm,#short

Command Action. (PC) ←(PC) + 2
(Rm) ←(Rm) + #short

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 0B(s)(00v)H

0 0 0 0	1 0 1 1	s s s s	0 0 v v

Binary mode **Source mode**

Instruction length. 3 2

Number of clocks. 1 1

Hex. [A5] Instruction code command code

INC WRj,#short

Command Action. (PC) ←(PC) + 2
(WRj) ←(WRj) + #short

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 0B(t)(01v)H

0 0 0 0	1 0 1 1	t t t t	0 1 v v

Binary mode **Source mode**

Instruction length. 3 2

Number of clocks. 1 1

Hex. [A5] Instruction code command code

INC DRk,#short

Command Action. (PC) ←(PC) + 2
(DRk) ←(DRk) + #short

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 0B(u)(11v)H

0 0 0 0	1 0 1 1	u u u u	1 1 v v

Binary mode **Source mode**

Instruction length. 3 2

Number of clocks. 1 1

Hex. [A5] Instruction code command code

Function. Incremental data pointer

Description: Adds 1 to the 16-bit data pointer. performs a 16-bit increment (modulo 2^16); the low byte (DPL) of the data pointer is incremented to the high byte (DPH) when it overflows from 0FFH to 00H. A high byte (DPH) overflow does not increment the high word of the extended data pointer (DPX = DR56). Only the N and Z flag bits are affected.

Command Operation.

(PC) ←(PC) + 1

(DPTR) ←(DPTR) + 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] A3H

1	0	1	0	0	0	1	1

	Binary mode	Source mode
Instruction length.	1	1
Number of clocks.	1	1
Hex.	command code	command code

JB

Function: Jump if set

Explanation: If the specified bit is 1, then jump to the specified address; otherwise, continue to the next instruction. Calculates the jump target by adding PC to the signed relative offset in the third byte of the instruction, after which PC is incremented to the first byte of the next instruction. Does not modify the detected bit. Does not affect the sign bit.

JB Bit51,rel

Instruction operation. (PC)←(PC) + 3 if (bit51) = 1 then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 20H

0 0 1 0	0 0 0 0	bit address	relative address (computing)

	Binary Mode	Source mode
Instruction length: 3	3	3
Number of clocks.	1/3	1/3
Hex.	Command Code	Command Code

JB Bit,rel

Instruction operation. (PC)←(PC) + 3 if (bit) = 1 then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] A92(y)H

1 0 1 0	1 0 0 1	0 0 1 0	0 y y y	direct addres	relative address (computing)

JBC

	Binary mode	Source mode
Instruction length.	5	4

Number of clocks.	1/3 [A5]
Hex.	Instruction code
Fun	**ction:** If set, jump and clear the zero bit command **Description:** If the specified bit is 1, jump to the specified address; otherwise, continue to the next instruction. In either case, clear the specified bit. Calculates the jump target by adding PC to the signed relative offset in the third byte of the instruction, then increments PC to the first byte of the next instruction. Does not affect the sign bit. Note: When this instruction is used to test an output pin, the value used as raw data will be read from the output data latch, not the input pin.

JBC Bit51, rel

Instruction operation. (PC)←(PC) + 3 if (bit51) = 1 then (PC) ← (PC) + rel

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 10H

0 0 0 1	0 0 0 0	bit address	relative address (computing)

Binary mode **Source mode**

Instruction length.	3	3
Number of clocks.	1/3	1/3
Hex.	command code	command code

JBC Bit,rel

Instruction operation. (PC)←(PC) + 3 if (bit) = 1 then (PC) ← (PC) + rel

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] A91(y)H

1 0 1 0	1 0 0 1	0 0 0 1	0 y y y	direct addres	relative address (computing)

Binary mode **Source mode**

Instruction length.	5	4
Number of clocks.	1/3	1/3
Hex.	[A5] Instruction code	command code

JC

Function: Jump if a rounding bit is set

Description: If the rounding flag bit is set, jump to the specified address; otherwise, continue to the next instruction. Calculates the jump target by adding PC to the signed relative offset in the second byte of the instruction, after which PC is incremented twice. Does not modify the detected bit. Does not affect the sign bit.

Instruction operation. (PC)←(PC) + 2 if (C) = 1

then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 40H

	Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1/3 command	1/3 command
Hex.	code	code

JE

Function. Jump if equal

Description. If the Z flag bit is set, jump to the specified address; otherwise continue to the next instruction. Calculate the jump target by adding PC to the signed relative offset in the second byte of the instruction, after which PC is incremented twice.

Instruction operation. (PC)←(PC) + 2 if (Z) = 1 then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 68H

0	1	1	0	1	0	0	0	relative address (computing)

Binary mode **Source mode**

Instruction length. 3 2

Number of clocks. 1/3 [A5] 1/3 command

Hex. Instruction code code

JG

Function. Jump if greater than

Description. If both the Z flag bit and the CY flag bit are cleared, jump to the specified address; otherwise, continue to the next instruction. Calculate the jump target by adding PC to the signed relative offset in the second byte of the instruction, then increment PC twice.

Command Operation. (PC)←(PC) + 2 if (Z=0 and C=0) = 1 then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 38H

0	0	1	1	1	0	0	0	relative address (computing)

Binary mode **Source mode**

Instruction length. 3 2

Number of clocks. 1/3 [A5] 1/3 command

Hex. Instruction code code

JLE

Function. Jump if less than or equal to

Description. If both the Z flag bit and the CY flag bit are set, jump to the specified address; otherwise, continue to the next instruction. Calculate the jump target by adding PC to the signed relative offset in the second byte of the instruction, then increment PC twice.

Command Operation. (PC)←(PC) + 2 if (Z=1 and C=1) = 1

then (PC) ← (PC) + rel

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 28H

0 0 1 0	1 0 0 0	relative address (computing)

Binary Mode **Source mode**

Instruction length: 3 3 2

Number of clocks. 1/3 1/3

Hex. [A5] Command Code Command Code

JMP

Function. Indirect Jump

Description. Adds the 8-bit unsigned contents of the accumulator to the 16-bit data pointer and loads the result and into the programme counter. This will be the next
The address extracted by the instruction. Performs 16-bit addition (modulo 2^16): the lower 8 bits of the rounding are transferred to the higher bits. Neither the accumulator nor the data pointer is changed. Flag bits are not affected.

Command Operation. (PC) ←(A) + (DPTR)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 73H

0	1	1	1	0	0	1	1

Binary Mode **Source mode**

Instruction length: 1 1

Number of clocks. 1/3 1/3

Hex. Command Code Command Code

JNB

Function: Jump if bit not set

Explanation: If the specified bit is zero, jump to the specified address; otherwise, continue to the next instruction. Calculates the jump target by adding PC to the signed relative offset in the third byte of the instruction, after which PC is incremented to the first byte of the next instruction. Does not modify the detected bit. Does not affect the sign bit.

JNB Bit51, rel

Instruction operation. (PC)←(PC) + 3 if (bit51) = 0 then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 30H

0	0	1	1	0	0	0	0	bit address							relative address (computing)

JNB Bit,rel

Binary mode **Source mode**

Instruction operation. 3

length. (PC)←(PC) + 3 if (bit) = 0

Number 1/3 then (PC) ← (PC) +

of clocks. command rel command

Impact Hex. code CY code AC OV N Z

Flags.

-	-	-	-	-

[Command Code] A93(y)H

1	0	1	0	1	0	0	1	0	0	1	1	0	y	y	y	direct addres	relative address (computi

						ng)

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	1/3	1/3
Hex.	[A5] Instruction code	command code

JNC

Function. Jump if the feed is not set

Description. If the rounding flag bit is zero, jump to the specified address; otherwise, continue to the next instruction. Calculate the jump target by adding PC to the signed relative offset in the second byte of the instruction, after which PC is incremented to the next instruction.

Instruction operation. (PC)←(PC) + 2 if (C) = 0 then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 50H

0 1 0 1	0 0 0 0	relative address (computing)
Binary mode	**Source mode**	

Instruction length.	2	2
Number of clocks.	1/3 command	1/3 command
Hex.	code	code

JNE

Function. Jump if not equal

Description. If the Z flag bit is cleared, jump to the specified address; otherwise, continue to the next instruction. Calculate the jump target by adding PC to the signed relative offset in the second byte of the instruction, after which PC is incremented twice.

Instruction operation. (PC)←(PC) + 2 if (Z) = 0 then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 78H

0 1 1 1	1 0 0 0	relative address (computing)
Binary mode	**Source mode**	

Instruction length.	3	2
Number of clocks.	1/3 [A5] Instruction	1/3 command
Hex.	code	code

JNZ

Function. Jump if accumulator is not zero

Description. If any bit of the accumulator is 1, jump to the specified address; otherwise continue to the next instruction. Calculate the jump target by adding PC to the signed relative offset in the second byte of the instruction, after which PC is incremented twice.

Instruction operation. (PC)←(PC)

+ 2 if (A) ≠ 0

then (PC) ← (PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 70H

0	1	1	1	0	0	0	0	relative address (computing)

Binary Mode **Source mode**

Instruction length: 2 2 2

Number of clocks. 1/3 1/3

Hex. Command Code Command Code

JSG

Function. Jump if greater than (signed)

Description. If the Z flag bit is cleared and the N flag bit and the OV flag bit have the same value, jump to the specified address; otherwise continue

Next instruction. Calculate the jump target by adding PC to the signed relative offset in the second byte of the instruction, after which PC is incremented by Twice.

Command operation.

CY	AC	OV	N	Z
-	-	-	-	-

if (Z=0 and N=OV)
then (PC) ← (PC) + 2
(PC) ← (PC) + rel

Impact Flags.

[Command Code]

0	0	0	1	1	0	0	0	relative address (computing)

18H

	Binary Mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1/3	1/3
Hex.	[A5] Instruction Code	Instruction code

JSGE

Function. Greater than or equal to jump (signed)

Description. If the N flag bit and the OV flag bit have the same value, jump to the specified address; otherwise continue to the next instruction. Setting the PC Add the signed relative offset in the second byte of the instruction to calculate the jump target, after which the PC is incremented twice.

Command operation.

CY	AC	OV	N	Z
-	-	-	-	-

if (N=OV)
then (PC) ← (PC) + 2
(PC) ← (PC) + rel

Impact Flags.

[Command Code]

0	1	0	0	1	0	0	0	relative address (computing)

58H

	Binary Mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1/3	1/3
Hex.	[A5] Instruction Code	Instruction code

JSL

Function. Jump if less than (signed)

Description. If the N flag bit and the OV flag bit have different values, jump to the specified address; otherwise continue to the next instruction. Setting the PC
Add the signed relative offset in the second byte of the instruction to calculate the jump target, after which the PC is incremented twice.

Command Action.

(PC) ← (PC) + 2

if (N≠OV)

then (PC) ← (PC) + rel

Impact Flags.

[Command Code] 48H

Binary Mode	**Source mode**

Instruction length.	3	2
Number of clocks.	1/3	1/3
Hex.	[A5] Instruction code	command code

JSLE

Function. Jump if less than or equal to (signed)

Description. jumps to the specified address if the Z flag bit is set or if the N flag bit and the OV flag bit have different values; no then continue to the next instruction. Calculate the jump target by adding PC to the signed relative offset in the second byte of the instruction, after which the PC is incremented twice.

Command Operation.

(PC) ← (PC) + 2

if (Z=1 or (N≠OV))

then (PC) ← (PC) + rel

0 0 0 0	1 0 0 0	relative address (computing)

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 08H

	Binary Mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1/3	1/3
Hex.	[A5] Instruction Code	Instruction code

JZ

Function. Jump if accumulator is zero

Description. If all bits of the accumulator are zero, jump to the specified address; otherwise continue to the next instruction. Add the PC to the instruction second The signed relative offset in the bytes is used to calculate the jump target, after which the PC is incremented twice. Does not modify the accumulator. Does not affect the mark Chi Bit.

Command Operation.

(PC) ← (PC) + 2

if (A) = 0

then (PC) ← (PC) + rel

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 60H

	Binary mode	Source mode	
Instruction length. Number of clocks. Hex.	2 0 1 1 0 1/3 command code	2 0 0 0 0 1/3 command code	relative address (computing)

LCALL

Function: Long Call

Description: Calls the subroutine at the specified address. This instruction adds 3 to the program counter to generate the address of the next instruction, then presses the 16-bit result onto the stack (low byte first), and adds 2 to the stack pointer. the high and low bytes of the PC are then loaded into the second and third bytes of the LCALL instruction, respectively. The program continues to execute the instruction at that address. Thus, a subroutine can start anywhere in the entire 64KB program memory address space. The flag bits are not affected.

LCALL addr16

Command Operation.

(PC)	←(PC) + 3
(SP)	←(SP) + 1
((SP))	←(PC.7:0)
(SP)	←(SP) + 1
((SP))	← (PC.15:8)
(PC)	←(addr.15:0)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 12H

0	0	0	1	0	0	1	0	high character address	classifier for segments, e.g. lessons, train wagons, biblical verses	Address Low Byte					

Binary mode **Source mode**

Instruction length.	3	3
Number of clocks.	3	3
Hex.	command code	command code

LCALL @WRj

Command Action.		
	(PC)	←(PC) + 3
	(sp)	←(SP) + 1
	((sp))	←(PC.7:0)
	(sp)	←(SP) + 1
	((sp))	← (PC.15:8)
	(pc)	←((WRj))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 99(t)4H

1 0 0 1	1 0 0 1	t t t t	0 1 0 0

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

LJMP

Function. Long Jump

Description. LJMP performs an unconditional jump to a specified address by loading the second and third bytes of the instruction into the PC high and low bytes, respectively. Thus, the target can be anywhere in the entire 64KB program memory address space. Flag bits are not affected.

LJMP addr16

Command Operation. (PC) ←(addr.15:0)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 02H

0 0 0 0	0 0 1 0	Address High Byte	Address Low Byte

	Binary Mode	Source mode
Instruction length: 3	3	3
Number of clocks.	3	3
Hex.	Command Code	Command Code

LJMP @WRj

Command Action. (PC) ←((WRj))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 89(t)4H

1 0 0 0	1 0 0 1	t t t t	0 1 0 0

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

MOV

Function. Handling variables

Description. The variable specified by the second operand is copied to the location specified by the first operand. The source byte is not affected. No other registers or flag bits are affected.This is the most flexible operation ever.24 combinations of source and target addressing modes are allowed.

MOV A,Rn

Command Operation. (PC) ←(PC) + 1
(A) ←(Rn)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] E8H - EFH

1 1 1 0	1 r r r

	Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV A,Direct

Command Operation. (PC) ←(PC) + 2
(A) ← (direct)

1 1 1 0	0 1 0 1	direct address

Note: MOV A, ACC is a valid instruction.

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] E5H

	Binary mode	Source mode
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

MOV A,@Ri

Command Action.	(PC)	←(PC) + 1
	(A)	←((Ri))

Impact Flags.	CY	AC	OV	N	Z

-	-	-	-	-

[Command Code] E6H, E7H

1 1 1 0	0 1 1 i

	Binary mode	Source mode
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV A,#DATA

Command Operation. (PC) ←(PC) + 2
(A) ←#data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 74H

0 1 1 1	0 1 0 0	immediate number

	Binary mode	Source mode
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

MOV Rn,A

Command Action. (PC) ←(PC) + 1
(Rn) ←(A)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] F8H - FFH

1 1 1 1	1 r r r

	Binary mode	Source mode
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV Rn,Direct

Command (PC) ←(PC) + 2

Operation. (Rn) ← (direct)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] A8H - AFH

1 0 1 0	1 r r r	direct address

	Binary mode	**Source mode**
Instruction length.	2	3
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV Rn,#DATA

Command Operation. (PC) ←(PC) + 2

(Rn) ←#data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 78H - 7FH

0 1 1 1	1 r r r	direct address

	Binary mode	Source mode
Instruction length.	2	3
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV Direct,A

Command Action. (PC) ←(PC) + 2
(direct) ←(A)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] F5H

1 1 1 1	0 1 0 1	direct address

	Binary mode	Source mode
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV Direct,Rn

Command Operation. (PC) ←(PC) + 2
(direct) ←(Rn)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 88H - 8FH

1 0 0 0	1 r r r	direct address

	Binary mode	Source mode
Instruction	2	3

	length.	
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV Direct,Direct

Command Operation. (PC) ←(PC) + 3
(direct) ← (direct)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	.

[Command Code] 85H

1 0 0 0	0 1 0 1	source sites	destination direct address

	Binary mode	Source mode
Instruction length.	3	3
Number of clocks.	1	1
Hex.	command code	command code

MOV Direct,@Ri

Command Action. (PC) ←(PC) + 2
(direct) ←((Ri))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 86H, 87H

1 0 0 0	0 1 1 i	direct address

Binary mode **Source mode**

Instruction length. 2 3

Number of clocks. 1 1

Hex. command code [A5] Instruction code

MOV Direct,#DATA

Command Action. (PC) ←(PC) + 2
(direct) ←#data

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 75H

0 1 1 1	0 1 0 1	direct address	immediate number

Binary mode **Source mode**

Instruction length. 3 3

Number of clocks. 1 1

Hex. command code command code

MOV @Ri,A

Command Operation. (PC) ←(PC) + 1
((Ri)) ←(A)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] F6H, F7H

1 1 1 1	0 1 1 i

Binary mode **Source mode**

Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV @Ri,Direct

Command Operation.	(PC)	←(PC) + 2
	((Ri))	← (direct)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] A6H, A7H

1 0 1 0	0 1 1 i	direct address

	Binary mode	**Source mode**
Instruction length.	2	3
Number of clocks.	1	1

Hex. Command Code [A5] Command Code

MOV @Ri,#DATA

Command Action. (PC) ←(PC) + 2
((Ri)) ←#data

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 76H, 77H

0 1 1 1	0 1 1 i	immediate number

Binary mode	**Source mode**
Instruction length.	2	3
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

MOV Rmd,Rms

Command Operation. (PC) ←(PC) + 2
(Rmd) ← (Rms)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7CH

0 1 1 1	1 1 0 0	s s s s	S S S S

Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV WRjd,WRjs

Command Action. (PC) ←(PC) + 2
(WRjd) ← (WRjs)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7DH

---

0 1 1 1	1 1 0 1	t t t t	T T T T
Binary mode	**Source mode**		

Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV DRkd,DRks

Command Operation.	(PC)	←(PC) + 2
	(DRkd)	← (DRks)

CY	AC	OV	N	Z
Impact Flags.				
-	-	-	-	-

[Command Code] 7FH

0 1 1 1	1 1 1 1	u u u u	U U U U
Binary mode	**Source mode**		

Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	comm and code

MOV Rm,#DATA

Command Action.	(PC)	←(PC) + 3
	(Rm)	←#data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 7E(s)0H

0 1 1 1	1 1 1 0	s s s s	0 0 0 0	immediate number	

Binary mode	**Source mode**
4	3
1	1
[A5] Instruction code	command code

Instruction length.
Number of clocks.
Hex.

MOV WRj,#DATA16

Command operation.	(PC)	←(PC) + 4
	(WRj)	←#data16

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 7E(t)4H

0 1 1 1	1 1 1 0	t t t t	0 1 0 0	Immediate High Byte	Immediate Low Byte

Binary mode	**Source mode**
5	4
1	1
[A5] Instruction code	command code

Instruction length.
Number of clocks.
Hex.

MOV DRk,#0DATA16

Command Operation. Impact Flags.	CY	AC	OV	N	Z
	(PC) -	←(PC) + 4	-	-	-
	(DRk)	←#0data16			

[Command Code] 7E(u)8H

0	1	1	1	1	1	1	0	u	u	u	u	1	0	0	0	Immediate High Byte	Immediate Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV DRk,#1DATA16

Command Operation.	(PC)	←(PC) + 4
	(DRk)	←#1data16
Impact Flags.		

[Command Code] 7E(u)CH

0 1 1 1	1 1 1 0	u u u u	1 1 0 0	Immediate High Byte	Immediate Low Byte	

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV Rm,Dir8

Command Operation.	(PC)	←(PC) + 3
	(Rm)	←(dir8)

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7E(s)1H

0 1 1 1	1 1 1 0	s s s s	0 0 0 1	direct address

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV WRj,Dir8

Command Action.	(PC)	←(PC) + 3
	(WRj)	←(dir8)

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7E(t)5H

0 1 1 1	1 1 1 0	t t t t	0 1 0 1	direct address

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1(2 for SFR)	1(2 for SFR)
Hex.	[A5] Instruction code	command code

Command Operation.

(PC) ←(PC) + 3

(DRk) ←(dir8)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code]

7E(u)DH

0 1 1 1	1 1 1 0	u u u u	1 1 0 1	direct addres

	Binary mode	**Source mode**
Instruction length.	4	3
	2(4 for SFR)	2(4 for SFR)
Number of clocks.	[A5] Instruction code	command code
Hex.		

MOV Rm,Dir16

Command Operation.

(PC) ←(PC) + 4

(Rm) ←(dir16)

Impact Flags.

CY	AC	OV	N	Z

-	-	-	-	-

[Command Code] 7E(s)3H

0 1 1 1	1 1 1 0	s s s s	0 0 1 1	Address High Byte	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV WRj,Dir16

Command Operation. (PC) ←(PC) + 4
(WRj) ←(dir16)

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7E(t)7H

0 1 1 1	1 1 1 0	t t t t	0 1 1 1	high character address	classifier for segments, e.g. lessons, train wagons, biblical verses	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4

Number of clocks.	2	2
Hex.	[A5] Instruction code	command code

MOV DRk,Dir16

Command Operation. (PC) ←(PC) + 4
(DRk) ←(dir16)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7E(u)FH

0 1 1 1	1 1 1 0	u u u u	1 1 1 1	Address High Byte	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	2	2
Hex.	[A5] Instruction code	command code

MOV Rm,@WRj

Command Action. (PC) ←(PC) + 3
(Rm) ←((WRj))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7E(t)9(s)0H

0 1 1 1	1 1 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	2	2
Hex.	[A5] Instruction code	command code

MOV Rm,@DRk

Command Operation. (PC) ←(PC) + 3

(Rm) ←((DRk))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 7E(u)B(s)0H

0 1 1 1	1 1 1 0	u u u u	1 0 1 1	s s s s	0 0 0 0

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

MOV WRjd,@WRjs

Command Action. (PC) ←(PC) + 3

(WRjd) ←((WRjs))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 0B(u)8(s)0H

0 0 0 0	1 0 1 1	u u u u	1 0 0 0	s s s s	0 0 0 0

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV WRj,@DRk

Command Action. (PC) ←(PC) + 3

(WRj) ←((DRk))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 0B(u)A(t)0H

0 0 0 0	1 0 1 1	u u u u	1 0 1 0	t t t t	0 0 0 0

	Binary mode	Source mode
Instruction length.	4	3
Number	4	4
	[A5]	command

of clocks.	Instruction code	
Hex.	code	

MOV Dir8,Rm

Command Operation.	(PC)	←(PC) + 3
	(dir8)	←(Rm)

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7A(s)1H

0 1 1 1	1 0 1 0	s s s s	0 0 0 1	direct address

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV Dir8,WRj

Command Operation.	(PC)	←(PC) + 3
	(dir8)	←(WRj)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7A(t)5H

0 1 1 1	1 0 1 0	t t t t	0 1 0 1	direct address
Binary mode	**Source mode**			

Instruction length.	4	3
Number of clocks.	2	2
Hex.	[A5] Instruction code	command code

MOV Dir8,DRk

Command Action.	(PC)	←(PC) + 3
	(dir8)	← (DRk)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7A(u)DH

0 1 1 1	1 0 1 0	u u u u	1 1 0 1	direct addres
Binary mode	**Source mode**			

Instruction length.	4	3
Number of clocks.	2	2
Hex.	[A5] Instruction code	command code

MOV Dir16,Rm

Command Action.	(PC)	←(PC) + 4
	(dir16)	←(Rm)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7A(s)3H

0 1 1 1	1 0 1 0	s s s s	0 0 1 1	Address High Byte	Address Low Byte
Binary mode	**Source**				

		mode
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV Dir16,WRj

Command Action.	(PC)	←(PC) + 4
	(dir16)	←(WRj)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 7A(t)7H

0 1 1 1	1 0 1 0	t t t t	0 1 1 1	Address High Byte	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	2	2

Hex. [A5] Instruction Code Command Code

MOV Dir16,DRk

Command Operation. (PC) ←(PC) + 4
(dir16) ← (DRk)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7A(u)FH

0 1 1 1 1 0 1 0	u u u u	1 1 1 1	Address High Byte	Address Low Byte

Binary mode **Source mode**

Instruction length. 5 4

Number of clocks. 2 2

Hex. [A5] Instruction code command code

MOV @WRj,Rm

Command Operation. (PC) ←(PC) + 3
((WRj)) ←(Rm)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7A(t)9(s)0H

0 1 1 1 1 0 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks. 1 1

Hex. [A5] Instruction code command code

MOV @DRk,Rm

Command Action. (PC) ←(PC) + 3
((DRk)) ←(Rm)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7A(u)B(s)0H

0 1 1 1 1 0 1 0	u u u u	1 0 1 1	s s s s	0 0 0 0

Binary mode **Source**

		mode
Instruction length.	4	3
Number of clocks.	4	4
Hex.	[A5] Instruction code	command code

MOV @WRjd,WRjs

Command Operation.	(PC)	←(PC) + 3
	((WRjd))	← (WRjs)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 1B(t)8(T)0H

0 0 0 1	1 0 1 1	t t t t	1 0 0 0	T T T T	0 0 0 0
Binary mode	**Source mode**				

Instruction length.	4	3
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

MOV @DRk,WRj

Command Operation.	(PC)	←(PC) + 3
	((DRk))	←(WRj)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 1B(u)A(t)0H

0 0 0 1	1 0 1 1	u u u u	1 0 1 0	t t t t	0 0 0 0

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	6	6
Hex.	[A5] Instruction code	command code

MOV Rm,@WRj+dis

Command Operation.	(PC)	←(PC) + 4
	(Rm)	←((WRj)+dis)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 09H

0 0 0 0	1 0 0 1	s s s s	t t t t	Offset High Byte	Offset Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV WRjd,@WRjs+dis

Command Operation.	(PC)	←(PC) + 4
	(WRjd)	←((WRjs)+dis)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

-	-	-	-	-

[Command Code] 49H

0 1 0 0	1 0 0 1	t t t t	T T T T	Offset High Byte	Offset Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5]	command
	Instruction code	code

MOV Rm,@DRk+dis

Command Action.	(PC)	←(PC) + 4
	(Rm)	←((DRk)+dis)
Impact Flags.		

[Command Code] 29H

0 0 1 0	1 0 0 1	s s s s	u u u u	Offset High Byte	Offset Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

MOV WRj,@DRk+dis

Command Operation.	(PC)	←(PC) + 4
	(WRj)	←((DRk)+dis)

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 69H

0 1 1 0	1 0 0 1	t t t t	u u u u	Offset High Byte	Offset Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	4	4
Hex.	[A5] Instruction code	command code

MOV @WRj+dis,Rm

Command Operation.	(PC)	←(PC) + 4
	((WRj)+dis)	←(Rm)

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 19H

0 0 0 1	1 0 0 1	s s s s	t t t t	Offset High Byte	Offset Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV @WRjd+dis,WRjs

Command Action.	(PC)	←(PC) + 4
	((WRjd)+dis)	← (WRjs)

CY	AC	OV	N	Z
C̄Y	AC̄	OV̄	N̄	Z̄

[Command Code] 59H

0 1 0 1	1 0 0 1	T T T T	t t t t	Offset High Byte	Offset Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

MOV @DRk+dis,Rm

Command Action.	(PC)	←(PC) + 4
	((DRk)+dis)	←(Rm)
Impact Flags.		

-	-	-	-	-

[Command Code] 39H

0 0 1 1	1 0 0 1	s s s s	u u u u	Offset High Byte	Offset Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	4	4
Hex.	[A5] Instruction code	command code

MOV @DRk+dis,WRj

CY	AC	OV	N	Z
-	-	-	-	-

Command Action. (PC) ←(PC) + 4
((DRk)+dis) ←(WRj)

Impact Flags.

[Command Code] 79H

0 1 1 1	1 0 0 1	t t t t	u u u u	Offset High Byte	Offset Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	6	6
Hex.	[A5] Instruction code	command code

MOV <dest-bit>,<src-bit>

Function. Handling position data

Description. The Boolean variable specified by the second operand (directly addressable bit) is copied into the incoming flag bit. No other registers or flag bits are affected.

MOV C,Bit51

Command Operation. (PC) ←(PC) + 2
(C) ←(bit51)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] A2H

1 0 1 0	0 0 1 0	bit address

	Binary Mode	Source mode
Instruction length: 2	2	2
Number of clocks.	1	1
Hex.	Command Code	Command Code

MOV C,Bit

Command operation.

(PC) ←(PC) + 3

(C) ←(bit)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code]

A9A(y)H

1	0	1	0	1	0	0	1	1	0	1	0	y	y	y	y	direct addres

Instruction length.

Binary mode 4

Source mode 3

Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOV Bit51,C

Command Action. (PC) ←(PC) + 2
(bit51) ←(C)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 92H

1 0 0 1	0 0 1 0	bit address

Binary mode	**Source mode**
2	2
1 command code	1 command code

Instruction length.
Number of clocks.
Hex.

MOV Bit,C

Command Operation. (PC) ←(PC) + 3
(bit) ← (C)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] A99(y)H

1 0 1 0	1 0 0 1	1 0 0 1	y y y y	direct addres

Binary mode	**Source mode**
4	3
1	1
[A5] Instruction code	command code

Instruction length.
Number of clocks.
Hex.

1 0 0 1	0 0 0 0	Immediate High Byte	Immediate Low Byte

MOV DPTR,#DATA16

Function Description: Data Pointer Load 16-bit Constant
The data pointer is loaded with the specified 16-bit constant.The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high byte and the third byte (DPL) is the low byte. The flag bits are not affected.

(PC) ←(PC) + 3

Command	DPH	← Immediate 15...8
	DPL	← immediate number 7...0
operation.		
Impact Flags.		
[Command Code]	90H	

	Binary Mode	**Source mode**
Instruction length.	3	3
Number of clocks.	1	1
Hex.	Command Code	command code

MOVC

Function.	Carrying code bytes
Description.	The MOVC instruction loads a code byte or a constant from program memory into the accumulator. The address of the byte taken out is the original no

The sum of the contents of the symbolic 8-bit accumulator and the contents of the 16-bit base address register, which can be either the data pointer or the PC; in the latter case, the PC is incremented to the address of the next instruction and then added to the accumulator; otherwise, the base address register remains unchanged. Performs 16-bit addition so that the lower 8 bits of the rounding can be transferred to the higher bits. Flag bits are not affected.

MOVC A,@A+DPTR

Command Operation. (PC) ←(PC) + 1
(A) ←((A) + (DPTR))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 93H

1	0	0	1	0	0	1	1

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks. 4 4

Hex. Command Code Command Code

MOVC A,@A+PC

Command Operation. (PC) ←(PC) + 1
(A) ←(A) + (PC))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 83H

1	0	0	0	0	0	1	1

Binary mode **Source mode**

Instruction length. 1 1

Number of clocks. 3 3

Hex. command code command code

MOVH DRk,#DATA16

Function. Carries a 16-bit immediate number to the high word of the dword register.

Description. Carries a 16-bit immediate number to the high word of a double word (32-bit) register. The lower word of the double word register remains unchanged.

Command Operation. (PC) ←(PC) + 4
(DRk).31-16 ←#data16

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 7A(u)CH

0 1 1 1	1 0 1 0	u u u u	1 1 0 0	Immediate High Byte	Immediate Low Byte

Binary Mode **Source mode**

Instruction length: 5 5 4

Number of clocks: 1 1 1

Hex. [A5] Command Code Command Code

MOVS WRj,Rm

Function. Carry 8-bit registers to 16-bit registers with signed extensions

Description. Carries the contents of the 8-bit register to the low byte of the 16-bit register. the high byte of the 16-bit register is padded with a sign extension, the sign being obtained from the highest significant bit of the 8-bit source number register.

Command Operation.	(PC)	←(PC) + 2
	(WRj).7-0	←(Rm)
	(WRj).15-8	← Symbol Expansion

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 1AH

0	0	0	1	1	0	1	0	t	t	t	t	s	s	s	s

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MOVX

Function: Handling of external

Description: The MOVX instruction transfers data between the accumulator and a byte of external data memory, so an X is appended to the MOV. There are two types of instructions, the difference being whether they provide an 8-bit or 16-bit indirect address to external data memory. In the first type, the contents of the current register set R0 or R1 provide an 8-bit address, in the second type of MOVX instruction, the data pointer generates a 16-bit address.

MOVX A,@Ri

Command Operation.	(PC)	←(PC) + 1
	(A)	←((Ri))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] E2H, E3H

1	1	1	0	0	0	1	i

	Binary Mode	**Source mode**
Instruction length: 1	1	1
Number of clocks: 3	3*	3*
Hex.	Command Code	Command Code

MOVX A,@DPTR

Command Operation.	(PC)	←(PC) + 1
	(A)	←((DPTR))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] E0H

1	1	1	0	0	0	0	0

	Binary mode	**Source mode**

Instruction length.	1	1
Number of clocks.	2*	2*
Hex.	command code	command code

MOVX @Ri,A

Command	**Operation.**	(PC)	←(PC) + 1
		((Ri))	←(A)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] F2H, F3H

1 1 1 1	0 0 1 i

	Binary mode	Source mode
Instruction length.	1	1
Number of clocks.	3*	3*
Hex.	command code	command code

MOVX @DPTR,A

Command Operation. (PC) ←(PC) + 1

((DPTR)) ←(A)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] F0H

1 1 1 1	0 0 0 0

	Binary mode	Source mode
Instruction length.	1	1
Number of clocks.	3*	3*
Hex.	command code	command code

MOVZ WRj,Rm

Function. Carry 8-bit registers to 16-bit registers and extend zeros

Description. Carries the contents of the 8-bit register to the low byte of the 16-bit register. the high byte of the 16-bit register is padded with zeros.

Command Operation. (PC) ←(PC) + 2

(WRj).7-0 ←(Rm)

(WRj).15-8 ←0

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] 0AH

0 0 0 0	1 0 1 0	t t t t	s s s s

	Binary Mode	Source mode
Instruction length: 3	3	2
Number of clocks: 1	1	1
Hex.	[A5] Command Code	Command Code

MUL

Function: Multiplication

Description:	ger in the source register by an unsigned integer in the destination register. Only register addressing is allowed. For 8 Multiplies an unsigned integer The result is a 16-bit operand. The highest valid byte of the result is stored in the low byte of the word in which the destination register is located. The least significant byte is stored in the byte register immediately following. If the product is greater than 255(0FFH), the OV flag position bit; otherwise, it is cleared. For 16-bit operands, the result is 32 bits. The highest valid word is stored in the lower word of the double word in which the destination register is located. The lowest valid word is stored in the immediately following word register. In this operation, if the product is greater than 0FFFFH, the OV flag bit is set, otherwise it is cleared.The CY flag bit is always cleared. N Flag position bit when the highest valid byte of the result is set. Z Flag bit when the result is zero.

MUL AB

Command Operation.	(PC)	←(PC) + 1	
	(A)	←(A) × (B)	-Result bits 7..0
	(B)	←(A) × (B)	-Results bit 15.8

Impact Flags.	CY	AC	OV	N	Z
	0	-	✓	✓	✓

[Command Code] A4H

1	0	1	0	0	1	0	0

	Binary mode	**Source mode**
Instruction length.	1	1
Number of clocks.	1	1
Hex.	command code	command code

MUL Rmd,Rms

Command Action.		(PC) ← (PC) + 2
	if	<dest>md = 0,2,4... ,14
		Rmd ←Rmd × high byte of Rms
		Rmd+1←Low byte of Rmd × Rms
	if	<dest>md = 1,3,5... ,15
		Rmd-1 ←Rmd × high byte of Rms
		Rmd ←Rmd × low byte of Rms

Impact Flags.	CY	AC	OV	N	Z
	0	-	✓	✓	✓

1	0	1	0	1	1	0	0	s	s	s	s	S	S	S	S

[Command Code] ACH

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

MUL WRjd,WRjs

Command Operation.		(PC) ← (PC) + 2
	if	<dest>jd = 0,4,8... ,28
		WRjd ← WRjd × high byte of WRjs
		WRjd+2←WRjd × low byte of WRjs
	if	<dest>jd = 2,6,10... ,30
		WRjd-2 ← WRjd × high byte of WRjs
		WRjd ← WRjd × low byte of WRjs

1	0	1	0	1	1	0	1	t	t	t	T	T	T	T	T

Impact	CY	AC	OV	N	Z

Flags.					
	0	-	✓	✓	✓

[Command Code] ADH

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

NOP

Function. No operation

Description. Continues execution of the next instruction. Does not affect any registers or flags except PC.

Command Operation. (PC) ←(PC) + 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Instruction Code] 00H

0	0	0	0	0	0	0	0

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks. 1 1

Hex. Command Code Command Code

ORL

Function. Logical or of a variable

Description. Performs a bitwise logical or operation between the specified variables, storing the result in the target operand. The destination operand can be a register, an accumulator, or a direct address. The two operands allow 12 combinations of addressing modes. When the target is an accumulator, the source can be register, direct, register indirect, or immediate addressing; when the target is a direct address, the source can be an accumulator or immediate. When the target is a register, the source can be register, immediate, direct, and indirect addressing. Only the N and Z flag bits are affected. NOTE: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pin.

ORL A,Rn

Command Operation. (PC) ←(PC) + 1
(A) ←(A) or (Rn)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 48H - 4FH

0	1	0	0	1	r	r	r

Binary mode **Source mode**

Instruction length. 1 2

Number of clocks. 1 1

Hex. command code [A5] Instruction code

ORL A, Dir

Command Operation. (PC) ←(PC) + 2
(A) ←(A) or ( ) (direct)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 45H

0 1 0 0	0 1 0 1	direct address

	Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

ORL A,@Ri

Command Operation. (PC) ←(PC) + 1

(A) ←(A) or ((Ri))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 46H, 47H

0 1 0 0	0 1 1 i
Binary mode	**Source mode**

	Binary mode	Source mode
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

ORL A,#DATA

Command Action. (PC) ←(PC) + 1

(A) ←(A) or #data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 44H

0 1 0 0	0 1 0 0	immediate number
Binary mode	**Source mode**	

	Binary mode	Source mode
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

ORL Dir, A

Command Action. (PC) ←(PC) + 1

(direct) ←(direct) or (A)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 42H

0 1 0 0	0 0 1 0	direct address	immediate number
Binary mode	**Source mode**		

	Binary mode	Source mode
Instruction	2	2

length.		
Number of clocks.	1 command code	1 command code
Hex.		

ORL Dir,#DATA

Command Action.	(PC)	←(PC) + 1
	(direct)	←(direct) or #data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 43H

	Binary mode	**Source mode**
Instruction length.	3	3

Number of clocks.	1	1
Hex.	command code	command code

ORL Rmd,Rms

Command Action. (PC) ←(PC) + 2
(Rmd) ←(Rms) or (Rmd)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 4CH

0	1	0	0	1	1	0	0	s	s	s	s	S	S	S	S

Binary mode **Source mode**

Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ORL WRjd,WRjs

Command Action. (PC) ←(PC) + 2
(WRjd) ←(WRjs) or (WRjd)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 4DH

0	1	0	0	1	1	0	1	t	t	t	t	S	S	S	

Binary mode **Source mode**

Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ORL Rm,#DATA

Command Action. (PC) ←(PC) + 3
(Rm) ←(Rm) or #data

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 4E(s)0H

0	1	0	0	1	1	1	0	s	s	s	s	0	0	0	0	immed

					iate numbe r

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ORL WRj,#DATA16

Command Action.	(PC)	←(PC) + 4
	(WRj)	←(WRj) or #data16

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 4E(t)4H

0 1 0 0	1 1 1 0	t t t t	0 1 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ORL Rm,Dir8

Command Action.	(PC)	←(PC) + 3
	(Rm)	←(Rm) or ( ) (dir8)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 4E(s)1H

0 1 0 0	1 1 1 0	s s s s	0 0 0 1	direct address

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ORL WRj,Dir8

Command Action.	(PC)	←(PC) + 3
	(WRj)	←(WRj) or (dir8)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 4E(t)5H

0 1 0 0	1 1 1 0	t t t t	0 1 0 1	direct address

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1(2 for SFR)	1(2 for SFR)
Hex.	[A5] Instruction code	command code

ORL Rm,Dir16

Command Action.	(PC)	←(PC) + 4
	(Rm)	←(Rm) or (dir16)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 4E(s)3H

0 1 0 0	1 1 1 0	s s s s	0 0 1 1	Address High Byte	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ORL WRj,Dir16

Command Operation.	(PC)	←(PC) + 4
	(WRj)	←(WRj) or (dir16)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 4E(t)7H

0	1	0	0	1	1	1	0	t	t	t	t	0	1	1	1	Address High Byte	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	2	2
Hex.	[A5] Instruction code	command code

ORL Rm,@WRj

Command Operation. (PC) ←(PC) + 3
(Rm) ←(Rm) or ((WRj))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 4E(t)9(s)0H

0	1	0	0	1	1	1	0	t	t	t	t	1	0	0	1	s	s	s	s	0	0	0	0

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

ORL Rm,@DRk

Command Action. (PC) ←(PC) + 3
(Rm) ←(Rm) or ((DRk))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 4E(u)B(s)0H

0	1	0	0	1	1	1	0	u	u	u	U	1	0	1	1	s	s	s	s	0	0	0	0

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

ORL CY,<src-bit>

Function. Logical or of a bit variable
Description. If the Boolean value is a logical 1, the feed flags the location bit; otherwise

the feed remains in its current state. A slash ("/") before an operand in assembly language indicates that the logical complement of the addressed bit is used as the value of the source, but the source bit itself is not affected. No other flag bits are affected.

ORL C,Bit51

Command Operation. (PC) ←(PC) + 2

(C) ←(C) or (bit51)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] 72H

0	1	1	1	0	0	1	0	bit address

Binary Mode **Source mode**

Instruction length: 2 2 2

Number of clocks: 1 1 1

Hex. Command Code Command Code

ORL C,/Bit51

Command Operation. (PC) ←(PC) + 2
(C) ←(C) or /(bit51)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] A0H

Binary mode	**Source mode**
1 0 1 0 0 0 0 0	bit address

Instruction length. 2 2

Number of clocks. 1 1

Hex. command code command code

ORL C,Bit

Command Action. (PC) ←(PC) + 2
(C) ←(C) or (bit)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] A97(y)H

1 0 1 0	1 0 0 1	0 1 1 1	0 y y y	direct addres

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks. 1 1

Hex. command code command code

ORL C,/Bit

Command Operation. (PC) ←(PC) + 2
(C) ←(C) or /(bit)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] A9E(y)H

1 0 1 0	1 0 0 1	1 1 1 0	0 y y y	direct addres

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks. 1 1

Hex. [A5] Instruction code command code

POP

Function. Popup Stack

Description. Reads the contents of an on-chip memory location addressed by the stack pointer and then decrements the stack pointer by 1. The value read in the previous memory location is transferred to the newly addressed location, which can be either 8-bit or 16-bit. Flag bits are not affected.

POP Dir8

Command Operation. (PC) ←(PC) + 2

(dir8) ←((SP))

(SP) ←(SP) - 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] D0H

1 1 0 1	0 0 0 0	direct address

Binary Mode **Source mode**

Instruction length: 2 2 2

Number of clocks. 1 1

Hex. Command Code Command Code

POP Rm

Command Operation. (PC) ←(PC) + 2

(Rm) ←((SP))

(SP) ←(SP) - 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Instruction Code] DA(s)8H

1 1 0 1	1 0 1 0	s s s s	1 0 0 0

Binary Mode **Source mode**

Instruction length: 3 3 2

Number of clocks. 1 1

Hex. [A5] Instruction Code Command Code

POP WRj

Command Operation. (PC) ←(PC) + 2

(WRj) ←((SP))

(SP) ←(SP) - 2

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] DA(t)9H

1 1 0 1	1 0 1 0	t t t t	1 0 0 1

Binary Mode **Source mode**

Instruction length: 3 3 2

Number of clocks. 1 1

Hex. [A5] Instruction Code Command Code

POP DRk

Command Operation. (PC) ←(PC) + 2

(DRk) ←((SP))

(SP) ←(SP) - 3

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] DA(u)BH

1 1 0 1	1 0 1 0	u u u u	1 0 1 1

Binary Mode **Source mode**

Instruction length: 3 3 2

Number of clocks. 1 1

Hex. [A5] Instruction Code Command Code

PUSH

Function. Pressing into the stack

Description. Adds the stack pointer by 1. Then copies the contents of the specified variable to the on-chip memory location addressed by the stack pointer. Flag bits are not affected.

PUSH Dir8

Command Operation.

(PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←(dir8)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] C0H

1 1 0 0	0 0 0 0	direct address

Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

PUSH #DATA

Command Operation.

(PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←#data

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] CA02H

1 1 0 0	1 0 1 0	0 0 0 0	0 0 1 0	immediate number

Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

Command Operation.

(PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←MSB #data
(SP) ←(SP) + 1
((SP)) ←LSB #data

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] CA06H

1	1	0	0	1	0	1	0	0	0	0	0	0	1	1	0	Immediate High Byte	Immediate Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1

Hex. [A5] Instruction Code Command Code

PUSH Rm

Command Operation.

(PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←(Rm)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] CA(s)8H

1 1 0 0	1 0 1 0	s s s s	1 0 0 0

Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex. [A5]	Instruction code	command code

PUSH WRj

Command Action.

(PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ←(WRj)
(SP) ←(SP) + 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] CA(t)9H

1 1 0 0	1 0 1 0	t t t t	1 0 0 1

Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex. [A5]	Instruction code	command code

PUSH DRk

Command Operation.

(PC) ←(PC) + 2
(SP) ←(SP) + 1
((SP)) ← (DRk)
(SP) ←(SP) + 3

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code]	CA(u)BH															
	1	1	0	0	1	0	1	0	u	u	u	u	1	0	1	1

	Binary mode	Source mode
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

RET

Function. Return from subroutine

Description. RET pops the high and low bytes of PC from the stack in turn. the stack pointer is subtracted by 2. the program continues at the result address, through the

Often followed by ACALL or LCALL. Flags are not affected.

Command Operation.

(PC15-8) ←((SP))
(SP) ←(SP) - 1
(PC7-0) ←((SP))
(SP) ←(SP) - 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 22H

0	0	1	0	0	0	1	0

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks. 3 3

Hex. Command Code Command Code

RETI

Function. Return from interrupt

Description. This instruction pops two or four bytes from the stackdepending on the INTR bit in the CONFIG1 registerIf the INTR = 0,RETI pops the high and low bytes of PC off the stack in sequence and uses them as the 16-bit return address of the FF: region. The stack pointer is decremented by 2. No other registers are affected, and neither PSW nor PSW1 is automatically restored to its preinterrupt state. If INTR = 1, RETI pops four bytes from the stack: PSW1 and the three bytes of PC. the three bytes of PC are the return address, which can be anywhere in the 16MB of memory. the stack pointer minus four. The stack pointer is decremented by four. psw1 is restored to its pre-interrupt state, but psw is not restored. No other registers are affected. For any value of INTR. the hardware restores the interrupt logic to receive other interrupts with the same priority as the one just processed. Program execution continues at the return address, which is usually the instruction after the interrupt request was detected. If a RETI instruction is executed while an interrupt of the same or lower priority is waiting, it is executed before the waiting interrupt is processed.

Command Operation.

INTR=1.		INTR=0.	
(PC15-8)	←((SP))	(PC15-8)	←((SP))
(SP)	←(SP) - 1	(sp)	←(SP) - 1
(PC7-0)	←((SP))	(pc7-0)	←((SP))
(SP)	←(SP) - 1	(sp)	←(SP) - 1
(pc23-16)	←((SP))		
(sp) psw1	←(SP) - 1		
(SP)	←((SP))		
	←(SP) - 1		

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 32H

0	0	1	1	0	0	1	0

Binary mode **Source mode**

Instruction 1 1

length.		
Number of clocks.	3	3
Hex.	command code	command code

RL

Function. Accumulator Cycle Left

Description. The eight bits in the accumulator are cycled one bit to the left. Bit 7 loops to bit 0. Only the N and Z flags are affected.

Command Operation. (PC) ←(PC) + 1
(An + 1) ←(An) n = 0-6
(A0) ← (A7)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 23H

0	0	1	0	0	0	1	1

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks: 1 1 1

Hex. Command Code Command Code

RLC

Function. Cyclic left shift with rounding flag accumulator

Description. The eight bits in the accumulator are shifted cyclically one bit to the left along with the rounding flag bits. Bit 7 is shifted into the rounding flag bit; the previous state of the rounding flag bit is shifted into the position of bit 0. The N and Z flags are also affected.

Command Operation. (PC) ←(PC) + 1
(An + 1) ←(An) n = 0-6
(A0) ← (C)
(C) ← (A7)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	✓	✓

[Command Code] 33H

0	0	1	1	0	0	1	1

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks. 1 1

Hex. Command Code Command Code

RR

Function. Accumulator loop right shift

Description. The eight bits in the accumulator are cycled one bit to the right. Bit 0 cycles to bit 7. Only the N and Z flags are affected.

Command Operation. (PC) ←(PC) + 1
(An) ←(An + 1) n = 0-6
(A7) ← (A0)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 03H

0	0	0	0	0	0	1	1

Binary Mode **Source mode**

Instruction length: 1 1 1

Number of clocks: 1 1 1

Hex. Command Code Command Code

RRC

Function. Cyclic right shift with rounding flag accumulator

Description. The eight bits in the accumulator are cyclically shifted one bit to the right along with the rounding flag bits. Bit 0 is shifted into the rounding flag bit; the previous state of the rounding flag bit is shifted into the position of bit 7. The N and Z flag bits are also affected.

Command Operation. (PC) ←(PC) + 1
(An) ←(An + 1) n = 0-6
(A7) ← (C)
(C) ← (A0)

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	✓	✓

[Command Code] 13H

0 0 0 1	0 0 1 1
Binary Mode	**Source mode**

Instruction length: 1 1 1
Number of clocks. 1 1
Hex. Command Code Command Code

SETB

Function Setting
Description. SETB sets the specified bit to 1. SETB can operate on the rounding flag bit or any directly addressable bit. SETB can operate on the rounding flag bit or any directly addressable bit without affecting other flag bits.

SETB C

Command Operation. (PC) ←(PC) + 1
(C) ←1

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	-	-

[Command Code] D3H

1 1 0 1	0 0 1 1
Binary Mode	**Source mode**

Instruction length: 1 1 1
Number of clocks: 1 1 1
Hex. Command Code Command Code

SETB Bit51

SETB Bit
Command operation. (PC) ←(PC) + 2
(bit51) ←1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] D2H

1 1 0 1	0 0 1 0	bit address
Binary mode	**Source mode**	

Instruction length. 2 2
Number of clocks. 3 command code 3 command code

Command Operation.		
	(PC)	←(PC) + 3

(bit) ←1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] A9D(y)H

1 0 1 0	1 0 0 1	1 1 0 1	0 y y y	direct addres

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

SJMP

Function: Short Jump

Description: The programme controls an unconditional jump to the specified address. The jump target is calculated by adding PC to the signed relative offset in the second byte of the instruction, then PC is incremented twice. Thus, the range of allowable targets is from 128 bytes before the instruction to 127 bytes after it. The sign bit is not affected.

Command Operation. (PC) ←(PC) + 2

(PC) ←(PC) + rel

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] 80H

1 0 0 0	0 0 0 0	relative address (computing)

	Binary Mode	**Source mode**
Instruction length: 2	2	2
Number of clocks.	3	3
Hex.	Command Code	Command Code

SLL

Function: Logical left shift by 1 bit Logical left shift by 1 bit

Description. Shifts the specified variable left by 1 bit, replacing the least significant bit with zero. The shifted bit is stored in the CY bit as the most significant bit (MSB).

The N and Z flag bits are also affected.

SLL Rm

Command Operation. (PC) ←(PC) + 2

(Rm).a + 1 ← (Rm).a

(Rm).0 ←0

CY ←(Rm).7

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	✓	✓

[Command Code] 3E(s)0H

0 0 1 1	1 1 1 0	s s s s	0 0 0 0

Binary Mode **Source mode**

Instruction length: 3 3 2

Number of clocks. 1 1

Hex. [A5] Instruction Code Command Code

SLL WRj

Command Operation.	(PC)	←(PC) + 2
	(WRj).b + 1	←(WRj).b
	(WRj).0	←0
	CY	←(WRj).15

Impact Flags.	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Command Code] 3E(t)4H

0	0	1	1	1	1	1	0	t	t	t	t	0	1	0	0

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1 [A5]	1 command
Hex. code	Instruction code	

SRA

Function. Arithmetic right shift by 1 bit (signed)

Description. Shifts the specified variable arithmetically right by 1 bit. The highest significant bit remains unchanged. The shifted bit (LSB) is stored in the CY bit. n and z Flag bits are also affected.

SRA Rm

Command Operation.	(PC)	←(PC) + 2
	(Rm).7	←(Rm).7
	(Rm).a	←(Rm).a+1
	CY	←(Rm).0

Impact Flags.	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Command Code] 0E(s)0H

0	0	0	0	1	1	1	0	s	s	s	s	0	0	0	0

	Binary Mode	**Source mode**
Instruction length: 3	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction Code	Command Code

SRA WRj

Command Operation.	(PC)	←(PC) + 2
	(WRj).15	←(WRj).15
	(WRj).b	←(WRj).b + 1
	CY	←(WRj).0

Impact Flags.	CY	AC	OV	N	Z
	✓	-	-	✓	✓

[Command Code] 0E(t)4H

0	0	0	0	1	1	1	0	t	t	t	t	0	1	0	0

	Binary Mode	**Source mode**

Instruction length: 3	3	2
Number of clocks.	1	1
Hex. [A5] Instruction Code		Command Code

SRL

Function: Logical right shift by 1 bit Logical right shift by 1 bit

Description. SRL shifts the specified variable right by 1 bit, replacing the highest significant bit with zero. The shifted bit (LSB) is stored in the CY bit.N and Z

Flag bits are also affected.

SRL Rm

Command Operation. (PC) ←(PC) + 2

(Rm).7 ←(Rm).0

(Rm).a ←(Rm).a+1

CY ←(Rm).0

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	✓	✓

[Command Code] 1E(s)0H

0 0 0 1	1 1 1 0	s s s s	0 0 0 0

Binary Mode **Source mode**

Instruction length: 3 3 2

Number of clocks. 1 1

Hex. [A5] Instruction Code Command Code

SRL WRj

Command Operation. (PC) ←(PC) + 2

(WRj).15 ←0

(WRj).b ←(WRj).b + 1

CY ←(WRj).0

Impact Flags.

CY	AC	OV	N	Z
✓	-	-	✓	✓

[Command Code] 1E(t)4H

0 0 0 1	1 1 1 0	t t t t	0 1 0 0

Binary Mode **Source mode**

Instruction length: 3 3 2

Number of clocks. 1 1

Hex. [A5] Instruction Code Command Code

SUB

Function. Phase Decrease

Description: Subtracts the specified variable from the target operand, leaving the result in the target operand. If bit 7 needs to be borrowed, then SUB sets the CY flag bit (borrow), otherwise clear the CY bit. When signed integers are subtracted, the OV flag bit indicates a negative number when a positive number is subtracted from a negative number, or a positive result when a negative number is subtracted from a positive number. Bit 7 in this description refers to the highest valid byte of the operand.
(8, 16, or 32 bits). The source operand allows four addressing modes: immediate, indirect, register, and direct addressing. All flag bits are affected except AC, which does not affect word and double word subtraction.

SUB Rmd,Rms

Command Operation. (PC) ←(PC) + 2

(Rmd) ←(Rms) - (Rmd)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 9CH

1	0	0	1	1	1	0	0	s	s	s	s	S	S	S	S

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

SUB WRjd,WRjs

Command Operation. (PC) ←(PC) + 2
(WRjd) ← (WRjs) - (WRjd)

Impact Flags.	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Command Code] 9DH

1	0	0	1	1	1	0	1	t	t	t	t	T	T	T	T

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

SUB DRkd,DRks

Command Operation. (PC) ←(PC) + 2
(DRkd) ←(DRks) - (DRkd)

Impact Flags.	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Command Code] 9FH

1	0	0	1	1	1	1	1	u	u	u	u	U	U	U	U

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	4	4
Hex.	[A5] Instruction code	command code

SUB Rm,#DATA

Command Action. (PC) ←(PC) + 3
(Rm) ←(Rm) - #data

Impact	CY	AC	OV	N	Z

Flags.					
	✓	✓	✓	✓	✓

[Command Code] 9E(s)0H

1	0	0	1	1	1	1	0	s	s	s	s	0	0	0	0	immediate number

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

SUB WRj,#DATA16

Command Operation. (PC) ←(PC) + 4

(WRj) ←(WRj) - #data16

Impact Flags.	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Command Code] 9E(t)4H

1 0 0 1	1 1 1 0	t t t t	0 1 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

SUB DRk,#0DATA16

Command operation.	(PC) ← (PC) + 4 (DRk) ←(DRk) - #data16

Impact Flags.	CY	AC	OV	N	Z
	✓	-	✓	✓	✓

[Command Code] 9E(u)8H

1 0 0 1	1 1 1 0	u u u u	1 0 0 0	Immediate High Byte	Immediate Low Byte

	Binary mode	Source mode
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

SUB Rm,Dir8

Command Operation.	(PC) ←(PC) + 3 (Rm) ←(Rm) - (dir8)

Impact Flags.	CY	AC	OV	N	Z
	✓	✓	✓	✓	✓

[Command Code] 9E(s)1H

1 0 0 1	1 1 1 0	s s s s	0 0 0 1	direct address

	Binary mode	Source mode
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

SUB WRj,Dir8

Command Operation.	(PC) ←(PC) + 3 (WRj) ←(WRj) - (dir8)

Impact	CY	AC	OV	N	Z

Flags.					
	✓	-	✓	✓	✓

[Command Code] 9E(t)5H

1 0 0 1	1 1 1 0	t t t t	0 1 0 1	direct address

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1(2 for SFR)	1(2 for SFR)
Hex.	[A5] Instruction code	command code

SUB Rm,Dir16

Command Operation. (PC) ←(PC) + 4
(Rm) ←(Rm) -)
(dir16)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 9E(s)3H

1 0 0 1	1 1 1 0	s s s s	0 0 1 1	Address High Byte	Address Low Byte

Binary mode **Source mode**

Instruction length. 5 4

Number of clocks. 1 1

Hex. [A5] Instruction code command code

SUB WRj,Dir16

Command Operation. (PC) ←(PC) + 4
(WRj) ←(WRj) - (dir16)

Impact Flags.

CY	AC	OV	N	Z
✓	-	✓	✓	✓

[Command Code] 9E(t)7H

1 0 0 1	1 1 1 0	t t t t	0 1 1 1	Address High Byte	Address Low Byte

Binary mode **Source mode**

Instruction length. 5 4

Number of clocks. 2 2

Hex. [A5] Instruction code command code

SUB Rm,@WRj

Command Operation. (PC) ←(PC) + 3
(Rm) ←(Rm) - ((WRj))

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 9E(t)9(s)0H

1 0 0 1	1 1 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

SUB Rm,@DRk

Command Operation.

(PC) ←(PC) + 3

(Rm) ←(Rm) - ((DRk))

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 9E(u)B(s)0H

1	0	0	1	1	1	1	0	u	u	u	U	1	0	1	1	s	s	s	s	0	0	0	0

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	3	3
Hex.	[A5] Instruction code	command code

SUBB A,<src-byte>

Function. Borrowed Subtraction

Description. SUBB subtracts the specified variable and rounding flag bits together from the accumulator, leaving the result in the accumulator. If bit 7 needs to be borrowed, SUBB sets the rounding (borrowing) flag bit, otherwise C is cleared (if C is set before the SUBB instruction is executed, this means that the previous step in the multiple exact subtraction needs to be borrowed, so the rounding bit is subtracted from the accumulator as well as the source operand). If bit 3 needs to be borrowed, AC is set, otherwise it is cleared. The OV flag bit indicates a negative number when subtracting a positive number from a negative number, or a positive number when subtracting a negative number from a positive number. The source operand allows four addressing modes: register, direct, register indirect, or immediate number addressing. All flag bits are affected.

SUBB A,Rn

Command Operation. (PC) ←(PC) + 1

(A) ←(A) - (C) - (Rn)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 98H - 9FH

1	0	0	1	1	r	r	r

Binary Mode **Source mode**

Instruction length: 1 2

Number of clocks: 1 1

Hex. Command Code [A5] Command Code

SUBB A,Direct

Command Action. (PC) ←(PC) + 2

(A) ←(A) - (C) - (direct)

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code]

95H 1	0	0	1	0	1	0	1	direct address

	Binary mode	**Source mode**
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

SUBB A,@Ri

Command Operation. (PC) ←(PC) + 1

(A) ←(A) - (C) - ((Ri))

Impact Flags.

CY	AC	OV	N	Z

	✓	✓	✓	✓	✓

[Command Code] 96H, 97H

1	0	0	1	0	1	1	i	

	Binary mode	**Source mode**
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

SUBB A,#DATA

Command Operation. (PC) ←(PC) + 2
(A) ←(A) - (C) - #data

Impact Flags.

CY	AC	OV	N	Z
✓	✓	✓	✓	✓

[Command Code] 94H

1 0 0 1	0 1 0 0	immediate number
Binary mode	**Source mode**	

Instruction length. 2 2

Number of clocks. 1 1

Hex. command code command code

SWAP A

Function. Accumulator exchange half byte

Description. SWAP A exchanges the low and high half-bytes (four-bit field) of the accumulator (bits 3 to 0 and 7 to 4). This operation can also be considered as a four-bit loop instruction. Only the N and Z flag bits are affected.

Command Operation. (PC) ←(PC) + 1
(A3-0) ← (A7-4)
(A7-4) ← (A3-0)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] C4H

1 1 0 0	0 1 0 0
Binary Mode	**Source mode**

Instruction length: 1 1 1

Number of clocks. 1 1

Hex. Command Code Command Code

TRAP

Function. Performed as NOP

Command Operation. (PC) ←(PC) + 1

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] B9H

1 0 1 1	1 0 0 1
Binary Mode	**Source mode**

Instruction length: 2 2 1

Number of clocks: 1 1 1

Hex. [A5] Instruction Code Command Code

XCH A,<byte>

Function. Accumulator swap byte variable

Description.	CH loads the contents of the specified variable into the accumulator and writes the previous accumulator contents to the specified variable. The source and target operands can be addressed using registers, direct or register indirect addressing. Flag bits are not affected.
XCH	

XCH A,Rn

Command Operation.	(PC)	←(PC) + 1
	(A)	↔ (Rn)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] C8H - CFH

1 1 0 0	1 r r r
Binary mode	**Source mode**

Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

XCH A, Direct

Command Operation.	(PC)	←(PC) + 2
	(A)	↔ (direct)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] C5H

1 1 0 0	0 1 0 1	direct address
Binary mode	**Source mode**	

Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

XCH A,@Ri

Command Operation.	(PC)	←(PC) + 1
	(A)	↔ ((Ri))

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	-	-

[Command Code] C6H, C7H

1 1 0 0	0 1 1 i
Binary mode	**Source mode**

Instruction length.	1	2
Number	1	1
	command	[A5]

of clocks.	code Hex.	Instruction code

XCHD A,@Ri

Function. Half-byte exchange of numbers

Description. XCHD exchanges the lower half-byte of the accumulator (bits 3 through 0, usually representing hexadecimal or BCD numbers) with the half-byte of the internal memory location indirectly addressed by the specified register. The high half-byte (bits 7 through 4) of each register is not affected. Flag bits are not affected.

Command Operation.

(PC) ←(PC) + 1

(A3-0) ↔ ((Ri)3-0)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	-	-

[Command Code] D6H, D7H

1 1 0 1	0 1 1 i

Binary Mode **Source mode**

Instruction length: 1 1 2

Number of clocks. 3 3

Hex. Command Code [A5] Command Code

XRL

Function: Logical or of a variable

Description: Performs a bitwise logical different-or operation between the specified variables, storing the result in the target. The target operand can be an accumulator, a register, or a direct address. These two operands allow 12 addressing mode combinations. When the target is an accumulator or register, the source can be register, direct, register indirect, or immediate addressing; when the target is a direct address, the source can be an accumulator or immediate. Only the N and Z flag bits are affected.
Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pin.

XRL A,Rn

Command Operation. (PC) ←(PC) + 1

(A) ← (A) xor (Rn)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 68H - 6FH

0 1 1 0	1 r r r

Binary Mode **Source mode**

Instruction length: 1 1 2

Number of clocks. 1 1

Hex. Command Code [A5] Command Code

XRL A, Direct

Command Operation. (PC) ←(PC) + 2

(A) ←(A) xor (direct)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 65H

0 1 1 0	0 1 0 1	direct address

Binary mode **Source mode**

Instruction length. 2 3

Number of clocks. 1 1

Hex. command code [A5] Instruction code

XRL A,@Ri

Command Operation. (PC) ←(PC) + 1

(A) ←(A) xor ((Ri))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 66H, 67H

0	1	1	0	0	1	1	i

	Binary mode	Source mode
Instruction length.	1	2
Number of clocks.	1	1
Hex.	command code	[A5] Instruction code

XRL A,#DATA

Command Operation.	(PC)	←(PC) + 2
	(A)	←(A) xor #data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 64H

0 1 1 0	0 1 0 1	immediate number

	Binary mode	Source mode
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

0 1 1 0	0 0 1 0	direct address

XRL Direct, A

Command Operation.	(PC)	←(PC) + 2
	(direct)	← (direct) xor (A)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 62H

0 1 1 0	0 0 1 1	direct address	immediate number

	Binary mode	Source mode
Instruction length.	2	2
Number of clocks.	1	1
Hex.	command code	command code

XRL Direct,#DATA

Command operation.	(PC)	←(PC) + 3
	(direct)	←(direct) xor #data

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 63H

	Binary mode	Source mode
Instruction length.	3	3
Number of clocks.	1	1
Hex.	command code	command code

XRL Rmd,Rms

Command Action. (PC) ←(PC) + 2

(Rmd) ← (Rms) xor (Rmd)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 6CH

0	1	1	0	1	1	0	0	s	s	s	s	S	S	S	S

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

XRL WRjd,WRjs

Command Action.
(PC) ←(PC) + 2
(WRjd) ← (WRjs) xor (WRjd)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 6DH

0	1	1	0	1	1	0	1	t	t	t	t	T	T	T	T

	Binary mode	**Source mode**
Instruction length.	3	2
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

XRL Rm,#DATA

Command Operation.
(PC) ←(PC) + 3
(Rm) ←(Rm) xor #data

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 6E(s)0H

0	1	1	0	1	1	1	0	s	s	s	s	0	0	0	0	immediate number

	Binary mode	**Source mode**
Instruction length.	4	3
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

XRL WRj,#DATA16

Command Action. (PC) ←(PC) + 4

(WRj) ←(WRj) xor #data16

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 6E(t)4H

														Immediate High Byte	Immediate Low Byte
0	1	1	0	1	1	0	1	t	t	t	t	0	1	0	0		

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	1	1
Hex.	[A5] Instruction code	command code

XRL Rm,Dir8

Command Operation. (PC) ←(PC) + 3

(Rm) ← (Rm) xor (dir8)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 6E(s)1H

0 1 1 0	1 1 1 0	s s s s	0 0 0 1	direct address

Binary mode	**Source mode**
Instruction length. 4	3
Number of clocks. 1	1
Hex. [A5] Instruction code	command code

XRL WRj,Dir8

Command Action. (PC) ←(PC) + 3

(WRj) ← (WRj) xor (dir8)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 6E(t)5H

0 1 1 0	1 1 0 1	t t t t	0 1 0 1	direct address

Binary mode	**Source mode**
Instruction length. 4	3
Number of clocks. 1(2 for SFR)	1(2 for SFR)
Hex. [A5] Instruction code	command code

XRL Rm, Dir16

Command Operation. (PC) ←(PC) + 4

(Rm) ←(Rm) xor (dir16)

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 6E(s)3H

0 1 1 0	1 1 1 0	s s s s	0 0 1 1	Address High Byte	Address Low Byte

Binary mode	**Source mode**
Instruction length. 5	4
Number of clocks. 2	2
Hex. [A5] Instruction	command code

code

XRL WRj,Dir16

Command Action.	(PC)	←(PC) + 4
	(WRj)	←(WRj) xor (dir16)

Impact Flags.	CY	AC	OV	N	Z
	-	-	-	✓	✓

[Command Code] 6E(t)7H

0 1 1 0	1 1 0 1	t t t t	0 1 1 1	Address High Byte	Address Low Byte

	Binary mode	**Source mode**
Instruction length.	5	4
Number of clocks.	2	2
Hex.	[A5] Instruction code	command code

XRL Rm,@WRj

Command Operation.
(PC) ←(PC) + 3
(Rm) ←(Rm) xor ((WRj))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 6E(t)9(s)0H

0 1 1 0	1 1 1 0	t t t t	1 0 0 1	s s s s	0 0 0 0

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks. 1 1

Hex. [A5] Instruction code command code

XRL Rm,@DRk

Command Action.
(PC) ←(PC) + 3
(Rm) ←(Rm) xor ((DRk))

Impact Flags.

CY	AC	OV	N	Z
-	-	-	✓	✓

[Command Code] 6E(u)B(s)0H

0 1 1 0	1 1 1 0	u u u u	1 0 1 1	s s s s	0 0 0 0

Binary mode **Source mode**

Instruction length. 4 3

Number of clocks. 3 3

Hex. [A5] Instruction code command code

Appendix B Foundations of Logic Algebra

--Users without microcomputer principles should start with this chapter

This chapter focuses on (i) the basics of arithmetic operations in digital devices - number system and coding; (ii) some common logical operations in digital circuits and their graphical symbols. They are the basis for learning the course of microcontroller. For users and students who have no basic knowledge of microcomputer principles, please start from this chapter.

B.1 number system and coding

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

There are many different number systems, and the commonly used ones are: binary, decimal and hexadecimal.

The progressive counting system involves dividing a number into different digits, adding up the digits one by one, and after a certain number of digits have been added, starting from zero and moving up to the next higher digit at the same time. There are three elements to the progressive counting system: the digital symbol, the pattern of progression, and the base of the count. The following table is a general introduction to each of the commonly used counting systems.

Commonly used number systems	Notation	digital symbol	system of writing numbers a base, such as decimal or binary (math)	counting base
binary system (math)	B	0, 1	lit. every second person enters a group	2
decimal system	D	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	lit. if every ten goes into one, one goes into the other	10
hexadecimal	H	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	lit. every 16th goes	16

			into 1st place	

We generally use decimal system for counting in our daily life. Binary is used in computers because it is simple, easy to implement and reliable, provides a favourable way for logic design, saves equipment and other advantages. In order to distinguish it from other systems, binary numbers are usually written in the lower right of the number of base 2, or add the back of the B said. Each bit of a binary number has only two possible digits, 0 and 1, so the counting base is 2. The addition and multiplication operations for binary numbers are as follows:

0 + 0 = 00 + 1 = 1 + 0 = 11 + 1 = 10

0 × 0 = 00 × 1 = 1 × 0 = 01 × 1 = 1

Since binary numbers are too long to be remembered, hexadecimal is often used as an abbreviation for binary in order to make it easier to describe. Hexadecimal is usually expressed with the trailing symbol H or the subscript 16 to show the difference.

B.1.1 digital-to-numeric conversion

Now we will introduce the conversion

between these common number systems.

I: Binary - Decimal Conversion

Method: Expand the binary number by weight (as in the following equation), and then add the values of each item by decimal number to get the corresponding equivalent decimal number.

For example, if N = (1101.101)B, what is the decimal number corresponding to N?

Expand by weights $N=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}=8+4+0+1+0.5+0+0.125=(13.625)D$

II: Decimal - Binary Conversion

METHODOLOGY: This is done in two parts i.e. whole number part and decimal part.

① Integer partial conversion (base division):

★ Divide the number we want to convert by the base of the binary (the base of the binary is 2) and use the remainder as the lowest bit of the binary;

★ Divide the previous quotient by the binary base (i.e., 2) and use the remainder as the second lowest binary digit;

★ Continue until the final quotient is zero, at which point the remainder is the highest binary digit.

② Fractional part conversion (base multiplication):

★ Multiply the decimal portion of the number to be converted by the base of binary (base 2 in binary), and use the resulting integer portion as the highest bit of the decimal portion of the binary;

★ Multiply the decimal portion from the previous step by the base of the binary (i.e., 2), and use the integer portion as the second highest digit of the decimal portion of the binary;

★ Continue the previous step until the fractional part becomes zero. Or it's okay to reach a predetermined requirement.

For example, to convert $(213.8125)_{10}$ to a binary number you can do the following: first convert the integer part:

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

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

Reincarnate the fractional part:

0.8125	
× 2	
1.6250	Integer part = 1 = k_{-1}
0.6250	
× 2	
1.2500	Integer part = 1 = k_{-2}
0.2500	
× 2	
0.5000	Integer part = 0 = k_{-3}
0.5000	
× 2	
1.0000	Integer part = 1 = k_{-4}

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

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

III: Binary-hexadecimal conversion

Methods: binary and hexadecimal to meet the relationship between 24, so the binary to be converted from low to high every 4-bit group, high enough to add "0" in front of the effective bit, and then each group of binary numbers can be converted to hexadecimal.

For example, convert (010111011110.11010010)B to a hexadecimal number:

(0101 1101 1110 . 1101 0010)B
↓ ↓ ↓ ↓ ↓
= (5 D E B 2)

Thus, (010111011110.11010010)B = (5DE.B2)H

IV: Hexadecimal - Binary Conversion

Methods: hexadecimal to binary conversion, the above binary conversion of hexadecimal process in reverse, that is, the conversion of hexadecimal only each bit with the equivalent of 4-bit binary instead of the line.

Example: Convert (C1B.C6)H to a binary number:

(C 1 B . C 6) H
↓ ↓ ↓ ↓ ↓
= (1100 0001 1011 1100 0110)B

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

V: Hexadecimal - Decimal Conversion

Method: Expand the hexadecimal number according to the weight (as in the following formula), and then add the values of each item according to the decimal number, you will get the corresponding equivalent decimal number.

For example, if N = (2A.7F)H, what is the decimal number corresponding to N?

Expand by weights $N = 2\times16^{1} +10\times16^{0} +7\times16^{-1} +15\times16^{-2} =32+10+0.4375+0.05859375 =(42.49609375)D$

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

VI: Decimal - Hexadecimal Conversion

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

B.1.2 Primary, inverse and complementary codes

In life, there are positive and negative numbers, in the computer how to express the positive and negative sign of the number?

In life, when the number is expressed is generally a positive number in front of a "+", negative numbers in front of a "-", but the computer does not recognise these, usually in front of the binary number to add a sign bit. Sign bit for "0" means "+", sign bit for "1" means "-". This form of binary number is called the original code. If the original code is positive, the inverse and complement of the original code are the same as the original code. If the original code is negative, the original code (except for the sign bit) by bit inverse, the new binary number is called the original code of the inverse code, the inverse code plus 1 for its complement.

The three forms, original, inverse and complementary, are summarised in the table below:

	truth value	source code	reverse code	binary code with 0 and 1 interchanged
positive number	+N	0N	0N	0N
negative number	-N	1N	$(2^n -1)+N$	$2^n +N$

Example 1: Find the +18 and -18 octets in their original, inverse, and complementary forms.

truth value	source code	reverse code	binary code with 0 and 1 interchanged
+18	00010010	00010010	00010010
-18	10010010	11101101	11101110

B.1.3 common coding

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

Decimal Encoding

A decimal number expressed in binary code is called decimal code. It has the form of binary, but also has the characteristics of decimal it can be used as a kind of contact between people and the digital system. There are many kinds of decimal code, the most commonly used one is BCD code, also known as 8421 code.

Below we list several common decimal codes in a table.

Type of code	8421 yards (BCD code)	Remaining 3	2421 yards	5211 yards	7321 yards

decimal number		yards			
0	0000	0011	0000	0000	0000
1	0001	0100	0001	0001	0001
2	0010	0101	0010	0100	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0111	0101
5	0101	1000	1011	1000	0110
6	0110	1001	1100	1001	0111
7	0111	1010	1101	1100	1000
8	1000	1011	1110	1101	1001
9	1001	1100	1111	1111	1010
authority	8421		2421	5211	7321

Decimal codes are divided into entitled and un-entitled codes. Powerful code means that each decimal digit is represented by a four-bit binary code, and each bit of the binary code has a fixed weight. Unprivileged code means that each bit of the binary code has no fixed weight. In the above table, 8421 code (i.e. BCD code), 2421 code, 5211 code and 7321 code are all entitled codes, while the remaining 3 codes are un-entitled codes.

II: Parity code

In the process of data access, arithmetic and transmission, it is inevitable that errors will occur, the "1" is wrong as "0" or "0" is wrong as "1". ". The parity check code is a code that checks for such errors. It is divided into two parts; the information bit and the parity bit. An odd number of "1s" is called an odd parity check, and an even number of "1s" is called an even parity check.

B.2 Several common logical operations and their graphical symbols

Commonly used operations in logic algebra are: and (AND), or (OR), not (NOT), and not (NAND), or not (NOR), and or not (NON)
(AND-NOR), different or (EXCLUSIVE OR), same or (EXCLUSIVE NOR), and so on. Where with (AND), or (OR), not (NOT)
The three most basic operations when it comes to arithmetic.

I: with operations and with gates

The operation of and: An event occurs when all the conditions that determine the outcome of the event are present at the same time. The logical variables A and B can be written as follows: Y=A-B

truth table		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

And Gate: A unitary circuit that performs and logical operation.
Graphical symbol for an and gate:

II: Or operations and or gates

Or operation: An event occurs if any one of the conditions determining the outcome of the event is satisfied. Logic variables A and B can be written in the form of an or operation as follows: Y = A + B

truth table		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Or Gate: A unitary circuit that performs or logic operations. Or gate graphic symbol:

III: Non-operational and non-gates

Non-operations: events do not occur when conditions are present; events occur when conditions are not present. The logic variable A can be written as Y=A′ for non-operational purposes.

truth table	
A	Y
0	1
1	0

Non-gate: a unitary circuit that performs non-logical operations.

Non-gated graphical symbols:

IV: With and without arithmetic and with and without graphical symbols

With and without operation: first carry out the operation with and then inverse the result, the final result is the result of the operation with and without. Logic variables A and B can be written as Y=(A-B)' when performing the operation with and without.

truth table		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

with non-graphical symbols: 

V: Conjugate operations and conjugate symbols

Or-Non operation: first perform or operation, and then inverse the result, the final result is the result of the or-non operation. Logic variables A and B can be written as Y=(A+B)' for non-conjugate operations.

truth table		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

or non-graphical symbols: 

VI: Conjugate or non-conjugate operations and conjugate or non-conjugate graphical symbols

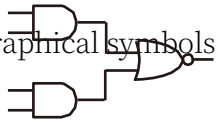
With or Without: There are 4 logical variables A, B, C, D. Assuming that A and B are a group and C and D are a group, and that the relationship between A and B and between C and D is and, as long as any of the groups A, B, or C, and D is 1 at the same time, the output Y is 0. The output Y is 1 only if each of these inputs is not 1 at the same time.

The logical variables A and B can be written as Y = (A-B + C-D)' when performing non-orthogonal operations

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

1	1	1	1	0

with or without graphical symbols:



VII: Different-or Operations and Different-or Graphical Symbols

Dissimilarity operation: when A and B are different, the output Y is 1; when A and B are the same, the output Y is 0. The dissimilarity operation between logical variables A and B can be written as follows: $Y = A \oplus B = (A\text{-}B') + (A'\text{-}B)$

truth table		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Different or graphic symbols: 

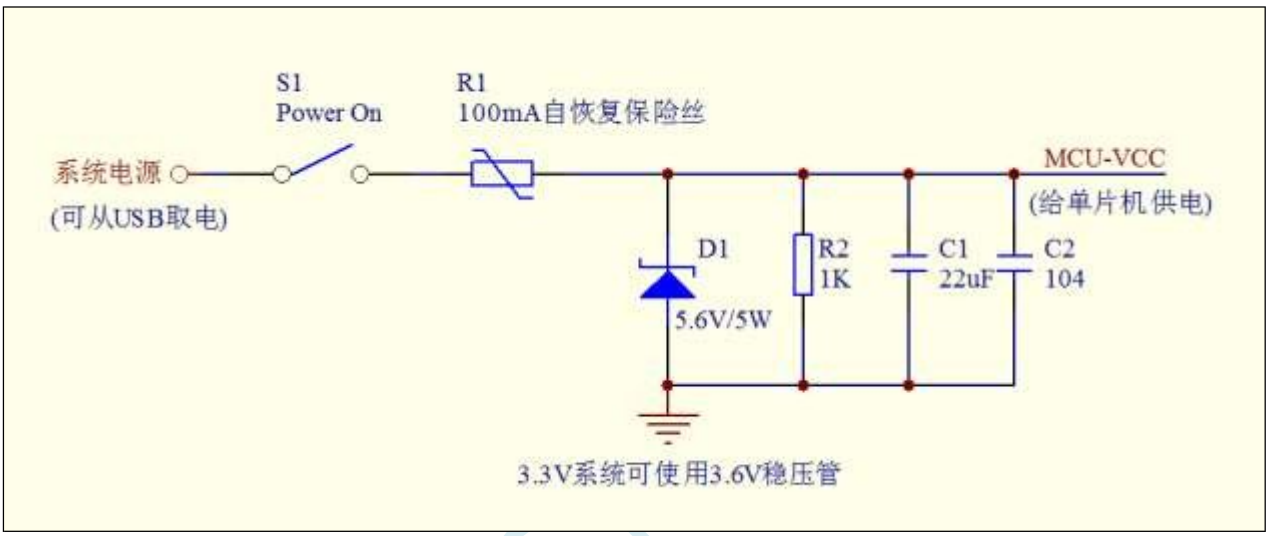
VIII: Same-or operations and same-or graphical symbols

Same-or operation: when A and B are different, the output Y is 0; when A and B are the same, the output Y is 1. The same-or operation for logical variables A and B can be written as follows: $Y = A \odot B = (A\text{-}B)+(A'\text{-}B')$

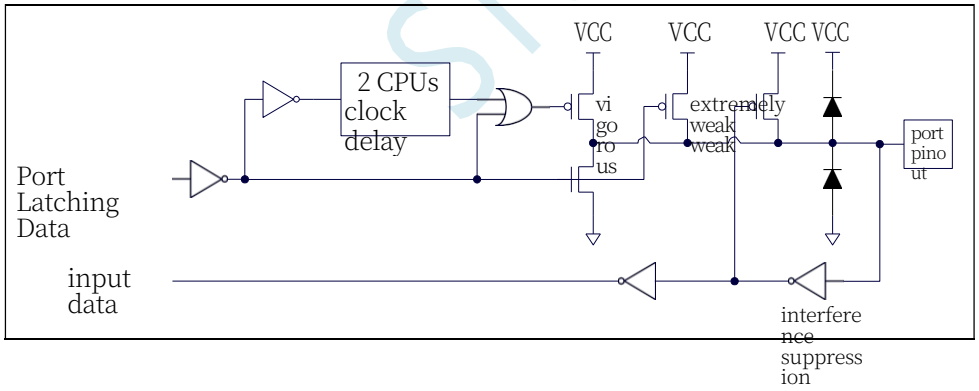
truth table		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Same or graphic symbols: 

Appendix C Suggested Power Management Circuits for a Microcontroller Minimum System



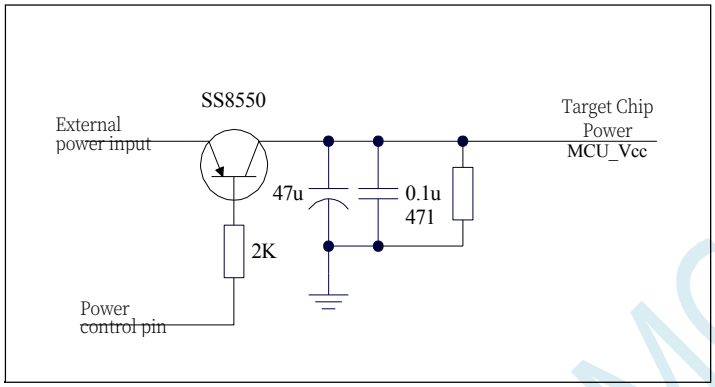
Internal structure of I/O



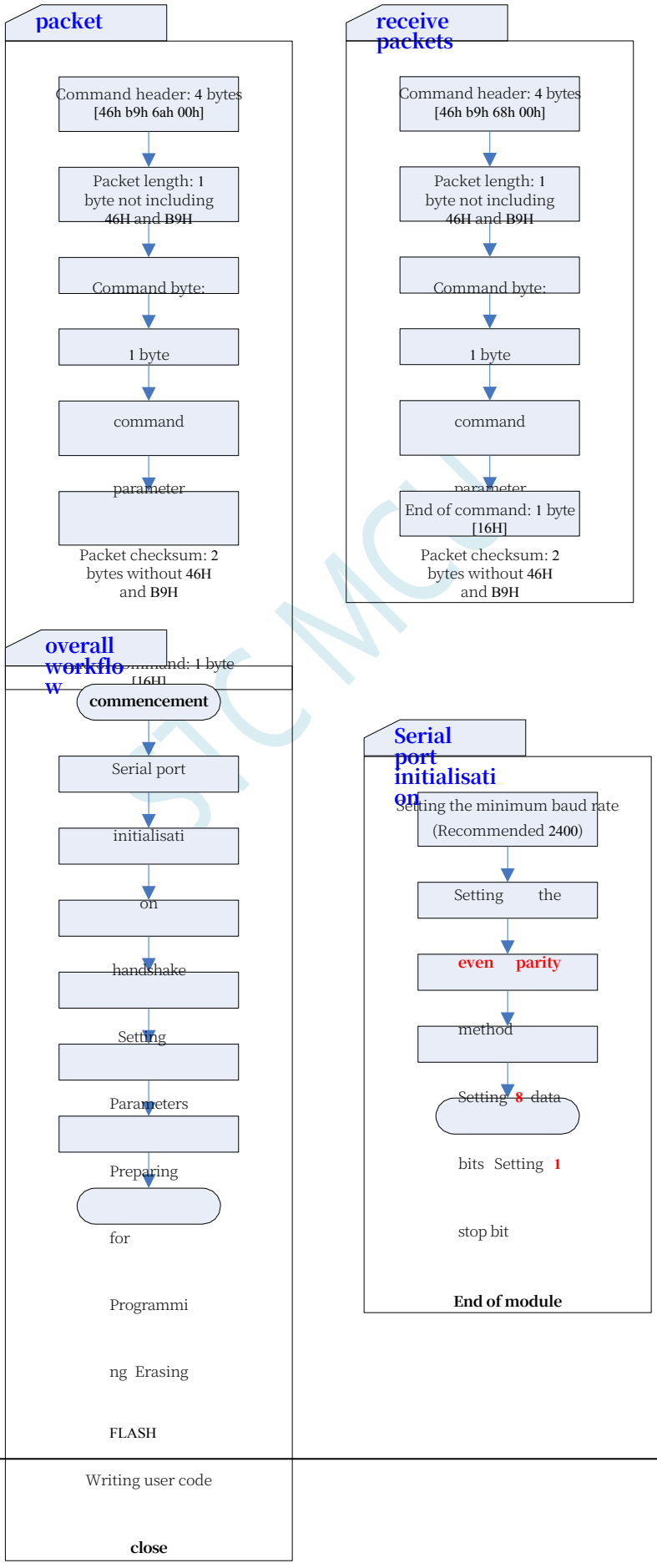
Appendix D ISP Download Example Programs for STC32G Series Microcontrollers Using Third-Party MCUs

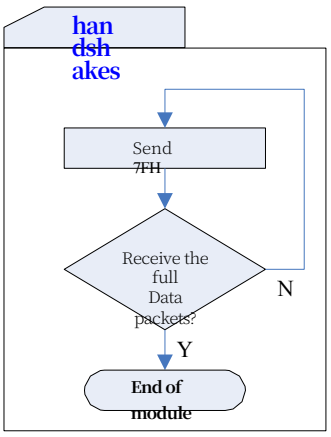
D.1 Power Control Reference Circuit

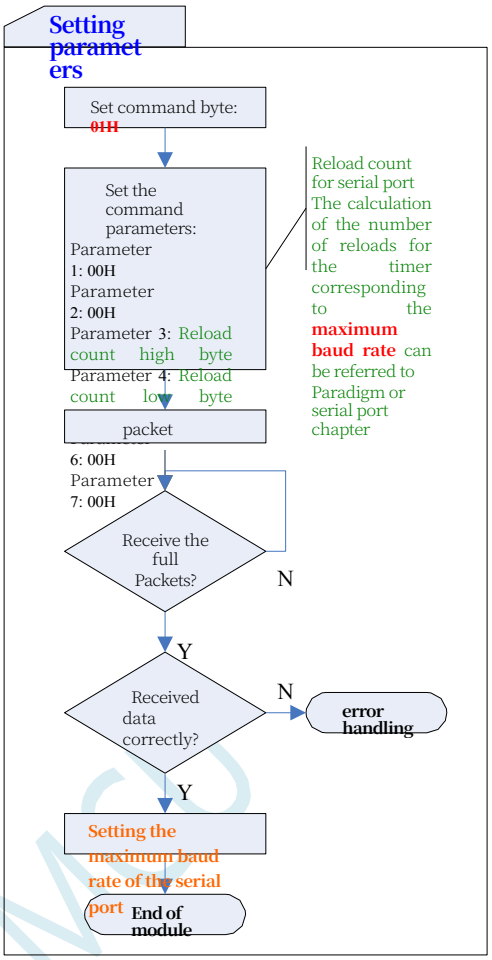
The STC chip ISP download requires a hardware reset of the target to enter ISP download mode. When using a third party MCU to perform an ISP download to the STC chip, the following power control circuit is recommended for this purpose.

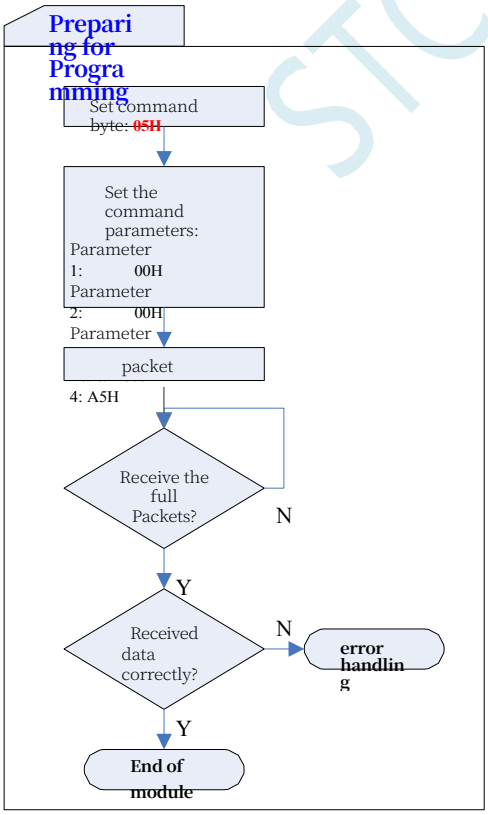


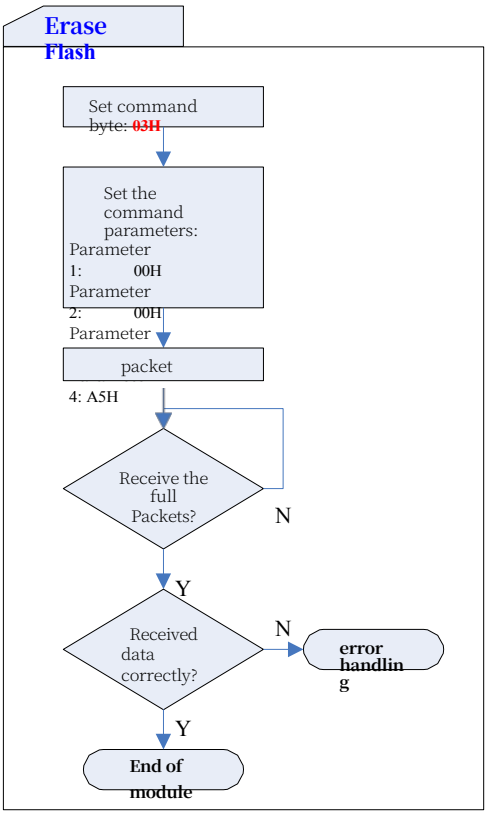
D.2 Communication Protocol Flowchart

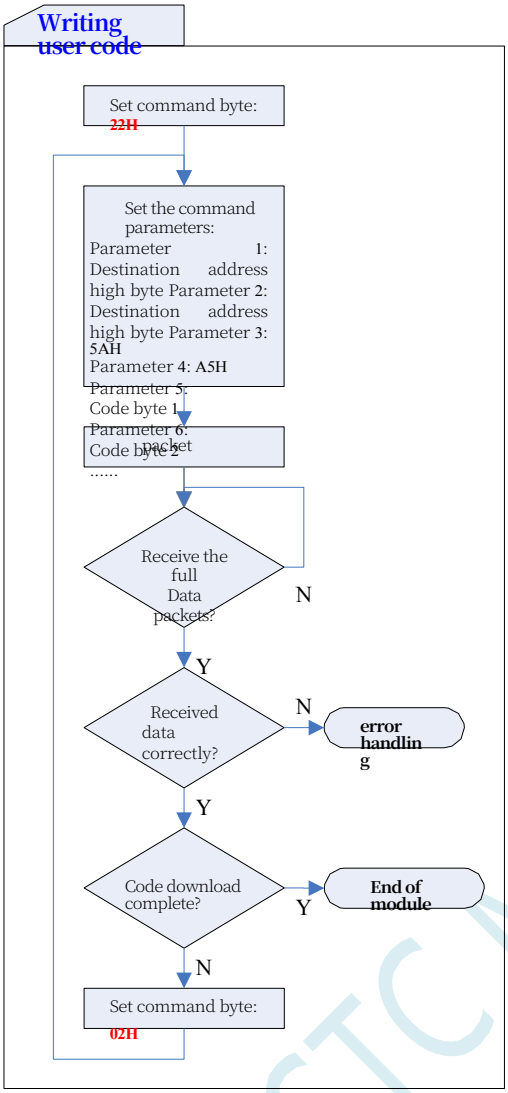



handshakes
Send 7FH
Receive the full Data packets?
N
Y
End of module


Setting parameters
Set command byte: 01H
Set the command parameters:
Parameter 1: 00H
Parameter 2: 00H
Parameter 3: Reload count high byte
Parameter 4: Reload count low byte
Reload count for serial port The calculation of the number of reloads for the timer corresponding to the maximum baud rate can be referred to Paradigm or serial port chapter
packet
6: 00H
Parameter 7: 00H
Receive the full Packets?
N
Y
Received data correctly?
N
error handling
Y
Setting the maximum baud rate of the serial port
End of module


Preparing for Programming
Set command byte: 05H
Set the command parameters:
Parameter 1: 00H
Parameter 2: 00H
Parameter
packet
4: A5H
Receive the full Packets?
N
Y
Received data correctly?
N
error handling
Y
End of module


Erase Flash
Set command byte: 03H
Set the command parameters:
Parameter 1: 00H
Parameter 2: 00H
Parameter
packet
4: A5H
Receive the full Packets?
N
Y
Received data correctly?
N
error handling
Y
End of module


Writing user code
Set command byte: 22H
Set the command parameters: Parameter 1: Destination address high byte Parameter 2: Destination address high byte Parameter 3: 5AH Parameter 4: A5H Parameter 5: Code byte 1 Parameter 6: Code byte 2 ……
Receive the full Data packets?
N
Y
Received data correctly?
N
error handling
Y
Code download complete?
Y
End of module
N
Set command byte: 02H

D.3 Reference code (C language)

C code

```
//Note: When you use this code to download STC8H series microcontroller, you must execute the Download code.
//Power up the target chip, otherwise the target chip will not be able to download correctly.

#include "reg51.h"

typedef    bit             BOOL.
typedef    unsigned char   BYTE;
typedef    unsigned short  WORD.

//Macro, Constant Definitions
#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          //Master Chip Operating Frequency
#define    BR(n)           (65536 - (FOSC/(n)+2) /4)   //Master Chip Serial Port Baud Rate Calculation Formula
                                              //The // plus 2 operation is to allow the Keil compiler to
                                              //Automatic implementation of rounding operations
#define    T1MS            (65536 - FOSC/1000)         //Master 1ms Timer Initial Value

#define    FUSER           24000000L          //STC32G Series Target Chip Operating Frequency
#define    RL(n)           (65536 - (fuser/(n)+2) /4)  //STC32G Series Target Chip Serial Baud Rate Calculation Formulas
                                              //The // plus 2 operation is to allow the Keil compiler to
                                              //Automatic implementation of rounding operations

sfr        auxr  =  0x8e;
sfr        p3m1  =  0xb1;
sfr        p3m0 = 0xb2.

//Variable definition
BOOL f1ms.                                    //1ms flag bit
BOOL UartBusy.                                //Serial port send busy flag bit
BOOL UartReceived.                            //Serial port data received flag bit
BYTE UartRecvStep.                            //Serial port data reception control
BYTE TimeOut.                                 //Serial communication timeout counter
BYTE xdata TxBuffer[256].                     //Serial port data send buffer
BYTE xdata RxBuffer[256].                     //Serial port data receive buffer
char code DEMO[256];                          //Demo code data

//Function declaration
```

```
Technical Manual
void InitUart(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 Successful
        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
    {
        //
        Downloa
        d failure
        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 procedure *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 initialisation
void Initial(void)
{
    UartBusy = FALSE;

    SCON = 0xd0;                    //Serial data mode must be 8-bit data + 1-bit parity 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 port data sending procedure
BYTE UartSend(BYTE dat)
{
    while (UartBusy);

    UartBusy = TRUE;
    ACC = dat;
    TB8 = P.
    SBUF = ACC.

    return dat.
}

//Serial communication initialisation
void CommInit(void)
{
    UartRecvStep = 0;
    TimeOut = 20;
    UartReceived =
    FALSE;
```

```
}

//Send serial communication packet 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();
}

// ISP downloads for STC32G series chips.
BOOL Download(BYTE *pdat, long size)
{
    BYTE offset;
    BYTE cnt.
    DWORD addr.                    //Over 64K code space, address should be defined as 4 bytes.

    //Handsh
    ake
    CommInit();
    while (1)
    {
        if (UartRecvStep == 0)
        {
            UartSend(0x7f);
            DelayXms(10).
        }
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x50) break;
            return FALSE;
        }
    }

    //set parameter (set the highest baud
    rate to be used from the chip)
    TxBuffer[0] = 0x01;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    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.
        }
}

//Preparation
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)
{
        if (addr < 0x10000)                    //The target address information of the program code should be
        obtained from the original HEX file.
        {
                                {
        }
        else
```

```
TxBuffer[0] |=
0x10;                   //Target programming address is FE:0000~FE:FFFF

TxBuffer[0] &=
~0x10;                  //Target programming address is FF:0000~FF:FFFF
```

```
            }
            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;
        }

        //Downloa
        d
        complete
        return
        TRUE;
        return
        TRUE;
        return
        TRUE;
        return
        TRUE;
        return
        TRUE;
        return
        TRUE;
        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.
};
```

Appendix ESTC-ISP Download Software Advanced Application

E.1 Publishing project procedures

Publishing the project programme function is mainly to package the user's programme code and related option settings into a super-simple user-interface executable file that can be downloaded and programmed directly to the target chip.

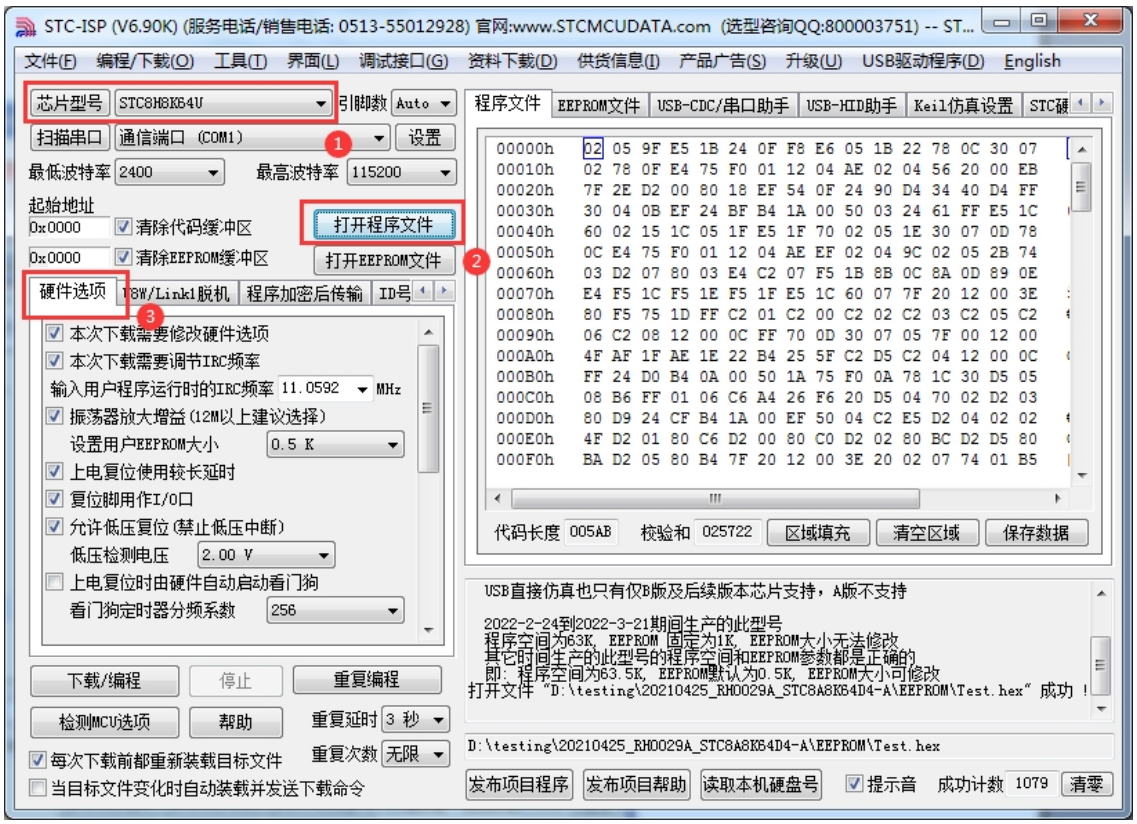
Regarding the interface, the user can customize it by himself (the user can modify the title, button name and help information of the released project), and at the same time, the user can also specify the hard disc number of the target computer and the ID number of the target chip. After specifying the hard disc number of the target computer, the user can control the release of the application program to be run only in the specified computer (to prevent the program from being easily stolen from the computer by the programmers, such as sending it away through the network, such as baking it away through the USB stick, to prevent it from being stolen from your computer. Send away, such as through the U disk baked away, can not be prevented, of course, steal your computer that would not be able to do that, so STC's offline download tool than the computer burning security, can limit the number of chips that can be burned, so that the front desk clerk Miss burn, so that the boss's wife to burn can be), copied to other computers, the application can not be run. Similarly, when the ID number of the target chip is specified, then the user code can only be downloaded to the corresponding ID number of the target chip (for a device to sell tens of millions of dollars of products are particularly useful - tanks, can be sent to the customer's own upgrades, do not need to risk their lives to run to the war in Iraq to upgrade the software), for the ID number is not consistent with the other chip, can not be downloaded for programming.

The detailed procedure for posting a project procedure is as follows:

1, first select the target chip model

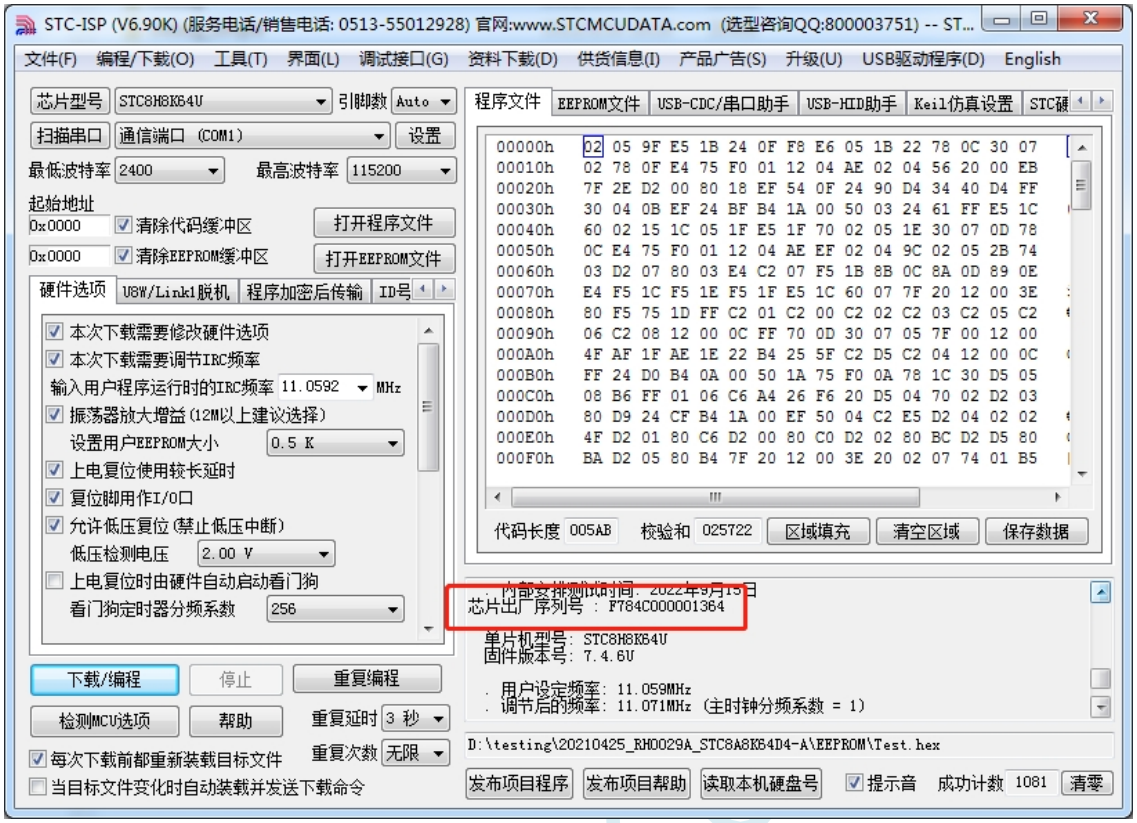
2, open the programme code file

3. Set the appropriate hardware options

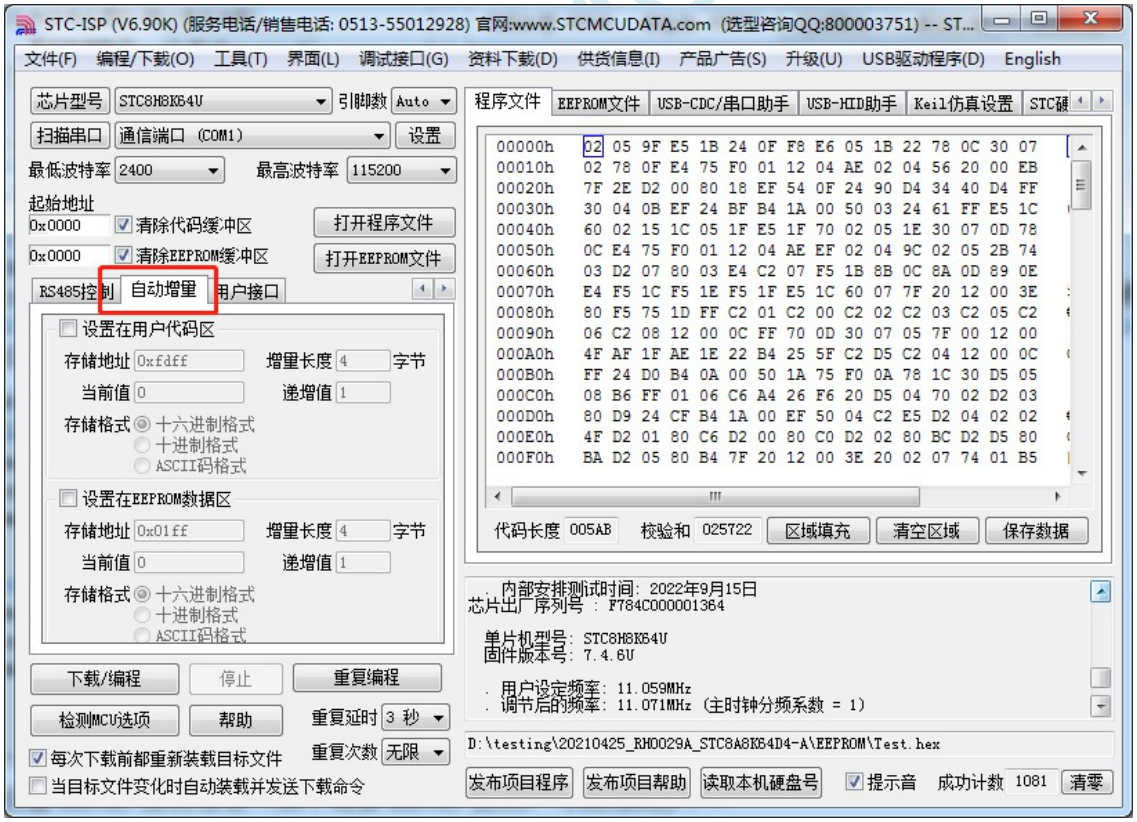


4, try to burn the chip, and note down the ID number of the target chip, as shown in the figure below, the chip's ID number is "F784C000001364".

(This step can be skipped if there is no need to verify the ID number of the target chip)

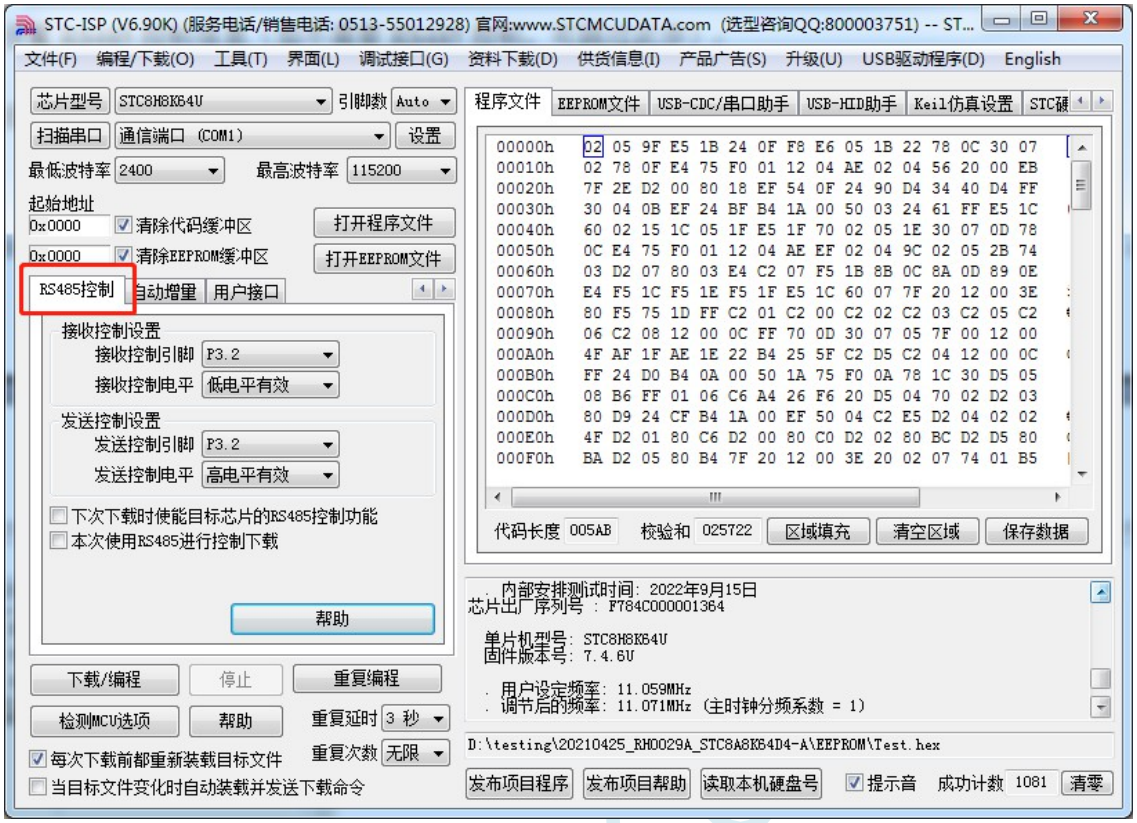


5、Set up automatic increment (if you do not need automatic increment, you can skip this

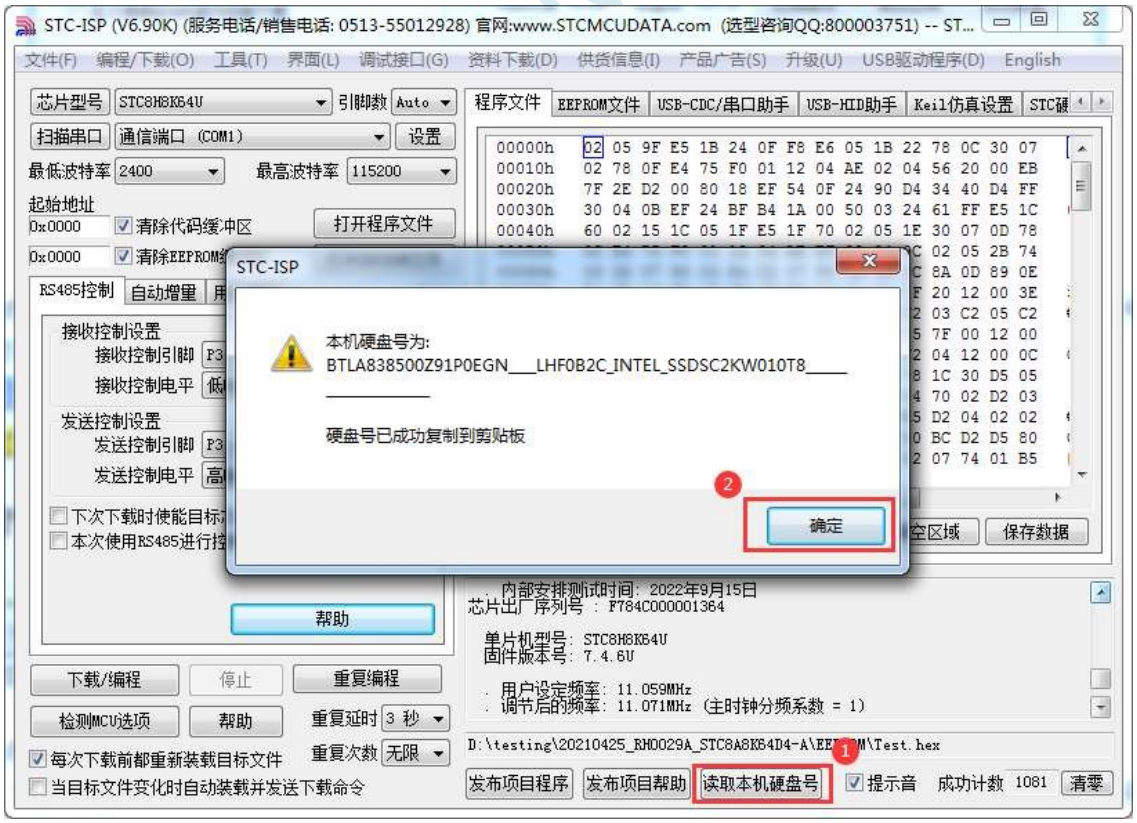


step)

6、Set RS485 control information (if you do not need RS485 control, you can skip this step)



7, click on the interface of the "read the local hard disc number" button, and note down the target computer's hard disc number (such as the target computer does not need to verify the hard disc number, you can skip this step)



8. Click the "Publish Project Procedures" button to enter the publishing application settings interface.

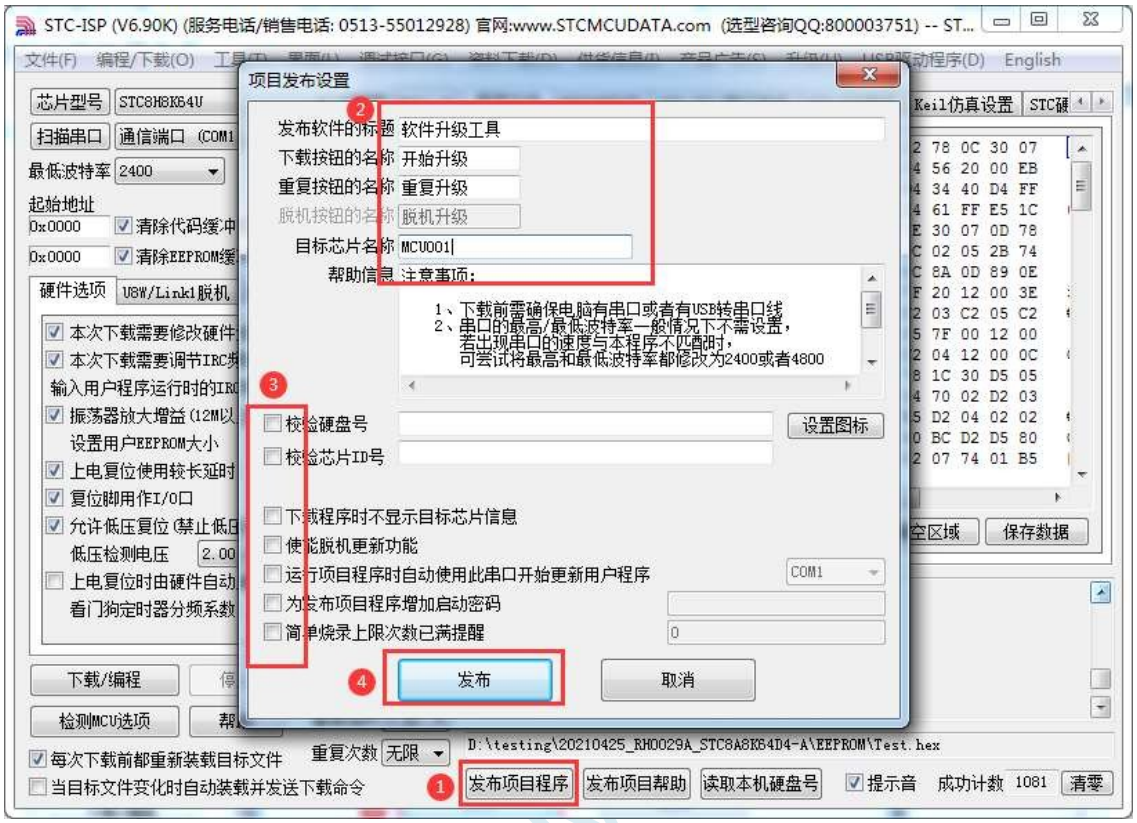
9, according to their respective needs, modify the title of the release software, the name of

the download button, the name of the repeat download button, the name of the automatic increment and help information

10, if you need to verify the target computer's hard disk number, you need to check the "check the hard disk number", and in the back of the text box, enter the previous

The hard disc number of the target computer

11, if you need to verify the ID number of the target chip, you need to check the "verify chip ID number", and enter the ID number of the target chip in the text box behind.



12, and finally click the release button, the project will be released to save the procedure, you can get the corresponding executable file. Published project programme to play the interface as follows

E.2 Program encrypted for transmission (to prevent the serial port from analysing the program when burning)

At present, all common serial port download and burn programming are using explicit communication (when the computer and the target chip communicate, or offline download board and the target chip communication), the problem: if the burner by analysing the data of serial port communication when downloading and burning programming, the masters are able to draw 2 wires out of the serial port when burning, and analyse the data of serial port communication to analyse the actual user program code. Of course, using STC's offline download board burn program is always better than using a computer to burn the program (to prevent burning personnel will easily steal the program from the computer, such as through the network to send away, such as through the USB flash drive baked away, can not be prevented, of course, stealing your computer that would not be able to do that, so the STC's offline download tool is safer than the computer to burn the tool to the receptionist clerk to burn the lady to burn the boss's wife to burn it all can be). Even the world's first STC offline download tool, to prevent the genius of the lawless elements in the process of offline download tool burning by analysing the serial port communication data to analyse the actual user program code, there is no way to meet the requirements, which requires the use of the latest STC microcontroller to provide program encryption after the transmission function.

Special Note: For the STC32G12K128 model, the FLASH size is 128K, when you need to use the function of transferring the programme after encrypting, you need to refer to the following two methods **(just choose one of them, and it is recommended to choose the method 1 which is easy to operate)**:

Method 1: Set the EEPROM to 128K, and then load the classified file directly into the programme file area of the STC-ISP download software, then you can carry out normal encrypted transmission.

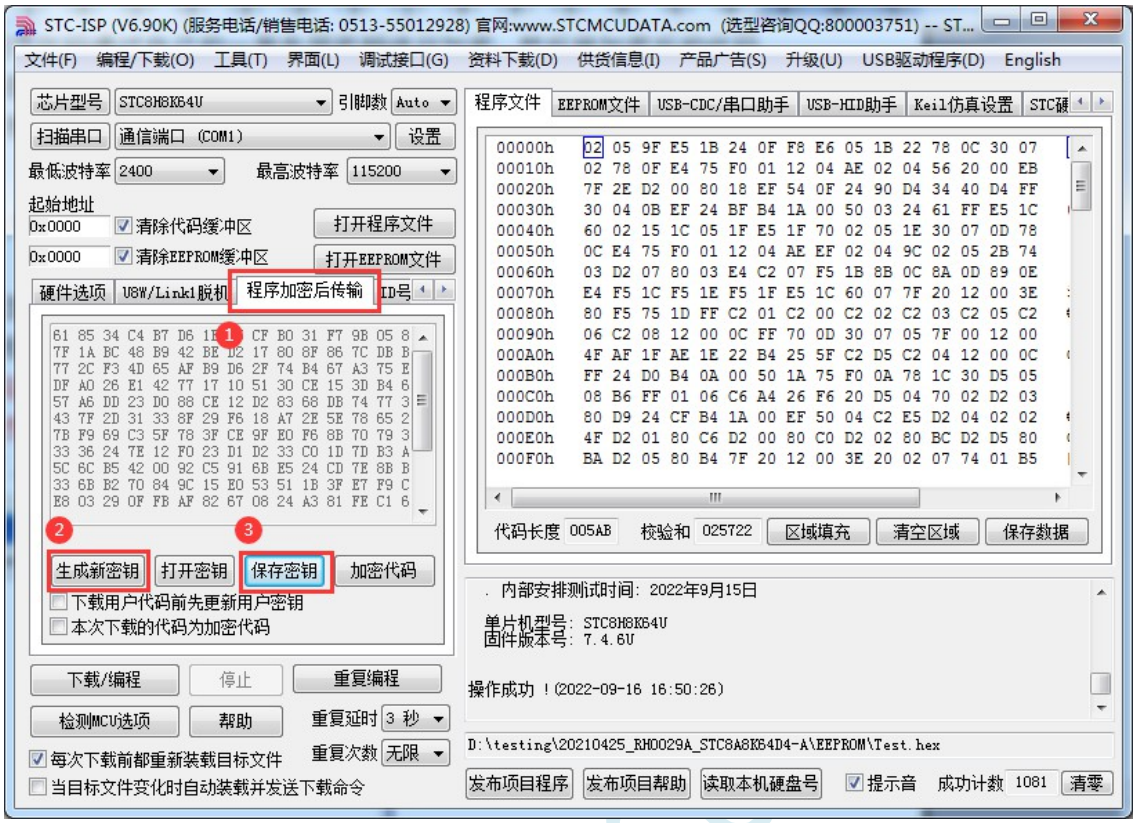
Method 2: Split the encrypted file [file 0.bin] into two files [file 1.bin, file 2.bin] and load them into the STC-ISP download software separately. For example, if the size of the EEPROM at the time of downloading is nK bytes, and the encrypted file [file 0.bin] is mK bytes, then you need to split the encrypted file [file 0.bin] into [file 1.bin] which is nK bytes, and [file 2.bin] which is (m-n)K bytes, where file 1.bin is the EEPROM file which needs to be loaded into EEPROM area, and file 2.bin is the code area for the file 2.bin, and file 2.bin is the code area for the file 1.bin, and file 2.bin is the code area for the file 2.bin. file 1.bin is an EEPROM file to be loaded into the EEPROM area, and file 2.bin is a code file to be loaded into the program file area. Load the file 1.bin and 2.bin are loaded into the EEPROM area and the programme file area of the STC-ISP download software, respectively, and normal encrypted transfers can be performed.

Program encryption after transmission and download is the user first program code through their own set of special keys for encryption, and then the encrypted code and then download through the serial port, at this time the download and transmission of encrypted files, analysed through the serial port is encrypted messy code, such as not by sending someone to sneak into your company to steal the encryption key inside your computer, it is not of any value, it will be able to play a role in the prevention of the program in the burning burner when it is burning the personnel! The purpose of analysing the code by monitoring the serial port.

The following steps are required to use the Transfer after Program Encryption function:

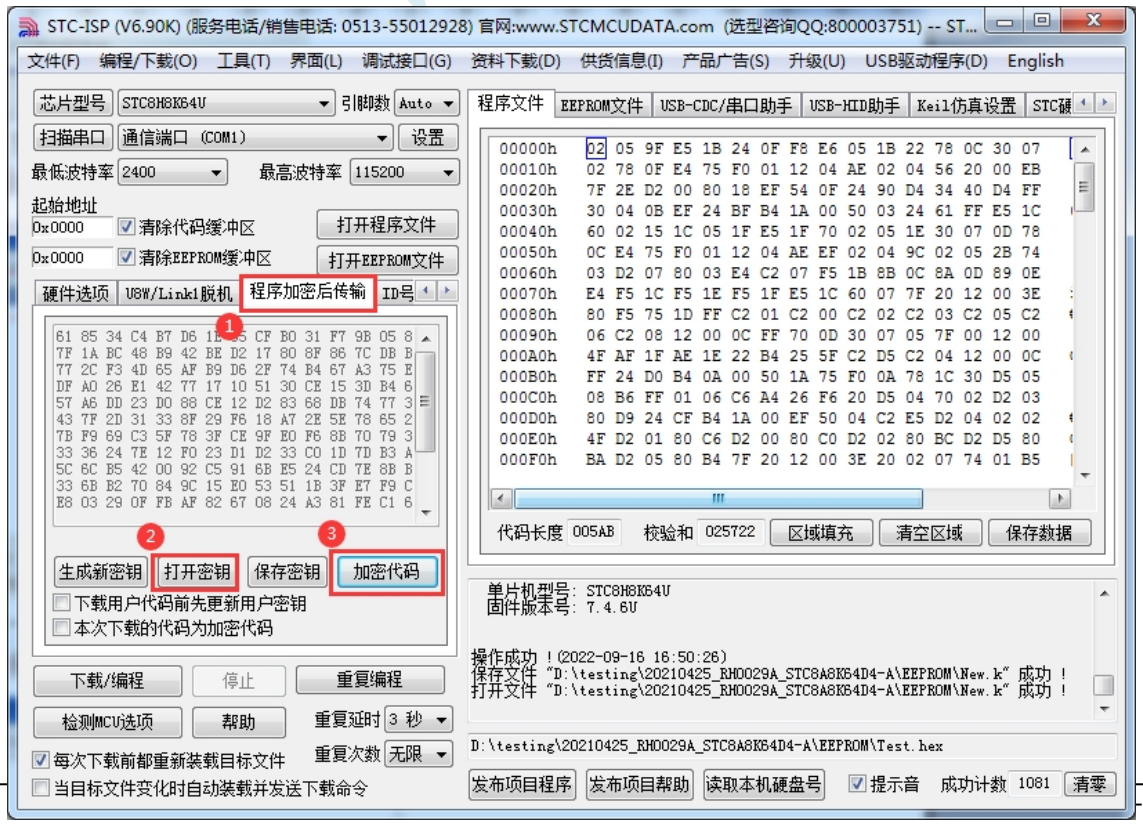
1. Generate and save a new key

As shown in the figure below, go to the "Transmission after encryption" page, click "Generate new key" button to display the newly generated 256-byte key in the buffer. Then click "Save Key" button, you can save the new key as a key file with ".K" extension (**Note: this key file must be saved, all future code files need to be encrypted with this key, and the key generation is non-repeating). K" extension (note: this key file must be saved, as it will be used to encrypt all future code releases, and the key is non-repeating, i.e., it is impossible to generate two identical keys at any one time, so it will be impossible to regain the key file if it is lost**). For example, we save the key as "New.k".



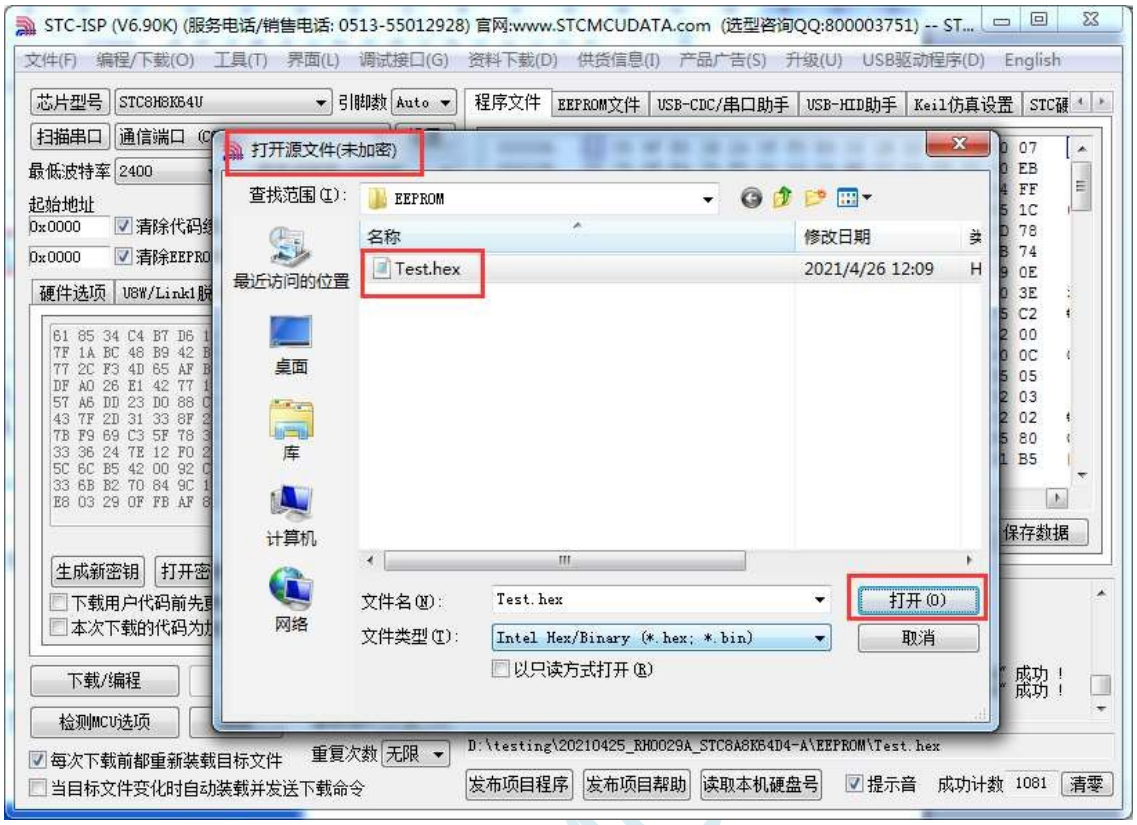
2. Encryption of code files

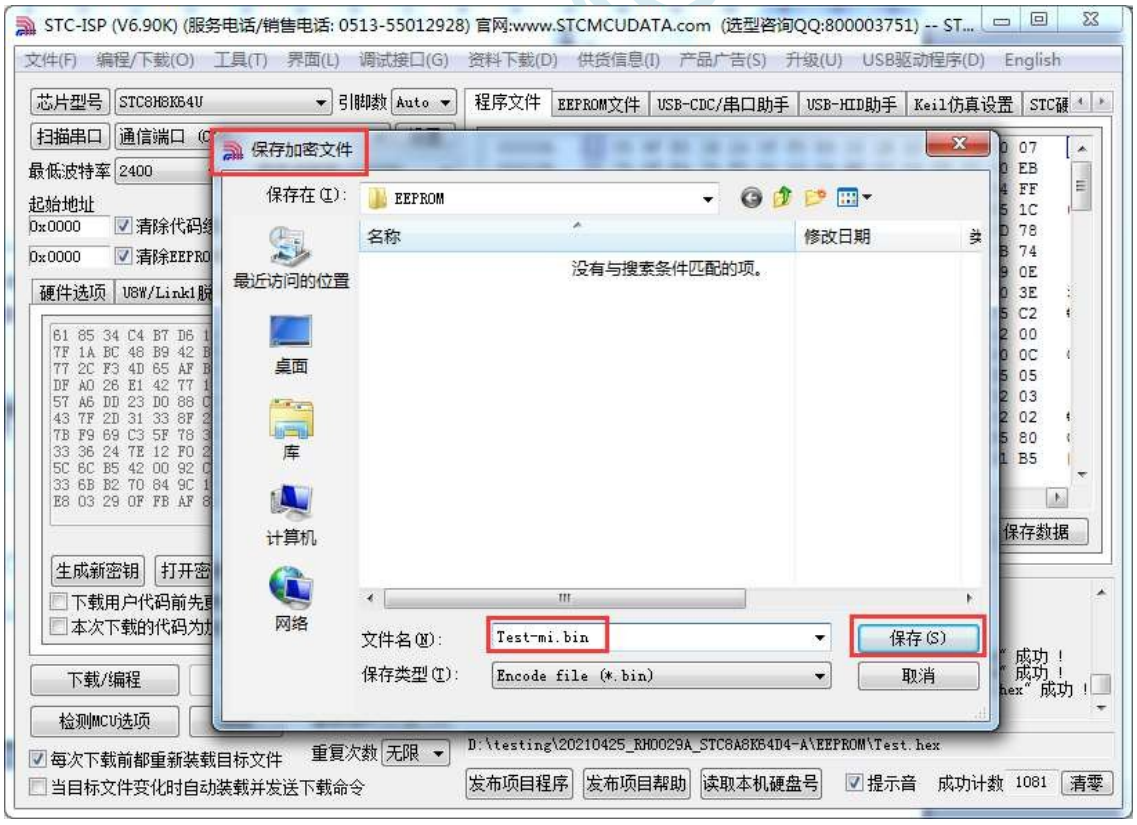
Before encrypting a file, you need to open our own key. If our key is already stored in the buffer, do not open it again. As shown in the figure below, click the "Open Key" button in the "Transfer after encryption" page, open the key file we saved before, such as "New.k", and then return to the "Transfer after encryption" page and click "Encryption Code". Click "Encrypt Code" button in the "Program Encryption Transfer" page, as shown in the following figure, the dialogue box of "Open Source File (Unencrypted)" will pop up firstly. The original unencrypted code file



Immediately after clicking the Open button, a similar dialogue box will pop up, but this time it is a dialogue box to save the encrypted file.

As shown in the figure below, click the Save button to save the encrypted file.



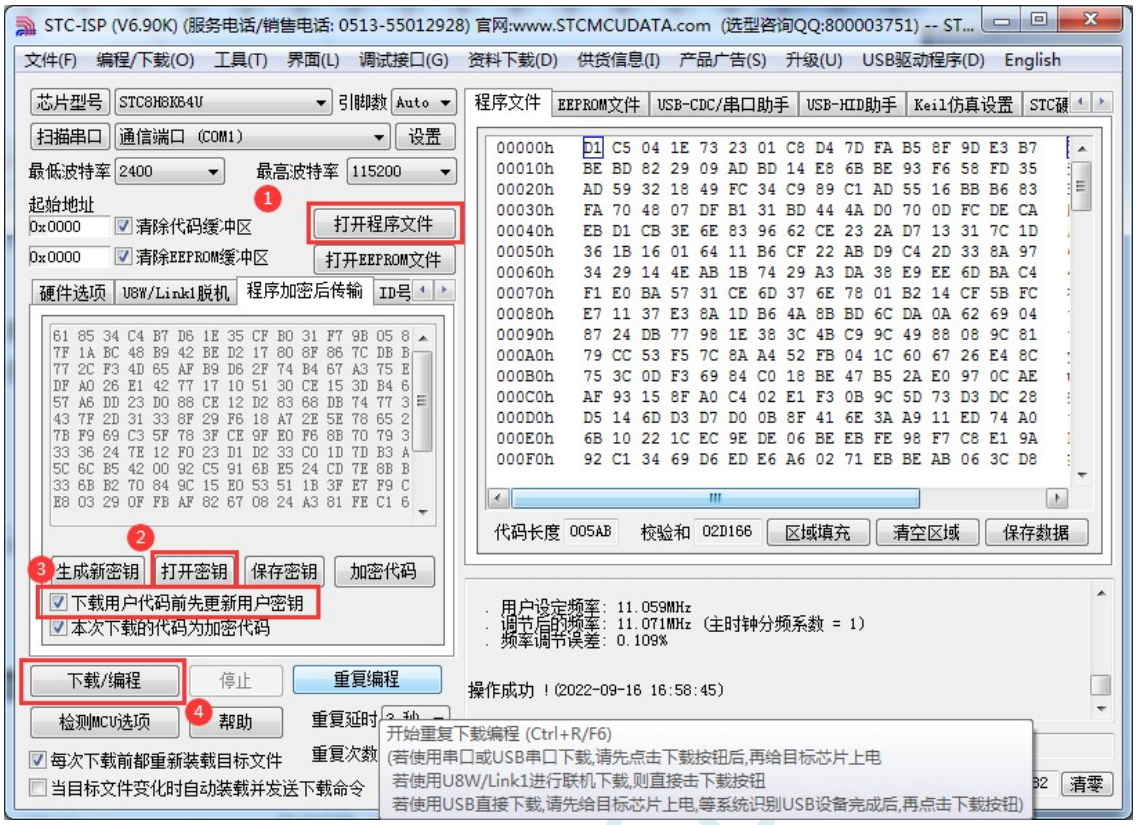


3. Update the user key into the target chip

Before updating the key, we need to open our own key. If our key is already stored in the buffer, do not open it again. As shown in the figure below, click the "Open Key" button in the "Custom Encryption Download" page to open the key file we saved before, for example, "New.k". After the key is opened, as shown in the figure below, tick the

"Update user key before downloading user code" option and "The downloaded code is encrypted code" option, then open the file we encrypted before, after opening it, click on the left side of the interface.

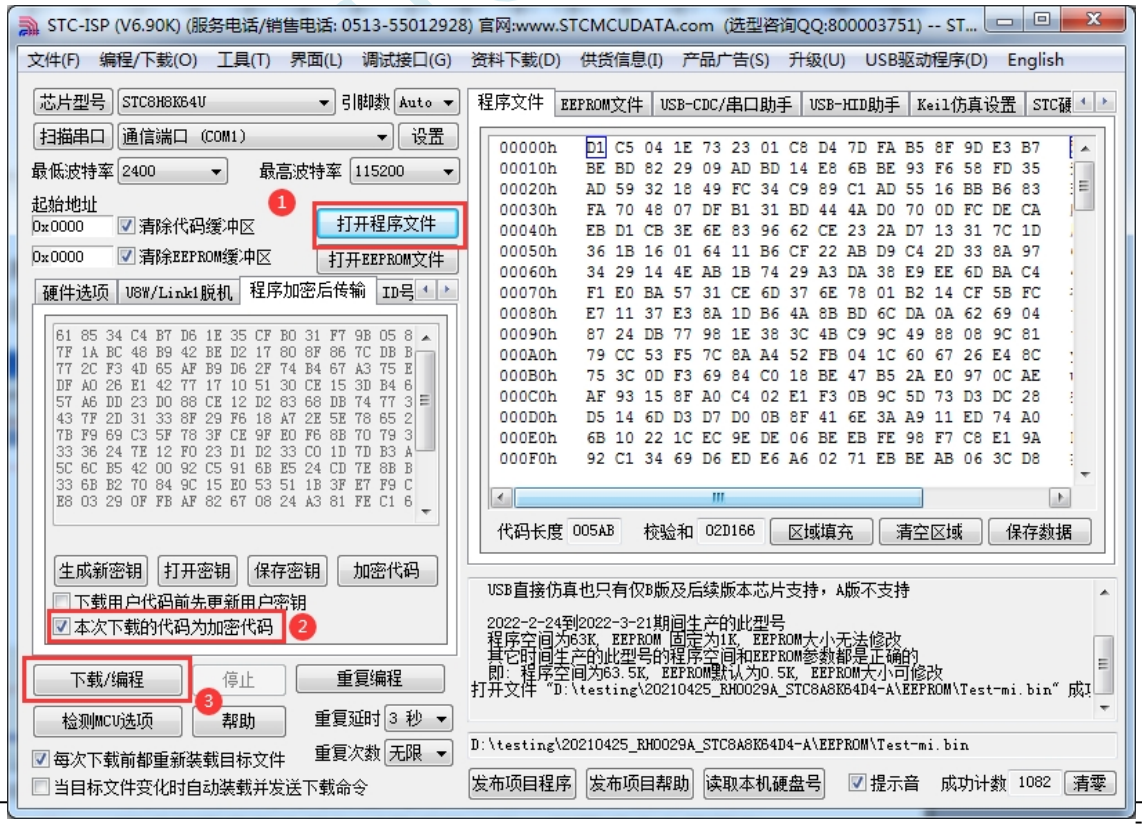
Download/Programming" button at the bottom corner of the screen, download the target



chip in the normal way to update the user key.

4. Encrypted update of user code

After the key is successfully updated, the target chip will have the function of receiving the encrypted code and restoring it. At this time, if you need to upgrade/update the code again, you only need to refer to the method in the second step, encrypt the target code, and then as follows



For a new STC microcontroller, steps 3 and 4 can be combined, i.e. updating the key to the target microcontroller and downloading the encrypted code to the microcontroller at the same time, if step 3 has already been performed (i.e., updating the key to the target chip).

If you have already downloaded the encrypted code, you only need to follow step 4 to update the code, and select the option of "The downloaded code is encrypted code" in the page of ''Transmission of encrypted code" (the option of "Update user key before downloading user code" does not need to be selected), then open the encrypted file, open it and click the "Download/Programming" button on the left corner of the interface to download the target chip in normal way. key" option does not need to be selected), and then open our previously encrypted file, open the interface, click on the lower-left corner of the "download/programming" button, according to the normal way of downloading the target chip can be completed with the user's own encrypted file to update the user's code for the purpose of (to prevent the burner in the burning of the programme by the burner by monitoring the serial number). (to prevent the code from being analysed by the burner through monitoring the serial port when burning the program).

E.3 Combined use of release project program + encrypted transmission of the program

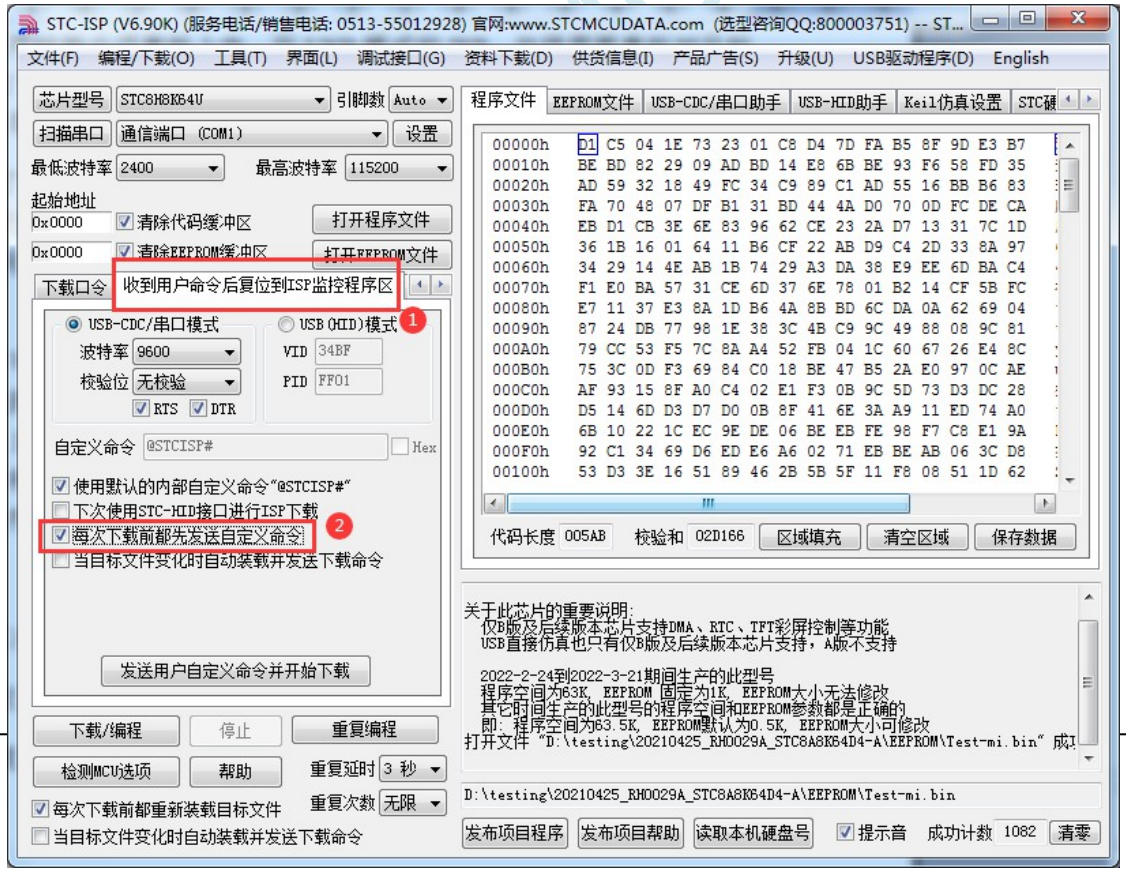
Two new special features can be used in conjunction with the release of the project programme and the encrypted transfer of the programme. Firstly, the encrypted program transfer ensures the confidentiality of the user code during the serial communication transfer when programming, while the release of the project program allows the end-user to update the software remotely (without the need for the solution company's personnel to be physically present). So the combination of the two functions, very suitable for solution companies / manufacturers in the software needs to be updated, so that the end user of the end product for the purpose of software updates, but also to ensure that the site burning personnel can not be analysed through the serial port of the useful procedures, is highly recommended for solution companies to use.

E.4 User-defined downloads (to enable non-stop downloads)

Downloading the user's target programme to the STC microcontroller is achieved by executing the internal ISP system code of the microcontroller and communicating with the host computer through the serial port or USB. However, the internal ISP system code of STC microcontroller will only be executed every time when the power is turned off and on again, which requires the user to re-power on the target microcontroller every time the user needs to update the programme, while the ISP in the USB mode requires the user to re-power on the target chip as well as pull down the P3.2 port to GND during the power-on process, which is not only necessary in the project in the development stage but also in the project in the development stage. For projects in the development stage, it is necessary to frequently modify and update the code, and it is very troublesome to re-power on the target chip every time it is downloaded.

STC microcontroller adds a soft reset register (IAP_CONTR) in the hardware design, which allows the user to set this register to decide whether the CPU resets to re-execute the user code or resets to the ISP area to execute the ISP system code. When 0x20 is written to the IAP_CONTR register, the CPU resets and re-executes the user code; when 0x60 is written to the IAP_CONTR register, the CPU resets and resets to the ISP area to execute the ISP system code.

To realise non-stop ISP download, users can design a code in the program, such as detecting a special key, or monitoring the serial port and waiting for a special serial port command, when the detection meets the download condition, the software will trigger the soft reset register to reset to the ISP area to execute the ISP system code, so as to realise non-stop ISP download. When the trigger condition is an external key, the user code can monitor the status of the key in real time. In order to synchronise the STC-ISP software with the user-triggered soft reset, it is necessary to use the function "Reset to ISP Monitor Area after Receiving User Command" provided in the STC-ISP software.



The steps to implement a non-stop ISP download are as follows:

1、Write user code and add serial port command monitoring procedure in user code

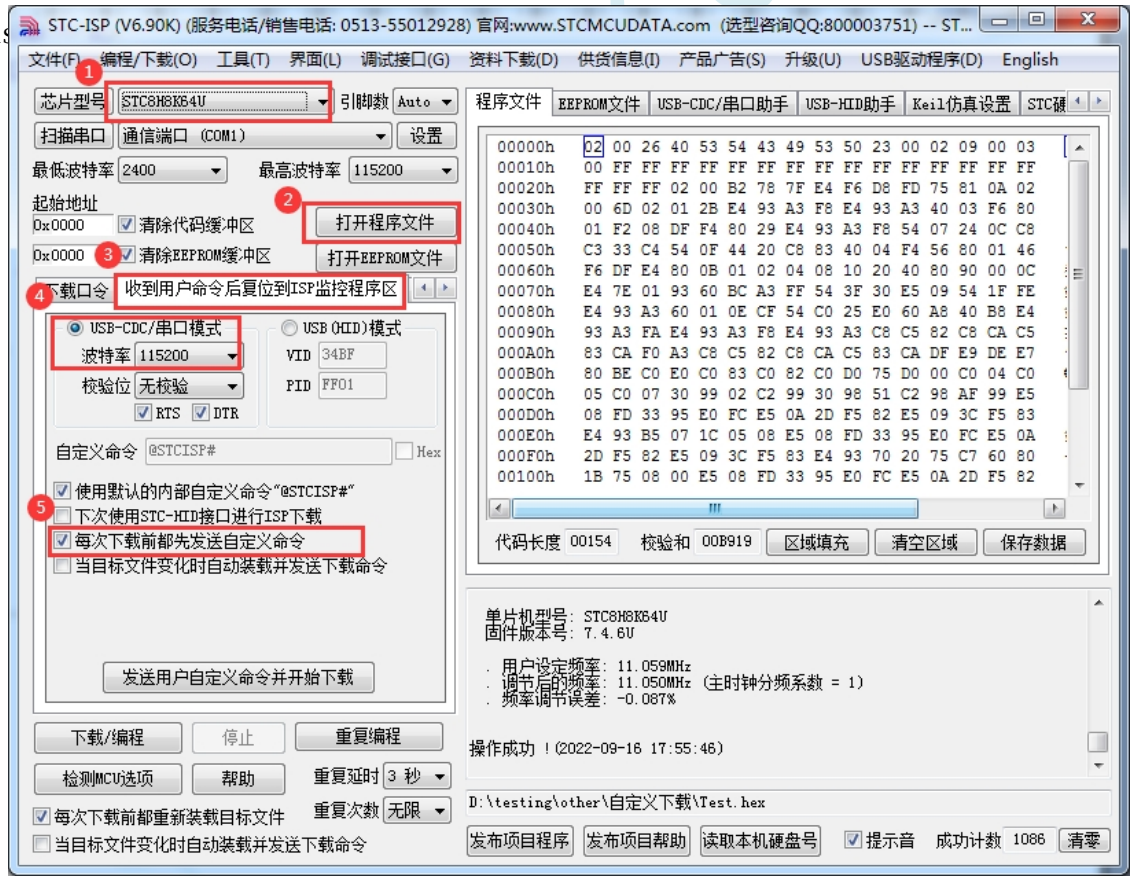
(The reference code is as follows, the test microcontroller model is STC8H8K64U)

```
#include "stc8h.h"

#define     FOSC            11059200UL
#define     BAUD            (65536 - (FOSC/115200+2)/4)
                                        The // plus 2 operation is to allow the Keil compiler to
                                        //Automatic implementation of rounding operations

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

void uart_isr() interrupt 4
{
    char dat.

    if (TI)
    {
        TI = 0;
    }

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

        if (dat == STCISPCMD[index])        // Judge whether the received data matches the current command character or not
        {
                            }
                    }

        }
        else
        {



        }
```

```
index++;                    [index] == '\0')          //determine if command
i                           //If match then index+1
    f                       matching is complete IAP_CONTR = 0x60;        //If
                            match complete then soft reset to ISP
    (
    S
    T
    C
    I
    S
    P
    C
    M
    D
                        index = 0;                          //if no match, need to
                        start from the beginning if (dat == STCISPCMD[index])
                            index++;

            void main()
            {
```

```
        p0m0 = 0x00; p0m1 = 0x00;
        p1m0 = 0x00; p1m1 = 0x00;
        p2m0 = 0x00; p2m1 = 0x00;
        p3m0 = 0x00; p3m1 = 0x00.

        SCON = 0x50;                    //Serial port initialisation
        auxr = 0x40; tmod
        = 0x00; th1 =
        baud >> 8; tl1 =
        baud.
        TR1 = 1;
        ES = 1;
        EA = 1;

        index = 0;                      //Initialisation command

        while (1);
    }
```

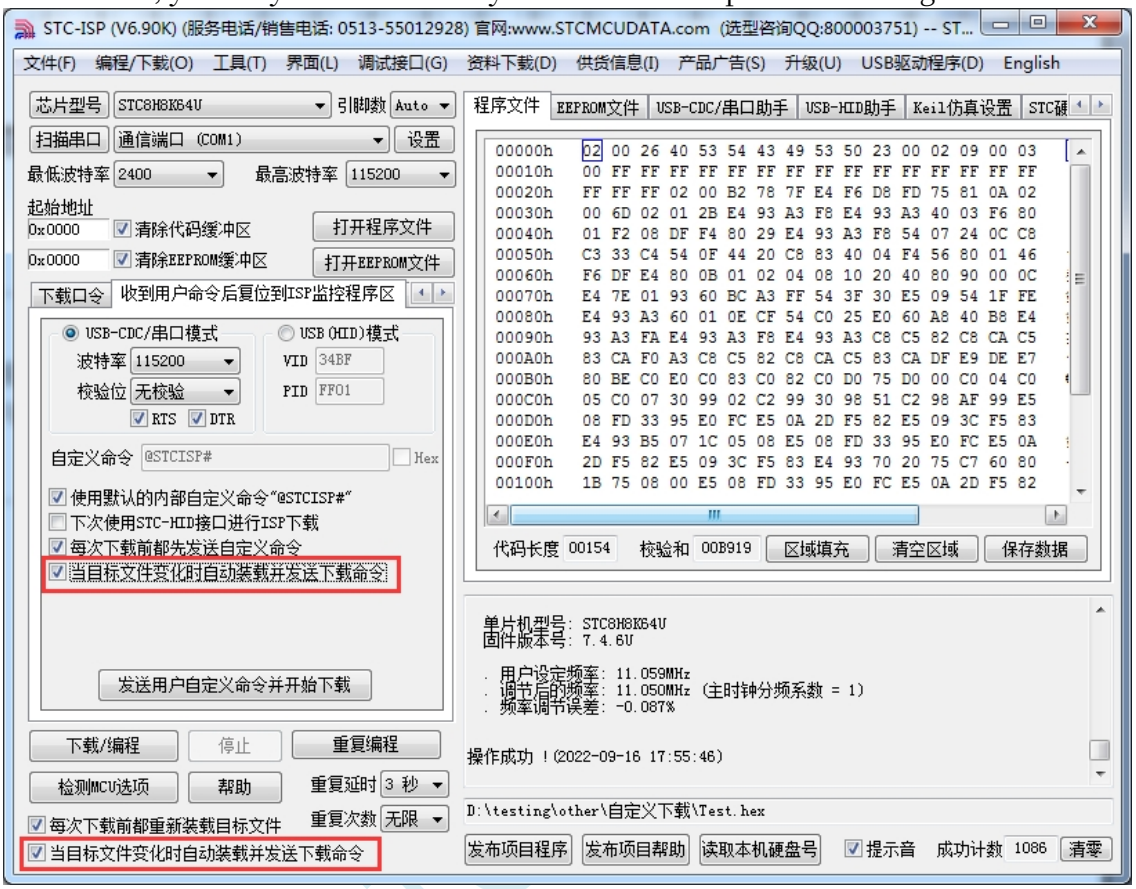
2. Set up the custom download command as shown in the following figure (the example us



3, the first time to download the target microcontroller needs to re-power on, and then each time to update only need to click on the download software in the "download/programming" button, the download software automatically sends the download command to the target microcontroller, the target microcontroller receives the command automatically reset to the system ISP area, you can realise the update of the user code

without stopping the power.

4, STC-ISP can also be realised in the project development phase, completely automatic download function, that is, when the download software detects the target code has been updated, it will automatically send the download command. To realise this function, you only need to tick any one of the two options in the figure below.



Appendix F Serial Interrupt Transceiver-MODBUS Protocol

C code

```
//Tested operating frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                    // see download software for header files

#define MAIN_Fosc        11059200L    //define master clock

/*************        Functional description **************
```

Please don't modify the programme first, directly download *"UART1.hex"* from *"08-Serial1* Interrupt *Transceiver-C Language-MODBUS* Protocol*" and* test it, *and* choose *11.0592MHZ as the* main frequency.
Test it before modifying the port.

Serial port *1* communicates according to *MODBUS-RTU* protocol. This example is a slave programme, the host is usually a computer.

This routine only supports multi-register read and multi-register write, the length of register is *64*, other commands can be added by user according to *MODBUS-RTU* protocol.

This example uses big-endian mode for data *(*consistent with

C51) and small-endian mode *for CRC16 (*consistent with *PC).*

Default parameters.
Serial port *1* settings are *1* bit start, *8 bits* data, *1* bit stop, no parity.
Serial port *1 (P3.0 P3.1): 9600bps.*

Timer *0* is used for timeout. *Timer 0 is used to* reset the timeout counter for each byte received by the serial port. When the serial port has been idle for more than *35bit* time *(9600bps* corresponds to *3.6ms)*, the reception is completed.
When users change the baud rate, they should be careful to change the timeout period.

This routine is just an application example, the *MODBUS-RTU* protocol is not in the scope of this example, users can search the Internet for the relevant protocol text reference. This example defines *64* registers, *the* access address is *0x1000~0x103f.*
Example command.
Write *4* registers *(8* bytes*).*
10 10 1000 0004 08 1234 5678 90AB CDEF 4930
Returns.
10 10 10 00 00 04 4B C6
Reads *4* registers.
10 03 1000 0004 4388
Returns.
10 03 08 12 34 56 78 90 ab cd ef 3d d5

Command error return message *(*custom*).*
0x90: Function code error. An unsupported function code was received.

0x91: Command length error.
0x92: Wrong number of registers or bytes written or read.
0x93: Register address error.

Note: *The* message is processed when the broadcast address *0x00* is received, but no answer is returned.

**********/

typedef unsigned char u8.

```
typedef unsigned int u16;
typedef unsigned
                    long
u32.

/**************        Local Constant Declaration **************/
#define    RX1_Length    128        /* Receive
                                    buffer length */
#define    TX1_Length    128        /* Send buffer
                                    length */


/**************        Local variable
declaration **************/ u8
xdataRX1_Buffer[RX1_Length];        //Receive buffer
u8 xdataTX1_Buffer[TX1_Length];     //transmit buffer

u8 RX1_cnt.          //Receive byte count.
u8 TX1_cnt.          //Transmit byte count
u8 TX1_number;       //number of
bytes to be sent u8 RX1_TimeOut.
                     //receive
timeout timer

bit B_RX1_OK.      // Receive data flag
bit B_TX1_Busy.    // send busy flag


/************* local function declaration **************/
void UART1_config(u32 brt, u8 timer, u8 io); // brt: communication baud rate, timer=2: baud rate using timer 2,
other values: u s e  Timer1 for baud rate. io=0: serial port 1 switch t o  P3.0 P3.1, =1: switch t o  P3.6 P3.7,
=2: switch t o P1.6 P1.7,                                    =3:switch t o  P4.3 P4.4.
    u8Timer0_Config(u8 t, u32 reload); //t=0: reload value is the number of master clock cycles, t=1: reload
value is the time (u n i t  us), return 0 correctly,
    return 1 load value is too large error.
      u16MODBUS_CRC16(u8 *p, u8 n);
u8    MODBUS_RTU(void).



#define              SL_ADDR0x10 /* This slave station address */
#define REG_ADDRESS 0x1000 /* register first
address                    */#define REG_LENGTH
                           64/* register length    */
      u16xdata modbus_reg[REG_LENGTH];      /* Register address */

//=====================================================================
// Functions: void main(void)
// Description: Main function
```

```
// Parameters: none.
// Returns: none.
// Version: VER1.0
// Date: 2018-4-2
// Remarks.
//========================================================================

void main(void)
{
```

```
u8 i;
u16 crc.

    EAXFR = 1;                                  //Enable access to XFR
    CKCON = 0x00;                               //Set the external data bus speed to fastest
    WTST = 0x00;                                //set the program code wait parameter.
                                                //Assign a value of 0 to set the CPU to execute the programme as fast as possible.

Timer0_Config(0, MAIN_Fosc / 10000); //t=0: reload value is the number of master clock cycles, (interrupt frequency, 20000 times/sec) UART1_config(9600UL, 1, 0); // brt: baud rate for communication, timer=2: baud rate is using timer 2, other values: use Timer1 for baud rate. timer1 for baud rate. io=0. Serial port 1 switches to P3.0 P3.1, =1: switches to P3.6 P3.7, =2: switches to P1.6 P1.7, =3: switches to P4.3 P4.4.

    EA = 1;

    while (1)
    {
        if(B_RX1_OK&& !B_TX1_Busy) //Receive data, perform MODBUS-RTU protocol parsing.
        {
            if(MODBUS_CRC16(RX1_Buffer, RX1_cnt) == 0) //first judge whether CRC16 is correct or not, if not, then ignore it, do not process it and do not return it.
information
            {
                if((RX1_Buffer[0] == 0x00) || (RX1_Buffer[0] == SL_ADDR)) // then determine if the station address is correct, the or if it is broadcast ground
address (no information returned)
                {
                    if(RX1_cnt > 2) RX1_cnt -= 2; //remove CRC16 checksum byte
                    i = MODBUS_RTU(); //MODBUS-RTU protocol parsing
                    if(i ! = 0) //Error handling
                    {
                        TX1_Buffer[0] =SL_ADDR; //station number address TX1_Buffer[1] =i;
                                                //error code
                        crc = MODBUS_CRC16(TX1_Buffer, 2);
                        TX1_Buffer[2] =(u8)(crc>>8); //CRC is small end mode TX1_Buffer[3] =(u8)crc;
                        B_TX1_Busy = 1;         //flag send busy TX1_cnt= 0;
                                                //transmit byte count TX1_number = 4;   //number of bytes to be sent TI = 1;
                                                //initiate send
                    }
```

```
            }
            RX1_cnt = 0;
            B_RX1_OK = 0;
        }
    }
}
```

```
/ * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  MODBUS_CRC (shift) *************** past test 06-11-27 *********
    Calculate CRC, call MODBUS_CRC16(&CRC,8); &CRC is first address, 8 is number of bytes
    CRC-16 for MODBUS
    CRC16=X16+X15+X2+1
    TEST: ---> ABCDEFGHIJ CRC16=0x0BEE 1627T
*/
//========================================================================
// Function: u16 MODBUS_CRC16(u8 *p, u8 n)
// Description: Calculate CRC16 function.
// Parameters: *p: Pointer to the data to be calculated.
//          n: number of bytes to count.
// Returns: CRC16 value.
// Version: V1.0, 2022-3-18 Liang Gong
//========================================================================
u16 MODBUS_CRC16(u8 *p, u8 n)
{
    u8 i;
    u16 crc16.

    crc16 = 0xffff; //preset the 1 6 - b i t  CRC register to
    0xffff (i.e. all 1s) do
    {
        crc16 ^= (u16)*p;       //Isolate the 8-bit data with the lower bit of the 16-bit CRC register and put the result
        in the CRC register.
        for(i=0; i<8; i++)      /8-bit data
        {
            if(crc16 & 1) crc16 = (crc16 >> 1) ^ 0xA001; // if the lowest bit is 0, shift the contents of the CRC register
one bit to the right (towards the lower bit) and fill the highest bit with 0.
                                        //Re-alternative polynomial 0xA001
            else crc16 >>= 1;                   //If the lowest bit is 0, shift the content of the CRC register one bit to the
            right (towards the lower bit) and fill the highest bit with 0.
        }
        p++;
    }while(--n ! = 0);
    return ( crc16);
}

/******************** modbus protocol ************************/
/*************************************************************************
```

Write Multiple Registers

Data. address	function code	Register address	Number of registers	Number of bytes written	Write data	CRC16
Offset.	0	1	2	3 4 5	6	7~	Last 2 bytes
Bytes.	1 byte	1 byte	2 byte 2 byte	1	byte2*n	byte2 byte

data 0x10 xxxx xxxx xxxx.... .xx xxxx

come (or go) back

Data. Address Function Code Register Address Number of Registers *CRC16*

Offset: *0 0 12 34 5 6 7*

Byte: byte byte bytebyte *byte*

addr 0x10xxxx xxxxxxxxxx

Read Multiple Registers

Data: Station number (address) Function Code Register address Number of registers *CRC16*

Offset: *0 0 12 34 56 7*

Byte: *1 byte 1 byte2 byte 2 byte2 byte*

addr 0x03xxxx xxxx xxxx

come (or go) back

Data: Station number (address) Function code Read byte number Read data *CRC16*

Offset: *0 0 1 2 3~* Last *2* bytes

Byte: *1 byte1 byte 1byte2*n byte 2 byte*

addr 0x03 xx xx..... .xx xxxx

Return error code

Data:Station number (address) Error Code *CRC16*

Offset: *0 0 1* Last *2* bytes

Byte: *1 byte1 byte2 byte*

addr 0x03 xxxx

```
**********************************************************************************/
u8 MODBUS_RTU(void)
{
    u8 i,j,k.
    u16    reg_addr;    //register address u8 reg_len;//number of registers to write u16 crc.

    if(RX1_Buffer[1] == 0x10)//write multiple registers
    {
        if(RX1_cnt < 9)    return 0x91;    //Command length error
        if((RX1_Buffer[4] != 0) || ((RX1_Buffer[5] *2) != RX1_Buffer[6]))return 0x92; //write register number and byte count error if((RX1_Buffer[5]==0) || (RX1_Buffer[5] > REG_LENGTH))return 0x92; //write register number error

        reg_addr = ((u16)RX1_Buffer[2] << 8) + RX1_Buffer[3]; //register address reg_len = RX1_Buffer[5] ; //number of written registers
        if((reg_addr+(u16)RX1_Buffer[5]) > (REG_ADDRESS+REG_LENGTH)) return 0x93; //register address error if(reg_addr< REG_ADDRESS)  return 0x93;//register address error
        if((reg_len*2+7) != RX1_cnt) return 0x91; //command length error

        j =reg_addr - REG_ADDRESS; //register data subscript for(k=7, i=0; i<reg_len; i++,j++)
        {
```

```
        modbus_reg[j] = ((u16)RX1_Buffer[k] << 8) + RX1_Buffer[k+1]; //write data, big end mode
        k += 2;
```

```
    }

    if(RX1_Buffer[0] != 0) //answer if not broadcast address
    {
      for(i=0; i<6; i++) TX1_Buffer[i] = RX1_Buffer[i]; // Answer to
      be returned crc = MODBUS_CRC16(TX1_Buffer, 6).
      TX1_Buffer[6] = (u8)(crc>>8); /CRC is small end mode
      TX1_Buffer[7] = (u8)crc;
      B_TX1_Busy = 1;     //flag send
      busy          TX1_cnt= 0;
                    //send byte count
      TX1_number = 8;     //number of
      bytes to send TI = 1;     //initiate
      sending
    }
  }
  else if(RX1_Buffer[1] == 0x03) //read multiple registers
  {
    if(RX1_Buffer[0] != 0) //answer if not broadcast address
    {
      if(RX1_cnt != 6)    return 0x91;     // command length
      error if(RX1_Buffer[4] != 0) return 0x92; //read
      register count error
      if((RX1_Buffer[5]==0) || (RX1_Buffer[5] > REG_LENGTH))return 0x92; //read register number error

      reg_addr = ((u16)RX1_Buffer[2] << 8) + RX1_Buffer[3]; //register
      address reg_len = RX1_Buffer[5] ; //read number of
      registers
      if((reg_addr+(u16)RX1_Buffer[5]) > (REG_ADDRESS+REG_LENGTH)) return 0x93; //register address
      errorif(reg_addr< REG_ADDRESS)          return 0x93;//register address error

      j = reg_addr - REG_ADDRESS; //register data
      subscript TX1_Buffer[0] = SL_ADDR;
      //station address TX1_Buffer[1] = 0x03;
                          //read function code
      TX1_Buffer[2] = reg_len*2; //return byte
      number

      for(k=3, i=0; i<reg_len; i++,j++)
      {
        TX1_Buffer[k++] = (u8)(modbus_reg[j] >> 8);//data for big
        end mode TX1_Buffer[k++] = (u8)modbus_reg[j];
      }
      crc = MODBUS_CRC16(TX1_Buffer, k);
      TX1_Buffer[k++] = (u8)(crc>>8); //CRC is small
      end mode TX1_Buffer[k++] = (u8)crc;
      B_TX1_Busy = 1;     //flag send
```

```
        busy
                    TX1_cnt= 0;
                    //transmit byte
        count TX1_number = k;  //number
        of bytes to be sent TI = 1;
                    //initiate transmit
    }
}
```

```
    else return 0x90; //function code error

    return 0; // correctly parsed
}


//========================================================================
// Function:u8 Timer0_Config(u8 t, u32 reload)
// Description: timer0 initialisation function.
// Parameters: t: reload value type, 0 means reload is system clock number, the rest means reload is time (us).
//       reload: reload value.
// Returns: 0: correct initialisation, 1: reload value too large, incorrect initialisation.
// Version: V1.0, 2018-3-5
//========================================================================
u8 Timer0_Config(u8 t, u32 reload) //t=0: reload value is the number of master clock cycles, t=1: reload value is the time (in us)
{
    TR0 = 0; //stop counting

    if(t != 0) reload = (u32)(((float)MAIN_Fosc * (float)reload)/1000000UL); //Reload is time(us), calculate the number of system clocks needed.
    if(reload >= (65536UL * 12)) return 1; //value too large, return error
    if(reload < 65536UL) AUXR |= 0x80;      //1T mode
    else
    {
        AUXR &= ~0x80; //12T mode
        reload = reload / 12;
    }
    reload = 65536UL - reload;
    TH0 = (u8)(reload >> 8);
    TL0 = (u8)(reload).

    ET0 = 1; // Allow
    interrupt TMOD
    &= 0xf0.
    TMOD |= 0; //Operating mode, 0: 16-bit auto-reload, 1: 16-bit timer/counter, 2: 8-bit auto-reload, 3: 16-bit auto-reload, non-maskable interrupt
    TR0 = 1;            //start running
    return 0;
}

//========================================================================
// Function: void timer0_ISR (void)  interrupt TIMER0_VECTOR
// Description: timer0 interrupt function.
// Parameters: none.
```

```
// Returns: none.
// Version: V1.0, 2016-5-12
//========================================================================
```

void timer0_ISR (void) interrupt 1

```
{
    if(RX1_TimeOut ! = 0)
    {
        if(--RX1_TimeOut == 0) //timeout
        {
            if(RX1_cnt ! = 0) //Receive with data
            {
                B_RX1_OK = 1; //flag that data block has been received
            }
        }
    }
}
```

```
//========================================================================
// Function: SetTimer2Baudraye(u16 dat)
// Description: Sets Timer2 to be the baud rate generator.
// Parameters: dat: Reload value for Timer2.
// Returns: none.
// Version: VER1.0
// Date: 2018-4-2
// Remarks.
//========================================================================
void SetTimer2Baudraye(u16 dat) // Select baud rate, 2: use Timer2 for baud rate, other values: use Timer1 for baud rate.
{
    AUXR &= ~(1<<4); //Timer stop
    AUXR &= ~(1<<3); //Timer2 set As
    TimerAUXR |=            (1<<2);//Timer2 set as
    1T mode TH2 = (u8)(dat >> 8); //Timer2 set as
    1T mode TH2 = (u8)(dat >> 8).
    TL2 = (u8)dat.
            IE2&= ~(1<<2); //disable interrupts
    AUXR |=     (1<<4); //Timer run enable
}
```

```
//========================================================================
// Function: void UART1_config(u32 brt, u8 timer, u8 io)
// Description: UART1 initialisation function.
// Parameters: brt        brt: Baud rate.
//         timer: timer for b a u d  rate, timer=2: timer 2 for b a u d  rate,  other values: use Timer1 for baud rate.
          //io: IO that serial port 1 switches to, io=0: serial port 1 switches to P3.0 P3.1, =1: switches to P3.6 P3.7, =2: switches to P1.6 P1.7, =3: switches to P4.3 P4.4.
// Returns: none.
// Version: VER1.0
```

```
// Date: 2018-4-2

// Remarks.
```

```
//=========================================================================
void UART1_config(u32 brt, u8 timer, u8 io) // brt: communication baud rate, timer=2: baud rate using timer 2,
other values: u s e Timer1 for baud rate. io=0: serial port 1 switch t o P3.0 P3.1, =1: switch t o P3.6 P3.7,
=2: switch t o P1.6 P1.7,                                    =3:switch t o P4.3 P4.4.
{
    brt     = 65536UL - (MAIN_Fosc / 4) / brt;
    if(timer == 2) // baud rate using timer
    2
    {
        AUXR |= 0x01;       //S1 BRT Use Timer2;
        SetTimer2Baudraye((u16)brt).
    }

    else        //Baud rate use timer 1
    {
        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)(brt >> 8);
        TL1 = (u8)brt.
        ET1 = 0; // Disable Timer1 interrupt.      // Disable Timer1 interrupt
        TR1 = 1; // Run Timer1.          // Run Timer1
    }
    P_SW1 &= ~0xc0; //default switch to
    P3.0 P3.1 if(io == 1)
    {
        P_SW1 |= 0x40;      //switch to P3.6
        P3.7 P3M1 &= ~0xc0;
        p3m0 &= ~0xc0.
    }
    else if(io == 2)
    {
        P_SW1 |= 0x80; //switch to P1.6 P1.7
        P1M1 &= ~0xc0;
        P1M0 &= ~0xc0.
    }
    else if(io == 3)
    {
        P_SW1 |= 0xc0; //switch to P4.3 P4.4
        P4M1 &= ~0x18;
        P4M0 &= ~0x18.
    }
    else
    {
        p3m1 &= ~0x03;
```

```
p3m0 &= ~0x03;
```

```
    }

    SCON = (SCON & 0x3f) | (1<<6); // 8-bit data, 1 start bit, 1 stop bit, no parity.      //8-bit data, 1 start bit, 1 stop bit, no
    parity
// PS= 1; // high priority
    interrupt  ES= 1; //
    interrupt allowed
    REN = 1; // receive
    allowed
}


//========================================================================
// Function: void UART1_ISR (void) interrupt UART1_VECTOR
// Description: Serial 1 interrupt function
// Parameters: none.
// Returns: none.
// Version: VER1.0
// Date: 2018-4-2
// Remarks.
//========================================================================
void UART1_ISR (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(!B_RX1_OK) // receive buffer idle
        {
            if(RX1_cnt >= RX1_Length) RX1_cnt = 0; RX1_Buffer[RX1_cnt++]
            = SBUF;
            RX1_TimeOut = 36; //Receive timeout timer, 35 bit times
        }
    }

    if(TI)
    {
        TI = 0;
        if(TX1_number ! = 0) //there is data to send
        {
            SBUF = TX1_Buffer[TX1_cnt++];
            TX1_number --;
        }
        else B_TX1_Busy = 0;
    }
}
```

Appendix G About whether to bake before reflow soldering

In accordance with the requirements of the International Moisture Sensitivity Level 3 (MSL3) specification, within 168 hours of unpacking a chip component in a vacuum package, the chip component must be

7 Within days, the reflow soldering patch must be completed, if not completed, it must be baked again at high temperature.

SOP/TSSOP plastic tube can not withstand high temperatures of more than 100 degrees, after unpacking the vacuum packaging must be reflow soldering within 7 days after the completion of the patch.

Otherwise, before reflow soldering to remove the plastic tube can not withstand high temperatures of more than 100 degrees, put into a metal tray, re-bake: 110 ~ 125 ℃, 4 ~ 8 Any hour will do.

LQFP/QFN/DFN tray can withstand more than 100 degrees of high temperature, after unpacking the vacuum package must be reflow soldering within 7 days of the completion of the patch, otherwise reflow soldering must be re-baked before: 110 ~ 125 ℃, 4 ~ 8 hours are possible

Appendix H How to use a multimeter to test the chip I/O port good or bad

According to the requirements of the international moisture sensitivity level 3 (MSL3) specification, the SMD components must be reflow soldered within 168 hours and 7 days after unpacking the vacuum package, and if not completed, they must be baked at high temperature again. If there is no high-temperature baking process, direct reflow soldering, the chip internal metal wires may be pulled off due to uneven heat inside and outside the chip, and the final phenomenon is the chip I/O port damage.

STC's microcontroller in the chip design, each I/O port has two protection diodes to VCC and GND, respectively, with a multimeter diode monitoring file can be measured. You can use this method to judge the good or bad condition of the I/O pins. Use the multimeter measurement method as follows (Note: Here we use a digital multimeter)

First of all, the multimeter will be adjusted to the diode detection block, the chip under test should not be powered, the **red pen** of the multimeter will be connected to the **GND pin** of the chip under test, and the **black pen will** be used to measure each I/O port in turn, if the parameter displayed by the multimeter is 0.7V or so, it means that the protection diode of the chip from the internal I/O to the GND is normal, i.e., the wire is intact, and if the parameter displayed is 0V, it means that the wire has been pulled off inside the chip. If the displayed parameter is 0V, it means that the wire inside the chip has been pulled off.

The above method is a way to detect the wiring inside the chip.

In addition, if the pins of the microcontroller are not protected on the user's board, the overcurrent or overvoltage may cause the I/O to be burned out. In order to detect whether the pins are burned out, in addition to using the above method to detect the protection diode from the I/O port to GND, it is also necessary to detect the protection diode from the I/O port to VCC. Use a multimeter to test the protection diode from the I/O port to VCC as follows:

First of all, the multimeter will be adjusted to the diode detection block, the chip under test should not be powered, the **black pen** of the multimeter will be connected to the **VCC pin** of the chip under test, and the **red pen will** be used to measure each I/O port in turn, if the parameter displayed by the multimeter is about 0.7V, it means that the protection diode of the chip's internal I/O to the VCC is normal, and if it is displayed with a parameter of 0V, it means that the chip's port has been damaged.

Appendix I High-volume production, how to eliminate specialised burn-in personnel, and how to have no burn-in sessions

Mass production, you will be the STC's MCU as the main control chip control board assembled into the device before you will STC MCU chip to your control board after the completion of the control board, you must test your control board good or bad. Don't say 100%, straight through no problem, that's a bar, not production, as long as the production, there will be false soldering, short circuit, part of the original paste wrong, part of the original procurement error.

So after the patch back, assembled to the inside of the shell before you have to test, your contains STC MCU control board good or bad, good to assemble, bad to repair salvage.

Test, mass production, must have test rack/connect our offline burning tool U8W/U8W-Mini/STC-USB Link1D underneath, also connect to other control sections

Connected via USER-VCC, P3.0, P3.1, GND, to be powered on each time by workers

Connected via S-VCC, P3.0, P3.1, GND, don't want you to turn on the power, STC's offline tool gives you automatic power supply

It costs under $500 to have a test stand made for you out there, which is plexiglass, jigs, and thimbles.

1 worker to test that your control board is working properly Manage 2 - 3 test stands

Operational Processes:

1. Snap your STC MCU control board into Test Rack 1.
2、Clip your STC MCU control board to Test Rack 2, the programme on Test Rack 1 has been burned in/not feeling the burning time.
3, test test rack 1 on the STC main control board function is normal, normal to the normal area, not normal, to the abnormal zone
4. Snap on a new untested, unprogrammed control board to Test Rack 1.
5、Test The untested control board/programme on the test stand 2 has been burnt without knowing when, replace it with a new untested and unburnt control board.
6. Cycle steps 3 to 5

======== does not need to arrange for a burner

Appendix J Notes on the 0xFD Issue in Keil Software

As we all know, all versions of Keil 8051 and 80251 compilers have a problem called 0xFD, which is mainly manifested in the fact that strings can not contain Chinese characters with 0xFD encoding, or else Keil software will skip 0xFD during compilation and mess up the code.

Keil's official response to this problem is that the 3 character encodings 0xfd, 0xfe, and 0xff are used internally by the Keil compiler, so 0xfd will be automatically skipped by the compiler if the code contains a string with 0xfd in it.

Keil official solution: add a 0xfd after the Chinese character with 0xfd code.

For example: printf("Maths"); //Keil compiled print will show garbled code.

printf("Number \xfd 学");//显示正常

Here "\xfd" is an escape character in standard C code, "\x" means the next 1~2 characters are hexadecimal. "\xfd" means insert the hexadecimal number 0xfd into the string.

Since the Chinese character code of "数" is 0xCAFD, Keil will skip the FD when compiling, and only compile the CA into the target file, and then manually add another 0xfd to the target file through the escape character to form a complete 0xCAFD, which can be displayed normally.

There are a lot of patches for 0xFD on the internet, but basically they are only valid for the old version of Keil software. The method of patching is to find the key code [80 FB FD] in the executable file and modify it to [80 FB FF]. This modification method is too simple to find the key code, and it is very easy to modify it to other irrelevant places, which will lead to some inexplicable problems when the compiled target file is running. Therefore, if the string in the code contains the following Chinese characters, it is recommended to use the solution provided by Keil to solve the problem.

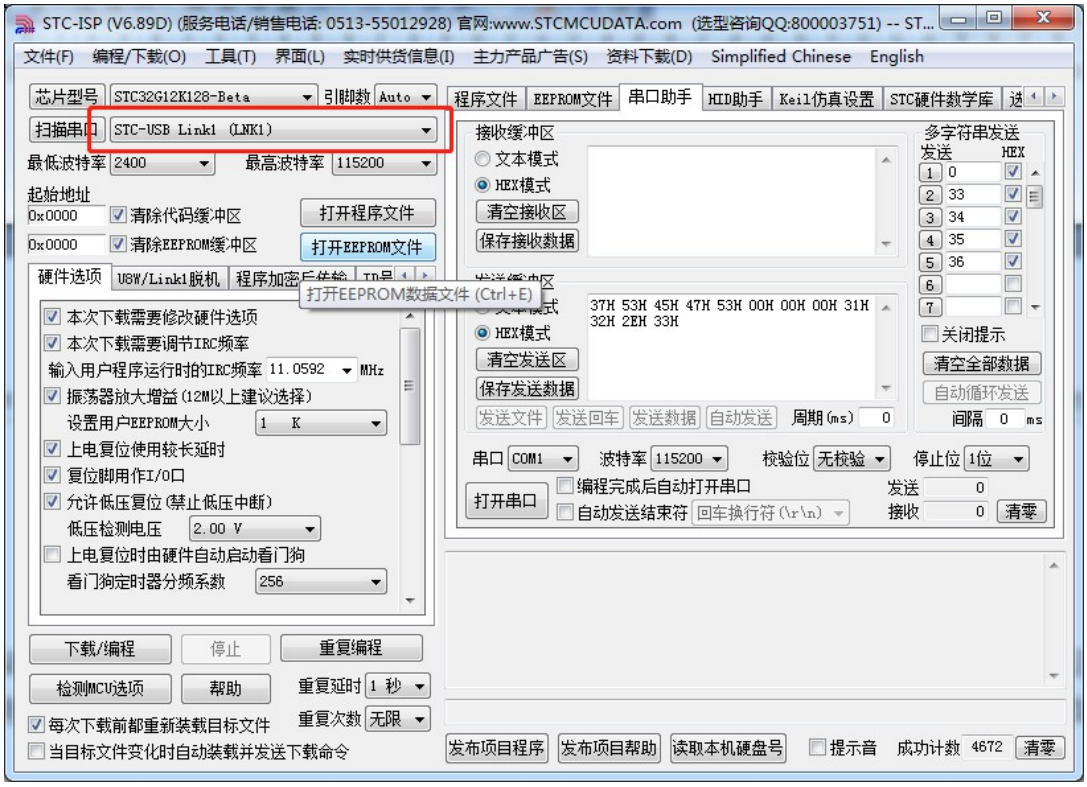
In GB2312, the Chinese characters containing the 0xfd code are as follows: 褒饼昌除待谍洱俘庚过糊积箭烬君魁例笼慢谬凝琵讫驱三升数她听妄锡淆旋妖引育札正铸 佚冽邶埤荤蔟摭啐帻帻噴猃泯潺妪纨鍾琮槊辇錠酏觚鳊鼾

In addition, the Keil project path name must not contain Chinese characters with 0xFD encoding, otherwise the Keil software will not be able to compile the project correctly.

Appendix KSTC-USB Link1 Tool Usage Precautions

K.1 Correct identification of tools

When the STC-USB Link1 tool is shipped from the factory, the STC-USB Link1 control programme has been burned into the main control chip. Under normal circumstances,



when the tool is connected to the computer, "STC-USB Link1 (LNK1)" will be recognised in the STC-ISP download software immediately, as shown in the figure below.

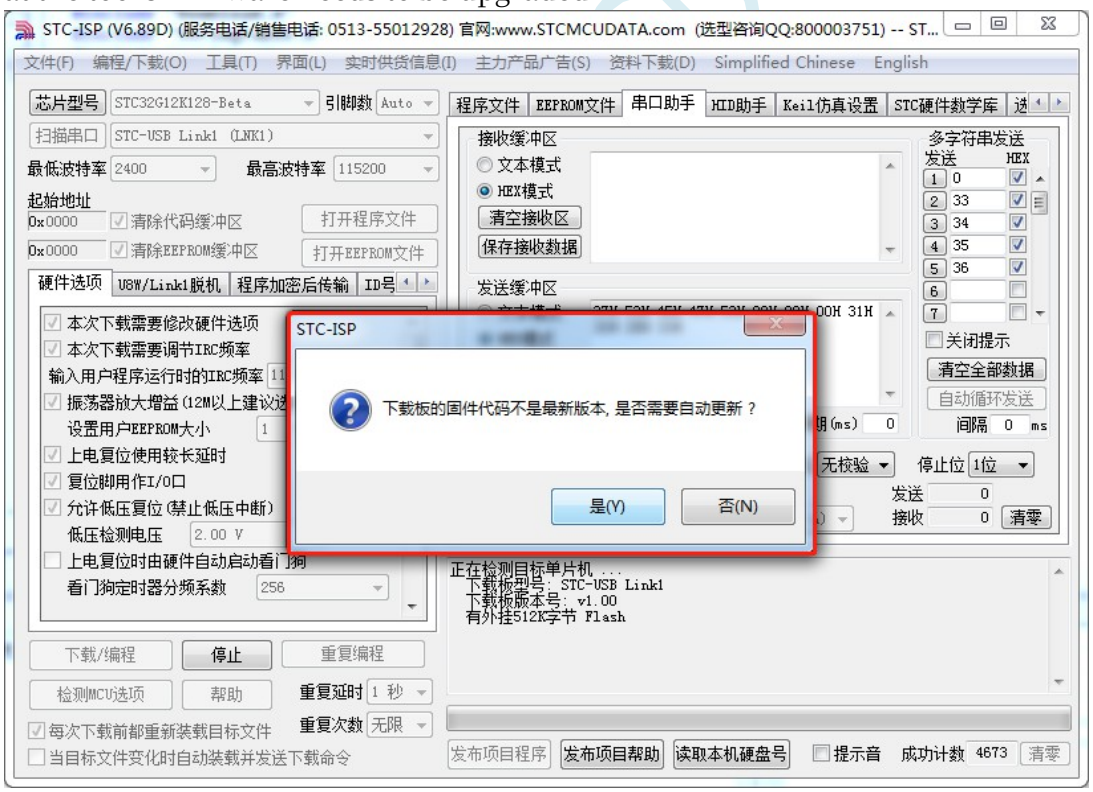
Once correctly recognised, you can use the STC-USB Link1 for online ISP download or offline ISP download.

If the tool does not recognise "STC-USB Link1 (LNK1)" when it is connected to the computer, but is always recognised as "STC USB Writer (HID1)", please make sure that the toggle switches on the tool in the position shown in the figure below Is the toggle switch on the tool in the position shown in the figure below set to the "Burn & Emulate" position?

After turning the switch to the burn and emulation position and reconnecting the tool to the computer, the "STC-USB Link1 (LNK1)" can be correctly recognised.

K.2 Automatic upgrade of tool firmware

When using the tool for ISP download, the software pops up the following screen, indicating that the tool's firmware needs to be upgraded



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

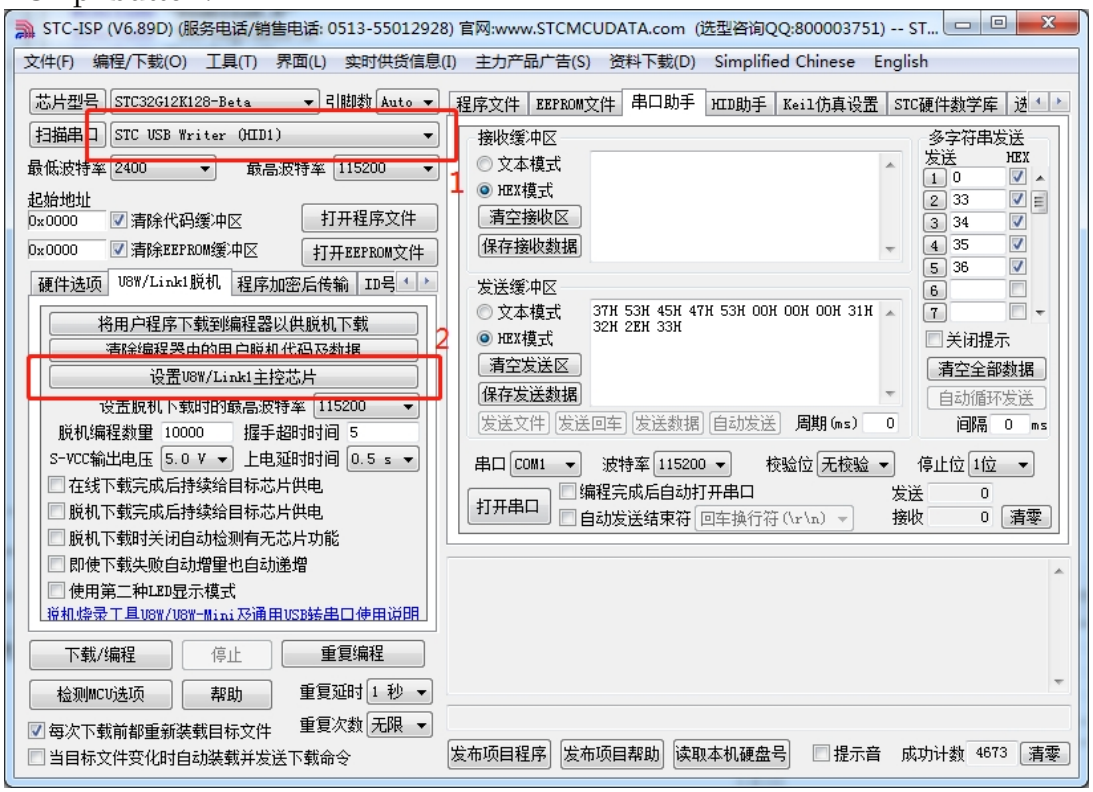
K.3 Access to the updated firmware

Please do not power off the tool during the automatic upgrade process. If there is an abnormal power failure during the upgrade process, or the control programme of the main control chip is lost due to other reasons, resulting in the download software not being able to

correctly identify the "STC-USB Link1 (LNK1)", then you need to manually burn the control programme. Firstly, turn the toggle switch to "Update Tool Firmware" (as shown below).

Wait for the STC-ISP download software to recognise the "STC USB Writer (HID1)", then click "Set U8W/Link1" as shown in the figure below.

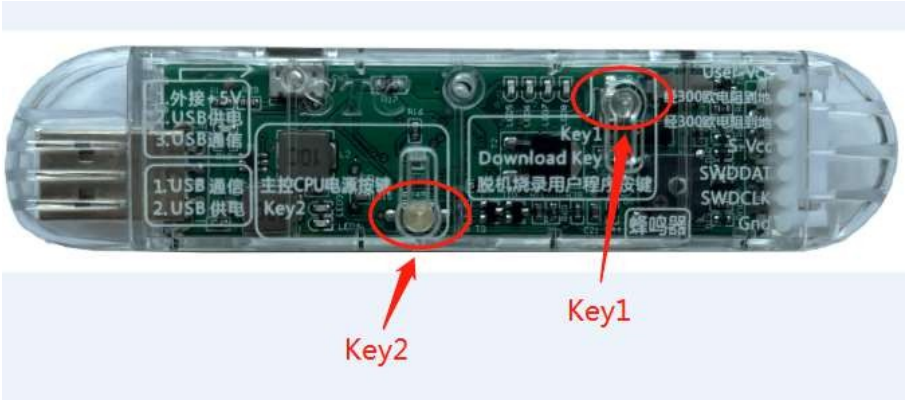
Master Chip" button.



After programming, always remember to turn the toggle switch back to the "Burn & Emulate" position.

K.4 Accessing the firmware update 2

If there is no toggle switch on subsequent versions of the tool, to access the "Update Tool Firmware": first connect the tool to the computer using the USB cable, then press and hold Key1 on the tool, then press Key2 and wait for the STC-ISP downloader software to recognise the "STC USB Writer (HID1)" before releasing Key1. STC USB Writer (HID1)" and then release Key1.

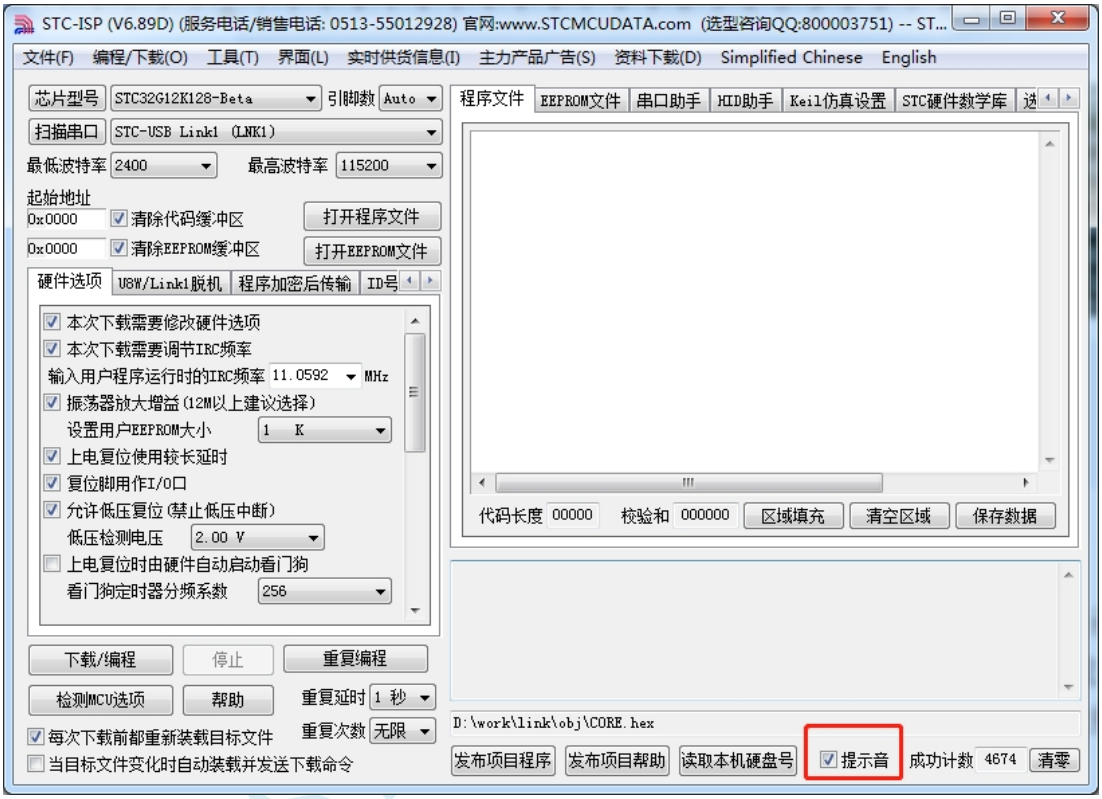

1.外接+5V
2.USB供电
3.USB通信
1.USB 通信
2.USB 供电
主控CPU电源按键
Key2
Key1
Download Key
脱机烧录用户程序按键
蜂鸣器
SWDDAT
SWDCLK
Key2
Key1

K.5 STC-USB Link1 Operating Indicator Description

The STC-USB Link1 operating indicator modes are as follows:

1、During ISP online or offline download, 4 LEDs will display in the form of running lights.

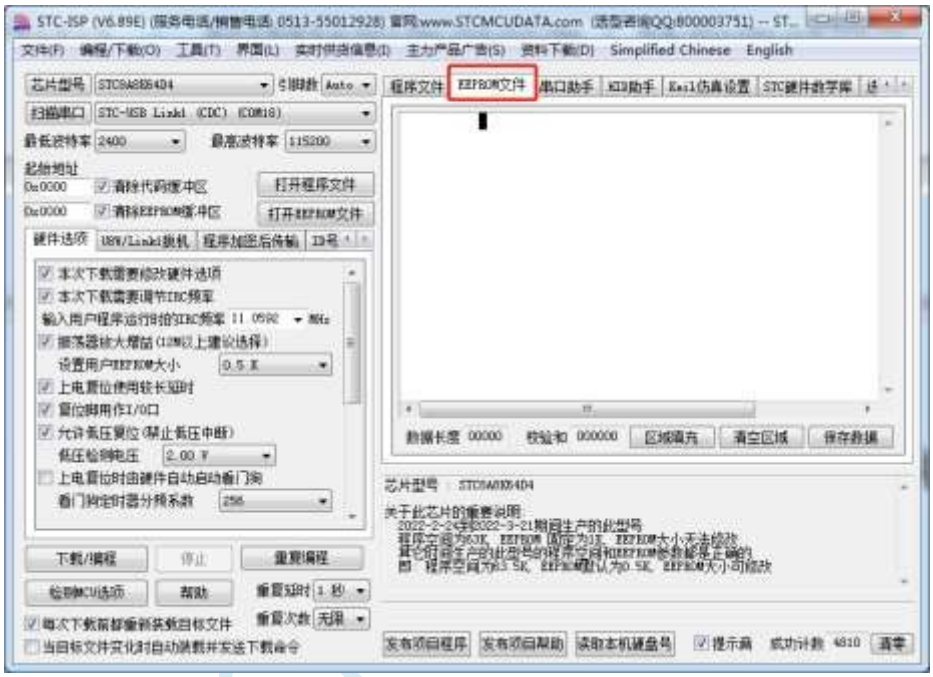
2、After the download is completed, if the download is correct, the 4 LEDs will flash at the same time and the buzzer will emit two short beeps; if the download fails, the 4 LEDs will go out at the same time and the buzzer will emit one short and one long beep. (The switch of the beeps is set in the ISP download software as shown in the following figure.)



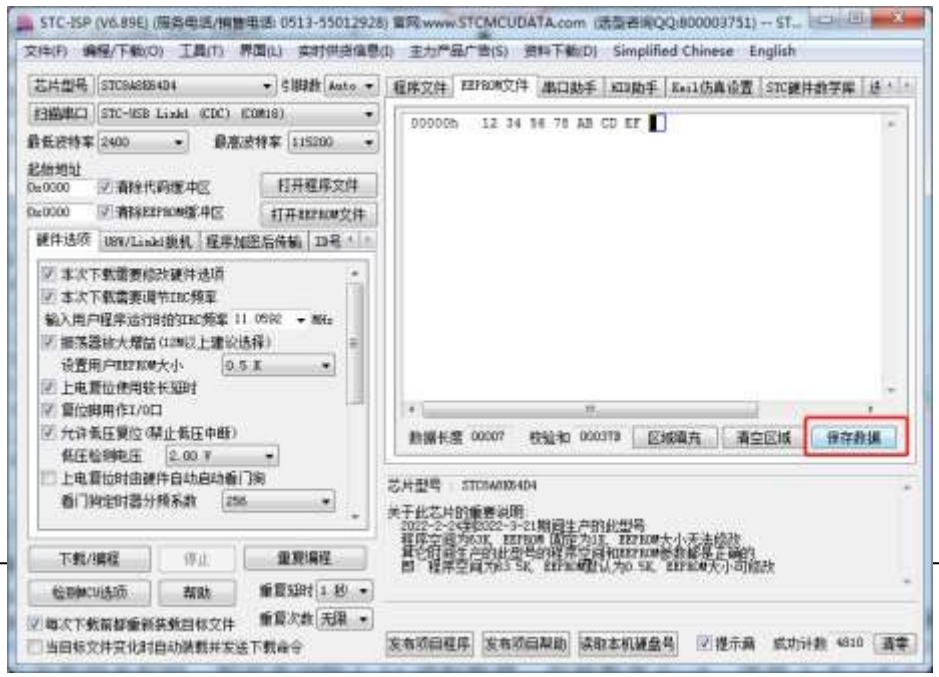
Appendix L How to create and edit with STC-ISP download software

EEPROM File

Open any version of STC-ISP download software, select the "EEPROM" page, and click the data window as follows



When the black rectangle cursor appears, you can manually input hexadecimal data, including numbers 0 to 9, letters A to F (case sensitive) After data input is complete, click the "Save Data" button to save the EEPROM data.



Appendix MSTC32 Series Header File Definitions

```
#ifndef   STC32G_H_
#define   STC32G_H_

//////////////////////////////////////////////////
#include <intrins.h>
```

//Include this header file without including "REG51.H".

```
sfr     P0      =     0x80;
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.
sfr     SP      =     0x81;
sfr     DPL     =     0x82;
sfr     DPH     =     0x83;
sfr     DPXL    =     0x84;
sfr     SPH     =     0x85;
sfr     PCON    =     0x87;
sbit    SMOD    =     PCON^7.
sbit    SMOD0   =     PCON^6.
sbit    LVDF    =     PCON^5.
sbit    POF     =     PCON^4.
sbit    GF1     =     PCON^3.
sbit    GF0     =     PCON^2.
sbit    PD      =     PCON^1.
sbit    IDL     =     PCON^0.
sfr     TCON    =     0x88;
sbit    TF1     =     TCON^7.
sbit    TR1     =     TCON^6.
sbit    TF0     =     TCON^5.
sbit    TR0     =     TCON^4.
sbit    IE1     =     TCON^3.
```

```
sbit    IT1     =   TCON^2.
sbit    IE0     =   TCON^1.
```

```
sbit    IT0     =   TCON^0.
sfr     TMOD    =   0x89;
sbit    T1_GATE =   TMOD^7.
sbit    T1_CT   =   TMOD^6.
sbit    T1_M1   =   TMOD^5.
sbit    T1_M0   =   TMOD^4.
sbit    T0_GATE =   TMOD^3.
sbit    T0_CT   =   TMOD^2.
sbit    T0_M1   =   TMOD^1.
sbit    T0_M0   =   TMOD^0.
sfr     TL0     =   0x8a;
sfr     TL1     =   0x8b;
sfr     TH0     =   0x8c;
sfr     TH1     =   0x8d;
sfr     AUXR    =   0x8e;
sbit    T0x12   =   AUXR^7.
sbit    T1x12   =   AUXR^6.
sbit    S1M0x6  =   AUXR^5.
sbit    T2R     =   AUXR^4.
sbit    T2_CT   =   AUXR^3.
sbit    T2x12   =   AUXR^2.
sbit    EXTRAM  =   AUXR^1.
sbit    S1BRT   =   AUXR^0.
sfr     INTCLKO =   0x8f;
sbit    EX4     =   INTCLKO^6.
sbit    EX3     =   INTCLKO^5.
sbit    EX2     =   INTCLKO^4.
sbit    T2CLKO  =   INTCLKO^2.
sbit    T1CLKO  =   INTCLKO^1.
sbit    T0CLKO  =   INTCLKO^0.
sfr     P1      =   0x90;
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.
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     AUXR2   =   0x97;
```

```
sbit    CANSEL    =    auxr2^3;
sbit    CAN2EN    =    auxr2^2;
sbit    CANEN     =    auxr2^1.
sbit    LINEN     =    AUXR2^0.
sfr     SCON      =    0x98;
sbit    SM0       =    SCON^7.
sbit    SM1       =    SCON^6.
sbit    SM2       =    SCON^5.
sbit    REN       =    SCON^4.
sbit    TB8       =    SCON^3.
sbit    RB8       =    SCON^2.
sbit    TI        =    SCON^1.
sbit    RI        =    SCON^0.
sfr     SBUF      =    0x99;
sfr     S2CON     =    0x9a;
sbit    S2SM0     =    S2CON^7.
sbit    S2SM1     =    S2CON^6.
sbit    S2SM2     =    S2CON^5.
sbit    S2REN     =    S2CON^4.
sbit    S2TB8     =    S2CON^3.
sbit    S2RB8     =    S2CON^2.
sbit    S2TI      =    S2CON^1.
sbit    S2RI      =    S2CON^0.
sfr     S2BUF     =    0x9b;
sfr     IRCBAND   =    0x9d;
sbit    USBCKS    =    IRCBAND^7.
sbit    USBCKS2   =    IRCBAND^6.
sbit    HIRCSEL1  =    IRCBAND^1.
sbit    HIRCSEL0  =    IRCBAND^0.
sfr     LIRTRIM   =    0x9e;
sfr     IRTRIM    =    0x9f;
sfr     P2        =    0xa0;
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.
sfr     BUS_SPEED =    0xa1.
sfr     P_SW1     =    0xa2.
sbit    S1_S1     =    P_SW1^7.
sbit    S1_S0     =    P_SW1^6.
sbit    CAN_S1    =    P_SW1^5.
sbit    CAN_S0    =    P_SW1^4.
```

```
sbit    SPI_S1  =   P_SW1^3.
sbit    SPI_S0  =   P_SW1^2.
sbit    LIN_S1  =   P_SW1^1.
sbit    LIN_S0  =   P_SW1^0.
sfr     V33TRIM =   0xa3.
sfr     BGTRIM  =   0xa5.
sfr     VRTRIM  =   0xa6.
sfr     IE      =   0xa8;
sbit    EA      =   IE^7.
sbit    ELVD    =   IE^6.
sbit    EADC    =   IE^5.
sbit    ES      =   IE^4.
sbit    ET1     =   IE^3.
sbit    EX1     =   IE^2.
sbit    ET0     =   IE^1.
sbit    EX0     =   IE^0.
sfr     SADDR   =   0xa9.
sfr     WKTCL   =   0xaa.
sfr     WKTCH   =   0xab.
sfr     S3CON   =   0xac.
sbit    S3SM0   =   S3CON^7.
sbit    S3ST3   =   S3CON^6.
sbit    S3SM2   =   S3CON^5.
sbit    S3REN   =   S3CON^4.
sbit    S3TB8   =   S3CON^3.
sbit    S3RB8   =   S3CON^2.
sbit    S3TI    =   S3CON^1.
sbit    S3RI    =   S3CON^0.
sfr     S3BUF   =   0xad.
sfr     TA      =   0xae.
sfr     IE2     =   0xaf.
sbit    EUSB    =   IE2^7.
sbit    ET4     =   IE2^6.
sbit    ET3     =   IE2^5.
sbit    ES4     =   IE2^4.
sbit    ES3     =   IE2^3.
sbit    ET2     =   IE2^2.
sbit    ESPI    =   IE2^1.
sbit    ES2     =   IE2^0.
sfr     P3      =   0xb0;
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.
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3.
sfr     P4M0    =   0xb4;
sfr     IP2     =   0xb5;
sbit    PUSB    =   IP2^7.
sbit    PI2C    =   IP2^6.
sbit    PCMP    =   IP2^5.
sbit    PX4     =   IP2^4.
sbit    PPWMB   =   IP2^3.
sbit    PPWMA   =   IP2^2.
sbit    PSPI    =   IP2^1.
sbit    PS2     =   IP2^0.
sfr     IP2H    =   0xb6.
sbit    PUSBH   =   IP2H^7.
sbit    PI2CH   =   IP2H^6.
sbit    PCMPH   =   IP2H^5.
sbit    PX4H    =   IP2H^4.
sbit    PPWMBH  =   IP2H^3.
sbit    PPWMAH  =   IP2H^2.
sbit    PSPIH   =   IP2H^1.
sbit    PS2H    =   IP2H^0.
sfr     IPH     =   0xb7;
sbit    PLVDH   =   IPH^6.
sbit    PADCH   =   IPH^5.
sbit    PSH     =   IPH^4.
sbit    PT1H    =   IPH^3.
sbit    PX1H    =   IPH^2.
sbit    PT0H    =   IPH^1.
sbit    PX0H    =   IPH^0.
sfr     IP      =   0xb8;
sbit    PLVD    =   IP^6.
sbit    PADC    =   IP^5.
sbit    PS      =   IP^4.
sbit    PT1     =   IP^3.
sbit    PX1     =   IP^2.
sbit    PT0     =   IP^1.
sbit    PX0     =   IP^0.
sfr     SADEN   =   0xb9;
sfr     P_SW2   =   0xba.
sbit    EAXFR   =   P_SW2^7.
sbit    I2C_S1  =   P_SW2^5.
sbit    I2C_S0  =   P_SW2^4.
sbit    CMPO_S  =   P_SW2^3.
```

```
sbit    S4_S        =   P_SW2^2.
sbit    S3_S        =   P_SW2^1.
sbit    S2_S        =   P_SW2^0.
sfr     P_SW3       =   0xbb.
sbit    I2S_S1      =   P_SW3^7.
sbit    I2S_S0      =   P_SW3^6.
sbit    S2SPI_S1    =   P_SW3^5.
sbit    S2SPI_S0    =   P_SW3^4.
sbit    S1SPI_S1    =   P_SW3^3.
sbit    S1SPI_S0    =   P_SW3^2.
sbit    CAN2_S1     =   P_SW3^1.
sbit    CAN2_S0     =   P_SW3^0.
sfr     ADC_CONTR       = 0xbc.
sbit    ADC_POWER       = ADC_CONTR^7.
sbit    ADC_START =     ADC_CONTR^6.
sbit    ADC_FLAG =      ADC_CONTR^5.
sbit    ADC_EPWMT       = ADC_CONTR^4.
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL =      0xbe;
sfr     P4          =   0xc0.
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.
sfr     WDT_CONTR       = 0xc1.
sbit    WDT_FLAG =      WDT_CONTR^7.
sbit    EN_WDT          WDT_CONTR^5.
        =EN_WDT
sbit    CLR_WDT     =   WDT_CONTR^4.
sbit    IDL_WDT     =   WDT_CONTR^3.
sfr     IAP_DATA    =   0xc2.
sfr     IAP_ADDRH =     0xc3.
sfr     IAP_ADDRL =     0xc4.
sfr     IAP_CMD     =   0xc5.
sfr     IAP_TRIG    =   0xc6.
sfr     IAP_CONTR =     0xc7.
sbit    IAPEN       =   IAP_CONTR^7.
sbit    SWBS        =   IAP_CONTR^6.
sbit    SWRST       =   IAP_CONTR^5.
sbit    CMD_FAIL    =   IAP_CONTR^4.
sfr     P5          =   0xc8.
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.
sfr     P5M1    =   0xc9.
sfr     P5M0    =   0xca.
sfr     P6M1    =   0xcb.
sfr     P6M0    =   0xcc;
sfr     SPSTAT  =   0xcd.
sbit    SPIF    =   SPSTAT^7.
sbit    WCOL    =   SPSTAT^6.
sfr     SPCTL   =   0xce;
sbit    SSIG    =   SPCTL^7.
sbit    SPEN    =   SPCTL^6.
sbit    DORD    =   SPCTL^5.
sbit    MSTR    =   SPCTL^4.
sbit    CPOL    =   SPCTL^3.
sbit    CPHA    =   SPCTL^2.
sbit    SPR1    =   SPCTL^1.
sbit    SPR0    =   SPCTL^0.
sfr     SPDAT   =   0xcf.
sfr     PSW     =   0xd0;
sbit    CY      =   PSW^7.
sbit    AC      =   PSW^6.
sbit    F0      =   PSW^5.
sbit    RS1     =   PSW^4.
sbit    RS0     =   PSW^3.
sbit    OV      =   PSW^2.
sbit    P       =   PSW^0.
sfr     PSW1    =   0xd1;
sfr     T4H     =   0xd2.
sfr     T4L     =   0xd3.
sfr     T3H     =   0xd4;
sfr     T3L     =   0xd5.
sfr     T2H     =   0xd6.
sfr     T2L     =   0xd7.
sfr     USBCLK  =   0xdc;
sfr     T4T3M   =   0xdd.
sbit    T4R     =   T4T3M^7.
sbit    T4_CT   =   T4T3M^6.
sbit    T4x12   =   T4T3M^5.
sbit    T4CLKO  =   T4T3M^4.
sbit    T3R     =   T4T3M^3.
sbit    T3_CT   =   T4T3M^2.
```

```
sbit    t3x12     =    t4t3m^1;
sbit    t3clko    =    t4t3m^0.
sfr     adccfg    =    0xde.
sbit    RESFMT    =    ADCCFG^5.
sfr     IP3       =    0xdf.
sbit    PI2S      =    IP3^3.
sbit    PRTC      =    IP3^2.
sbit    PS4       =    IP3^1.
sbit    PS3       =    IP3^0.
sfr     ACC       =    0xe0;
sfr     P7M1      =    0xe1;
sfr     P7M0      =    0xe2.
sfr     DPS       =    0xe3.
sfr     CMPCR1    =    0xe6.
sbit    CMPEN     =    CMPCR1^7.
sbit    CMPIF     =    CMPCR1^6.
sbit    PIE       =    CMPCR1^5.
sbit    NIE       =    CMPCR1^4.
sbit    CMPOE     =    CMPCR1^1.
sbit    CMPRES    =    CMPCR1^0.
sfr     CMPCR2    =    0xe7;
sbit    INVCMPO   =    CMPCR2^7.
sbit    DISFLT    =    CMPCR2^6.
sfr     P6        =    0xe8;
sbit    P60       =    P6^0.
sbit    P61       =    P6^1.
sbit    P62       =    P6^2.
sbit    P63       =    P6^3.
sbit    P64       =    P6^4.
sbit    P65       =    P6^5.
sbit    P66       =    P6^6.
sbit    P67       =    P6^7.
sfr     WTST      =    0xe9.
sfr     CKCON     =    0xea.
sfr     MXAX      =    0xeb.
sfr     USBDAT    =    0xec;
sfr     DMAIR     =    0xed.
sfr     IP3H      =    0xee.
sbit    PI2SH     =    IP3H^3.
sbit    PRTCH     =    IP3H^2.
sbit    PS4H      =    IP3H^1.
sbit    PS3H      =    IP3H^0.
sfr     AUXINTIF  =    0xef;
sbit    INT4IF    =    AUXINTIF^6.
sbit    INT3IF    =    AUXINTIF^5.
sbit    INT2IF    =    AUXINTIF^4.
```

```
sbit    T4IF     =    auxintif^2;
sbit    T3IF     =    auxintif^1.
sbit    T2IF     =    AUXINTIF^0.
sfr     B        =    0xf0;
sfr     CANICR   =    0xf1;
sbit    PCAN2H   =    CANICR^7.
sbit    CAN2IF   =    CANICR^6.
sbit    CAN2IE   =    CANICR^5.
sbit    PCAN2L   =    CANICR^4.
sbit    PCANH    =    CANICR^3.
sbit    CANIF    =    CANICR^2.
sbit    CANIE    =    CANICR^1.
sbit    PCANL    =    CANICR^0.
sfr     USBCON   =    0xf4;
sbit    ENUSB    =    USBCON^7.
sbit    USBRST   =    USBCON^6.
sbit    PS2M     =    USBCON^5.
sbit    PUEN     =    USBCON^4.
sbit    PDEN     =    USBCON^3.
sbit    DFREC    =    USBCON^2.
sbit    DP       =    USBCON^1.
sbit    DM       =    USBCON^0.
sfr     IAP_TPS  =    0xf5.
sfr     IAP_ADDRE =   0xf6.
sfr     ICHECR   =    0xf7;
sfr     P7       =    0xf8;
sbit    P70      =    P7^0.
sbit    P71      =    P7^1.
sbit    P72      =    P7^2.
sbit    P73      =    P7^3.
sbit    P74      =    P7^4.
sbit    P75      =    P7^5.
sbit    P76      =    P7^6.
sbit    P77      =    P7^7.
sfr     LINICR   =    0xf9.
sbit    PLINH    =    LINICR^3.
sbit    LINIF    =    LINICR^2.
sbit    LINIE    =    LINICR^1.
sbit    PLINL    =    LINICR^0.
sfr     LINAR    =    0xfa.
sfr     LINDR    =    0xfb;
sfr     USBADR   =    0xfc.
sfr     S4CON    =    0xfd;
sbit    S4SM0    =    S4CON^7.
sbit    S4ST4    =    S4CON^6.
sbit    S4SM2    =    S4CON^5.
```

```
sbit    S4REN   =   S4CON^4.
sbit    S4TB8   =   S4CON^3.
sbit    S4RB8   =   S4CON^2.
sbit    S4TI    =   S4CON^1.
sbit    S4RI    =   S4CON^0.
sfr     S4BUF   =   0xfe;
sfr     RSTCFG  =   0xff;
sbit    ENLVR   =   RSTCFG^6.
sbit    P54RST  =   RSTCFG^4.

// The following special function registers are located in the extended RAM area
//To access these registers, you need to set EAXFR to 1 before you can read/write them normally.
    //EAXFR = 1;
//or
    //P_SW2 |= 0x80.

////////////////////////////////////////////////
//7E:FF00H-7E:FFFFH
////////////////////////////////////////////////



////////////////////////////////////////////////
//7E:FE00H-7E:FEFFH
////////////////////////////////////////////////

#define CLKSEL          (*(unsigned char volatile far *)0x7efe00)
#define CLKDIV          (*(unsigned char volatile far *)0x7efe01)
#define HIRCCR          (*(unsigned char volatile far *)0x7efe02)
#define XOSCCR          (*(unsigned char volatile far *)0x7efe03)
#define IRC32KCR        (*(unsigned char volatile far *)0x7efe04)
#define MCLKOCR         (*(unsigned char volatile far *)0x7efe05)
#define IRCDB           (*(unsigned char volatile far *)0x7efe06)
#define IRC48MCR        (*(unsigned char volatile far *)0x7efe07)
#define X32KCR          (*(unsigned char volatile far *)0x7efe08)
#define IRC48ATRIM      (*(unsigned char volatile far *)0x7efe09)
#define IRC48BTRIM      (*(unsigned char volatile far *)0x7efe0a)
#define HSCLKDIV        (*(unsigned char volatile far *)0x7efe0b)

#define P0PU            (*(unsigned char volatile far *)0x7efe10)
#define P1PU            (*(unsigned char volatile far *)0x7efe11)
#define P2PU            (*(unsigned char volatile far *)0x7efe12)
#define P3PU            (*(unsigned char volatile far *)0x7efe13)
#define P4PU            (*(unsigned char volatile far *)0x7efe14)
#define P5PU            (*(unsigned char volatile far *)0x7efe15)
#define P6PU            (*(unsigned char volatile far *)0x7efe16)
```

```
#define  P7PU          (*(unsigned char volatile far *)0x7efe17)
#define  P0NCS         (*(unsigned char volatile far *)0x7efe18)
#define  P1NCS         (*(unsigned char volatile far *)0x7efe19)
#define  P2NCS         (*(unsigned char volatile far *)0x7efe1a)
#define  P3NCS         (*(unsigned char volatile far *)0x7efe1b)
#define  P4NCS         (*(unsigned char volatile far *)0x7efe1c)
#define  P5NCS         (*(unsigned char volatile far *)0x7efe1d)
#define  P6NCS         (*(unsigned char volatile far *)0x7efe1e)
#define  P7NCS         (*(unsigned char volatile far *)0x7efe1f)
#define  P0SR          (*(unsigned char volatile far *)0x7efe20)
#define  P1SR          (*(unsigned char volatile far *)0x7efe21)
#define  P2SR          (*(unsigned char volatile far *)0x7efe22)
#define  P3SR          (*(unsigned char volatile far *)0x7efe23)
#define  P4SR          (*(unsigned char volatile far *)0x7efe24)
#define  P5SR          (*(unsigned char volatile far *)0x7efe25)
#define  P6SR          (*(unsigned char volatile far *)0x7efe26)
#define  P7SR          (*(unsigned char volatile far *)0x7efe27)
#define  P0DR          (*(unsigned char volatile far *)0x7efe28)
#define  P1DR          (*(unsigned char volatile far *)0x7efe29)
#define  P2DR          (*(unsigned char volatile far *)0x7efe2a)
#define  P3DR          (*(unsigned char volatile far *)0x7efe2b)
#define  P4DR          (*(unsigned char volatile far *)0x7efe2c)
#define  P5DR          (*(unsigned char volatile far *)0x7efe2d)
#define  P6DR          (*(unsigned char volatile far *)0x7efe2e)
#define  P7DR          (*(unsigned char volatile far *)0x7efe2f)
#define  P0IE          (*(unsigned char volatile far *)0x7efe30)
#define  P1IE          (*(unsigned char volatile far *)0x7efe31)
#define  P2IE          (*(unsigned char volatile far *)0x7efe32)
#define  P3IE          (*(unsigned char volatile far *)0x7efe33)
#define  P4IE          (*(unsigned char volatile far *)0x7efe34)
#define  P5IE          (*(unsigned char volatile far *)0x7efe35)
#define  P6IE          (*(unsigned char volatile far *)0x7efe36)
#define  P7IE          (*(unsigned char volatile far *)0x7efe37)

#define  LCMIFCFG      (*(unsigned char volatile far *)0x7efe50)
#define  LCMIFCFG2     (*(unsigned char volatile far *)0x7efe51)
#define  LCMIFCR       (*(unsigned char volatile far *)0x7efe52)
#define  LCMIFSTA      (*(unsigned char volatile far *)0x7efe53)
#define  LCMIFDATL     (*(unsigned char volatile far *)0x7efe54)
#define  LCMIFDATH     (*(unsigned char volatile far *)0x7efe55)

#define  RTCCR         (*(unsigned char volatile far *)0x7efe60)
#define  RTCCFG        (*(unsigned char volatile far *)0x7efe61)
#define  RTCIEN        (*(unsigned char volatile far *)0x7efe62)
#define  RTCIF         (*(unsigned char volatile far *)0x7efe63)
#define  ALAHOUR       (*(unsigned char volatile far *)0x7efe64)
```

```
#define  ALAMIN                  (*(unsigned char volatile far *)0x7efe65)
#define  ALASEC                  (*(unsigned char volatile far *)0x7efe66)
#define  ALASSEC                 (*(unsigned char volatile far *)0x7efe67)
#define  INIYEAR                 (*(unsigned char volatile far *)0x7efe68)
#define  INIMONTH                (*(unsigned char volatile far *)0x7efe69)
#define  INIDAY                  (*(unsigned char volatile far *)0x7efe6a)
#define  INIHOUR                 (*(unsigned char volatile far *)0x7efe6b)
#define  INIMIN                  (*(unsigned char volatile far *)0x7efe6c)
#define  INISEC                  (*(unsigned char volatile far *)0x7efe6d)
#define  INISSEC                 (*(unsigned char volatile far *)0x7efe6e)
#define  YEAR                    (*(unsigned char volatile far *)0x7efe70)
#define  MONTH                   (*(unsigned char volatile far *)0x7efe71)
#define  DAY                     (*(unsigned char volatile far *)0x7efe72)
#define  HOUR                    (*(unsigned char volatile far *)0x7efe73)
#define  MIN                     (*(unsigned char volatile far *)0x7efe74)
#define  SEC                     (*(unsigned char volatile far *)0x7efe75)
#define  SSEC                    (*(unsigned char volatile far *)0x7efe76)

#define  I2CCFG                  (*(unsigned char volatile far *)0x7efe80)
#define  ENI2C                   0x80
#define  I2CMASTER               0x40
#define  I2CSLAVE                0x00
#define  I2CMSCR                 (*(unsigned char volatile far *)0x7efe81)
#define  EMSI                    0x80
#define  MS_IDLE                 0x00
#define  MS_START                0x01
#define  MS_SENDDAT              0x02
#define  MS_RECVACK              0x03
#define  MS_RECVDAT              0x04
#define  MS_SENDACK              0x05
#define  MS_STOP                 0x06
#define  MS_START_SENDDAT_RECVACK    0x09
#define  MS_SENDDAT_RECVACK          0x0a
#define  MS_RECVDAT_SENDACK          0x0b
#define  MS_RECVDAT_SENDNAK          0x0c
#define  I2CMSST                 (*(unsigned char volatile far *)0x7efe82)
#define  MSBUSY                  0x80
#define  MSIF                    0x40
#define  MSACKI                  0x02
#define  MSACKO                  0x01
#define  I2CSLCR                 (*(unsigned char volatile far *)0x7efe83)
#define  ESTAI                   0x40
#define  ERXI                    0x20
#define  ETXI                    0x10
#define  ESTOI                   0x08
#define  SLRST                   0x01
```

```
#define  I2CSLST        (*(unsigned char volatile far *)0x7efe84)
#define  SLBUSY         0x80
#define  STAIF          0x40
#define  RXIF           0x20
#define  TXIF           0x10
#define  STOIF          0x08
#define  TXING          0x04
#define  SLACKI         0x02
#define  SLACKO         0x01
#define  I2CSLADR       (*(unsigned char volatile far *)0x7efe85)
#define  I2CTXD         (*(unsigned char volatile far *)0x7efe86)
#define  I2CRXD         (*(unsigned char volatile far *)0x7efe87)
#define  I2CMSAUX       (*(unsigned char volatile far *)0x7efe88)
#define  WDTA           0x01

#define  SPFUNC         (*(unsigned char volatile far *)0x7efe98)
#define  RSTFLAG        (*(unsigned char volatile far *)0x7efe99)
#define  RSTCR0         (*(unsigned char volatile far *)0x7efe9a)
#define  RSTCR1         (*(unsigned char volatile far *)0x7efe9b)
#define  RSTCR2         (*(unsigned char volatile far *)0x7efe9c)
#define  RSTCR3         (*(unsigned char volatile far *)0x7efe9d)
#define  RSTCR4         (*(unsigned char volatile far *)0x7efe9e)
#define  RSTCR5         (*(unsigned char volatile far *)0x7efe9f)

#define  TM0PS          (*(unsigned char volatile far *)0x7efea0)
#define  TM1PS          (*(unsigned char volatile far *)0x7efea1)
#define  TM2PS          (*(unsigned char volatile far *)0x7efea2)
#define  TM3PS          (*(unsigned char volatile far *)0x7efea3)
#define  TM4PS          (*(unsigned char volatile far *)0x7efea4)
#define  ADCTIM         (*(unsigned char volatile far *)0x7efea8)
#define  T3T4PS         (*(unsigned char volatile far *)0x7efeac)
#define  ADCEXCFG       (*(unsigned char volatile far *)0x7efead)
#define  CMPEXCFG       (*(unsigned char volatile far *)0x7efeae)

#define  PWMA_ETRPS     (*(unsigned char volatile far *)0x7efeb0)
#define  PWMA_ENO       (*(unsigned char volatile far *)0x7efeb1)
#define  PWMA_PS        (*(unsigned char volatile far *)0x7efeb2)
#define  PWMA_IOAUX     (*(unsigned char volatile far *)0x7efeb3)
#define  PWMB_ETRPS     (*(unsigned char volatile far *)0x7efeb4)
#define  PWMB_ENO       (*(unsigned char volatile far *)0x7efeb5)
#define  PWMB_PS        (*(unsigned char volatile far *)0x7efeb6)
#define  PWMB_IOAUX     (*(unsigned char volatile far *)0x7efeb7)
#define  CANAR          (*(unsigned char volatile far *)0x7efebb)
#define  CANDR          (*(unsigned char volatile far *)0x7efebc)
#define  PWMA_CR1       (*(unsigned char volatile far *)0x7efec0)
#define  PWMA_CR2       (*(unsigned char volatile far *)0x7efec1)
```

```
#define  PWMA_SMCR          (*(unsigned char volatile far *)0x7efec2)
#define  PWMA_ETR           (*(unsigned char volatile far *)0x7efec3)
#define  PWMA_IER           (*(unsigned char volatile far *)0x7efec4)
#define  PWMA_SR1           (*(unsigned char volatile far *)0x7efec5)
#define  PWMA_SR2           (*(unsigned char volatile far *)0x7efec6)
#define  PWMA_EGR           (*(unsigned char volatile far *)0x7efec7)
#define  PWMA_CCMR1         (*(unsigned char volatile far *)0x7efec8)
#define  PWMA_CCMR2         (*(unsigned char volatile far *)0x7efec9)
#define  PWMA_CCMR3         (*(unsigned char volatile far *)0x7efeca)
#define  PWMA_CCMR4         (*(unsigned char volatile far *)0x7efecb)
#define  PWMA_CCER1         (*(unsigned char volatile far *)0x7efecc)
#define  PWMA_CCER2         (*(unsigned char volatile far *)0x7efecd)
#define  PWMA_CNTRH         (*(unsigned char volatile far *)0x7efece)
#define  PWMA_CNTRL         (*(unsigned char volatile far *)0x7efecf)
#define  PWMA_PSCRH         (*(unsigned char volatile far *)0x7efed0)
#define  PWMA_PSCRL         (*(unsigned char volatile far *)0x7efed1)
#define  PWMA_ARRH          (*(unsigned char volatile far *)0x7efed2)
#define  PWMA_ARRL          (*(unsigned char volatile far *)0x7efed3)
#define  PWMA_RCR           (*(unsigned char volatile far *)0x7efed4)
#define  PWMA_CCR1H         (*(unsigned char volatile far *)0x7efed5)
#define  PWMA_CCR1L         (*(unsigned char volatile far *)0x7efed6)
#define  PWMA_CCR2H         (*(unsigned char volatile far *)0x7efed7)
#define  PWMA_CCR2L         (*(unsigned char volatile far *)0x7efed8)
#define  PWMA_CCR3H         (*(unsigned char volatile far *)0x7efed9)
#define  PWMA_CCR3L         (*(unsigned char volatile far *)0x7efeda)
#define  PWMA_CCR4H         (*(unsigned char volatile far *)0x7efedb)
#define  PWMA_CCR4L         (*(unsigned char volatile far *)0x7efedc)
#define  PWMA_BKR           (*(unsigned char volatile far *)0x7efedd)
#define  PWMA_DTR           (*(unsigned char volatile far *)0x7efede)
#define  PWMA_OISR          (*(unsigned char volatile far *)0x7efedf)
#define  PWMB_CR1           (*(unsigned char volatile far *)0x7efee0)
#define  PWMB_CR2           (*(unsigned char volatile far *)0x7efee1)
#define  PWMB_SMCR          (*(unsigned char volatile far *)0x7efee2)
#define  PWMB_ETR           (*(unsigned char volatile far *)0x7efee3)
#define  PWMB_IER           (*(unsigned char volatile far *)0x7efee4)
#define  PWMB_SR1           (*(unsigned char volatile far *)0x7efee5)
#define  PWMB_SR2           (*(unsigned char volatile far *)0x7efee6)
#define  PWMB_EGR           (*(unsigned char volatile far *)0x7efee7)
#define  PWMB_CCMR1         (*(unsigned char volatile far *)0x7efee8)
#define  PWMB_CCMR2         (*(unsigned char volatile far *)0x7efee9)
#define  PWMB_CCMR3         (*(unsigned char volatile far *)0x7efeea)
#define  PWMB_CCMR4         (*(unsigned char volatile far *)0x7efeeb)
#define  PWMB_CCER1         (*(unsigned char volatile far *)0x7efeec)
#define  PWMB_CCER2         (*(unsigned char volatile far *)0x7efeed)
#define  PWMB_CNTRH         (*(unsigned char volatile far *)0x7efeee)
#define  PWMB_CNTRL         (*(unsigned char volatile far *)0x7efeef)
```

```
#define   PWMB_PSCRH          (*(unsigned char volatile far *)0x7efef0)
#define   PWMB_PSCRL          (*(unsigned char volatile far *)0x7efef1)
#define   PWMB_ARRH           (*(unsigned char volatile far *)0x7efef2)
#define   PWMB_ARRL           (*(unsigned char volatile far *)0x7efef3)
#define   PWMB_RCR            (*(unsigned char volatile far *)0x7efef4)
#define   PWMB_CCR5H          (*(unsigned char volatile far *)0x7efef5)
#define   PWMB_CCR5L          (*(unsigned char volatile far *)0x7efef6)
#define   PWMB_CCR6H          (*(unsigned char volatile far *)0x7efef7)
#define   PWMB_CCR6L          (*(unsigned char volatile far *)0x7efef8)
#define   PWMB_CCR7H          (*(unsigned char volatile far *)0x7efef9)
#define   PWMB_CCR7L          (*(unsigned char volatile far *)0x7efefa)
#define   PWMB_CCR8H          (*(unsigned char volatile far *)0x7efefb)
#define   PWMB_CCR8L          (*(unsigned char volatile far *)0x7efefc)
#define   PWMB_BKR            (*(unsigned char volatile far *)0x7efefd)
#define   PWMB_DTR            (*(unsigned char volatile far *)0x7efefe)
#define   PWMB_OISR           (*(unsigned char volatile far *)0x7efeff)

////////////////////////////////////////////////
//7E:FD00H-7E:FDFFH
////////////////////////////////////////////////
#define   PWM2_OISR           (*(unsigned char volatile far *)0x7efeff)

#define   P0INTE              (*(unsigned char volatile far *)0x7efd00)
#define   P1INTE              (*(unsigned char volatile far *)0x7efd01)
#define   P2INTE              (*(unsigned char volatile far *)0x7efd02)
#define   P3INTE              (*(unsigned char volatile far *)0x7efd03)
#define   P4INTE              (*(unsigned char volatile far *)0x7efd04)
#define   P5INTE              (*(unsigned char volatile far *)0x7efd05)
#define   P6INTE              (*(unsigned char volatile far *)0x7efd06)
#define   P7INTE              (*(unsigned char volatile far *)0x7efd07)
#define   P0INTF              (*(unsigned char volatile far *)0x7efd10)
#define   P1INTF              (*(unsigned char volatile far *)0x7efd11)
#define   P2INTF              (*(unsigned char volatile far *)0x7efd12)
#define   P3INTF              (*(unsigned char volatile far *)0x7efd13)
#define   P4INTF              (*(unsigned char volatile far *)0x7efd14)
#define   P5INTF              (*(unsigned char volatile far *)0x7efd15)
#define   P6INTF              (*(unsigned char volatile far *)0x7efd16)
#define   P7INTF              (*(unsigned char volatile far *)0x7efd17)
#define   P0IM0               (*(unsigned char volatile far *)0x7efd20)
#define   P1IM0               (*(unsigned char volatile far *)0x7efd21)
#define   P2IM0               (*(unsigned char volatile far *)0x7efd22)
#define   P3IM0               (*(unsigned char volatile far *)0x7efd23)
#define   P4IM0               (*(unsigned char volatile far *)0x7efd24)
#define   P5IM0               (*(unsigned char volatile far *)0x7efd25)
#define   P6IM0               (*(unsigned char volatile far *)0x7efd26)
#define   P7IM0               (*(unsigned char volatile far *)0x7efd27)
```

```
#define  P0IM1          (*(unsigned char volatile far *)0x7efd30)
#define  P1IM1          (*(unsigned char volatile far *)0x7efd31)
#define  P2IM1          (*(unsigned char volatile far *)0x7efd32)
#define  P3IM1          (*(unsigned char volatile far *)0x7efd33)
#define  P4IM1          (*(unsigned char volatile far *)0x7efd34)
#define  P5IM1          (*(unsigned char volatile far *)0x7efd35)
#define  P6IM1          (*(unsigned char volatile far *)0x7efd36)
#define  P7IM1          (*(unsigned char volatile far *)0x7efd37)
#define  P0WKUE         (*(unsigned char volatile far *)0x7efd40)
#define  P1WKUE         (*(unsigned char volatile far *)0x7efd41)
#define  P2WKUE         (*(unsigned char volatile far *)0x7efd42)
#define  P3WKUE         (*(unsigned char volatile far *)0x7efd43)
#define  P4WKUE         (*(unsigned char volatile far *)0x7efd44)
#define  P5WKUE         (*(unsigned char volatile far *)0x7efd45)
#define  P6WKUE         (*(unsigned char volatile far *)0x7efd46)
#define  P7WKUE         (*(unsigned char volatile far *)0x7efd47)

#define  PIN_IP         (*(unsigned char volatile far *)0x7efd60)
#define  PIN_IPH        (*(unsigned char volatile far *)0x7efd61)

#define  S2CFG          (*(unsigned char volatile far *)0x7efdb4)
#define  S2ADDR         (*(unsigned char volatile far *)0x7efdb5)
#define  S2ADEN         (*(unsigned char volatile far *)0x7efdb6)
#define  USARTCR1       (*(unsigned char volatile far *)0x7efdc0)
#define  USARTCR2       (*(unsigned char volatile far *)0x7efdc1)
#define  USARTCR3       (*(unsigned char volatile far *)0x7efdc2)
#define  USARTCR4       (*(unsigned char volatile far *)0x7efdc3)
#define  USARTCR5       (*(unsigned char volatile far *)0x7efdc4)
#define  USARTGTR       (*(unsigned char volatile far *)0x7efdc5)
#define  USARTBRH       (*(unsigned char volatile far *)0x7efdc6)
#define  USARTBRL       (*(unsigned char volatile far *)0x7efdc7)
#define  USART2CR1      (*(unsigned char volatile far *)0x7efdc8)
#define  USART2CR2      (*(unsigned char volatile far *)0x7efdc9)
#define  USART2CR3      (*(unsigned char volatile far *)0x7efdca)
#define  USART2CR4      (*(unsigned char volatile far *)0x7efdcb)
#define  USART2CR5      (*(unsigned char volatile far *)0x7efdcc)
#define  USART2GTR      (*(unsigned char volatile far *)0x7efdcd)
#define  USART2BRH      (*(unsigned char volatile far *)0x7efdce)
#define  USART2BRL      (*(unsigned char volatile far *)0x7efdcf)

#define  CHIPID          ( (unsigned char volatile far *)0x7efde0)

#define  CHIPID0        (*(unsigned char volatile far *)0x7efde0)
#define  CHIPID1        (*(unsigned char volatile far *)0x7efde1)
#define  CHIPID2        (*(unsigned char volatile far *)0x7efde2)
#define  CHIPID3        (*(unsigned char volatile far *)0x7efde3)
```

```
#define   CHIPID4          (*(unsigned char volatile far *)0x7efde4)
#define   CHIPID5          (*(unsigned char volatile far *)0x7efde5)
#define   CHIPID6          (*(unsigned char volatile far *)0x7efde6)
#define   CHIPID7          (*(unsigned char volatile far *)0x7efde7)
#define   CHIPID8          (*(unsigned char volatile far *)0x7efde8)
#define   CHIPID9          (*(unsigned char volatile far *)0x7efde9)
#define   CHIPID10         (*(unsigned char volatile far *)0x7efdea)
#define   CHIPID11         (*(unsigned char volatile far *)0x7efdeb)
#define   CHIPID12         (*(unsigned char volatile far *)0x7efdec)
#define   CHIPID13         (*(unsigned char volatile far *)0x7efded)
#define   CHIPID14         (*(unsigned char volatile far *)0x7efdee)
#define   CHIPID15         (*(unsigned char volatile far *)0x7efdef)
#define   CHIPID16         (*(unsigned char volatile far *)0x7efdf0)
#define   CHIPID17         (*(unsigned char volatile far *)0x7efdf1)
#define   CHIPID18         (*(unsigned char volatile far *)0x7efdf2)
#define   CHIPID19         (*(unsigned char volatile far *)0x7efdf3)
#define   CHIPID20         (*(unsigned char volatile far *)0x7efdf4)
#define   CHIPID21         (*(unsigned char volatile far *)0x7efdf5)
#define   CHIPID22         (*(unsigned char volatile far *)0x7efdf6)
#define   CHIPID23         (*(unsigned char volatile far *)0x7efdf7)
#define   CHIPID24         (*(unsigned char volatile far *)0x7efdf8)
#define   CHIPID25         (*(unsigned char volatile far *)0x7efdf9)
#define   CHIPID26         (*(unsigned char volatile far *)0x7efdfa)
#define   CHIPID27         (*(unsigned char volatile far *)0x7efdfb)
#define   CHIPID28         (*(unsigned char volatile far *)0x7efdfc)
#define   CHIPID29         (*(unsigned char volatile far *)0x7efdfd)
#define   CHIPID30         (*(unsigned char volatile far *)0x7efdfe)
#define   CHIPID31         (*(unsigned char volatile far *)0x7efdff)

//////////////////////////////////////////////////
//7E:FC00H-7E:FCFFH
//////////////////////////////////////////////////



//////////////////////////////////////////////////
//7E:FB00H-7E:FBFFH
//////////////////////////////////////////////////

#define   hspwma_cfg       (*(unsigned char volatile far
#define   hspwma_adr       *)0x7efbf0) (*(unsigned char volatile
#define   hspwma_dat       far *)0x7efbf1) (*(unsigned char
                           volatile far *)0x7efbf2)

#define   HSPWMB_CFG       (*(unsigned char volatile far *)0x7efbf4)
#define   HSPWMB_ADR       (*(unsigned char volatile far *)0x7efbf5)
```

```
#define   HSPWMB_DAT            (*(unsigned char volatile far *)0x7efbf6)
```

```
#define   HSSPI_CFG             (*(unsigned char volatile far *)0x7efbf8)
#define   HSSPI_CFG2            (*(unsigned char volatile far *)0x7efbf9)
#define   HSSPI_STA             (*(unsigned char volatile far *)0x7efbfa)

//To use the following macro, you need to set EAXFR to 1 first.
//Methods of use.
//        char val.
//
//        eaxfr = 1; read_hspwma(pwma_cr1,        // Enable access to XFR
//        val).                                   //Asynchronous Read
                                                  PWMA Group Registers
//        val |= 0x01;
//        WRITE_HSPWMA(PWMA_CR1, val).            //Asynchronous Write
                                                  PWMA Group Registers

#define   READ_HSPWMA(reg, dat)                 \windshield
{                                               \windshield
          while (HSPWMA_ADR & 0x80).            \windshield
          HSPWMA_ADR = ((char)&(reg)) | 0x80;   \windshield
          while (HSPWMA_ADR & 0x80).            \windshield
          (dat) = HSPWMA_DAT.                   \windshield
}

#define   WRITE_HSPWMA(reg, dat)                \windshield
{                                               \windshield
          while (HSPWMA_ADR & 0x80).            \windshield
          HSPWMA_DAT = (dat).                   \windshield
          HSPWMA_ADR = ((char)&(reg)) & 0x7f;   \windshield
}

#define   READ_HSPWMB(reg, dat)                 \windshield
{         \windshield
          while (HSPWMB_ADR & 0x80).            \windshield
          HSPWMB_ADR = ((char)&(reg)) | 0x80;   \windshield
          while (HSPWMB_ADR & 0x80).            \windshield
          (dat) = HSPWMB_DAT.                   \windshield
}

#define   WRITE_HSPWMB(reg, dat)                \windshield
{                                               \windshield
          while (HSPWMB_ADR & 0x80).            \windshield
          HSPWMB_DAT = (dat).                   \windshield
          HSPWMB_ADR = ((char)&(reg)) & 0x7f;   \windshield
}
```

```
//////////////////////////////////////////////////
//7E:FA00H-7E:FAFFH
```

```
#define   DMA_M2M_CFG           (*(unsigned char volatile far *)0x7efa00)
#define   DMA_M2M_CR            (*(unsigned char volatile far *)0x7efa01)
#define   DMA_M2M_STA           (*(unsigned char volatile far *)0x7efa02)
#define   DMA_M2M_AMT           (*(unsigned char volatile far *)0x7efa03)
#define   DMA_M2M_DONE          (*(unsigned char volatile far *)0x7efa04)
#define   DMA_M2M_TXAH          (*(unsigned char volatile far *)0x7efa05)
#define   DMA_M2M_TXAL          (*(unsigned char volatile far *)0x7efa06)
#define   DMA_M2M_RXAH          (*(unsigned char volatile far *)0x7efa07)
#define   DMA_M2M_RXAL          (*(unsigned char volatile far *)0x7efa08)

#define   DMA_ADC_CFG           (*(unsigned char volatile far *)0x7efa10)
#define   DMA_ADC_CR            (*(unsigned char volatile far *)0x7efa11)
#define   DMA_ADC_STA           (*(unsigned char volatile far *)0x7efa12)
#define   DMA_ADC_RXAH          (*(unsigned char volatile far *)0x7efa17)
#define   DMA_ADC_RXAL          (*(unsigned char volatile far *)0x7efa18)
#define   DMA_ADC_CFG2          (*(unsigned char volatile far *)0x7efa19)
#define   DMA_ADC_CHSW0         (*(unsigned char volatile far *)0x7efa1a)
#define   DMA_ADC_CHSW1         (*(unsigned char volatile far *)0x7efa1b)

#define   DMA_SPI_CFG           (*(unsigned char volatile far *)0x7efa20)
#define   DMA_SPI_CR            (*(unsigned char volatile far *)0x7efa21)
#define   DMA_SPI_STA           (*(unsigned char volatile far *)0x7efa22)
#define   DMA_SPI_AMT           (*(unsigned char volatile far *)0x7efa23)
#define   DMA_SPI_DONE          (*(unsigned char volatile far *)0x7efa24)
#define   DMA_SPI_TXAH          (*(unsigned char volatile far *)0x7efa25)
#define   DMA_SPI_TXAL          (*(unsigned char volatile far *)0x7efa26)
#define   DMA_SPI_RXAH          (*(unsigned char volatile far *)0x7efa27)
#define   DMA_SPI_RXAL          (*(unsigned char volatile far *)0x7efa28)
#define   DMA_SPI_CFG2          (*(unsigned char volatile far *)0x7efa29)

#define   DMA_UR1T_CFG          (*(unsigned char volatile far *)0x7efa30)
#define   DMA_UR1T_CR           (*(unsigned char volatile far *)0x7efa31)
#define   DMA_UR1T_STA          (*(unsigned char volatile far *)0x7efa32)
#define   DMA_UR1T_AMT          (*(unsigned char volatile far *)0x7efa33)
#define   DMA_UR1T_DONE         (*(unsigned char volatile far *)0x7efa34)
#define   DMA_UR1T_TXAH         (*(unsigned char volatile far *)0x7efa35)
#define   DMA_UR1T_TXAL         (*(unsigned char volatile far *)0x7efa36)
#define   DMA_UR1R_CFG          (*(unsigned char volatile far *)0x7efa38)
#define   DMA_UR1R_CR           (*(unsigned char volatile far *)0x7efa39)
#define   DMA_UR1R_STA          (*(unsigned char volatile far *)0x7efa3a)
#define   DMA_UR1R_AMT          (*(unsigned char volatile far *)0x7efa3b)
#define   DMA_UR1R_DONE         (*(unsigned char volatile far *)0x7efa3c)
#define   DMA_UR1R_RXAH         (*(unsigned char volatile far *)0x7efa3d)
#define   DMA_UR1R_RXAL         (*(unsigned char volatile far *)0x7efa3e)
```

```
#define   DMA_UR2T_CFG          (*(unsigned char volatile far *)0x7efa40)
#define   DMA_UR2T_CR           (*(unsigned char volatile far *)0x7efa41)
#define   DMA_UR2T_STA          (*(unsigned char volatile far *)0x7efa42)
#define   DMA_UR2T_AMT          (*(unsigned char volatile far *)0x7efa43)
#define   DMA_UR2T_DONE         (*(unsigned char volatile far *)0x7efa44)
#define   DMA_UR2T_TXAH         (*(unsigned char volatile far *)0x7efa45)
#define   DMA_UR2T_TXAL         (*(unsigned char volatile far *)0x7efa46)
#define   DMA_UR2R_CFG          (*(unsigned char volatile far *)0x7efa48)
#define   DMA_UR2R_CR           (*(unsigned char volatile far *)0x7efa49)
#define   DMA_UR2R_STA          (*(unsigned char volatile far *)0x7efa4a)
#define   DMA_UR2R_AMT          (*(unsigned char volatile far *)0x7efa4b)
#define   DMA_UR2R_DONE         (*(unsigned char volatile far *)0x7efa4c)
#define   DMA_UR2R_RXAH         (*(unsigned char volatile far *)0x7efa4d)
#define   DMA_UR2R_RXAL         (*(unsigned char volatile far *)0x7efa4e)

#define   DMA_UR3T_CFG          (*(unsigned char volatile far *)0x7efa50)
#define   DMA_UR3T_CR           (*(unsigned char volatile far *)0x7efa51)
#define   DMA_UR3T_STA          (*(unsigned char volatile far *)0x7efa52)
#define   DMA_UR3T_AMT          (*(unsigned char volatile far *)0x7efa53)
#define   DMA_UR3T_DONE         (*(unsigned char volatile far *)0x7efa54)
#define   DMA_UR3T_TXAH         (*(unsigned char volatile far *)0x7efa55)
#define   DMA_UR3T_TXAL         (*(unsigned char volatile far *)0x7efa56)
#define   DMA_UR3R_CFG          (*(unsigned char volatile far *)0x7efa58)
#define   DMA_UR3R_CR           (*(unsigned char volatile far *)0x7efa59)
#define   DMA_UR3R_STA          (*(unsigned char volatile far *)0x7efa5a)
#define   DMA_UR3R_AMT          (*(unsigned char volatile far *)0x7efa5b)
#define   DMA_UR3R_DONE         (*(unsigned char volatile far *)0x7efa5c)
#define   DMA_UR3R_RXAH         (*(unsigned char volatile far *)0x7efa5d)
#define   dma_ur3r_rxal         (*(unsigned char volatile far *)0x7efa5e)

#define   DMA_UR4T_CFG          (*(unsigned char volatile far *)0x7efa60)
#define   DMA_UR4T_CR           (*(unsigned char volatile far *)0x7efa61)
#define   DMA_UR4T_STA          (*(unsigned char volatile far *)0x7efa62)
#define   DMA_UR4T_AMT          (*(unsigned char volatile far *)0x7efa63)
#define   DMA_UR4T_DONE         (*(unsigned char volatile far *)0x7efa64)
#define   DMA_UR4T_TXAH         (*(unsigned char volatile far *)0x7efa65)
#define   DMA_UR4T_TXAL         (*(unsigned char volatile far *)0x7efa66)
#define   DMA_UR4R_CFG          (*(unsigned char volatile far *)0x7efa68)
#define   DMA_UR4R_CR           (*(unsigned char volatile far *)0x7efa69)
#define   DMA_UR4R_STA          (*(unsigned char volatile far *)0x7efa6a)
#define   DMA_UR4R_AMT          (*(unsigned char volatile far *)0x7efa6b)
#define   DMA_UR4R_DONE         (*(unsigned char volatile far *)0x7efa6c)
#define   DMA_UR4R_RXAH         (*(unsigned char volatile far *)0x7efa6d)
#define   DMA_UR4R_RXAL         (*(unsigned char volatile far *)0x7efa6e)
```

```
#define   DMA_LCM_CFG           (*(unsigned char volatile far *)0x7efa70)
#define   DMA_LCM_CR            (*(unsigned char volatile far *)0x7efa71)
#define   DMA_LCM_STA           (*(unsigned char volatile far *)0x7efa72)
#define   DMA_LCM_AMT           (*(unsigned char volatile far *)0x7efa73)
#define   DMA_LCM_DONE          (*(unsigned char volatile far *)0x7efa74)
#define   DMA_LCM_TXAH          (*(unsigned char volatile far *)0x7efa75)
#define   DMA_LCM_TXAL          (*(unsigned char volatile far *)0x7efa76)
#define   DMA_LCM_RXAH          (*(unsigned char volatile far *)0x7efa77)
#define   DMA_LCM_RXAL          (*(unsigned char volatile far *)0x7efa78)

#define   DMA_M2M_AMTH          (*(unsigned char volatile far *)0x7efa80)
#define   DMA_M2M_DONEH         (*(unsigned char volatile far *)0x7efa81)
#define   DMA_SPI_AMTH          (*(unsigned char volatile far *)0x7efa84)
#define   DMA_SPI_DONEH         (*(unsigned char volatile far *)0x7efa85)
#define   DMA_LCM_AMTH          (*(unsigned char volatile far *)0x7efa86)
#define   DMA_LCM_DONEH         (*(unsigned char volatile far *)0x7efa87)
#define   DMA_UR1T_AMTH         (*(unsigned char volatile far *)0x7efa88)
#define   DMA_UR1T_DONEH        (*(unsigned char volatile far *)0x7efa89)
#define   DMA_UR1R_AMTH         (*(unsigned char volatile far *)0x7efa8a)
#define   DMA_UR1R_DONEH        (*(unsigned char volatile far *)0x7efa8b)
#define   DMA_UR2T_AMTH         (*(unsigned char volatile far *)0x7efa8c)
#define   DMA_UR2T_DONEH        (*(unsigned char volatile far *)0x7efa8d)
#define   DMA_UR2R_AMTH         (*(unsigned char volatile far *)0x7efa8e)
#define   dma_ur2r_doneh        (*(unsigned char volatile far *)0x7efa8f)
#define   DMA_UR3T_AMTH         (*(unsigned char volatile far *)0x7efa90)
#define   DMA_UR3T_DONEH        (*(unsigned char volatile far *)0x7efa91)
#define   DMA_UR3R_AMTH         (*(unsigned char volatile far *)0x7efa92)
#define   dma_ur3r_doneh        (*(unsigned char volatile far *)0x7efa93)
#define   DMA_UR4T_AMTH         (*(unsigned char volatile far *)0x7efa94)
#define   DMA_UR4T_DONEH        (*(unsigned char volatile far *)0x7efa95)
#define   DMA_UR4R_AMTH         (*(unsigned char volatile far *)0x7efa96)
#define   DMA_UR4R_DONEH        (*(unsigned char volatile far *)0x7efa97)

#define   DMA_I2CT_CFG          (*(unsigned char volatile far *)0x7efa98)
#define   DMA_I2CT_CR           (*(unsigned char volatile far *)0x7efa99)
#define   DMA_I2CT_STA          (*(unsigned char volatile far *)0x7efa9a)
#define   DMA_I2CT_AMT          (*(unsigned char volatile far *)0x7efa9b)
#define   DMA_I2CT_DONE         (*(unsigned char volatile far *)0x7efa9c)
#define   DMA_I2CT_TXAH         (*(unsigned char volatile far *)0x7efa9d)
#define   DMA_I2CT_TXAL         (*(unsigned char volatile far *)0x7efa9e)
#define   DMA_I2CR_CFG          (*(unsigned char volatile far *)0x7efaa0)
#define   DMA_I2CR_CR           (*(unsigned char volatile far *)0x7efaa1)
#define   DMA_I2CR_STA          (*(unsigned char volatile far *)0x7efaa2)
#define   DMA_I2CR_AMT          (*(unsigned char volatile far *)0x7efaa3)
#define   DMA_I2CR_DONE         (*(unsigned char volatile far *)0x7efaa4)
#define   DMA_I2CR_RXAH         (*(unsigned char volatile far *)0x7efaa5)
```

```
#define   DMA_I2CR_RXAL         (*(unsigned char volatile far *)0x7efaa6)

#define   DMA_I2CT_AMTH         (*(unsigned char volatile far *)0x7efaa8)
#define   DMA_I2CT_DONEH        (*(unsigned char volatile far *)0x7efaa9)
#define   DMA_I2CR_AMTH         (*(unsigned char volatile far *)0x7efaaa)
#define   DMA_I2CR_DONEH        (*(unsigned char volatile far *)0x7efaab)

#define   DMA_I2C_CR            (*(unsigned char volatile far *)0x7efaad)
#define   DMA_I2C_ST1           (*(unsigned char volatile far *)0x7efaae)
#define   DMA_I2C_ST2           (*(unsigned char volatile far *)0x7efaaf)


//////////////////////////////////////////////////

//sfr     CANICR      =         0xf1;
//#define CANAR                 (*(unsigned char volatile far *)0x7efebb)
//#define CANDR                 (*(unsigned char volatile far *)0x7efebc)

//To use the following macro, you need to set EAXFR to 1 first.
//Methods of use.
//      char dat.
//
//      EAXFR = 1;                          // Enable access to XFR
//      dat = READ_CAN(RX_BUF0).            //Read CAN Register
//      WRITE_CAN(TX_BUF0, 0x55).           //Write CAN Register

#define   READ_CAN(reg)         (CANAR = (reg), CANDR)
#define   WRITE_CAN(reg, dat)   (CANAR = (reg), CANDR = (dat))

#define   MR            0x00
#define   CMR           0x01
#define   SR            0x02
#define   ISR           0x03
#define   IMR           0x04
#define   RMC           0x05
#define   BTR0          0x06
#define   BTR1          0x07
#define   TM0           0x06
#define   TM1           0x07
#define   TX_BUF0       0x08
#define   TX_BUF1       0x09
#define   TX_BUF2       0x0a
#define   TX_BUF3       0x0b
```

```
#define   RX_BUF0       0x0c
#define   RX_BUF1       0x0d
#define   RX_BUF2       0x0e
```

```
#define   RX_BUF3          0x0f
#define   ACR0             0x10
#define   ACR1             0x11
#define   ACR2             0x12
#define   ACR3             0x13
#define   AMR0             0x14
#define   AMR1             0x15
#define   AMR2             0x16
#define   AMR3             0x17
#define   ECC              0x18
#define   RXERR            0x19
#define   TXERR            0x1a
#define   ALC              0x1b

//////////////////////////////////////////////////
//LIN Control Regiter
//////////////////////////////////////////////////

//sfr     LINICR    =        0xf9.
//sfr     LINAR     =        0xfa.
//sfr     LINDR     =        0xfb;

//Methods of use.
//        char dat.
//
//        dat = READ_LIN(LBUF);          //Read CAN Register
//        WRITE_LIN(LBUF, 0x55).         //Write CAN Register

#define   READ_LIN(reg)              (LINAR = (reg), LINDR)
#define   WRITE_LIN(reg, dat)        (linar = (reg), lindr = (dat))

#define   LBUF             0x00
#define   LSEL             0x01
#define   LID              0x02
#define   LER              0x03
#define   LIE              0x04
#define   LSR              0x05
#define   LCR              0x05
#define   DLL              0x06
#define   DLH              0x07
#define   HDRL             0x08
#define   HDRH             0x09
#define   HDP              0x0A
```

```
//////////////////////////////////////////////////
//USB Control Regiter
```

```
//////////////////////////////////////////////////

//sfr     USBCLK      =       0xdc;
//sfr     USBDAT      =       0xec;
//sfr     USBCON      =       0xf4;
//sfr     USBADR      =       0xfc.

//Methods of use.
//      char dat.
//
//      READ_USB(CSR0, dat).                    //Read USB Register
//      WRITE_USB(FADDR, 0x00).                 //Write USB Register

#define READ_USB(reg, dat)          \
{                                   \
        while (USBADR & 0x80);      \
        USBADR = (reg) | 0x80;      \
        while (USBADR & 0x80);      \
        (dat) = USBDAT.
}       \

#define WRITE_USB(reg, dat)         \
{                                   \
        while (USBADR & 0x80);      \
        USBADR = (reg) & 0x7f;      \
        USBDAT = (dat).
}       \
```

```
#define  USBBASE        0
#define  FADDR          (USBBASE + 0)
#define  UPDATE         0x80
#define  POWER          (USBBASE + 1)
#define  ISOUD          0x80
#define  USBRST         0x08
#define  USBRSU         0x04
#define  USBSUS         0x02
#define  ENSUS          0x01
#define  INTRIN1        (USBBASE + 2)
#define  EP5INIF        0x20
#define  EP4INIF        0x10
#define  EP3INIF        0x08
#define  EP2INIF        0x04
#define  EP1INIF        0x02
#define  EP0IF          0x01
#define  INTROUT1       (USBBASE + 4)
#define  EP5OUTIF       0x20
#define
#define
#define
#define
#define
#define
```

```
#define  EP4OUTIF        0x10
#define  EP3OUTIF        0x08
#define  EP2OUTIF        0x04
#define  EP1OUTIF        0x02
#define  INTRUSB         (USBBASE + 6)
#define  SOFIF           0x08
#define  RSTIF           0x04
#define  RSUIF           0x02
#define  SUSIF           0x01
#define  INTRIN1E        (USBBASE + 7)
#define  EP5INIE         0x20
#define  EP4INIE         0x10
#define  EP3INIE         0x08
#define  EP2INIE         0x04
#define  EP1INIE         0x02
#define  EP0IE           0x01
#define  INTROUT1E       (USBBASE + 9)
#define  EP5OUTIE        0x20
#define  EP4OUTIE        0x10
#define  EP3OUTIE        0x08
#define  EP2OUTIE        0x04
#define  EP1OUTIE        0x02
#define  INTRUSBE        (USBBASE + 11)
#define  SOFIE           0x08
#define  RSTIE           0x04
#define  RSUIE           0x02
#define  SUSIE           0x01
#define  FRAME1          (USBBASE + 12)
#define  FRAME2          (USBBASE + 13)
#define  INDEX           (USBBASE + 14)
#define  INMAXP          (USBBASE + 16)
#define  CSR0            (USBBASE + 17)
#define  SSUEND          0x80
#define  SOPRDY          0x40
#define  SDSTL           0x20
#define  SUEND           0x10
#define  DATEND          0x08
#define  STSTL           0x04
#define  IPRDY           0x02
#define  OPRDY           0x01
#define  INCSR1          (USBBASE + 17)
#define  INCLRDT         0x40
#define  INSTSTL         0x20
#define  INSDSTL         0x10
#define  INFLUSH         0x08
#define  INUNDRUN        0x04
```

```
#define  INFIFONE       0x02
#define  INIPRDY        0x01
#define  INCSR2         (USBBASE + 18)
#define  INAUTOSET      0x80
#define  INISO          0x40
#define  INMODEIN       0x20
#define  INMODEOUT      0x00
#define  INENDMA        0x10
#define  INFCDT         0x08
#define  OUTMAXP        (USBBASE + 19)
#define  OUTCSR1        (USBBASE + 20)
#define  OUTCLRDT       0x80
#define  OUTSTSTL       0x40
#define  OUTSDSTL       0x20
#define  OUTFLUSH       0x10
#define  OUTDATERR      0x08
#define  OUTOVRRUN      0x04
#define  OUTFIFOFUL     0x02
#define  OUTOPRDY       0x01
#define  OUTCSR2        (USBBASE + 21)
#define  OUTAUTOCLR     0x80
#define  OUTISO         0x40
#define  OUTENDMA       0x20
#define  OUTDMAMD       0x10
#define  COUNT0         (USBBASE + 22)
#define  OUTCOUNT1      (USBBASE + 22)
#define  OUTCOUNT2      (USBBASE + 23)
#define  FIFO0          (USBBASE + 32)
#define  FIFO1          (USBBASE + 33)
#define  FIFO2          (USBBASE + 34)
#define  FIFO3          (USBBASE + 35)
#define  FIFO4          (USBBASE + 36)
#define  FIFO5          (USBBASE + 37)
#define  UTRKCTL        (USBBASE + 48)
#define  UTRKSTS        (USBBASE + 49)

//////////////////////////////////////////////////
//Interrupt                 Vector
//////////////////////////////////////////////////

#define  INT0_VECTOR            0           //0003H
#define  TMR0_VECTOR            1           //000BH
#define  INT1_VECTOR            2           //0013H
#define  TMR1_VECTOR            3           //001BH
#define  UART1_VECTOR           4           //0023H
#define  ADC_VECTOR             5           //002BH
```

```
#define  LVD_VECTOR        6     //0033H
//#define PCA_VECTOR       7     //003BH
#define  UART2_VECTOR      8     //0043H
#define  SPI_VECTOR        9     //004BH
#define  INT2_VECTOR       10    //0053H
#define  INT3_VECTOR       11    //005BH
#define  TMR2_VECTOR       12    //0063H
#define  USER_VECTOR       13    //006BH
#define  BRK_VECTOR        14    //0073H
#define  ICEP_VECTOR       15    //007BH
#define  INT4_VECTOR       16    //0083H
#define  UART3_VECTOR      17    //008BH
#define  UART4_VECTOR      18    //0093H
#define  TMR3_VECTOR       19    //009BH
#define  TMR4_VECTOR       20    //00A3H
#define  CMP_VECTOR        21    //00ABH
//#define PWM_VECTOR       22    //00B3H
//#define PWMFD_VECTOR     23    //00BBH
#define  I2C_VECTOR        24    //00C3H
#define  USB_VECTOR        25    //00CBH
#define  PWMA_VECTOR       26    //00D3H
#define  PWMB_VECTOR       27    //00DBH
#define  CAN1_VECTOR       28    //00E3H
#define  CAN2_VECTOR       29    //00EBH
#define  LIN_VECTOR        30    //00F3H

#define  RTC_VECTOR        36    //0123H
#define  P0INT_VECTOR      37    //012BH
#define  P1INT_VECTOR      38    //0133H
#define  P2INT_VECTOR      39    //013BH
#define  P3INT_VECTOR      40    //0143H
#define  P4INT_VECTOR      41    //014BH
#define  P5INT_VECTOR      42    //0153H
#define  P6INT_VECTOR      43    //015BH
#define  P7INT_VECTOR      44    //0163H
#define  DMA_M2M_VECTOR    47    //017BH
#define  DMA_ADC_VECTOR    48    //0183H
#define  DMA_SPI_VECTOR    49    //018BH
#define  DMA_UR1T_VECTOR   50    //0193H
#define  DMA_UR1R_VECTOR   51    //019BH
#define  DMA_UR2T_VECTOR   52    //01A3H
#define  DMA_UR2R_VECTOR   53    //01ABH
#define  DMA_UR3T_VECTOR   54    //01B3H
#define  DMA_UR3R_VECTOR   55    //01BBH
#define  DMA_UR4T_VECTOR   56    //01C3H
#define  DMA_UR4R_VECTOR   57    //01CBH
```

```
#define   DMA_LCM_VECTOR      58          //01D3H
#define   LCM_VECTOR          59          //01DBH
#define   DMA_I2CT_VECTOR     60          //01E3H
#define   DMA_I2CR_VECTOR     61          //01EBH
#define   I2S_VECTOR          62          //01F3H
#define   DMA_I2ST_VECTOR     63          //01FBH
#define   DMA_I2SR_VECTOR     64          //0203H

//////////////////////////////////////////////////
#define   EAXSFR()        EAXFR = 1       /* MOVX A ,@DPTR/MOVX @DPTR,A Instruction
                                           The object of the operation is the extended SFR(XSFR) */
#define   EAXRAM()        EAXFR = 0       /* MOVX A ,@DPTR/MOVX @DPTR,A Instruction
                                           The object of the operation is Extended RAM (XRAM) */.

//////////////////////////////////////////////////
#define   NOP1()          _nop_()
#define   NOP2()          NOP1(),NOP1()
#define   NOP3()          NOP2(),NOP1()
#define   NOP4()          NOP3(),NOP1()
#define   NOP5()          NOP4(),NOP1()
#define   NOP6()          NOP5(),NOP1()
#define   NOP7()          NOP6(),NOP1()
#define   NOP8()          NOP7(),NOP1()
#define   NOP9()          NOP8(),NOP1()
#define   NOP10()         NOP9(),NOP1()
#define   NOP11()         NOP10(),NOP1()
#define   NOP12()         NOP11(),NOP1()
#define   NOP13()         NOP12(),NOP1()
#define   NOP14()         NOP13(),NOP1()
#define   NOP15()         NOP14(),NOP1()
#define   NOP16()         NOP15(),NOP1()
#define   NOP17()         NOP16(),NOP1()
#define   NOP18()         NOP17(),NOP1()
#define   NOP19()         NOP18(),NOP1()
#define   NOP20()         NOP19(),NOP1()
#define   NOP21()         NOP20(),NOP1()
#define   NOP22()         NOP21(),NOP1()
#define   NOP23()         NOP22(),NOP1()
#define   NOP24()         NOP23(),NOP1()
#define   NOP25()         NOP24(),NOP1()
#define   NOP26()         NOP25(),NOP1()
#define   NOP27()         NOP26(),NOP1()
```

```
#define   NOP28()       NOP27(),NOP1()
#define   NOP29()       NOP28(),NOP1()
#define   NOP30()       nop29(),nop1()
```

```
#define  NOP31()        NOP30(),NOP1()
#define  NOP32()        nop31(),nop1()
#define  NOP33()        NOP32(),NOP1()
#define  NOP34()        NOP33(),NOP1()
#define  NOP35()        NOP34(),NOP1()
#define  NOP36()        NOP35(),NOP1()
#define  NOP37()        NOP36(),NOP1()
#define  NOP38()        NOP37(),NOP1()
#define  NOP39()        NOP38(),NOP1()
#define  NOP40()        nop39(),nop1()
#define  NOP(N)         NOP##N()


#endif
```

Appendix N Electrical Characteristics

Absolute maximum rating

parameters	minimum value	maximum values	unit (of measure)	clarification
Storage temperature	-55	+155	℃	
operating temperature	-40	+85	℃	Uses an internal high-speed IRC (36M) and an external crystal oscillator
	-40	+125	℃	When the temperature is higher than 85℃, please use an external high-temperature-resistant crystal. It is recommended that the operating frequency be kept below 24M.
operating voltage	1.9	5.5	V	When the working temperature is lower than -40℃, the working voltage shall not be lower than 3.0V.
VDD Voltage to ground	-0.3	+5.5	V	
I/O Port Voltage to Ground	-0.3	VDD+0.3	V	

DC Characteristics (VSS=0V, VDD=5.0V, Test Temperature=25℃)

grade	parameters	realm				test environment
		minimum value	typical value	maximum values	unit (of measure)	
IPD	Power-down mode current	-	0.6	-	uA	
IWKT	Power-down wake-up timer	-	4.4	-	uA	
ILVD	Low Voltage Detection Module	-	430	-	uA	
IIDL	Idle Mode Current (6MHz)	-	0.99	-	mA	
	Idle mode current (11.0592MHz)	-	1.14	-	mA	
	Idle Mode Current (24MHz)	-	1.37	-	mA	
	Idle Mode Current (Internal 32KHz)	-	0.57	-	mA	
INOR	Normal Mode Current (6MHz)	-	1.65	-	mA	
	Normal mode current (11.0592MHz)	-	2.29	-	mA	
	Normal Mode Current (24MHz)	-	3.49	-	mA	
	Normal Mode Current (Internal 32KHz)	-	0.57	-	mA	
ICC	Normal operating mode current	-	4	20	mA	
VIL1	Input Low Level	-	-	1.32	V	Turn on the Schmidt

I/O tempo	I/O high current drive, I/O fast changeover		36		MHz	PxDR=0, PxSR=0
	I/O low current drive, I/O fast changeover		32		MHz	PxDR=1, PxSR=0
	I/O high current drive, I/O slow conversion		26		MHz	PxDR=0, PxSR=1
	I/O small current drive, I/O slow switching		22		MHz	PxDR=1, PxSR=1
comparator	top speed		10		MHz	Disable all analogue and digital filtering
	Analogue filter time		0.1		us	
	Digital filtering time		0		systems clocks	LCDTY = 0
			n+2			LCDTY=n (n=1~63)
IPD2	Power-down mode power consumption with comparator enabled	-	460	-	uA	
IPD3	Power-down mode power consumption when LVD is enabled	-	520	-	uA	

DC Characteristics (VSS=0V, VDD=3.3V, Test Temperature=25°C)

grade	parametric	realm				test environment
		minimum value	typical value	maximum values	unit (of measure)	
IPD	Power-down mode current	-	0.4	-	uA	
IWKT	Power-down wake-up timer	-	1.5	-	uA	
ILVD	Low Voltage Detection Module	-	364	-	uA	
IIDL	Idle Mode Current (6MHz)	-	0.89	-	mA	
	Idle mode current (11.0592MHz)	-	1.05	-	mA	
	Idle Mode Current (24MHz)	-	1.28	-	mA	
	Idle Mode Current (Internal 32KHz)	-	0.47	-	mA	
INOR	Normal Mode Current (6MHz)	-	1.55	-	mA	
	Normal mode current (11.0592MHz)	-	2.19	-	mA	
	Normal Mode Current (24MHz)	-	3.38	-	mA	
	Normal Mode Current (Internal 32KHz)	-	0.47	-	mA	
ICC	Normal operating mode current	-	4	20	mA	
VIL1	Input Low Level	-	-	0.99	V	Turn on the Schmidt trigger.
		-	-	1.07	V	Shutting down the Schmidt trigger
VIH1	Input High (Normal I/O)	1.18	-	-	V	Turn on the Schmidt trigger.
		1.09	-	-	V	Shutting down the Schmidt trigger
VIH2	Input high (reset pin)	0.99	-	1.18	V	
IOL1	Output low level of the irrigation current	-	20	-	mA	Port Voltage 0.45V
IOH1	Output high level current (bidirectional mode)	200	270	-	uA	
IOH2	Output high level current (push-pull mode)	-	20	-	mA	Port Voltage 2.4V
IIL	Logic 0 Input Current	-	-	50	uA	Port Voltage 0V
ITL	Logic 1 to 0 transfer current	100	270	600	uA	Port Voltage 2.0V
RPU	I/O port pull-up resistor	5.8	5.9	6.0	kΩ	

I/O tempo	I/O high current drive, I/O fast changeover		25		MHz	PxDR=0, PxSR=0
	I/O low current drive, I/O fast changeover		22		MHz	PxDR=1, PxSR=0
	I/O high current drive, I/O slow switching		16		MHz	PxDR=0, PxSR=1

	I/O small current drive, I/O slow switching		12		MHz	PxDR=1, PxSR=1
comparator	top speed		10		MHz	Disable all analogue and digital filtering
	Analogue filter time		0.1		us	
	Digital filtering time		0		systems clocks	LCDTY = 0
			n+2			LCDTY=n (n=1~63)
IPD2	Power-down mode power consumption with comparator enabled	-	400	-	uA	
IPD3	Power-down mode power consumption when LVD is enabled	-	470	-	uA	

Internal IRC Temperature Drift Characteristics (Reference Temperature 25°C)

temp	realm		
	minimum value	typical value	maximum values
-40°C～85°C		-1.38 per cent to +1.42 per cent	
-20°C～65°C		-0.88 per cent to +1.05 per cent	

Low Voltage Reset Threshold Voltage (Test Temperature 25°C)

(military) rank	input voltage		
	minimum value	typical value	maximum values
POR		1.9V	
LVR0		2.0V	
LVR1		2.4V	
LVR2		2.7V	
LVR3		3.0V	

Appendix O Updating the Record

- **2022/11/14**
  1. Minimum system power management reference circuit diagram added to the appendix
  2. The capacitors in the download reference circuit are strongly recommended to use the combination 22uF + 0.1uF (104)
  3. Fixed name error in STC32G12K128-LQDP48 pinout diagram

- **2022/10/31**
  1. Collate the instructions for the STC-USB Link1D tool and put the instructions for the STC-USB Link1 tool in the appendix.

- **2022/10/27**
  1. Add ISP download communication protocol flowchart
  2. Addition of the section on instructions for use of the connection cable that accompanies the STC-USB Link1D tool (P15.17.3)

- **2022/10/21**
  1. Update RTC power consumption parameters
  2. Updated pinout (added USB-TypeA download interface schematic)
  3. Update USB Download Reference Wiring Diagram
  4. All baud rate calculations for serial port examples are rounded up
  5. Corrected incorrect S2CON setting in some serial port 2 examples.

- **2022/10/12**
  1. Advanced PWM chapter adds a sample program for hardware implementation of pulse counting.
  2. Hardware USB download reference wiring diagram for adding a 3.3V system
  3. Add DMAIR register trigger description
  4. Add RTC real-world wiring diagram description

- **2022/9/21**
  1. Advanced PWM section added "Generate 3 PWMs with 120 degree phase difference" example program.
  2. Add a reference circuit diagram for the "Read/Write Off-Chip Expansion RAM"

example.

- **2022/9/20**

3. Updated Selection Price List
4. Addition of STC-ISP advanced application chapter

5. Add section on disabling mandatory digital signatures for drivers

- **2022/9/16**

1. Correction of clerical errors in the document
2. Updating the official website

- **2022/9/9**

1. Update Maximum Operating Frequency in STC32G8K64 Series Selection Price List
2. Fixed clerical errors in the I/O port chapter
3. Schematic diagram of adding protection diodes to the I/O internal structure drawing
4. Proposal to add "Decoupling capacitors for Vcc and Gnd" to the minimum system of each series.

- **2022/9/1**

1. Updated STC32G8K64 Series Selection Price List
2. Addition of a description of the Program Memory Cache (CACHE) control register (Memory section)
3. Add instructions for using the enhanced double data pointer

- **2022/8/26**

1. Fixed misdescription of some registers in the RTC chapter.
2. Added a sample program "TFT swipe using SPI_DMA+LCM_DMA double buffer" to the DMA chapter.

- **2022/8/24**

1. Fixed misdescription of some registers in the I2C chapter.
2. Fix PxDR register misdescription
3. Example program to add SPI mode to a USART.
4. Added a brief description of the CDC protocol to the USB chapter.

- **2022/8/4**

1. Corrected a clerical error in some of the descriptions in the DMA section.
2. Corrected description of SFRs partially affected by the LOCK bit in the Advanced PWM section.

- **2022/7/11**

1. Adding the STC-USB Link1D tool is an instruction manual.

- **2022/7/6**

1. Fixed ADC_DMA channel enable register description error.
2. Adding a note on the use of the SWRSTF register bit in the reset register RSTFLAG

- **2022/6/13**

1. Add automated batch production process description in appendix
2. Adds methods to create and edit EEPROM files using STC-ISP software.
3. Add product authorisation
4. Add STC32G header file definitions in the appendix
5. Example program to update a timer for external counting
6. Add PSW1 register description
7. Type-C connector wiring diagram added to USB download reference wiring diagrams

- **2022/6/6**

1. Add description of hardware options related to ISP downloads (Chapter 5)
2. Add section in appendix describing '0xFD' problem in Keil software
3. In the capture example program in the Advanced PWM chapter, add the principle, example description, and note description.
4. Advanced PWM section adds a sample program to capture the period and duty cycle of 4 signals at the same time.

- **2022/5/31**

1. Fixed a bug in the network name in the capture block diagram of the Advanced PWM chapter.
2. Adding a reference wiring diagram for USB direct download of the PDIP40 packaged chip
3. Memory section adds EAXFR register usage description subsection
4. Added reference wiring diagrams for ISP downloads using STC-USB Link1 and reordered chapters.
5. Example of adding an internal pull-up resistor to the enable I/O port
6. Adding STC-USB Link1 Tool Usage Precautions

- **2022/5/19**

1. Fixed typo in comparator chapter
2. I/O port section added Explanation of the precautions for using I/O ports
3. Fixed the bug that the S1BRT register was not set in the example of serial port 1 using timer 1 as baud rate generator.
4. Updating the description of the endpoint packet size register in the USB chapter
5. Memory section adds description of CKCON registers
6. Adding CKCON Initialisation Statements to Sample Programs
7. Interrupt entry address comparison table for STC32G and STC8G/8H added to the Interrupt section.

- **2022/5/5**

8. Corrected interrupt number error in the sample programme in the Advanced PWM

chapter.

9. Add the method of identifying the first pin and identifying the chip version by the chip silkscreen below the pinout.
10. Added a description of the P3.2 port when using USB direct download.
11. Added section on how to install Keil's C51, C251 and MDK at the same time.

- **2022/4/22**

1. Update serial port baud rate calculation instructions

2. Add the chapter "How to use a multimeter to test the chip I/O port" to the appendix.
3. Add a section on how to use the third-party extended interrupt tool.

- **2022/4/18**

1. Add instructions for using the CANAR and CANDR registers
2. Adding LINAR and LINDR register usage instructions
3. Add a description of the bit-addressable region and examples of the definition and use of bit variables to the memory section (9.2.8).

- **2022/4/7**

1. Correction of USB Chapter Register Error Description (INCSR1)
2. Correct the AUXR2 register address
3. Adding an Advanced PWM Cycle Trigger ADC Sample Program
4. Add CAN transmitting and receiving assembly example program

- **2022/3/30**

1. Add MODBUS protocol sample program to the serial port chapter.
2. Encoder sample programme added to Advanced PWM section
3. Correction of incorrect translations in the instruction set
4. Updated instructions for the MDU32 library and the FPMU library

- **2022/3/28**

1. Translating instruction sets into Chinese
2. Appendix adds a chapter on the fundamentals of logic algebra
3. Correcting errors in DMA-related register names

- **2022/3/17**

1. Addition of a section on instructions for writing assembler programmes
2. Addition of STC8H series projects and STC32G series projects conversion instructions.

- **2022/3/15**

1. Addition of a description of the operating voltage, operating frequency and operating temperature to the electrical characteristics
2. Add a physical drawing of the STC-USB Link1 tool and a hardware connection schematic.
3. Updating the EEPROM Example Program
4. ADC section adds sample program to invert the operating voltage using the internal 1.19V reference voltage signal source.

- **2022/3/11**

1. Addition of automatic frequency calibration chapter and sample program
2. Adding access to off-chip extended RAM example program
3. Added the description of "How to reserve EEPROM space when setting up a project".

4. Added a step-by-step description of "Emulating STC32G12K128 series microcontrollers with STC-USB Link1".
5. Addition of sections on "Methods of creating a multifile project" and "Handling of interrupt numbers greater than 31".
6. Added sample code for "ISP download of STC32G using a third-party MCU" to the appendix.

- **2022/3/9**

1. Modifying the Document Cover Characteristic Description
2. Modifying errors in the interrupt list in the Interrupts chapter
3. Updating RTC Chapter Register Descriptions
4. Clock description added to clock tree block diagram
5. Fixed register naming error in DMA section.
6. Adding project settings for over 64K codes
7. Add EEPROM address description, add EEPROM example program.
8. Updated BLDC Brushless DC Motor Drive Wiring Diagrams

- **2022/3/2**

1. Moved "Classic Circuit Diagram for DACs Using I/O and R-2R Resistor Voltage Dividing" to ADC chapter and appendices
2. Addition of sample programmes to the Clock chapter
3. High-speed PWM chapter and sample program added.
4. High-speed SPI chapter and sample program added.

- **2022/3/1**

1. Added "Classic wiring diagram for DAC using I/O and R-2R resistor divider" to I/O section.
2. Updating the clock tree in the Clocks chapter

- **2022/2/28**

1. Updating the description of the memory chapter
2. In all the example programs, the statement "WTST" has been added to set the wait time for the CPU to read the program code from FLASH.
   = 0;", i.e., no additional wait is actually required to achieve the fastest possible code execution by the CPU.

- **2022/1/19**

1. Rearrange all headings of the document
2. Translating the FPMU Single-Precision Floating-Point Operator Chapter
3. Updating the description of the memory chapter

- **2022/1/18**

  1. Add RTC real-time clock, LCM colour screen, DMA description chapter

- **2022/1/17**

  1. Completed the first edition of the STC32G Series Microcontroller Technical Reference Manual document.