

# STC32G series microcontroller

## technical reference manual

ÿ 10 32-bit accumulators  
ÿ 16 16-bit accumulators  
ÿ 16 8-bit accumulators  
ÿ 32-bit addition and subtraction instructions  
ÿ 16-bit multiply and divide instructions  
ÿ 32-bit multiply and divide (MDU32)  
ÿ 32-bit arithmetic comparison instructions  
ÿ All SFRs (80H~FFH) all support bit addressing  
ÿ eedata (20H~7FH) all support bit addressing  
ÿ Single clock 32/16/8 bit data read and write (edata)  
ÿ Single clock port read and write  
ÿ The theoretical stack depth can reach 64K (Actually depends on edata)  
ÿ FreeRTOS for STC32G12K128: The efficient and stable version of STC official transplant has been released

## Table of contents

1	<b>Overview .....</b>	1
2	<b>Features, Price and Pinouts .....</b>	2
2.1	STC32G12K128-LQFP64/LQFP48/LQFP32/PDIP40.....	2
2.1.1	Features and Price .....	2
2.1.2	Pin Diagram, Minimum System .....	5
2.1.3	Pin Description .....	9
2.2	STC32G8K64-LQFP48/LQFP32/PDIP40 .....	17
2.2.1	Features and Price .....	17
2.2.2	Pin Diagram, Minimum System .....	20
2.2.3	Pin Description .....	twenty two
2.3	STC32F12K60-LQFP48/LQFP32/PDIP40 .....	29
2.3.1	Features and Price .....	29
2.3.2	Pin Diagram, Minimum System .....	32
2.3.3	Pin Description .....	35
3	<b>Function foot switch.....</b>	43
3.1	Function pins switch related registers .....	43
3.1.1	Peripheral port switch control register 1 (P_SW1).....	43
3.1.2	Peripheral port switch control register 2 (P_SW2).....	44
3.1.3	Peripheral port switch control register 3 (P_SW3).....	45
3.1.4	Clock Select Register (MCLKOCR) .....	45
3.1.5	T3/T4 selection register (T3T4PIN).....	46
3.1.6	Advanced PWM Select Register (PWMMn_PS).....	46
3.1.7	Advanced PWM function pin selection register (PWMx_ETRPS) .....	47
3.2	Sample Program .....	49
3.2.1	Serial port 1 switching.....	49
3.2.2	Serial port 2 switching.....	49
3.2.3	Serial port 3 switching.....	50
3.2.4	Serial port 4 switching.....	50
3.2.5	SPI switching.....	51
3.2.6	I2C switching.....	52
3.2.7	Comparator Output Switching .....	52
3.2.8	Master Clock Output Switching .....	53
4	<b>Package Dimensions.....</b>	55
4.1	LQFP32 Package Dimensions (9mm*9mm).....	55
4.2	QFN32 Package Dimensions (4mm*4mm).....	56
4.3	LQFP48 Package Dimensions (9mm*9mm).....	57
4.4	QFN48 Package Dimensions (6mm*6mm).....	58
4.5	LQFP64 Package Dimensions (12mm*12mm).....	59
4.6	QFN64 Package Dimensions (8mm*8mm).....	60
4.7	STC32G series MCU naming rules.....	61

<b>5</b>	Compilation, establishment of simulation development environment and <b>ISP download</b> .....	<b>62</b>
5.1	Install Keil.....	62
5.1.1	Install C251 Compilation Environment .....	62
5.1.2	How to install Keil's C51, C251 and MDK at the same time.....	65
5.2	Add model and header files to Keil .....	66
5.3	Creating and setting up a project with super 64K code (Source mode) .....	68
5.3.1	Set project path and project name.....	68
5.3.2	Select the target microcontroller model (STC32G12K128).....	69
5.3.3	Adding source code files to the project.....	70
5.3.4	Setting item 1 ("CPU Mode" select Source mode).....	71
5.3.5	Set item 2 ("Memory Model" select XSmall mode).....	72
5.3.6	Set item 3 ("Code Rom Size" select Large or Huge mode) .....	75
5.3.7	Setting item 4 (related settings of super 64K code) .....	76
5.3.8	Setting item 5 (HEX file format setting).....	77
5.4	Assembling code based on STC32G series in Keil.....	78
5.4.1	How to write an assembler with a code size of less than 64K.....	78
5.4.2	How to write an assembler with a code size of more than 64K.....	79
5.5	STC8H series projects converted to STC32G series .....	81
5.6	STC32G series projects converted to STC8H series .....	83
5.7	The easy way to get help in Keil software.....	85
5.8	How to create a multi-file project in Keil.....	88
5.9	About the processing of compiling errors in Keil when the interrupt number is greater than 31.....	92
5.9.1	Use the popular INTERRUPT number extension tool on the Internet .....	92
5.9.2	Transferring with reserved interrupt numbers.....	94
5.10	How to set the reserved EEPROM space when the program exceeds 64K.....	104
5.11	Use STC-USB Link1 to emulate STC32G series MCU (Note: other models cannot be emulated)	
	106	
5.11.1	Getting to know the STC-USB Link1 tool.....	106
5.11.2	Hardware connection method.....	106
5.11.3	Install the emulation driver.....	107
5.11.4	Making an Emulated Chip .....	109
5.11.5	Creating and setting up a project in the Keil software.....	110
5.11.6	Compile, Download and Simulate .....	114
5.12	Precautions for using the STC-USB Link1 tool.....	116
5.12.1	Correct identification of tools.....	116
5.12.2	Tool firmware auto-upgrade.....	117
5.12.3	Enter the method to update the firmware .....	117
5.12.4	Enter method 2 to update firmware .....	118
5.12.5	STC-USB Link1 working indicator description.....	119
5.13	Precautions for using STC-USB Link1D tool.....	120
5.13.1	Tool Interface Description .....	120
5.13.2	STC-USB Link1D Practical Application.....	121
5.13.3	Correct identification of tools.....	123
5.13.4	Tool firmware auto-upgrade.....	124

5.13.5	Enter the method to update the firmware .....	124
5.13.6	STC-USB Link1D driver installation steps.....	125
5.13.7	How to turn off the virtual CD-ROM autoplay popup.....	131
5.14	Programming with ISP .....	133
5.15	Instructions for ISP download-related hardware options.....	134
5.16	The method of resetting the user program to the system area for ISP download in USB mode (without power failure).....	135
5.17	ISP Download Typical Application Circuit Diagram .....	138
5.17.1	Hardware USB direct ISP download, emulation follow-up support.....	138
5.17.2	Use STC-USB Link1 tool to download, support ISP online and offline download.....	140
5.17.3	Use U8-Mini tool to download, support ISP online and offline download.....	141
5.17.4	Use U8W tool to download, support ISP online and offline download.....	142
5.17.5	U8W pass-through mode, can be used for emulation, serial communication.....	143
5.17.6	Download using a general USB-to-serial tool.....	144
5.17.7	Download using PL2303-GL.....	145
<b>6</b>	<b>Clock Management .....</b>	<b>146</b>
6.1	System Clock Control.....	146
6.2	Related Registers.....	147
6.2.1	USB Clock Control Register (USBCLK) .....	147
6.2.2	System Clock Select Register (CLKSEL) .....	149
6.2.3	Clock Divider Register (CLKDIV) .....	149
6.2.4	Internal High Speed High Precision IRC Control Register (HIRCCR).....	149
6.2.5	External Oscillator Control Register (XOSCCR) .....	150
6.2.6	Internal 32KHz low-speed IRC control register (IRC32KCR) .....	150
6.2.7	Master Clock Output Control Register (MCLKOCR) .....	151
6.2.8	High Speed Oscillator Settling Time Control Register (IRCDB).....	151
6.2.9	Internal 48MHz High Speed IRC Control Register (IRC48MCR).....	151
6.2.10	External 32K Oscillator Control Register (X32KCR) .....	152
6.2.11	High Speed Clock Divider Register (HSCLKDIV) .....	152
6.3	STC32G series internal IRC frequency adjustment .....	153
6.3.1	IRC Band Select Register (IRCBAND) .....	153
6.3.2	Internal IRC Frequency Trim Register (IRTRIM).....	153
6.3.3	Internal IRC Frequency Trim Register (LIRTRIM) .....	153
6.3.4	Clock Divider Register (CLKDIV) .....	154
6.4	External crystal oscillator and external clock circuit.....	155
6.4.1	External crystal oscillator input circuit.....	155
6.4.2	External clock input circuit (P1.6 cannot be used as normal I/O).....	155
6.5	Sample Program .....	156
6.5.1	Selecting the Internal High Speed IRC (HIRC) as the System Clock Source.....	156
6.5.2	Selecting the internal IRC (IRC32K) as the system clock source.....	156
6.5.3	Select the internal 48M IRC (IRC48M) as the system clock source.....	157
6.5.4	Selecting an external high-speed crystal oscillator (XOSC) as the system clock source.....	158
6.5.5	Select an external low-speed crystal oscillator (X32K) as the system clock source.....	158
6.5.6	Selecting the internal PLL as the system clock source.....	159
6.5.7	Selecting the master clock (MCLK) as the high-speed peripheral clock source.....	160

6.5.8 Selecting the internal PLL clock as the high-speed peripheral clock source.....	161
6.5.9 Select system clock (SYSCLK) as USB clock source.....	161
6.5.10 Selecting the internal PLL clock as the USB clock source.....	162
6.5.11 Select the internal USB dedicated 48M IRC as the USB clock source.....	163
6.5.12 Main Clock Divided Output .....	164
<b>7 Automatic frequency calibration, automatic frequency tracking (CRE).....</b>	<b>165</b>
7.1 Related Registers.....	165
7.1.1 CRE Control Register (CRECR) .....	165
7.1.2 CRE Calibration Count Value Register (CRECNT).....	166
7.1.3 CRE Calibration Error Value Register (CRERES) .....	166
7.2 Sample Program .....	167
7.2.1 Auto-calibrate the internal high-speed IRC (HIRC).....	167
<b>8 Dedicated timers for reset, watchdog, wake-up from power-down and power management.....</b>	<b>169</b>
8.1 System reset.....	169
8.1.1 Watchdog Control Register (WDT_CONTR).....	169
8.1.2 IAP Control Register (IAP_CONTR).....	170
8.1.3 RESET CONFIGURATION REGISTER (RSTCFG) .....	170
8.1.4 Reset Flag Register (RSTFLAG) .....	171
8.1.5 Reset Control Register (RSTCRx) .....	172
8.1.6 Low-level power-on reset reference circuit (generally not required) .....	173
8.1.7 Low-level key reset reference circuit.....	173
8.1.8 Traditional 8051 high-level power-on reset reference circuit.....	174
8.2 System Power Management.....	175
8.2.1 Power Control Register (PCON) .....	175
8.3 Power-down wake-up timer.....	176
8.3.1 Power-down wake-up timer count registers (WKTCL, WKTCH) .....	176
8.4 Sample Program .....	178
8.4.1 Watchdog Timer Application .....	178
8.4.2 Soft reset to achieve custom download .....	178
8.4.3 Low Voltage Detection .....	179
8.4.4 Power saving mode.....	180
8.4.5 Wake-up from power saving mode using INT0/INT1/INT2/INT3/INT4 pin interrupts.....	181
8.4.6 Wake-up from power saving mode using T0/T1/T2/T3/T4 pin interrupts.....	182
8.4.7 Wake-up from power saving mode using RxD/RxD2/RxD3/RxD4 pin interrupt .....	184
8.4.8 Use I2C SDA pin to wake up the MCU power saving mode.....	186
8.4.9 Wake-up from power-saving mode using the power-down wake-up timer.....	187
8.4.10 LVD interrupt wake-up power saving mode, it is recommended to use the power-down wake-up timer together.....	187
8.4.11 Comparator interrupt wake-up from power-saving mode, it is recommended to use the power-down wake-up timer together.....	189
8.4.12 Using the LVD function to detect the operating voltage (battery voltage) .....	190
<b>9 Memory (32-bit access, 16-bit access, 8-bit access).....</b>	<b>193</b>
9.1 Program memory.....	194
9.1.1 Program Read Wait Control Register (WTST) .....	195
9.2 Data Memory (32-bit access, 16-bit access, 8-bit access) .....	196
9.2.1 Keil Options Memory Model Settings .....	197

9.2.2 Internal edata-RAM (C language declaration keyword is edata) .....	199
9.2.3 Program Status Register (PSW) .....	200
9.2.4 Program Status Register 1 (PSW1).....	200
9.2.5 Internal xdata-RAM (C language declaration keyword is xdata).....	201
9.2.6 Auxiliary Register (AUXR) .....	201
9.2.7 External xdata-RAM.....	201
9.2.8 BUS SPEED CONTROL REGISTER (BUS_SPEED) .....	201
9.2.9 External Data Bus Clock Control Register (CKCON) .....	202
9.2.10 Bit-addressable data memory in STC32G series microcontrollers.....	203
9.2.11 Instructions for the use of the extended SFR enable register EAXFR.....	205
9.3 Unique ID number and important parameter (CHIPID) stored in read-only special function register .....	206
9.3.1 Interpretation of CHIP's globally unique ID number.....	208
9.3.2 Interpretation of the internal reference signal source of CHIP.....	208
9.3.3 Interpretation of the internal 32K IRC oscillation frequency of CHIP.....	208
9.3.4 Interpretation of high-precision IRC parameters of CHIP.....	210
9.3.5 Interpretation of test time parameters of CHIP.....	211
9.3.6 Interpretation of the chip package form number of CHIP.....	211
9.4 Sample Program .....	212
9.4.1 Read the internal 1.19V reference signal source value.....	212
9.4.2 Read the global unique ID number.....	213
9.4.3 Read the frequency of the 32K power-down wake-up timer.....	214
9.4.4 User-Defined Internal IRC Frequency.....	216
9.4.5 Read and write off-chip extended RAM.....	218
<b>10 Special Function Registers (SFR, XFR) .....</b>	<b>219</b>
10.1 STC32G12K128 series.....	219
10.2 STC32G8K64 series.....	221
10.3 STC32F12K60 series.....	223
10.4 List of Special Function Registers (SFR: 0x80-0xFF) .....	225
10.5 Extended Special Function Register List (XFR: 0x7EFE00-0x7FEFFF) .....	228
10.6 Extended Special Function Register List (XFR: 0x7EFD00-0x7EFDFF).....	233
10.7 Extended Special Function Register List (XFR: 0x7EFB00-0x7EFBFF) .....	236
10.8 Extended Special Function Register List (XFR: 0x7EFA00-0x7EFAFF) .....	236
<b>11 I/O port.....</b>	<b>241</b>
11.1 I/O port related registers.....	241
11.1.1 Port Data Register (Px) .....	244
11.1.2 Port Mode Configuration Registers (PxM0, PxM1).....	244
11.1.3 Port Pull-Up Resistor Control Register (PxPU).....	245
11.1.4 Port Schmitt Trigger Control Register (PxNCS) .....	245
11.1.5 PORT TRANSLATION SPEED CONTROL REGISTER (PxSR) .....	245
11.1.6 Port Drive Current Control Register (PxDR) .....	246
11.1.7 Port digital signal input enable control register (PxIE) .....	246
11.1.8 Port Pull-Down Resistor Control Register (PxPD) .....	246
11.2 Configuring I/O Ports.....	248
11.3 I/O block diagram.....	249

11.3.1	Quasi-bidirectional port (weak pull-up).....	249
11.3.2	Push-Pull Output .....	249
11.3.3	Hi-Z Input .....	250
11.3.4	Open-Drain Output .....	250
11.3.5	Added 4.1K pull-up resistor and 10K pull-down resistor.....	251
11.3.6	How to set the external output speed of the I/O port.....	252
11.3.7	How to set the current drive capability of I/O port.....	253
11.3.8	How to reduce the external radiation of I/O ports.....	253
11.4	Sample Program .....	254
11.4.1	Port mode settings (applicable to all I/Os).....	254
11.4.2	Bidirectional port read and write operations (applicable to all I/Os).....	254
11.4.3	Turn on the internal pull-up resistor of the I/O port (applicable to all I/Os).....	255
11.5	A typical triode control circuit.....	257
11.6	Typical Light Emitting Diode Control Circuit .....	258
11.7	3V/5V device I/O port interconnection in mixed voltage power supply system.....	258
11.8	How to make I/O port low level when power-on reset.....	259
11.9	Using 74HC595 to drive 8 nixie tubes (serial extension, 3 wires) wiring diagram.....	260
<b>12</b>	<b>Interrupt the system .....</b>	<b>261</b>
12.1	STC32G Series Interrupt Sources.....	261
12.2	STC32G Interrupt Structure Diagram.....	263
12.3	STC32G Series Interrupt List.....	264
12.4	Interrupt Related Registers.....	268
12.4.1	Interrupt Enable Register (Interrupt Enable Bit).....	270
12.4.2	Interrupt Request Register (Interrupt Flag Bit) .....	279
12.4.3	Interrupt Priority Register .....	284
12.5	Sample Program .....	292
12.5.1	INT0 interrupt (rising edge and falling edge), can support both rising edge and falling edge.....	292
12.5.2	INT0 interrupt (falling edge) .....	293
12.5.3	INT1 interrupt (rising edge and falling edge), can support both rising and falling edge.....	293
12.5.4	INT1 interrupt (falling edge) .....	294
12.5.5	INT2 interrupt (falling edge), only falling edge interrupts are supported.....	295
12.5.6	INT3 interrupt (falling edge), only falling edge interrupts are supported.....	296
12.5.7	INT4 interrupt (falling edge), only falling edge interrupts are supported.....	296
12.5.8	Timer 0 Interrupt .....	297
12.5.9	Timer 1 Interrupt .....	298
12.5.10	Timer 2 Interrupt .....	299
12.5.11	Timer 3 Interrupt .....	300
12.5.12	Timer 4 Interrupt .....	300
12.5.13	UART1 Interrupt .....	301
12.5.14	UART2 Interrupt .....	302
12.5.15	UART3 Interrupt .....	303
12.5.16	UART4 Interrupt .....	304
12.5.17	ADC Interrupt.....	305
12.5.18	LVD Interrupt.....	306

12.5.19	Comparator Interrupt .....	307
12.5.20	SPI Interrupt.....	307
12.5.21	I2C Interrupt.....	308
<b>13</b>	Ordinary I/O ports can be interrupted, not traditional external interrupts .....	310
13.1	I/O Port Interrupt Related Registers.....	310
13.1.1	Port Interrupt Enable Register (PxINTE).....	311
13.1.2	Port Interrupt Flag Register (PxINTF) .....	311
13.1.3	PORT INTERRUPT MODE CONFIGURATION REGISTER (PxIM0, PxIM1).....	312
13.1.4	Port Interrupt Priority Control Registers (PINIPL, PINIPH).....	312
13.1.5	Port Interrupt Power-Down Wake-Up Enable Register (PxWKUE).....	313
13.2	Sample Program .....	314
13.2.1	Port 0 falling edge interrupt .....	314
13.2.2	Rising edge of port 1 interrupt.....	315
13.2.3	P2 port low level interrupt.....	317
13.2.4	P3 port high level interrupt .....	319
<b>14</b>	Timer/Counter (24 -bit timer, 8 -bit prescaler + 16 -bit auto-reload) .....	321
14.1	Timer related registers .....	321
14.2	Timer 0/1 .....	323
14.2.1	Timer 0/1 Control Register (TCON).....	323
14.2.2	Timer 0/1 Mode Register (TMOD) .....	323
14.2.3	Timer 0 Mode 0 (16-Bit Auto-Reload Mode) .....	324
14.2.4	Timer 0 Mode 1 (16-bit non-reloadable mode) .....	325
14.2.5	Timer 0 Mode 2 (8-Bit Auto-Reload Mode) .....	326
14.2.6	Timer 0 Mode 3 (Non-Maskable Interrupt 16-Bit Auto-Reload, RTOS Metronome).....	326
14.2.7	Timer 1 Mode 0 (16-Bit Auto-Reload Mode) .....	327
14.2.8	Timer 1 Mode 1 (16-bit non-reloadable mode) .....	328
14.2.9	Timer 1 Mode 2 (8-Bit Auto-Reload Mode) .....	329
14.2.10	Timer 0 count register (TL0, TH0).....	329
14.2.11	Timer 1 count register (TL1, TH1).....	329
14.2.12	Auxiliary Register 1 (AUXR).....	330
14.2.13	Interrupt and Clock Output Control Register (INTCLKO) .....	330
14.2.14	Timer 0 8-bit prescaler register (TM0PS).....	330
14.2.15	Timer 1 8-bit Prescaler Register (TM1PS).....	330
14.2.16	Calculation formula of timer 0.....	331
14.2.17	Calculation formula of timer 1.....	331
14.3	Timer 2 .....	332
14.3.1	Auxiliary Register 1 (AUXR).....	332
14.3.2	Interrupt and Clock Output Control Register (INTCLKO) .....	332
14.3.3	Timer 2 Count Registers (T2L, T2H).....	332
14.3.4	Timer 2 8-bit Prescaler Register (TM2PS).....	332
14.3.5	Timer 2 working mode.....	333
14.3.6	Calculation formula of timer 2.....	333
14.4	Timer 3/4 .....	334
14.4.1	Timer 3/4 function pin switching.....	334

14.4.2 Timer 4/3 Control Register (T4T3M).....	334	14.4.3 Timer 3 Counting Registers (T3L, T3H) .....	335	14.4.4 Timer 4 Count Registers (T4L, T4H).....	335
14.4.5 Timer 3 8-bit prescaler register (TM3PS) .....	335	14.4.6 Timer 4 8-bit prescaler register (TM4PS) .....	335	14.4.7 Timer 3 Operating mode.....	335
14.4.8 Timer 4 working mode.....	336	14.4.9 Timer 3 Calculation Formula .....	337	14.4.10 Timer 4 Calculation Formula .....	337
14.4.11 Sample					
14.5 14.5.1 Timer 0 (grades 0..16-bit auto-reload), used as timer.....	338	14.5.2.....	338	14.5.3	
Timer 0 (Mode 1 - 16-bit without auto-reload ), used as a timer .....	338				
Timer 0 (mode 2 - 8-bit auto-reload), used as a timer .....	339	14.5.4 Timer 0 (Mode 3 - 16-bit Auto reload non-maskable interrupt), used as timer .....	340	14.5.5 Timer 0 ( External counting - extended T0 for external falling edge interrupt) .....	341
14.5.6 Timer 0 (measurement pulse width - INT0 high level width) .....	342				
14.5.7 Timer 0 (mode 0), clock divider output .....	343	14.5.8 Timer 1 (Mode 0- 16-bit auto-reload), used as a timer .....	343	14.5.9 Timer 1 (mode 1 - 16-bit without auto-reload), used as a timer .....	344
14.5.10 Timer 1 (Mode 2 -8-bit auto-reload), Use as a timer .....	345	14.5.11 Timer 1 (External Counting-Extended T1 for External Falling Edge Interrupt).....	346	14.5.12 Timer 1 (measurement pulse width - INT1 high level width) .....	346
14.5.13 Timer 1 (mode 0), clock divider output .....	347	14.5.14 Timer 1 (mode 0) as serial port 1 baud rate generator.....	348	14.5.15 Timer 1 ( Mode 2) Do serial port 1 baud rate generator .....	350
14.5.16 Timer 2 (16-bit auto-reload), used as a timer.....	351	14.5.17 Timer 2 (external counting - extended T2 is an external falling edge interrupt) .....	352	14.5.18 Timer 2, clock divider output.....	353
14.5.19 Timer 2 do Serial 1 Baud Rate Generator .....	354	14.5.20 Timer 2 as serial port 2 Baud Rate Generator.....	355	14.5.21 Timer 2 as serial port 3 baud rate generator .....	357
14.5.22 Timer 2 as serial port 4 baud rate generator.....	359	14.5.23 Timer 3 (16-bit auto-reload), used as a timer.....	360	14.5.24 Timer 3 (external counting - extended T3 is external falling edge interrupt).....	361
14.5.25 Timer 3, clock divider output .....	362	14.5.26 Timer 3 as serial port 3 baud Rate Generator .....	363	14.5.27 Timer 4 (16-bit auto-reload) , used as a timer .....	364
14.5.28 Timer 4 (external counting - extended T4 is an external falling edge interrupt) .....	365	14.5.29 Timer 4. Clock frequency division output .....	366	14.5.30 Timer 4 as serial port 4 baud rate generator .....	367
<b>Synchronous/Asynchronous Serial Communication (USART1, USART2).....</b>	<b>369</b>				
15	Serial port function pin switch .....	369			
15.1	Serial port related registers .....	370			
15.2	Serial port 1 (synchronous/asynchronous serial port				
15.3	USART) .....	371			

15.3.1	Serial port 1 control register (SCON) .....	371
15.3.2	Serial port 1 data register (SBUF).....	371
15.3.3	Power Management Register (PCON) .....	372
15.3.4	Auxiliary Register 1 (AUXR). ....	372
15.3.5	Serial port 1 mode 0, mode 0 baud rate calculation formula.....	372
15.3.6	Serial port 1 mode 1, mode 1 baud rate calculation formula.....	373
15.3.7	Serial port 1 mode 2, mode 2 baud rate calculation formula.....	376
15.3.8	Serial port 1 mode 3, mode 3 baud rate calculation formula.....	377
15.3.9	Automatic Address Recognition.....	377
15.3.10	Serial port 1 slave address control register (SADDR, SADEN) .....	377
15.3.11	Serial Port 1 Synchronous Mode Control Register 1 (USARTCR1).....	378
15.3.12	Serial Port 1 Synchronous Mode Control Register 2 (USARTCR2).....	379
15.3.13	Serial Port 1 Synchronous Mode Control Register 3 (USARTCR3).....	380
15.3.14	Serial port 1 synchronous mode control register 4 (USARTCR4).....	380
15.3.15	Serial Port 1 Synchronous Mode Control Register 5 (USARTCR5).....	380
15.3.16	Serial port 1 synchronous mode guard time register (USARTGTR) .....	381
15.3.17	Serial Port 1 Synchronous Mode Baud Rate Register (USARTBR).....	381
15.4	Serial port 2 (synchronous/asynchronous serial port USART2) .....	382
15.4.1	Serial port 2 control register (S2CON) .....	382
15.4.2	Serial port 2 data register (S2BUF).....	382
15.4.3	Serial port 2 configuration register (S2CFG).....	383
15.4.4	Serial port 2 mode 0, mode 0 baud rate calculation formula.....	383
15.4.5	Serial port 2 mode 1, mode 1 baud rate calculation formula.....	384
15.4.6	Serial port 2 mode 2, mode 2 baud rate calculation formula.....	386
15.4.7	Serial port 2 mode 3, mode 3 baud rate calculation formula.....	387
15.4.8	Serial port 2 automatic address recognition.....	388
15.4.9	Serial port 2 slave address control register (S2ADDR, S2ADEN) .....	388
15.4.10	Serial port 2 synchronous mode control register 1 (USART2CR1).....	389
15.4.11	Serial port 2 synchronous mode control register 2 (USART2CR2).....	390
15.4.12	Serial port 2 synchronous mode control register 3 (USART2CR3).....	390
15.4.13	Serial port 2 synchronous mode control register 4 (USART2CR4).....	391
15.4.14	Serial port 2 synchronous mode control register 5 (USART2CR5).....	391
15.4.15	Serial Port 2 Synchronous Mode Guard Time Register (USART2GTR) .....	392
15.4.16	Serial Port 2 Synchronous Mode Baud Rate Register (USART2BR).....	392
15.5	Sample Program .....	393
15.5.1	Serial port 1 uses timer 2 as baud rate generator.....	393
15.5.2	Serial port 1 uses timer 1 (mode 0) as a baud rate generator.....	394
15.5.3	Serial port 1 uses timer 1 (mode 2) as a baud rate generator.....	396
15.5.4	Serial port 2 uses timer 2 as baud rate generator.....	398
16	<b>Asynchronous serial communication (UART3, UART4).....</b>	400
16.1	Serial port function pin switch.....	400
16.2	Serial port related registers.....	400
16.3	Serial port 3 (asynchronous serial port UART3) .....	401
16.3.1	Serial port 3 control register (S3CON) .....	401

16.3.2	Serial port 3 data register (S3BUF).....	401
16.3.3	Serial port 3 mode 0, mode 0 baud rate calculation formula.....	402
16.3.4	Serial port 3 mode 1, mode 1 baud rate calculation formula.....	402
16.4	Serial port 4 (asynchronous serial port UART4) .....	404
16.4.1	Serial port 4 control register (S4CON) .....	404
16.4.2	Serial port 4 data register (S4BUF).....	404
16.4.3	Serial port 4 mode 0, mode 0 baud rate calculation formula.....	405
16.4.4	Serial port 4 mode 1, mode 1 baud rate calculation formula.....	405
16.5	Serial port precautions.....	406
16.6	Sample Program .....	408
16.6.1	Serial port 3 uses timer 2 as baud rate generator.....	408
16.6.2	Serial port 3 uses timer 3 as baud rate generator.....	409
16.6.3	Serial port 4 uses timer 2 as baud rate generator.....	411
16.6.4	Serial port 4 uses timer 4 as baud rate generator.....	413
17	<b>Comparator, Brownout Detection, Internal Fixed Compare Voltage .....</b>	<b>415</b>
17.1	Comparator Internal Structure .....	415
17.2	Comparator function pin switching.....	415
17.3	Comparator Related Registers .....	416
17.3.1	Comparator Control Register 1 (CMPCR1) .....	416
17.3.2	Comparator Control Register 2 (CMPCR2) .....	416
17.3.3	Comparator Extended Configuration Register (CMPEXCFG).....	417
17.4	Sample Program .....	418
17.4.1	Comparator usage (interrupt method) .....	418
17.4.2	Use of Comparator (Inquiry Mode).....	419
17.4.3	Multiplexed Application of Comparator (Comparator + ADC Input Channel) .....	420
17.4.4	Comparator for external power-down detection (user data should be saved to EEPROM in time during power-down process) .....	421
17.4.5	Comparator detection operating voltage (battery voltage) .....	422
18	<b>IAP/EEPROM .....</b>	<b>425</b>
18.1	EEPROM operation time .....	425
18.2	EEPROM Related Registers.....	425
18.2.1	EEPROM Data Register (IAP_DATA) .....	426
18.2.2	EEPROM Address Register (IAP_ADDR) .....	426
18.2.3	EEPROM Command Register (IAP_CMD) .....	426
18.2.4	EEPROM Trigger Register (IAP_TRIG) .....	427
18.2.5	EEPROM Control Register (IAP_CONTR).....	427
18.2.6	EEPROM Erase Wait Time Control Register (IAP_TPS).....	427
18.3	EEPROM size and address.....	428
18.4	Sample Program .....	430
18.4.1	EEPROM Basic Operation .....	430
18.4.2	Using MOV to read EEPROM .....	431
18.4.3	Using serial port to send EEPROM data.....	433
18.4.4	Serial port 1 read and write EEPROM-with MOV read.....	435
18.4.5	Password erasing and writing-multi-sector backup-serial port 1 operation.....	442
19	<b>ADC analog-to-digital conversion, traditional DAC implementation.....</b>	<b>452</b>

19.1	ADC Related Registers.....	452
19.1.1	ADC Control Register (ADC_CONTR), PWM Triggered ADC Control.....	452
19.1.2	ADC CONFIGURATION REGISTER (ADCCFG) .....	453
19.1.3	ADC Conversion Result Registers (ADC_RES, ADC_RESL).....	454
19.1.4	ADC Timing Control Register (ADCTIM).....	454
19.2	ADC Static Characteristics.....	456
19.3	ADC related calculation formulas.....	456
19.3.1	ADC Speed Calculation Formula .....	456
19.3.2	ADC conversion result calculation formula.....	456
19.3.3	Inverse ADC input voltage calculation formula.....	456
19.3.4	Calculation formula of reverse working voltage.....	457
19.4	ADC Application Reference Circuit Diagram .....	458
19.4.1	Generic Precision ADC Reference Circuit Diagram.....	458
19.4.2	High-Precision ADC Reference Circuit Diagram.....	459
19.5	Sample Program .....	459
19.5.1	ADC basic operation (query mode).....	459
19.5.2	ADC Basic Operation (Interrupt Mode).....	460
19.5.3	Formatting ADC conversion results.....	461
19.5.4	Using ADC Channel 15 (Internal 1.19V Reference Signal Source) to Measure External Voltage or Battery Voltage .....	462
19.5.5	Circuit diagram of ADC as key scan application.....	465
19.5.6	Detecting Negative Voltage Reference Circuit Diagram.....	466
19.5.7	Application of Commonly Used Adding Circuits in ADCs.....	467
19.6	Classic circuit diagram for implementing a DAC using I/O and R-2R resistor divider.....	468
19.7	Reference circuit diagram for implementing a 16-bit DAC using PWM.....	469
20	<b>Synchronous Serial Peripheral Interface (SPI) .....</b>	<b>470</b>
20.1	SPI function pin switch.....	470
20.2	SPI related registers.....	470
20.2.1	SPI Status Register (SPSTAT).....	471
20.2.2	SPI Control Register (SPCTL), SPI Speed Control.....	471
20.2.3	SPI Data Register (SPDAT).....	472
20.3	SPI communication method.....	473
20.3.1	Single Master Single Slave .....	473
20.3.2	Mutual master-slave .....	473
20.3.3	Single master and multiple slaves .....	474
20.4	Configuring SPI.....	475
20.5	Data Mode .....	477
20.6	Sample Program .....	478
20.6.1	SPI single-master single-slave system host program (interrupt mode) .....	478
20.6.2	SPI single master single slave system slave program (interrupt mode) .....	479
20.6.3	SPI single-master single-slave system host program (query method) .....	480
20.6.4	SPI single master single slave system slave program (query method) .....	480
20.6.5	SPI mutual master-slave system program (interrupt mode) .....	481
20.6.6	SPI mutual master-slave system program (query method) .....	482
	<b>High Speed SPI (HSSPI) .....</b>	<b>485</b>

21.1	Related Registers.....	485
21.1.1	High Speed SPI Configuration Register (HSSPI_CFG).....	485
21.1.2	High Speed SPI Configuration Register 2 (HSSPI_CFG2).....	485
21.1.3	High Speed SPI Status Register (HSSPI_STA) .....	486
21.2	Sample Program .....	487
21.2.1	Enable high speed mode of SPI.....	487
	I2C bus .....	489
22.1	I2C function pin switching.....	489
22.2	I2C related registers .....	489
22.3	I2C master mode.....	490
22.3.1	I2C Configuration Register (I2CCFG), Bus Speed Control .....	490
22.3.2	I2C Master Control Register (I2CMSCR).....	491
22.3.3	I2C Master Auxiliary Control Register (I2CMSAUX).....	492
22.3.4	I2C Master Status Register (I2CMSST).....	492
22.4	I2C Slave Mode.....	494
22.4.1	I2C Slave Control Register (I2CSLCR).....	494
22.4.2	I2C Slave Status Register (I2CSLST).....	494
22.4.3	I2C Slave Address Register (I2CSLADR).....	495
22.4.4	I2C Data Registers (I2CTXD, I2CRXD).....	496
22.5	Sample Program .....	497
22.5.1	I2C host mode to access AT24C256 (interrupt mode).....	497
22.5.2	I2C host mode access to AT24C256 (inquiry mode).....	499
22.5.3	I2C host mode access to PCF8563.....	501
22.5.4	I2C Slave Mode (Interrupt Mode).....	504
22.5.5	I2C Slave Mode (Inquiry Mode).....	506
	22.5.6 Test Master Code for I2C Slave Mode Code.....	507
	Advanced PWM.....	510
23.1	Introduction .....	513
23.2	Main Features .....	513
23.3	Time base unit.....	514
23.3.1	Reading and writing a 16-bit counter.....	515
23.3.2	16-bit PWMA_ARR register write operation .....	515
23.3.3	Prescaler.....	515
23.3.4	Count Up Mode.....	516
23.3.5	Down Counting Mode.....	517
23.3.6	Center Alignment Mode (Count Up/Down).....	519
23.3.7	Repeat Counter .....	521
23.4	Clock/Trigger Controller .....	522
23.4.1	Prescaled Clock (CK_PSC) .....	522
23.4.2	Internal clock source (fMASTER) .....	523
	23.4.3 External Clock Source Mode 1.....	523
	23.4.4 External Clock Source Mode 2.....	524
23.4.5	Trigger Synchronization .....	525
	23.4.6 Synchronization with PWMB.....	528

23.5	Capture/Compare Channel .....	531
23.5.1	Writing process of 16-bit PWMA_CCRi register.....	533
23.5.2	Input module.....	533
23.5.3	Input Capture Mode .....	533
23.5.4	Output module.....	536
23.5.5	Forced output mode.....	537
23.5.6	Output Compare Mode.....	537
23.5.7	PWM mode.....	538
23.5.8	Using the Brake Function (PWMFLT) .....	544
23.5.9	Clearing the OCiREF signal on an external event.....	546
23.5.10	Encoder Interface Mode .....	547
23.6	Interruption .....	549
23.7	PWMA/PWMB Register Descriptions.....	550
23.7.1	Function pin switching (PWMy_PS).....	550
23.7.2	Advanced PWM function pin selection register (PWMy_ETRPS) .....	551
23.7.3	Output Enable Register (PWMy_ENO) .....	551
23.7.4	Output Additional Enable Register (PWMy_IOAUX).....	552
23.7.5	Control Register 1 (PWMy_CR1) .....	553
23.7.6	Control register 2 (PWMy_CR2), and real-time trigger ADC.....	555
23.7.7	Slave Mode Control Register (PWMy_SMCR).....	556
23.7.8	External Trigger Register (PWMy_ETR) .....	558
23.7.9	Interrupt Enable Register (PWMy_IER) .....	559
23.7.10	Status Register 1 (PWMy_SR1).....	559
23.7.11	Status Register 2 (PWMy_SR2).....	560
23.7.12	Event Generation Register (PWMy_EGR) .....	560
23.7.13	Capture/Compare Mode Register 1 (PWMy_CCMR1).....	561
23.7.14	Capture/Compare Mode Register 2 (PWMy_CCMR2).....	564
23.7.15	Capture/Compare Mode Register 3 (PWMy_CCMR3).....	565
23.7.16	Capture/Compare Mode Register 4 (PWMy_CCMR4).....	566
23.7.17	Capture/Compare Enable Register 1 (PWMy_CCER1).....	567
23.7.18	Capture/Compare Enable Register 2 (PWMy_CCER2).....	568
23.7.19	Counter High 8 Bits (PWMy_CNTRH).....	569
23.7.20	Counter low 8 bits (PWMy_CNTRL) .....	569
23.7.21	High 8 bits of prescaler (PWMy_PSCRH), output frequency calculation formula.....	569
23.7.22	Prescaler Lower 8 Bits (PWMy_PSCRL) .....	570
23.7.23	Auto-reload register upper 8 bits (PWMy_ARRH).....	570
23.7.24	Auto-reload register lower 8 bits (PWMy_ARRL).....	570
23.7.25	Repeat Counter Register (PWMy_RCR) .....	570
23.7.26	Capture/Compare Register 1/5 Upper 8 Bits (PWMy_CCR1H).....	570
23.7.27	Capture/Compare Register 1/5 Lower 8 Bits (PWMy_CCR1L).....	571
23.7.28	Capture/Compare Register 2/6 Upper 8 Bits (PWMy_CCR2H).....	571
23.7.29	Capture/Compare Register 2/6 Lower 8 Bits (PWMy_CCR2L).....	571
23.7.30	Capture/Compare Register 3/7 Upper 8 Bits (PWMy_CCR3H).....	571
23.7.31	Capture/Compare Register 3/7 Lower 8 Bits (PWMy_CCR3L).....	571

23.7.32	Capture/Compare Register 4/8 Upper 8 Bits (PWMr_CCR4H).....	572
23.7.33	Capture/Compare Register 4/8 Lower 8 Bits (PWMr_CCR4L).....	572
23.7.34	Brake Register (PWMr_BKR) .....	572
23.7.35	Dead Time Register (PWMr_DTR).....	573
23.7.36	Output Idle Status Register (PWMr_OISR).....	573
23.8	Sample Program .....	575
23.8.1	BLDC brushless DC motor (with HALL).....	575
23.8.2	BLDC brushless DC motor drive (without HALL) .....	585
23.8.3	Implementing an encoder using advanced PWM.....	593
23.8.4	Quadrature Encoder Mode .....	596
23.8.5	Single-pulse mode (trigger control pulse output) .....	597
23.8.6	GATE MODE (INPUT LEVEL ENABLE COUNTER) .....	598
23.8.7	External Clock Mode.....	600
23.8.8	Input capture mode to measure pulse period (capture rising edge to rising edge or falling edge to falling edge) .....	602
23.8.9	Input capture mode measurement pulse high width (capture rising edge to falling edge) .....	604
23.8.10	Input capture mode measurement pulse low width (capture falling edge to rising edge) .....	606
23.8.11	Input Capture Mode Simultaneous Measurement of Pulse Period and Duty Cycle .....	608
23.8.12	Capture the period and duty cycle of 4 input signals at the same time.....	610
23.8.13	The method of outputting PWM waveform with duty cycle of 100% and 0% (taking PWM1P as an example) .....	616
23.8.14	PWM Complementary Outputs with Dead-Time Control.....	617
23.8.15	PWM port as external interrupt (falling edge interrupt or rising edge interrupt).....	618
23.8.16	Output waveform with arbitrary period and arbitrary duty cycle .....	619
23.8.17	Use CEN of PWM to start PWMA timer and trigger ADC in real time.....	620
23.8.18	PWM Period Repeated Triggering of ADC.....	621
23.8.19	Reference circuit diagram for implementing 16-bit DAC with PWM.....	622
23.8.20	Complementary SPWM with PWM.....	622
23.8.21	Advanced PWM Output - Frequency Adjustable - Pulse Counting.....	625
	High Speed Advanced PWM (HSPWM) .....	629
24.1	Related Registers.....	629
24.1.1	HSPWM Configuration Register (HSPWMn_CFG).....	629
24.1.2	HSPWM Address Register (HSPWMn_AD).....	629
24.1.3	HSPWM Data Register (HSPWMn_DAT).....	630
24.2	Sample Program .....	631
24.2.1	Enabling High-Speed Mode with Advanced PWM (Asynchronous Mode).....	631
25	<b>USB Universal Serial Bus .....</b>	<b>634</b>
25.1	USB Related Registers.....	634
25.1.1	USB Control Register (USBCON).....	634
25.1.2	USB Clock Control Register (USBCLK).....	635
25.1.3	USB ADDRESS REGISTER (USBADR) .....	636
25.1.4	USB Indirect Address Data Register (USBDAT).....	636
25.2	USB Controller Register (SIE) .....	636
25.2.1	USB Function Address Register (FADDR).....	637
25.2.2	USB Power Control Register (POWER) .....	637
25.2.3	USB Endpoint IN Interrupt Flag (INTRIN1).....	638

25.2.4	USB Endpoint OUT Interrupt Flag (INTROUT1).....	638
25.2.5	USB Power Interrupt Flag (INTRUSB) .....	639
25.2.6	USB Endpoint IN Interrupt Enable Register (INTRIN1E) .....	639
25.2.7	USB Endpoint OUT Interrupt Enable Register (INTROUT1E) .....	640
25.2.8	USB Power Interrupt Enable Register (INTRUSB) .....	640
25.2.9	USB data frame number register (FRAMEn).....	641
25.2.10	USB Endpoint Index Register (INDEX).....	641
25.2.11	Maximum packet size for IN endpoints (INMAXP) .....	641
25.2.12	USB Endpoint 0 Control Status Register (CSR0).....	641
25.2.13	IN Endpoint Control Status Register 1 (INCSR1) .....	642
25.2.14	IN Endpoint Control Status Register 2 (INCSR2) .....	643
25.2.15	Maximum packet size for OUT endpoints (OUTMAXP) .....	643
25.2.16	OUT Endpoint Control Status Register 1 (OUTCSR1) .....	643
25.2.17	OUT Endpoint Control Status Register 2 (OUTCSR2) .....	644
25.2.18	OUT Length of USB Endpoint 0 (COUNT0) .....	644
25.2.19	OUT length of USB endpoints (OUTCOUNTn) .....	645
25.2.20	FIFO Data Access Registers (FIFOOn) for USB Endpoints .....	645
25.2.21	USB Trace Control Register (UTRKCTL).....	645
25.2.22	USB Trace Status Register (UTRKSTS).....	646
25.3	USB Product Development Considerations .....	646
25.4	Sample Program .....	647
25.4.1	HID Human Interface Device Example.....	647
25.4.2	HID (Human Interface Device) protocol example.....	658
25.4.3	CDC (Communication Device Class) protocol example.....	658
25.4.4	Example of USB keyboard based on HID protocol.....	658
25.4.5	Example of USB mouse based on HID protocol.....	658
25.4.6	Example based on WINUSB protocol.....	658
25.4.7	MSC (Mass Storage Class) protocol example .....	658
<b>26</b>	<b>RTC real-time clock.....</b>	<b>660</b>
26.1	RTC related registers .....	660
26.1.1	RTC Control Register (RTCCR).....	661
26.1.2	RTC Configuration Register (RTCCFG) .....	661
26.1.3	RTC Interrupt Enable Register (RTCIEN) .....	661
26.1.4	RTC Interrupt Request Register (RTCIF).....	662
26.1.5	RTC alarm setting register.....	662
26.1.6	RTC real-time clock initial value setting register.....	663
26.1.7	RTC real-time clock count register.....	663
26.2	RTC reference circuit diagram (without VBAT pin).....	665
26.3	RTC actual combat roadmap.....	666
26.4	Sample Program .....	667
26.4.1	Serial port print RTC clock example.....	667
<b>27</b>	<b>LCM interface (8/16 -bit color screen module I8080/M6800 interface) .....</b>	<b>670</b>
27.1	LCM interface function pin switch.....	670
27.2	LCM-related registers.....	670

27.2.1	LCM INTERFACE CONFIGURATION REGISTER (LCMIFCFG).....	671
27.2.2	LCM Interface Configuration Register 2 (LCMIFCFG2).....	671
27.2.3	LCM Interface Control Register (LCMIFCR).....	672
27.2.4	LCM Interface Status Register (LCMIFSTA).....	672
27.2.5	LCM Interface Data Registers (LCMIFDATL, LCMIFDATH) .....	672
27.3	I8080/M6800 Mode LCM Interface Timing Diagram.....	673
27.3.1	I8080 mode.....	673
27.3.2	M6800 mode.....	674
<b>28</b>	<b>DMA (Bulk Data Transfer) .....</b>	<b>675</b>
28.1	DMA-related registers.....	675
28.2	Memory-to-memory data read and write (M2M_DMA).....	680
28.2.1	M2M_DMA Configuration Register (DMA_M2M_CFG).....	680
28.2.2	M2M_DMA Control Register (DMA_M2M_CR).....	680
28.2.3	M2M_DMA Status Register (DMA_M2M_STA).....	680
28.2.4	M2M_DMA Transfer Total Byte Register (DMA_M2M_AMT).....	681
28.2.5	M2M_DMA Transfer Completion Byte Register (DMA_M2M_DONE) .....	681
28.2.6	M2M_DMA Transmit Address Register (DMA_M2M_TXAx) .....	681
28.2.7	M2M_DMA Receive Address Register (DMA_M2M_RXAx).....	681
28.3	ADC data automatic storage (ADC_DMA).....	682
28.3.1	ADC_DMA Configuration Register (DMA_ADC_CFG).....	682
28.3.2	ADC_DMA Control Register (DMA_ADC_CR) .....	682
28.3.3	ADC_DMA Status Register (DMA_ADC_STA) .....	682
28.3.4	ADC_DMA Receive Address Register (DMA_ADC_RXAx) .....	682
28.3.5	ADC_DMA Configuration Register 2 (DMA_ADC_CFG2).....	683
28.3.6	ADC_DMA Channel Enable Register (DMA_ADC_CHSWx) .....	683
28.3.7	Data storage format of ADC_DMA.....	684
28.4	Data exchange between SPI and memory (SPI_DMA) .....	686
28.4.1	SPI_DMA Configuration Register (DMA_SPI_CFG) .....	686
28.4.2	SPI_DMA Control Register (DMA_SPI_CR).....	686
28.4.3	SPI_DMA Status Register (DMA_SPI_STA).....	687
28.4.4	SPI_DMA Transfer Total Bytes Register (DMA_SPI_AMT) .....	687
28.4.5	SPI_DMA Transfer Completion Byte Register (DMA_SPI_DONE).....	687
28.4.6	SPI_DMA Transmit Address Register (DMA_SPI_TXAx).....	687
28.4.7	SPI_DMA Receive Address Register (DMA_SPI_RXAx).....	688
28.4.8	SPI_DMA Configuration Register 2 (DMA_SPI_CFG2) .....	688
28.5	Data exchange between serial port 1 and memory (UR1T_DMA, UR1R_DMA).....	689
28.5.1	UR1T_DMA Configuration Register (DMA_UR1T_CFG).....	689
28.5.2	UR1T_DMA Control Register (DMA_UR1T_CR) .....	689
28.5.3	UR1T_DMA Status Register (DMA_UR1T_STA) .....	689
28.5.4	UR1T_DMA Transfer Total Bytes Register (DMA_UR1T_AMT).....	690
28.5.5	UR1T_DMA Transfer Complete Byte Register (DMA_UR1T_DONE).....	690
28.5.6	UR1T_DMA Transmit Address Register (DMA_UR1T_TXAx) .....	690
28.5.7	UR1R_DMA Configuration Register (DMA_UR1R_CFG) .....	690
28.5.8	UR1R_DMA Control Register (DMA_UR1R_CR).....	691

28.5.9 UR1R_DMA Status Register (DMA_UR1R_STA).....	691
28.5.10 UR1R_DMA Transfer Total Bytes Register (DMA_UR1R_AMT) .....	691
28.5.11 UR1R_DMA Transfer Complete Byte Register (DMA_UR1R_DONE) .....	691
28.5.12 UR1R_DMA Receive Address serial port 2 and <del>Register (DMA_UR1R_RXAx)</del> DMA.....	693
28.6.1 UR2T_DMA Configuration Register (DMA_UR2T_CFG) .....	693
28.6.2 UR2T_DMA Control Register (DMA_UR2T_CR) .....	693
28.6.3 UR2T_DMA Status Register (DMA_UR2T_STA) .....	693
28.6.4 UR2T_DMA Transfer Total Bytes Register (DMA_UR2T_AMT).....	694
28.6.5 UR2T_DMA Transfer Completion Word Section Register (DMA_UR2T_DONE).....	694
28.6.6 UR2T_DMA Transmit Address Register (DMA_UR2T_TXAx) .....	694
28.6.7 UR2R_DMA Configuration Register (DMA_UR2R_CFG) .....	694
28.6.8 UR2R_DMA Control Register (DMA_UR2R_CR).....	695
28.6.9 UR2R_DMA Status Register (DMA_UR2R_STA) .....	695
28.6.10 UR2R_DMA Transfer Total Bytes Register (DMA_UR2R_AMT) .....	695
28.6.11 UR2R_DMA transfer done byte register (DMA_UR2R_DONE) .....	695
28.6.12 UR2R_DMA Receive Address Register (DMA_UR2R_RXAx) .....	695
28.7 port 3 and memory (UR3T_DMA, UR3R_DMA).....	697
28.7.1 UR3T_DMA Configuration Register (DMA_UR3T_CFG) .....	697
28.7.2 UR3T_DMA Control Register (DMA_UR3T_CR) .....	697
28.7.3 UR3T_DMA Status Register (DMA_UR3T_STA) .....	697
28.7.4 UR3T_DMA Transfer Total Bytes Register (DMA_UR3T_AMT).....	698
28.7.5 UR3T_DMA Transfer Done Byte Register (DMA_UR3T_DONE).....	698
28.7.6 UR3T_DMA Transmit Address Register (DMA_UR3T_TXAx) .....	698
28.7.7 UR3R_DMA Configuration Register (DMA_UR3R_CFG) .....	698
28.7.8 UR3R_DMA Control Register (DMA_UR3R_CR).....	699
28.7.9 UR3R_DMA Status Register (DMA_UR3R_STA).....	699
28.7.10 UR3R_DMA Transfer Total Bytes Register (DMA_UR3R_AMT) .....	699
28.7.11 UR3R_DMA Transfer Complete Byte Register (DMA_UR3R_DONE) .....	699
28.7.12 UR3R_DMA Receive Address Register (DMA_UR3R_RXAx) .....	699
28.8 Data exchange between serial port 4 and memory (UR4T_DMA, UR4R_DMA).....	701
28.8.1 UR4T_DMA Configuration Register (DMA_UR4T_CFG) .....	701
28.8.2 UR4T_DMA Control Register (DMA_UR4T_CR) .....	701
28.8.3 UR4T_DMA Status Register (DMA_UR4T_STA) .....	701
28.8.4 UR4T_DMA Transfer Total Byte Register (DMA_UR4T_AMT) .....	702
28.8.5 UR4T_DMA Transfer Complete Byte Register (DMA_UR4T_DONE) .....	702
28.8.6 UR4T_DMA Transmit Address Register (DMA_UR4T_TXAx) .....	702
28.8.7 UR4R_DMA Configuration Register (DMA_UR4R_CFG) .....	702
28.8.8 UR4R_DMA Control Register (DMA_UR4R_CR) .....	703
28.8.9 UR4R_DMA Status Register (DMA_UR4R_STA) .....	703
28.8.10 UR4R_DMA Transfer Total Bytes Register (DMA_UR4R_AMT) .....	703
28.8.11 UR4R_DMA transfer complete byte register (DMA_UR4R_DONE) .....	703
28.8.12 UR4R_DMA Receive Address Register (DMA_UR4R_RXAx) .....	703
28.9 Register (DMA_UR4R_RXAx).....	703
28.9.1 Data read and write between LCM and memory.....	703

28.9.1 LCM_DMA Configuration Register (DMA_LCM_CFG) .....	705
28.9.2 LCM_DMA Control Register (DMA_LCM_CR).....	705 28.9.3
LCM_DMA Status Register ( DMA_LCM_STA).....	706 28.9.4 LCM_DMA
Transfer Total Bytes Register (DMA_LCM_AMT) .....	706 28.9.5
LCM_DMA Transfer Completion Byte Register (DMA_LCM_DONE) .....	706 28.9.6
LCM_DMA Transmit Address Register (DMA_LCM_TXAx) .....	706 28.9.7
LCM_DMA Receive Address Register (DMA_LCM_RXAx). ....	706 Data exchange
28.10        between I2C and memory (I2CT_DMA, I2CR_DMA).....	707 28.10.1 I2CT_DMA
Configuration Register (DMA_I2CT_CFG).....	707 28.10.2 I2CT_DMA
Control Register (DMA_I2CT_CR) .....	707 28.10.3 I2CT_DMA
status Register (DMA_I2CT_STA) .....	707 28.10.4 I2CT_DMA
Transfer Total Byte Register (DMA_I2CT_AMT).....	708 28.10.5
I2CT_DMA Transfer Completion Byte Register (DMA_I2CT_DONE)..	708 28.10.6
I2CT_DMA Transmit Address Register (DMA_I2CT_TXAx) .....	708
28.10.7 I2CR_DMA Configuration Register (DMA_I2CR_CFG) .....	708
28.10.8 I2CR_DMA Control Register (DMA_I2CR_CR).....	709
28.10.9 I2CR_DMA Status Register (DMA_I2CR_STA) .....	709 28.10.10
I2CR_DMA Transfer Total Byte Register (DMA_I2CR_AMT) .....	709
28.10.11 I2CR_DMA Transfer Done Byte Register (DMA_I2CR_DONE) .....	709
28.10.12 I2CR_DMA Receive Address Register (DMA_I2CR_RXAx) .....	709
28.10.13 I2C_DMA Control Register (DMA_I2C_CR ) .....	710 28.10.14
I2C_DM A Status Register (DMA_I2C_ST).....	710 Data exchange
28.11        between I2S and memory (I2ST_DMA, I2SR_DMA).....	711 28.11.1
I2ST_DMA Configuration Register (DMA_I2ST_CFG).....	711
28.11.2 I2ST_DMA Control Register (DMA_I2ST_CR) .....	711
28.11.3 I2ST_DMA Status Register (DMA_I2ST_STA) .....	711
28.11.4 I2ST_DMA Transfer Total Byte Register (DMA_I2ST_AMT).....	712 28.11.5
I2ST_DMA transfer done byte register (DMA_I2ST_DONE) .....	712 28.11.6 I2ST_DMA
Transmit Address Register (DMA_I2ST_TXAx) .....	712 28.11.7 I2SR_DMA
Configuration Register (DMA_I2SR_CFG) .....	712 28.11.8 I2SR_DMA
Control Register (DMA_I2SR_CR).....	713 28.11.9 I2SR_DMA Status
Register (DMA_I2SR_STA) .....	713 28.11.10 I2SR_DMA
Transfer Total Byte Register (DMA_I2SR_AMT) .....	713 28.11.11 I2SR_DMA
Transfer Done Byte Register (DMA_I2SR_DONE) .....	713 28.11.12
I2SR_DMA Receive Address Register (DMA_I2SR_RXAx).....	713
28.12        Sample Programs .....	
28.12.1 Serial port 1 interrupt mode and computer send and receive test - DMA receive timeout interrupt.....	715 28.12.2 Serial port 1 interrupt mode and computer sending and
29            receiving test - DMA data verification.....	720 CAN
29.1        bus .....	728 CAN
29.2        function pin switch .....	728
CAN-related registers .....	728
29.2.1 Auxiliary Register 2 (AUXR2) .....	729
29.2.2 CAN Bus Interrupt Control Register (CANICR).....	729 29.2 .3 CAN Bus A

29.2.4 CAN Bus Data Register (CANDR).....	730
29.3 CAN Internal Function Register .....	730
29.3.1 CAN Mode Register (MR) .....	731
29.3.2 CAN Command Register (CMR).....	731
29.3.3 CAN Status Register (SR).....	732
29.3.4 CAN Interrupt/Acknowledge Register (ISR/IACK).....	732
29.3.5 CAN Interrupt Register (IMR).....	733
29.3.6 CAN Frame Receive Counter (RMC).....	733
29.3.7 CAN bus clock register 0 (BTR0).....	734
29.3.8 CAN Bus Clock Register 1 (BTR1).....	734
29.3.9 CAN bus data frame transmit buffer (TXBUFn).....	734
29.3.10 CAN bus data frame receive buffer (RXBUFn) .....	734
29.3.11 CAN Bus Acceptance Code Register (ACRn).....	735
29.3.12 CAN Bus Acceptance Mask Register (AMRn).....	735
29.3.13 CAN Bus Error Information Register (ECC).....	738
29.3.14 CAN Bus Receive Error Counter (RXERR) .....	738
29.3.15 CAN bus transmit error counter (TXERR).....	738
29.3.16 CAN Bus Arbitration Loss Register (ALC).....	739
29.4 Sample Program .....	740
29.4.1 CAN bus frame format.....	740
29.4.2 Example of CAN bus standard frame transmission and reception.....	741
29.4.3 Example of sending and receiving extended frame on CAN bus.....	744
29.4.4 CAN Bus Standard Frame Transceiver Test (Assembly).....	748
<b>30 LIN bus.....</b>	<b>757</b>
30.1 LIN function pin switch.....	757
30.2 LIN related registers.....	757
30.2.1 Auxiliary Register 2 (AUXR2).....	757
30.2.2 LIN Bus Interrupt Control Register (LINICR) .....	757
30.2.3 LIN Bus Address Register (LINAR) .....	758
30.2.4 LIN Bus Data Register (LINDR) .....	758
30.3 LIN Internal Function Register .....	758
30.3.1 LIN Data Register (LBUF) .....	759
30.3.2 LIN Data Address Register (LSEL) .....	759
30.3.3 LIN Frame ID Register (LID) .....	759
30.3.4 LIN Error Register (LER).....	759
30.3.5 LIN Interrupt Enable Register (LIE) .....	760
30.3.6 LIN Status Register (Read Only Register) (LSR) .....	760
30.3.7 LIN Control Register (Write-Only Register) (LCR) .....	760
30.3.8 LIN Baud Rate Register (DLL/DLH) .....	761
30.3.9 LIN Header Delay Count Register (HDRL/HDRH) .....	761
30.3.10 LIN Header Delay Divider Register (HDP).....	761
30.4 Sample Program .....	762
30.4.1 LIN Bus Master Transceiver Example .....	762
30.4.2 LIN Bus Slave Transceiver Example.....	766

30.4.3	Example of simulating a LIN bus using a serial port.....	770
30.4.4	Introduction to LIN Bus Timing.....	775
<b>31</b>	<b>32-bit hardware multiplication and division unit (MDU32) .....</b>	<b>778</b>
31.1	Associated Special Function Registers.....	778
31.2	Operation Execution Schedule .....	778
31.3	MDU32 Arithmetic Operations .....	779
31.3.1	32-bit multiplication.....	779
31.3.2	32-bit unsigned division.....	779
31.3.3	32-bit signed division.....	779
31.4	Sample Program .....	780
<b>32</b>	<b>Single-Precision Floating Point Unit (FPMU) .....</b>	<b>783</b>
32.1	Introduction to FPMU Floating Point Operators.....	783
32.2	Associated Special Function Registers.....	783
32.3	Operation Execution Schedule .....	783
32.4	FPMU Basic Arithmetic Operations.....	785
32.4.1	Floating point addition (+) .....	785
32.4.2	Floating point subtraction (-) .....	785
32.4.3	Floating point multiplication (x) .....	785
32.4.4	Floating point division (÷) .....	785
32.4.5	Floating point square root/square root (sqrt) .....	786
32.4.6	Floating point comparison (comp) .....	786
32.4.7	Floating point detection (check) .....	786
32.5	FPMU Trigonometric Functions.....	788
32.5.1	The sine function (sin) .....	788
32.5.2	Cosine function (cos) .....	788
32.5.3	Tangent function (tan) .....	788
32.5.4	The arc tangent function (arctan) .....	788
32.6	FPMU data conversion operations.....	789
32.6.1	Floating point to 8-bit integer (float ÿ char).....	789
32.6.2	Converting a floating point number to a 16-bit integer (float ÿ short).....	789
32.6.3	Converting a floating point number to a 32-bit integer (float ÿ long).....	789
32.6.4	8-bit integer to floating point number (char ÿ float).....	789
32.6.5	16-bit integer to floating point number (short ÿ float).....	790
32.6.6	32-bit integer to floating point number (long ÿ float).....	790
32.7	FPMU coprocessor control operations.....	791
32.7.1	Initializing the coprocessor .....	791
32.7.2	Clear exception .....	791
32.7.3	Read Status Register .....	791
32.7.4	Write Status Register .....	791
32.7.5	Read Control Register .....	791
32.7.6	Write Control Register .....	791
32.8	Sample Program .....	792
<b>Appendix A</b>	<b>Instruction Set.....</b>	<b>795</b>
A.1	Introduction to the instruction set .....	795

A.1.1	BINARY MODE AND SOURCE MODE .....	795
A.1.2	Instruction Set Flags.....	795
A.1.3	Instruction List (Order of Functions) .....	796
A.1.4	Instruction List (Machine Code Sorting) .....	803
A.2	Instruction details .....	807
<b>Appendix B</b>	<b>Fundamentals of Logical Algebra .....</b>	<b>889</b>
B.1	Number Systems and Coding .....	889
B.1.1	Number system conversion.....	889
B.1.2	Original code, inverse code and complement code.....	893
B.1.3	Common Codes .....	893
B.2	Several commonly used logical operations and their graphic symbols.....	894
Appendix C ,	<b>ISP download sample program for STC32G series MCU using third-party MCU .....</b>	<b>897</b>
Appendix D ,	Serial port interrupt sending and receiving - MODBUS protocol .....	904
Appendix E ,	<b>About whether to bake before reflow .....</b>	<b>914</b>
Appendix F ,	How to use a multimeter to detect whether the chip I/O port is good or not.....	915
Appendix G ,	<b>Mass production, how to save specialized programming personnel, and how to have no programming link.....</b>	<b>916</b>
Appendix H ,	A note on the <b>0xFD</b> problem in <b>Keil</b> software .....	917
Appendix I ,	How to make and edit EEPROM files with <b>STC-ISP</b> download software .....	918
Appendix J ,	<b>STC32</b> series header file definition .....	919
Appendix K ,	<b>Electrical Characteristics.....</b>	<b>948</b>
Appendix L	<b>update record.....</b>	<b>952</b>

# 1 Overview

STC32G series microcontrollers are microcontrollers that do not require external crystal oscillators and external resets.

As the target 32-bit 8051 MCU, under the same operating frequency, the STC32G series MCU is about 70 times faster than the traditional 8051.

STC32G series microcontrollers are single-clock (1T) microcontrollers produced by STC. They are wide voltage/high speed/high reliability/low power consumption/strong antistatic/

A new generation of 32-bit 8051 microcontroller with strong anti-interference, super encryption.

MCU integrates high-precision R/C clock ( $\pm 0.3\%$ ,  $+25^\circ$  at room temperature),  $-1.38\% \sim +1.42\%$  temperature drift ( $-40^\circ \sim +85^\circ$ ),

$-0.88\% \pm 1.05\%$  temperature drift ( $-20^\circ \sim +65^\circ$ ). 4MHz~33MHz wide range can be set during ISP programming, which can completely save external expensive

Crystal oscillator and external reset circuit (high reliability reset circuit has been integrated inside, 4-level reset threshold voltage is optional during ISP programming).

There are 4 optional clock sources inside the MCU: the internal high-precision IRC clock (the frequency can be adjusted during ISP programming), the internal 32KHz low

High-speed IRC, external 4M~33M crystal oscillator or external clock signal and internal PLL output clock. The clock source can be freely selected in the user code, when

After the clock source is selected, it can be divided by an 8-bit frequency divider, and then the clock signal can be provided to the CPU and various peripherals (such as timer, serial port, SPI, etc.).  
Wait).

The MCU provides two low-power modes: IDLE mode and STOP mode. In IDLE mode, the MCU stops supplying the clock to the CPU,

The CPU has no clock, and the CPU stops executing instructions, but all peripherals are still working, and the power consumption is about 1.3mA (6MHz operating frequency).

Rate). The STOP mode is the main clock stop mode, that is, the traditional power-down mode/power-down mode/stop mode. At this time, the CPU and all peripherals

All stop working, and the power consumption can be reduced to less than 1uA.

The MCU provides a wealth of digital peripherals (4 serial ports, 5 timers, 2 groups for three-phase motor control capable of outputting complementary/symmetrical/  
16-bit advanced PWM timer with dead-time control signals and I2C, SPI, USB, CAN, LIN) interface and analog peripherals (SuperSpeed

12-bit ADC, comparator), can meet the design needs of the majority of users.

The STC32G series MCU has 268 powerful instructions, including 32-bit addition and subtraction instructions and 16-bit multiplication and division instructions. hardware expanded  
32-bit hardware multiplication and division unit MDU32 (including 32-bit by 32-bit and 32-bit by 32-bit).

The STC32G series microcontroller integrates an enhanced double data pointer. Through program control, the data pointer can be automatically incremented or decremented.  
Subtraction function and automatic switching function of two sets of data pointers.

Product	Taste	end	different	same	step	string	mouth	same	step	string	mouth	high	class	Compare	device	MDU32	ADC	I2C	SPI	USB	CAN	LIN	RS485	RS232	UART
STC32G12K128 series	60	2	2	5	15CH*12B	••••						• 2	•••••												
STC32G8K64 series	45	2	2	5	15CH*12B	••••						2	•••••												
STC32F12K60 Series	45	2	2	5	15CH*12B	••••••	2	••••••																	

## 2 Features, price and pins

### 2.1 STC32G12K128-LQFP64/LQFP48/LQFP32/PDIP40

#### 2.1.1 Features and Price

Selection price (no external crystal oscillator, no external reset, 12-bit ADC, 15 channels)

Features and Price												Available	
variable bytes	wake up after power off	9 bit	External pins can also wake up from power-down	8	There are 12 digits	, 4 levels	, yes	Spot goods					
STC32G12K64	1.9-5.5	64K	4K	8K	2	64K	60	yes	yes	2	2	3	yes
STC32G12K128	1.9-5.5	128K	4K	8K	2	IAP	60	yes	yes	2	2	3	yes

Kernel

Ultra-high-speed 32-bit 8051 core (1T), about 70 times faster than traditional 8051

49 interrupt sources, 4 levels of interrupt priority

Support online simulation

Working voltage

1.9V~5.5V (When the working temperature is lower than -40°C, the working voltage shall not be lower than 3.0V)

Operating

temperature -40°C~85°C (internal high-speed IRC (36MHz or below) and external crystal can be used)

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

Flash memory

Maximum 128K bytes FLASH program memory (ROM), used to store user code

Support user-configured EEPROM size, 512-byte single page erasing, and erasing times can reach more than 100,000 times

Support hardware USB direct download and common serial port download

Support hardware SWD real-time simulation, P3.0/P3.1 (requires STC-USB Link1 tool)

SRAM, a total of 12K bytes

4K bytes internal SRAM (edata) ~ 8K bytes

internal expansion RAM (internal xdata)

ÿ Usage Note: (It is strongly recommended not to use **idata** and **pdata** to declare variables)

#### ÿ Clock control

ÿ Internal high precision IRC (up and down adjustment during ISP programming)

ÿ Error ±0.3% (25°C at room temperature)

ÿ -1.35%+1.30% temperature drift (full temperature range, -40°C~85°C)

ÿ -0.76%+0.98% temperature drift (temperature range, -20°C~65°C)

ÿ Internal 32KHz low speed IRC (large error)

ÿ External crystal oscillator (4MHz ~ 33MHz) and external clock, there is a special external clock to interfere with the internal circuit, which can be started by software

ÿ Internal PLL output clock (Note: 96MHz/144MHz output by PLL can be independently used as the clock source of high-speed PWM and high-speed SPI)

Users can freely choose the above 4 clock sources

#### ÿ Reset

ÿ Hardware reset

ÿ Power-on reset, the reset voltage is 1.7V~1.9V. (effective when the chip does not enable the low-voltage reset function)

ÿ Reset pin reset, P5.4 is the I/O port by default when leaving the factory, P5.4 pin can be set as the reset pin during ISP download (Note: when setting P5.4

When the pin is a reset pin, the reset level is low)

ÿ Watchdog overflow reset

ÿ Low-voltage detection reset, providing 4-level low-voltage detection voltage: 2.0V, 2.4V, 2.7V, 3.0V.

ÿ Software reset

ÿ Software write reset trigger register

#### ÿ Interrupt

ÿ Provide 49 interrupt sources: INT0, INT1, INT2, INT3, INT4, timer 0, timer 1, timer 2, timer 3, timer 4,

USART1, USART2, UART3, UART4, ADC analog-to-digital conversion, LVD low voltage detection, SPI, I2C, comparator, PWMA,

PWMB, USB, CAN, CAN2, LIN, LCMIF color screen interface interrupt, RTC real-time clock, all I/O interrupts (8 groups),

Serial port 1 DMA receive and send interrupt, serial port 2 DMA receive and send interrupt, serial port 3 DMA receive and send interrupt, serial

Port 4 DMA receive and transmit interrupt, I2C DMA receive and transmit interrupt, SPI DMA interrupt, ADC DMA interrupt,

LCD driver DMA interrupt and memory-to-memory DMA interrupt.

ÿ Provides 4 levels of interrupt priority

#### ÿ Digital Peripherals

ÿ Five 16-bit timers: timer 0, timer 1, timer 2, timer 3, timer 4, of which mode 3 of timer 0 has NMI

(non-maskable interrupt) function, mode 0 of timer 0 and timer 1 is 16-bit auto-reload mode

ÿ 2 high-speed synchronous/asynchronous serial ports: serial port 1 (USART1), serial port 2 (USART2), the fastest baud rate clock source can be FOSC/4. support

Synchronous serial port mode, asynchronous serial port mode, SPI mode, LIN mode, infrared mode (IrDA), smart card mode (ISO7816)

ÿ 2 high-speed asynchronous serial ports: serial port 3, serial port 4, the fastest baud rate clock source can be FOSC/4

ÿ 2 sets of advanced PWM, can realize 8-channel (4 sets of complementary symmetry) PWM with dead zone control, and support external abnormal detection function

ÿ SPI: 3 sets of hardware SPI (one independent SPI, two USART SPI modes) support master mode and slave mode and master/slave automatic

Switching (Note: a group of independent SPI can support DMA, and the SPI of two groups of USART does not support DMA)

ÿ I2C: Support master mode and slave mode

ÿ ICE: Hardware support emulation

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

ÿ USB: USB2.0/USB1.1 compatible with full-speed USB, 6 bidirectional endpoints, support 4 endpoint transfer modes (control transfer, interrupt transfer, batch transfer

transfers and isochronous transfers), each endpoint has a 64-byte buffer

ÿ CAN: Two independent CAN 2.0 control units

ÿ LIN: 3 sets of hardware LIN (one set of independent LIN, two sets of USART LIN mode) an independent LIN control unit (support 1.3 and 2.1

Version)

- ÿ MDU32: Hardware 32-bit multiplier and divider (including 32-bit divided by 32-bit, 32-bit multiplied by 32-bit)
- ÿ I/O port interrupt: All I/O supports interrupt, each group of I/O interrupt has independent interrupt entry address, all I/O interrupt can support 4 kinds of interrupt
  - Mode: high level interrupt, low level interrupt, rising edge interrupt, falling edge interrupt. The I/O port interrupt can be wake-up from power-down, and there are 4 levels in the interrupt priority.
- ÿ LCD driver module: support 8080 and 6800 interfaces and 8-bit and 16-bit data width

Data to memory, serial port 1/2/3/4 to receive data to memory, serial port 1/2/3/4 to send data from memory, ADC automatic sampling data to Memory (simultaneous averaging), LCD driver sending data from memory, and copying data from memory to memory

- ÿ Hardware digital ID: support 32 bytes

#### ÿ Analog peripherals

- ÿ ADC: Ultra-high-speed ADC, supports 12-bit high-precision 15-channel (channel 0 to channel 14) analog-to-digital conversion, and channel 15 of ADC is used for testing
  - Internal reference voltage (when the chip is shipped from the factory, the internal reference voltage is adjusted to 1.19V, with an error of ±1%)

- ÿ Comparator: A set of comparators

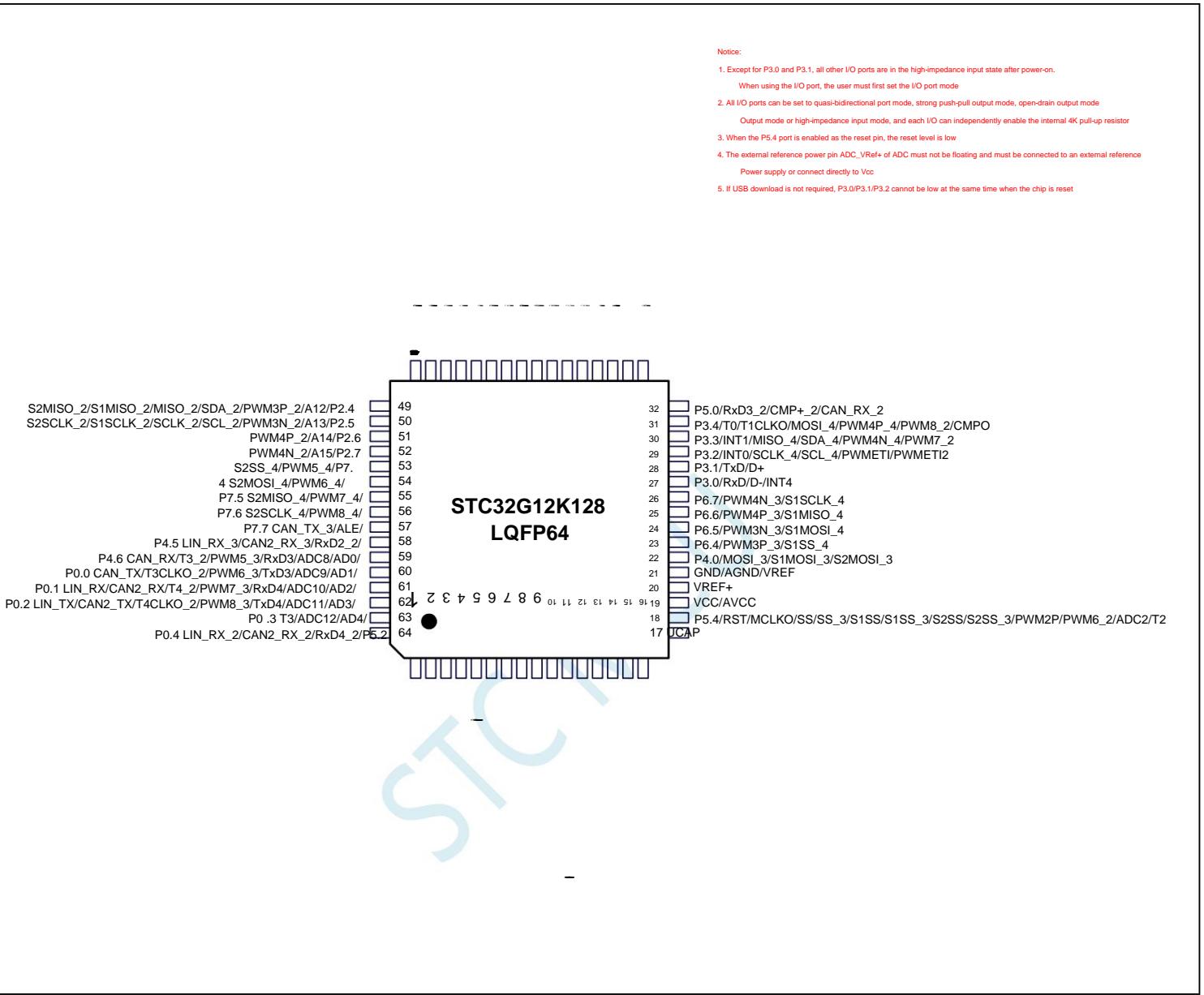
#### ÿ GPIO

- ÿ Up to 60 GPIOs: P0.0~P0.7, P1.0~ P1.7 (without 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 4 modes: quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode, high-impedance input mode
  - Except for P3.0 and P3.1, all other IO ports are in the high-impedance input state after power-on. The user must set the IO port first when using the IO port.
- ÿ In addition, each I/O can independently enable the internal 4K pull-up resistor

#### ÿ Packaging

- ÿ LQFP64, LQDP48, LQFP32, PDIP40

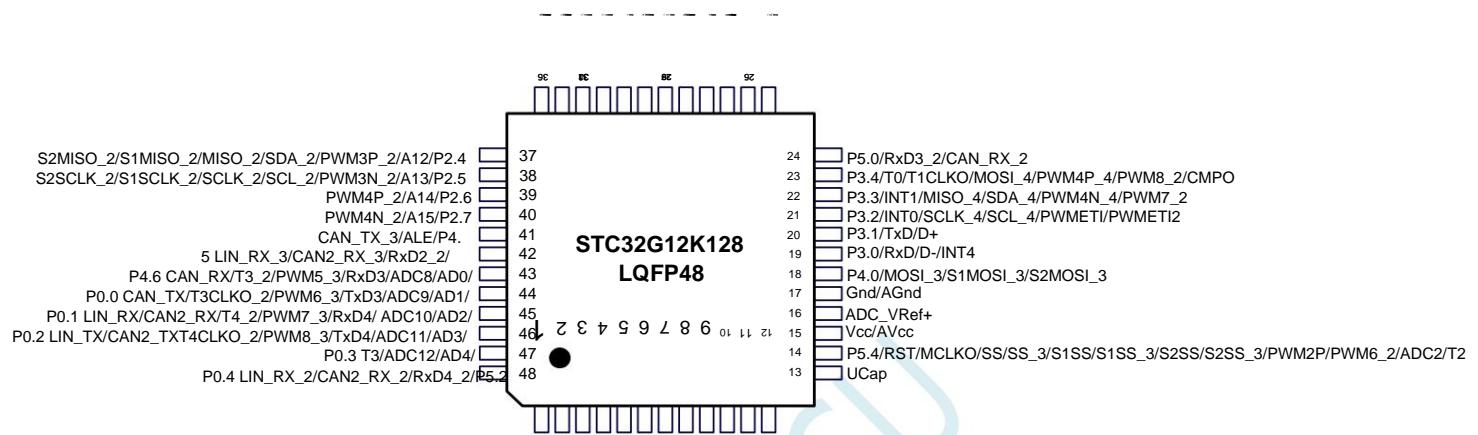
## 2.1.2 Pin Diagram, Minimum System



Looking at the chip silk screen The small dot at the bottom left is the first foot. Looking at the chip silk screen The last letter on the bottom line is the chip version number

## Notice:

1. Except for P3.0 and P3.1, all other I/O ports are in the high-impedance input state after power-on.  
When using the I/O port, the user must first set the I/O port mode
2. All I/O ports can be set to quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode  
Output mode or high-impedance input mode, and each I/O can independently enable the internal 4K pull-up resistor
3. When the P5.4 port is enabled as the reset pin, the reset level is low
4. The external reference power pin ADC\_VRef+ of ADC must not be floating and must be connected to an external reference  
Power supply or connect directly 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

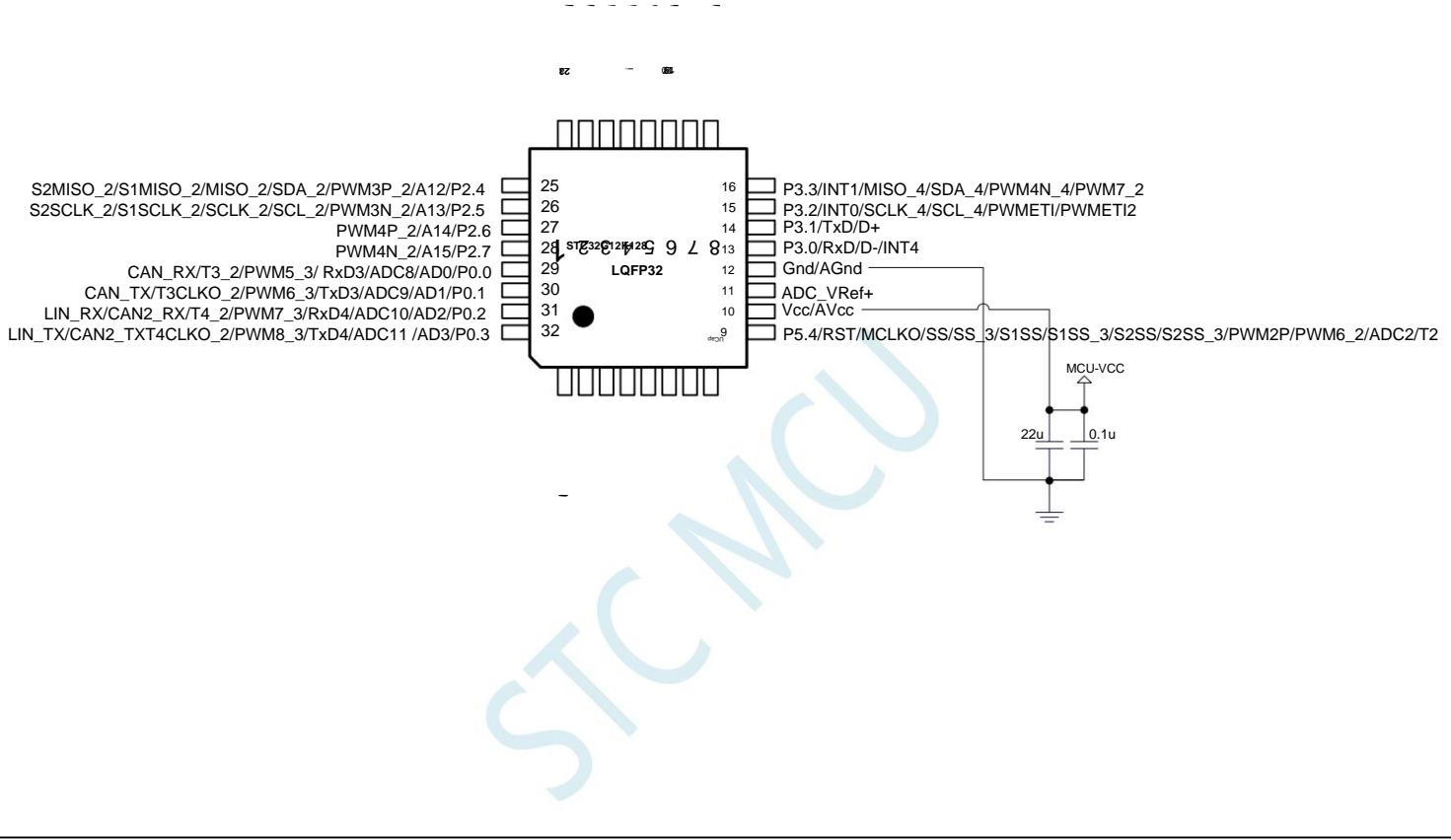


Looking at the chip silk screen The small dot at the bottom left is the  
first foot. Looking at the chip silk screen The last letter on the bottom line is the chip version number

### Hardware USB direct download reference circuit diagram

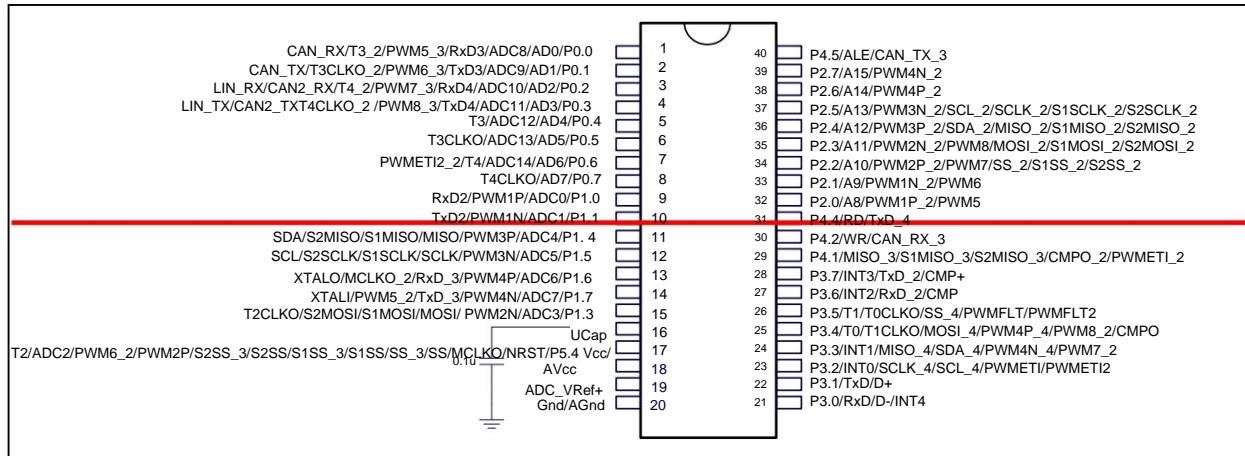
## Notice:

1. Except for P3.0 and P3.1, all other I/O ports are in the high-impedance input state after power-on.  
When using the I/O port, the user must first set the I/O port mode
2. All I/O ports can be set to quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode  
Output mode or high-impedance input mode, and each I/O can independently enable the internal 4K pull-up resistor
3. When the P5.4 port is enabled as the reset pin, the reset level is low
4. The external reference power pin ADC\_VRef+ of ADC must not be floating and must be connected to an external reference  
Power supply or connect directly 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 silk screen The small dot at the bottom left is the  
first foot. Looking at the chip silk screen The last letter on the bottom line is the chip version number

### Hardware USB direct download reference circuit diagram



### Hardware USB direct download reference circuit diagram

## 2.1.3 Pin description

Numbering				name	kind type	illustrate
LQFP64	LQFP48	LQFP32	PDI/P40			
1	1			P5.3	I/O Standard IO port	
				TxD4_2	O Send pin of serial port 4	
				CAN2_TX_2	O CAN2 bus transmit pin	
				LIN_TX_2	O LIN bus transmit pin	
2	2		6	P0.5	I/O Standard IO port	
				AD5	I address bus	
				ADC13	I ADC analog input channel 13	
				T3CLKO	O Timer 3 clock division output	
3	3		7	P0.6	I/O Standard IO port	
				AD6	I address bus	
				ADC14	I ADC analog input channel 14	
				T4	I Timer 4 external clock input	
				PWMFLT2_2	I External abnormal detection pin for enhanced PWM	
4	4		8	P0.7	I/O Standard IO port	
				AD7	I address bus	
				T4CLKO	O Timer 4 clock division output	
5				P6.0	I/O Standard IO port	
				PWM1P_3	I/O PWM1 capture input and pulse output positive	
6				P6.1	I/O Standard IO port	
				PWM1N_3	I/O Capture input and pulse output negative pole of PWM1	
7				P6.2	I/O Standard IO port	
				PWM2P_3	I/O PWM2 capture input and pulse output positive	
8				P6.3	I/O Standard IO port	
				PWM2N_3	I/O Capture input and pulse output negative pole of PWM2	
9	5	1	9	P1.0	I/O Standard IO port	
				ADC0	I ADC analog input channel 0	
				PWM1P	I/O PWM1 capture input and pulse output positive	
				RxD2	I Receive pin of serial port 2	
10	6	2	10	P1.1	I/O Standard IO port	
				ADC1	I ADC analog input channel 1	
				PWM1N	I/O PWM1 capture input and pulse output negative	
				TxD2	I Send pin of serial port 2	

Numbering				name type		illustrate
LQFP64	LQFP48	LQFP32	PDIP40			
11	7			P4.7	I/O Standard IO port	
				TxD2_2	I Send pin of serial port 2	
				CAN2_TX_3 O	CAN2 bus transmit pin	
				LIN_TX_3 O	LIN bus transmit pin	
12	8	3	11	P1.4	I/O Standard IO port	
				ADC4	I ADC analog input channel 4	
				PWM3P I/O PWM3	capture input and pulse output positive	
				MISO	I/O SPI Master Input Slave Output	
				S1MISO	I/O USART1 - SPI master input slave output	
				S2MISO	I/O USART2 - SPI master input slave output	
				SDA	Data lines for I/O I2C interface	
13	9	4	12	P1.5	I/O Standard IO port	
				ADC5	I ADC analog input channel 5	
				PWM3N I/O Capture	input and pulse output negative pole of PWM3	
				SCLK	I/O SPI clock pin	
				S1SCLK	I/O USART1 - SPI clock pin	
				S2SCLK	I/O USART2 - SPI clock pin	
				SCL	I/O I2C clock line	
14	10	5	13	P1.6	I/O Standard IO port	
				ADC6	I ADC analog input channel 6	
				RxD_3	I Receive pin of serial port 1	
				PWM4P I/O PWM4	capture input and pulse output positive	
				MCLKO_2 O	Main clock frequency division output	
				XTALO	O Output pin of external crystal oscillator	
15	11	6	14	P1.7	I/O Standard IO port	
				ADC7	I ADC analog input channel 7	
				TxD_3	O Send pin of serial port 1	
				PWM4N I/O Capture	input and pulse output negative pole of PWM4	
				PWM5_2 I/O Capture	input and pulse output of PWM5	
				XTALI	I Input pin of external crystal oscillator/external clock	
16	12	7	15	P1.3	I/O Standard IO port	
				ADC3	I ADC analog input channel 3	
				MOSI	I/O SPI master output slave input	
				S1MOSI	I/O USART1 - SPI master output slave input	
				S2MOSI	I/O USART2 - SPI master output slave input	
				PWM2N I/O Catch	input and pulse output negative pole of PWM2	
				T2CLKO	O Timer 2 clock division output	
17	13	8	16	UCAP	I USB core power regulator pin	

Numbering				name type		illustrate
LQFP64	LQFP48	LQFP32	PDIP40			
18	14	9	17	P5.4	I/O Standard IO port	
				RST	I reset pin	
				MCLK0	O Main clock frequency division output	
				SS_3	I SPI slave selection pin (master is output)	
				SS	I SPI slave selection pin (master is output)	
				S1SS_3	I USART1 - SPI slave selection pin (master is output)	
				S1SS	I USART1 - SPI slave selection pin (master is output)	
				S2SS_3	I USART2 Slave selection pin of SPI (master is output)	
				S2SS	I USART2 Slave selection pin of SPI (master is output)	
				PWM2P	I/O PWM2 capture input and pulse output positive	
				PWM6_2	I/O Capture input and pulse output of PWM6	
				T2	I Timer 2 external clock input	
				ADC2	I ADC analog input channel 2	
19	15	10	18	Vcc	VCC power pin	
				AVcc	VCC ADC power supply pin	
20	16	11	19	Vref+	I ADC reference voltage pin	
twenty one	17	12	20	Gnd	GND ground	
				Agnd	GND ADC ground	
				Vref-	I ADC reference voltage ground	
twenty two	18			P4.0	I/O Standard IO port	
				MOSI_3	I/O SPI master output slave input	
				S1MOSI_3	I/O USART1 - SPI master output slave input	
				S2MOSI_3	I/O USART2 - SPI master output slave input	
twenty three				P6.4	I/O Standard IO port	
				PWM3P_3	I/O Capture input and pulse output positive of PWM3	
				S1SS_4	I USART1 - SPI slave selection pin (master is output)	
twenty four				P6.5	I/O Standard IO port	
				PWM3N_3	I/O Capture input and pulse output negative pole of PWM3	
				S1MOSI_4	I/O USART1 - SPI master output slave input	
25				P6.6	I/O Standard IO port	
				PWM4P_3	I/O Capture input and pulse output positive of PWM4	
				S1MISO_4	I/O USART1 - SPI master input slave output	
26				P6.7	I/O Standard IO port	
				PWM4N_3	I/O Capture input and pulse output negative pole of PWM4	
				S1SCLK_4	I/O USART1 - SPI clock pin	
27	19	13	twenty one	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	

Numbering				name type		illustrate
LQFP64	LQFP48	LQFP32	PDIP40			
28	20	14	twenty two	P3.1	I/O Standard IO port	
				D+	I/O USB data port	
				TxD	O Send pin of serial port 1	
29	twenty one	15	twenty three	P3.2	I/O Standard IO port	
				INT0	I External interrupt 0	
				SCLK_4	I/O SPI clock pin	
				SCL_4	I/O I2C clock line	
				PWMETI	I PWM external trigger input pin	
				PWMETI2	I PWM external trigger input pin 2	
30	twenty two	16	twenty four	P3.3	I/O Standard IO port	
				INT1	I External interrupt 1	
				MISO_4	I/O SPI Master Input Slave Output	
				SDA_4	Data lines for I/O I2C interface	
				PWM4N_4	I/O Capture input and pulse output negative pole of PWM4	
				PWM7_2	I/O Capture input and pulse output of PWM7	
31	twenty three	17	25	P3.4	I/O Standard IO port	
				T0	I Timer 0 external clock input	
				T1CLKO	O Timer 1 clock division output	
				MOSI_4	I/O SPI master output slave input	
				PWM4P_4	I/O Capture input and pulse output positive of PWM4	
				PWM8_2	I/O Capture input and pulse output of PWM8	
				CMPO	O Comparator output	
32	twenty four			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 receiving pin	
33	25			P5.1	I/O Standard IO port	
				TxD3_2	O Send pin of serial port 3	
				CMP+_3	I Comparator positive input	
				CAN_TX_2	O CAN bus transmit pin	
34	26	18	26	P3.5	I/O Standard IO port	
				T1	I Timer 1 external clock input	
				T0CLKO	O Timer 0 clock divide output	
				SS_4	I SPI slave selection pin (master is output)	
				PWMFLT	I External abnormal detection pin for enhanced PWM	
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	

Numbering				name type		illustrate
LQFP64	LQFP48	LQFP32	PDIP40			
36	28	20	28	P3.7	I/O Standard IO port	
				INT3	I External interrupt 3	
				TxD_2	O Send pin of serial port 1	
				CMP+	I Comparator positive input	
37				P7.0	I/O Standard IO port	
				CAN_RX_4	I CAN bus receiving pin	
38				P7.1	I/O Standard IO port	
				CAN_TX_4	O CAN bus transmit pin	
39				P7.2	I/O Standard IO port	
				CAN2_RX_4	I CAN2 bus receiving pin	
				LIN_RX_4	I LIN bus receiving pin	
40				P7.3	I/O Standard IO port	
				CAN2_TX_4	O CAN2 bus transmit pin	
				LIN_TX_4	O LIN bus transmit pin	
				PWMETI_3	I PWM external trigger input pin	
41	29		29	P4.1	I/O Standard IO port	
				MISO_3	I/O SPI Master Input Slave Output	
				S1MISO_3	I/O USART1 - SPI master input slave output	
				S2MISO_3	I/O USART2 - SPI master 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 of external bus	
				CAN_RX_3	I CAN bus receiving pin	
43	31			P4.3	I/O Standard IO port	
				RxD_4	I Receive pin of serial port 1	
				SCLK_3	I/O SPI clock pin	
				S1SCLK_3	I/O USART1 - SPI clock pin	
				S2SCLK_3	I/O USART2 - SPI clock pin	
44	32		31	P4.4	I/O Standard IO port	
				RD	O read signal line of external bus	
				TxD_4	O Send pin of serial port 1	
45	33	Safety one	32	P2.0	I/O Standard IO port	
				A8	I address bus	
				PWM1P_2	I/O Capture input and pulse output positive of PWM1	
				PWM5	I/O Capture input and pulse output of PWM5	

Numbering				name type		illustrate
LQFP64	LQFP48	LQFP32	PDIP40			
46	34	Twenty two	33	P2.1	I/O Standard IO port	
				A9	I address bus	
				PWM1N_2 I/O	Capture input and pulse output negative pole of PWM1	
				PWM6	I/O Capture input and pulse output of PWM6	
47	35	Twenty three	34	P2.2	I/O Standard IO port	
				A10	I address bus	
				SS_2	I SPI slave select pin (master is output)	
				S1SS_2	I USART1 - SPI slave selection pin (master is output)	
				S2SS_2	I USART2 Slave selection pin of SPI (master is output)	
				PWM2P_2 I/O	Capture input and pulse output positive of PWM2	
				PWM7	I/O Capture input and pulse output of PWM7	
48	36	Twenty four	356	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 master output slave input	
				S2MOSI_2 I/O	USART2 - SPI master output slave input	
				PWM2N_2 I/O	Capture input and pulse output negative pole of PWM2	
				PWM8	I/O Capture input and pulse output of PWM8	
49	37	25	36	P2.4	I/O Standard IO port	
				A12	I address bus	
				MISO_2	I/O SPI Master Input Slave Output	
				S1MISO_2 I/O	USART1 - SPI master input slave output	
				S2MISO_2 I/O	USART2 - SPI master input slave output	
				SDA_2	Data lines for I/O I2C interface	
				PWM3P_2 I/O	Capture input and pulse output positive of PWM3	
50	38	26	37	P2.5	I/O Standard IO port	
				A13	I address bus	
				SCLK_2	I/O SPI clock pin	
				S1SCLK_2 I/O	USART1 - SPI clock pin	
				S2SCLK_2 I/O	USART2 - SPI clock pin	
				SCL_2	I/O I2C clock line	
				PWM3N_2 I/O	Capture input and pulse output negative pole of PWM3	
51	39	27	38	P2.6	I/O Standard IO port	
				A14	I address bus	
				PWM4P_2 I/O	Capture input and pulse output positive of PWM4	

Numbering				name type		illustrate
LQFP64	LQFP48	LQFP32	PDIP40			
52	40	28	39	P2.7	I/O Standard IO port	
				A15	I address bus	
				PWM4N_2	I/O PWM4 capture input and pulse output negative	
53				P7.4	I/O Standard IO port	
				PWM5_4	I/O Capture input and pulse output of PWM5	
				S2SS_4	I USART2 Slave selection pin of SPI (master is output)	
54				P7.5	I/O Standard IO port	
				PWM6_4	I/O Capture input and pulse output of PWM6	
				S2MOSI_4	I/O USART2 - SPI slave input master 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 master input slave output	
56				P7.7	I/O Standard IO port	
				PWM8_4	I/O Capture input and pulse output of PWM8	
				S2SCLK_4	I/O USART2 - SPI clock pin	
57	41		40	P4.5	I/O Standard IO port	
				ALE	O address latch signal	
				CAN_TX_3	O CAN bus send 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 receiving pin	
				LIN_RX_3	I LIN bus receiving pin	
59	43	29	1	P0.0	I/O Standard IO port	
				AD0	I address bus	
				ADC8	I ADC analog input channel 8	
				RxD3	I Receive pin of serial port 3	
				PWM5_3	I/O Capture input and pulse output of PWM5	
				CAN_RX	I CAN bus receiving pin	

Numbering				name type		illustrate
LQFP64	LQFP48	LQFP32	PDIP40			
60	44	30	2	P0.1	I/O Standard IO port	
				AD1	I address bus	
				ADC9	I ADC analog input channel 9	
				TxD3	O Send pin of serial port 3	
				PWM6_3	I/O Capture input and pulse output of PWM6	
				CAN_TX	O CAN bus send pin	
61	45	31	3	P0.2	I/O Standard IO port	
				AD2	I address bus	
				ADC10	I ADC analog 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 receiving pin	
				LIN_RX	I LIN bus receiving pin	
62	46	32	4	P0.3	I/O Standard IO port	
				AD3	I address bus	
				ADC11	I ADC analog input channel 11	
				TxD4	O Send pin of serial port 4	
				PWM8_3	I/O Capture input and pulse output of PWM8	
				CAN2_TX	O CAN2 bus send pin	
				LIN_TX	O LIN bus send pin	
63	47	5		P0.4	I/O Standard IO port	
				AD4	I address bus	
				ADC12	I ADC analog 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 receiving pin	
				LIN_RX_2	I LIN bus receiving pin	

## 2.2 STC32G8K64-LQFP48/LQFP32/PDIP40

### 2.2.1 Features and Price

ÿ Selection price (no external crystal oscillator, no external reset, 12-bit ADC, 15 channels)

	Traditional	Performance	Function	Hardware	Software	Development	Support	Price and Package	Availability	Sample delivery in April
STC32G6K48	1.9-5.5	48K 2K 6K 2 48K 45 yes yes 2 2 2 yes yes yes 5	variable byte Can do stack or variable bytes up after bit poweroff	Synchro	External pins can also wake up from power-down )	8 with 12 with 4 levels with yes yes yes yes yes yes \$4 ÿ ÿ ÿ	USE			
STC32G8K64	1.9-5.5	64K 2K 6K 2 IAP 45 yes yes 2 2 2 yes yes yes 5				8 with 12 with 4 levels with yes \$4 ÿ ÿ ÿ				

ÿ Kernel

ÿ Ultra-high-speed 32-bit 8051 core (1T), about 70 times faster than traditional 8051

ÿ 48 interrupt sources, 4 levels of interrupt priority

ÿ Support online simulation

ÿ Working voltage

ÿ 1.9V~5.5V (When the working temperature is lower than -40°C, the working voltage shall not be lower than 3.0V)

ÿ Working temperature

ÿ -40°C~85°C (internal high-speed IRC (36MHz or below) and external crystal can be used)

ÿ -40°C~125°C (When the temperature is higher than 85°C, please use an external high temperature resistant crystal oscillator, and the operating frequency should be controlled below 24MHz)

ÿ Flash memory

ÿ Maximum 64K bytes FLASH program memory (ROM) for storing user code

ÿ Support user-configured EEPROM size, 512-byte single page erasing, and erasing times can reach more than 100,000 times

ÿ Support hardware USB direct download and common serial port download

ÿ Support hardware SWD real-time simulation, P3.0/P3.1 (requires STC-USB Link1 tool)

ÿ SRAM, a total of 6K bytes

ÿ 2K bytes internal SRAM (edata)

ÿ 6K bytes internal extended RAM (internal xdata) **ÿ Usage note:**

(It is strongly recommended not to use **idata** and **pdata** to declare variables)

ÿ Clock control

- ÿ Internal high precision IRC (up and down adjustment during ISP programming)
    - ÿ Error  $\pm 0.3\%$  (25°C at room temperature)
    - ÿ  $-1.35\% \pm 1.30\%$  temperature drift (full temperature range, -40°C to 85°C)
    - ÿ  $-0.76\% \pm 0.98\%$  temperature drift (temperature range, -20°C to 65°C)
  - ÿ Internal 32KHz low speed IRC (large error)
  - ÿ External crystal oscillator (4MHz to 33MHz) and external clock
  - ÿ Internal PLL output clock (Note: 96MHz/144MHz output by PLL can be independently used as the clock source of high-speed PWM and high-speed SPI)
- Users can freely choose the above 4 clock sources

#### ÿ Reset

##### ÿ Hardware reset

- ÿ Power-on reset, the reset voltage is 1.7V~1.9V. (effective when the chip does not enable the low-voltage reset function)
- ÿ Reset pin reset, P5.4 is the I/O port by default when leaving the factory, P5.4 pin can be set as the reset pin during ISP download (Note: when setting P5.4)
  - When the pin is a reset pin, the reset level is low
- ÿ Watchdog overflow reset
  - ÿ Low-voltage detection reset, providing 4-level low-voltage detection voltage: 2.0V, 2.4V, 2.7V, 3.0V.

##### ÿ Software reset

##### ÿ Software write reset trigger register

#### ÿ Interrupt

- ÿ Provide 48 interrupt sources: INT0, INT1, INT2, INT3, INT4, timer 0, timer 1, timer 2, timer 3, timer 4,
  - USART1, USART2, UART3, UART4, ADC analog-to-digital conversion, LVD low voltage detection, SPI, I2C, comparator, PWMA, PWMB, CAN, CAN2, LIN, LCMIF color screen interface interrupt, RTC real-time clock, all I/O interrupts (8 groups), serial port 1
  - DMA receive and transmit interrupt of serial port 2, DMA receive and transmit interrupt of serial port 2, DMA receive and transmit interrupt of serial port 3, serial port 4
  - DMA receive and transmit interrupt, I2C DMA receive and transmit interrupt, SPI DMA interrupt, ADC DMA interrupt, LCD driver
  - Active DMA interrupts and memory-to-memory DMA interrupts.
- ÿ Provides 4 levels of interrupt priority

#### ÿ Digital Peripherals

- ÿ Five 16-bit timers: timer 0, timer 1, timer 2, timer 3, timer 4, of which mode 3 of timer 0 has NMI
  - (non-maskable interrupt) function, mode 0 of timer 0 and timer 1 is 16-bit auto-reload mode
- ÿ 2 high-speed synchronous/asynchronous serial ports: serial port 1 (USART1), serial port 2 (USART2), the fastest baud rate clock source can be FOSC/4. support
  - Synchronous serial port mode, asynchronous serial port mode, SPI mode, LIN mode, infrared mode (IrDA), smart card mode (ISO7816)

##### ÿ 2 high-speed asynchronous serial ports: serial port 3, serial port 4, the fastest baud rate clock source can be FOSC/4

##### ÿ 2 sets of advanced PWM, can realize 8-channel (4 sets of complementary symmetry) PWM with dead zone control, and support external abnormal detection function

##### ÿ SPI: Support master mode and slave mode and master/slave automatic switching

##### ÿ I2C: Support master mode and slave mode

##### ÿ ICE: Hardware support emulation

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

##### ÿ CAN: Two independent CAN 2.0 control units

##### ÿ LIN: An independent LIN control unit (supports versions 1.3 and 2.1), USART1 and USART2 can support two sets of LIN

##### ÿ MDU32: Hardware 32-bit multiplier and divider (including 32-bit divided by 32-bit, 32-bit multiplied by 32-bit)

##### ÿ I/O port interrupt: All I/O supports interrupt, each group of I/O interrupt has independent interrupt entry address, all I/O interrupt can support 4 kinds of interrupt

Mode: high level interrupt, low level interrupt, rising edge interrupt, falling edge interrupt. The I/O port interrupt can be wake-up from power-down, and there are 4 levels in the interrupt priority.

##### ÿ LCD driver module: supports 8080 and 6800 interfaces and 8-bit and 16-bit data width

##### ÿ DMA: Support SPI shift receive data to memory, SPI shift transmit data in memory, I2C transmit data in memory, I2C receive data

Data to memory, serial port 1/2/3/4 to receive data to memory, serial port 1/2/3/4 to send data from memory, ADC automatic sampling data to

Memory (simultaneous averaging), LCD driver sending data from memory, and copying data from memory to memory

ÿ Hardware digital ID: support 32 bytes

ÿ Analog peripherals

ÿ ADC: Ultra-high-speed ADC, supports 12-bit high-precision 15-channel (channel 0 to channel 14) analog-to-digital conversion, and channel 15 of ADC is used for testing

Internal reference voltage (when the chip is shipped from the factory, the internal reference voltage is adjusted to 1.19V, with an error of ±1%)

ÿ Comparator: A set of comparators

ÿ GPIO

ÿ 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

ÿ All GPIOs support the following 4 modes: quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode, high-impedance input mode

ÿ Except for P3.0 and P3.1, all other IO ports are in the high-impedance input state after power-on. The user must set the IO port first when using the IO port.

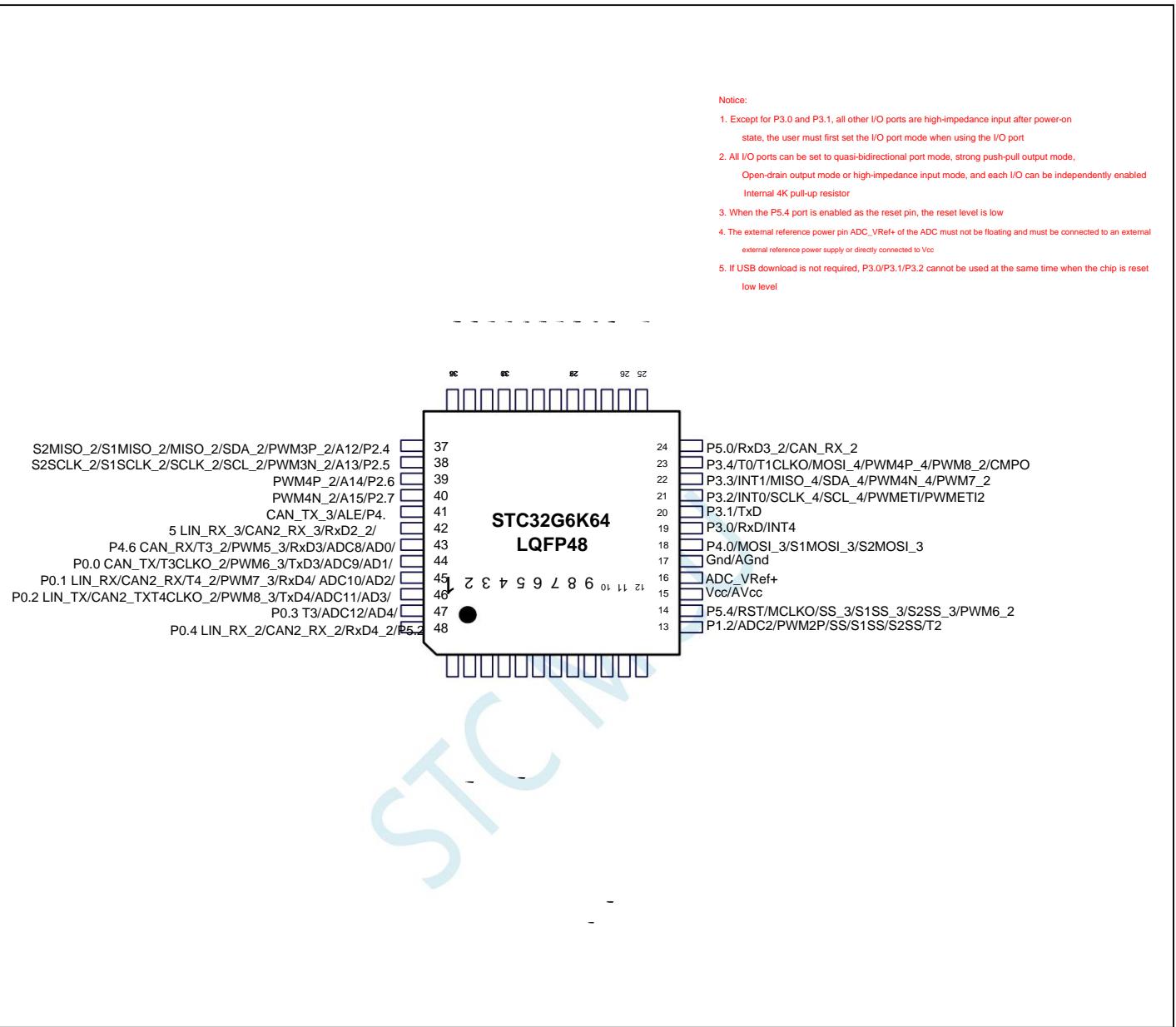
model

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

ÿ Packaging

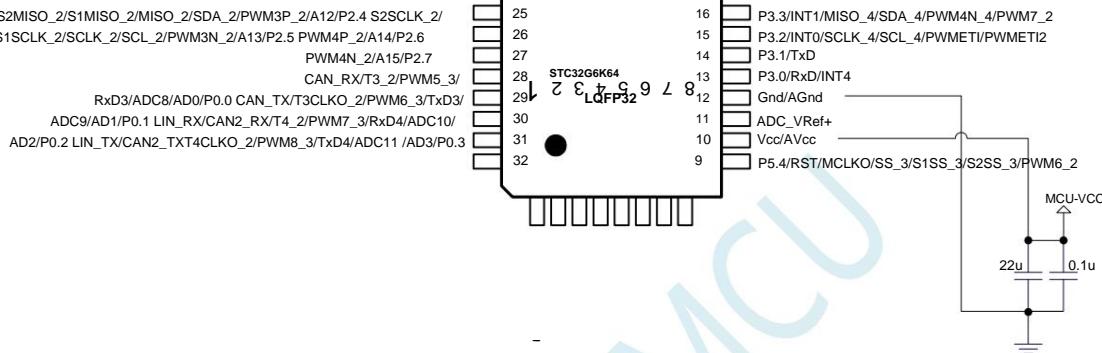
ÿ LQFP48, LQFP32, PDIP40

## 2.2.2 Pin Diagram, Minimum System

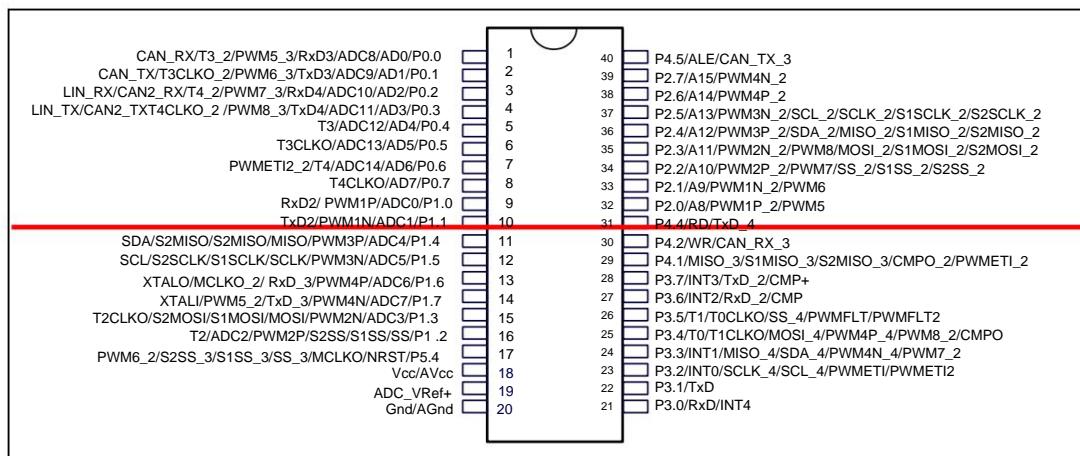


Looking at the chip silk screen The small dot at the bottom left is the

first foot. Looking at the chip silk screen The last letter on the bottom line is the chip version number



Looking at the chip silk screen The small dot at the bottom left is the first foot. Looking at the chip silk screen The last letter on the bottom line is the chip version number



## 2.2.3 Pin description

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
1			P5.3	I/O Standard IO port	
			TxD4_2	O Send pin of serial port 4	
			CAN2_TX_2	O CAN2 bus send pin	
			LIN_TX_2	O LIN bus send pin	
2		6	P0.5	I/O Standard IO port	
			AD5	I address bus	
			ADC13	I ADC analog input channel 13	
			T3CLKO	O Timer 3 clock division output	
3		7	P0.6	I/O Standard IO port	
			AD6	I address bus	
			ADC14	I ADC analog input channel 14	
			T4	I Timer 4 external clock input	
			PWMFLT2_2	I External abnormal detection pin for enhanced PWM	
4		8	P0.7	I/O Standard IO port	
			AD7	I address bus	
			T4CLKO	O Timer 4 clock division output	
5	1	9	P1.0	I/O Standard IO port	
			ADC0	I ADC analog input channel 0	
			PWM1P	I/O PWM1 capture input and pulse output positive	
			RxD2	I Receive pin of serial port 2	
6	2	10	P1.1	I/O Standard IO port	
			ADC1	I ADC analog input channel 1	
			PWM1N	I/O PWM1 capture input and pulse output negative	
			TxD2	I Send pin of serial port 2	
7			P4.7	I/O Standard IO port	
			TxD2_2	I Send pin of serial port 2	
			CAN2_TX_3	O CAN2 bus send pin	
			LIN_TX_3	O LIN bus send pin	
8	3	11	P1.4	I/O Standard IO port	
			ADC4	I ADC analog input channel 4	
			PWM3P	I/O PWM3 capture input and pulse output positive	
			MISO	I/O SPI master input slave output	
			S1MISO	I/O USART1 - SPI master input slave output	
			S2MISO	I/O USART2 - SPI master input slave output	
			SDA	Data lines for I/O I2C interface	

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
9	4	12	P1.5	I/O Standard IO port	
			ADC5	I ADC	analog input channel 5
			PWM3N	I/O PWM	M3 capture input and pulse output negative
			SCLK	I/O	SPI clock pin
			S1SCLK	I/O USART1 - SPI clock pin	
			S2SCLK	I/O USART2 - SPI clock pin	
			SCL	I/O I2C	clock line
10	5	13	P1.6	I/O Standard IO port	
			ADC6	I ADC	analog input channel 6
			RxD_3	I Receive	pin of serial port 1
			PWM4P	I/O PWM	M4 capture input and pulse output positive
			MCLKO_2	O Main	clock divided output
			XTALO	O Output	pin of external crystal oscillator
11	6	14	P1.7	I/O Standard IO port	
			ADC7	I ADC	analog input channel 7
			TxD_3	O Send	pin of serial port 1
			PWM4N	I/O PWM	M4 capture input and pulse output negative
			PWM5_2	I/O Capture	input and pulse output of PWM5
			XTALI	I Input	pin of external crystal oscillator/external clock
12	7	15	P1.3	I/O Standard IO port	
			ADC3	I ADC	analog input channel 3
			MOSI	I/O	SPI master output slave input
			S1MOSI	I/O USART1 - SPI master output slave input	
			S2MOSI	I/O USART2 - SPI master output slave input	
			PWM2N	I/O PWM	M2 capture input and pulse output negative
			T2CLKO	O Timer	2 clock division output
13	8	16	P1.2	I/O Standard IO port	
			ADC2	I ADC	analog input channel 2
			PWM2P	I/O PWM	M2 capture input and pulse output positive
			SS	I	SPI slave selection pin (master is output)
			S1SS	I USART1 - SPI slave selection pin (master is output)	
			S2SS	I USART2	Slave selection pin of SPI (master is output)
			T2	I Timer	2 external clock input

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
14	9	17	P5.4	I/O Standard IO port	
			RST	I reset pin	
			MCLKO	O Main clock divided output	
			SS_3	I SPI slave selection pin (master is output)	
			S1SS_3	I USART1 - SPI slave selection pin (master is output)	
			S2SS_3	I USART2-SPI slave selection pin (master is output)	
			PWM6_2	I/O Capture input and pulse output of PWM6	
15	10	18	Vcc	VCC power pin	
			AVcc	VCC ADC power supply pin	
16	11	19	Vref+	I ADC reference voltage pin	
17	12	20	Gnd	GND ground	
			Agnd	GND ADC ground	
			Vref-	I ADC reference voltage ground	
18			P4.0	I/O Standard IO port	
			MOSI_3	I/O SPI master output slave input	
			S1MOSI_3	I/O USART1 - SPI master output slave input	
			S2MOSI_3	I/O USART2 - SPI master output slave input	
19	13	Twenty one	P3.0	I/O Standard IO port	
			RxD	I Receive pin of serial port 1	
			INT4	I External interrupt 4	
20	14	Twenty two	P3.1	I/O Standard IO port	
			TxD	O Send pin of serial port 1	
Twenty one	15	Twenty three	P3.2	I/O Standard IO port	
			INT0	I External interrupt 0	
			SCLK_4	I/O SPI clock pin	
			SCL_4	I/O I2C clock line	
			PWMET1	I PWM external trigger input pin	
			PWMET2	I PWM external trigger input pin 2	

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
Twenty two	16	Twenty four	P3.3	I/O Standard IO port	
			INT1	I External interrupt 1	
			MISO_4	I/O SPI Master Input Slave Output	
			SDA_4	Data lines for I/O I2C interface	
			PWM4N_4	I/O PWM4 capture input and pulse output negative	
			PWM7_2	I/O Capture input and pulse output of PWM7	
Twenty three	17	25	P3.4	I/O Standard IO port	
			T0	I Timer 0 external clock input	
			T1CLKO	O Timer 1 clock division output	
			MOSI_4	I/O SPI master output slave input	
			PWM4P_4	I/O PWM4 capture input and pulse output positive	
			PWM8_2	I/O Capture input and pulse output of PWM8	
			CMPO	O Comparator output	
Twenty four		Twenty six	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 receiving pin	
25	18	26	P5.1	I/O Standard IO port	
			TxD3_2	O Send pin of serial port 3	
			CMP+_3	I Comparator positive input	
			CAN_TX_2	O CAN bus send pin	
26	18	26	P3.5	I/O Standard IO port	
			T1	I Timer 1 external clock input	
			T0CLKO	O Timer 0 clock divide output	
			SS_4	I SPI slave selection pin (master is output)	
			PWMFLT	I External abnormal detection pin for enhanced PWM	
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	

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
28	20	28	P3.7	I/O Standard IO port	
			INT3	I External interrupt 3	
			TxD_2	O Send pin of serial port 1	
			CMP+	I Comparator positive input	
29		29	P4.1	I/O Standard IO port	
			MISO_3	I/O SPI Master Input Slave Output	
			S1MISO_3	I/O USART1 - SPI master input slave output	
			S2MISO_3	I/O USART2 - SPI master 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 of external bus	
			CAN_RX_3	I CAN bus receiving pin	
31		31	P4.3	I/O Standard IO port	
			RxD_4	I Receive pin of serial port 1	
			SCLK_3	I/O SPI clock pin	
			S1SCLK_3	I/O USART1 - SPI clock pin	
			S2SCLK_3	I/O USART2 - SPI clock pin	
32		31	P4.4	I/O Standard IO port	
			RD	O Read signal line of external bus	
			TxD_4	O Send pin of serial port 1	
33	Twenty two	32	P2.0	I/O Standard IO port	
			A8	I address bus	
			PWM1P_2	I/O PWM1 capture input and pulse output positive	
			PWM5	I/O Capture input and pulse output of PWM5	
34	Twenty two	33	P2.1	I/O Standard IO port	
			A9	I address bus	
			PWM1N_2	I/O PWM1 capture input and pulse output negative	
			PWM6	I/O Capture input and pulse output of PWM6	
35	Twenty three	34	P2.2	I/O Standard IO port	
			A10	I address bus	
			SS_2	I SPI slave selection pin (master is output)	
			S1SS_2	I USART1 - SPI slave selection pin (master is output)	
			S2SS_2	I USART2 - Slave selection pin of SPI (master is output)	
			PWM2P_2	I/O PWM2 capture input and pulse output positive	
			PWM7	I/O Capture input and pulse output of PWM7	

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
36	thirty six	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 master output slave input	
			S2MOSI_2	I/O USART2 - SPI master output slave input	
			PWM2N_2	I/O PWM2 capture input and pulse output negative	
			PWM8	I/O Capture input and pulse output of PWM8	
37	25	36	P2.4	I/O Standard IO port	
			A12	I address bus	
			MISO_2	I/O SPI Master Input Slave Output	
			S1MISO_2	I/O USART1 - SPI master input slave output	
			S2MISO_2	I/O USART2 - SPI master input slave output	
			SDA_2	Data lines for I/O I2C interface	
			PWM3P_2	I/O PWM3 capture input and pulse output positive	
38	26	37	P2.5	I/O Standard IO port	
			A13	I address bus	
			SCLK_2	I/O SPI clock pin	
			S1SCLK_2	I/O USART1 - SPI clock pin	
			S2SCLK_2	I/O USART2 - SPI clock pin	
			SCL_2	I/O I2C clock line	
			PWM3N_2	I/O PWM3 capture input and pulse output negative	
39	27	38	P2.6	I/O Standard IO port	
			A14	I address bus	
			PWM4P_2	I/O PWM4 capture input and pulse output positive	
40	28	39	P2.7	I/O Standard IO port	
			A15	I address bus	
			PWM4N_2	I/O PWM4 capture input and pulse output negative	
41		40	P4.5	I/O Standard IO port	
			ALE	O address latch signal	
			CAN_TX_3	O CAN bus send pin	

Numbering		name	type	illustrate
LQFP48	PDIP40			
42		P4.6	I/O Standard IO port	
		RxD2_2	I Receive pin of serial port 2	
		CAN2_RX_3	I CAN2 bus receiving pin	
		LIN_RX_3	I LIN bus receiving pin	
43	29	1	P0.0	I/O Standard IO port
			AD0	I address bus
			ADC8	I ADC analog input channel 8
			RxD3	I Receive pin of serial port 3
			PWM5_3	I/O Capture input and pulse output of PWM5
			CAN_RX	I CAN bus receiving pin
44	30	2	P0.1	I/O Standard IO port
			AD1	I address bus
			ADC9	I ADC analog input channel 9
			TxD3	O Send pin of serial port 3
			PWM6_3	I/O Capture input and pulse output of PWM6
			CAN_TX	O CAN bus send pin
45	31	3	P0.2	I/O Standard IO port
			AD2	I address bus
			ADC10	I ADC analog 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 receiving pin
			LIN_RX	I LIN bus receiving pin
46	32	4	P0.3	I/O Standard IO port
			AD3	I address bus
			ADC11	I ADC analog input channel 11
			TxD4	O Send pin of serial port 4
			PWM8_3	I/O Capture input and pulse output of PWM8
			CAN2_TX	O CAN2 bus send pin
			LIN_TX	O LIN bus send pin
47		5	P0.4	I/O Standard IO port
			AD4	I address bus
			ADC12	I ADC analog 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 receiving pin
			LIN_RX_2	I LIN bus receiving pin

## 2.3 STC32F12K60-LQFP48/LQFP32/PDIP40

### 2.3.1 Features and Price

ÿ Selection price (no external crystal oscillator, no external reset, 12-bit ADC, 15 channels)

	STC32F12K60-LQFP48	STC32F12K60-LQFP32	STC32F12K60-PDIP40	Price and Package
Availability	Can be ordered now			
Sample delivery in A few days	Can be ordered now			

ÿ Kernel

ÿ Ultra-high-speed 32-bit 8051 core (1T), about 70 times faster than traditional 8051

ÿ 50 interrupt sources, 4 interrupt priority levels

ÿ Support online simulation

ÿ Working voltage

ÿ 1.9V~5.5V (When the working temperature is lower than -40°C, the working voltage shall not be lower than 3.0V)

ÿ Working temperature

ÿ -40°C~85°C (internal high-speed IRC (36MHz or below) and external crystal can be used)

ÿ -40°C~125°C (When the temperature is higher than 85°C, please use an external high temperature resistant crystal oscillator, and the operating frequency should be controlled below 24MHz)

ÿ Flash memory

ÿ Maximum 60K bytes FLASH program memory (ROM), used to store user code

ÿ Support user-configured EEPROM size, 512-byte single page erasing, and erasing times can reach more than 100,000 times

ÿ Support hardware USB direct download and common serial port download

ÿ Support hardware SWD real-time simulation, P3.0/P3.1 (requires STC-USB Link1 tool)

ÿ SRAM, a total of 12K bytes

ÿ 8K bytes internal SRAM (edata)

ÿ 4K bytes internal extended RAM (internal xdata) **ÿ Usage note:**

(It is strongly recommended not to use **idata** and **pdata** to declare variables)

ÿ Clock control

ÿ Internal high precision IRC (up and down adjustment during ISP programming)  
 ÿ Error  $\pm 0.3\%$  (25 $^{\circ}$ C at room temperature)  
 ÿ  $-1.35\% \pm 1.30\%$  temperature drift (full temperature range, -40 $^{\circ}$ C to 85 $^{\circ}$ C)  
 ÿ  $-0.76\% \pm 0.98\%$  temperature drift (temperature range, -20 $^{\circ}$ C to 65 $^{\circ}$ C)

ÿ Internal 32KHz low speed IRC (large error)

ÿ External crystal oscillator (4MHz to 33MHz) and external clock

ÿ Internal PLL output clock (Note: 96MHz/144MHz output by PLL can be independently used as the clock source of high-speed PWM and high-speed SPI)

Users can freely choose the above 4 clock sources

## ÿ Reset

## ÿ Hardware reset

ÿ Power-on reset, the reset voltage is 1.7V~1.9V. (effective when the chip does not enable the low-voltage reset function)  
 ÿ Reset pin reset, P5.4 is the I/O port by default when leaving the factory, P5.4 pin can be set as the reset pin during ISP download (Note: when setting P5.4

When the pin is a reset pin, the reset level is low)

## ÿ Watchdog overflow reset

ÿ Low-voltage detection reset, providing 4-level low-voltage detection voltage: 2.0V, 2.4V, 2.7V, 3.0V.

## ÿ Software reset

ÿ Software write reset trigger register

## ÿ Interrupt

ÿ Provide 50 interrupt sources: INT0, INT1, INT2, INT3, INT4, timer 0, timer 1, timer 2, timer 3, timer 4,  
 USART1, USART2, UART3, UART4, ADC analog-to-digital conversion, LVD low voltage detection, SPI, I2C, comparator, PWMA,  
 PWMB, USB, CAN, CAN2, LIN, LCMIF color screen interface interrupt, RTC real-time clock, all I/O interrupts (6 groups),  
 I2S audio interface, DMA receive and transmit interrupt of I2S audio interface, DMA receive and transmit interrupt of serial port 1, DMA of serial port 2  
 Receive and transmit interrupt, serial port 3 DMA receive and transmit interrupt, serial port 4 DMA receive and transmit interrupt, I2C DMA receive  
 and transmit interrupts, DMA interrupts for SPI, DMA interrupts for ADC, DMA interrupts for LCD drivers, and memory-to-memory DMA  
 interrupt.

ÿ Provides 4 levels of interrupt priority

## ÿ Digital Peripherals

ÿ Five 16-bit timers: timer 0, timer 1, timer 2, timer 3, timer 4, of which mode 3 of timer 0 has NMI  
 (non-maskable interrupt) function, mode 0 of timer 0 and timer 1 is 16-bit auto-reload mode  
 ÿ 2 high-speed synchronous/asynchronous serial ports: serial port 1 (USART1), serial port 2 (USART2), the fastest baud rate clock source can be FOSC/4. support  
 Synchronous serial port mode, asynchronous serial port mode, SPI mode, LIN mode, infrared mode (IrDA), smart card mode (ISO7816)  
 ÿ 2 high-speed asynchronous serial ports: serial port 3, serial port 4, the fastest baud rate clock source can be FOSC/4  
 ÿ 2 sets of advanced PWM, can realize 8-channel (4 sets of complementary symmetry) PWM with dead zone control, and support external abnormal detection function  
 ÿ SPI: Support master mode and slave mode and master/slave automatic switching  
 ÿ I2C: Support master mode and slave mode  
 ÿ ICE: Hardware support emulation  
 ÿ RTC: supports year, month, day, hour, minute, second, sub-second (1/128 second), and supports clock interrupt and a set of alarms  
 ÿ USB: USB2.0/USB1.1 compatible with full-speed USB, 6 bidirectional endpoints, support 4 endpoint transfer modes (control transfer, interrupt transfer, batch transfer  
 transfers and isochronous transfers), each endpoint has a 64-byte buffer  
 ÿ I2S: Audio interface  
 ÿ CAN: Two independent CAN 2.0 control units  
 ÿ LIN: An independent LIN control unit (supports versions 1.3 and 2.1), USART1 and USART2 can support two sets of LIN  
 ÿ MDU32: Hardware 32-bit multiplier and divider (including 32-bit divided by 32-bit, 32-bit multiplied by 32-bit)  
 ÿ I/O port interrupt: All I/O supports interrupt, each group of I/O interrupt has independent interrupt entry address, all I/O interrupt can support 4 kinds of interrupt

Mode: high level interrupt, low level interrupt, rising edge interrupt, falling edge interrupt. The I/O port interrupt can be wake-up from power-down, and there are 4 levels in the interrupt priority.

ÿ LCD driver module: supports 8080 and 6800 interfaces and 8-bit and 16-bit data width

ÿ DMA: Support SPI shift receive data to memory, SPI shift transmit data in memory, I2C transmit data in memory, I2C receive data

Data to memory, serial port 1/2/3/4 to receive data to memory, serial port 1/2/3/4 to send data from memory, ADC to automatically sample data to memory (calculate the average value at the same time), LCD driver to send data from memory , and memory-to-memory data replication

ÿ Hardware digital ID: support 32 bytes

ÿ Analog peripherals

ÿ ADC: Ultra-high-speed ADC, supports 12-bit high-precision 15-channel (channel 0 to channel 14) analog-to-digital conversion, and channel 15 of ADC is used for testing

Internal reference voltage (when the chip is shipped from the factory, the internal reference voltage is adjusted to 1.19V, with an error of ±1%)

ÿ Comparator: A set of comparators

ÿ GPIO

ÿ Up to 44 GPIOs: P0.0~P0.7, P1.0~ P1.7 (without P1.2), P2.0~P2.7, P3.0~P3.7, P4.0~ P4.7, P5.0~P5.4

ÿ All GPIOs support the following 4 modes: quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode, high-impedance input mode All are

high impedance input state, user must set IO port first when using IO port

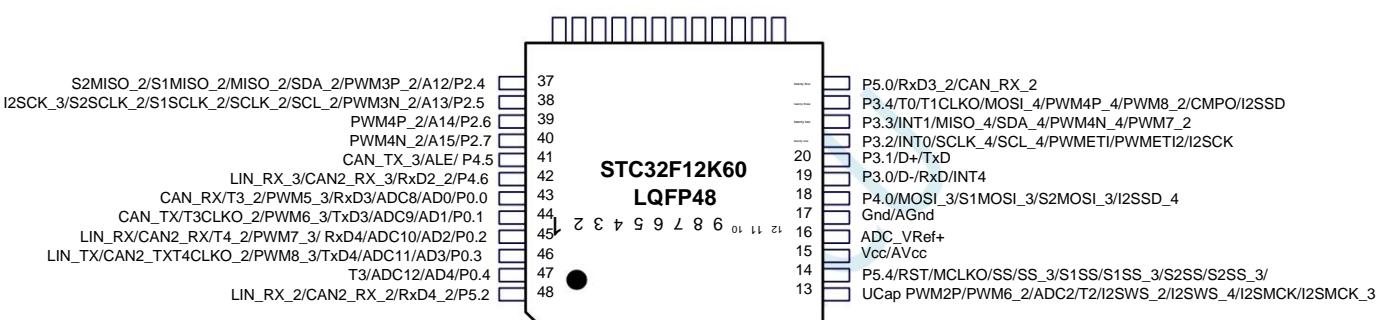
model

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

ÿ Packaging

ÿ LQFP48, LQFP32, PDIP40

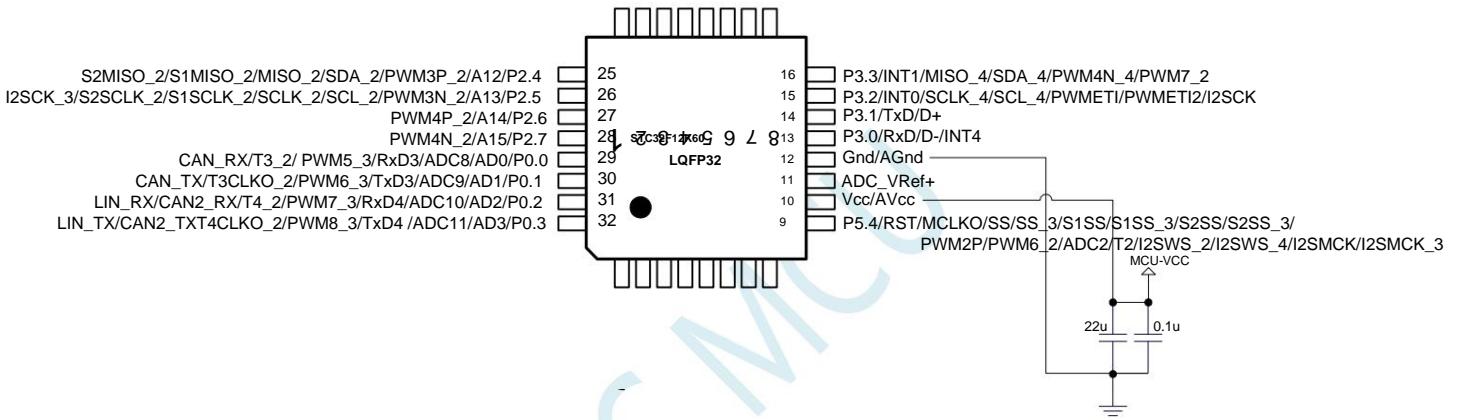
## 2.3.2 Pin Diagram, Minimum System



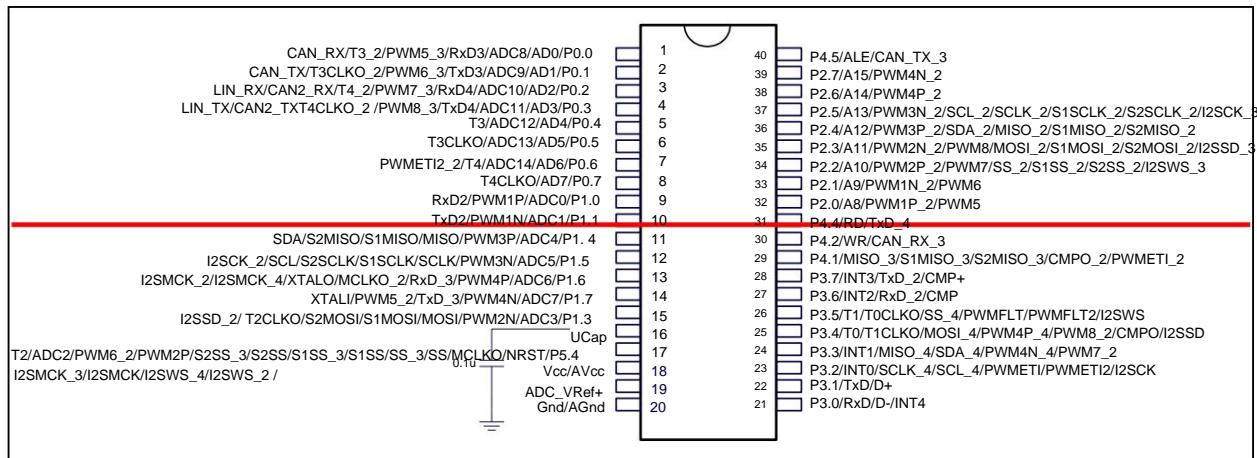
Looking at the chip silk screen The small dot at the bottom left is the first foot. Looking at the chip silk screen The last letter on the bottom line is the chip version number

## Notice:

1. Except for P3.0 and P3.1, all other I/O ports are in the high-impedance input state after power-on.  
When using the I/O port, the user must first set the I/O port mode
2. All I/O ports can be set to quasi-bidirectional port mode, strong push-pull output mode, open-drain output mode  
Output mode or high-impedance input mode, and each I/O can independently enable the internal 4K pull-up resistor
3. When the P5.4 port is enabled as the reset pin, the reset level is low
4. The external reference power pin ADC\_VRef+ of ADC must not be floating and must be connected to an external reference  
Power supply or connect directly 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 silk screen The small dot at the bottom left is the first foot. Looking at the chip silk screen The last letter on the bottom line is the chip version number



### 2.3.3 Pin description

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
1			P5.3	I/O Standard IO port	
			TxD4_2	O Send pin of serial port 4	
			CAN2_TX_2	O CAN2 bus send pin	
			LIN_TX_2	O LIN bus send pin	
2		6	P0.5	I/O Standard IO port	
			AD5	I address bus	
			ADC13	I ADC analog input channel 13	
			T3CLKO	O Timer 3 clock division output	
3		7	P0.6	I/O Standard IO port	
			AD6	I address bus	
			ADC14	I ADC analog input channel 14	
			T4	I Timer 4 external clock input	
			PWMFLT2_2	I External abnormal detection pin for enhanced PWM	
4		8	P0.7	I/O Standard IO port	
			AD7	I address bus	
			T4CLKO	O Timer 4 clock division output	
5	1	9	P1.0	I/O Standard IO port	
			ADC0	I ADC analog input channel 0	
			PWM1P	I/O PWM1 capture input and pulse output positive	
			RxD2	I Receive pin of serial port 2	
6	2	10	P1.1	I/O Standard IO port	
			ADC1	I ADC analog input channel 1	
			PWM1N	I/O PWM1 capture input and pulse output negative	
			TxD2	I Send pin of serial port 2	
7			P4.7	I/O Standard IO port	
			TxD2_2	I Send pin of serial port 2	
			CAN2_TX_3	O CAN2 bus send pin	
			LIN_TX_3	O LIN bus send pin	
8	3	11	P1.4	I/O Standard IO port	
			ADC4	I ADC analog input channel 4	
			PWM3P	I/O PWM3 capture input and pulse output positive	
			MISO	I/O SPI Master Input Slave Output	
			S1MISO	I/O USART1 - SPI master input slave output	
			S2MISO	I/O USART2 - SPI master input slave output	
			SDA	Data lines for I/O I2C interface	

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
9	4	12	P1.5	I/O Standard IO port	
			ADC5	I ADC	analog input channel 5
			PWM3N	I/O PWM	M3 capture input and pulse output negative
			SCLK	I/O	SPI clock pin
			S1SCLK	I/O USART1 - SPI clock pin	
			S2SCLK	I/O USART2 - SPI clock pin	
			SCL	I/O I2C	clock line
			I2SCK_2	I/O I2S	clock line
10	5	13	P1.6	I/O Standard IO port	
			ADC6	I ADC	analog input channel 6
			RxD_3	I	Receive pin of serial port 1
			PWM4P	I/O PWM	M4 capture input and pulse output positive
			MCLKO_2	O Main	clock divided output
			XTALO	O Output	pin of external crystal oscillator
			I2SMCK_2	O I2S	main clock line
			I2SMCK_4	O I2S	main clock line
11	6	14	P1.7	I/O Standard IO port	
			ADC7	I ADC	analog input channel 7
			TxD_3	O Send	pin of serial port 1
			PWM4N	I/O PWM	M4 capture input and pulse output negative
			PWM5_2	I/O Capture	input and pulse output of PWM5
			XTALI	I Input	pin of external crystal oscillator/external clock
12	7	15	P1.3	I/O Standard IO port	
			ADC3	I ADC	analog input channel 3
			MOSI	I/O	SPI master output slave input
			S1MOSI	I/O USART1 - SPI master output slave input	
			S2MOSI	I/O USART2 - SPI master output slave input	
			PWM2N	I/O PWM	M2 capture input and pulse output negative
			T2CLKO	O Timer	2 clock division output
			I2SSD_2	I/O I2S	data line
13	8	16	UCAP	I USB	core power regulator pin

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
14	9	17	P5.4	I/O	Standard IO port
			RST	I	reset pin
			MCLKO	O	Main clock divided output
			SS_3	I	SPI slave selection pin (master is output)
			SS	I	SPI slave selection pin (master is output)
			S1SS_3	I	USART1 - SPI slave selection pin (master is output)
			S1SS	I	USART1 - SPI slave selection pin (master is output)
			S2SS_3	I	USART2-SPI slave selection pin (master is output)
			S2SS	I	USART2-SPI slave selection pin (master is output)
			PWM2P	I/O	PWM2 capture input and pulse output positive
			PWM6_2	I/O	Capture input and pulse output of PWM6
			T2	I	Timer 2 external clock input
			ADC2	I	ADC analog input channel 2
			I2SWS_2	I/O	Channel selection line for I2S
15	10	18	I2SWS_4	I/O	Channel selection line for I2S
			I2SMCK	O	I2S main clock line
			I2SMCK_3	O	I2S main clock line
16	11	19	Vcc	VCC	power pin
			AVcc	VCC	ADC power supply pin
17	12	20	Vref+	I	ADC reference voltage pin
18			Gnd	GND	ground
			Agnd	GND	ADC ground
			Vref-	I	ADC reference voltage ground
			P4.0	I/O	Standard IO port
			MOSI_3	I/O	SPI master output slave input
19	13	twenty one	S1MOSI_3	I/O	USART1 - SPI master output slave input
			S2MOSI_3	I/O	USART2 - SPI master output slave input
			I2SSD_4	I/O	I2S data line
			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

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
20	14	Twenty two	P3.1	I/O Standard IO port	
			D+	I/O USB data port	
			TxD	O Send pin of serial port 1	
Twenty one	15	Twenty three	P3.2	I/O Standard IO port	
			INT0	I External interrupt 0	
			SCLK_4	I/O SPI clock pin	
			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 line	
Twenty two	16	Twenty four	P3.3	I/O Standard IO port	
			INT1	I External interrupt 1	
			MISO_4	I/O SPI master input slave output	
			SDA_4	Data lines for I/O I2C interface	
			PWM4N_4	I/O PWM4 capture input and pulse output negative	
			PWM7_2	I/O Capture input and pulse output of PWM7	
Twenty three	17	25	P3.4	I/O Standard IO port	
			T0	I Timer 0 external clock input	
			T1CLKO	O Timer 1 clock division output	
			MOSI_4	I/O SPI master output slave input	
			PWM4P_4	I/O PWM4 capture input and pulse output positive	
			PWM8_2	I/O Capture input and pulse output of PWM8	
			CMPO	O Comparator output	
			I2SSD	I/O I2S data line	
Twenty four			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 receiving pin	
25			P5.1	I/O Standard IO port	
			TxD3_2	O Send pin of serial port 3	
			CMP+_3	I Comparator positive input	
			CAN_TX_2	O CAN bus send pin	

Numbering			name	type	illustrate
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 divide output	
			SS_4	I SPI slave selection pin (master is output)	
			PWMFLT	I External abnormal detection pin for enhanced PWM	
			I2SWS	Channel select line for I/O 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 Send pin of serial port 1	
			CMP+	I Comparator positive input	
29		29	P4.1	I/O Standard IO port	
			MISO_3	I/O SPI master input slave output	
			S1MISO_3	I/O USART1 - SPI master input slave output	
			S2MISO_3	I/O USART2 - SPI master 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 of external bus	
			CAN_RX_3	I CAN bus receiving pin	
31			P4.3	I/O Standard IO port	
			RxD_4	I Receive pin of serial port 1	
			SCLK_3	I/O SPI clock pin	
			S1SCLK_3	I/O USART1 - SPI clock pin	
			S2SCLK_3	I/O USART2 - SPI clock pin	
			I2SCK_4	I/O I2S clock line	
32		31	P4.4	I/O Standard IO port	
			RD	O read signal line of external bus	
			TxD_4	O Send pin of serial port 1	
33	Twenty one	32	P2.0	I/O Standard IO port	
			A8	I address bus	
			PWM1P_2	I/O PWM1 capture input and pulse output positive	
			PWM5	I/O Capture input and pulse output of PWM5	

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
34	Twenty two	33	P2.1	I/O Standard IO port	
			A9	I address bus	
			PWM1N_2	I/O PWM1 capture input and pulse output negative	
			PWM6	I/O Capture input and pulse output of PWM6	
35	Thirty three	34	P2.2	I/O Standard IO port	
			A10	I address bus	
			SS_2	I SPI slave selection pin (master is output)	
			S1SS_2	I USART1 - SPI slave selection pin (master is output)	
			S2SS_2	I USART2 Slave selection pin of SPI (master is output)	
			PWM2P_2	I/O PWM2 capture input and pulse output positive	
			PWM7	I/O Capture input and pulse output of PWM7	
			I2SWS_3	I/O Channel selection line for I2S	
36	Twenty four	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 master output slave input	
			S2MOSI_2	I/O USART2 - SPI master output slave input	
			PWM2N_2	I/O PWM2 capture input and pulse output negative	
			PWM8	I/O Capture input and pulse output of PWM8	
			I2SSD_3	I/O I2S data line	
37	25	36	P2.4	I/O Standard IO port	
			A12	I address bus	
			MISO_2	I/O SPI master input slave output	
			S1MISO_2	I/O USART1 - SPI master input slave output	
			S2MISO_2	I/O USART2 - SPI master input slave output	
			SDA_2	Data lines for I/O I2C interface	
			PWM3P_2	I/O PWM3 capture input and pulse output positive	
38	26	37	P2.5	I/O Standard IO port	
			A13	I address bus	
			SCLK_2	I/O SPI clock pin	
			S1SCLK_2	I/O USART1 - SPI clock pin	
			S2SCLK_2	I/O USART2 - SPI clock pin	
			SCL_2	I/O I2C clock line	
			PWM3N_2	I/O PWM3 capture input and pulse output negative	
39	27	38	I2SCK_3	I/O I2S clock line	
			P2.6	I/O Standard IO port	
			A14	I address bus	
			PWM4P_2	I/O PWM4 capture input and pulse output positive	

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
40	28	39	P2.7	I/O Standard IO port	
			A15	I address bus	
			PWM4N_2	I/O PWM4 capture input and pulse output negative	
41		40	P4.5	I/O Standard IO port	
			ALE	O address latch signal	
			CAN_TX_3	O CAN bus send pin	
42			P4.6	I/O Standard IO port	
			RxD2_2	I Receive pin of serial port 2	
			CAN2_RX_3	I CAN2 bus receiving pin	
			LIN_RX_3	I LIN bus receiving pin	
43	29	1	P0.0	I/O Standard IO port	
			AD0	I address bus	
			ADC8	I ADC analog input channel 8	
			RxD3	I Receive pin of serial port 3	
			PWM5_3	I/O Capture input and pulse output of PWM5	
			CAN_RX	I CAN bus receiving pin	
44	30	2	P0.1	I/O Standard IO port	
			AD1	I address bus	
			ADC9	I ADC analog input channel 9	
			TxD3	O Send pin of serial port 3	
			PWM6_3	I/O Capture input and pulse output of PWM6	
			CAN_TX	O CAN bus send pin	
45	31	3	P0.2	I/O Standard IO port	
			AD2	I address bus	
			ADC10	I ADC analog 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 receiving pin	
			LIN_RX	I LIN bus receiving pin	

Numbering			name	type	illustrate
LQFP48	LQFP32	PDIP40			
46	32	4	P0.3	I/O Standard IO port	
			AD3	I address bus	
			ADC11	I ADC analog input channel 11	
			TxD4	O Send pin of serial port 4	
			PWM8_3	I/O Capture input and pulse output of PWM8	
			CAN2_TX	O CAN2 bus send pin	
			LIN_TX	O LIN bus send pin	
47		5	P0.4	I/O Standard IO port	
			AD4	I address bus	
			ADC12	I ADC analog 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 receiving pin	
			LIN_RX_2	I LIN bus receiving pin	

## 3 function pins switch

The special peripheral serial port, SPI, PWM, I2C, CAN, LIN and bus control pins of STC32G series microcontrollers can be used in multiple I/O Switch directly to realize time-division multiplexing of one peripheral as multiple devices.

### 3.1 Function pins switch related registers

symbol	describe	address	Bit addresses and symbols								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		

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
MCLKOCR	Master Clock Output Control Register	7EFFE05H MCLKO_S									0000,0000
PWMA_PS	PWMA toggle register	7EFEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWMB_PS	PWMB toggle register	7EFEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWMA_ETRPS	ETRPS ERT toggle register for PWMA 7EFEB0H							BRKAPS ETRAPS[1:0]			xxx,x000
PWMB_ETRPS	ETRPS ERT toggle register for PWMB 7EFEB4H							BRKBPS ETRBPS[1:0]			xxx,x000
T3T4PIN	T3/T4 selection register	7EFEEACH	-	-	-	-	-	-	-	-	T3T4SEL xxxx,xxx0

#### 3.1.1 Peripheral Port Switch Control Register 1 (P\_SW1)

Symbol	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 selection bit

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

CAN\_S[1:0]: CAN function pin selection bit

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 selection bit

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 selection bit

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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

### 3.1.2 Peripheral Port Switch Control Register 2 (P\_SW2)

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

EAXFR: Extended RAM Area Special Function Register (XFR) Access Control Register

0: Disable access to XFR

1: Enable access to XFR.

When you need to access XFR , you must first set EAXFR to 1 to perform normal reading and writing on XFR . It is recommended to initialize at power-on

Set it directly to 1, do not modify it later

I2C\_S[1:0]: I2C function pin selection bit

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

CMPO\_S: Comparator output pin select bit

CMPO_S 0	CMPO
	P3.4
1	P4.1

S4\_S: Serial port 4 function pin selection bit

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

S3\_S: Serial port 3 function pin selection bit

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

S2\_S: Serial port 2 function pin selection bit

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

### 3.1.3 Peripheral port switch control register 3 (P\_SW3)

Symbol 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 selection bit of 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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

S1SPI\_S[1:0]: SPI function pin selection bit of 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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

CAN2\_S[1:0]: CAN2 function pin selection bit

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 selection bit

I2S_S[1:0]	I2SCK	I2SSD I2SM	I2SCK 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)

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

MCLKO\_S: Master clock output pin selection bit

MCLKO_S MCLKO
0
1

### 3.1.5 T3/T4 selection register (T3T4PIN)

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

T3T4SEL: T3/T3CLKO/T4/T4CLKO pin selection 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 (PWMMn\_PS)

Symbol 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 selection 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 selection 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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

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

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

### 3.1.7 Advanced PWM function pin selection register (PWMy\_ETRPS)

symbol	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 select bits for advanced PWMA

ETRAPS[1:0] PWMETI1	
00	P3.2
01	P4.1
10	P7.3
11	-

ETRBPS[1:0]: External trigger pin ERIB selection bits of advanced PWMB

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

BRKAPS: Brake foot PWMFLT selection bit for advanced PWMA

BRKAPS	PWMFLT
0	P3.5
1	Comparator output

BRKBPS: Brake foot PWMFLT2 selection bit of advanced PWMB

BRKBPS	PWMFLT2
--------	---------

0	P3.5
1	Comparator output

STCMCU

## 3.2 Example program

### 3.2.1 Serial port 1 switch

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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 switch

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    P0M0 = 0x00;
```

---

```

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

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                         // For header files, see Download Software

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    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);
}

```


Enable **XFR**  
access to set external data bus speed to fastest  
Set the program code to wait for parameters,  
assign to **CPU** The speed of executing the program is set to the fastest

### 3.2.4 Serial port 4 switching

---

```
//The test frequency is 11.0592MHz

#ifndefinclude "stc8h.h"
#include "stc32g.h" //For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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

---

```
//The test frequency is 11.0592MHz

#ifndefinclude "stc8h.h"
#include "stc32g.h" //For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

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

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    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; //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

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software

```

```

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                            //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                                //assign to      CPU The speed of executing the program is set to the fastest

    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

    while (1);
}

```

### 3.2.8 Main clock output switching

---

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software

void main()
{
    EAXFR = 1;                                Enable      XFR
    CKCON = 0x00;                            access to set external data bus speed to fastest
    WTST = 0x00;                            Set the program code to wait for parameters,
                                                //assign to      CPU The speed of executing the program is set to the fastest

    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);  
}
```

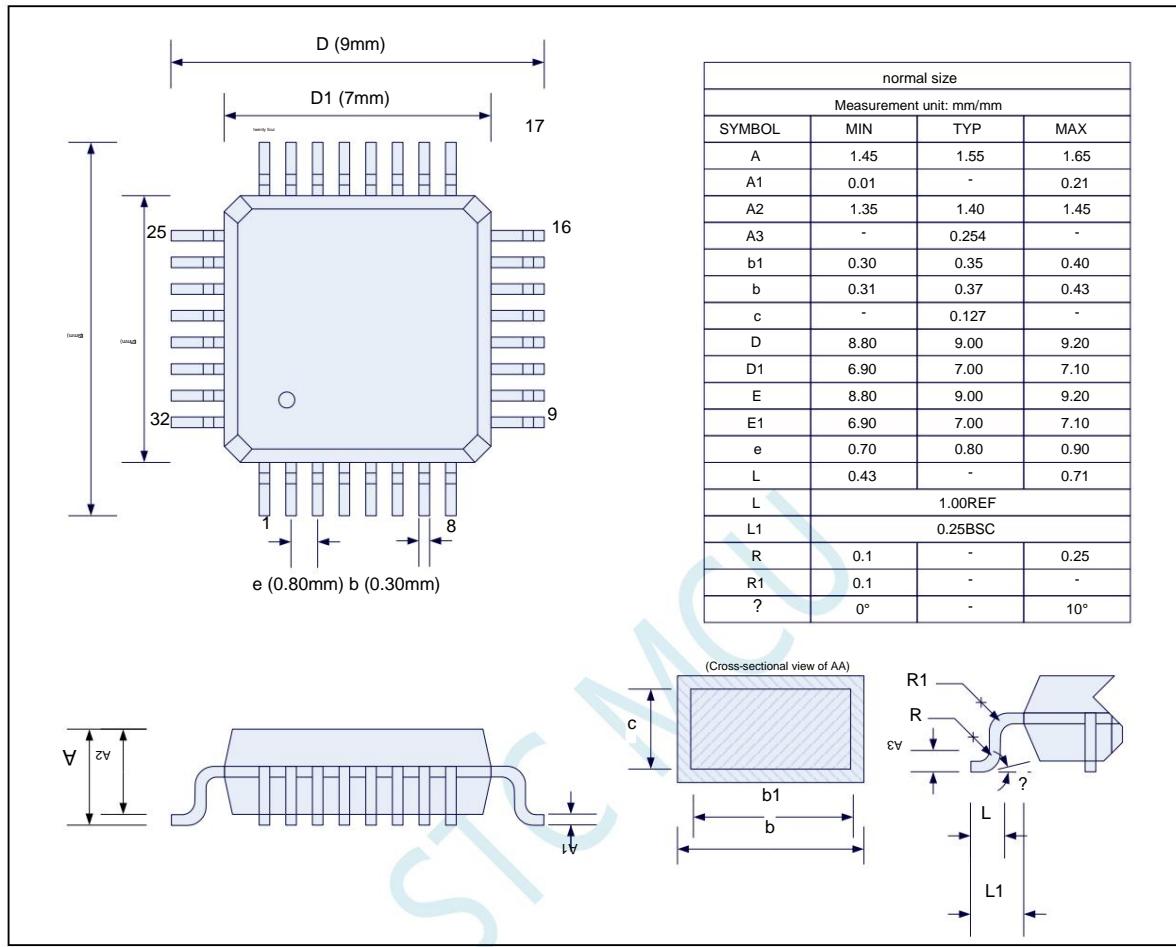
---

---

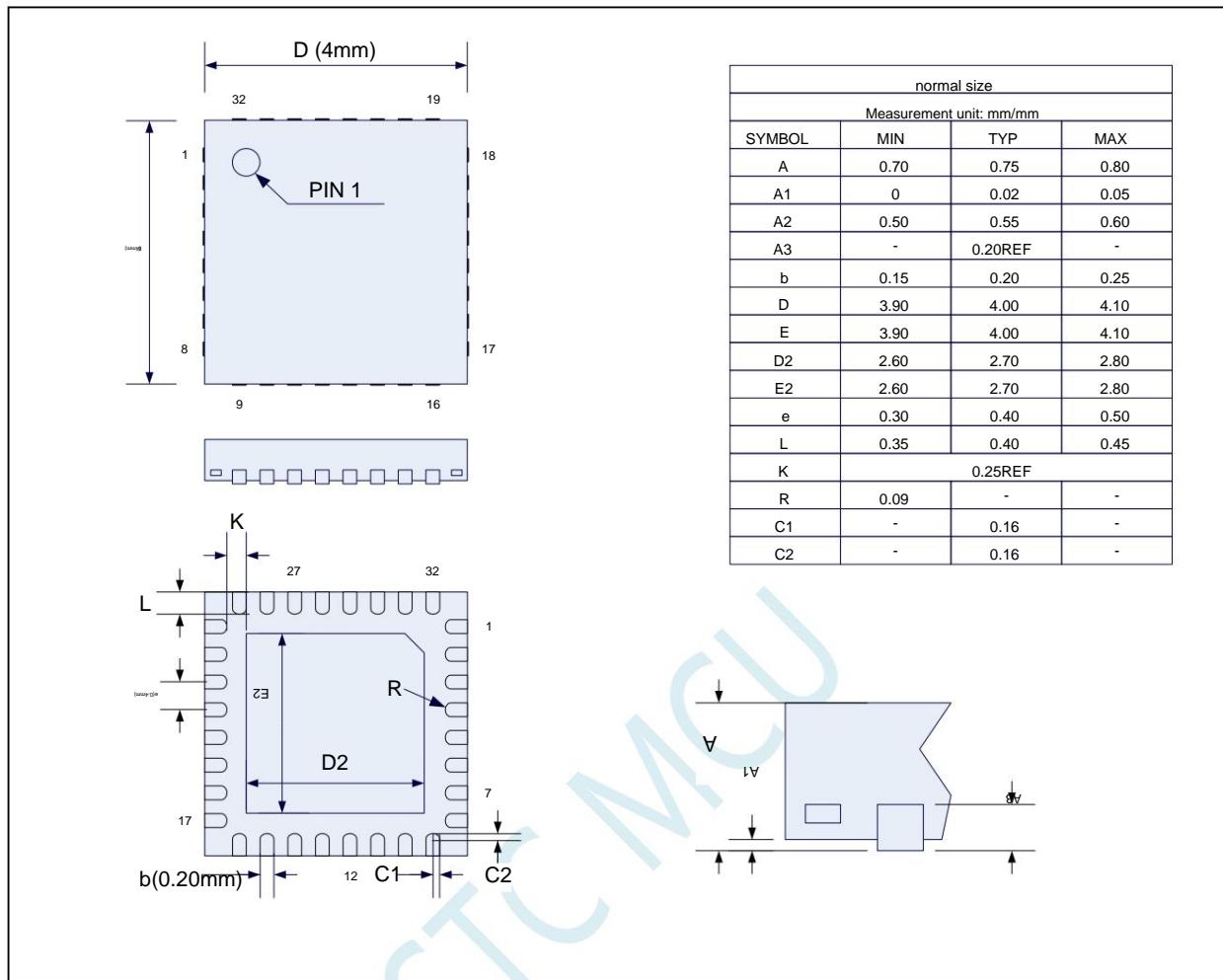
STCMCU

## 4 Package Dimensions

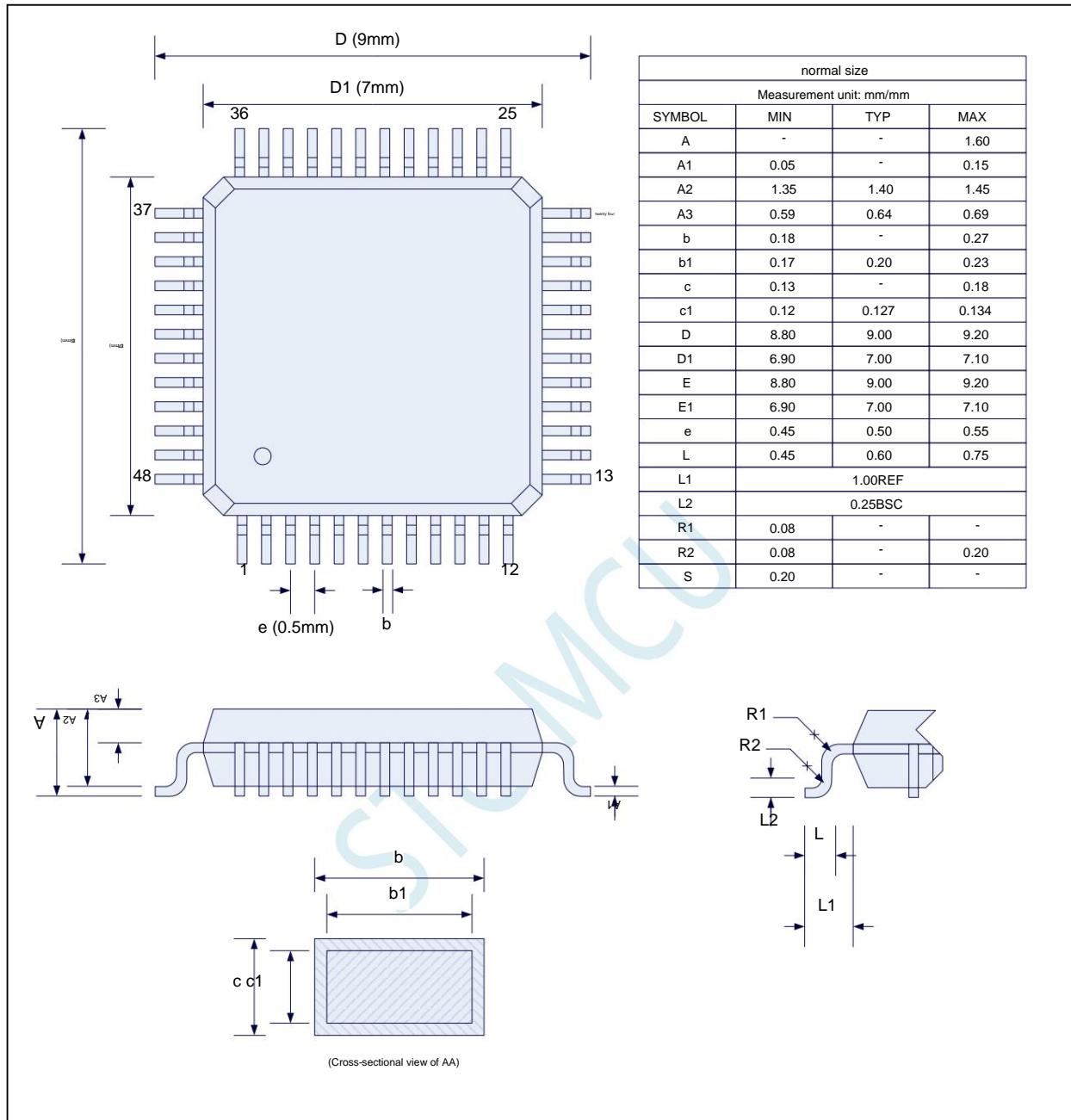
### 4.1 LQFP32 Package Dimensions (9mm\*9mm)



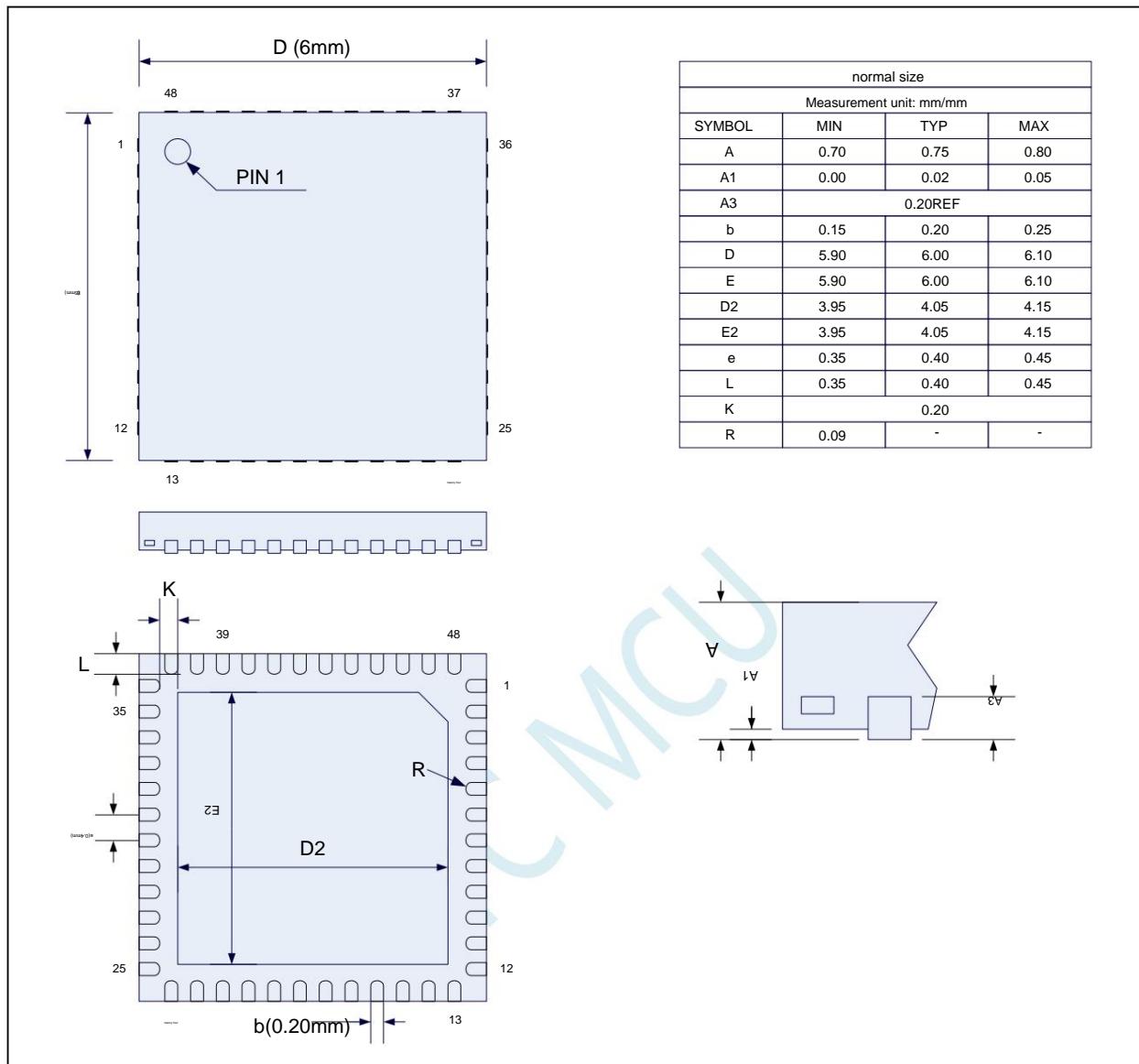
## 4.2 QFN32 Package Dimensions (4mm\*4mm)



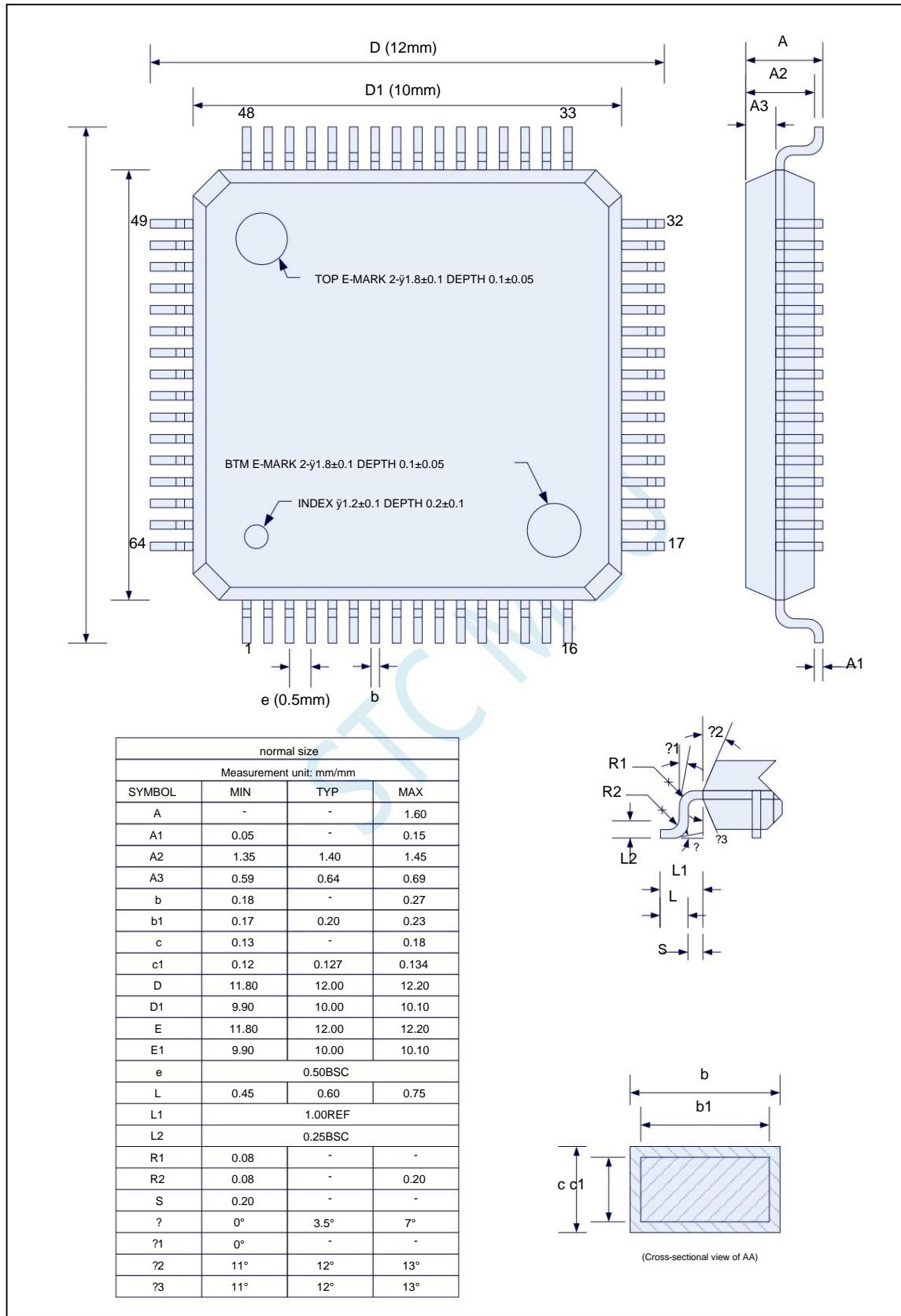
## 4.3 LQFP48 Package Dimensions (9mm\*9mm)



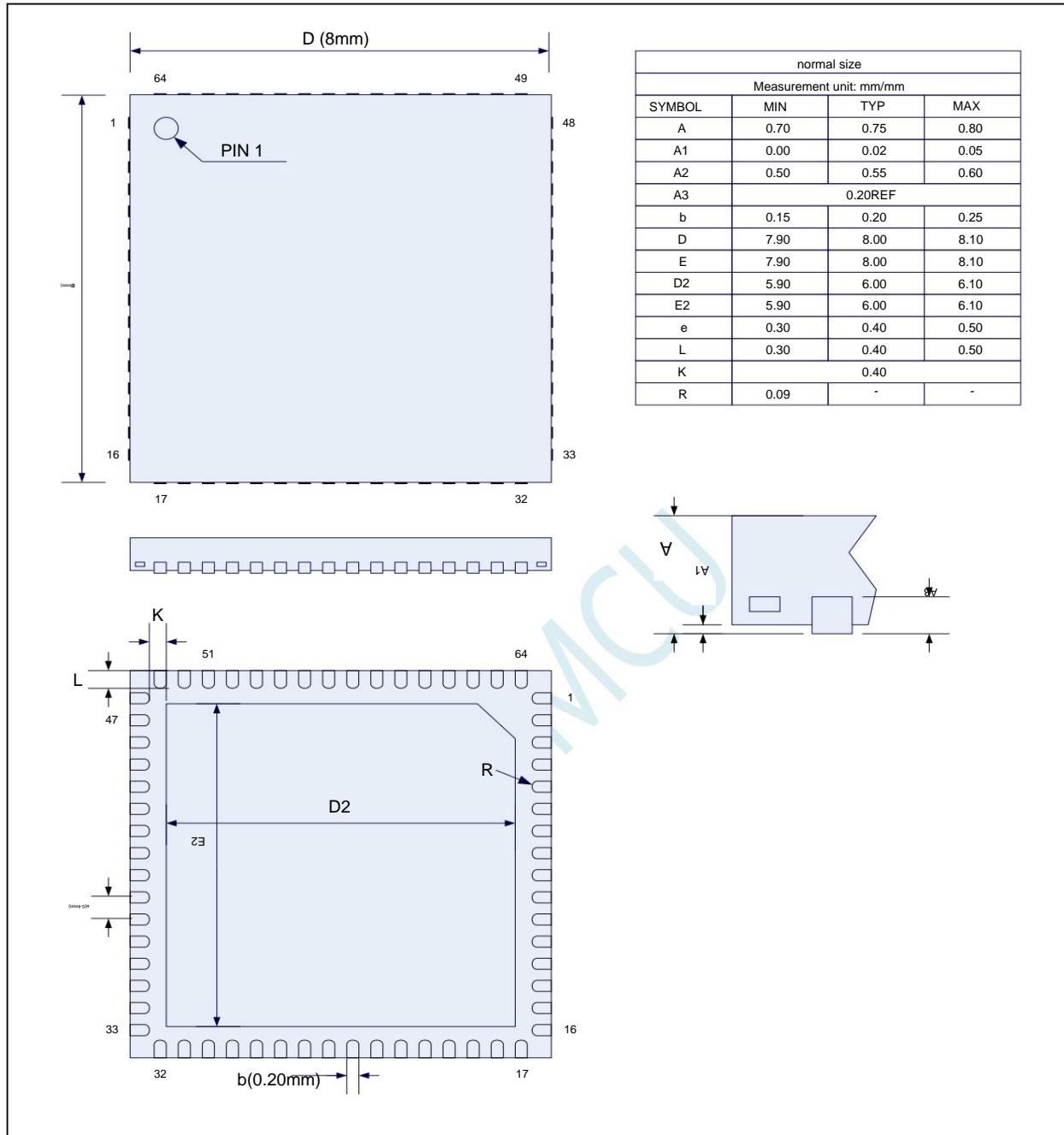
## 4.4 QFN48 Package Dimensions (6mm\*6mm)



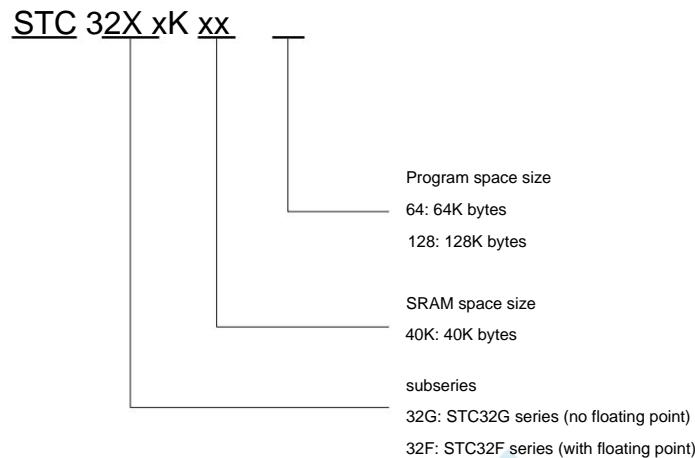
## 4.5 LQFP64 Package Dimensions (12mm\*12mm)



## 4.6 QFN64 Package Dimensions (8mm\*8mm)



## 4.7 STC32G series microcontroller naming rules



## 5. Establishment of compilation and simulation development environment and **ISP** download

### 5.1 Install Keil

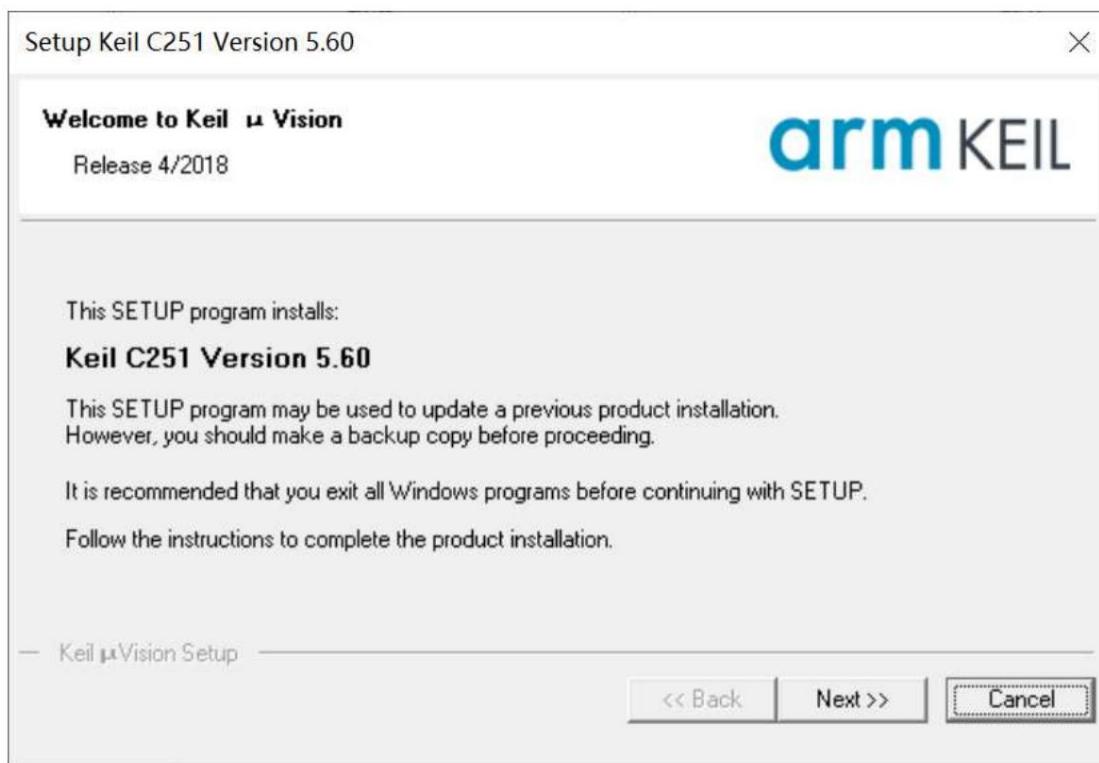
#### 5.1.1 Install C251 Compilation Environment

First log in to Keil's official website and download the latest version of the C251 installation package. The download link is as follows:

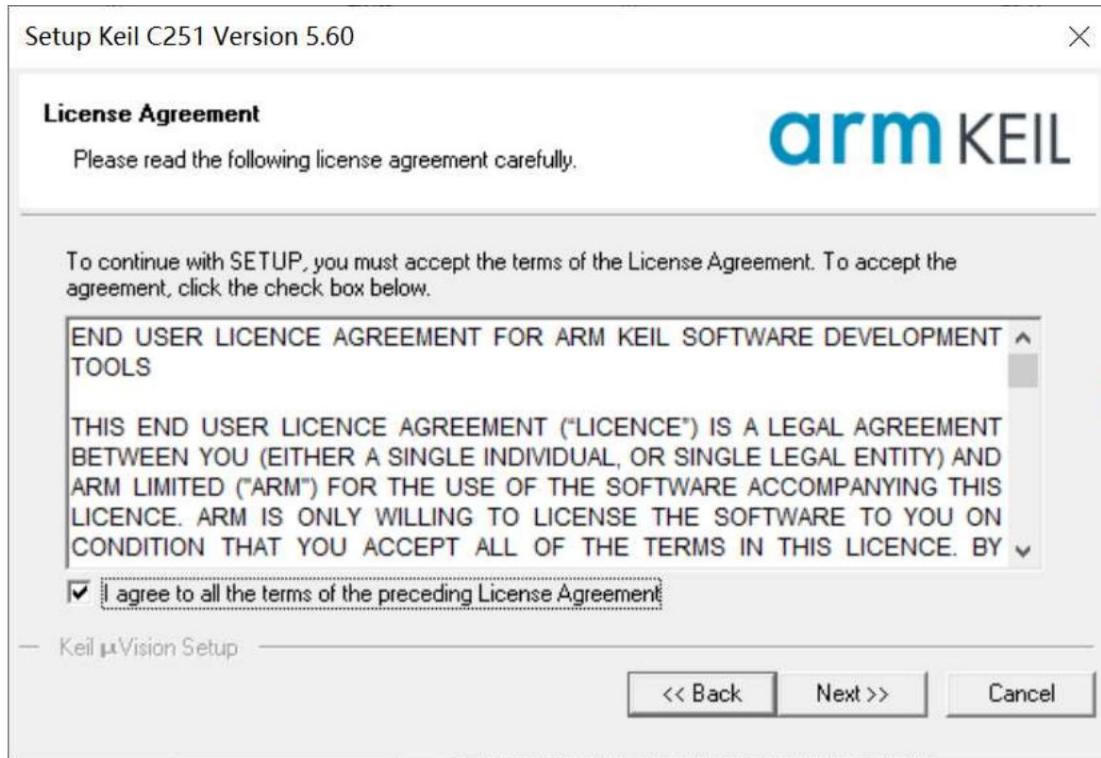
[Keil Product Downloads](#)

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

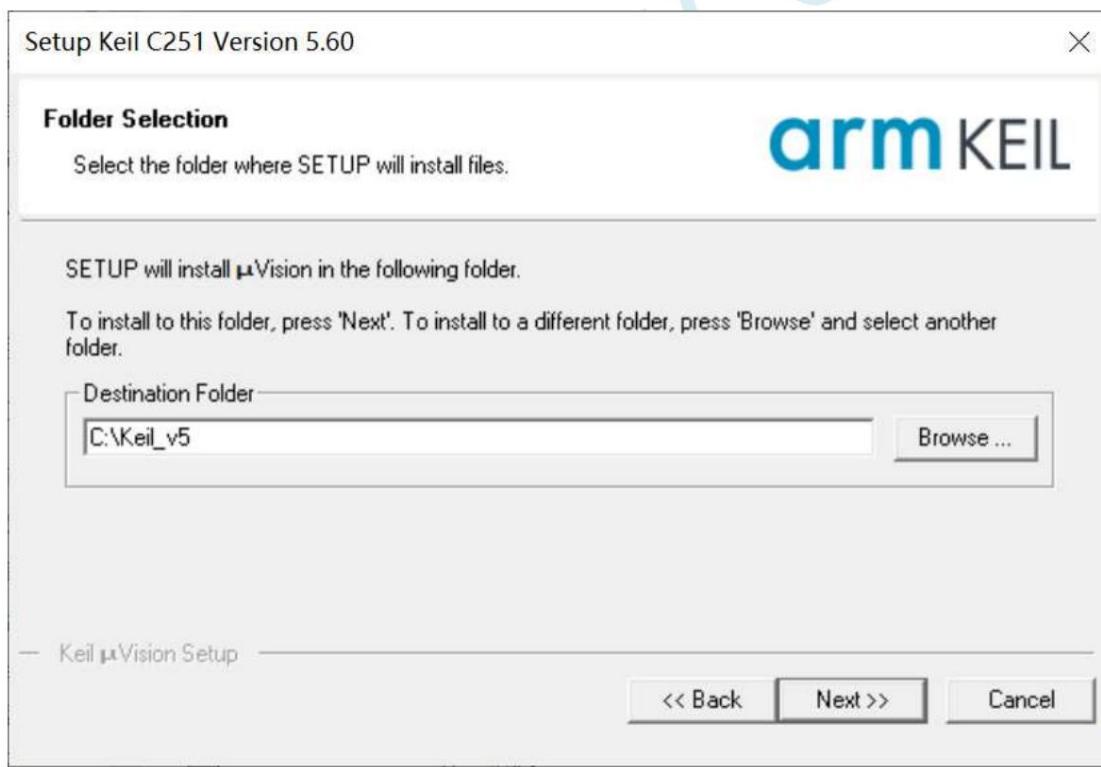
Double-click the downloaded installation package to start the installation, click "Next":



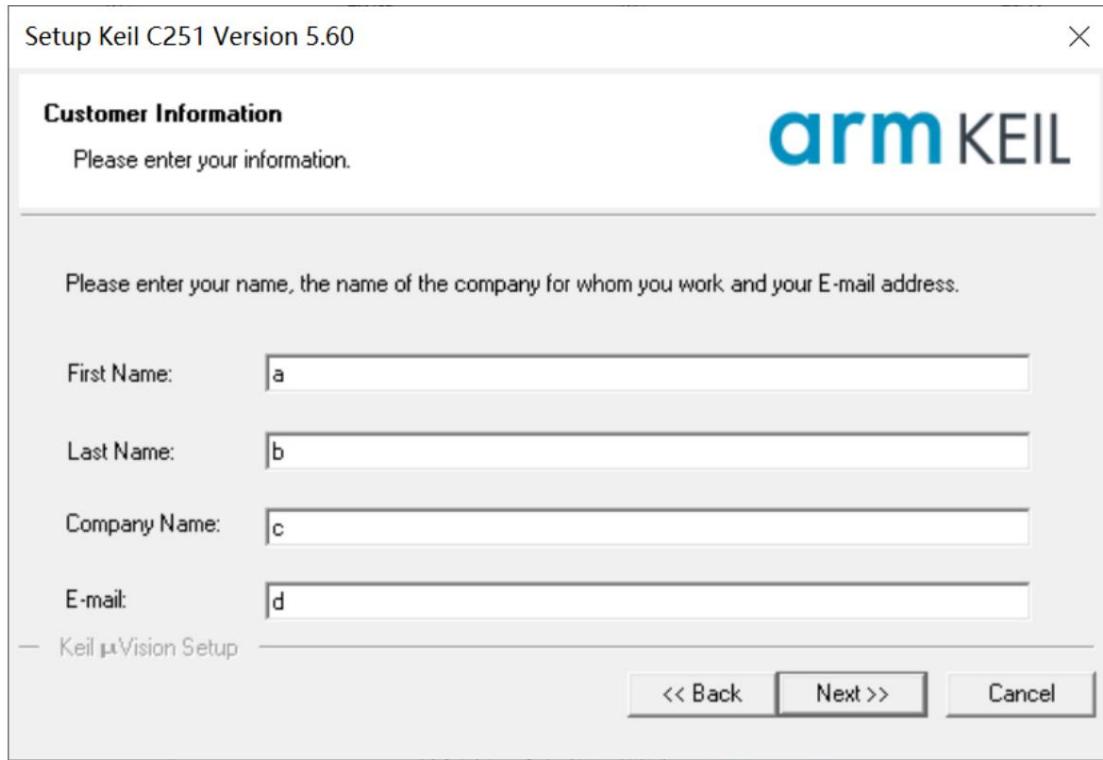
Check "I agree to all the terms of the preceding License Agreement" and click "Next":



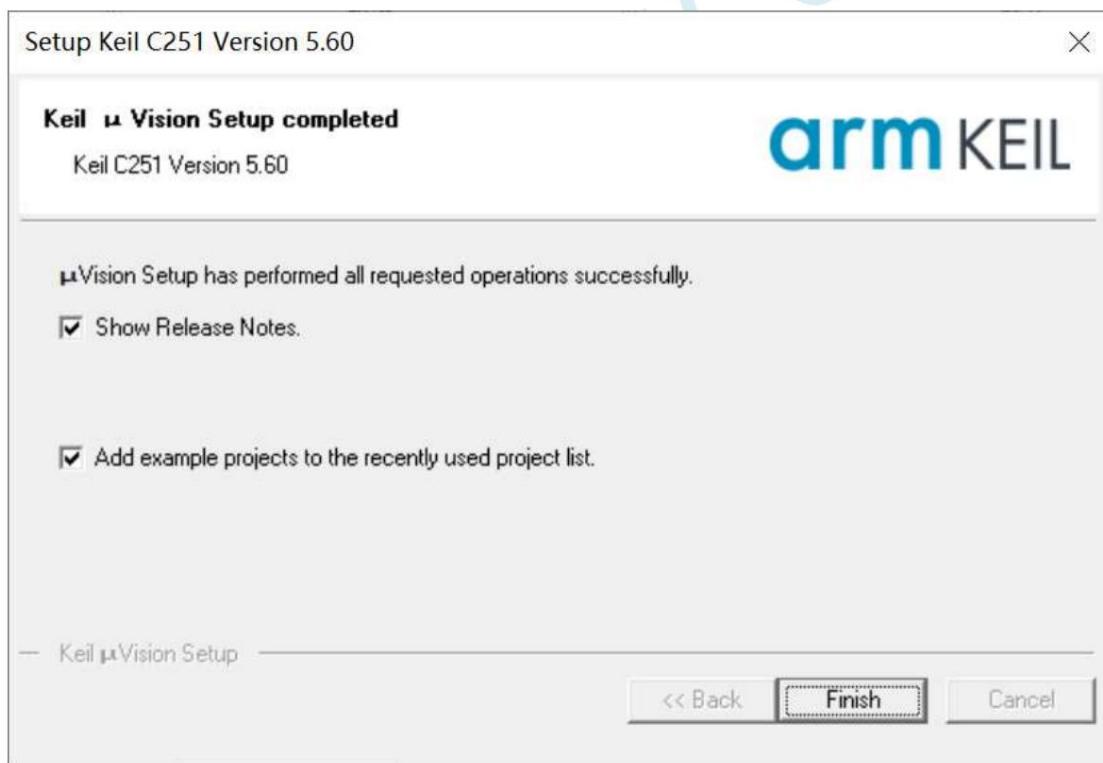
Select the installation directory and click "Next":



Fill in the personal information and click "Next":

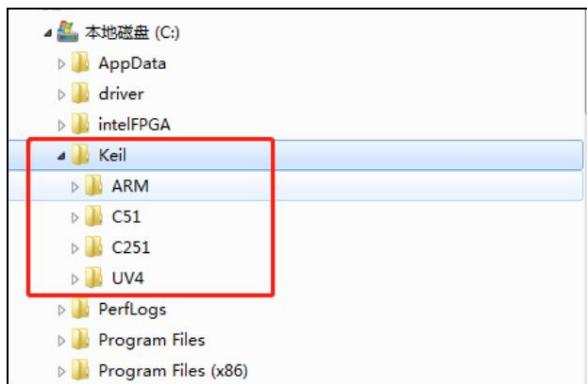


After the installation is complete, click "Finish" to end.

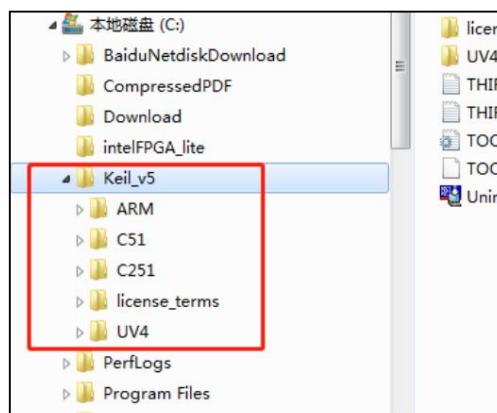


## 5.1.2 How to install Keil's C51, C251 and MDK at the same time

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



The default installation directory of the new version of Keil software is C:\Keil\_v5, C51, C251 and MDK will be installed in C:\Keil\_v5 respectively in the C51, C251 and ARM directories under the directory, as shown in the following figure.



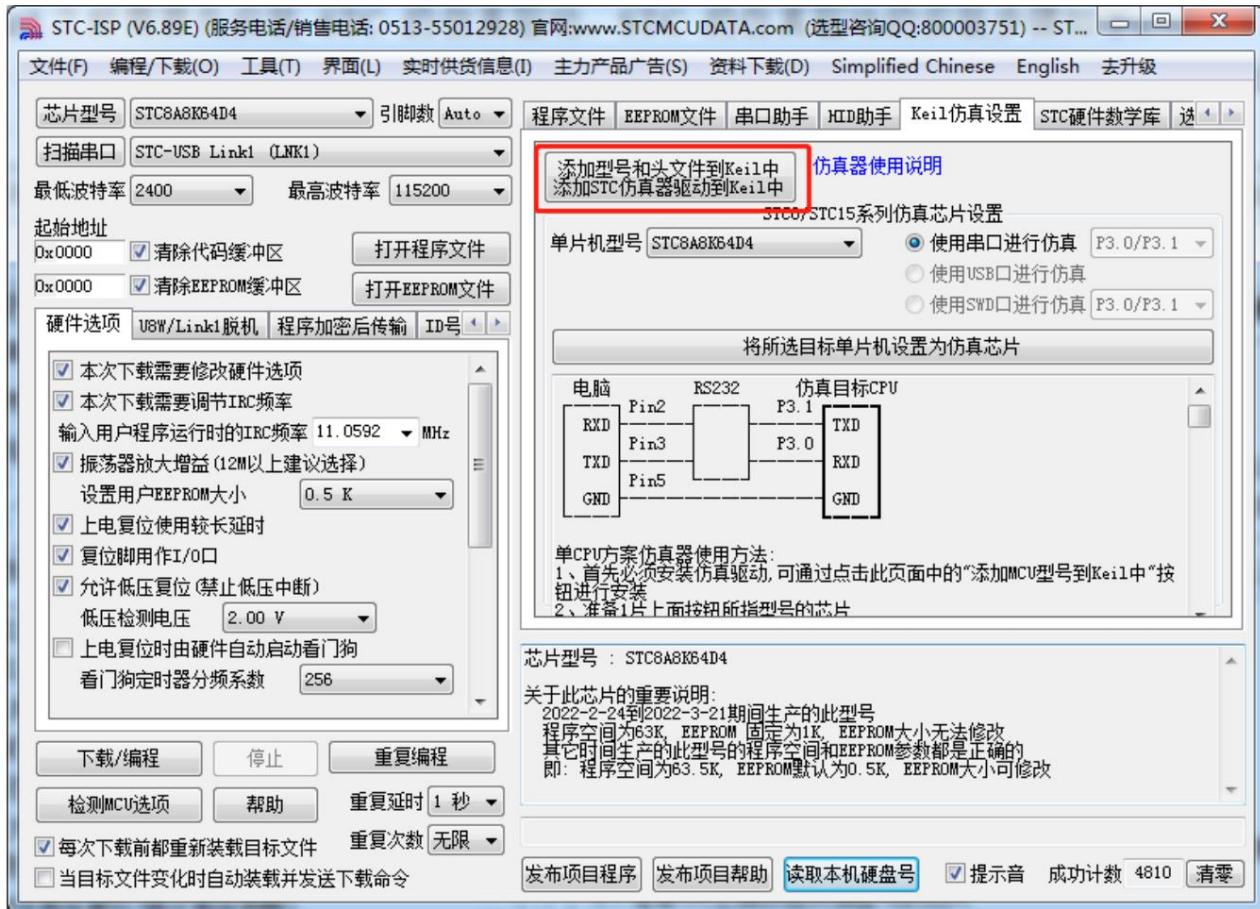
Whether it is a new version or an old version, C51, C251 and MDK are installed in different directories and there is no conflict. software harmony  
It is also carried out by the three softwares separately. The software that has been installed and set up before will not be changed due to the subsequent installation of new software.  
Therefore, you only need to install it in the default way, and the Keil software will handle it automatically.

## 5.2 Add model and header files to Keil

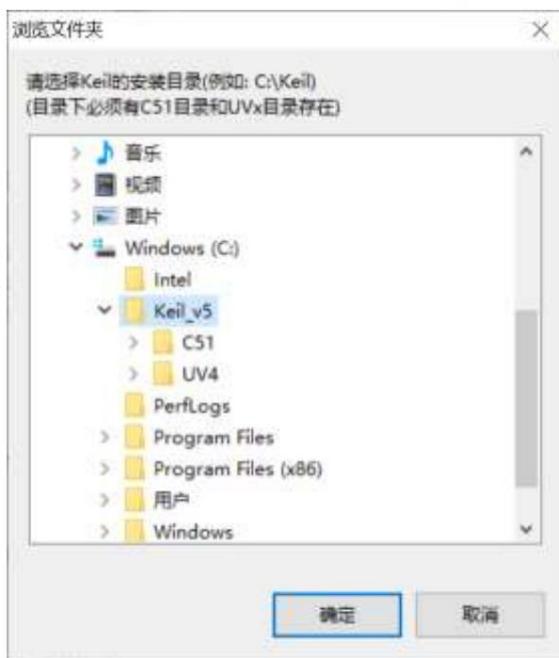
Before using Keil, you need to install the STC emulation driver. The installation steps of the STC emulation driver are as follows:

First open STC's ISP to download the software, then click "Add Model and Header" in the "Keil Emulation Settings" page in the functional area on the right side of the software.

File to Keil Add STC emulator driver to Keil" button:



After pressing, the following screen will appear:



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



That means the driver is installed correctly

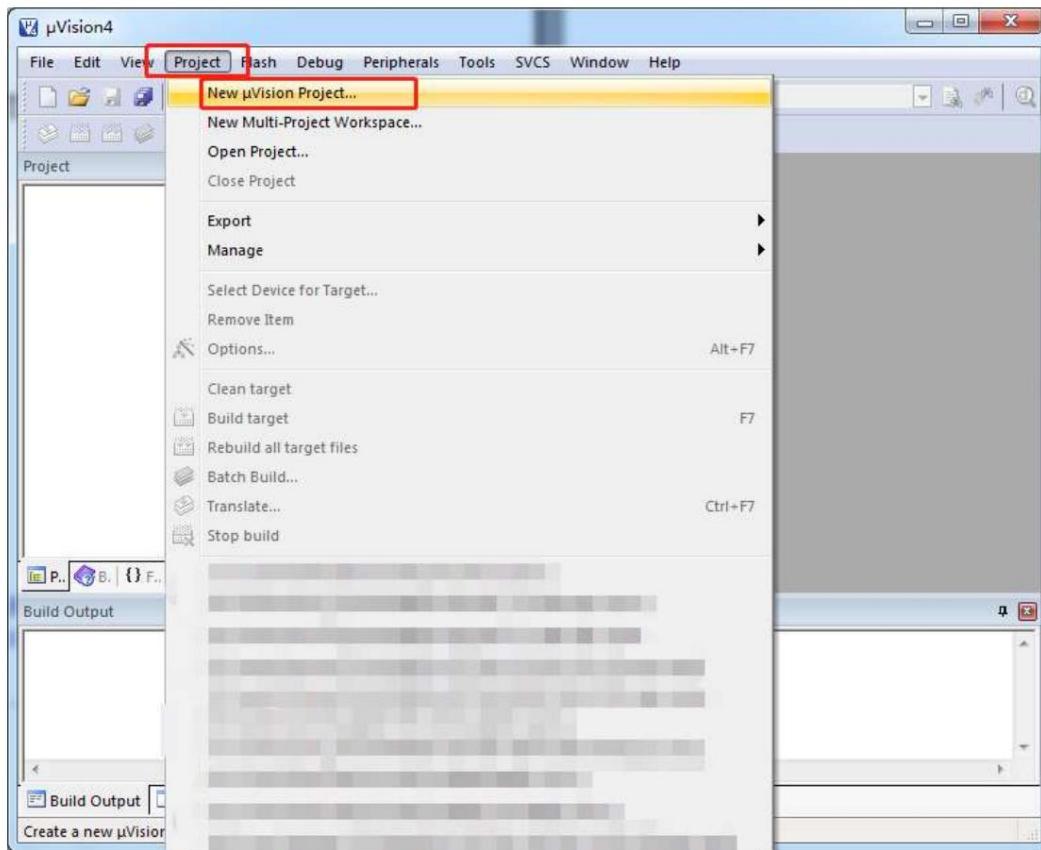
The header files are copied to the "C251\INC\STC" directory under the Keil installation directory by default.

Use "#include <STC32G.H>" or "#include "STC32G.H"" to include in C code can be used correctly

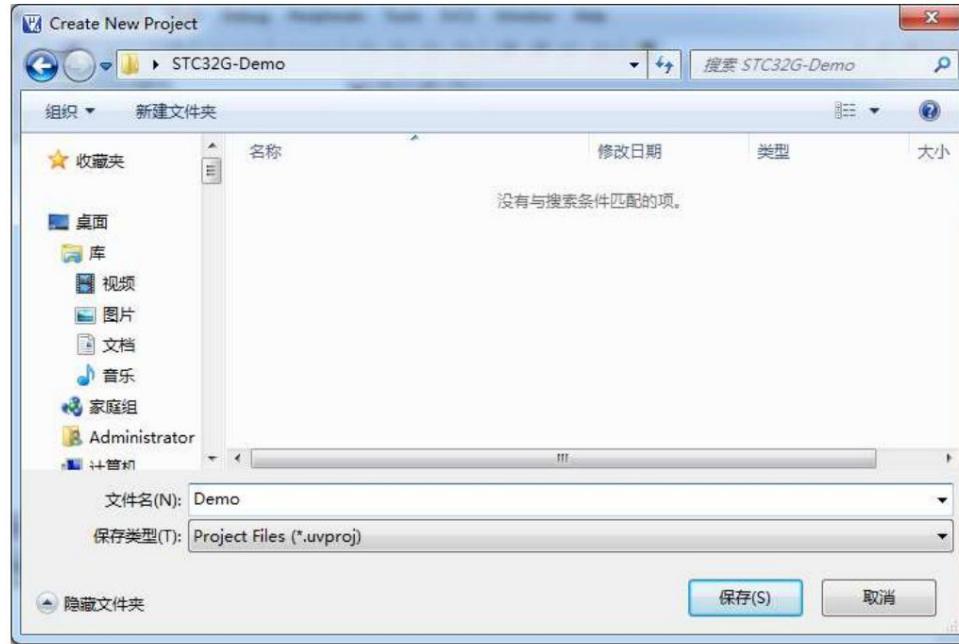
## 5.3 Create and set a project with super 64K program code (Source mode)

### 5.3.1 Set the project path and project name

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

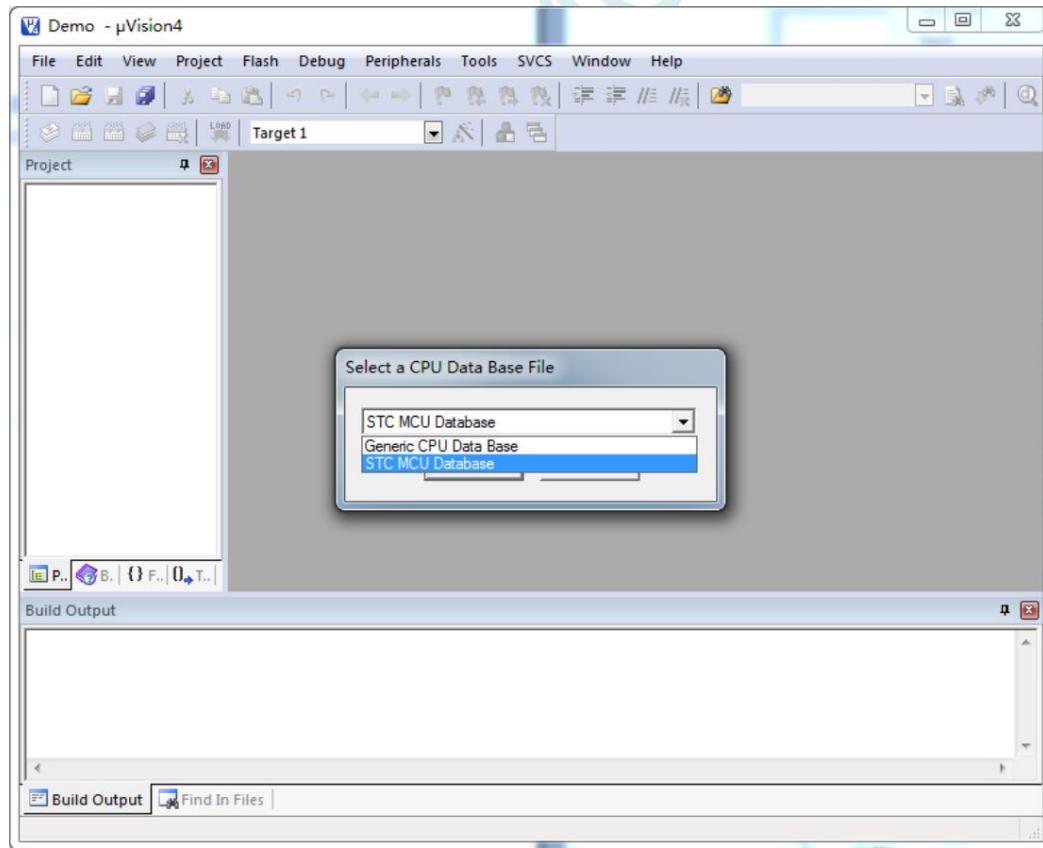


Locate the directory in the prepared project folder and enter a project name (eg: Demo)

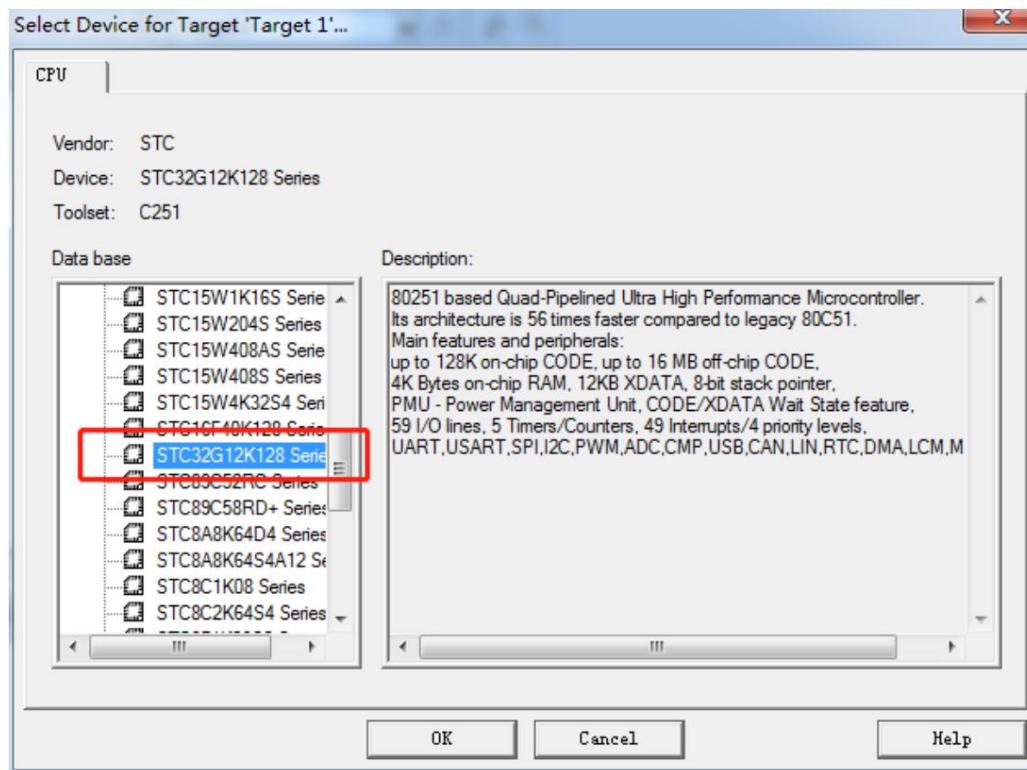


### 5.3.2 Select the target microcontroller model (STC32G12K128)

In the pop-up "Select a CPU Data Base File" window, select "STC MCU Database"

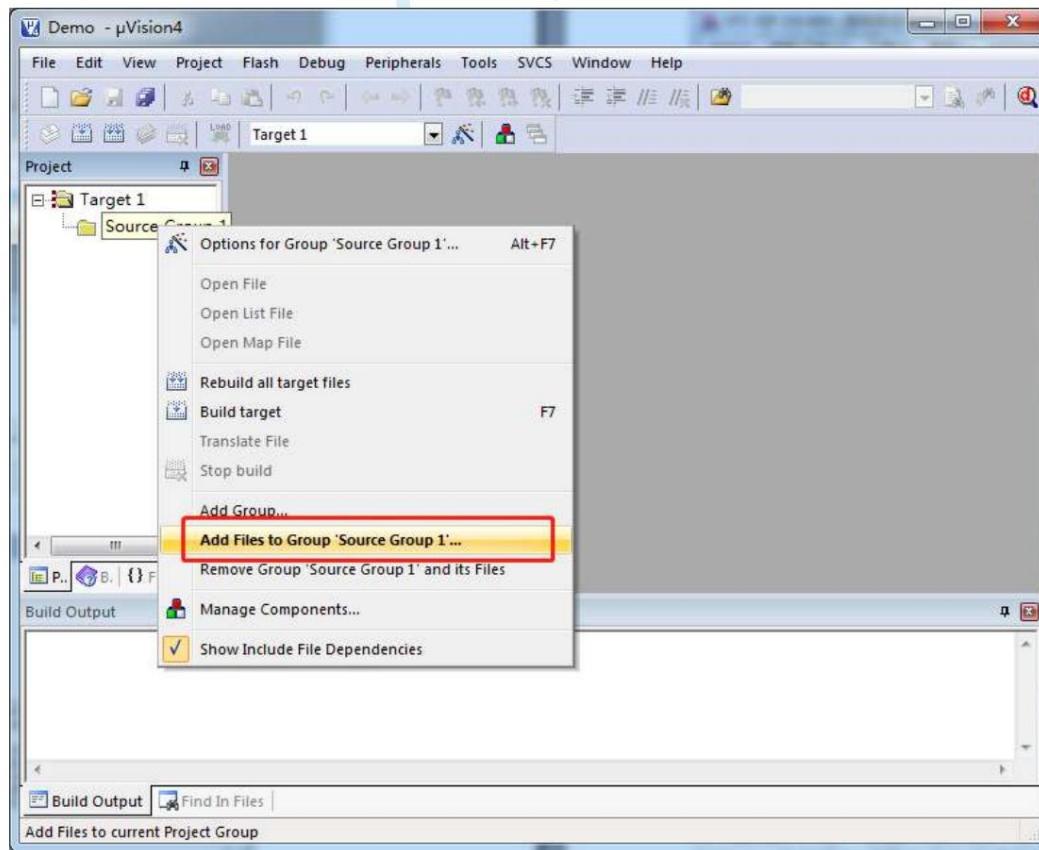


Select the correct target microcontroller model in the "Select Device for Target..." window (eg: STC32G12K128)

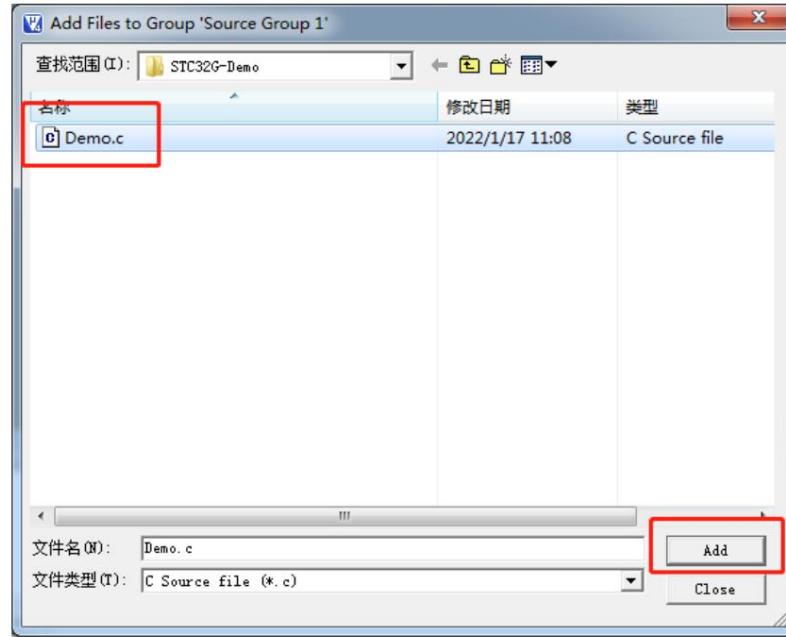


### 5.3.3 Adding source code files to the project

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

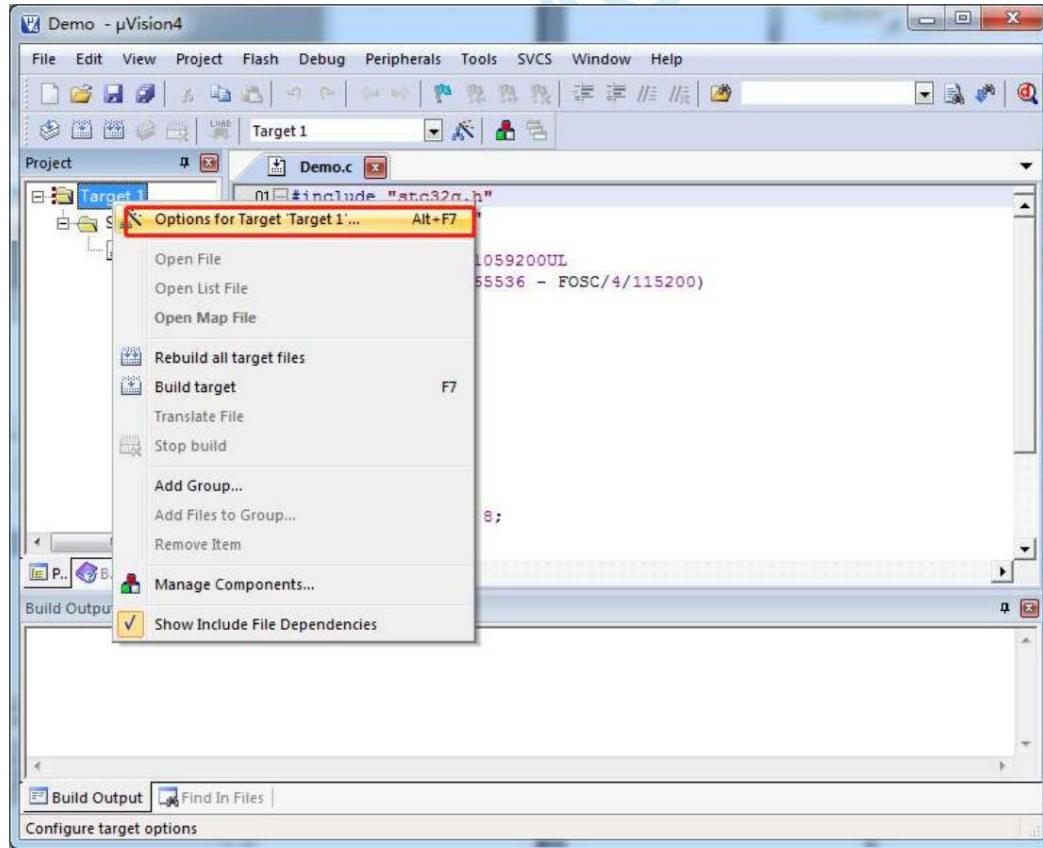


Select the edited code file to add to the project



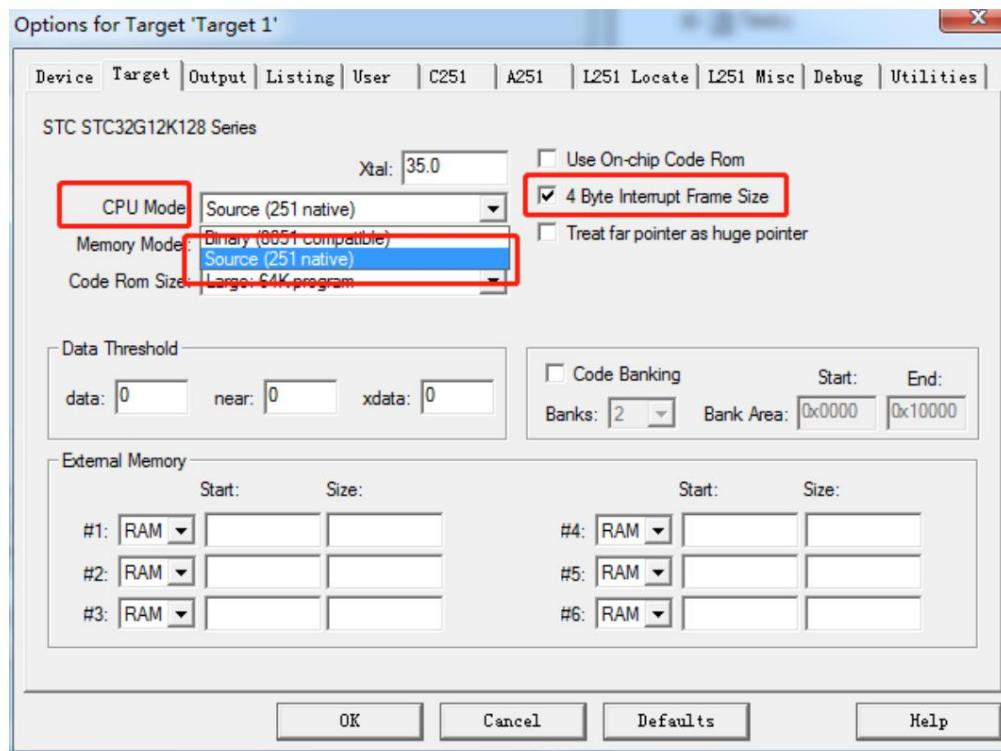
### 5.3.4 Set item 1 ("CPU Mode" select Source mode)

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



In the pop-up "Options for Target" select "Source (251 Native)"

In the "Target 1" window, select the "Target" option page, and select the "CPU Mode" drop-down option.



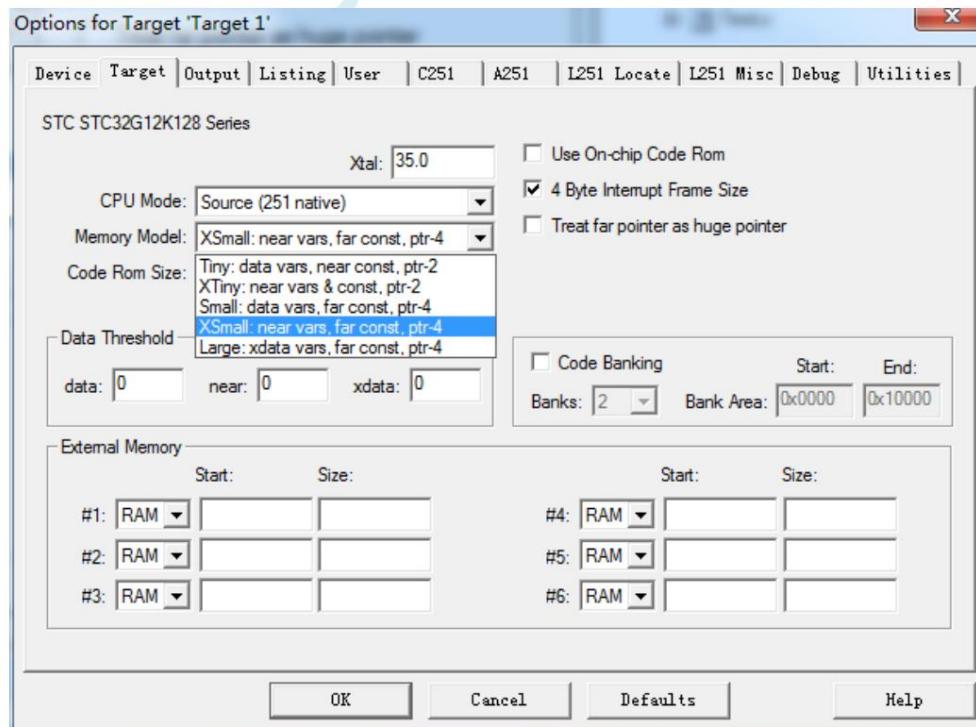
The command mode of 80251 has two modes: "Binary" and "Source". STC32G series currently only supports "Source" mode.

Since the STC32G series MCU pushes and pops out of the stack in the 4-byte mode, it is recommended to select "4 Byte Interrupt Frame Size" item also ticked

### 5.3.5 Set item 2 ("Memory Model" select XSmall mode)

Select the "XSmall: ..." mode in the "Memory Model" drop-down option. The memory mode of 80251 is as follows in the Keil environment

The 5 modes shown in the figure:



The various modes are compared in the following table:

Memory Model	Default variable type (data memory)	Default constant type (program memory)	Default pointer variable type	Pointer access range
Tiny mode	data	near	2 bytes	00:0000 to 00:FFFF
XTiny mode	edata	near	2 bytes	00:0000 to 00:FFFF
Small mode	data	far	4 bytes	00:0000 to FF:FFFF
<b>XSmall mode</b>	<b>edata</b>	<b>far</b>	<b>4 bytes 00:0000 to FF:FFFF</b>	
Large mode	xdata	far	4 bytes	00:0000 to FF:FFFF

Since the program logic address of STC32G is FE:0000H-FF:FFFFH, it needs to use a 24-bit address line to access it correctly.

The amount type (program memory type) must use the "far" type, and the default pointer variable must be 4 bytes.

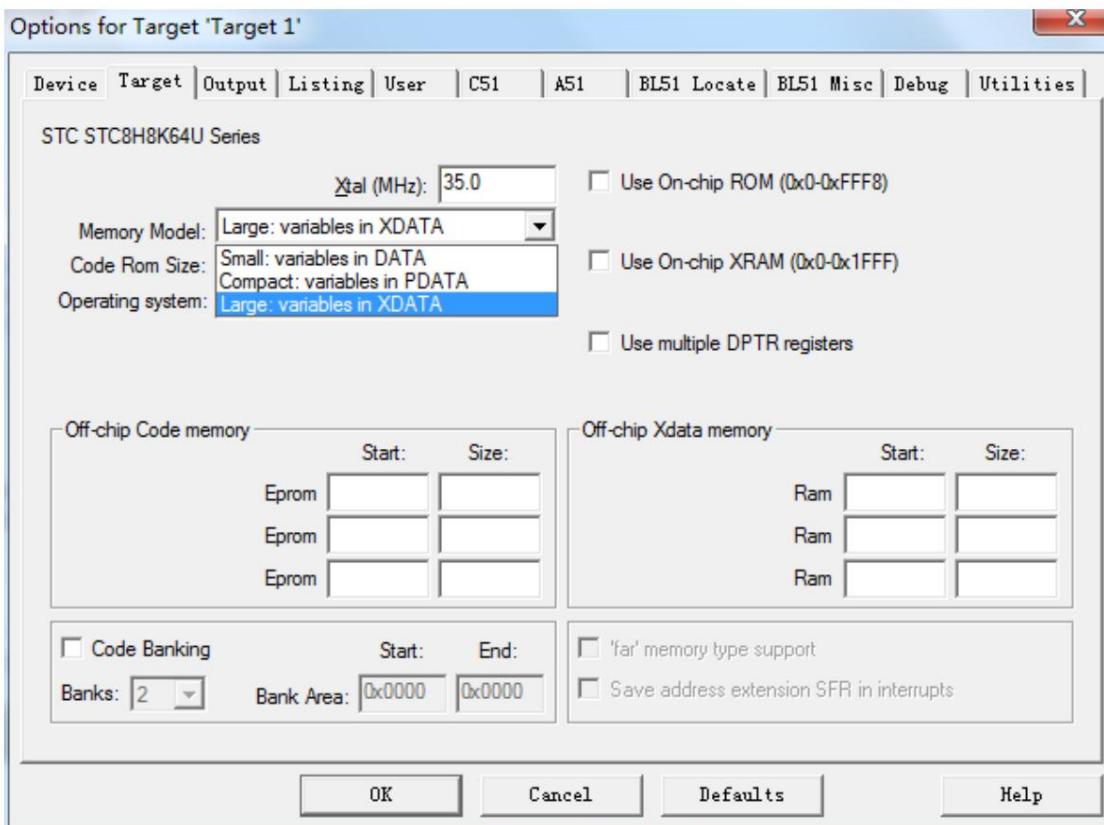
Therefore, it is not recommended to use the "Small", "Tiny" and "XTiny" modes. It is recommended to use the "XSmall" mode, which defaults to the variable defined in internal RAM (edata), single clock access, fast access speed, and STC32G12K128 series chips have 12K edata available;

When using "Small" mode, the variables are defined in the internal RAM (data) by default, single-clock access, and fast access (data is only 128 words by default section, when the user's RAM requirement exceeds 128 bytes, the Keil compiler will report an error, and the user needs to switch the storage mode to XSmall)

The amount is limited and it is easy to report errors, so it is not recommended to use it; it is not recommended to use the "Large" mode, although this mode can also correctly access all the STC32G 16M addressing space, but "Large" mode defines variables in the internal extended RAM (xdata) by default, and access requires 2~3 clocks.

slow

Corresponding to the STC8H8K64U series, the "Memory Model" in the Keil software has the following 3 options



The various modes are compared in the following table:

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
In order to	xdata	64K bytes (theoretical value)	X:0000 to X:FFFF

achieve higher efficiency in **Large mode**, it is generally recommended to select "Small" mode. When the compiler displays "error C249: 'DATA' SEGMENT"

"TOO LARGE" error, you need to manually allocate some relatively large arrays to the XDATA area through "xdata" (e.g.

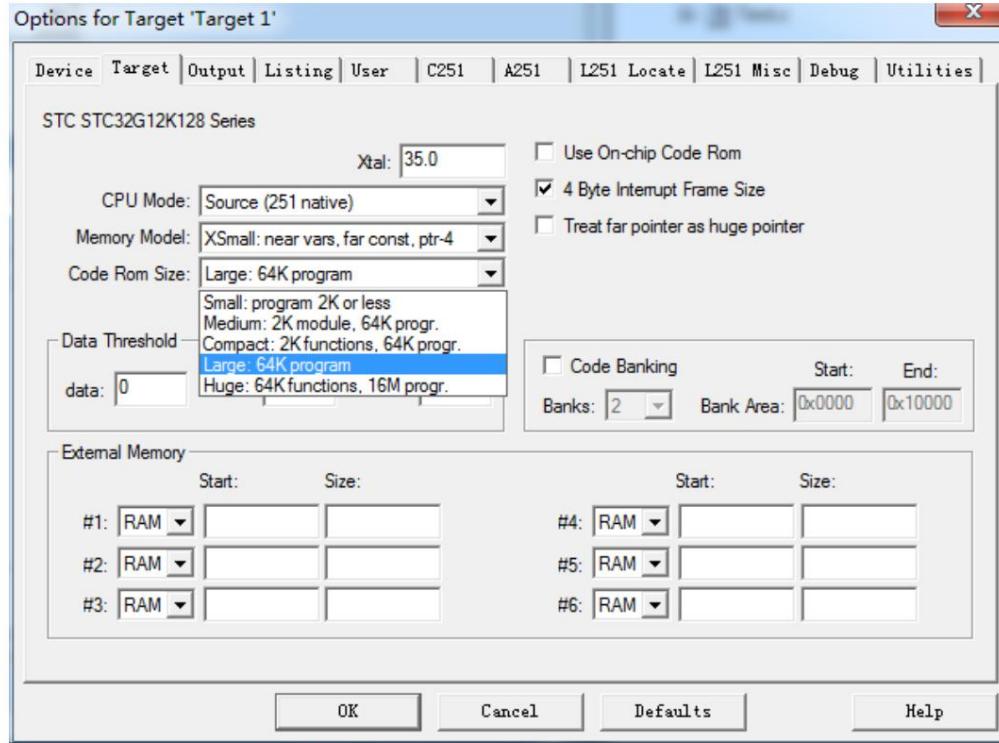
Such as: char xdata buffer[256];)

## 5.3.6 Set item 3 (“Code Rom Size” select Large or Huge mode

Mode)

Select "Large: ..." or "Huge: ..." mode from the drop-down option of "Code Rom Size"

The code size mode of 80251 has 5 modes in the Keil environment as shown in the following figure:

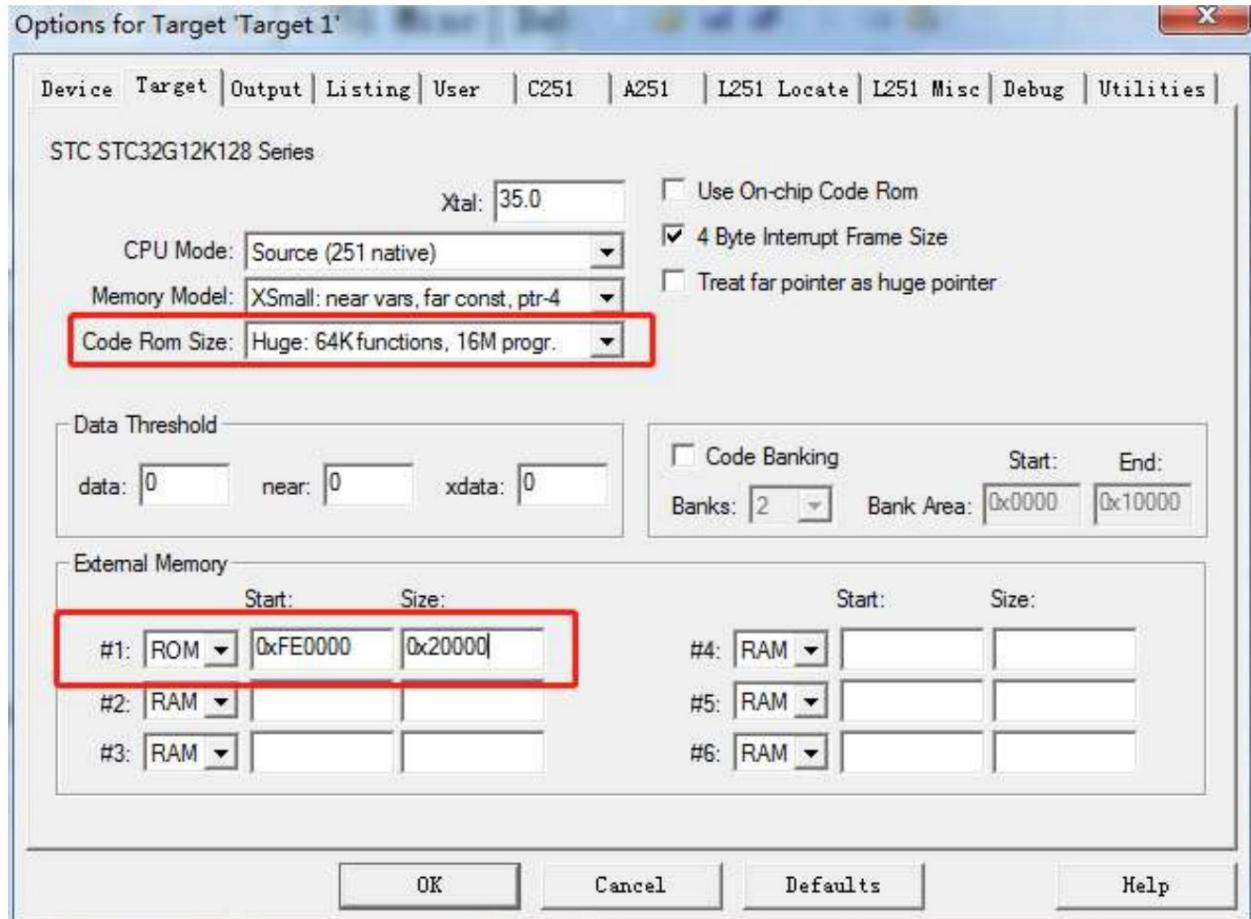


The various modes are compared in the following table:

Code Rom Size	jump/call instruction	code size limit	
		single function/module/file code size	total code size
Small mode	AJMP/ACALL	2K	2K
Medium mode	Internal module code uses 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 uses LJMP/ECALL External module code using EJMP/ECALL <i>(In a multi-file project, the code inside the file is an internal module block code, the code of other files is external module code)</i>	64K	16M

### 5.3.7 Setting item 4 (relevant settings of super 64K code)

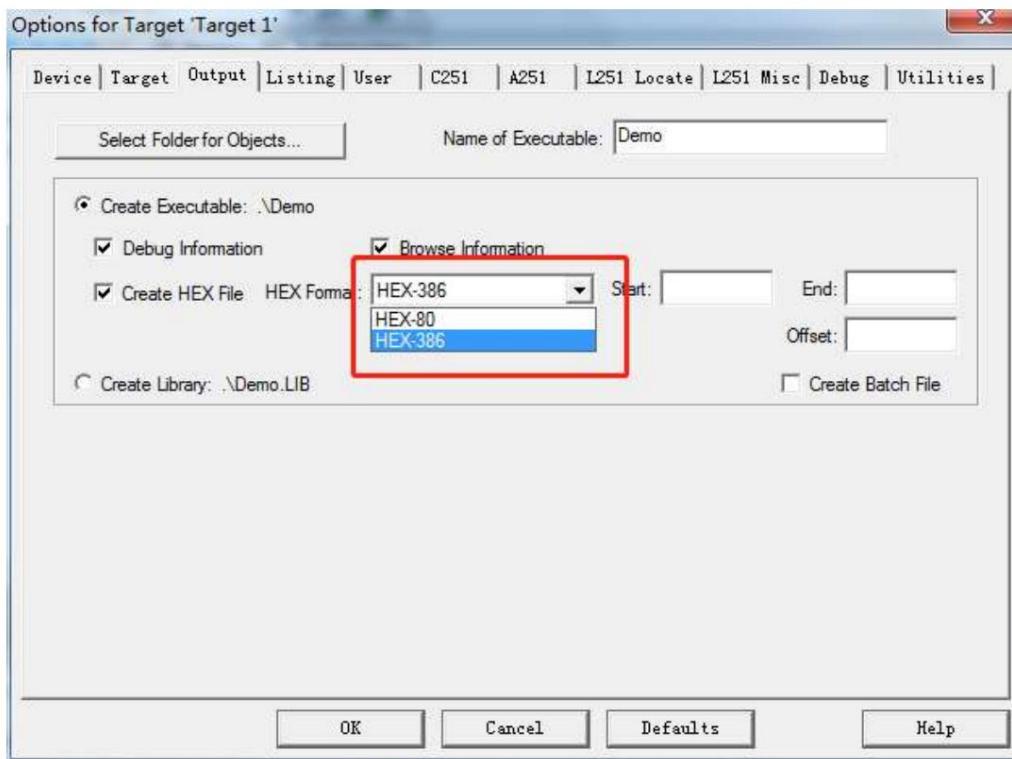
If the code size is within 64K, select "Large" mode. If the code size exceeds 64K, you need to select "Huge" mode, and it is necessary to ensure that the code size of a single function and a single file must be within 64K bytes, and the data volume of a single table must also be Must be within 64K bytes. At the same time, you need to make the settings as shown in the following figure:



### 5.3.8 Setting item 5 (HEX file format setting)

"Options for Target" In the Target 1" window, select the "Output" tab, and check the "Create HEX File" option.

If the program space exceeds 64K, the "HEX format" must select the "HEX-386" mode. Only the program space is within 64K, the "HEX format" can select the "HEX-80" mode;



After completing the above settings, click the compile button as shown in the figure below, if the code has no errors, the HEX file can be generated

## 5.4 Assembly code writing based on STC32G series in Keil

### 5.4.1 How to write an assembler with a code size of less than 64K

```

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

ORG      0000H      ;reset entry address
JMP      RESET      ;1. 64K Program-sized code can directly use the 0000H address
;       The compiler will automatically link the code where it starts
;2. Statements are available at the interrupt vector and the compiler will automatically
;       According to the actual compilation situation, intelligently replace it with A JMP/LJMP/EJMP

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:
NOP
NOP
RETI      ;interrupt function
;64K Program size interrupt usage RETI return

TIMER0_ISR
NOP
NOP
RETI

INT1_ISR
NOP
NOP
RETI

TIMER1_ISR
NOP
NOP
RETI

RESET:
MOV      SPX, #0100H      ;set stack pointer initial value
MOV      WTST, #0

MOV      P0M0,#0      ;system initialization
MOV      P0M1,#0

```

**MAINLOOP:**

<b>INC</b>	<b>P0</b>	
<b>LCALL</b>	<b>DELAY</b>	<b>;64K Program size function usage LCALL or ACALL transfer</b>
<b>JMP</b>	<b>MAINLOOP</b>	

**DELAY:**

<b>MOV</b>	<b>WR0,#5</b>
------------	---------------

**DELAY1:**

<b>DEC</b>	<b>WR0,#1</b>
<b>JNE</b>	<b>DELAY1</b>
<b>RET</b>	

**;64K Program size function usage RET return**

<b>END</b>	
------------	--

**5.4.2 How to write an assembler program whose code size exceeds 64K**

<b>P0</b>	<b>DATA</b>	<b>080H</b>
<b>P0M1</b>	<b>DATA</b>	<b>093H</b>
<b>P0M0</b>	<b>DATA</b>	<b>094H</b>
<b>WTST</b>	<b>DATA</b>	<b>0E9H</b>

<b>ORG</b>	<b>OFF:0000H</b>
<b>JMP</b>	<b>RESET</b>

**;reset entry address**  
**; 1. sized code exceeds program ,**  
**; ORG ;2. be used 0xx:xxxx**  
**; JMP**  
**Statements are available at the interrupt vector and the compiler will automatically**  
**; According to the actual compilation situation, intelligently replace it with AJMP/LJMP/EJMP**

<b>ORG</b>	<b>OFF:0003H</b>
<b>JMP</b>	<b>INT0_ISR</b>
<b>ORG</b>	<b>OFF:000BH</b>
<b>JMP</b>	<b>TIMER0_ISR</b>
<b>ORG</b>	<b>OFF:0013H</b>
<b>JMP</b>	<b>INT1_ISR</b>
<b>ORG</b>	<b>OFF:001BH</b>
<b>JMP</b>	<b>TIMER1_ISR</b>

**;interrupt entry address**

<b>ORG</b>	<b>OFF:0200H</b>
------------	------------------

**INT0\_ISR:**

<b>NOP</b>	<b>; interrupt function</b>
<b>NOP</b>	
<b>RETI</b>	<b>; interrupt return</b>

**TIMER0\_ISR:**

<b>NOP</b>
<b>NOP</b>
<b>RETI</b>

**INT1\_ISR:**

<b>NOP</b>
<b>NOP</b>
<b>RETI</b>

**TIMER1\_ISR:**

*NOP*  
*NOP*  
*RETI*

*RESET:*

*MOV SPX, #0100H* ;set stack pointer initial value  
*MOV WTST, #0*  
  
*MOV P0M0,#0* ;system initialization  
*MOV P0M1,#0*  
  
*MAINLOOP:*

*INC P0*  
*ECALL DELAY* ;<sub>overtake</sub> 64K Program size function usage *ECALL* transfer  
*JMP MAINLOOP*

*ORG OFE:0000H*

*DELAY:*

*MOV WR0, #1000*

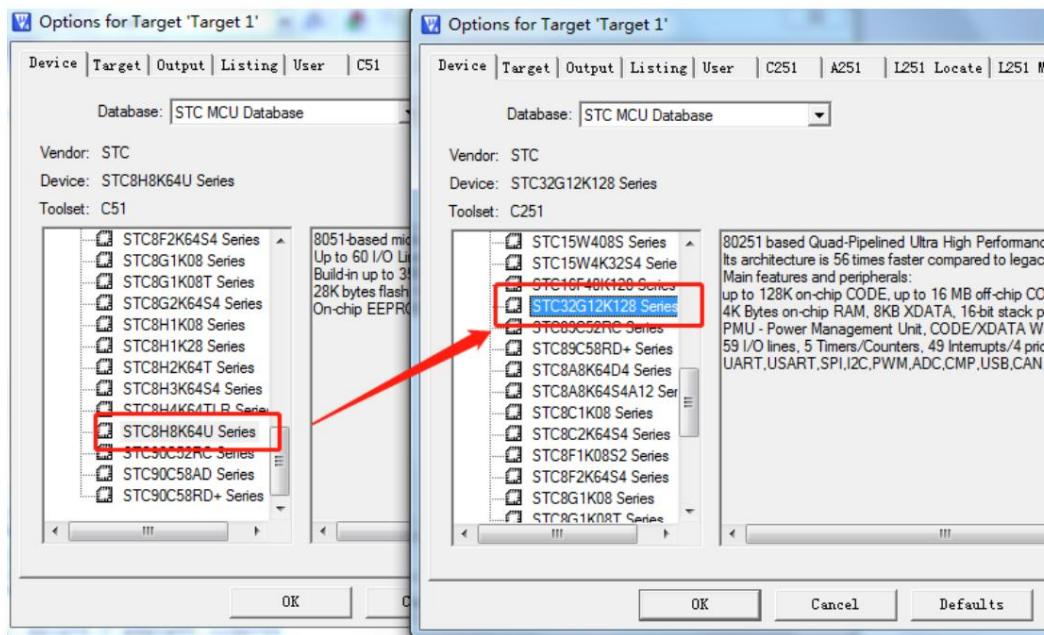
*DELAY1:*

*DEC WR0,#1*  
*JNE DELAY1*  
*ERET* ;<sub>overtake</sub> 64K Program size function usage *ERET* return

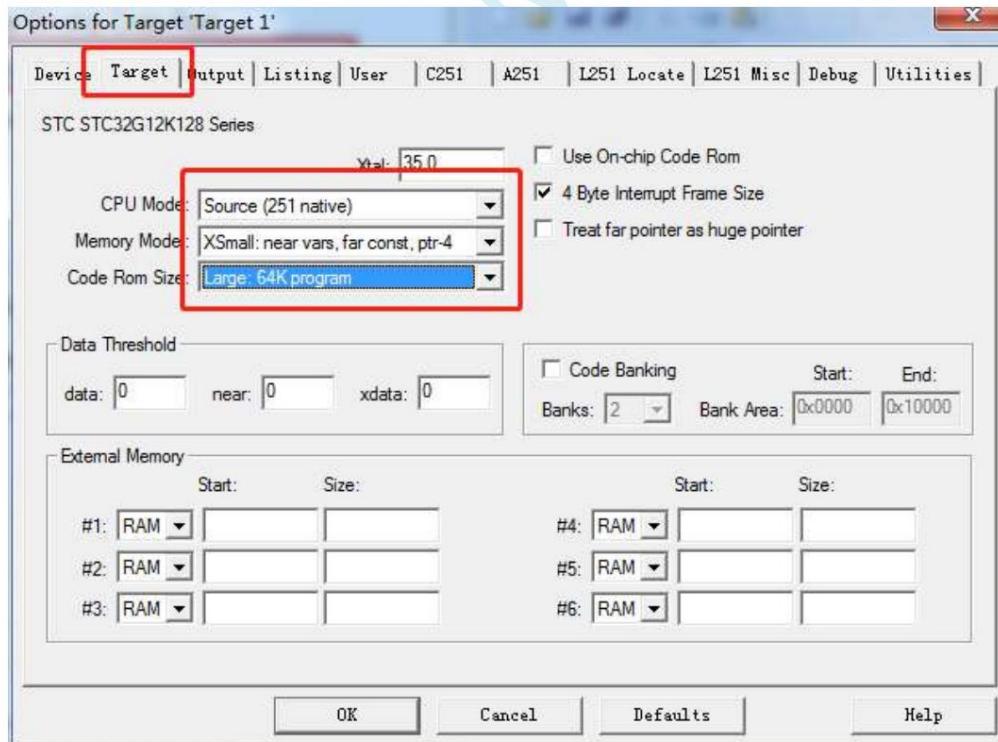
*END*

## 5.5 STC8H series projects are converted to STC32G series

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



2. In the "Target" page of "Options for Target 'Target1'", set as shown below



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

```
01 // #include "stc8h.h"
02
03
04 #include "stc32g.h"
05
06 #include "intrins.h"
07
08 void delay()
09 {
10     int i;
11
12     for (i=0; i<1000; i++)
13     {
14         _nop_();
15         _nop_();
16         _nop_();
17         _nop_();
18     }
19 }
20
21 void main()
```

Build Output

```
Build target 'Target 1'
compiling Test.c...
linking...
Program Size: data=8.0 edata+hdata=256 xdata=0 const=0 code=53
"Test" - 0 Error(s), 0 Warning(s).
```

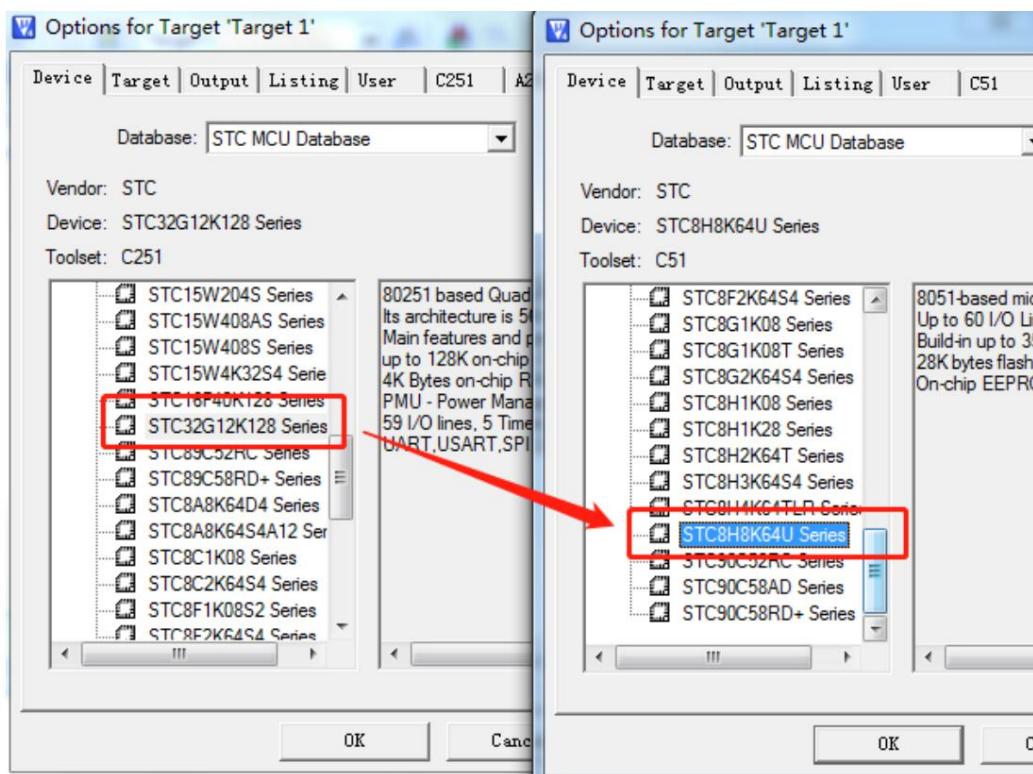
Simulation

Since most of the SFR and XFR of the STC32G series and STC8H series are completely compatible, after the above settings,

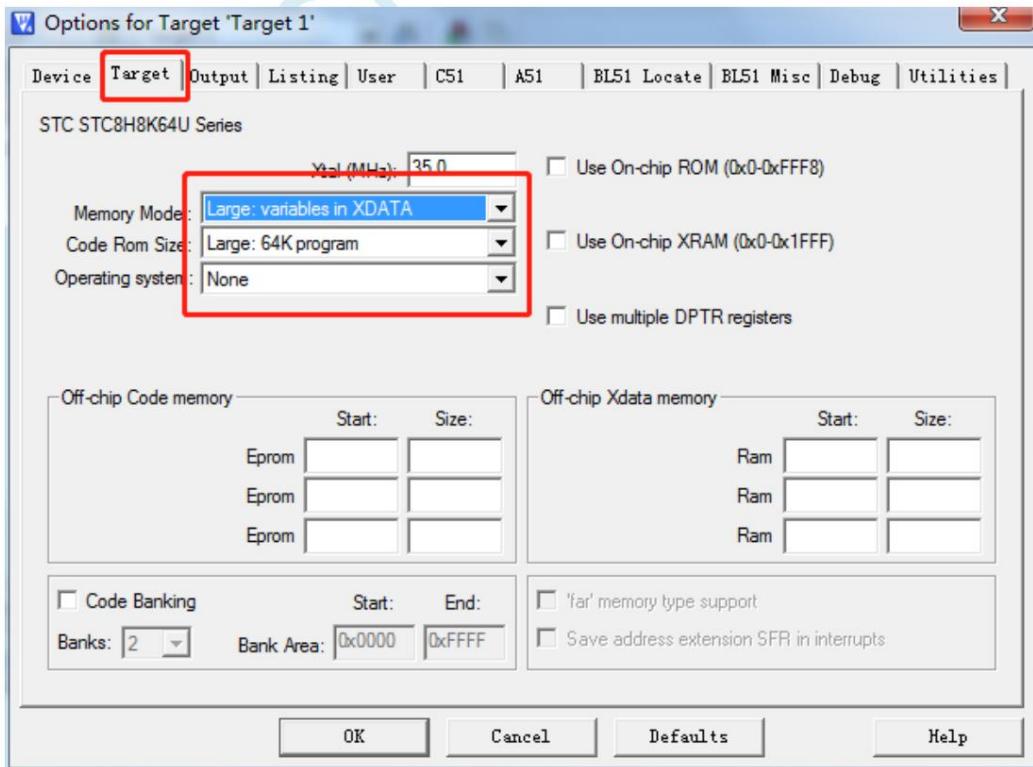
The compilation can basically pass. If there is a small amount of incompatibility, it can be slightly modified.

## 5.6 STC32G series projects are converted to STC8H series

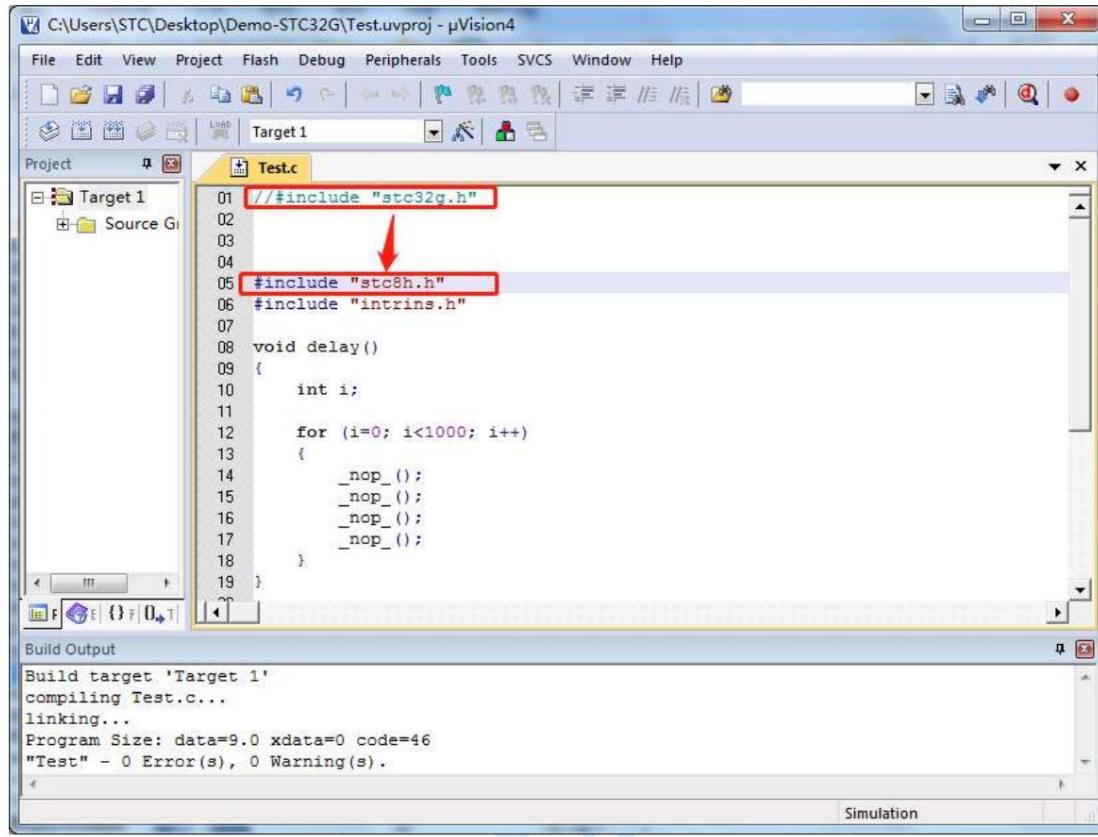
1. Change the target MCU model. As shown in the figure below, change the original MCU model "STC32G12K128" of the STC32G project to Model "STC8H8K64U"



2. In the "Target" page of "Options for Target "Target1""", set as shown below



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



```
01 // #include "stc32g.h"
02
03
04
05 #include "stc8h.h"           ← Red arrow points here
06 #include "intrins.h"
07
08 void delay()
09 {
10     int i;
11
12     for (i=0; i<1000; i++)
13     {
14         _nop_();
15         _nop_();
16         _nop_();
17         _nop_();
18     }
19 }
```

Build Output

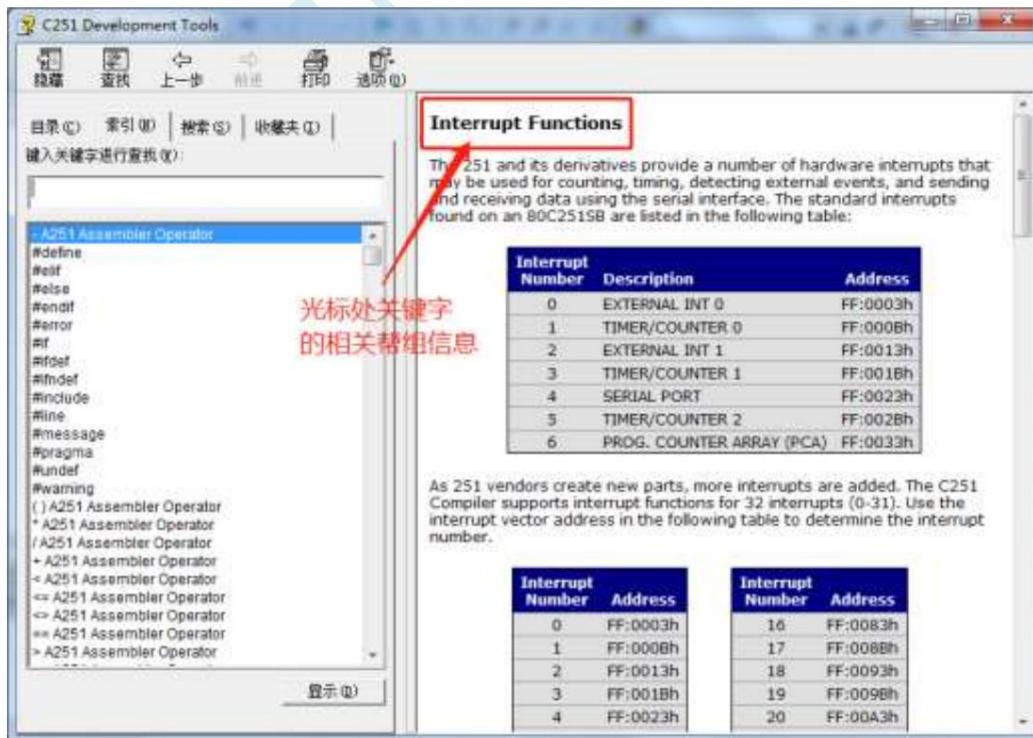
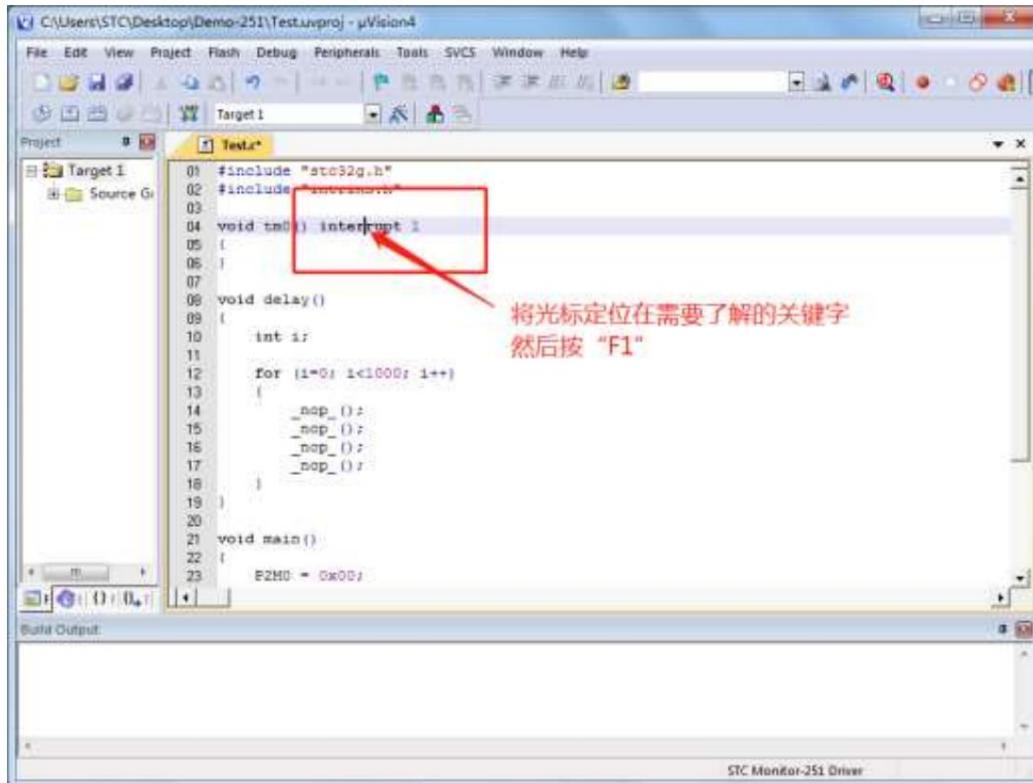
```
Build target 'Target 1'
compiling Test.c...
linking...
Program Size: data=9.0 xdata=0 code=46
"Test" - 0 Error(s), 0 Warning(s).
```

Simulation

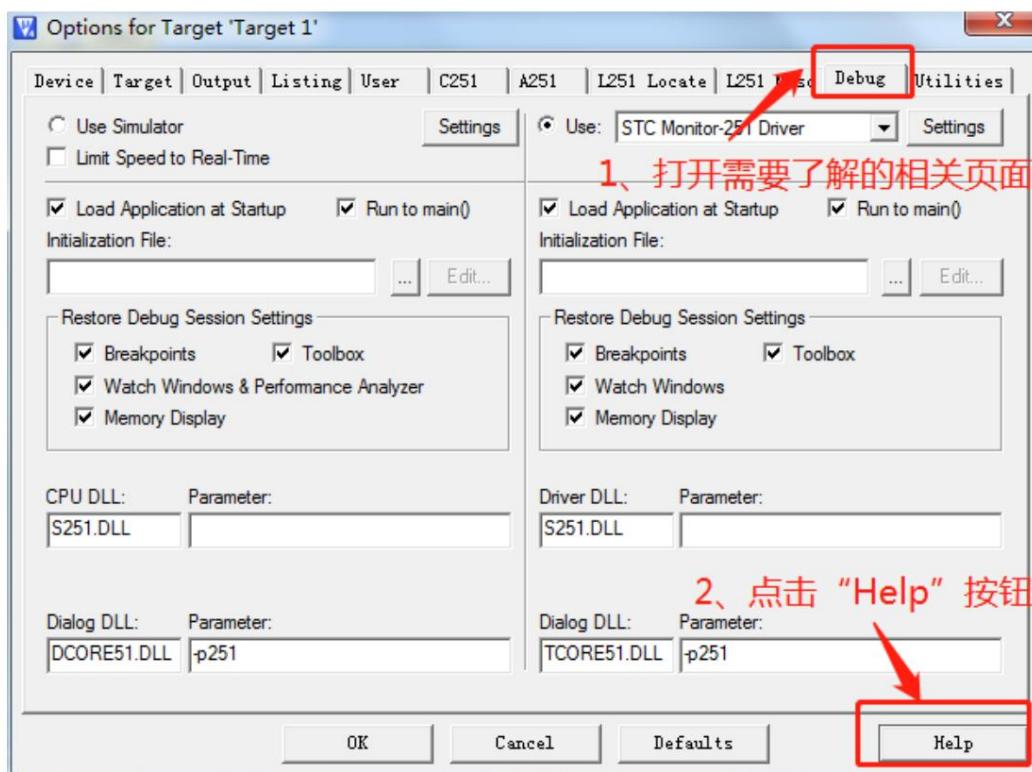
Since most of the SFR and XFR of the STC32G series and STC8H series are completely compatible, after the above settings, The compilation can basically pass. If there is a small amount of incompatibility, it can be slightly modified.

## 5.7 The easy way to get help in Keil software

Keil software provides a complete set of help files. For general software usage and programming problems, the help groups that directly use Keil software are basically the same. can be resolved. As shown below:

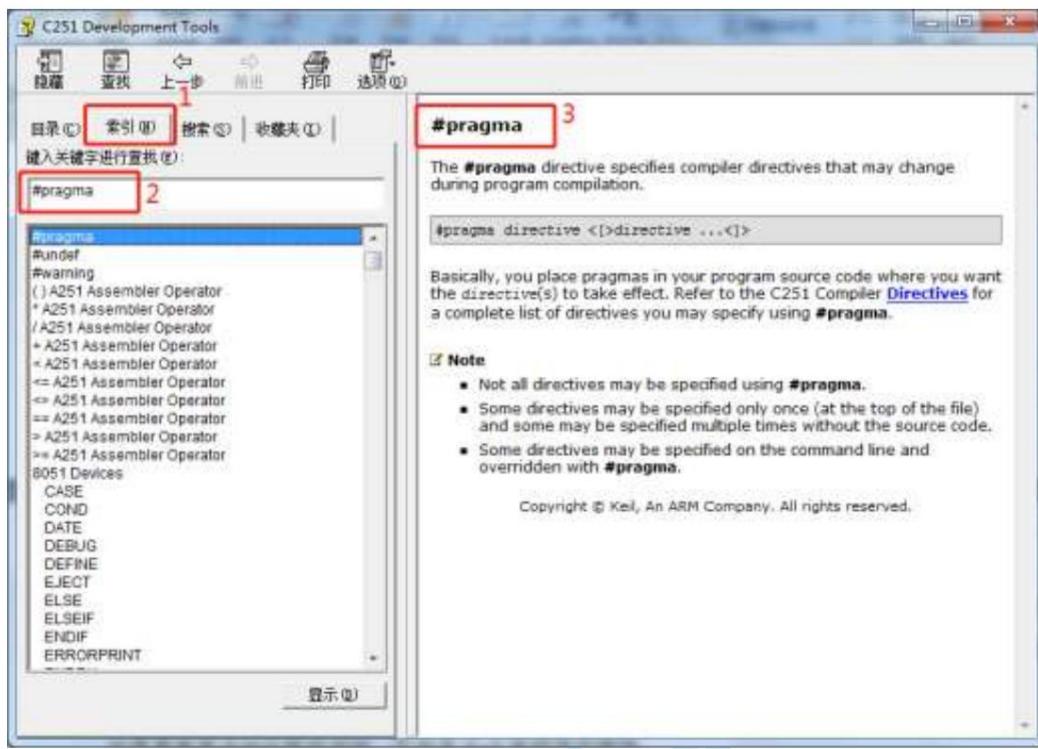


If you need to know the relevant settings in the project settings, you can get the help group as shown in the figure below.



In addition, you can also directly input the content you want to know in the help group window. For example, if you need to know how to set special pragmas in your program, you can

Enter "#pragma" in the search box as shown below



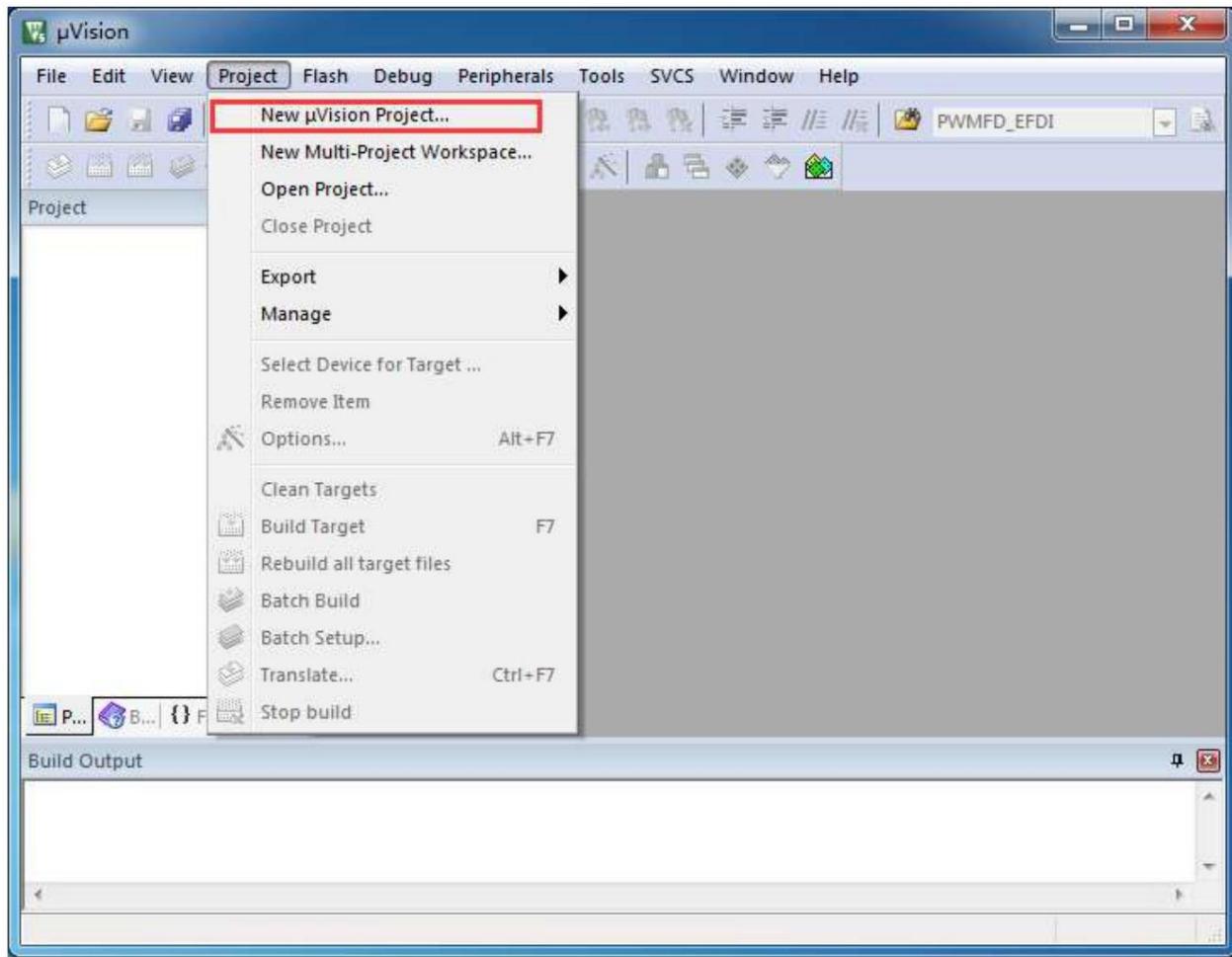
If you need more detailed help group details, you can log in to Keil's official website for inquiries.

## 5.8 Methods of building multi-file projects in Keil

In Keil, generally relatively small projects have only one source file, but for some slightly complex projects, multiple source files are often required.

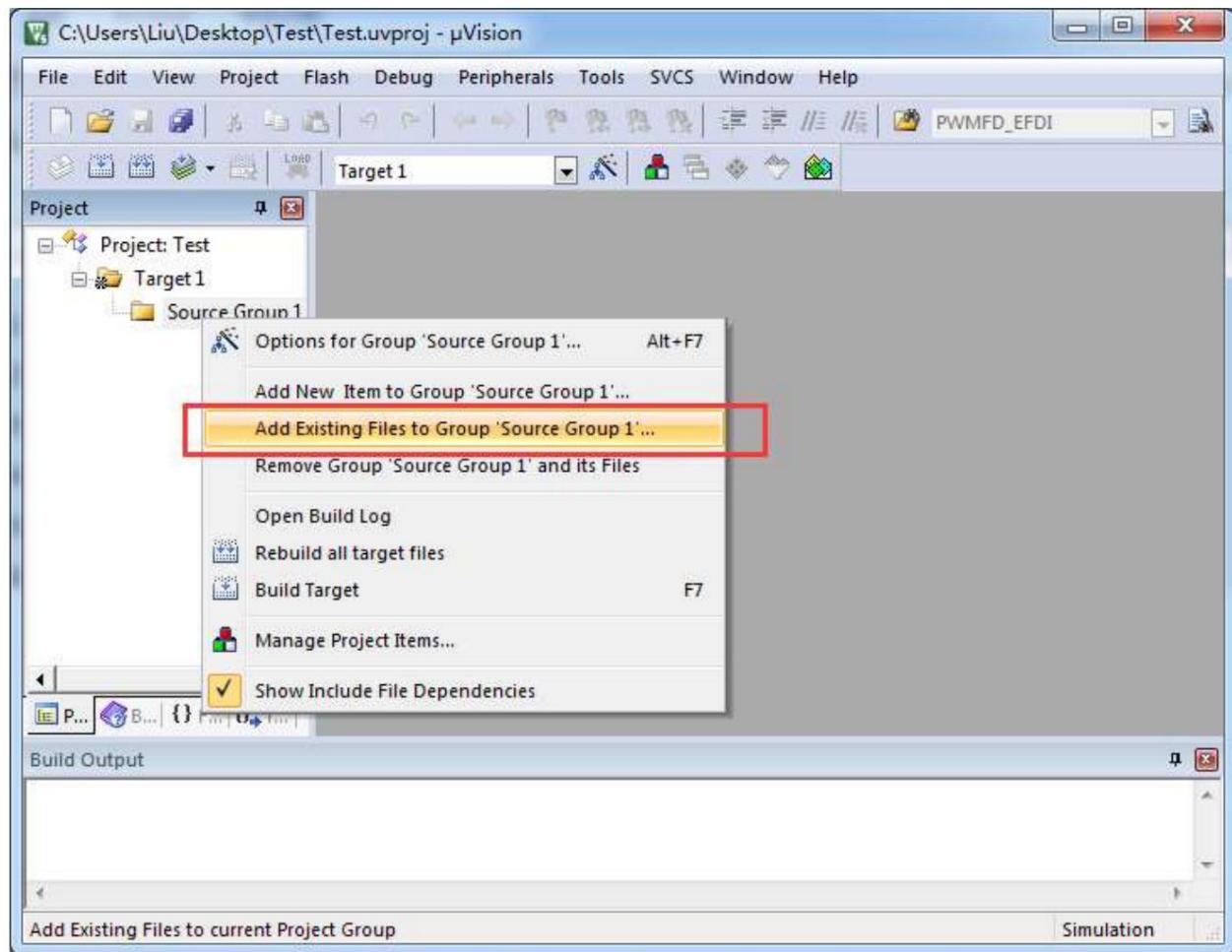
Here's how to create a multi-file project:

1. First open Keil, select "New uVision Project..." in the menu "Project"

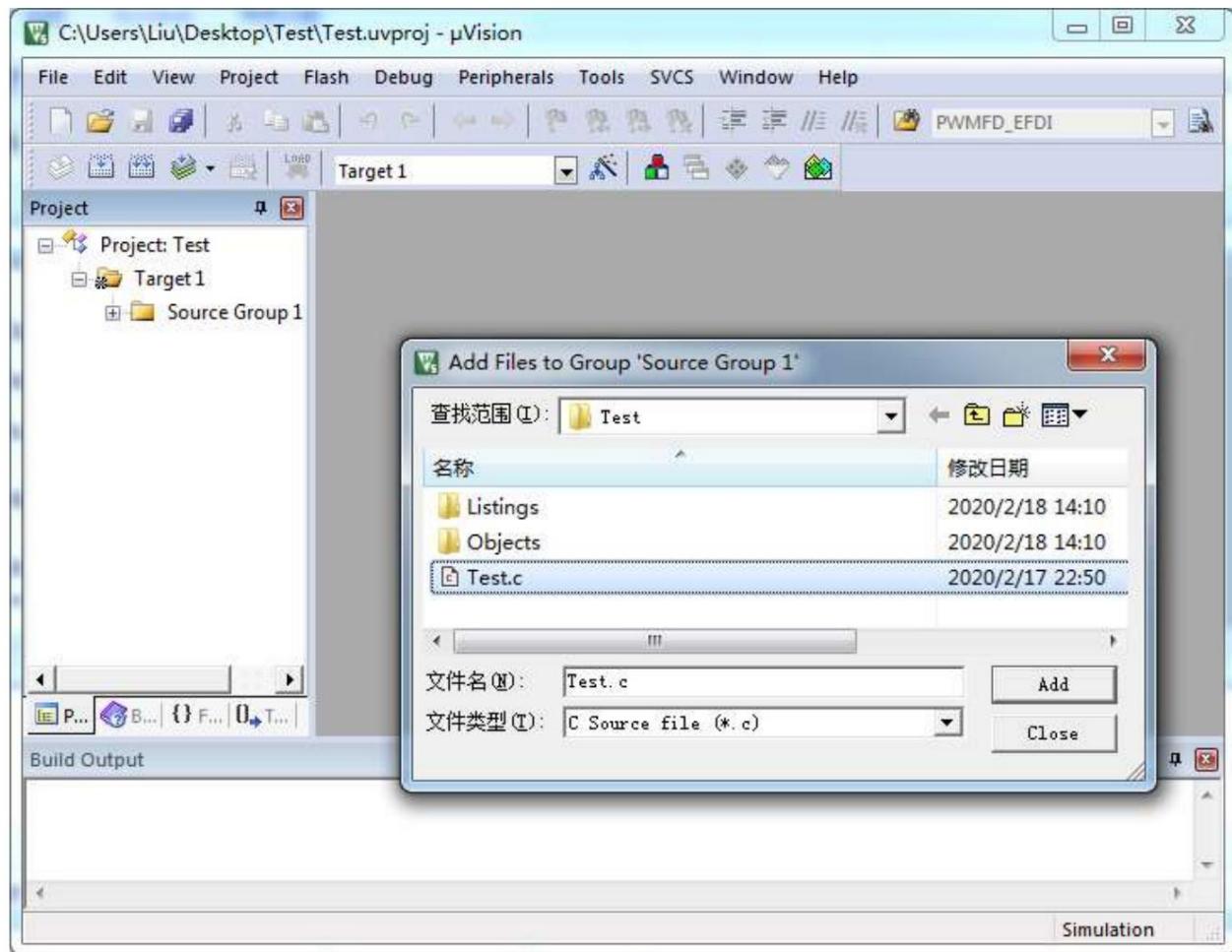


To complete the establishment of an empty project

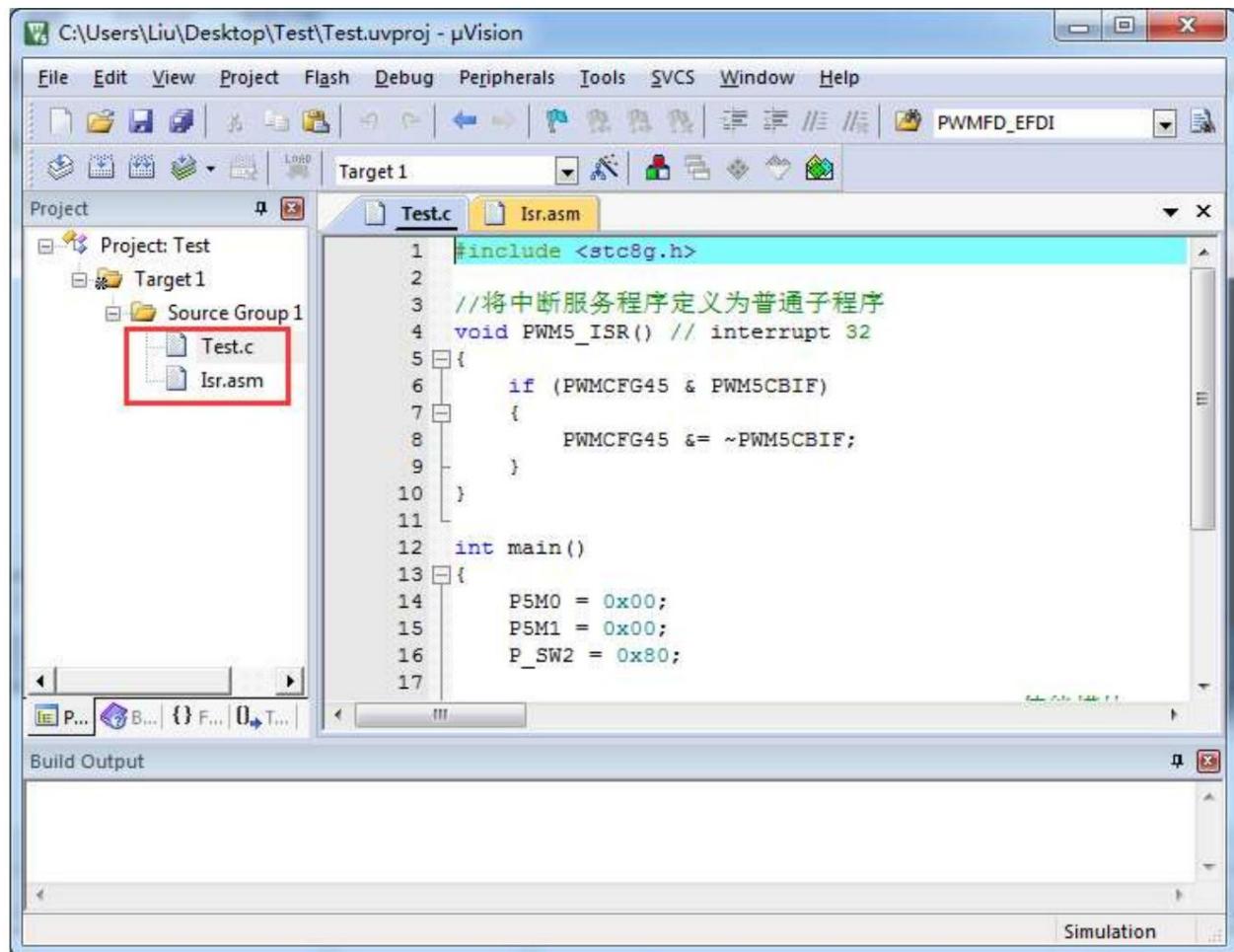
2. In the project tree of the empty project, right-click "Source Group 1" and select "Add Existing Files to Group "Source Group 1" ..." in the context menu



3. In the pop-up file dialog box, add the source file multiple times



As shown in the figure below, the establishment of a multi-file project can be completed



## 5.9 Handling of compiling errors in Keil when the interrupt number is greater than 31

Note: At present, the C51 and C251 compilers of various versions of Keil only support 32 interrupt numbers (0~31). No. more than 32 are required. But for the current Keil version, the method in this chapter can only be used for temporary solution.

### 5.9.1 Use the popular Internet extension tool

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



Click the "Open" button, locate the Keil installation directory, and click "OK". Since the version of Keil is constantly updated, and there are too many early versions, it is impossible to collect them all. Here are the tested versions of C51.EXE and C251.EXE.

Tested versions of **C51.EXE** :

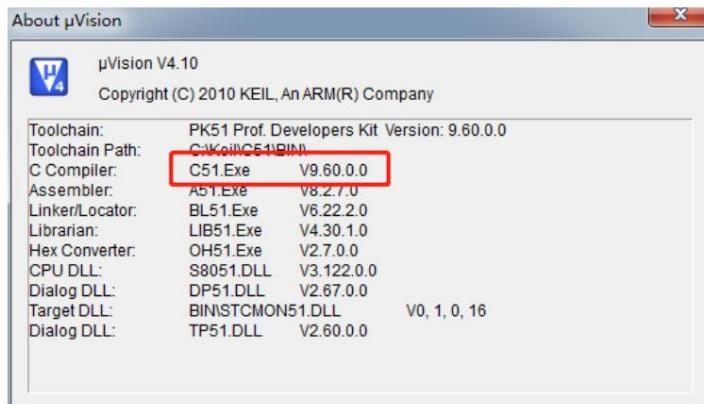
V6.12.0.1  
V8.8.0.1  
V9.0.0.1  
V9.1.0.1  
V9.53.0.0  
V9.54.0.0  
V9.57.0.0  
V9.59.0.0  
V9.60.0.0

Tested versions of **C251.EXE** :

V5.57.0.0  
V5.60.0.0

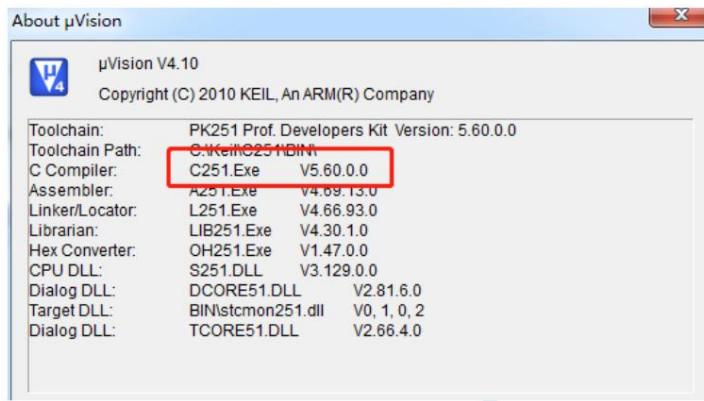
How to check the **C51.EXE** version:

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



Check the method of **C251.EXE** version:

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



## 5.9.2 Use reserved interrupt number for transfer

In Keil's C251 compilation environment, the interrupt number only supports 0~31, that is, the interrupt vector must be less than 0100H.

Interrupt Number	Address
0	FF:0003h
1	FF:000Bh
2	FF:0013h
3	FF:001Bh
4	FF:0023h
5	FF:002Bh
6	FF:0033h
7	FF:003Bh
8	FF:0043h
9	FF:004Bh
10	FF:0053h
11	FF:005Bh
12	FF:0063h
13	FF:006Bh
14	FF:0073h
15	FF:007Bh
16	FF:0083h
17	FF:008Bh
18	FF:0093h
19	FF:009Bh
20	FF:00A3h
21	FF:00ABh
22	FF:00B3h
23	FF:00BBh
24	FF:00C3h
25	FF:00CBh
26	FF:00D3h
27	FF:00DBh
28	FF:00E3h
29	FF:00EBh
30	FF:00F3h
31	FF:00FBh

The **interrupt** function attribute specifies that the associated function is an interrupt service routine. For example:

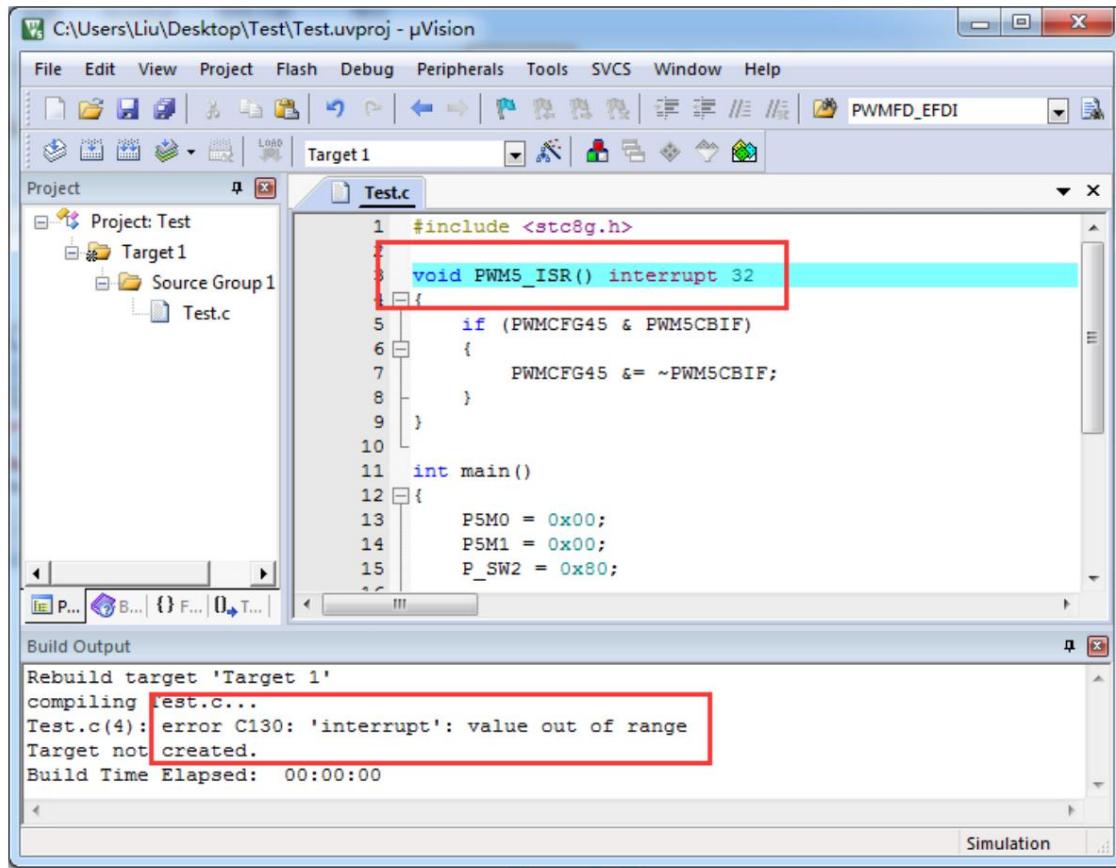
```
unsigned int interruptcnt;
```

The following table is a list of interrupts for all current series of STC:

interrupt number	interrupt vector	interrupt type
0	0003 H	INT0
1	000B H Timer 0	
2	0013 H	INT1
3	001B H Timer 1	
4	0023 H 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	0063H Timer 2	
13	006B H	
14	0073H System	internal interrupt

15	007BH System	internal interrupt
16	0083H	INT4
17	008BH Serial	port 3
18	0093H Serial	port 4
19	009BH Timer	3
20	00A3H Timer	4
twenty one	00ABH Comparator	
twenty four	00C3H	I2C
25	00CBH USB	
26	00D3H PWMA	
27	00DBH PWMB	
28	00E3H	CAN1
29	00EBH CAN2	
30	00F3H	LIN
36	0123H RTC	
37	012BH P0 port	interrupt
38	0133H P1 port	interrupt
39	013BH P2 port	interrupt
40	0143H P3 port	interrupt
41	014BH P4 port	interrupt
42	0153H P5 port	interrupt
43	015BH P6 port	interrupt
44	0163H P7 port	interrupt
45	016BH P8 port	interrupt
46	0173H 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 subsequent interrupt service routines will compile errors in keil, as shown in the following figure:



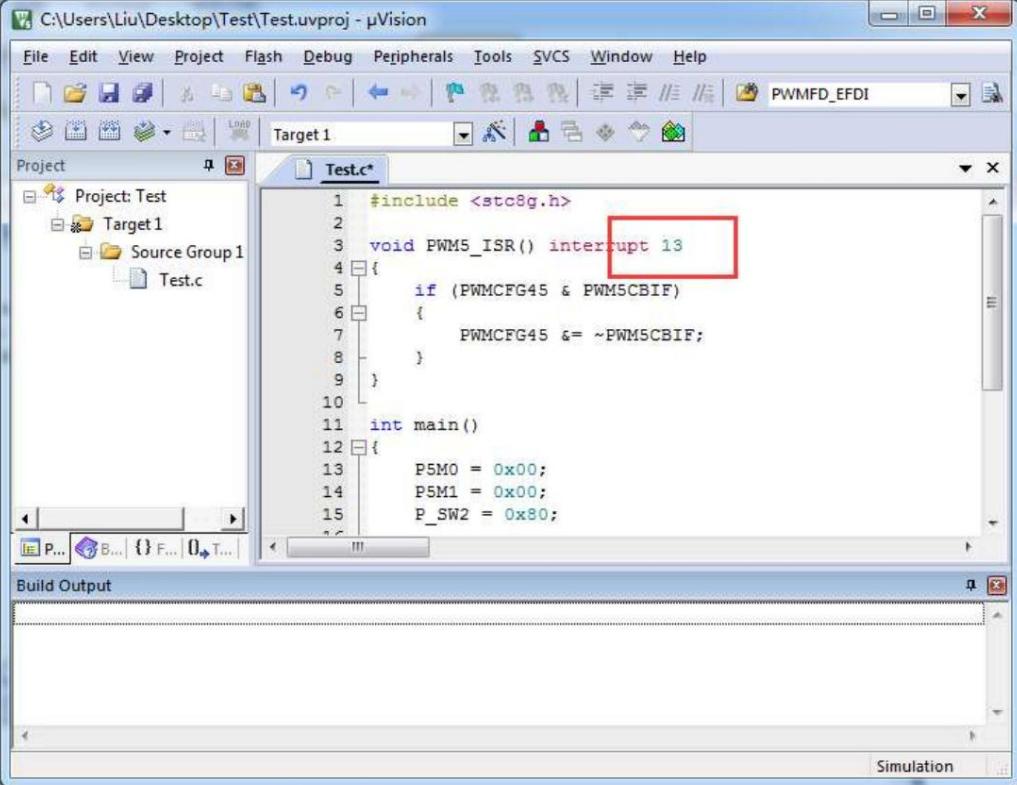
There are three ways to deal with this error: (all require the help of assembly code, and method 1 is preferred)

## Method 1: Borrow interrupt vector 13

Among the interrupts 0~31, the 13th is the reserved interrupt number, we can borrow this interrupt number

The operation steps are as follows:

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

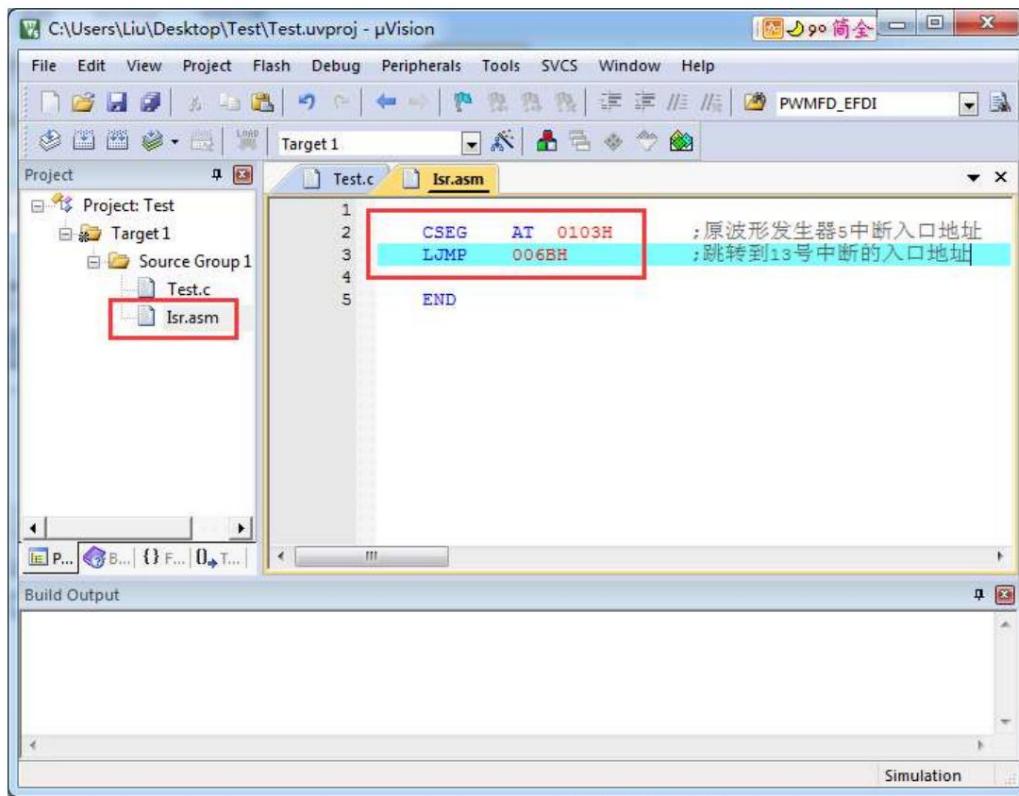


The screenshot shows the µVision IDE interface with a project named "Test". The "Test.c" file is open, displaying the following C code:

```
1 #include <stc8g.h>
2
3 void PWM5_ISR() interrupt 13
4 {
5     if (PWMCFG45 & PWM5CBIF)
6     {
7         PWMCFG45 &= ~PWM5CBIF;
8     }
9 }
10
11 int main()
12 {
13     P5M0 = 0x00;
14     P5M1 = 0x00;
15     P_SW2 = 0x80;
16 }
```

A red box highlights the line "interrupt 13" in the function definition.

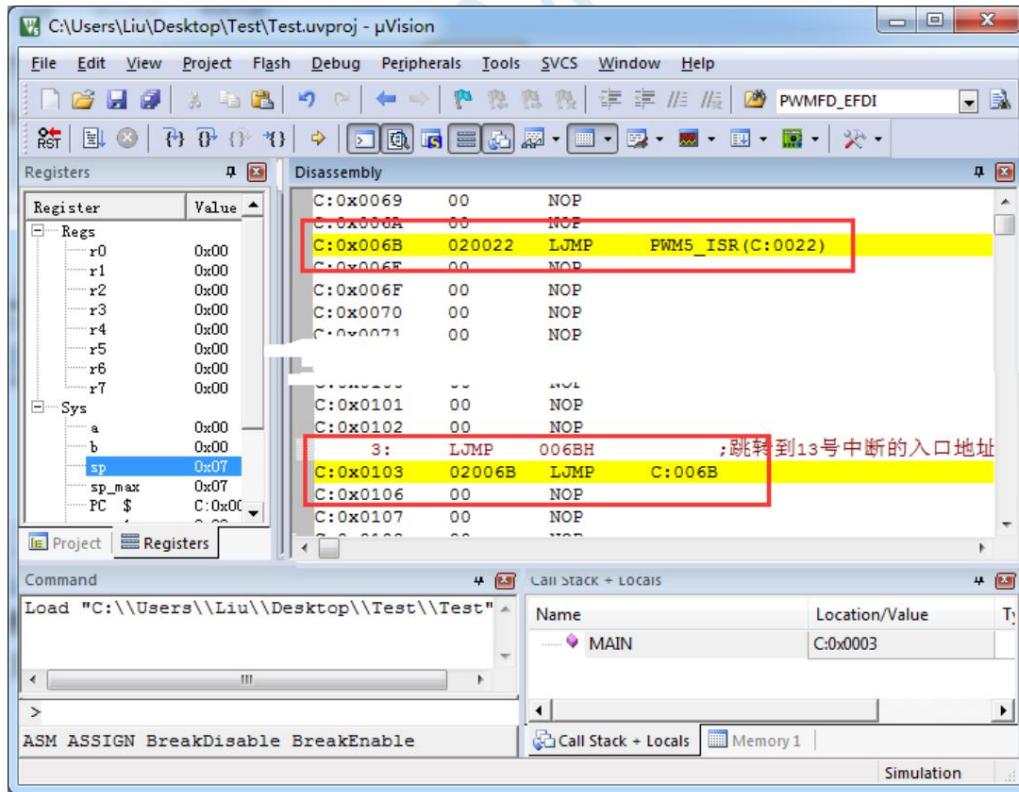
2. Create a new assembly language file, such as "isr.asm", add it to the project, and add a "LJMP 006BH" to the address "0103H", as shown below:



3. The compilation can be passed.

At this time, after compiling with Keil's C51 compiler, there is a "LJMP PWM5\_ISR" at 006BH, and a line at 0103H

"LJMP 006BH", as shown below:



When the PWM5 interrupt occurs, the hardware will automatically jump to address 0103H to execute "LJMP 006BH", and then execute "LJMP 006BH" at 006BH.

Line "LJMP PWM5\_ISR" to jump to the real interrupt service routine, as shown below:

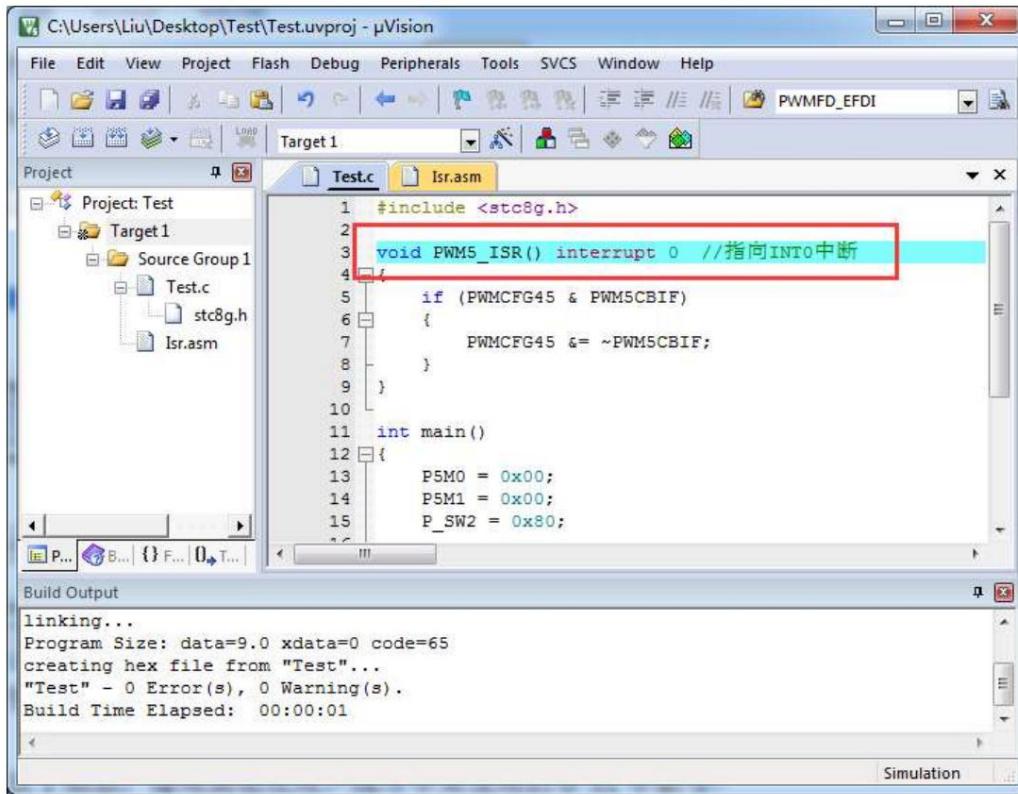
The screenshot shows the μVision IDE interface with the following details:

- File menu:** File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help.
- Toolbar:** Includes icons for RST, Open, Save, Build, Run, Stop, and various simulation and memory access tools.
- Registers window:** Shows the state of registers (Regs and Sys) and the stack pointer (sp) which is highlighted in blue.
- Disassembly window:** Displays assembly code for the PWM5\_ISR() interrupt. The RETI instruction at address C:0x002E is circled in red.
- Call Stack + Locals window:** Shows the current call stack and local variable values.
- Command window:** Displays the command to load the project.
- ASM ASSIGN BreakDisable BreakEnable:** A status message indicating assembly assignment and break control settings.

After the execution of the interrupt service routine is completed, return through the RETI instruction. The entire interrupt response process just executes one more LJMP statement. already.

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

For example, in the user's code, if the INTO interrupt is not used, the above code can be modified similar to method 1:



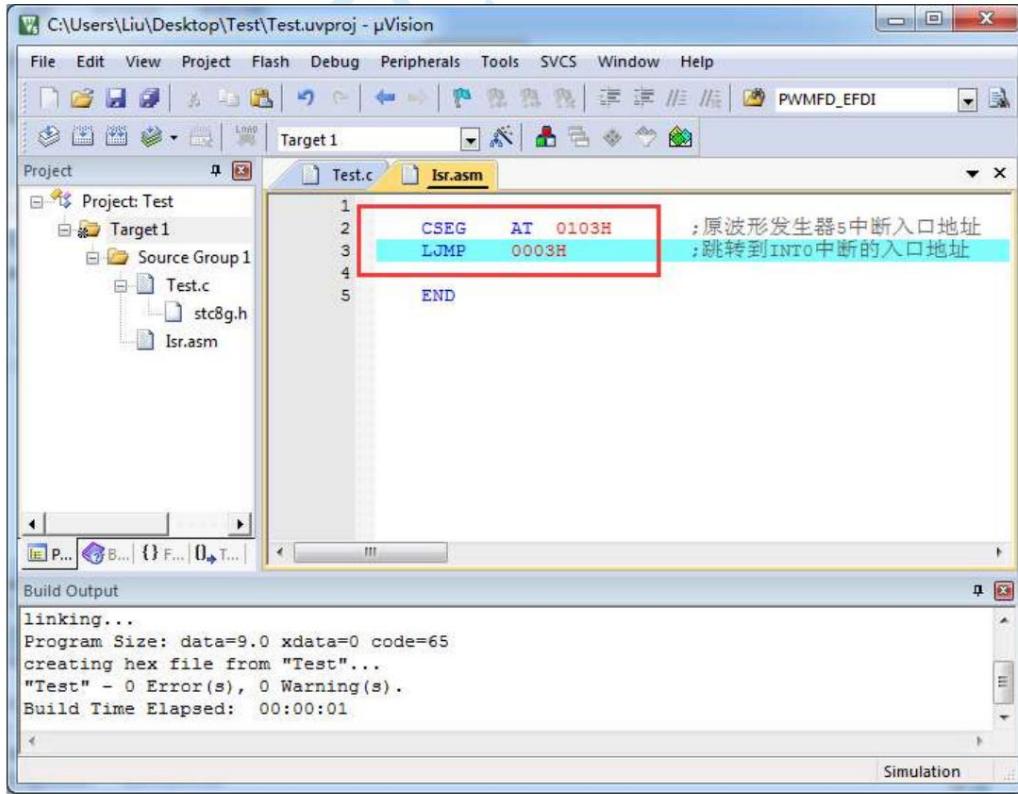
The screenshot shows the μVision IDE interface with the project 'Test' open. The assembly file 'Isr.asm' is selected. The code is as follows:

```

1 #include <stc8g.h>
2
3 void PWM5_ISR() interrupt 0 //指向INT0中断
4 {
5     if (PWMCFG45 & PWM5CBIF)
6     {
7         PWMCFG45 &= ~PWM5CBIF;
8     }
9 }
10
11 int main()
12 {
13     P5M0 = 0x00;
14     P5M1 = 0x00;
15     P_SW2 = 0x80;

```

The line 'void PWM5\_ISR() interrupt 0 //指向INT0中断' is highlighted with a red box.



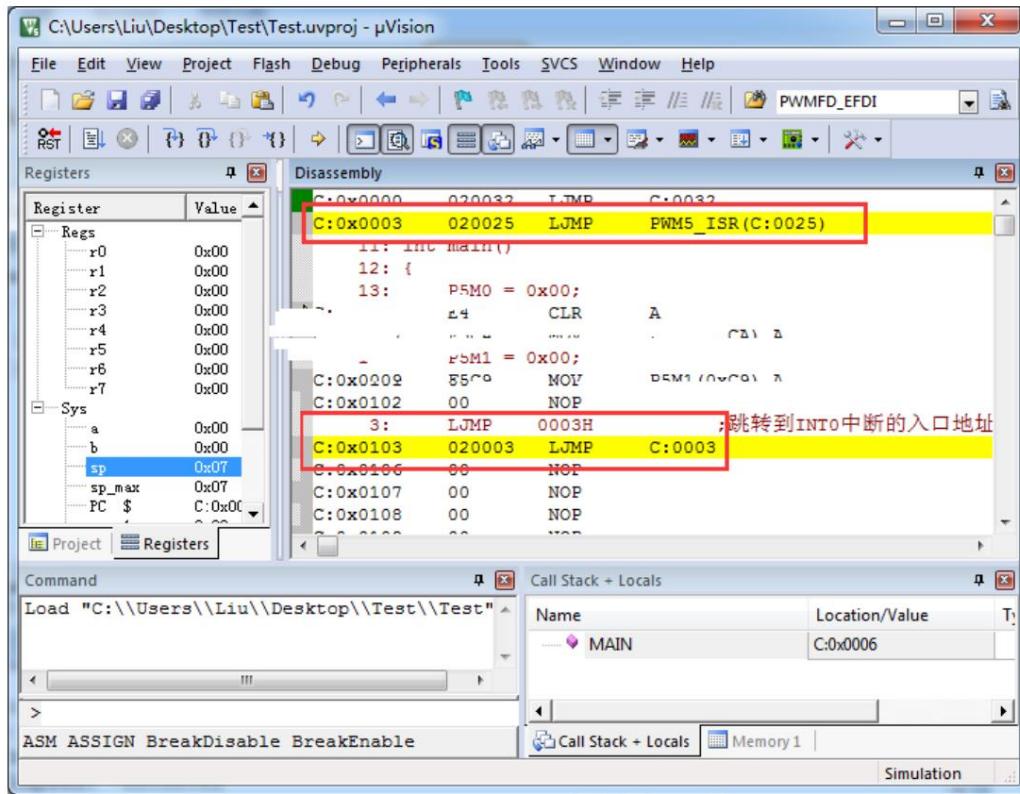
The screenshot shows the μVision IDE interface with the project 'Test' open. The assembly file 'Isr.asm' is selected. The code is as follows:

```

1 CSEG    AT 0103H ;原波形发生器5中断入口地址
2 LJMP    0003H ;跳转到INT0中断的入口地址
3
4 END

```

The lines 'CSEG AT 0103H ;原波形发生器5中断入口地址' and 'LJMP 0003H ;跳转到INT0中断的入口地址' are highlighted with a red box.



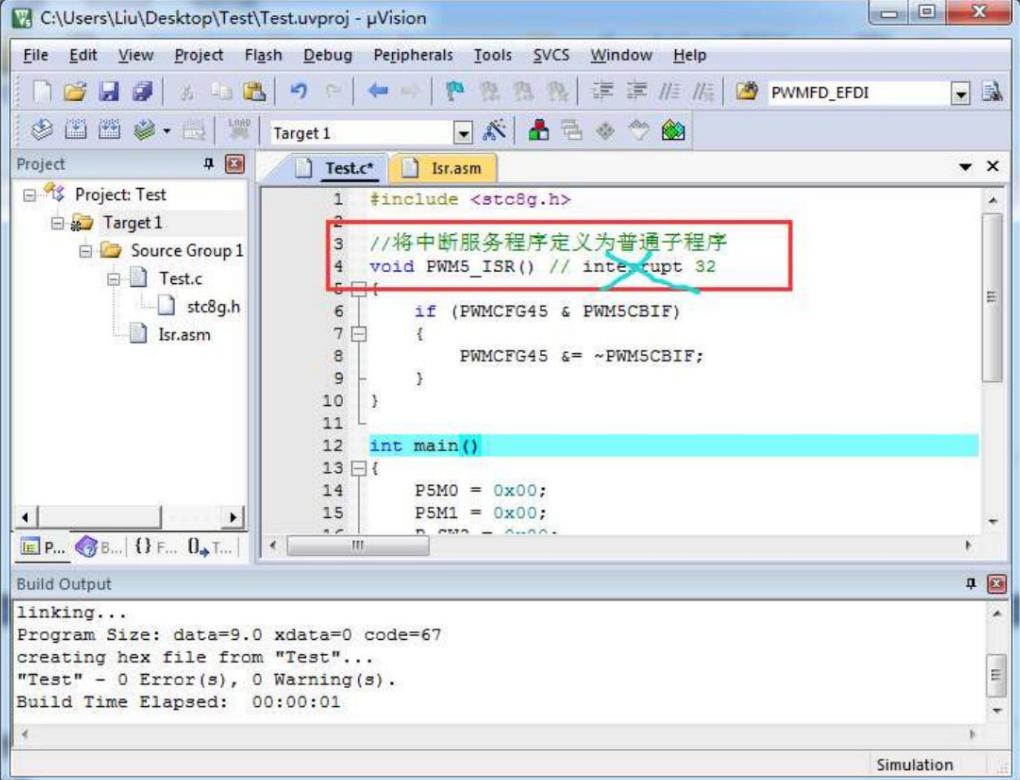
The execution effect is the same as that of method 1. This method is suitable for situations where multiple interrupt numbers greater than 31 need to be remapped.

Method 3: Define the interrupt service routine as a subroutine, and then in the interrupt entry address in the assembly code

Execute the service routine using the **LCALL** instruction

The operation steps are as follows:

1. First, remove the "interrupt" attribute from the interrupt service routine and define it as a common subroutine



The screenshot shows the µVision IDE interface with the project 'Test' open. The assembly file 'Isr.asm' is selected. The code in the editor is:

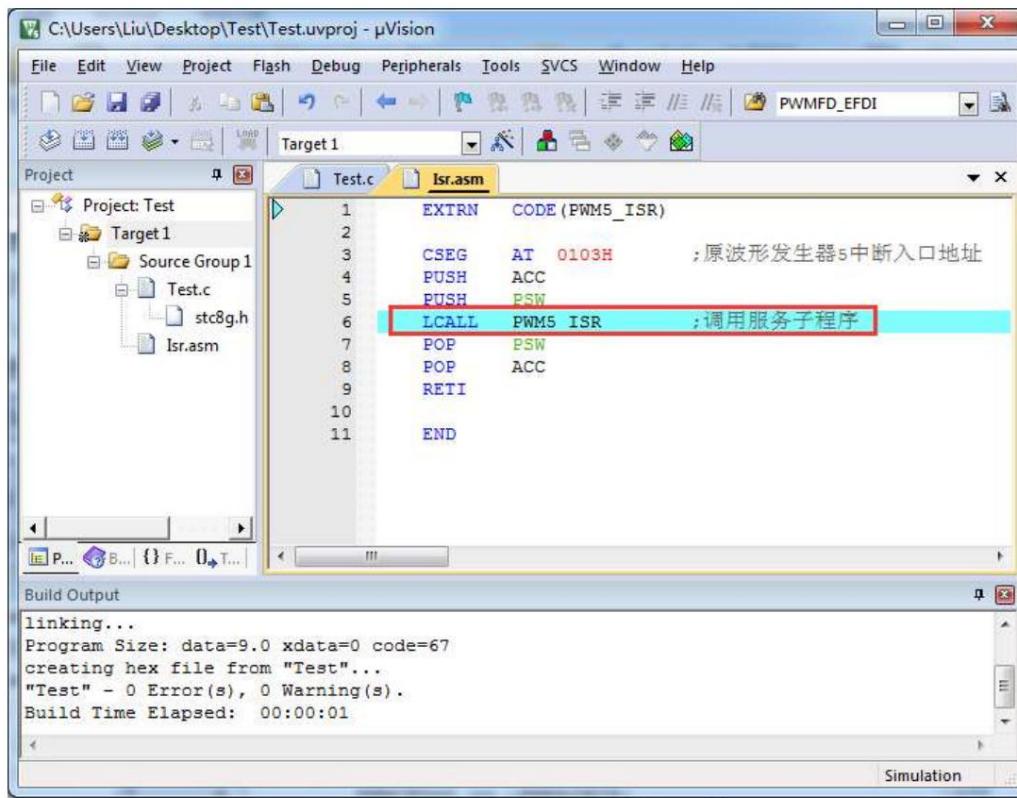
```
1 #include <stc8g.h>
2
3 //将中断服务程序定义为普通子程序
4 void PWM5_ISR() // interrupt 32
5 {
6     if (PWMCFG45 & PWM5CBIF)
7     {
8         PWMCFG45 &= ~PWM5CBIF;
9     }
10 }
11
12 int main()
13 {
14     P5M0 = 0x00;
15     P5M1 = 0x00;
16     P5M2 = 0x00;
```

A red box highlights the line 'void PWM5\_ISR() // interrupt 32'. A green arrow points from the text '将中断服务程序定义为普通子程序' (Define the interrupt service routine as a common subroutine) above the code to this line. The line 'interrupt 32' is crossed out with a blue marker.

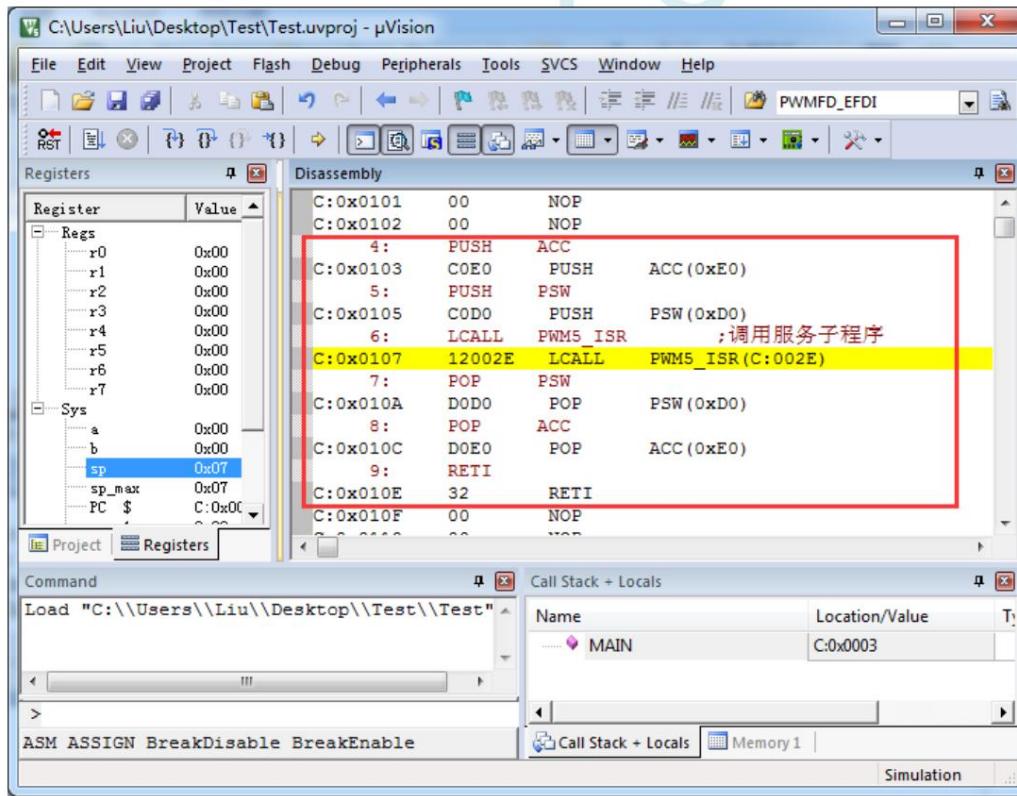
The build output window shows:

```
linking...
Program Size: data=9.0 xdata=0 code=67
creating hex file from "Test"...
"Test" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

2. Then enter the code shown in the following figure in the 0103H address of the assembly file



3. After the compilation is passed, you can find that the interrupt service routine is located at the address of 0103H

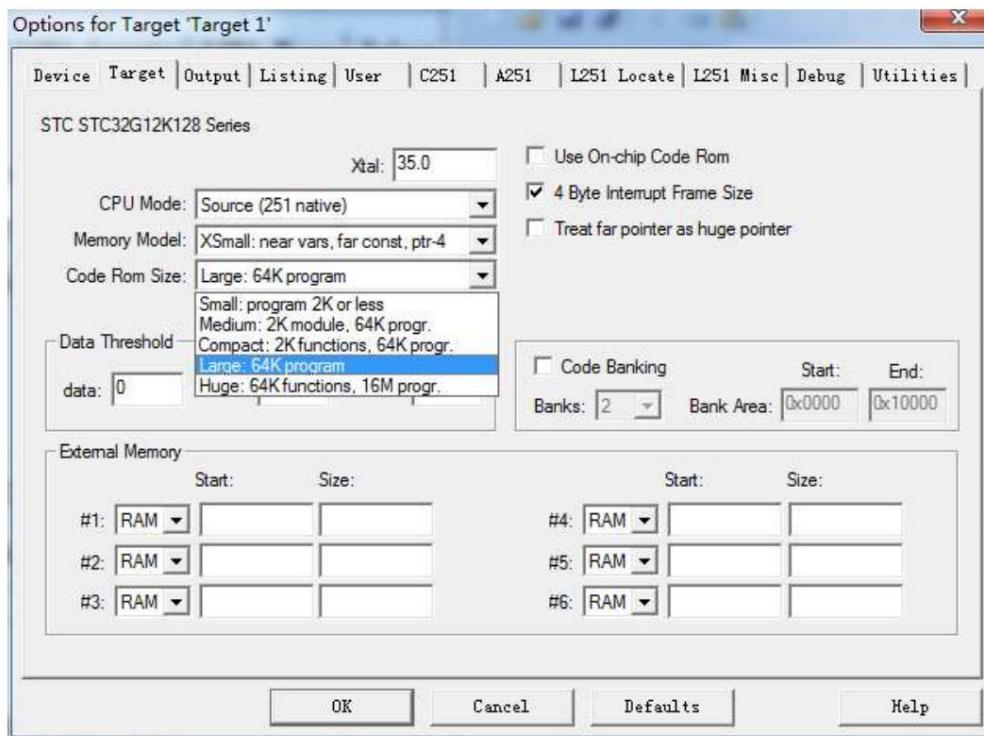


This method does not need to remap the interrupt entry, but there is a problem with this method, which registers need to be pushed into the heap in the assembly file

The stack needs to be determined by the user looking at the disassembly code of the C program. Generally, it includes PSW, ACC, B, DPL, DPH and R0-R7. remove PSW must be pushed to the stack, and which other registers are used in the user subroutine, which registers must be pushed to the stack.

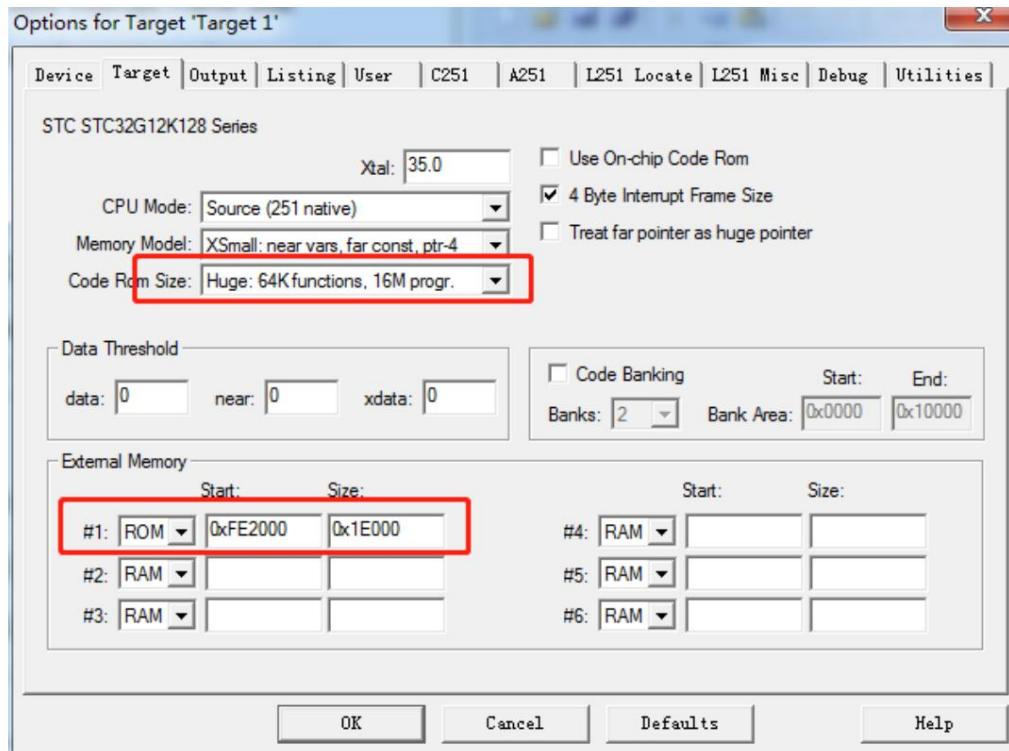
## 5.10 How to set the reserved EEPROM space when the program exceeds 64K

If the user code size is within 64K, you can set the "Code Rom Size" to "Large" mode, and the Keil compiler will automatically link all the codes in the address range of FF:0000~FF:FFFF. 64K of FE:0000~FE:FFFF can be used for user's EEPROM size setting, any use.

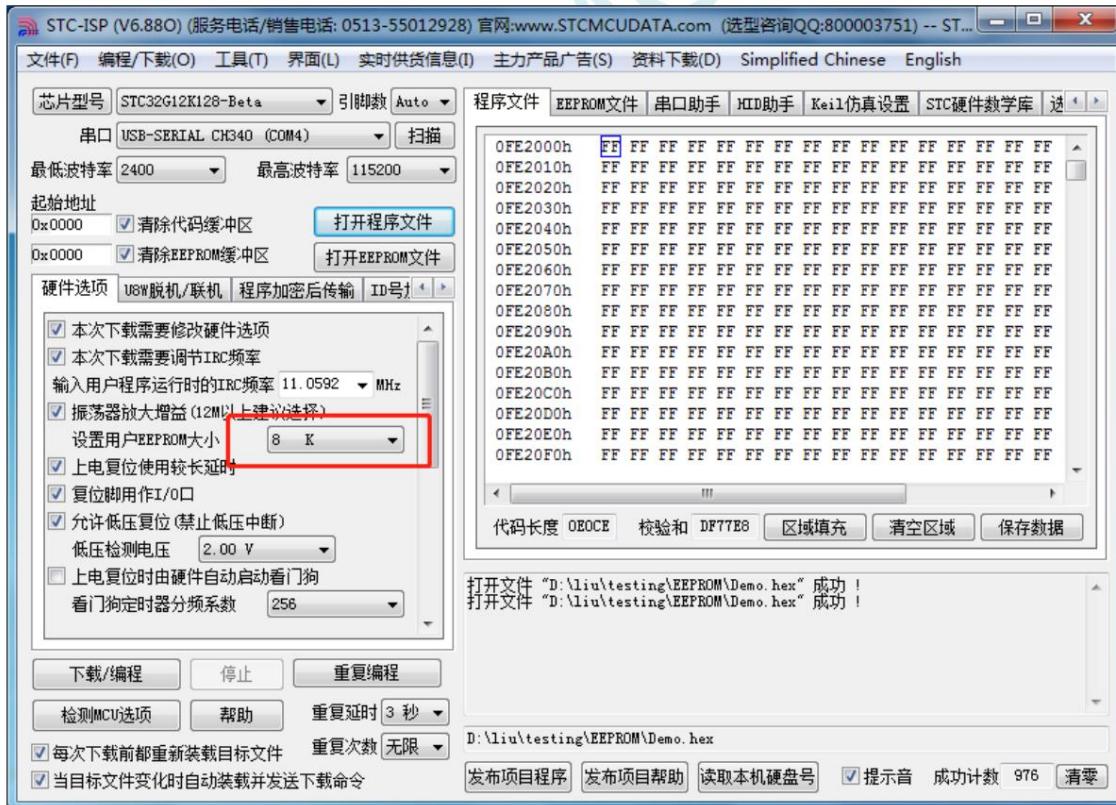


If the user code size exceeds 64K, "Code Rom Size" must be set to "Huge" mode, at this time the Keil compiler is in the chain. When connecting a code block, the reset code and interrupt vector code are first linked to the address starting from FF:0000, and other code blocks will automatically start from the address sample set by the user starts to be stored. Since the address of the EEPROM of STC32G12K128 in FLASH is fixed as FE:0000, so in order for the compiler not to put the user code in the EEPROM area, the following corresponding settings must be made:

For example, the user needs the size of the EEPROM to be 8K, that is, the FE:0000~FE:1FFF area of the FLASH address is the EEPROM area. The FE:2000~FF:FFFF area of the FLASH address is the user code area. Then you need to set the following in Keil:



The following settings are required in the ISP download software:



## 5.11 Use STC-USB Link1 to emulate STC32G series MCU

(Note: other models cannot be simulated)

### 5.11.1 Get to know the STC-USB Link1 tool



Note: You can choose one of the tool's large USB connector and small USB interface to connect to the computer.

### 5.11.2 Hardware connection method

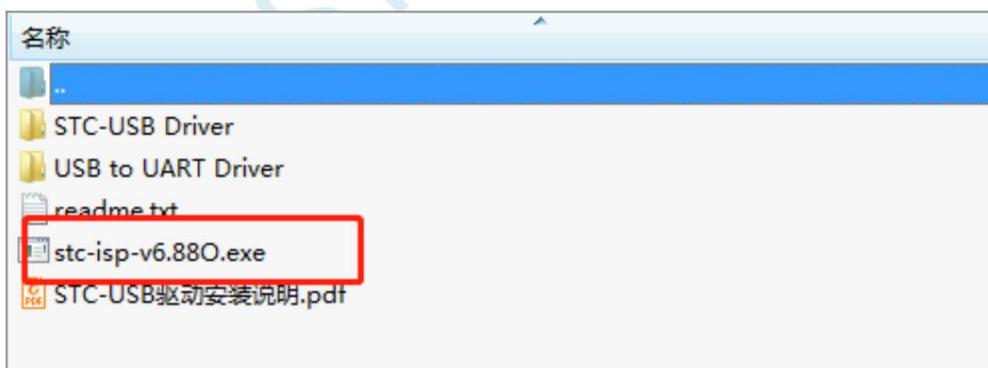


### 5.11.3 Install the emulation driver

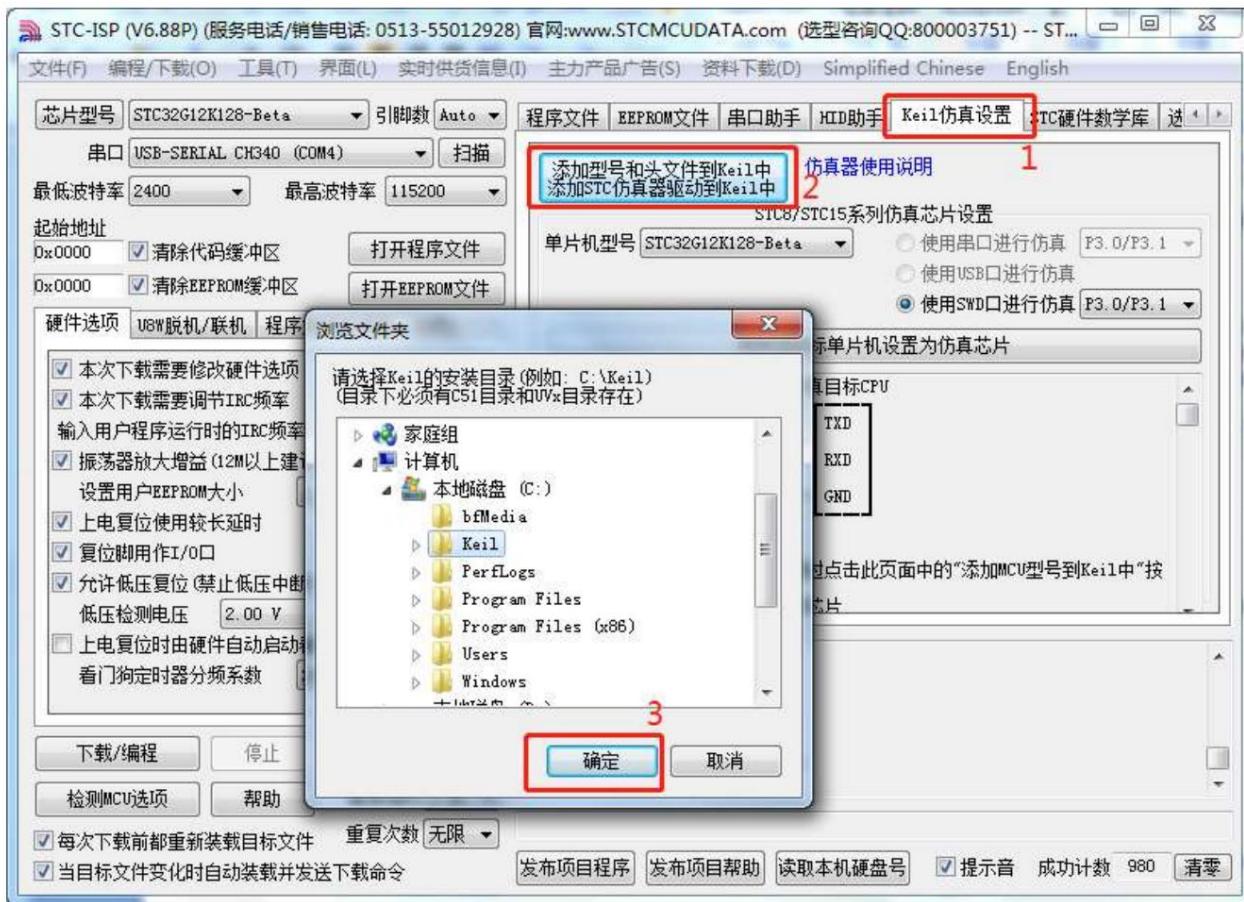
First download the latest STC-ISP download software from STC official website



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

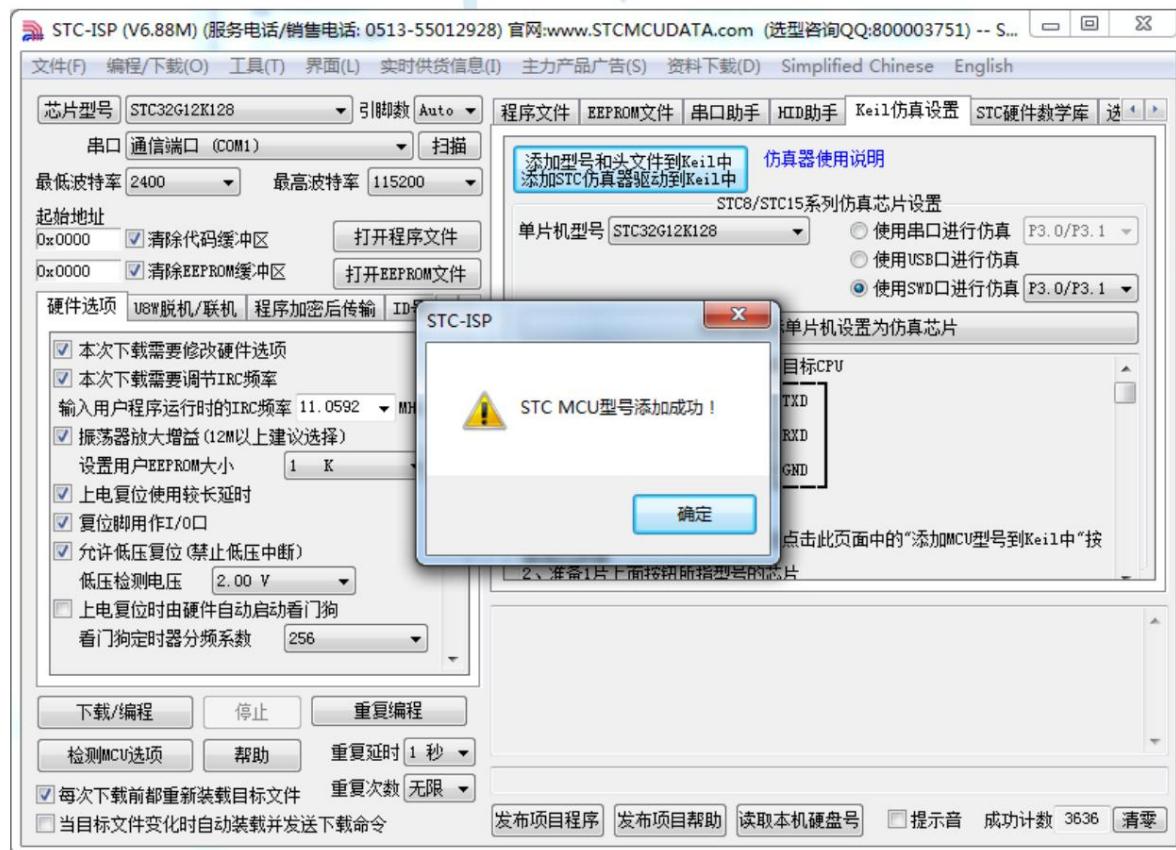


Click the "Add Model and Header File..." button in the "Keil Simulation Settings" page of the download software (Figure "2" below)



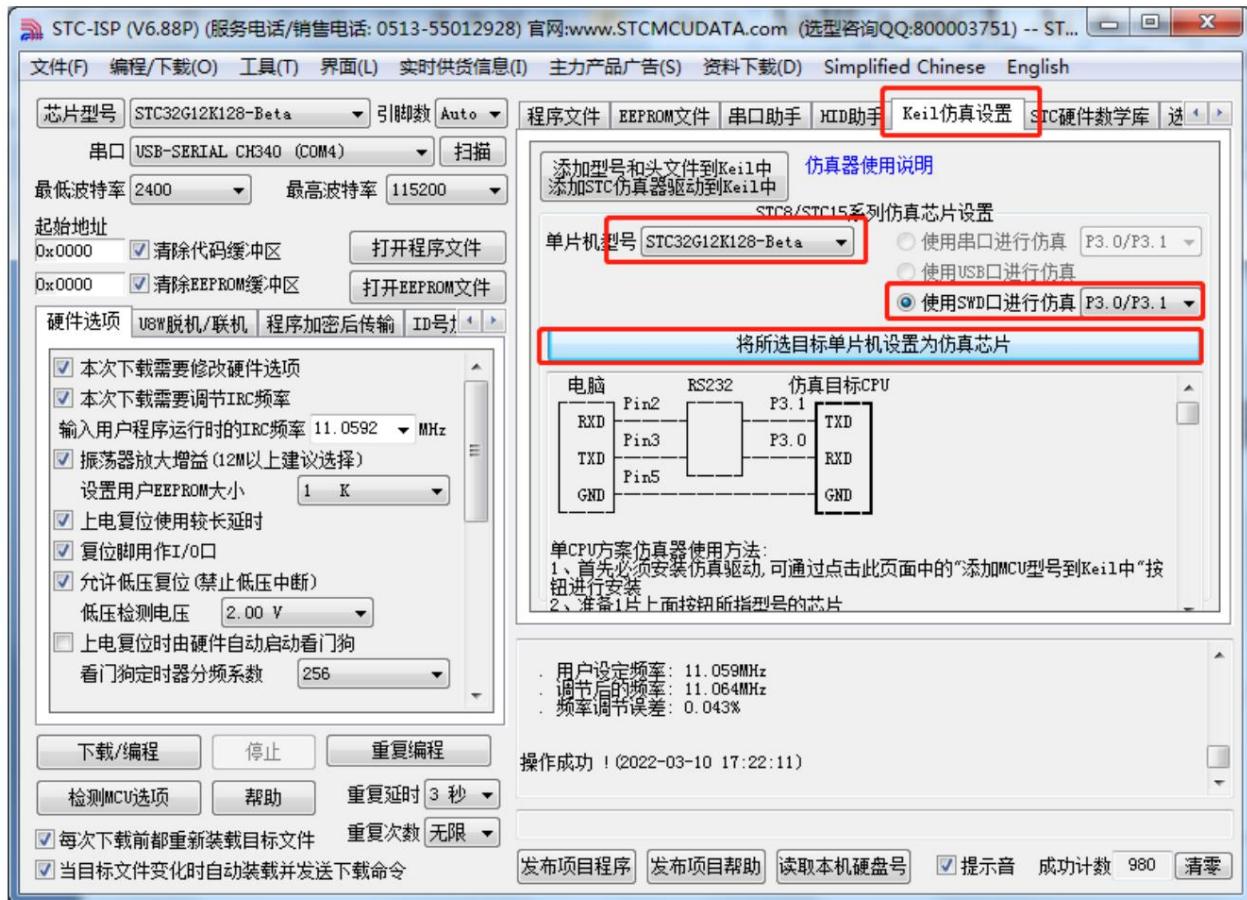
In the pop-up "Browse for Folder" window, select the Keil installation directory (usually Keil's installation directory is "c:\keil"), click

After clicking OK, if "STC MCU model added successfully" pops up, it means that the driver has been installed.



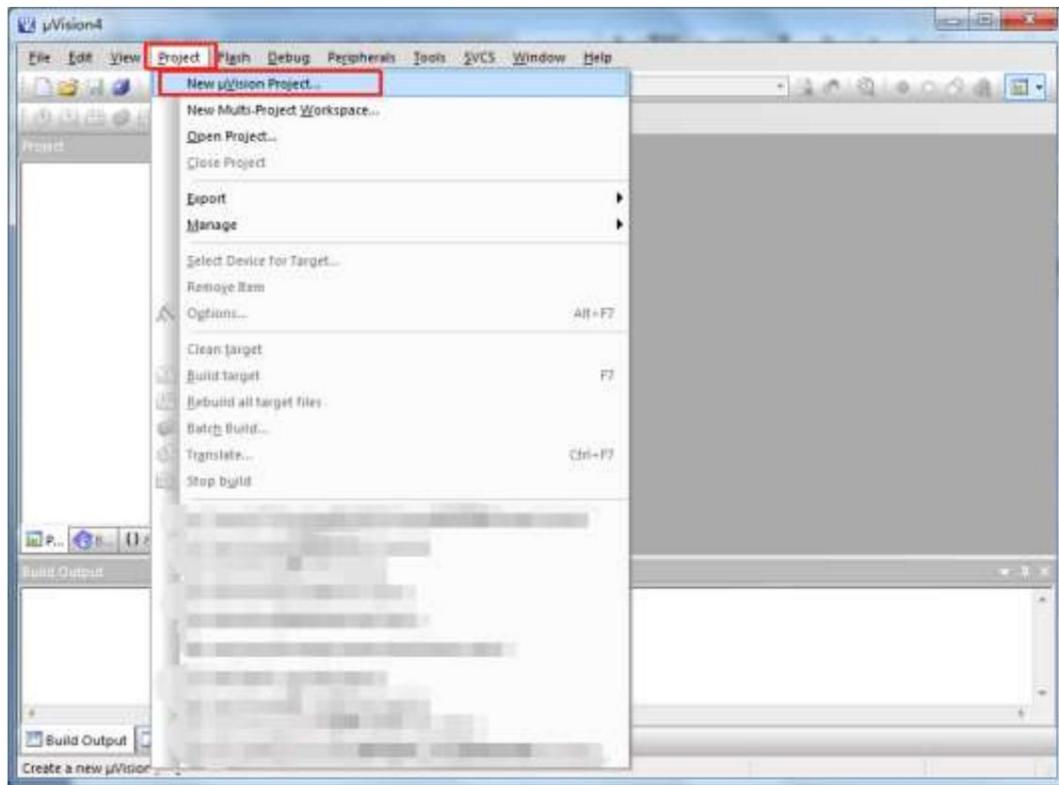
## 5.11.4 Making Emulation Chip

The hardware emulation function is not enabled by default when the chip leaves the factory. To enable the hardware emulation function, you need to use the ISP to download the software. set up.



Select "Use SWD port for emulation", after the download is complete, the chip has emulation function.

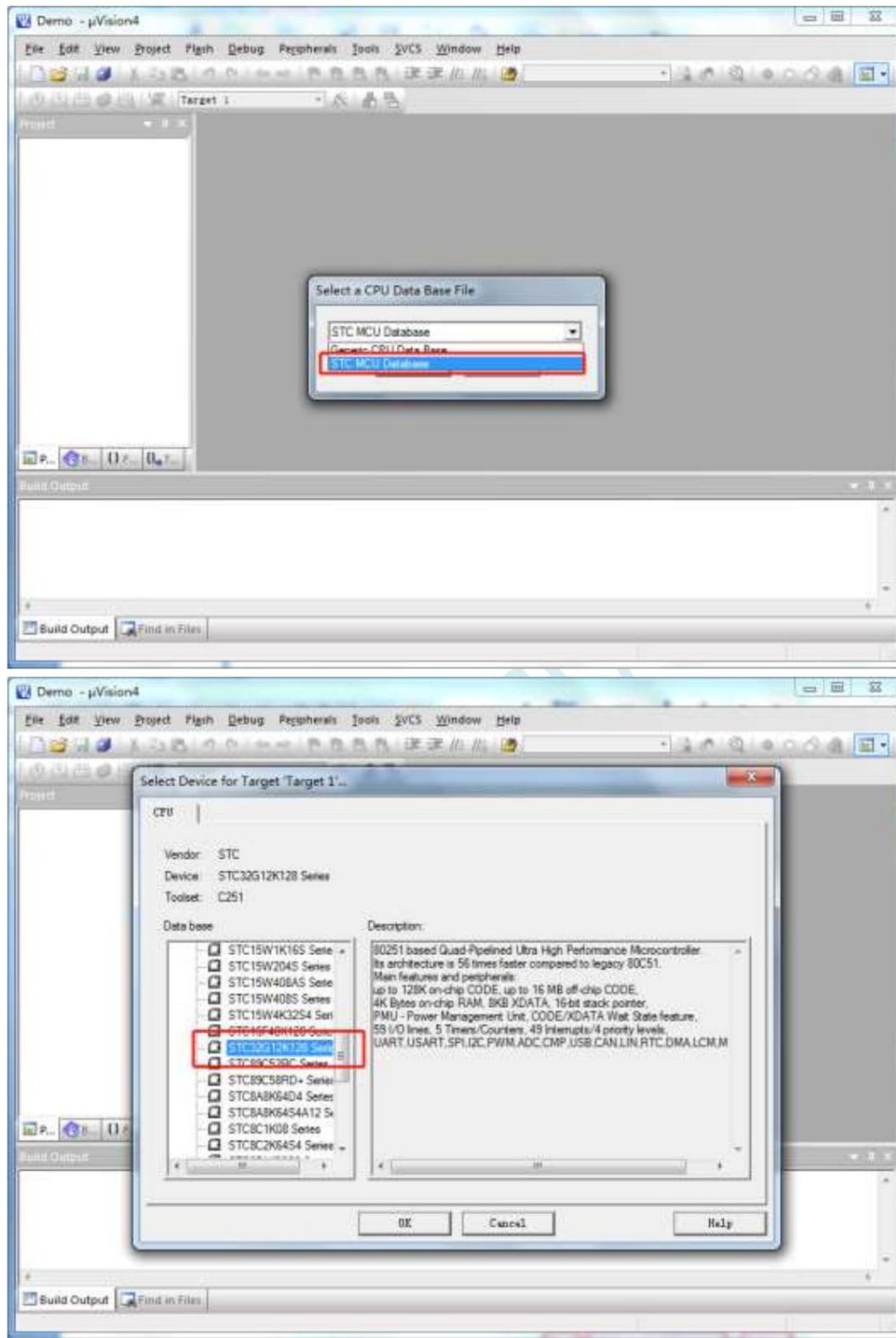
### 5.11.5 Create and set up a project in Keil software



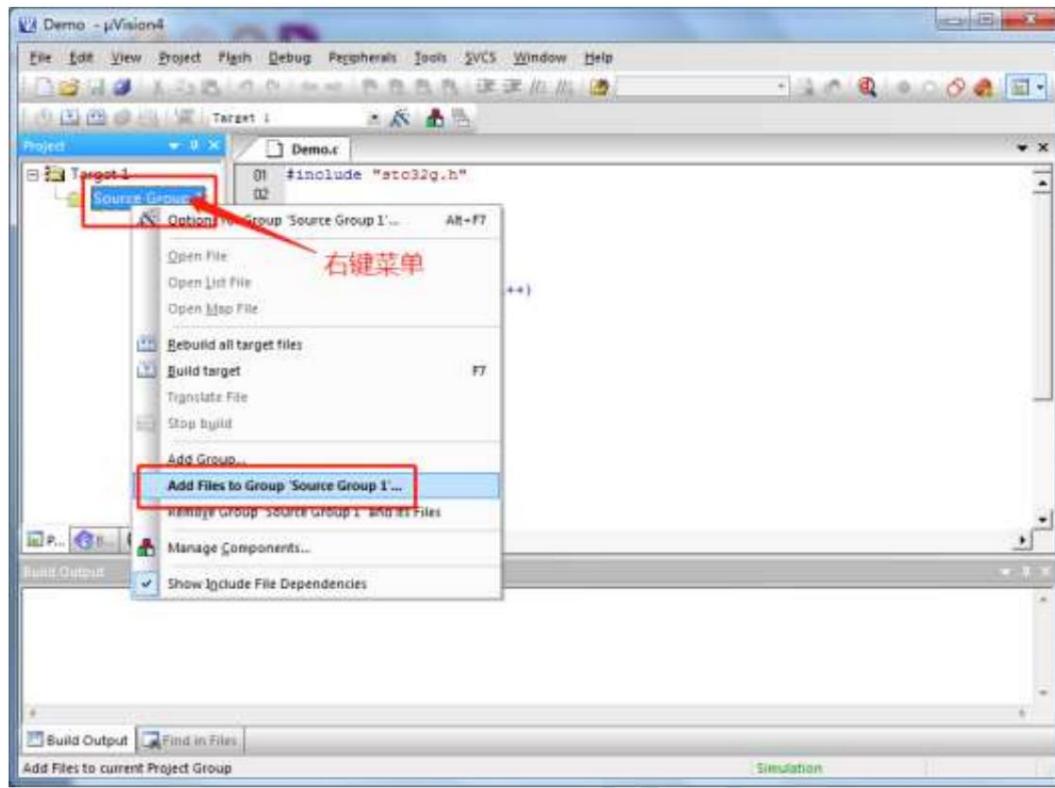
Specify the project path and enter the project name



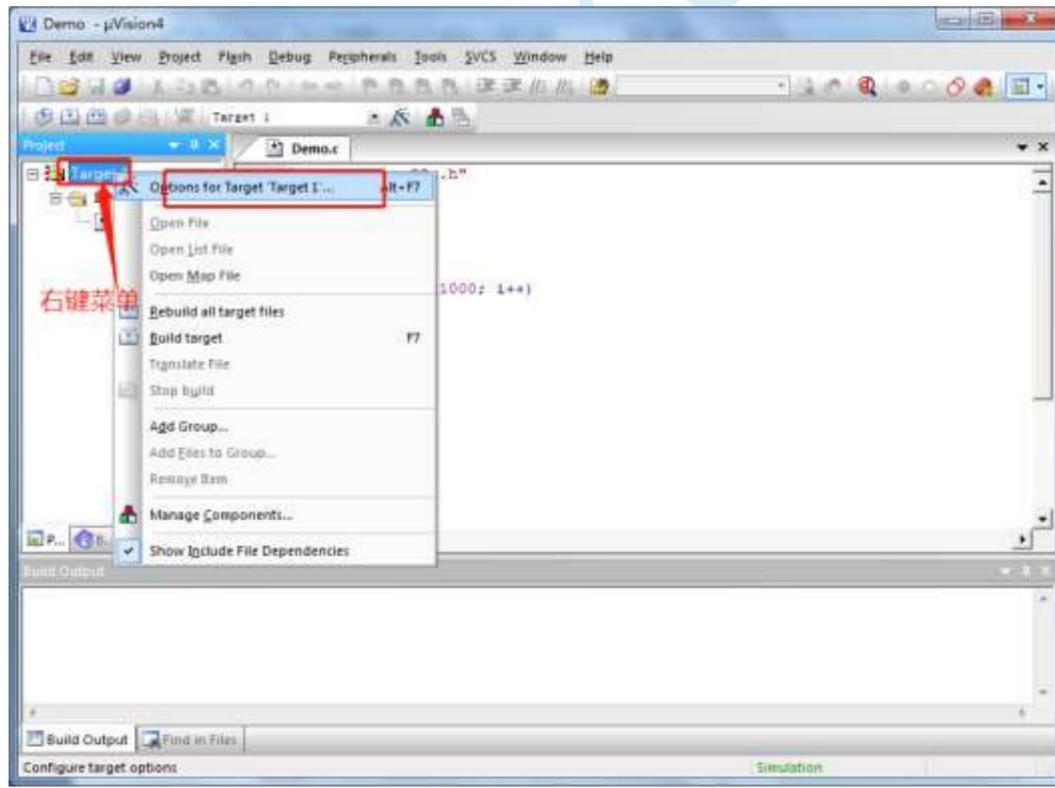
Select the target chip model: STC32G12K128

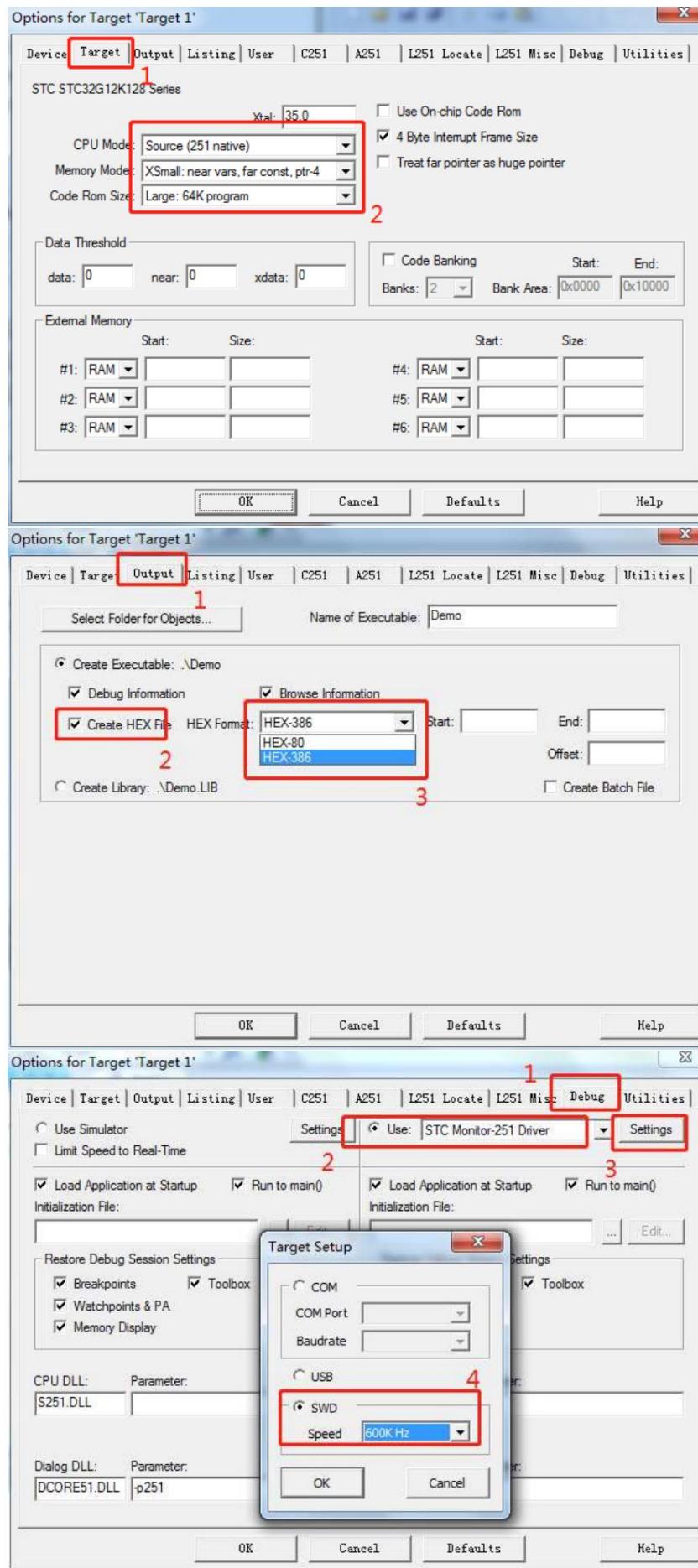


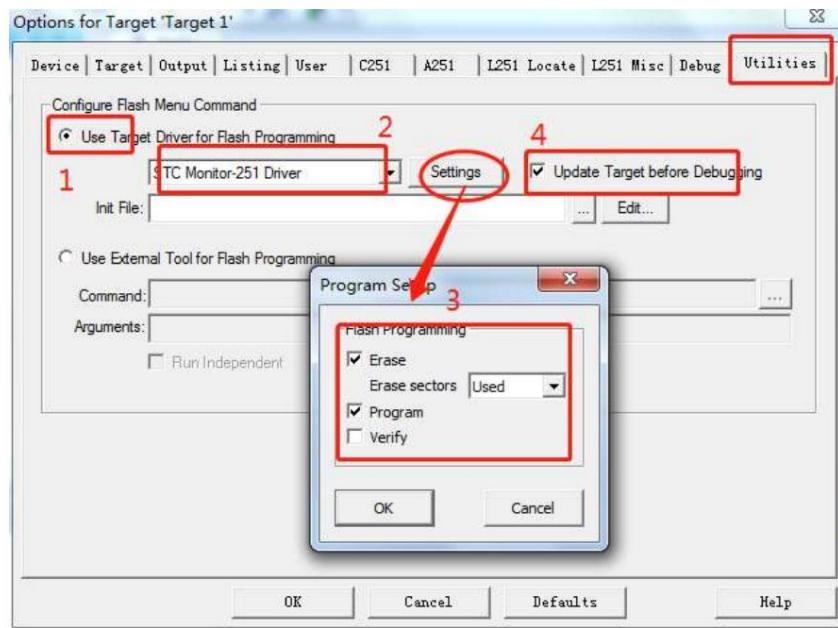
Create code files and add to the project



Project settings

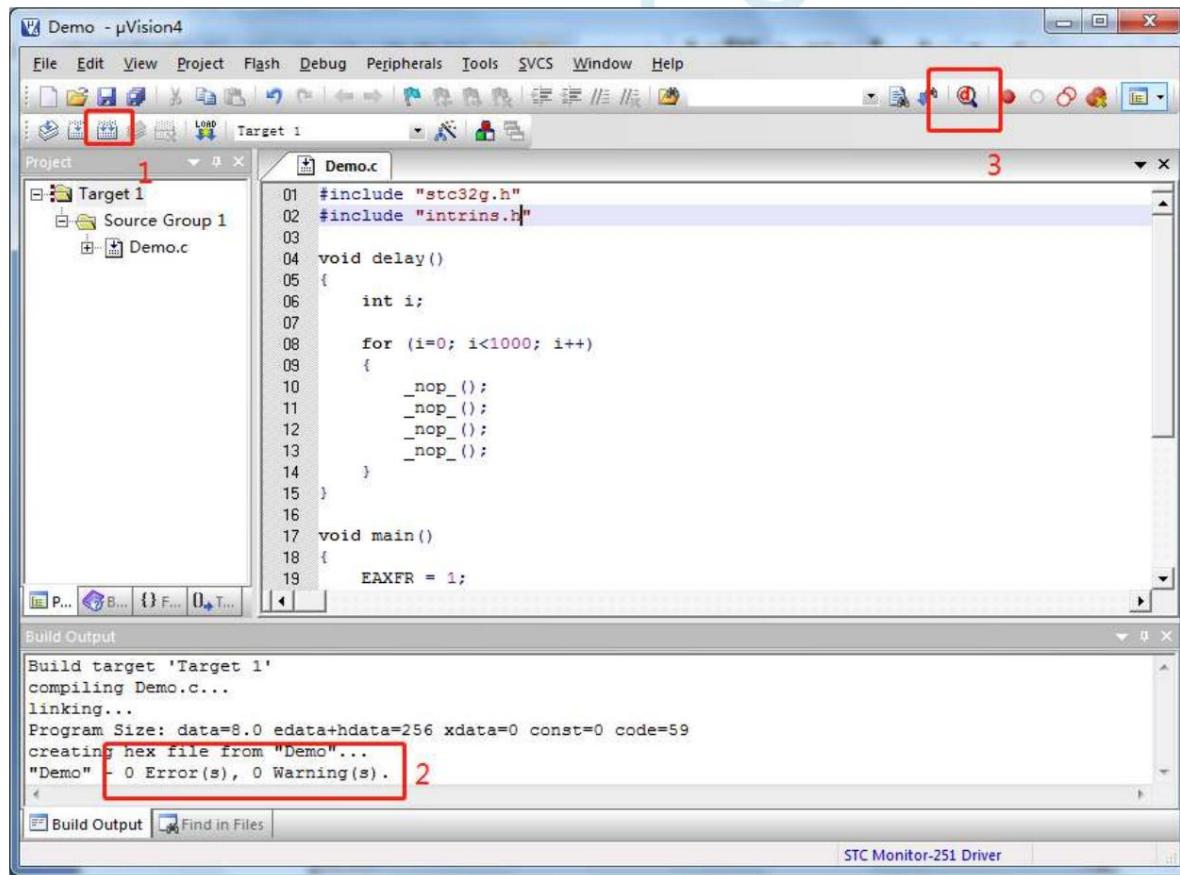


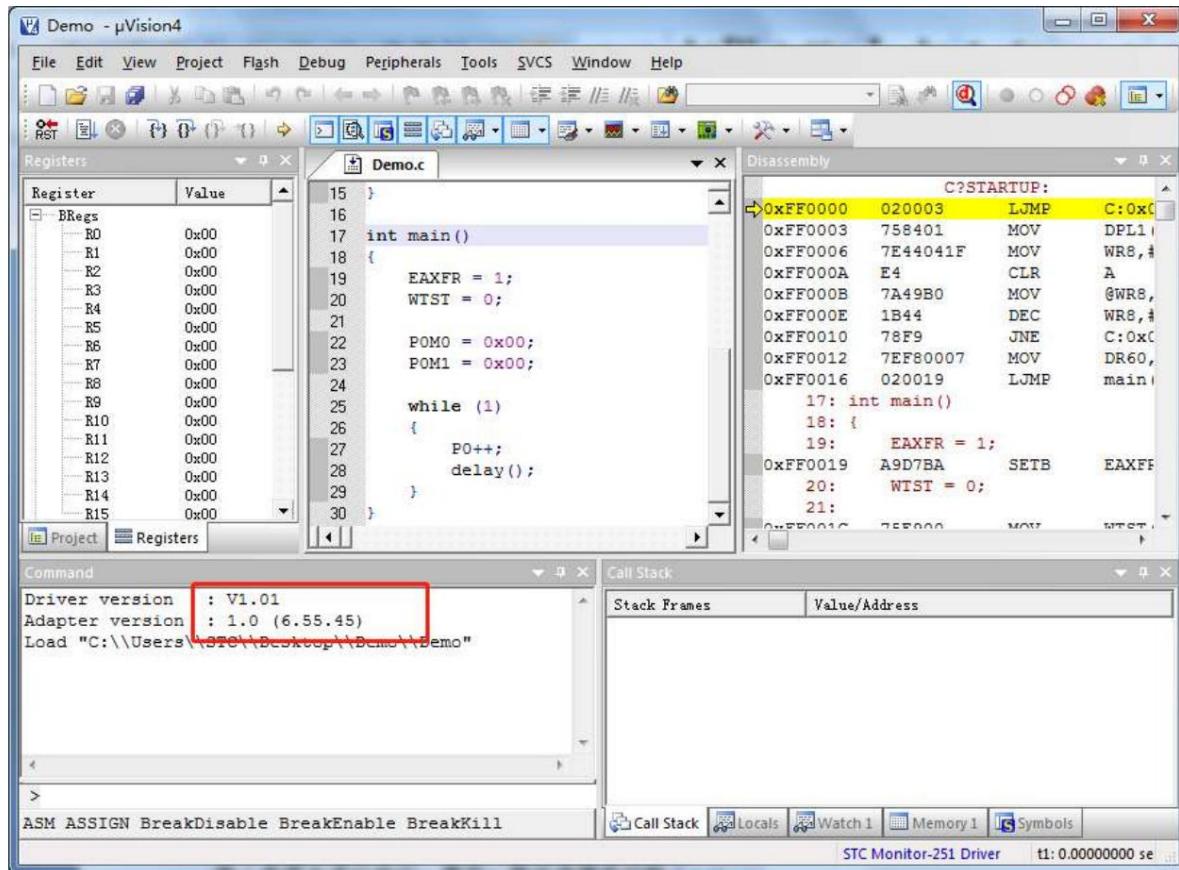




### 5.11.6 Compile, download and simulate

In the Keil environment, after editing the source code and compiling without errors, you can start the simulation





If the chip fabrication and connection are correct, the simulation driver version will be displayed as shown in the figure above, and the user code can be downloaded to the microcontroller correctly.

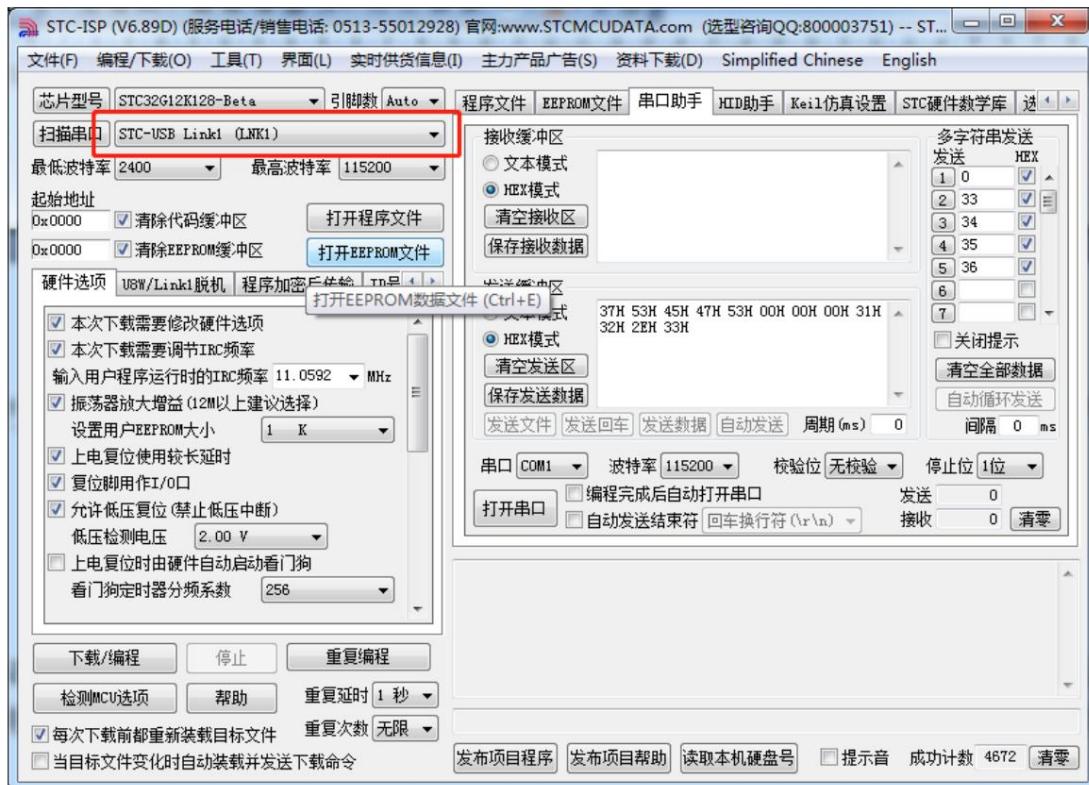
Then you can perform debugging functions such as running, single stepping, and breakpoints.

## 5.12 Precautions for using STC-USB Link1 tool

### 5.12.1 Correct identification of tools

When the STC-USB Link1 tool is shipped from the factory, the control program of the STC-USB Link1 has been programmed in the main control chip. Normally, work

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



After correct identification, STC-USB Link1 can be used for online ISP download or offline ISP download.

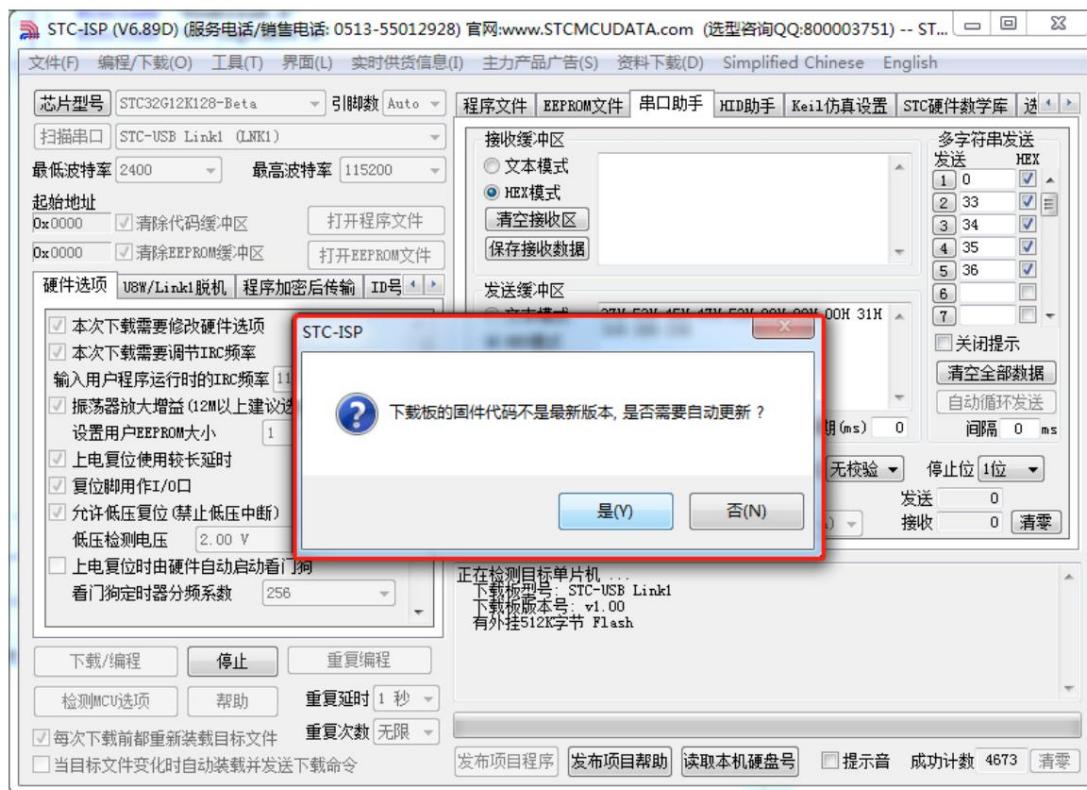
If the tool is connected to the computer, "STC-USB Link1 (LNK1)" cannot be recognized, but it is always recognized as "STC USB Writer" (HID1), please confirm whether the toggle switch on the tool as shown in the figure below is set to the "burning and emulation" position



After turning the switch to the burning and emulation position, reconnect the tool to the computer to correctly identify the "STC-USB Link1 (LNK1)"

### 5.12.2 Automatic upgrade of tool firmware

When using the tool to download ISP, the software will pop up the following screen, indicating that the firmware of the tool needs to be upgraded



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

### 5.12.3 Enter the method of updating the firmware

Please do not power off the tool during the automatic upgrade of the tool. If there is an abnormal power failure during the upgrade process, or other reasons cause the main control chip

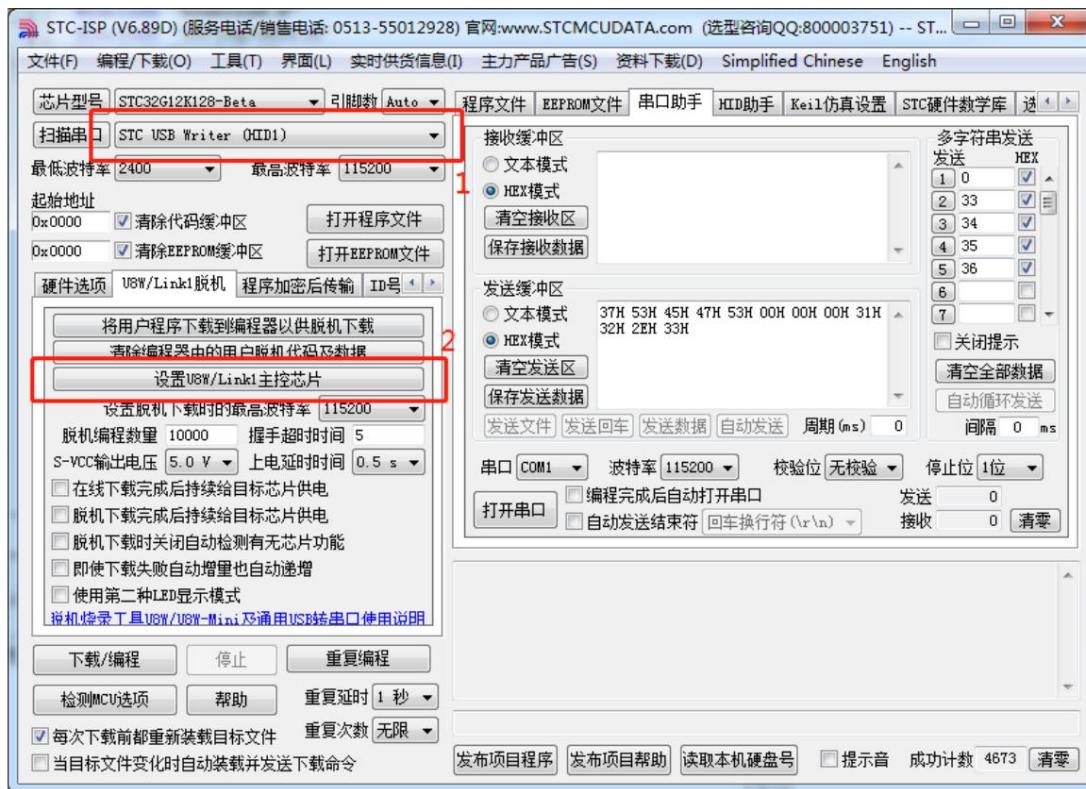
If the control program is lost, the downloaded software cannot correctly identify "STC-USB Link1 (LNK1)", you need to manually program the control program.

First, turn the toggle switch to "Update Tool Firmware" (as shown below)



After the STC-ISP download software recognizes the "STC USB Writer (HID1)", click "Set U8W/Link1" as shown in the figure below

Main Control Chip\* button.

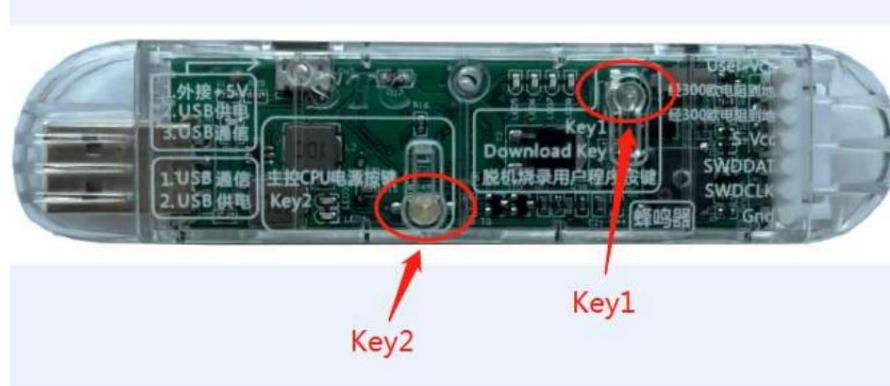


After programming, be sure to turn the toggle switch back to the "burning and emulation" position

#### 5.12.4 Enter method 2 to update firmware

If there is no toggle switch on the tool of the subsequent version, enter the method of "Update Tool Firmware": first use the USB cable to connect the tool to the computer.

Connect, then first press and hold Key1 on the tool and do not release, then press Key2, wait for the STC-ISP download software to recognize "STC USB Writer (HID1)" and then release Key1.

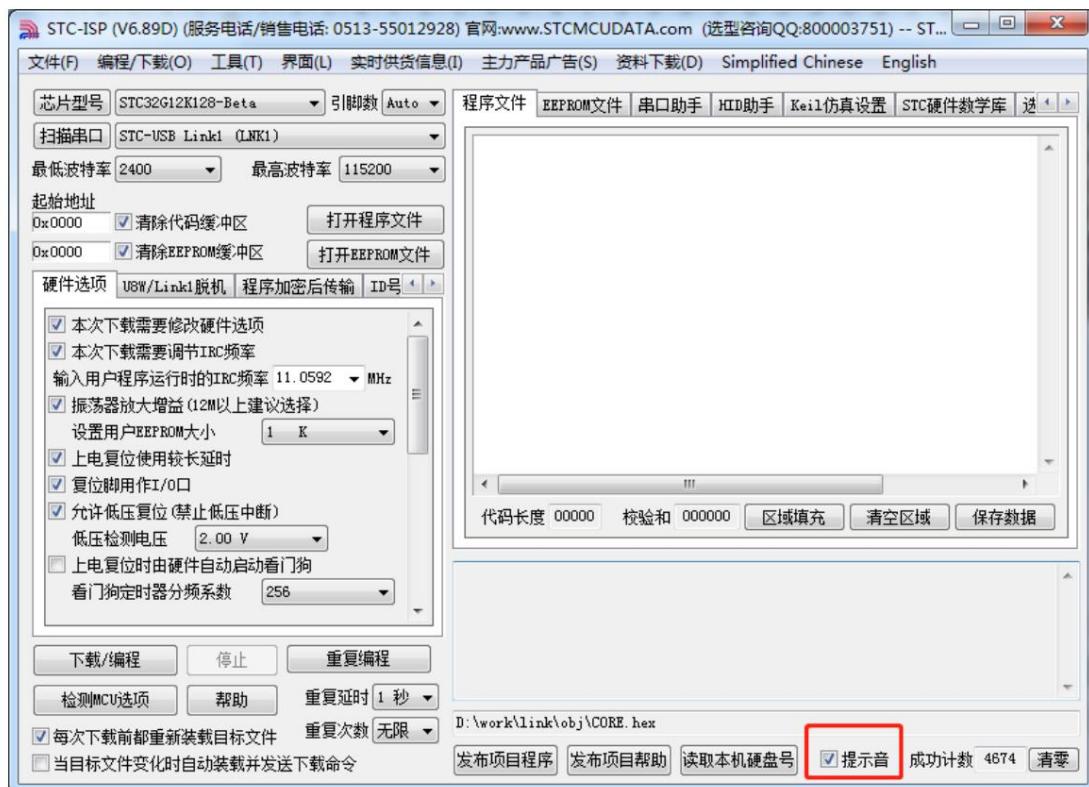


## 5.12.5 STC-USB Link1 working indicator description

STC-USB Link1 working indicator mode is as follows:

1. During the ISP online or offline download process, 4 LEDs are displayed in the form of running water 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, 4

All LEDs are off at the same time, and the buzzer emits a short beep and a long beep. (The switch of the prompt tone is shown in the figure below in the ISP download software set up in the place shown)



## 5.13 Precautions for using STC-USB Link1D tool

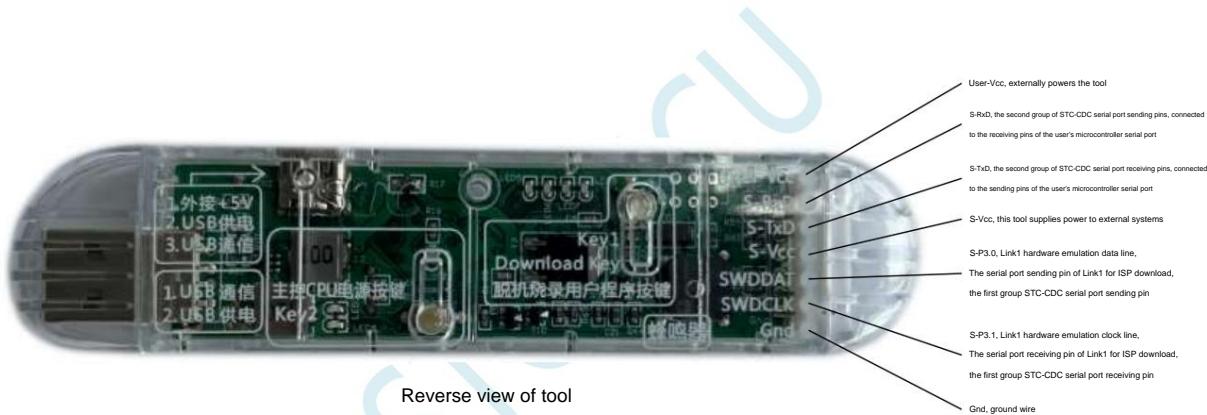
### 5.13.1 Tool interface description

STC-USB Link1D tool is an upgraded version of STC-USB Link1, with two functions added on the basis of STC-USB Link1

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



Tool front view



Pin No.	Interface	Name	User-Vcc	Interface function
1			Only the tool is powered by the user system	
2		S-RxD	Group 2 STC-CDC serial port sending pin, connected to the receiving pin of the user's microcontroller serial port	
3		S-TxD	Group 2 STC-CDC serial port receiving pin, connected to the user's microcontroller serial port sending pin	
4		S-Vcc	only powers user system from this tool	
5	S-P3.0			The serial port sending pin when using Link1 for ISP download, connect to P3.0 of the target microcontroller The data pin when using Link1 for SWD hardware emulation, connect to the SWDDAT of the target microcontroller The first group of STC-CDC serial port sending pins, connected to the user's microcontroller serial port receiving pins
6	S-P3.1			The serial port receiving pin when using Link1 for ISP download, connect to P3.1 of the target microcontroller The clock pin when using Link1 for SWD hardware emulation, connect to the SWDCLK of the target microcontroller The receiving pins of the first group of STC-CDC serial ports are connected to the sending pins of the serial port of the user's microcontroller
7		Gnd	ground wire	

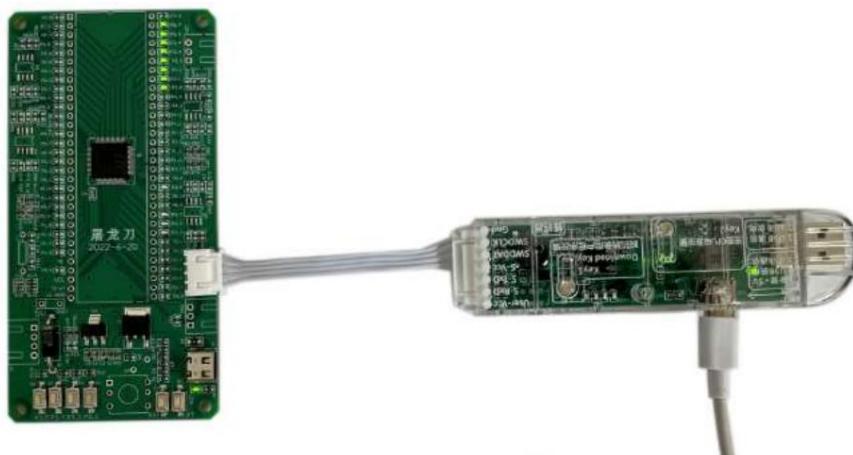
## 5.13.2 Practical application of STC-USB Link1D

1. Use the STC-USB Link1D tool to perform SWD hardware simulation on the STC32G series MCU

Connect S-Vcc, S-P3.0, S-P3.1 and GND of the tool to M-Vcc, S-P3.1 and GND of the target microcontroller as shown in the figure below.

Connect P3.0 (SWDDAT), P3.1 (SWDCLK), and GND, and then refer to the hardware emulation steps in the previous chapter.

Steps and settings to perform SWD hardware emulation



2. Use the STC-USB Link1D tool to emulate the serial port of the STC15 and STC8 series

The S-Vcc, S-P3.0, S-P3.1 and GND of the tool are respectively connected with the M-Vcc, P3.0 (RXD), P3.1 (TXD),

Connect to GND, then select the serial port number corresponding to STC-CDC1 in Keil simulation settings, and then refer to STC15/STC8

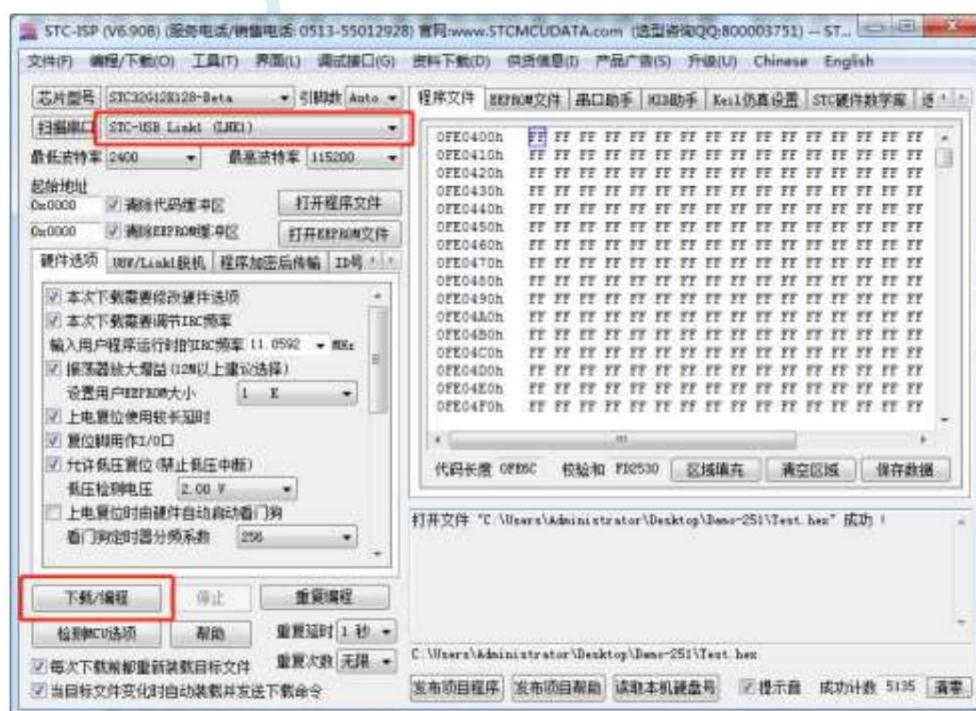
The emulation steps and settings in the direct serial port emulation chapter in the series data sheet can be used for serial port emulation.

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 respectively connected with the M-Vcc, P3.0 (RXD), P3.1 (TXD),

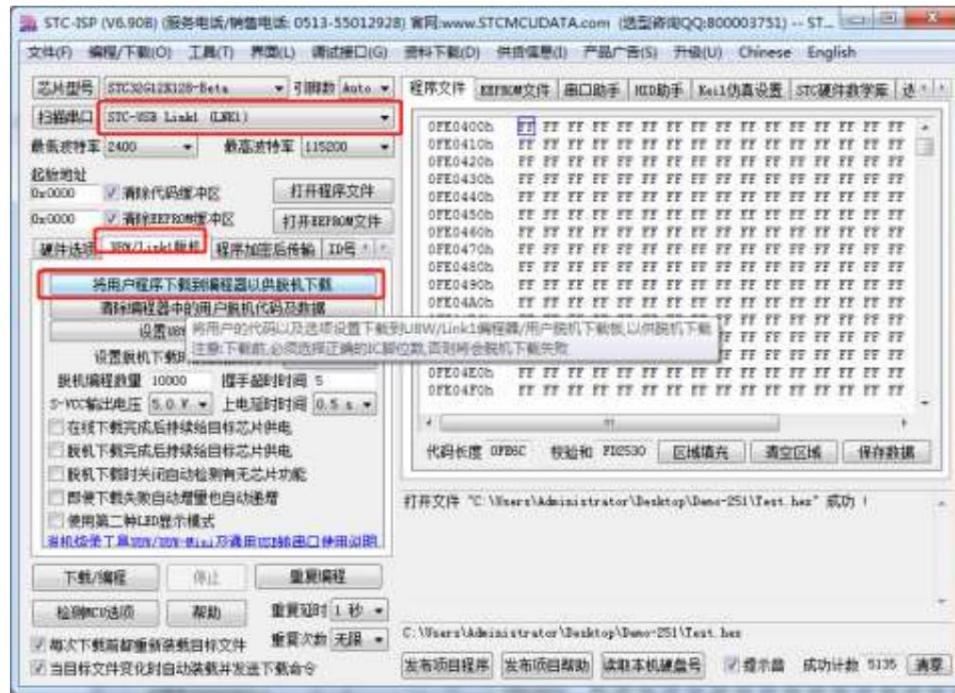
GND is connected, select "STC-USB Link1 (LNK1)" in the serial port number in the STC-ISP download software, open the program

file and set relevant hardware options, then click the "Download/Program" button to download the ISP online



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

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



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

(TxD) and GND, and then press the "Key1" button on the tool to download the target chip offline (ie  
No need for PC control, independent ISP download)

5. The STC-USB Link1D tool is used as a general-purpose USB serial port tool

The STC-USB Link1D tool provides two STC-CDC serial ports, which can be used as a general-purpose USB-specific serial port tool.

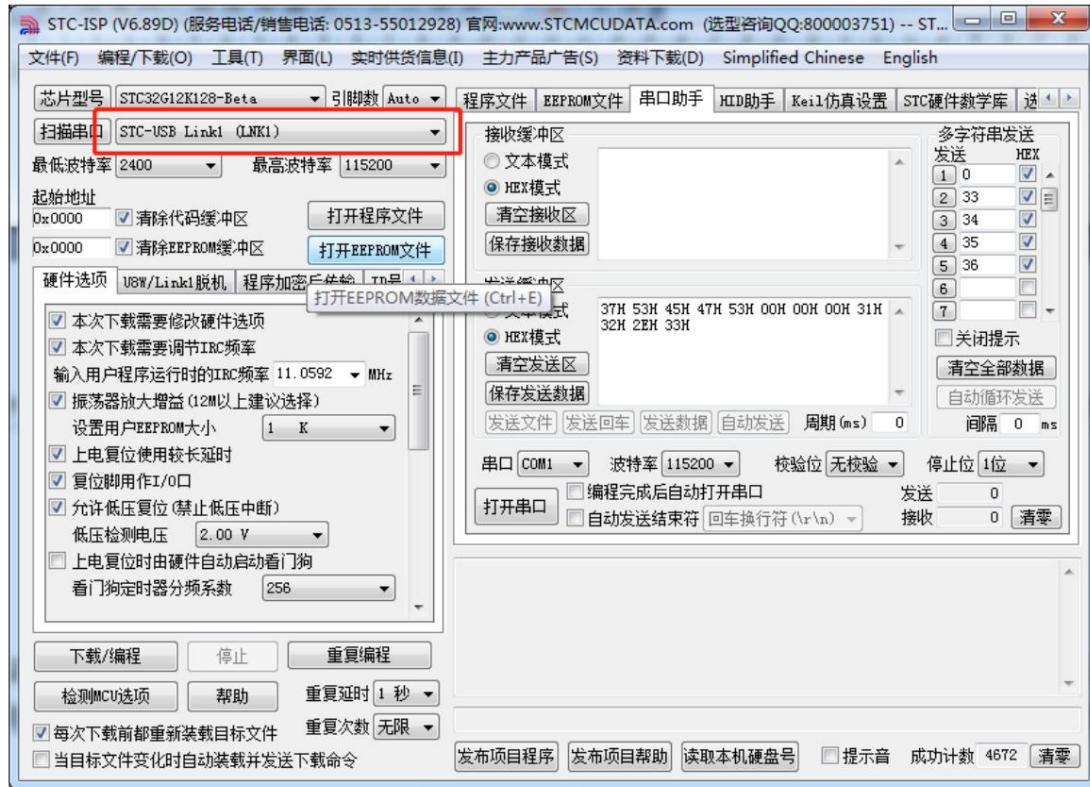
The first serial port CDC1 shares S-P3.0 and S-P3.1 ports with hardware emulation and ISP download, while the second serial port CDC2 is an independent serial port, so it is recommended to use S-P3.0 and S-P3.1 as emulation and ISP download.

When using the serial port tool, use the CDC2 corresponding to S-TxD and S-RxD. (Note: In the absence of usage conflicts, CDC1 and CDC2 can be used independently as a general-purpose USB serial port tool)

### 5.13.3 Correct identification of tools

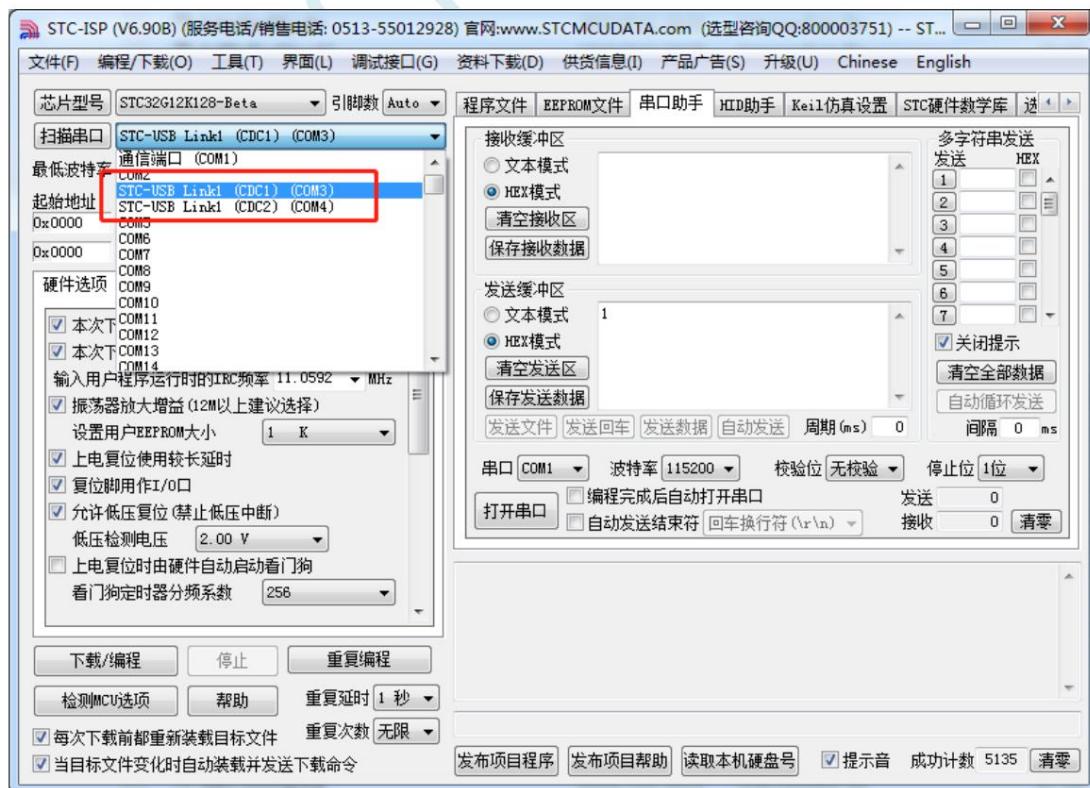
When the STC-USB Link1 tool is shipped from the factory, the control program of the STC-USB Link1 has been programmed in the main control chip. Normally, work

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



After correct identification, STC-USB Link1 can be used for online ISP download or offline ISP download.

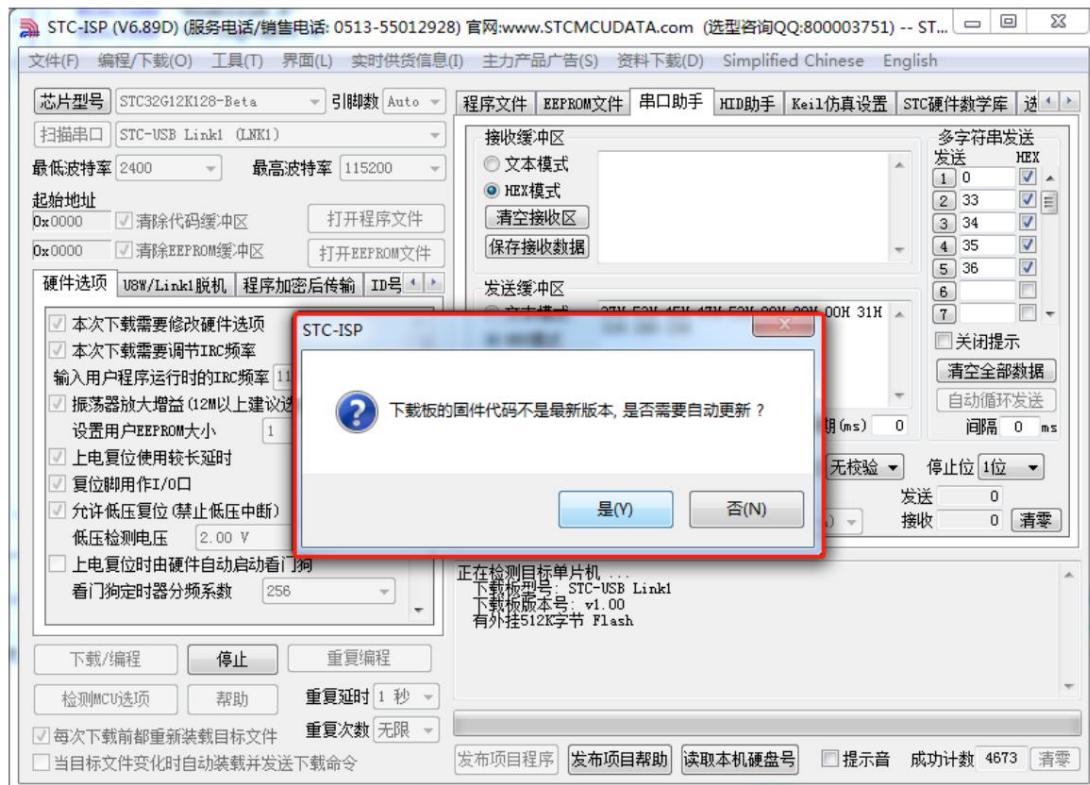
After the driver is installed successfully, two STC-CDC serial ports will be automatically recognized, as shown in the following figure:



Can be used as a general-purpose USB-to-serial tool.

#### 5.13.4 Automatic upgrade of tool firmware

When using the tool to download ISP, the software will pop up the following screen, indicating that the firmware of the tool needs to be upgraded

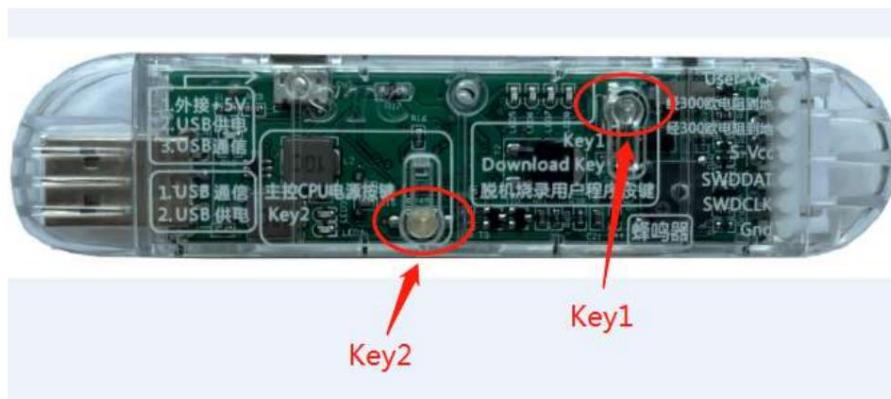


Click the "Yes" button and the tool will automatically start the upgrade.

#### 5.13.5 Enter the method to update the firmware

First connect the tool to the computer with a USB cable, then first press and hold Key1 on the tool and do not release it, then press Key2, etc.

Release Key1 after the STC-ISP download software recognizes "STC USB Writer (HID1)".



## 5.13.6 STC-USB Link1D driver installation steps

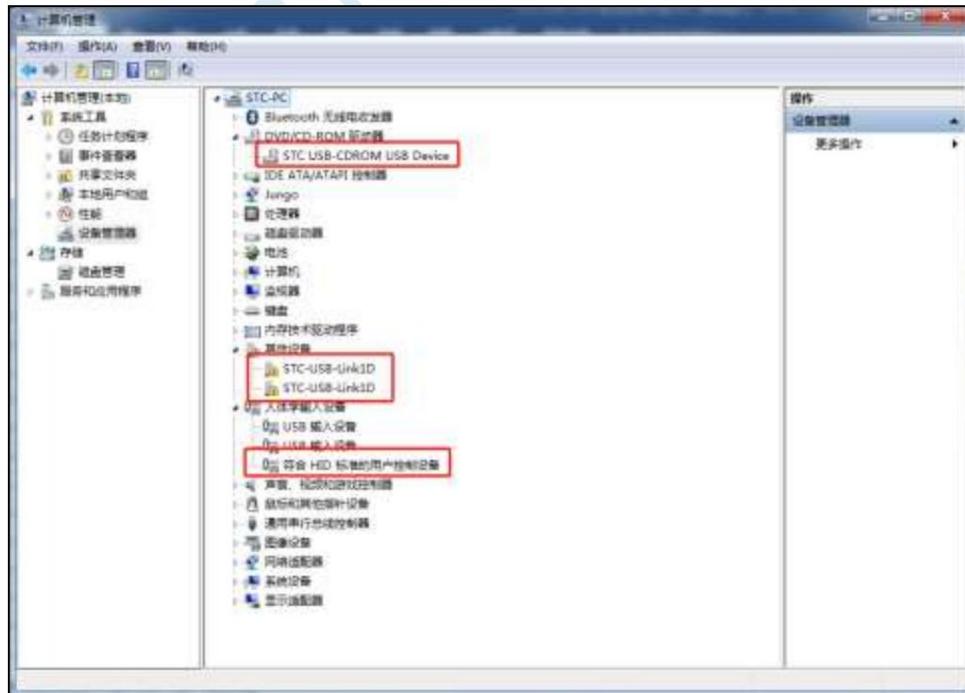
1. Insert the STC-USB Link1D tool into the USB port of the computer, the computer will display the following screen



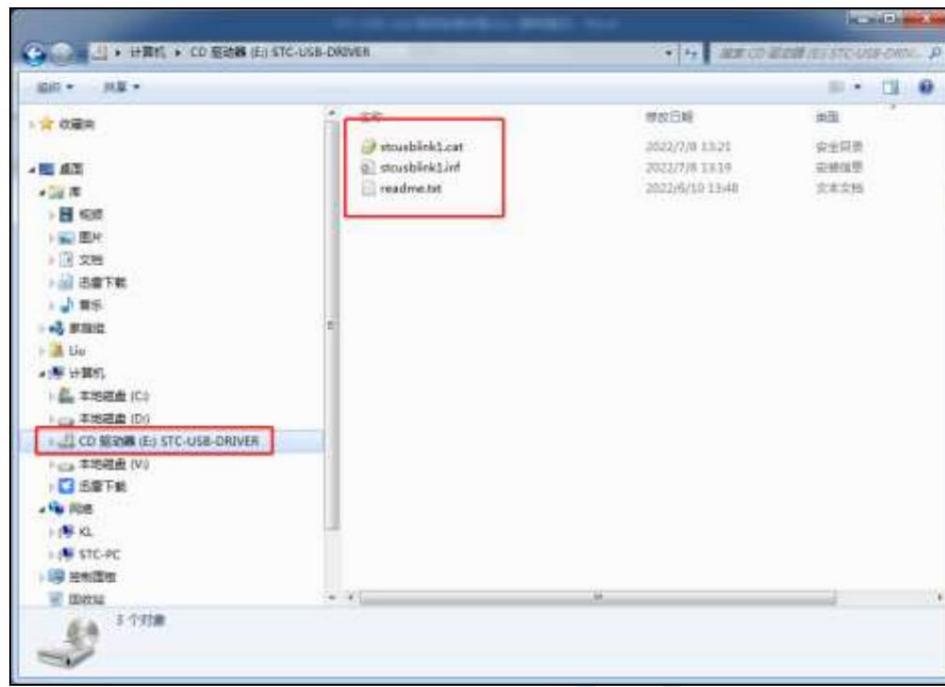
2. After the automatic installation of the driver is completed, in the device manager of the computer, it shows that the HID connector in the STC-USB Link1D device has been automatically recognized.

port and USB optical drive interface, but the two CDC virtual serial ports will have a yellow exclamation mark, indicating that the virtual serial port driver has not been installed successfully.

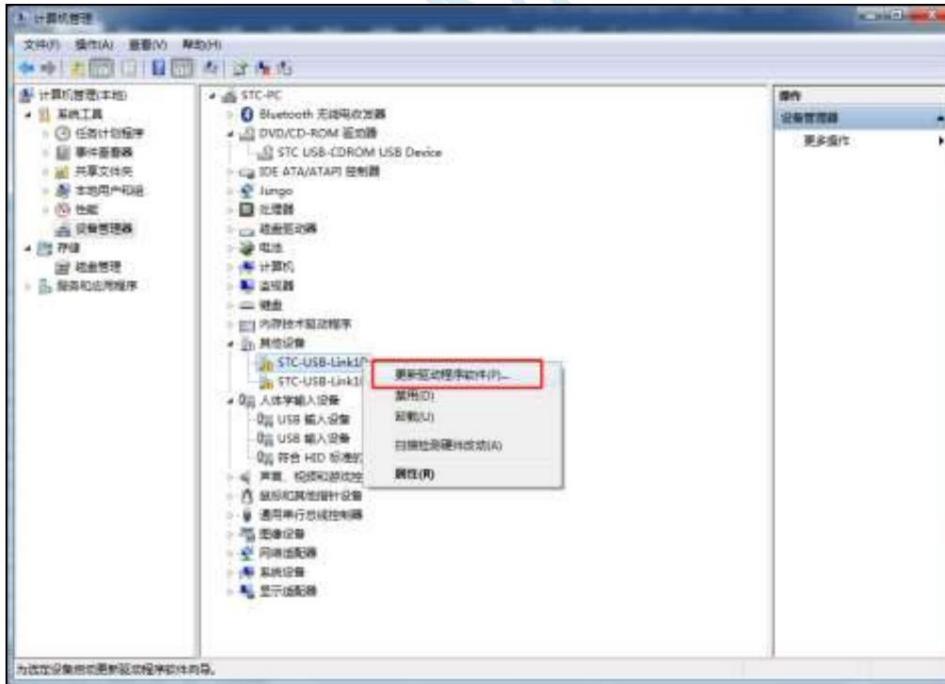
To install manually. As shown below



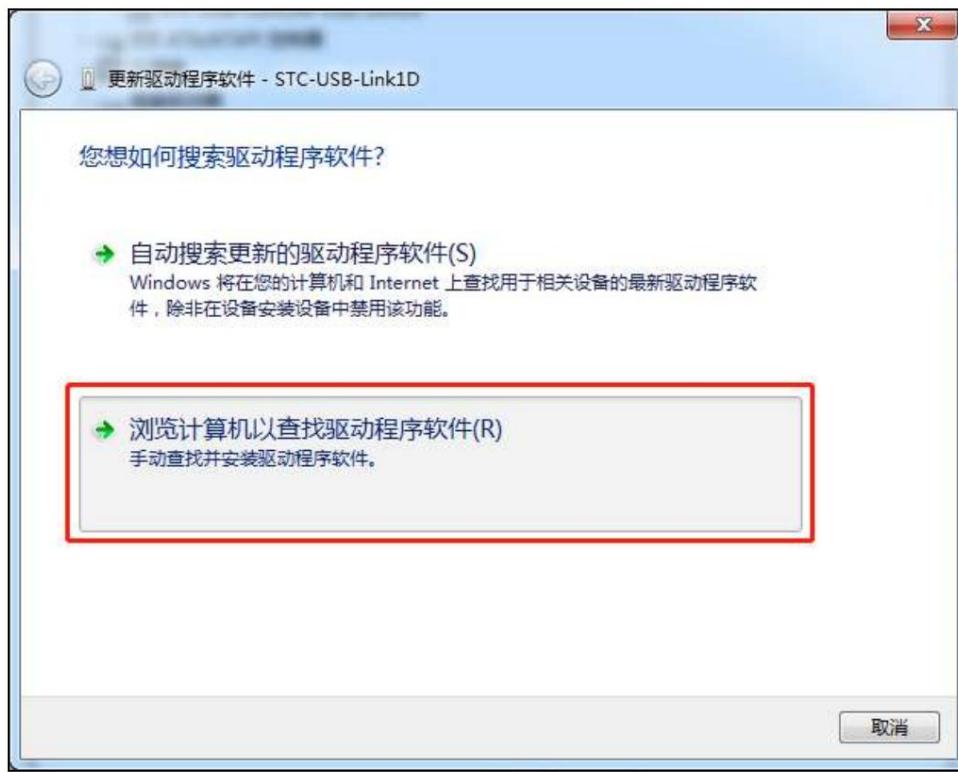
In the Windows Explorer, open the automatically recognized USB CD-ROM, which has a virtual serial port driver.



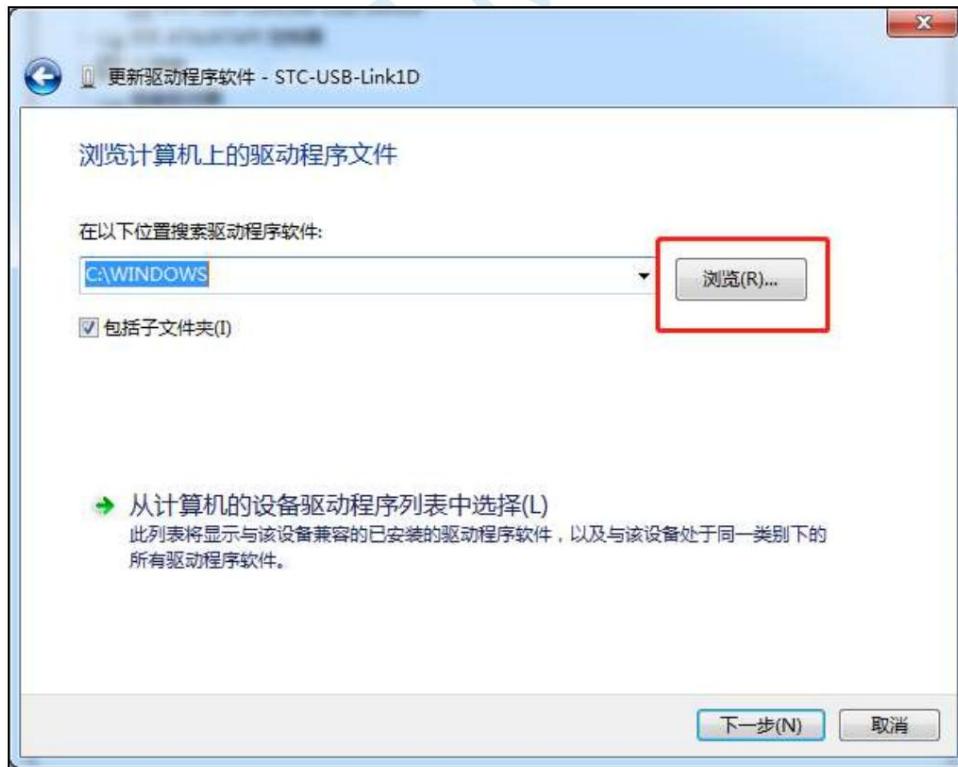
3. The steps to manually install the virtual serial port driver are as follows: First, find the first "STC-USB Link1D" with a yellow exclamation mark in the device manager, right-click the mouse, and select "Update Driver Software (P)" in the right-click menu. ..."



4. In the pop-up "Update Driver Software" window, click "Browse my computer for driver software"



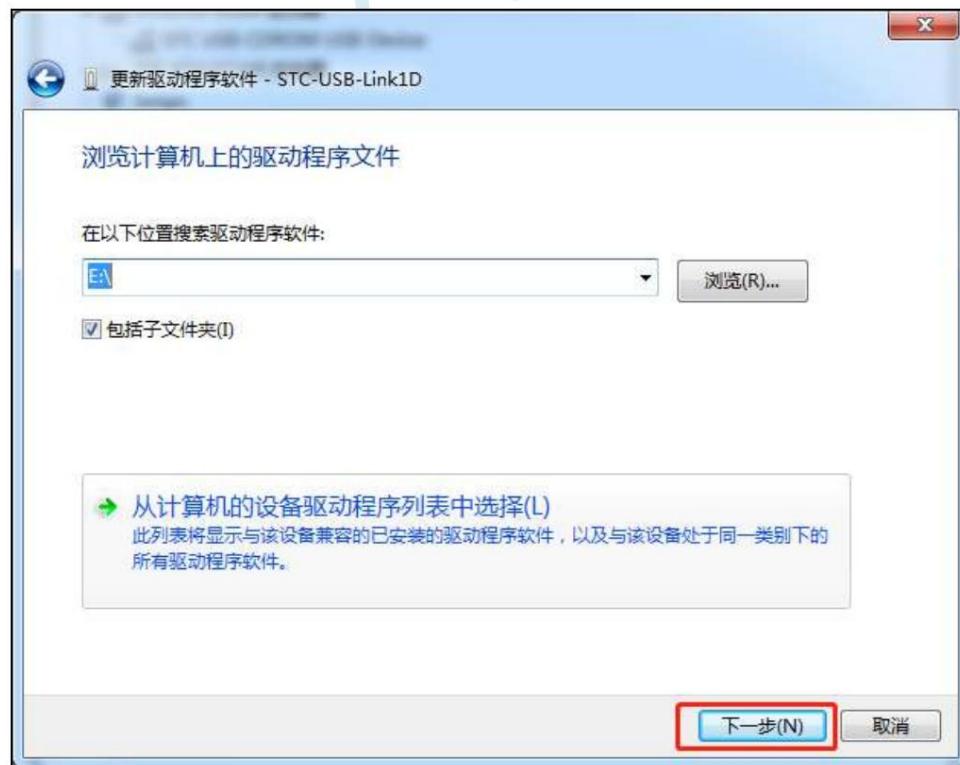
5. Click the "Browse" button in the following screen



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



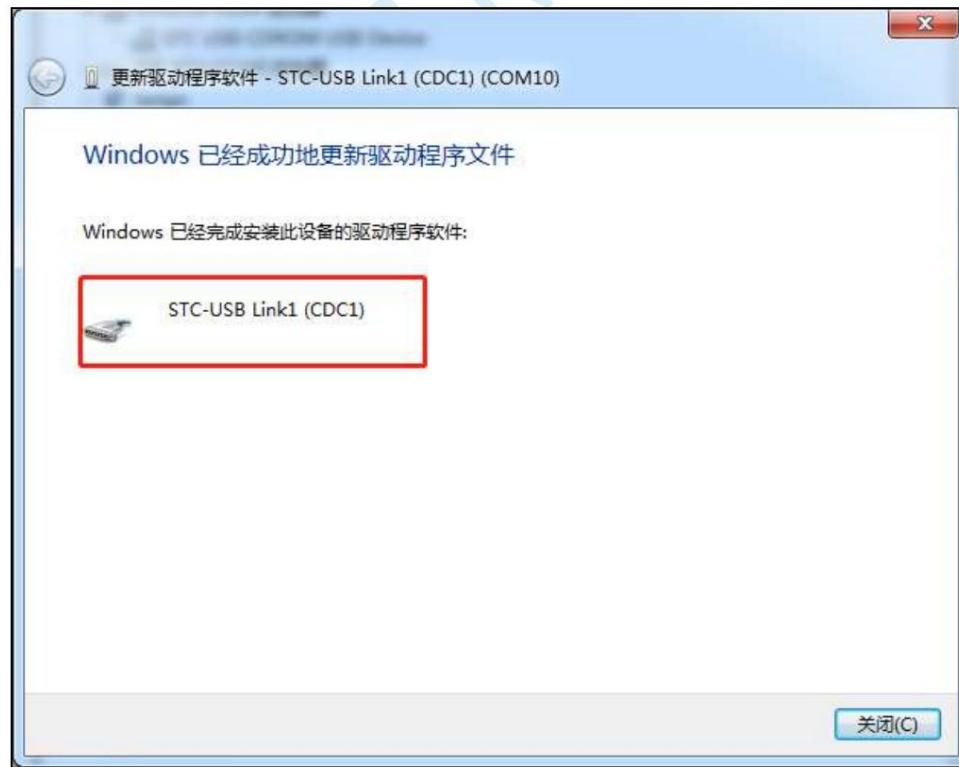
7. As shown in the figure below, click the "Next" button to start installing the driver



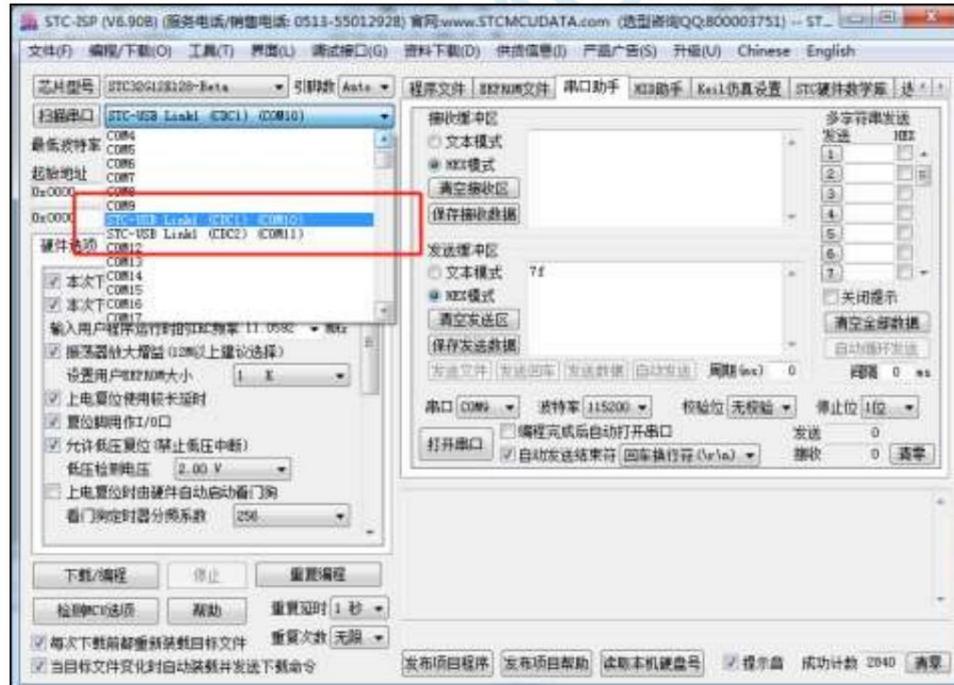
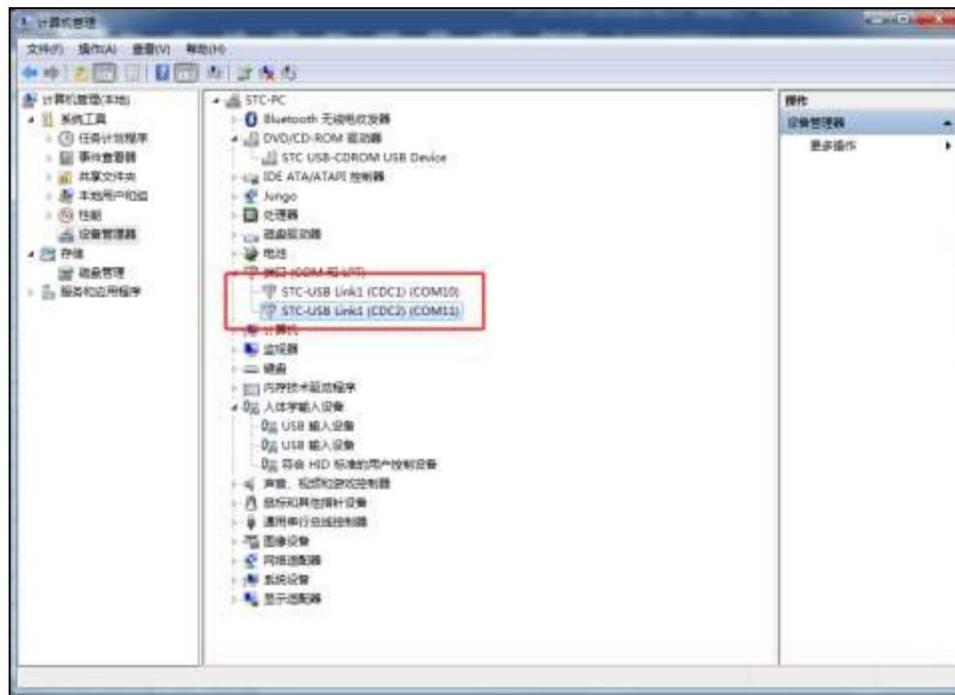
8. During the installation process, the "Windows Security" pop-up window will pop up, click "Always install this driver software"



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



10. The installation method of the second CDC virtual serial port driver is similar to the first one. After the drivers of the two virtual serial ports are installed, The STC-USB Link1D virtual serial port with the driver installed can be found in the device manager and STC-ISP software. (STC-ISP download software You may need to click the "Scan Serial Port" button to rescan to find the serial port)



### 5.13.7 How to close the virtual CD-ROM auto-play pop-up window

STC-USB Link1D is a combination device with multiple USB interfaces, which contains a virtual CD-ROM drive, which contains CDC virtual string

The driver of the port is convenient for customers to install the driver when they use the tool for the first time. After the driver is installed, it will no longer be used during the subsequent use.

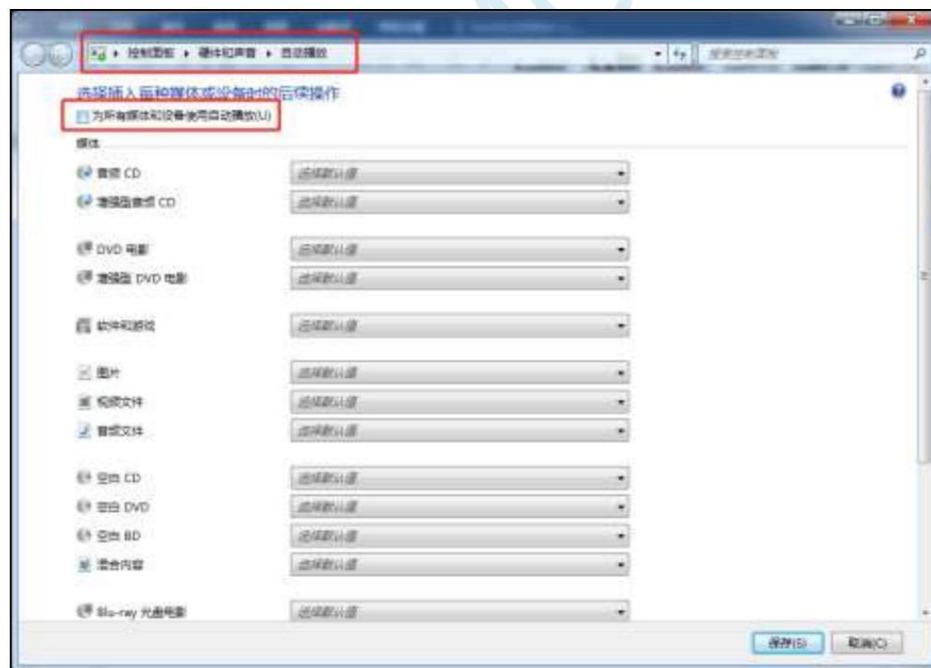
A virtual CD-ROM is required, but every time the STC-USB Link1D tool is inserted into the USB port of the computer, there will still be an auto-play pop-up of the virtual CD-ROM. window, as shown below:



Method 1. Turn off the autoplay popup through the system settings in Windows

In "Control Panel" → "Hardware and Sound" → "AutoPlay", uncheck "Use AutoPlay for all media and devices".

Then save and exit. As shown below

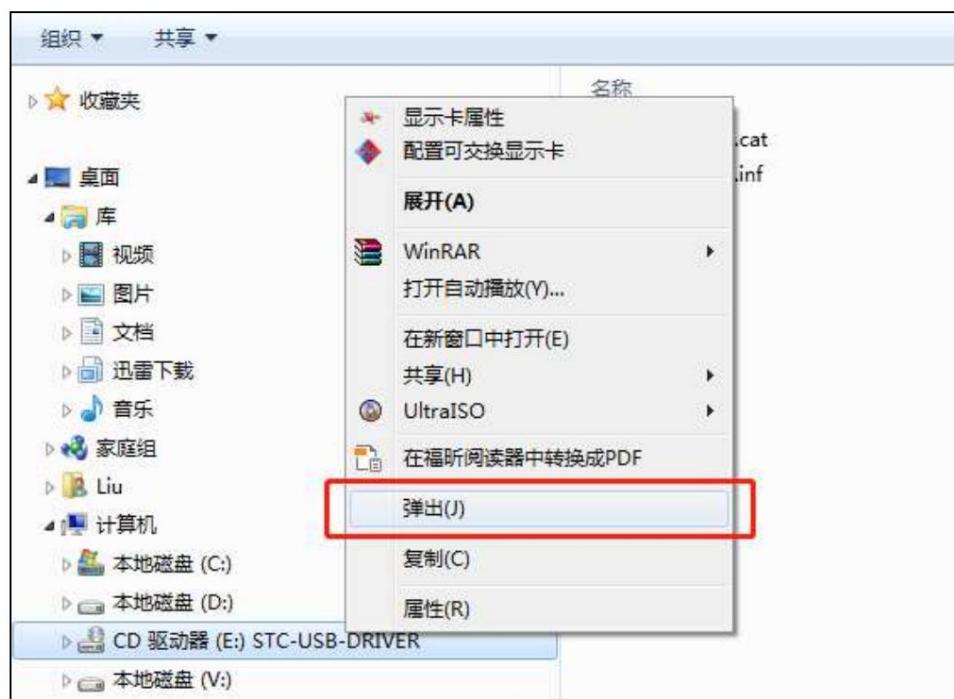


Method 2. By ejecting the media device in the virtual optical drive

Right-click on the virtual CD-ROM icon, and click "Eject" in the right-click menu (Note: Ejecting the media means erasing media data, i.e.

Permanently delete the driver from the media, irrecoverable even by power cycle of the tool. Remake Link1 main control chip or tool firmware

The media data in the virtual CD-ROM will be restored during automatic upgrade. )



The effect after the data pops up



## 5.14 Use ISP for programming

First use the serial port tool to connect the chip to be programmed with the computer correctly, and then open the STC ISP download software (for example: "STC-ISP (Ver6.87Q)" and later versions)



In the above interface, the following points should be noted:

1. Select the MCU model to be programmed, such as: "STC32G12K128"
2. The serial port must select 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 dialog box to open the program code file that appears:

After the file is opened correctly, set the corresponding hardware options, and then click the "Download/Program" button in the interface to start the download process:

At this point, the ISP software begins to try to shake hands with the microcontroller to detect the target microcontroller.

**Next, the microcontroller needs to be powered on, and the handshake signal is detected during the power-on reset process, and the download process starts to start downloading.**

If the download is successful, a prompt screen indicating that the operation is successful will appear.

## 5.15

### Instructions for ISP download related hardware options

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

It needs to be powered on again to take effect: after the option is modified, the target chip needs to be powered off once (power failure), and then powered on again, the new settings will take effect

Dynamic adjustment, effective immediately: This ISP download is valid

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

Valid at next download: After the option is modified, it will take effect at the next download, and the modification is invalid for this ISP download

Write with each download: After selecting this option, the additional data will be written together with this download, irrespective of the next download

## 5.16 The method of resetting the user program to the system area for ISP download in USB mode (not power failure)

When the project is in the development stage, it is necessary to repeatedly download the user code to the target chip for code verification, and use the USB mode to

STC's microcontroller performs normal ISP download. It needs to short-circuit the P3.2 port to GND first, and then power on the target chip again.

However, it will make the burning steps of the project in the development stage more cumbersome. To this end, the STC microcontroller adds a special function register IAP\_CONTR,

When the user writes 0x60 to this register, the software can be reset to the system area, and the ISP download can be performed without power failure.

**Note:** When the user program is soft reset to the system area, if P3.0/D- and P3.1/D+ have been connected to the USB port of the computer, the system code

It will automatically enter the USB download mode and wait for the ISP to download. At this time, it is not necessary to connect P3.2 to the ground.

The following two methods are described below:

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

Here, use the button of port P3.2 to trigger soft reset and the method of "short-circuit P3.2 port to GND, and then re-power on the target chip"

The law is different. In the main loop of the user program, judge the level state of the P3.2 port, when it is detected that the level of the P3.2 port is 0, trigger the software reset

Go to the system area to download the USB ISP. When the key of port P3.2 is in the release state, the level read by the user program from port P3.2 is 1.

When you need to reset to ISP for USB download, just manually press P3.2.

The sample program for judging the level of P3.2 in the program is as follows:

---

```
//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

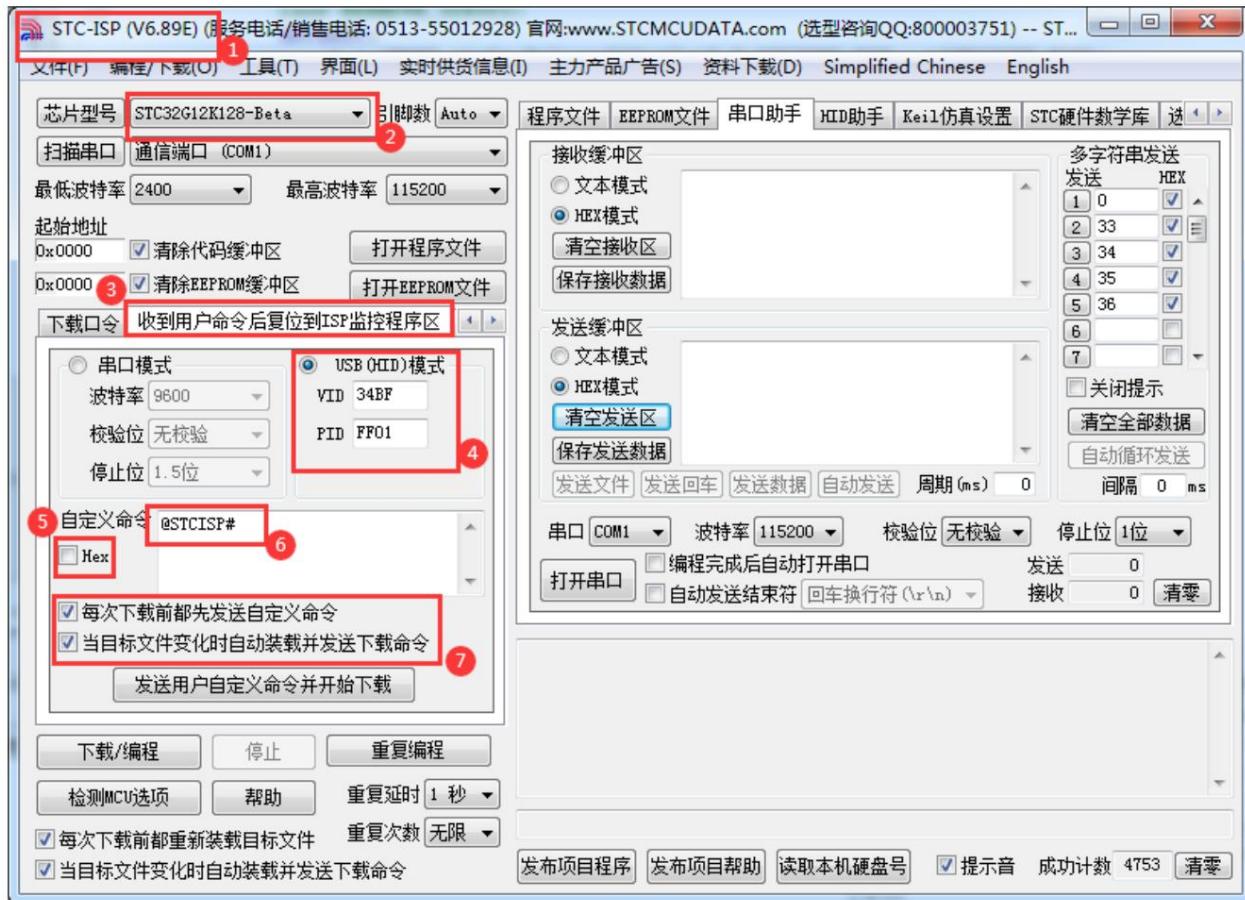
    P3M0 = 0x00;
    P3M1 = 0x00;
    P32 = 1;

    while (1)
    {
        if (!P32) IAP_CONTR = 0x60; //When the detected level is low
        //Software reset to system area

        ...
    }
}
```

---

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



1. Download the latest version of the STC-ISP download software
2. Select the correct MCU model
3. Open the option page of "Reset to ISP monitoring program area after receiving user command"
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".  
PID is "FF01"
5. Select HEX mode or text mode
6. Set the custom download command, which needs to be consistent with the custom command in the code
7. Select the above two items, when the target code is recompiled, the STC-ISP download software will automatically send a reset command and start automatically ISP download in USB mode

Note: If you need to use this mode, you must add the "stc\_usb\_hid.lib" code library provided by STC to the project, and follow the steps below

Set the custom download command as shown.

The screenshot shows the µVision4 IDE interface for a project named "stc\_usb\_hid". The left pane displays the project structure under "Target 1 / Source Group 1", which includes files like stc.h, intrins.h, stdio.h, string.h, stc32g.h, config.h, and usb.h, along with a library file "stc\_usb\_hid.LIB" highlighted with a red box and labeled "1 将STC提供的HID代码包加入项目中". The right pane shows the "main.c" source code:

```
01 #include "stc.h"
02 #include "usb.h"
03
04 void sys_init();
05 void DelayXms(int n);
06
07 char *USER_DEVICEDESC = NULL;
08 char *USER_PRODUCTDESC = NULL;
09
10 char *USER_STCISPCMD = "@STCISP#"; //设置自动复位到ISP区自
11
12 BYTE xdata cod[8];
13
14 void main()
15 {
16     sys_init();
17     usb_init();
18     EA = 1; //调用USB初始化代码
19
20     while (1)
21     {
```

Annotations with red numbers 1, 2, and 3 point to specific parts of the code:

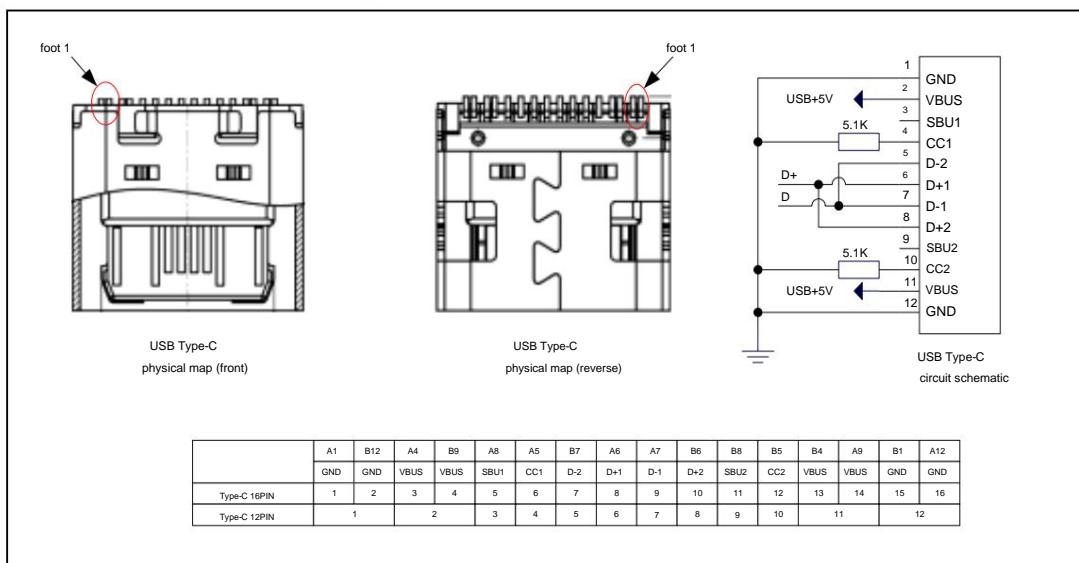
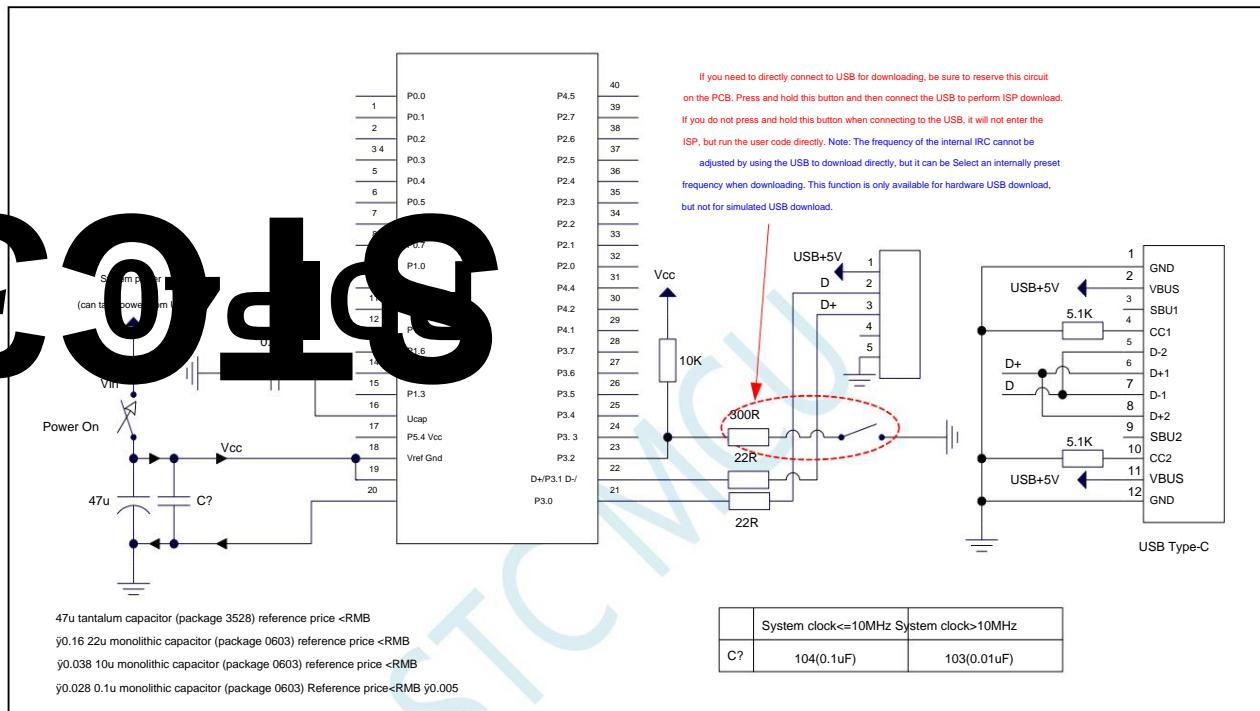
- Annotation 1: Points to the "stc\_usb\_hid.LIB" entry in the project tree.
- Annotation 2: Points to the line "char \*USER\_STCISPCMD = "@STCISP#";" with the comment "//设置自动复位到ISP区自".
- Annotation 3: Points to the line "EA = 1;" with the comment "调用USB初始化代码".

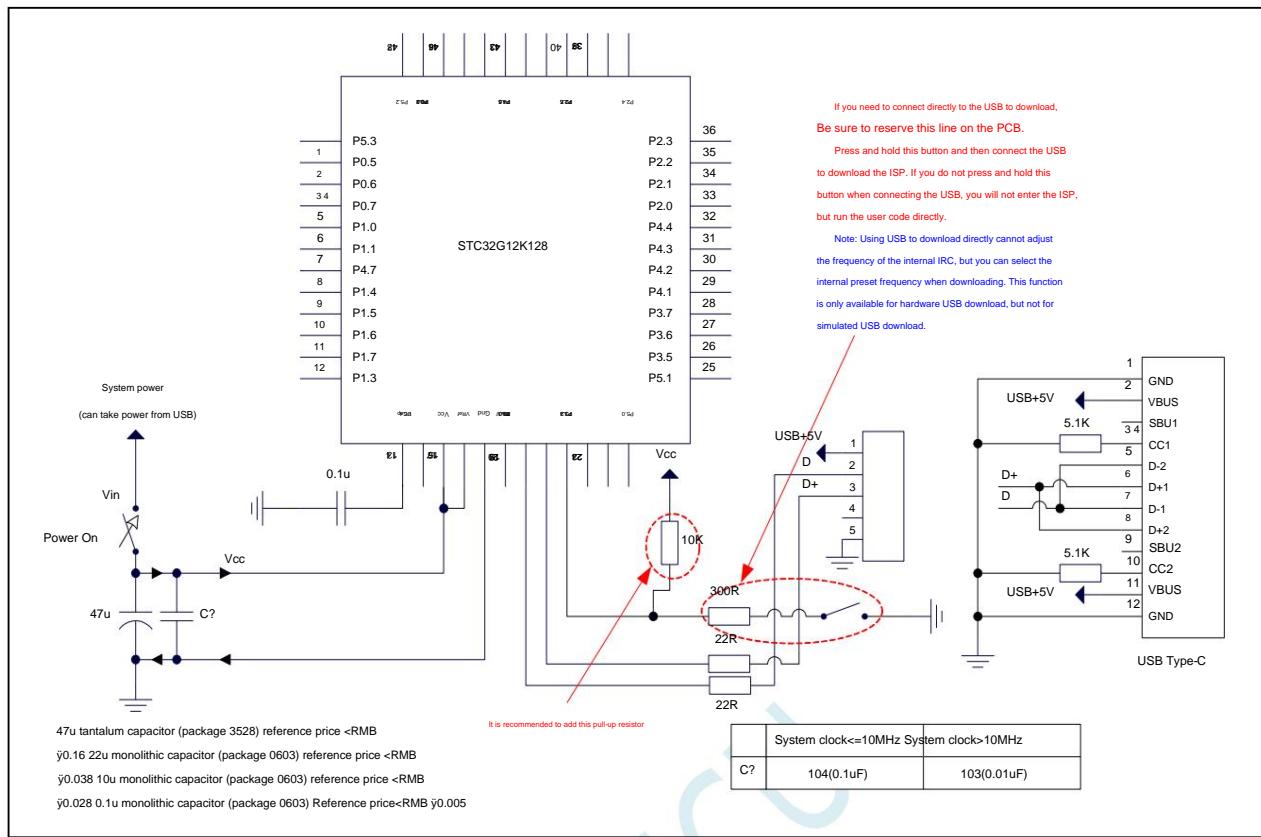
For the detailed code, please refer to "76-Printing data information through USB HID protocol-available for debugging"

## 5.17 ISP download typical application circuit diagram

### 5.17.1 Hardware USB direct ISP download, emulation follow-up support

Note 1: When using USB to download, it is necessary to connect P3.2 to Gnd for normal download (P3.2 only needs to be connected to Gnd during the power-on process, and it does not need a direct Gnd during the download process, and it does not matter all the time) Note 2 : If no USB download is required, P3.0/P3.1/P3.2 must not be low at the same time when the chip is reset, otherwise the chip will always be in USB download mode without running user code





#### ISP download steps:

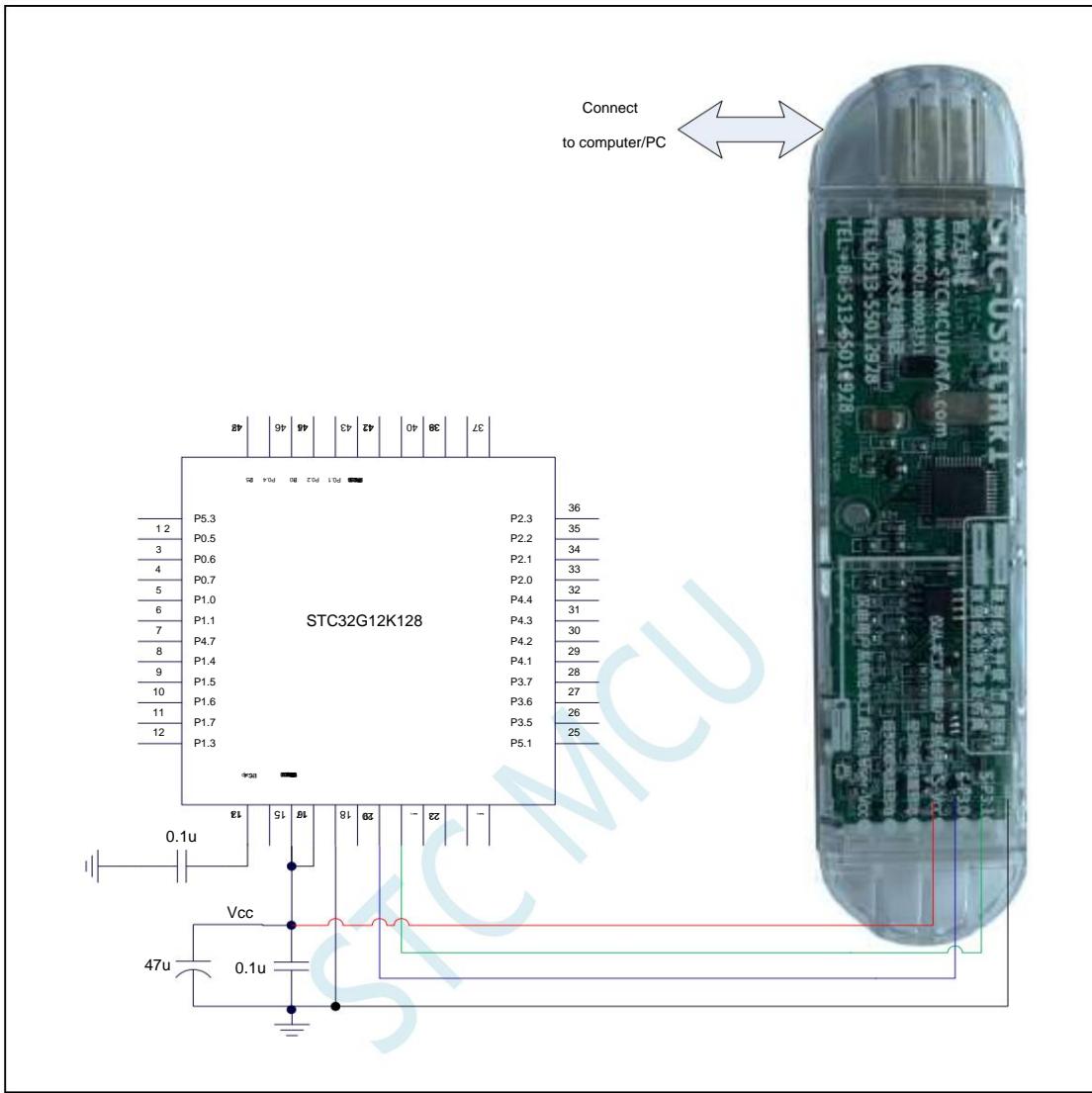
1. Power off the target chip 2.

Connect P3.0/P3.1 to the USB port as shown in the figure 3. Short-circuit P3.2 and GND 4.

Power on the target chip, And wait for the "STC **USB Writer (HID1)**" to be automatically  
recognized in the STC-ISP download software 5. Click the "Download/Program" button in the download software (Note: The sequence of  
operations is different from the serial port download, do not click first The download button must wait until the computer recognizes the "STC **USB Writer (HID1)**"  
device, and then click the download button to start the download) 6. Start ISP download Note: At present, it has been found that when using the USB cable  
to supply power for ISP download, because the USB cable If it is too thin, the voltage drop on the USB cable is too large, resulting in insufficient power supply  
during ISP downloading, so please use a USB enhanced cable when using the USB cable to supply power for ISP downloading.

When the user uses hardware USB to download the STC32G12K128 series ISP to the STC32G12K128 series, the frequency of the internal IRC cannot be  
adjusted, but the user can choose the 16 preset frequencies (respectively 5.5296M, 6M, 11.0592M, 12M, 18.432M, 20M), 22.1184M,  
24M, 27M, 30M, 33.1776M, 35M, 36.864M, 40M, 44.2368M and 48M, different series may be different, the specific frequency list of downloaded software shall  
prevail). When downloading, the user can only select one of the frequencies from the drop-down list, and cannot manually enter other frequencies. (Any frequency  
between 4M and 48M can be input when using serial port to download).

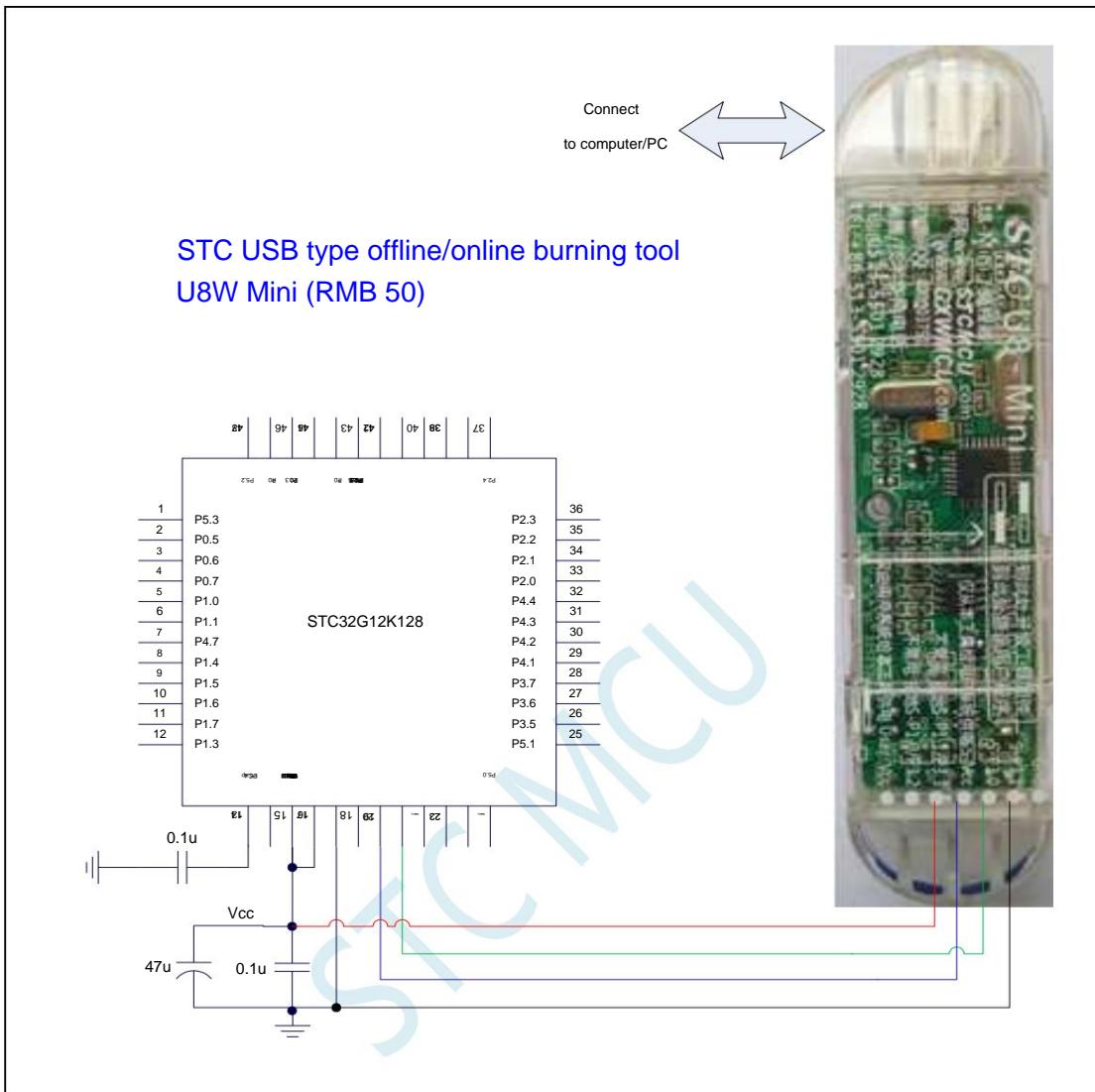
## 5.17.2 Use STC-USB Link1 tool to download, support ISP online and offline download



### ISP download steps:

1. Connect STC-USB Link1 to the target chip according to the connection method shown in the figure 2.
  2. Click the "Download/Program" button in the STC-ISP download software
  3. Start ISP download
- USB Link1 supplies power to the target system, and the total current of the target system cannot be greater than 200mA, otherwise the download will fail. Note: At present, it has been found that when using **USB power supply for ISP download**, because the **USB cable** is too thin, the voltage drop on the **USB cable** is too large, resulting in insufficient power supply during **ISP download**, so please be sure to use **USB power for ISP download**. **Use a USB cable**.

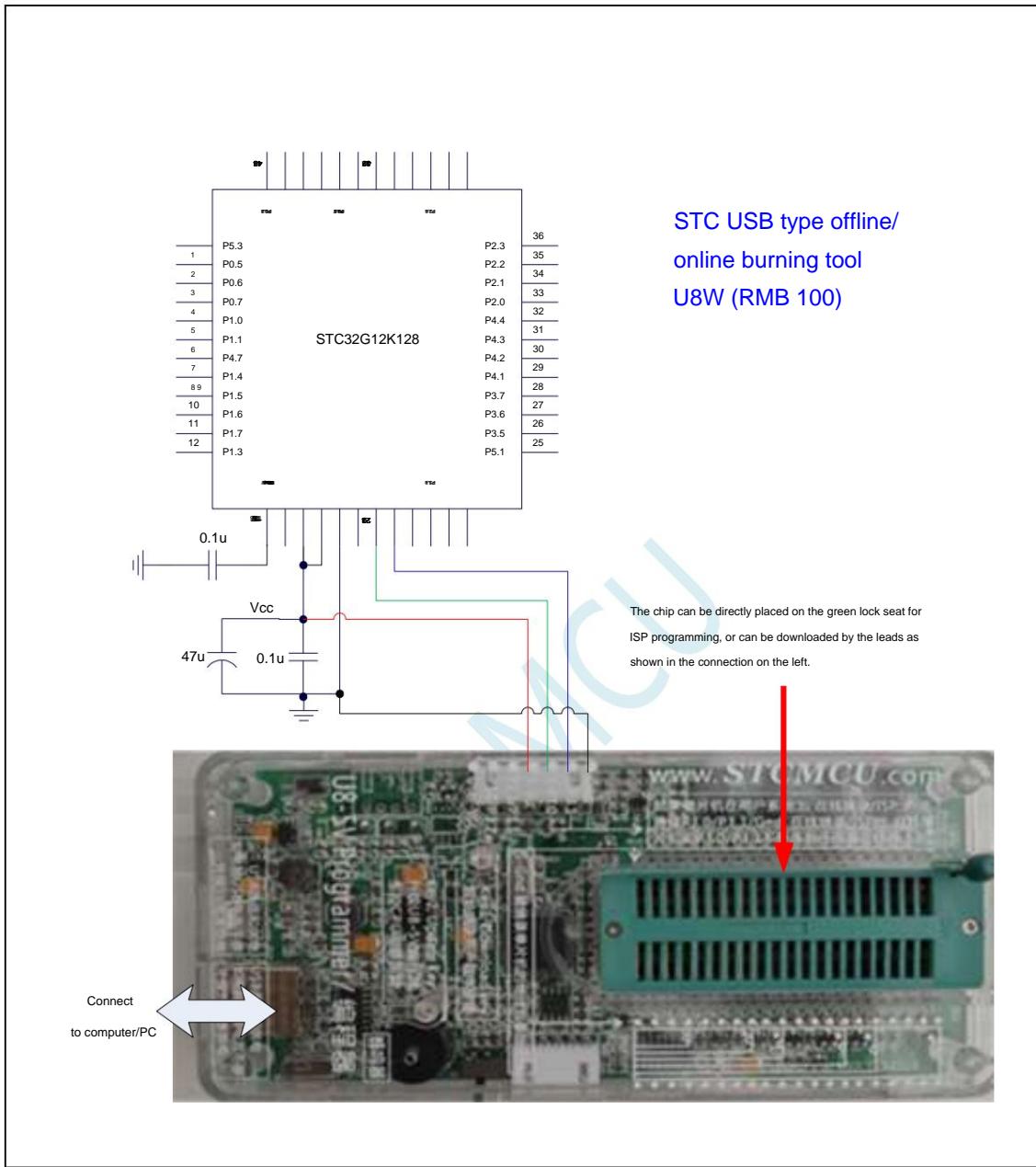
### 5.17.3 Use U8-Mini tool to download, support ISP online and offline download



#### ISP download steps:

4. Connect the U8-Mini to the target chip according to the connection method shown in the figure.
5. Click the "Download/Program" button in the STC-ISP download software.
6. Start the ISP download. Note: If U8-Mini is used Supply power to the target system, the total current of the target system cannot be greater than 200mA, otherwise the download will fail. Note: At present, it has been found that when using **USB** power supply for **ISP** download, because the **USB** cable is too thin, the voltage drop on the **USB** cable is too large, resulting in insufficient power supply during **ISP** download, so please be sure to use **USB** power for **ISP** download. Use a **USB** cable.

### 5.17.4 Use U8W tool to download, support ISP online and offline download



**ISP download steps (connection method): 1.**

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

Start the ISP download Note: if using U8W supplies power to the target system, and the total current of the target system cannot be greater than 200mA, otherwise the download will fail.

**ISP download steps (on-board mode): 1.**

Place the chip in the direction that pin 1 is close to the locking wrench and the pins are aligned downward. 2. Click the "download/programming" button in the STC-ISP download software to start ISP download

### 5.17.5 U8W pass-through mode, which can be used for emulation and serial communication

To emulate with the **U8W**, you must first set the **U8W to pass-through mode**. U8W/U8W-Mini realizes **USB -to-serial pass-through mode**

The formula method is as follows:

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

2. After the U8W/U8W-Mini is powered on, it is in the normal download mode. At this time, press and hold the Key1 (download) button on the tool and do not release it, then press it again.

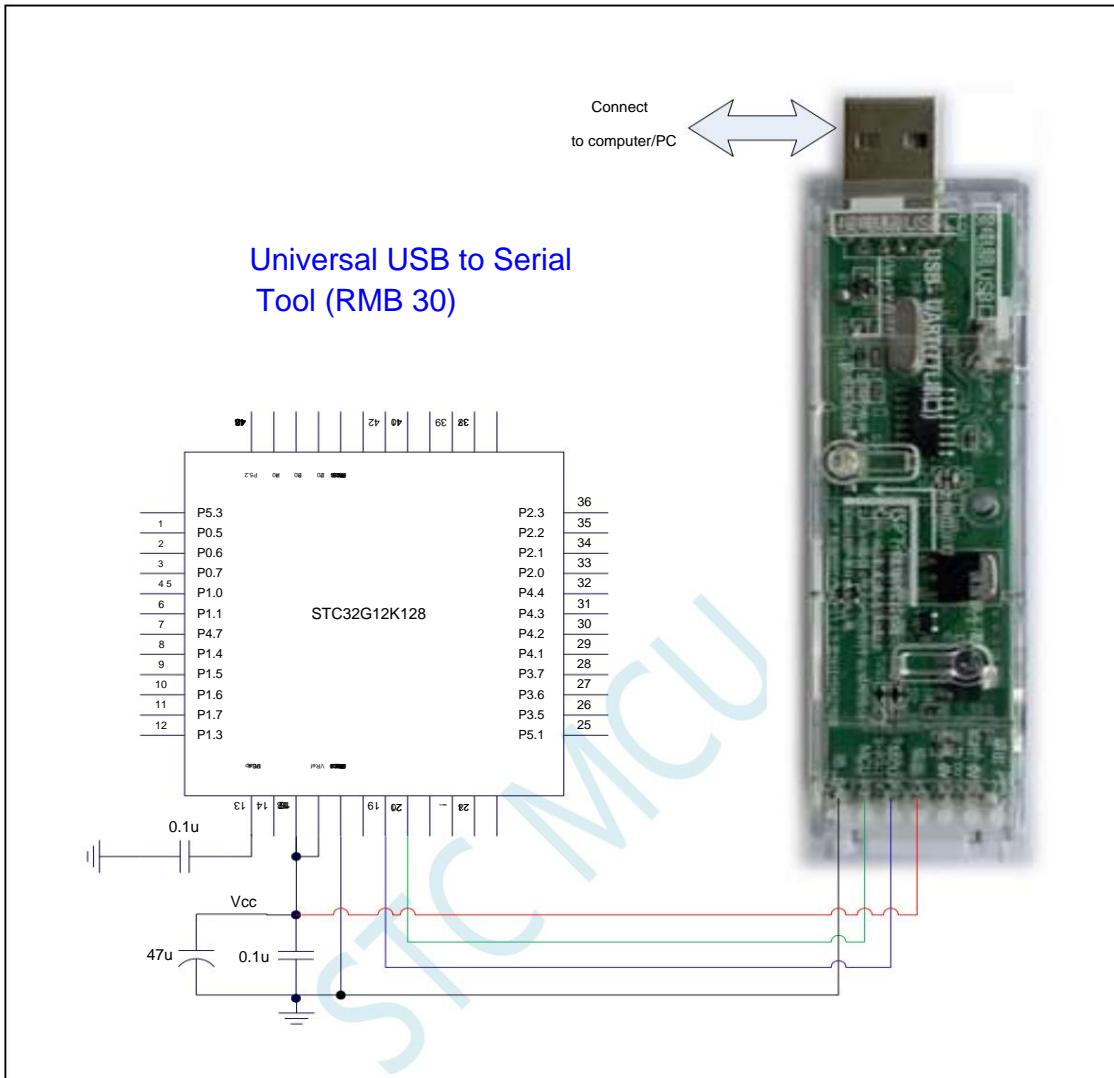
Key2 (power) button, then release the Key2 (power) button, then release the Key1 (download) button, the U8W/U8W-Mini will

Enter USB-to-serial pass-through mode. (Press **Key1** → Press **Key2** → Release **Key2** → Release **Key1**)

3. The U8W/U8W-Mini tool that enters the pass-through mode is just a simple USB to serial port and does not have the offline download function. If you need to restore it

The original function of U8W/U8W-Mini, just need to press the Key2 (power) button again.

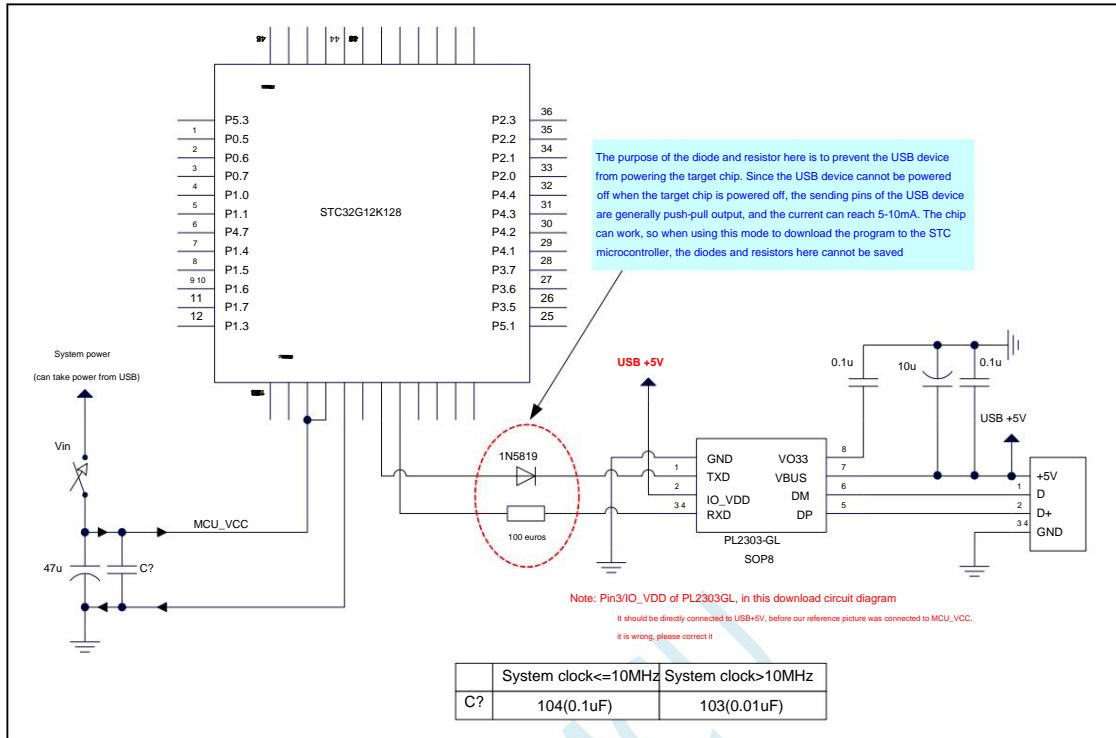
### 5.17.6 Download using general USB to serial port tool



#### ISP download steps:

1. Connect the universal USB to serial port tool to the target chip according to the connection method shown in the figure.
2. Press the power button to confirm that the target chip is in a power-off state (the power-on indicator is off). NOTE: When the tool is powered on for the first time there is no external power supply, so if you use this tool for the first time, you can skip this step.
3. Click the "Download/Program" button in the STC-ISP download software.
4. Press the power button again to power on the target chip (the power-on indicator is on).
5. Start the ISP download. When the **USB** cable is powered for **ISP download**, because the **USB** cable is too thin, the voltage **drop** on the **USB** cable is too large, resulting in insufficient **power** supply during **ISP downloading**.

## 5.17.7 Download using PL2303-GL



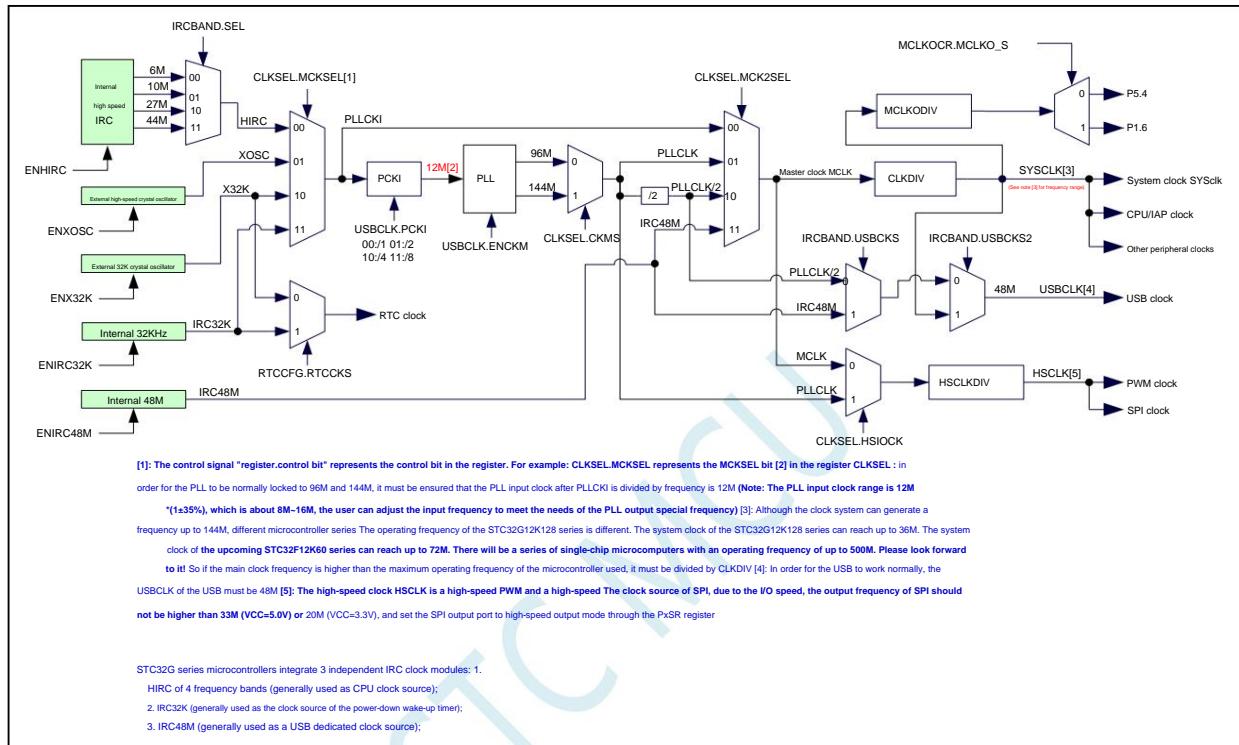
### ISP download steps:

1. Power off the target chip, be careful not to power off the USB-to-serial chip (such as CH340, PL2303-GL, etc.) 2. Since the sending pin of the USB-to-serial chip is generally a strong push-pull output, it must be in the target A diode should be connected in series between the P3.0 port of the chip and the sending pin of the USB-to-serial chip, otherwise the target chip cannot be completely powered off, and the goal of powering off the target chip cannot be achieved. 3. Click the "Download/Program" button in the **STC - ISP** download software. 4. Power on the **target chip**. 5. Start **ISP download**. The voltage drop on the line is too large, resulting in insufficient power supply during ISP download, so please be sure to use a USB reinforced cable when using the **USB cable to supply power for ISP download**.

## 6 Clock management

### 6.1 System Clock Control

The system clock controller provides the clock source for the CPU and all peripheral systems of the single-chip microcomputer. There are 4 clock sources for the system clock to choose from: internal high-precision IRC, internal 32KHz IRC (large error), external crystal oscillator, internal PLL output clock . The user can enable and disable each clock source separately through the program, and provide the clock frequency division internally to achieve the purpose of reducing power consumption.



System and peripheral clock selection reference table (refer to the sample program for detailed settings)

Master clock selection (MCLK)	Internal IRC	Internal high-speed IRC
		(HIRC) Internal low-speed IRC
		(IRC32K) Internal USB dedicated 48M high-speed IRC
	External crystal	(IRC48M) External high-speed crystal oscillator (XOSC)
		External low-speed crystal oscillator (X32K) Internal PLL
	PLL	clock (PLL) Internal PLL clock divided by 2 (PLL/2) Main clock (MCLK) Internal PLL clock (PLL) Internal
High-speed peripheral clock selection (HSIOCK)	USB dedicated 48M high-speed IRC (IRC48M) Internal PLL clock divided by 2 (PLL/2) System clock (SYSCLK) Note: The system clock (SYSCLK)	
	is the main clock (MCLK) through Clock divided by CLKDIV	
USB clock selection (48M)		

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 oscillator (XOSC)
10	External low-speed crystal oscillator (X32K)
11	Internal low-speed IRC (IRC32K)

## 6.2 Related registers

## Related registers

symbol	describe	address	Bit addresses and symbols								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 xx	xxx,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

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	Clock selection 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 1xx,xxx0	
XOSCCR	External crystal oscillator control register 7EFE03H ENXOSC XTYPE			GAIN			XCFILTER[1:0]			XOSCST 000x,00x0	
IRC32KCR	Internal 32K oscillator control register 7EFE04H ENIRC32K		-	-	-	-	-	-	-	IRC32KST 0xx,xxx0	
MCLKOCR	Master Clock Output Control Register 7EFE05H MCLKO_S		MCLKODIV[6:0]								0000,0000
IRCDB	Internal high-speed oscillator stabilization time control 7EFE06H										1000,0000
IRC48MCR	Internal 48M oscillator control register 7EFE07H ENIRC48M		-	-	-	-	-	-	-	IRC48MST 0xx,xxx0	
X32KCR	External 32K crystal oscillator control register 7FFE08H ENX32K GAIN32K				-	-	-	-	-	X32KST 00xx,xxx0	
HSCLKDIV	High-speed clock divider register 7EFE0BH										0000,0010
HPLLCR	High-Speed PLL Control Register	7EFE0CH ENHPLL	-	-	-	-			HPLLDIV[3:0]		0xxx,0000
HPLLPSR	High-speed PLL prescaler register 7EFE0DH		-	-	-	-			HPLL_PREDIV[3:0]		xxxx,0000

### 6.2.1 USB Clock Control Register (USBCLK)

Symbol 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 multiplication

1: Enable PLL frequency multiplication

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: Disable clock tracking

1: Enable clock frequency tracking

STCMCU

### 6.2.2 System Clock Select Register (CLKSEL)

symbol	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: The main clock MCLK is the high-speed I/O clock source

1: PLL outputs 96MHz/144MHz PLLCLK as the high-speed I/O clock source

MCK2SEL[1:0]: main clock source selection

MCK2SEL[1:0]	main clock source
00	Clock source selected by MCKSEL
01	Internal PLL output
10	Internal PLL output/2
11	Internal 48MHz high-speed IRC

MCKSEL[1:0]: main clock source selection

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

### 6.2.3 Clock Divider Register (CLKDIV)

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

CLKDIV: Main clock frequency division factor. The system clock SYSCLK is the frequency-divided clock signal of the main clock MCLK.

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

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

symbolic address	B7	B6	B5	B4	B3	B2	B1	B0
HIRCCR 7EFE02H ENHIRC		-	-	-	-	-	-	HIRCST

ENHIRC: Internal high-speed high-precision IRC enable bit

0: Disable the internal high precision IRC

1: Enable internal high precision IRC

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

When the internal IRC is enabled from the stop state, it takes a period of time for the oscillator frequency to stabilize. When the oscillator frequency

After stabilization, the clock controller will automatically set the HIRCST flag bit. So when the user program needs to switch the clock to use the internal IRC

When the oscillator is set, ENHIRC=1 must be set to enable the oscillator, and then the oscillator stable flag bit HIRCST must be checked until the flag bit

When it becomes 1, the clock source can be switched.

## 6.2.5 External Oscillator Control Register (XOSCCR)

symbolic 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: Turn off the 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 oscillator). The signal source only needs to be 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-jamming control register

00: This option can be selected when the external crystal oscillator frequency is 48M and below

01: This option can be selected when the external crystal oscillator frequency is 24M and below

1x: This option can be selected when the external crystal oscillator frequency is 12M and below

GAIN: External crystal oscillator oscillation gain control bit

0: Disable oscillation gain (low gain)

1: Enable oscillation gain (high gain)

XOSCST: External crystal oscillator frequency stability flag. (read-only bit)

When the external crystal oscillator is enabled from the stop state, it must pass a period of time before the frequency of the oscillator is stable.

After the frequency is stable, the clock controller will automatically set the XOSCST flag bit. So when the user program needs to switch the clock to use

When using a crystal oscillator, you must first set ENXOSC=1 to enable the oscillator, and then keep querying the oscillator stability flag XOSCST,

Clock source switching is not possible until the flag bit becomes 1.

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

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

ENIRC32K: Internal 32K low-speed IRC enable bit

0: Disable the internal 32K low-speed IRC

1: Enable internal 32K low-speed IRC

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

When the internal 32K low-speed IRC is enabled from the vibration-stop state, it must pass a period of time before the frequency of the oscillator is stable.

After the clock controller frequency is stable, the clock controller will automatically set the IRC32KST flag bit to 1. So when the user program needs to switch the clock to enable

When using the internal 32K low-speed IRC, you must first set ENIRC32K=1 to enable the oscillator, and then always query the oscillator stable flag

Bit IRC32KST, the clock source switching is not possible until the flag bit becomes 1.

### 6.2.7 Master Clock Output Control Register (MCLKOCR)

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

MCLKODIV[6:0]: Main clock output frequency division factor

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

MCLKO\_S: System clock output pin selection

0: The system clock is divided and output to port P5.4

1: The system clock is divided and output to port P1.6

### 6.2.8 High-speed oscillator stabilization time control register (IRCDB)

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

IRCDB[7:0]: Internal high-speed oscillator stabilization 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)

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

ENIRC48M: Internal 48M high-speed IRC enable bit

0: Disable the internal 48M high-speed IRC

1: Enable the internal 48M high-speed IRC

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

When the internal 48M high-speed IRC is enabled from the vibration-stop state, a period of time must pass before the frequency of the oscillator is stable.

After the oscillator frequency is stable, the clock controller will automatically set the IRC48MST flag to 1. So when the user program needs to switch the clock to

When using the internal 48M high-speed IRC, you must first set ENIRC48M=1 to enable the oscillator, and then keep querying the oscillator stability standard.

When the flag bit IRC48MST is set, the clock source switching can not be performed until the flag bit becomes 1.

### 6.2.10 External 32K Oscillator Control Register (X32KCR)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
X32KCR	7EFE	08H	ENX32K	GAIN32K	-	-	-	-	X32KST

ENX32K: External 32K crystal oscillator enable bit

0: Turn off the external 32K crystal oscillator

1: Enable external 32K crystal oscillator

GAIN32K: External 32K crystal oscillator oscillation gain control bit

0: Disable 32K oscillation gain (low gain)

1: Enable 32K oscillation gain (high gain)

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

### 6.2.11 High Speed Clock Divider Register (HSCLKDIV)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
HSCLKDIV	7EFE	0BH							

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 sources/2
3	High-speed I/O clock source/3
...	...
x	High-speed I/O clock source/x
...	...
255	High-speed I/O clock sources/255

## 6.3 STC32G series internal IRC frequency adjustment

STC32G series microcontrollers are integrated with a high-precision internal IRC oscillator. After the user uses the ISP download software to download ISP download software will automatically adjust according to the frequency selected/set by the user. Generally, the frequency value can be adjusted to less than  $\pm 0.3\%$ . The temperature drift of the latter frequency in the whole temperature range (-40°C~85°C) can reach -1.35%~1.30%.

The internal IRC of STC32G series has 4 frequency bands, the center frequencies of the frequency bands are 6MHz, 10MHz, 27MHz and 44MHz respectively. The adjustment range of each frequency band is about  $\pm 27\%$  (note: different chips and different production batches may have a manufacturing error of about 5%).

Note: For general users, the adjustment of the internal IRC frequency does not need to be concerned, because the frequency adjustment works during the ISP download has been done automatically. Therefore, if the user does not need to adjust the frequency by himself, then the following four related registers cannot be modified at will, otherwise May cause operating frequency changes.

The internal IRC frequency adjustment is mainly adjusted using the following 4 registers

### 6.3.1 IRC Band Select Register (IRCBAND)

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

USBCKS/USBCKS2: USB Clock Select Register

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

SEL[1:0]: Band selection

00: select 6MHz frequency band

01: Select 10MHz frequency band

10: Select 27MHz frequency band

11: Select 44MHz frequency band

### 6.3.2 Internal IRC Frequency Trim Register (IRTRIM)

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

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

IRTRIM can adjust the IRC frequency in 256 levels. The frequency value adjusted by each level is linearly distributed as a whole.

Department will fluctuate. Macroscopically, the frequency adjusted by each stage is about 0.24%, that is, the frequency ratio of IRTRIM when IRTRIM is (n+1) is about 0.24% faster when (n). But due to the IRC frequency adjustment not every stage is 0.24% (the

The maximum value is about 0.55%, the minimum value is about 0.02%, and the overall average is about 0.24%), so it will cause local fluctuations.

### 6.3.3 Internal IRC Frequency Trim Register (LIRTRIM)

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

LIRTRIM: Internal high precision IRC frequency trim register

### 6.3.4 Clock Divider Register (CLKDIV)

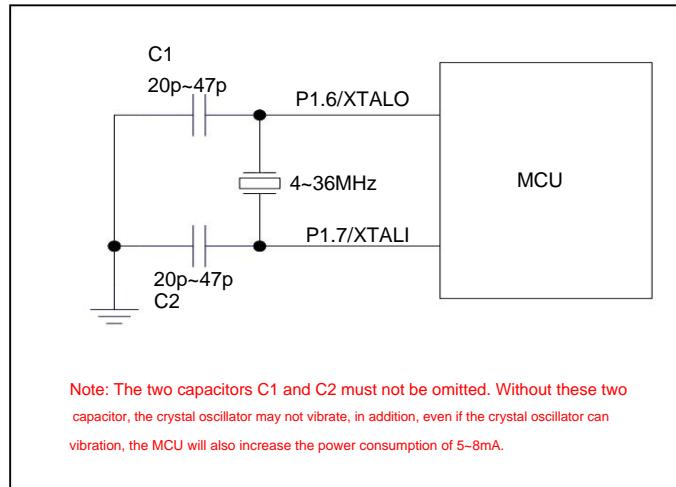
symbol	address B7	B6	B5	B4	B3	B2	B1	B0
CLKDIV	7EFE01H							

CLKDIV: Main clock frequency division factor. The system clock SYSCLK is the frequency-divided clock signal of the main clock MCLK.

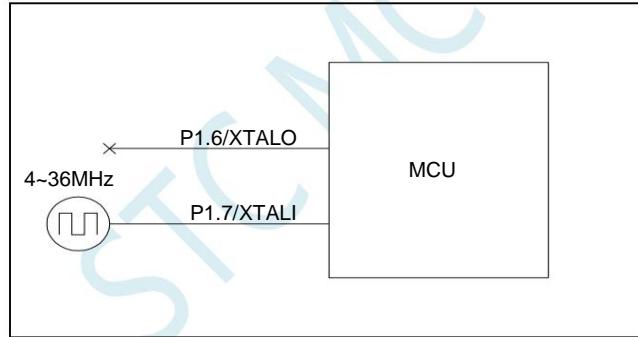
CLKDIV	system clock frequency
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
...	...
x	MCLK/x
...	...
255	MCLK/255

## 6.4 External crystal oscillator and external clock circuit

### 6.4.1 External crystal oscillator input circuit



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



Note: When using the internal clock as the main clock source, P1.6/P1.7 can be used as normal I/O

## 6.5 Sample Program

### 6.5.1 Select Internal High Speed IRC (HIRC) as System Clock Source

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

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

    HIRCCR = 0x80; //Note: After the chip is powered on, the system will automatically start the HIRC ,
    while (!(HIRCCR & 1)); //internal high-speed and select it as the system clock, so it is generally not necessary to do so.
    CLKSEL = 0x00; //Set as follows
                  // Start internal high speed HIRC
                  // waiting for clock stabilization
                  // select internal high speed HIRC

    while (1);
}
```

---

### 6.5.2 Select Internal IRC (IRC32K) as System Clock Source

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
```

---

```
//assign as 0 can be CPU The speed of executing the program is set to the fastest

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;
while (!(IRC32KCR & 1));
CLKDIV = 0x00; CLKSEL
= 0x03; //Start the internal 32K IRC
//wait for the clock to stabilize
//The clock is not divided
//select inside 32K

while (1);
}
```

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

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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;
    while (!(IRC48MCR & 1));
    CLKDIV = 0x02; CLKSEL
    = 0x0c; //Start the internal 48M IRC
    //wait for the clock to stabilize
    //clock division
    //chooseIRC48M
```

```

    while (1);
}

```

#### 6.5.4 Select external high-speed crystal oscillator (XOSC) as system clock source

```

//The test frequency is 11.0592MHz

#ifndefinclude "stc8h.h"
#include "stc32g.h"                                     //For header files, see Download Software

void main()
{
    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to      CPU The speed of executing the program is set to the fastest

    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 the external crystal
    while (!(XOSCCR & 1));                                //Wait for the clock to stabilize
    CLKDIV = 0x00;                                         //The clock is not divided
    CLKSEL = 0x01;                                         //Choose an external crystal

    while (1);
}

```

#### 6.5.5 Select external low-speed crystal oscillator (X32K) as system clock source

```

//The test frequency is 11.0592MHz

#ifndefinclude "stc8h.h"
#include "stc32g.h"                                     //For header files, see Download Software

void main()
{
    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to      CPU The speed of executing the program is set to the fastest
}

```

```

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;
while (!(X32KCR & 1));
CLKDIV = 0x00;
CLKSEL = 0x02;
                                // Start the external 32K crystal
                                // Wait for the clock to stabilize
                                // The clock is not divided
                                // select external 32K crystal oscillator

while (1);
}

```

## 6.5.6 Selecting Internal PLL as System Clock Source

```

//The test frequency is 12MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software

void delay()
{
    int i;

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

void main()
{
    EAXFR = 1;                                     //Enable      XFR
    CKCON = 0x00;                                   //access to set external data bus speed to fastest
    WTST = 0x00;                                    //Set the program code to wait for parameters,
                                                    //assign to      CPU The speed of executing the program is set to the fastest

    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; //choose PLL of 96M as as PLL the output clock of
// CLKSEL |= 0x80; //choose PLL 144M PLL the output clock of

USBCLK &= ~0x60; //PLL The input clock 12M select 1 frequency division
USBCLK |= 0x00; //PLL is the input clock 24M then 2 frequency division
// USBCLK |= 0x20; //PLL is the input clock 48M select 4 frequency division
// USBCLK |= 0x40; //PLL is the input clock 96M select then 8 select
// USBCLK |= 0x60; //start up PLL

USBCLK |= 0x80; //wait PLL frequency lock, recommended 50us above

delay(); //Clock frequency division, the frequency division factor must be set before the ,,
CLKDIV = 0x04; //main clock selects the high-speed frequency, otherwise the program will crash
//Select clock source
CLKSEL &= 0xf0; //Select PLL
CLKSEL |= 0x04;

while (1);
}

```

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

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    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; //Select the master clock MCLK ) as a high-speed peripheral clock source
    CLKSEL &= ~0x40;

    while (1);
}

```

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

---

```
//The test frequency is 12MHz

#ifndef include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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; //choosePLL of 96M as asPLL the output clock of
    //CLKSEL |= 0x80; //choosePLL 144M PLL the output clock of

    USBCLK &= ~0x60; //PLL The input clock 12M select frequency 1 division
    USBCLK |= 0x00; //PLL is the input clock 24M select frequency 2 division
    //USBCLOCK |= 0x20; //PLL is the input clock 48M select frequency 4 division
    //USBCLOCK |= 0x40; //PLL is the input clock 96M select frequency 8 division
    //USBCLOCK |= 0x60;

    USBCLK |= 0x80; //start upPLL

    delay(); //wait PLL frequency lock, recommended 50us above

    HSCLKDIV = 0x00; //choosePLL Clock as high-speed peripheral clock source
    CLKSEL |= 0x40;

    while (1);
}
```

---

### 6.5.9 Select System Clock (SYSCLK) as USB Clock Source

---

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

```
#ifndef include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
```

```

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                             //Set the program code to wait for parameters,
                                              //assign to      CPU The speed of executing the program is set to the fastest

    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 the system clock (SYSCLK) as      USB clock source

    while (1);
}

```

### 6.5.10 Selecting Internal PLL Clock as **USB** Clock Source

---

```

//The test frequency is 12MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                           //For header files, see Download Software

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                             //Set the program code to wait for parameters,
                                              //assign to      CPU The speed of executing the program is set to the fastest

    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;                         //choose PLL of 96M as asPLL the output clock of
    // CLKSEL |= 0x80;                         //choose PLL 144M PLL the output clock of

```

```

USBCLK &= ~0x60;
USBCLK |= 0x00;
// USBCLK |= 0x20;
// USBCLK |= 0x40;
// USBCLK |= 0x60;

USBCLK |= 0x80; // start up PLL

delay(); // wait PLL frequency lock, recommended 50us above

IRCband &= ~0xc0; // choosePLL clock( 96M/2=48M ) as USB clock source

while (1);
}

```

### 6.5.11 Select the internal **USB** dedicated **48M IRC** as the **USB** clock source

//The test frequency is **11.0592MHz**

```

#ifndef "stc8h.h"
#include "stc32g.h"

void main()
{
    EAXFR = 1; //For header files, see Download Software
    CKCON = 0x00; //Enable XFR
    WTST = 0x00; //access to set external data bus speed to fastest
    // Set the program code to wait for parameters,
    // assign to CPU The speed of executing the program is set to the fastest

    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 the internal 48M IRC
    while (!(IRC48MCR & 1)); // wait for the clock to stabilize

    IRCBAND |= 0x80; // chooseIRC48M as USB clock source
    IRCBAND &= ~0x40;

    while (1);
}

```

### 6.5.12 Main clock frequency division output

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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;
    // MCLKOCR = 0x02;
    MCLKOCR = 0x04; // Main clock output to main P5.4 mouth
    // MCLKOCR = 0x84; // clock frequency 2 output to main clock P5.4 mouth
    // frequency division 4 output to main clock P5.4 mouth
    // output to 4 P1.6 mouth

    while (1);
}
```

---

## 7 Automatic frequency calibration, automatic frequency tracking (CRE)

product line	Automatic frequency tracking
STC32G12K128 series	
STC32G8K64 series	●
STC32F12K60 series	●

Some STC32G MCU series have a built-in frequency automatic calibration module (CRE). The CRE module uses an external 32.768KHz crystal oscillator automatically adjusts the IRTRIM register of the internal high-speed IRC (HIRC) to achieve the function of automatic frequency calibration. Need to use During automatic calibration, it is only necessary to set the count value and error range of the target frequency according to the given formula, and then start the CRE module, the hardware will Perform automatic frequency calibration. When the frequency of HIRC reaches the error range set by the user, the calibration completion flag will be set.

### 7.1 Related registers

symbol	describe	address	Bit addresses and symbols								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)

Symbol 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: Turn off the CRE module.

1: Enable the CRE module.

MONO: Auto-calibrate stride control

0: Single step mode. The hardware automatically increments or decrements IRTRIM by one every calibration cycle.

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

The single-step mode is more accurate than the two-step mode after calibration of the IRC, but the auto-calibration time is longer than the 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 greater than 50MHz).

CREINC: CRE calibration is being adjusted upwards. Read-only bit.

CREDEC: CRE calibration is being down-regulated. Read-only bit.

CRERDY: CRE calibration complete status. Read-only bit.

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

1: CRE calibration completed.

## 7.1.2 CRE Calibration Count Value Register (CRECNT)

Symbol 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.

Target calibration value calculation formula:

**Low frequency mode (CREHF=0):**  $CRECNT = (16 * \text{target frequency (Hz)}) / 32768$

**High frequency mode (CREHF=1):**  $CRECNT = (8 * \text{target frequency (Hz)}) / 32768$

(See the sample program for detailed settings)

## 7.1.3 CRE Calibration Error Value Register (CRERES)

Symbol 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 oscillator, the final calibration value cannot match the target value set by CRECNT.

All are consistent, so an error range must be set through the CRERES register.

Calibration error calculation formula:

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

(The error range is generally controlled within 1% to 0.3%, and it is not recommended to exceed this range)

(See the sample program for detailed settings)

## 7.2 Example program

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

For example: the calibration target frequency is 22.1184MHz, and the calibration error range is  $\pm 0.5\%$

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

That is, set CRECNTH to 2AH, CRECNTL to 30H, and CRERES to  $10800 * 0.5\% = 54$  (36H)

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software

#define CNT22M      (16 * 22118400L) / 32768           // The calibration target frequency 22.1184M
#define RES22M      (CNT22M *5 / 1000)                  // is set to the calibration error of 5%

void main()
{
    EAXFR = 1;                                         // Enable XFR
    CKCON = 0x00;                                       // access to set external data bus speed to fastest
    WTST = 0x00;                                         // Set the program code to wait for parameters,
                                                       // assign to CPU The speed of executing the program is set to the fastest

    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 the external crystal
    while (!(X32KCR & 1));                                // Wait for the clock to stabilize

    IRCBAND &= ~0x03;                                     // select frequency band
    IRCBAND |= 0x02;                                      // select internal high speed HIRC is the system clock

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

    while (1)
    {
        if (CRECR & 0x01)
        {
            // Frequency auto-calibration completed
        }
    }
}

```

}  
}  
}

---

---

STCMCU

## 8 Reset, watchdog, special timer and power supply for wake-up after power-down

### manage

#### 8.1 System reset

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

When the hardware is reset, the values of all registers will be reset to their initial values, and the system will re-read all hardware options. At the same time according to the hardware The power-on wait time set by the option is used to wait for power-on. Hardware reset mainly includes:

- ÿ Power-on reset
- ÿ Low voltage reset
- ÿ **Reset pin reset (low level reset)**
- ÿ Watchdog reset

When the software is reset, except the registers related to the clock remain unchanged, the values of all the other registers will be reset to the initial values, and the software resets.

Bit does not re-read all hardware options. Software reset mainly includes:

- ÿ Reset triggered by writing SWRST of IAP\_CONTR

##### Related registers

symbol	describe	address	Bit addresses and symbols								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	

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
RSTFLAG	reset flag register 7EFE99H		-	-	-	LVDRSTF	WDRSTF	SWRSTF	ROMOVF	EXRSTF xxx	,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	RSTLCD	RSTLED	RSTTKS	RSTCM	RSTADC	0000,0000			
RSTCR4	Reset Control Register 4 7EFE9EH		-	-	-	-	-	-	-	RSTMU	xxxx,xxx0
RSTCR5	Reset Control Register 5 7EFE9FH		-	-	-	-	-	-	-	-	xxxx,xxxx

#### 8.1.1 Watchdog Control Register (WDT CONTR)

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

WDT\_FLAG: Watchdog overflow flag

When the watchdog overflows, the hardware will automatically set this bit to 1, and it needs to be cleared by software.

EN\_WDT: Watchdog enable bit

0: No effect on the microcontroller

1: Start the watchdog timer

#### CLR\_WDT: Watchdog Timer Clear

0: No effect on the microcontroller

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

#### IDL\_WDT: Watchdog control bit in IDLE mode

0: Watchdog stop counting in IDLE mode

1: The watchdog continues to count in IDLE mode

#### WDT\_PS[2:0]: Watchdog timer clock frequency division factor

WDT_PS[2:0]	Frequency division factor	12M overflow time at main frequency	20M overflow time at main frequency
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 milliseconds	
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 formula for calculating the watchdog overflow time is as follows:

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

### 8.1.2 IAP Control Register (IAP\_CONTR)

Symbol address B7	B6	B5	B4	B3	B2 B1	B0
IAP CONTR C7H	IAPEN	SWBS	SWRST	CMD FAIL	-	-

#### SWBS: Software Reset Boot Selection

0: Execute code from user program area after software reset. The data in the user data area remains unchanged.

1: Execute code from the system ISP area after software reset. The data in the user data area will be initialized.

#### SWRST: Software Reset Trigger Bit

0: No effect on the microcontroller

1: trigger software reset

### 8.1.3 Reset Configuration Register (RSTCFG)

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

#### ENLVR: Low Voltage Reset Control Bit

0: Disable low voltage reset. A low voltage interrupt is generated when the system detects a low voltage event

1: Enable low voltage reset. Automatic reset when the system detects a low voltage event

#### P54RST: RST pin function selection

0: RST pin is used as 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)

Symbol	address	B7		B6	B5	B4	B3	B2	B1	B0
RSTFLAG	7EFEE99H		-			LVDRSTF	WDTRSTF	SWRSTF	ROMOVF	EXRSTF

LVDRSTF: LVD low voltage reset flag

read 0: meaningless

Read 1: The current reset is triggered by the LVD low voltage reset (power-on reset default value is 1)

Write 0: no effect

Write 1: Clear the LVDRST flag

WDTRSTF: Watchdog Reset Flag

Read 0: meaningless (power-on reset default value is 0)

Read 1: The current reset was triggered by a watchdog overflow

Write 0: no effect

Write 1: Clear the WDTRST flag

SWRSTF: Soft reset flag

read 0: meaningless

Read 1: The current reset is triggered by software writing SWRST (IAP\_CONTR.5) (the default value for user program reset is 1)

Write 0: no effect

Write 1: Clear the SWRST flag

ROMOVF: Code area overflow flag

Read 0: meaningless (power-on reset default value is 0)

Read 1: The current reset is triggered by a code area overflow caused by the CPU executing code to a non-program area

Write 0: no effect

Write 1: Clear the ROMOV flag

EXRSTF: External reset flag

Read 0: meaningless (power-on reset default value is 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 EXRST flag

Instructions for soft reset of user program to system area for USB-ISP download:

When power on, P3.2 needs to be grounded at the same time to enter the USB-ISP download mode. In order to facilitate the user to control the ISP download independently, the following are especially added:

When the user program is soft reset to the system area, the USB-ISP download can be performed without P3.2 being grounded. The judgment method of this function is to

After entering the ISP, judge whether the SWRSTF register bit is 1. If it is 1, it means that the user soft resets to the system area, and P3.2 does not need to be grounded.

Otherwise, P3.2 needs to be grounded. If the user needs software soft reset to the system area or key reset or watchdog reset, it needs to be reset.

For USB-ISP download, keep the SWRSTF register bit as 1, otherwise please write the SWRSTF register bit when the user code is initialized

1, to clear SWRSTF.

## 8.1.5 Reset Control Register (RSTCRx)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
RSTCR0	7EFE9AH	-	-	-	-	RSTM34	TM2	-	RSTM01
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

RSTMn: TIMER0/1/2/3/4 reset control bits

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 reset control bit

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 Drive Reset Control Bit

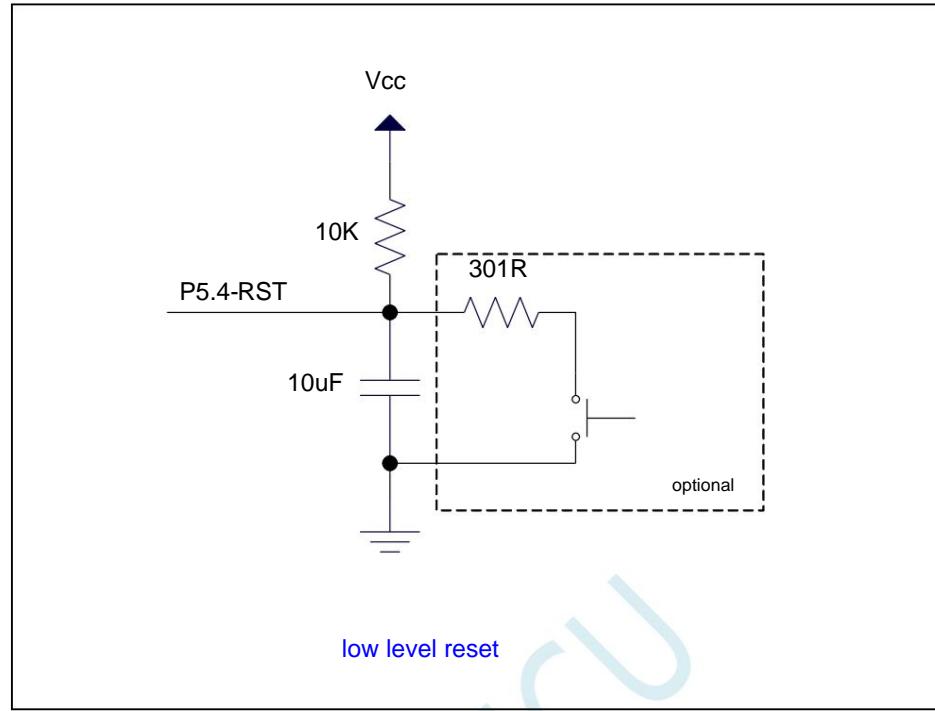
RSTDMA: DMA reset control bit

RSTFPU: FPU reset control bit

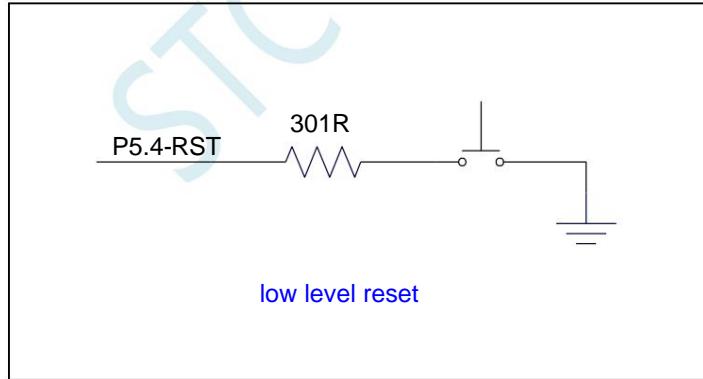
RSTMDU: MDU32 reset control bit

Write 1: reset the corresponding peripheral module module, which needs to be cleared by software

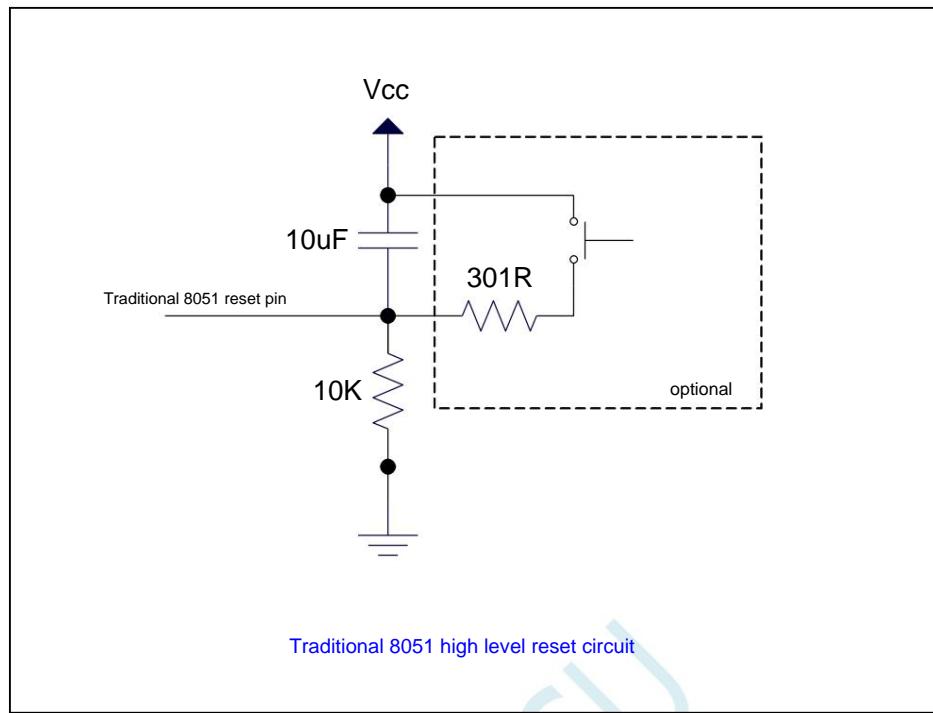
### 8.1.6 Low-level power-on reset reference circuit (generally not required)



### 8.1.7 Low-level key reset reference circuit



### 8.1.8 Traditional 8051 high-level power-on reset reference circuit



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

## 8.2 System Power Management

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
PCON	Power Control Register	87H	\$MOD	\$MDO		LVDF	POF	GF1	GF0	PD	IDL 0011,0000

### 8.2.1 Power Control Register (PCON)

Symbol	address	B7		B6	B5	B4	B3	B2	B1		B0
PCON	87H		SMOD	\$MDO	LVDF	POF	GF1	GF0	PD		IDL

LVDF: Low voltage detection flag. When the system detects a low-voltage event, the hardware automatically sets this bit to 1 and issues an interrupt request 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, and the CPU and all peripherals stop working. It is automatically cleared by hardware after wake-up.

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

Changeless)

IDL: IDLE (idle) mode control bit

0: no effect

1: The microcontroller enters IDLE mode, only the CPU stops working, and other peripherals are still running. Automatically reset by hardware after wake-up

## 8.3 Power-down wake-up timer

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

### 8.3.1 Power-down wake-up timer count registers (WKTCI, WKTCH)

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

WKTEN: Enable control bit for power-down wake-up timer

- 0: Disable power-down wake-up timer
- 1: Enable power-down wake-up timer

If the built-in special timer for power-down wake-up of the STC32G series microcontroller is enabled (by software setting the WKTEN in the WKTCH register Bit 1), when the MCU enters the power-down mode/stop mode, the dedicated timer for wake-up from power-down starts to count, when the count value matches the user-set value When the values are equal, the dedicated timer for power-down wake-up wakes up the MCU. After the MCU wakes up, the program enters the power-down mode from the last time the MCU was set. The next statement of the statement begins to execute down. After power-off wake-up, you can get the MCU status by reading the contents of WKTCH and WKTCI. Sleep time in power-down mode.

Please note here: the value written by the user in the register {WKTCH[6:0], WKTCI[7:0]} must be 1 less than the actual count value. as user To count 10 times, write 9 into registers {WKTCH[6:0], WKTCI[7:0]}. Likewise, if the user needs to count 32767 times, then 7FFEH (ie 32766) should be written to {WKTCH[6:0], WKTCI[7:0]}. (Count value **0** and count value **32767** are reserved internally, user cannot use)

The internal power-down wake-up timer has its own internal clock, and the time that the power-down wake-up timer counts once is determined by the clock. The clock frequency of the internal power-down wake-up timer is about 32KHz, of course, the error is large. The user can read the RAM area F8H and F9H by reading the (F8H stores the high byte of the frequency, F9H stores the low byte) to obtain the clock frequency recorded by the internal power-down wake-up timer when it leaves the factory Rate.

The calculation formula of the count time of the special timer for power-down wake-up is as follows: (Fwt is obtained from the RAM area F8H and F9H Clock frequency of internal power-down wake-up dedicated timer)

$$\text{Power-down wake-up timer timing} = \frac{106 \times 16 \times \text{count times}}{\text{Fwt}} \text{ (microseconds)}$$

Assuming Fwt=32KHz, there are:

{WKTCH[6:0], WKTCI[7:0]} Power-down wake-up dedicated timer count time	
0 (reserved internally)	
1	106÷32K×16×(1+1) ≈ 1 ms
9	106÷32K×16×(1+9) ≈ 5 ms
99	106÷32K×16×(1+99) ≈ 50 ms
999	106÷32K×16×(1+999) ≈ 0.5 seconds
4095	106÷32K×16×(1+4095) ≈ 2 seconds

32766	106÷32K×16x(1+32766) ÷ 16 seconds
32767 (reserved internally)	

STCMCU

## 8.4 Example program

### 8.4.1 Watchdog Timer Application

---

//The test frequency is **11.0592MHz**

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software

void main()
{
    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to      CPU The speed of executing the program is set to the fastest

    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      0.5s
    WDT_CONTR = 0x24;                                     //about enable watchdog overflow time   1s
// WDT_CONTR = 0x27;                                     //watchdog overflow time is about test  8s
    P32 = 0;                                            //Clear watchdog otherwise system reset

    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 Scan Module
        MotorDriver();                                    //Motor drive module
    }
}

```

---

### 8.4.2 Soft reset to achieve custom download

---

//The test frequency is **11.0592MHz**

```
//#include "stc8h.h"
```

```

#include "stc32g.h"                                     // For header files, see Download Software

void main()
{
    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to      CPU The speed of executing the program is set to the fastest

    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;                           //check to P3.2 and P3.3 at the same time for 0 reset to ISP
        }
    }
}

```

#### 8.4.3 Low voltage detection

---

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software

#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; //Enable XFR
CKCON = 0x00; //access to set external data bus speed to fastest
WTST = 0x00; //Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

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; //Low-voltage reset when enabled does not generate low-LVD interrupt
RSTCFG = LVD3V0; // voltage interrupt when enabled
ELVD = 1; //enable LVD
EA = 1;

while (1);
}

```

#### 8.4.4 Power saving mode

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void INT0_Isr() interrupt 0
{
    P34 = ~P34; //test port
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

    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; //EnableINT0 interrupt for wake-up MCU
EA = 1;
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_idl = 1; //MCU enter IDLE
// PD = 1; //MCU enter power-down mode
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
P35 = 0;

while (1);
}

```

#### 8.4.5 Use INT0/INT1/INT2/INT3/INT4 pin interrupt to wake up power saving mode

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#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      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                              //assign to      CPU The speed of executing the program is set to the fastest

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

    IT0 = 0;                                //enable INT0 Rising and falling edge interrupts
    // IT0 = 1;                                //enable INT0 falling edge interrupt
    EX0 = 1;                                //enable INT0 interrupt

    IT1 = 0;                                //enable INT1 Rising and falling edge interrupts
    // IT1 = 1;                                //enable INT1 falling edge interrupt
    EX1 = 1;                                //enable INT1 interrupt

    EX2 = 1;                                //enable INT2 falling edge interrupt
    EX3 = 1;                                //enable INT3 falling edge interrupt
    EX4 = 1;                                //enable INT4 falling edge interrupt

    EA = 1;

    PD = 1;                                //MCU enter power-down mode
    _nop_();                           //This statement will be executed first after MCU is woken up
                                         //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 power saving mode

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                         //For header files, see Download Software
#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 XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                              //Set the program code to wait for parameters,
                                              //assign to CPU The speed of executing the program is set to the fastest

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

    TMOD = 0x00;                               //65536-11.0592M/ 12/1000
    TL0 = 0x66;                                //start timer
    TH0 = 0xfc;                                //Enable timer interrupt

    TR0 = 1;                                   //start timer
    ET0 = 1;                                   //Enable timer interrupt

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

    T2L = 0x66;                               //65536-11.0592M/ 12/1000
    T2H = 0xfc;                                //start timer
    T2R = 1;                                   
}

```

```

ET2 = 1; //Enable timer interrupt

T3L = 0x66; //65536-11.0592M/ 12/1000
T3H = 0xfc;
T3R = 1; //start timer
ET3 = 1; //Enable timer interrupt

T4L = 0x66; //65536-11.0592M/ 12/1000
T4H = 0xfc;
T4R = 1; //start timer
ET4 = 1; //Enable timer interrupt

EA = 1;

PD = 1; //MCU enter power-down mode
_nop_(); //The interrupt service routine will not be entered immediately after      ,
_nop_(); //power-off wake-up, but will not enter the interrupt service routine until the timer overflows
_nop_();
_nop_();
_nop_();

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

```

#### 8.4.7 Use RxD/RxD2/RxD3/RxD4 pin interrupt to wake up power saving mode

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#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      XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
}

```

**WTST = 0x00;** // Set the program code to wait for parameters,  
**//assign to CPU** The speed of executing the program is set to the fastest

**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/ Wake-up on falling edge  
**// S1\_S1 = 1; S1\_S0 = 0;** P3.6 // RXD\_3/Wake-up on falling edge  
**// S1\_S1 = 1; S1\_S0 = 1;** P1.6 // 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

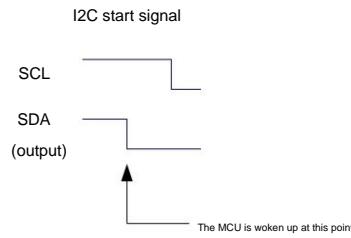
**S4\_S = 0;** // RXD4/P0.2 Wake-up on falling edge  
**// S4\_S = 1;** // RXD4\_2/P5.2 Wake-up on falling edge

**ES = 1;** // Enable serial interrupt  
**ES2 = 1;** // Enable serial interrupt  
**ES3 = 1;** // Enable serial interrupt  
**ES4 = 1;** // Enable serial interrupt  
**EA = 1;**

**PD = 1;** // MCU enter power-down mode  
**\_nop();** // Will not enter the interrupt service routine after power-down wake-up  
**\_nop();**  
**\_nop();**  
**\_nop();**

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

### 8.4.8 Use the SDA pin of I2C to wake up the MCU power saving mode




---

// The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void i2c_isr() interrupt 24
{
    I2CSLST &= ~0x40;
}

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    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;
    //I2C_S1 = 0; I2C_S0 = 1;
    //I2C_S1 = 1; I2C_S0 = 1;

    I2CCFG = 0x80;
    I2CSLCR = 0x40;
    EA = 1;

    PD = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();

    while (1)

```

STCMCU  
 Enable XFR  
 access to set external data bus speed to fastest  
 Set the program code to wait for parameters,  
 //Assign to 0      CPU    The speed of executing the program is set to the fastest

//SDA/P1.4 // Wake-up on falling edge  
 SDA\_2/P2.4 // Wake-up on falling edge  
 SDA\_4/P3.3 // Wake-up on falling edge

enable I2C Slave mode of the module  
 //enable start signal interrupt

//MCU // enter power-down mode  
 Will not enter the interrupt service routine after power-down wake-up

```

{
    P11 = ~P11;
}

```

---

#### 8.4.9 Using the Power-down Wake-up Timer to Wake up from Power-Saving Mode

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1;                                     //Enable      XFR
    CKCON = 0x00;                                   //access to set external data bus speed to fastest
    WTST = 0x00;                                    //Set the program code to wait for parameters,
                                                    //assign to CPU The speed of executing the program is set to the fastest

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

    WKTCI = 0xff;                                  // Set the power-down wake-up clock to approximately 1 seconds
    WKTCH = 0x87;

    while (1)
    {
        _nop_();
        _nop_();
        PD = 1;                                      //MCU enter power-down mode
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        P11 = ~P11;
    }
}

```

---

#### 8.4.10 LVD interrupt wake-up power saving mode, it is recommended to use power-down wake-up timer together

It is not recommended to start the LVD and the comparator in the power saving mode of the clock stop vibration, otherwise the hardware system will automatically start the internal 1.19V high precision

Reference source, this high-precision reference source has corresponding anti-temperature drift and adjustment circuits, which will increase the power consumption of about 300uA, and the MCU input After entering the clock stop mode, the current only consumes about 0.4uA when the working voltage is 3.3V, so it is not recommended to open the LVD and LVD when entering the clock stop mode. Comparators. If you really need to use it, it is recommended to enable the power-down wake-up timer. The power-down wake-up timer will only increase the power consumption by about 1.4uA. The power consumption is generally acceptable for the system. Let the power-down wake-up timer wake up the MCU every 5 seconds. After wake-up, LVD, comparator, ADC can be used Detect the external battery voltage, the detection work takes about 1mS before entering the clock stop/power saving mode, the average current added is less than 1uA, Then the overall power consumption is about 2.8uA (0.4uA + 1.4uA + 1uA).

---

//The test frequency is **11.0592MHz**

```

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#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 XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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; //The interrupt flag needs to be cleared at power-on
    RSTCFG = LVD3V0; //Set voltage to enable 3.0V
    ELVD = 1; //interrupt LVD
    EA = 1;

    PD = 1; //MCU enter power-down mode
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();

    while (1)

```

```

{
    P11 = ~P11;
}

```

---

#### 8.4.11 Comparator interrupt wake-up power saving mode, it is recommended to use power-down wake-up timer

It is not recommended to start the LVD and the comparator in the power saving mode of the clock stop vibration, otherwise the hardware system will automatically start the internal 1.19V high precision reference source, this high-precision reference source has corresponding anti-temperature drift and adjustment circuits, which will increase the power consumption of about 300uA, and the MCU input After entering the clock stop mode, the current only consumes about 0.4uA when the working voltage is 3.3V, so it is not recommended to open the LVD and LVD when entering the clock stop mode. Comparators. If you really need to use it, it is recommended to enable the power-down wake-up timer. The power-down wake-up timer will only increase the power consumption by about 1.4uA. The power consumption is generally acceptable for the system. Let the power-down wake-up timer wake up the MCU every 5 seconds. After wake-up, LVD, comparator, ADC can be used Detect the external battery voltage, the detection work takes about 1mS before entering the clock stop/power saving mode, the average current added is less than 1uA, Then the overall power consumption is about 2.8uA (0.4uA + 1.4uA + 1uA).

---

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h"
#include "intrins.h"

void CMP_Isr() interrupt
21 {
    CMPIF = 0;
    P10 = !P10;
}

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    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;
    PIE = 1; NIE = 1;
    CMPOE = 1;
    EA = 1;

    PD = 1; //MCU
}

// For header files, see Download Software
// clear interrupt flag
// test port
// Enable XFR
// access to set external data bus speed to fastest
// Set the program code to wait for parameters,
// assign to CPU The speed of executing the program is set to the fastest
// enter power-down mode

```

```

_nop_();
_nop_();
_nop_();
_nop_();

while (1)
{
    P11 = ~P11;
}

}

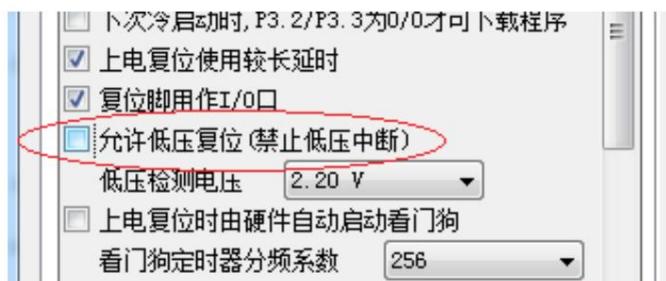
```

---

#### 8.4.12 Using the LVD function to detect the operating voltage (battery voltage)

If you need to use the LVD function to detect the battery voltage, you need to remove the low-voltage reset function during ISP download, as shown in the figure "Allow low-voltage reset"

The tick option of the hardware option of "bit (disable low-voltage interrupt)" needs to be removed




---

//The test frequency is 11.0592MHz

```

##include "stc8h.h"
#include "stc32g.h"
#include "intrins.h" //For header files, see Download Software

#define FOSC 11059200UL //Define as an unsigned long integer to avoid calculation 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 the fastest. Set
WTST = 0x00;                                // the program code to wait for the parameter, and
                                                // assign it to O CPU The speed of executing the program is set to the fastest

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

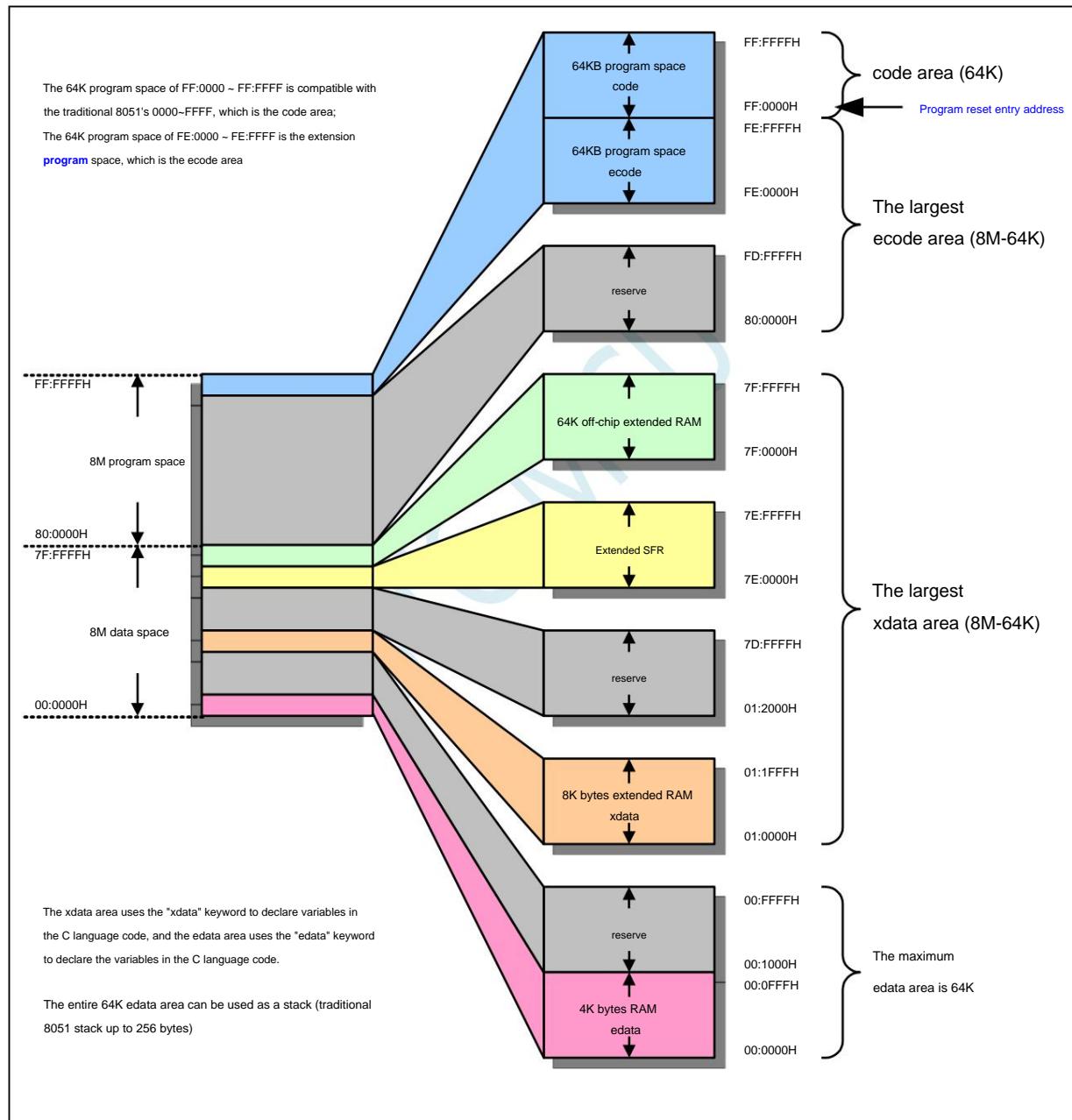
}

STCMCU

## 9 memories (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 to access external program memory, all program memory of the microcontroller is on-chip Flash memory and cannot access external program memory.

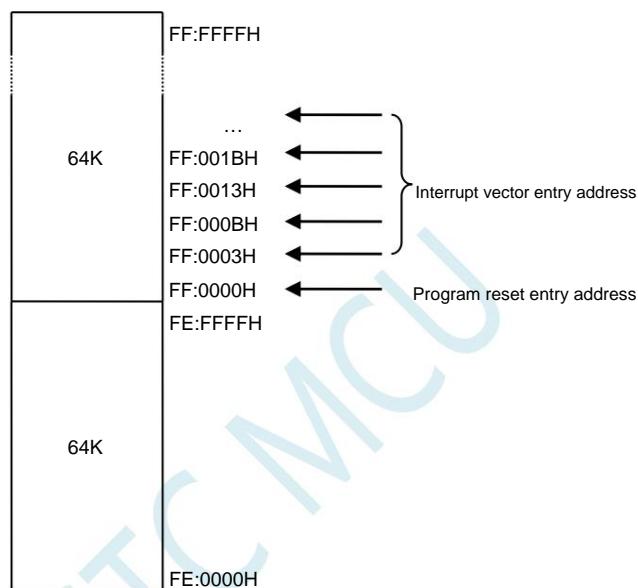
STC32G series microcontrollers integrate a large-capacity data memory inside. The data memory inside the STC32G series microcontroller is in the Both logically and logically are divided into two address spaces: internal RAM (edata) and internal extended RAM (xdata).



## 9.1 Program memory

The program memory is used to store information such as user programs, fixed data, and tables.

MCU series	Flash program memory (ROM)	address range
STC32G12K128 series	128K bytes	FE:0000H~FF:FFFFH
STC32G8K64 series	64K bytes	FF:0000H~FF:FFFFH
STC32F12K60 series	60K bytes	FF:0000H~FF:EFFFH



(STC32G series and traditional 8051 interrupt entry address comparison)

	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 program counter (PC) is FF:0000H, and program execution starts from FF:0000H. Another interrupt service routine

The entry address of the (aka interrupt vector) is also located in the program memory location. In program memory, each interrupt has a fixed entry address,

When the interrupt occurs and is responded to, the microcontroller will automatically jump to the corresponding interrupt entry address to execute the program. of external interrupt 0 (INT0)

The entry address of the interrupt service routine is FF:0003H, the entry address of the timer/counter 0 (TIMER0) interrupt service routine is FF:000BH, and the external

The entry address of the interrupt service routine of interrupt 1 (INT1) is FF:0013H, and the entry of the interrupt service routine of timer/counter 1 (TIMER1)

The address is FF:001BH and so on. For more entry addresses (interrupt vectors) of interrupt service routines, please refer to the Interrupt Introduction chapter.

Since the interval between adjacent interrupt entry addresses is only 8 bytes, it is generally impossible to save the complete interrupt service routine.

The address area of the interrupt response stores an unconditional transfer instruction, pointing to the space where the interrupt service routine is actually stored for execution.

STC32G series microcontrollers all contain Flash data memory (EEPROM). Read/write data in bytes, in 512 bytes

Erasing is performed in units of pages, and it can be programmed and erased more than 100,000 times online, which improves the flexibility and convenience of use.

## 9.1.1 Program Read Wait Control Register (WTST)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
WTST	E9H						WTST[7:0]

WTST[7:0]: CPU read program memory wait time control

WTST[7:0] wait clock number	
0	0 clocks
1	1 clock
...	...
7	7 clocks
8	reserve
...	reserve
255	reserve

The number of actual execution clocks per instruction = the number of instruction clocks + the number of waiting clocks in the program memory

Note: For STC32G12K128 series A-version chips, the power-on default value of the WTST register is 7. When the user's operating frequency is below 35MHz,

It is recommended to modify it to 0, which can speed up the speed of the CPU running the program.

## 9.2 Data memory (32-bit access, 16-bit access, 8-bit access)

The edata area of STC32G can perform single-clock read and write access to 32-BIT/16-BIT/8-BIT data, and the xdata area can 16-BIT/8-BIT data read and write access. The current maximum storage depth of the SRAM in the edata area has been designed to be 64K bytes; the xdata area The maximum storage depth of the SRAM is 8M bytes.

The special function register 32-BIT SFR32 (such as ADC\_DATA32) will be added in the future, such as mapping the logical address of SFR32 in edata area, you can support 32-BIT/16-BIT/8-BIT access to the newly added special function registers;

The special function register 16-BIT SFR16 (such as ADC\_DATA16) will be added in the future, such as mapping the logical address of SFR16 in xdata area, you can support 16-BIT/8-BIT access to the newly added special function registers

The RAM integrated in the STC32G series microcontroller can be used to store the intermediate results and process data of program execution.

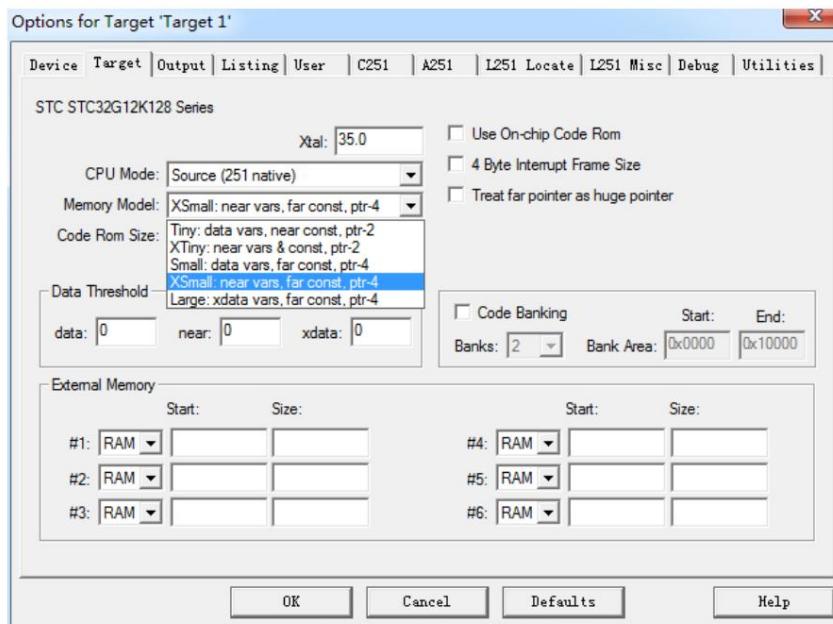
MCU series	Internal RAM (edata)	Internal expansion RAM (xdata)
STC32G12K128 series	4K bytes	8K bytes
STC32G8K64 series	2K bytes	6K bytes
STC32F12K60 series	8K bytes	4K bytes

EDATA: single clock access, fast access speed, can be used as stack, high cost;

XDATA: Access requires 2-3 clocks, the access speed is slow, the addressing space can reach 8M, and the cost is low.

## 9.2.1 Keil Options Memory Model Settings

For the memory mode of the STC32G series project, there are 5 modes in the Keil environment as shown in the following figure:



The various modes are compared in the following table:

Memory Model	Default variable type (data memory)	Default constant type (program memory)	Default pointer variable	pointer access range
Tiny mode	data	near	2 bytes	00:0000 to 00:FFFF
XTiny mode	edata	near	2 bytes	00:0000 to 00:FFFF
Small mode	data	far	4 bytes	00:0000 to FF:FFFF
<b>XSmall mode</b>	<b>edata</b>	<b>far</b>	<b>4 bytes 00:0000 to FF:FFFF</b>	
Large mode	xdata	far	4 bytes	00:0000 to FF:FFFF

Since the program logic address of STC32G is FE:0000H~FF:FFFFH, it needs to use a 24-bit address line to access it correctly.

The amount type (program memory type) must use the "far" type, and the default pointer variable must be 4 bytes.

Therefore, it is not recommended to use "Tiny" and "XTiny" modes. It is recommended to use "XSmall" mode, which defines variables internally by default.

RAM (edata), single clock access, fast access speed, and STC32G12K128 series chips have 12K edata that can be used; also "Small"

Mode, this mode defines variables in internal RAM (data) by default, single-clock access, fast access speed, (data is only 128 bytes by default, when

When the user's RAM requirement exceeds 128 bytes, the Keil compiler will report an error, and the user needs to switch the storage mode to XSmall); not recommended

Use "Large" mode, although this mode can also correctly access the full 16M address space of STC32G, but "Large" mode defaults to variable

Defined in the internal extended RAM (xdata), access requires 2~3 clocks, and the access speed is slow

**C language variable declaration suggestions for STC32G series microcontrollers :**

1. When the demand for user variables is small, it is recommended not to use keywords such as "edata, xdata" to declare variables, but to use the following methods directly

Declare the variable:

```
char      bCounter = 1;           //declare byte variable  
int       wCounter = 100; //declare a double-byte variable  
long      dwCounter = 0x1234; //declare a 4-byte variable
```

Then in the project options set "Memory Model" to "XSmall" to let the compiler automatically assign the declared variables to edata area

2. When the demand for user variables is close to or exceeds the size of (edata capacity of single-chip microcomputer - 1K), it is recommended to use "xdata" for the excess part.

The keys are forcibly assigned to the xdata area as follows:

```
int xdata pBuffer = 5;           //Use the xdata keyword to force allocation to the xdata area
```

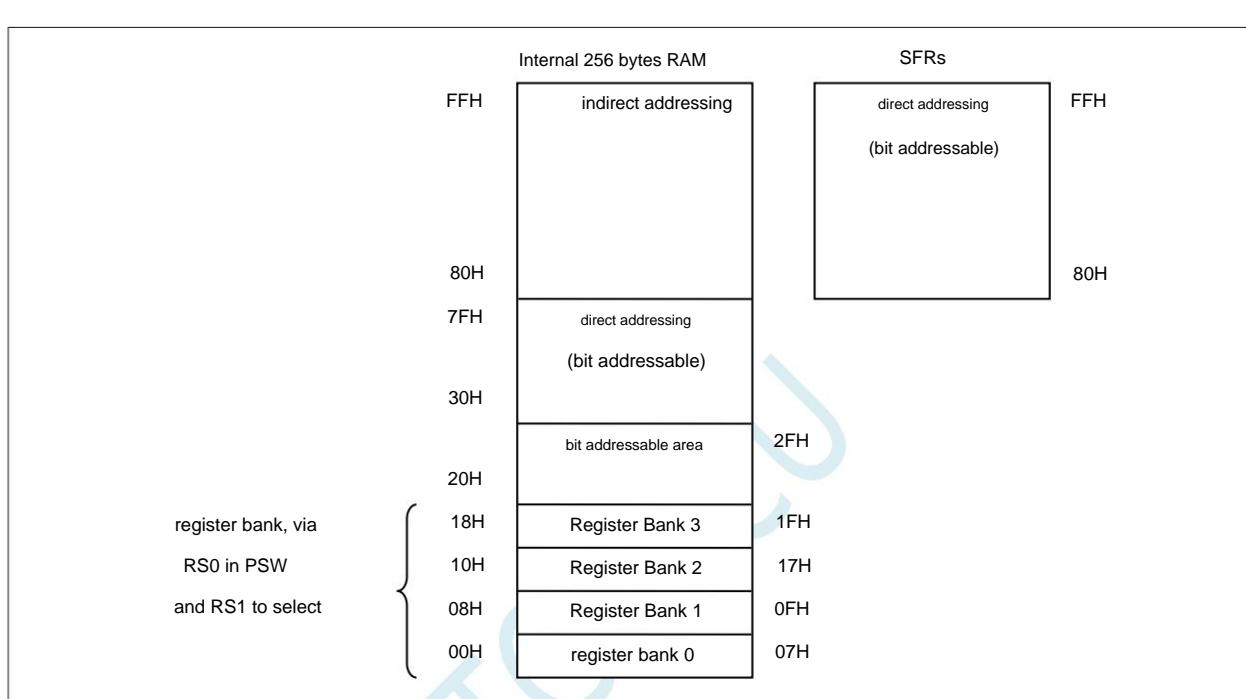
## 9.2.2 Internal edata-RAM (C language declaration keyword is edata)

The internal edata-RAM has a total of 4K bytes, and the 256 bytes at the low end of the 4K bytes are completely compatible with the 256 bytes DATA of the 8051, which can be divided into 2 parts: low 128 bytes RAM and high 128 bytes RAM. The lower 128 bytes of data memory are compatible with legacy 8051, both directly addressable.

Indirect addressing is also possible. The upper 128 bytes of RAM (extended in the 8052) can only be addressed indirectly. Special Function Register Distribution

In the 80H~FFH area, only direct addressing is possible.

The structure of the low-end 256-byte RAM is shown in the following figure:



The stack of STC32G12K128 is placed in EDATA, the theoretical depth can reach 64K in design, and 4K Bytes is actually placed; the stack of STC32G12K128 Ordinarily extended XDATA, the theoretical depth can reach (8M-64K) in design, and 8K Bytes is actually placed. So the SRAM of STC32G12K128 Total is 12K (4K edata + 8K xdata).

By declaring the variable in the **EDATA** area in the C language code , you can achieve 32-bit/16-bit/8-bit read and write operations with a single clock

```
char    edata    bCounter;           //Declare a byte variable in the EDATA area (single clock for 8-bit read and write operations)
int     edata    wCounter;          //Declare a double-byte variable in the EDATA area (single clock for 16-bit read and write operations)
long    edata    dwCounter;         //declare a 4-byte variable in the EDATA area (single clock for 32-bit read and write operations)
```

8-bit/16-bit read and write operations can be achieved by declaring variables in the **XDATA** area in the C language code

```
char    xdata    bCounter;          //declare byte variable in XDATA area (3/2 clock for 8-bit read/write)
int     xdata    wCounter;          //Declare a double-byte variable in the XDATA area (16-bit read/write at 3/2 clock)
```

### 9.2.3 Program Status Register (PSW)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0	
PSW	D0H	CY	AC	F0	<b>RS1</b>	<b>RS0</b>	OV	-	P

CY: carry flag

AC: Auxiliary carry flag

F0: Generic flag

OV: overflow flag

P: Even parity flag for ACC

RS1, RS0: Working register selection bits

RS1 RS0 working 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 16 byte units from 20H ~ 2FH. 20H~2FH units can be accessed by bytes like ordinary RAM units.

You can also access any bit in the unit individually, with a total of 128 bits, and the corresponding logical bit address range is 00H~7FH. The bit address range is 00H~7FH, the address of the lower 128 bytes of the internal RAM is also 00H~7FH. From the outside, the addresses of the two are the same.

A qualitative difference; a bit address points to a bit, while a byte address points to a byte unit, which are distinguished by different instructions in the program.

### 9.2.4 Program Status Register 1 (PSW1)

Symbol 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 carry flag (same as AC in PSW)

N: Negative flag bit of the calculation result, N is 1 when the highest bit of the calculation result is 1, otherwise it is 0

RS1, RS0: Working register selection bits (same as RS0, RS1 in PSW)

OV: overflow flag (same as OV in PSW)

Z: zero flag bit of calculation result, Z is 1 when the operation result is 0, otherwise it is 0

## 9.2.5 Internal xdata-RAM (C language declaration keyword is xdata)

STC32G series microcontrollers have integrated extended RAM on-chip. The method of accessing the internal extended RAM and the traditional 8051 microcontroller accessing the external RAM is the same. The method for expanding RAM is the same, but it does not affect the signals on ports such as P0, P2, and ports such as RD, WR, and ALE.

In assembly language, the internal extended RAM is accessed through the MOVX instruction,

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

In the C language, the storage type can be declared using the xdata keyword. like:

```
unsigned char xdata i;
```

(It is strongly recommended not to declare variables with the pdata keyword)

Whether the internal extended RAM of the microcontroller can be accessed is controlled by the EXTRAM bit in the auxiliary register AUXR.

## 9.2.6 Auxiliary Register (AUXR)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R T2_C/T T2x12	<b>EXTRAM</b>	S1BRT		

EXTRAM: Extended RAM Access Control

0: Disable access to external extended RAM.

1: Enable access to external extended RAM.

## 9.2.7 External xdata-RAM

STC32G series microcontrollers have the ability to expand 64KB external data memory. During access to external data memory, the WR/RD/ALE signal must be efficient. The special function register BUS\_SPEED of the STC32G series microcontroller to control the speed of the external 64K byte data bus is described as follows:

## 9.2.8 Bus Speed Control Register (BUS\_SPEED)

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

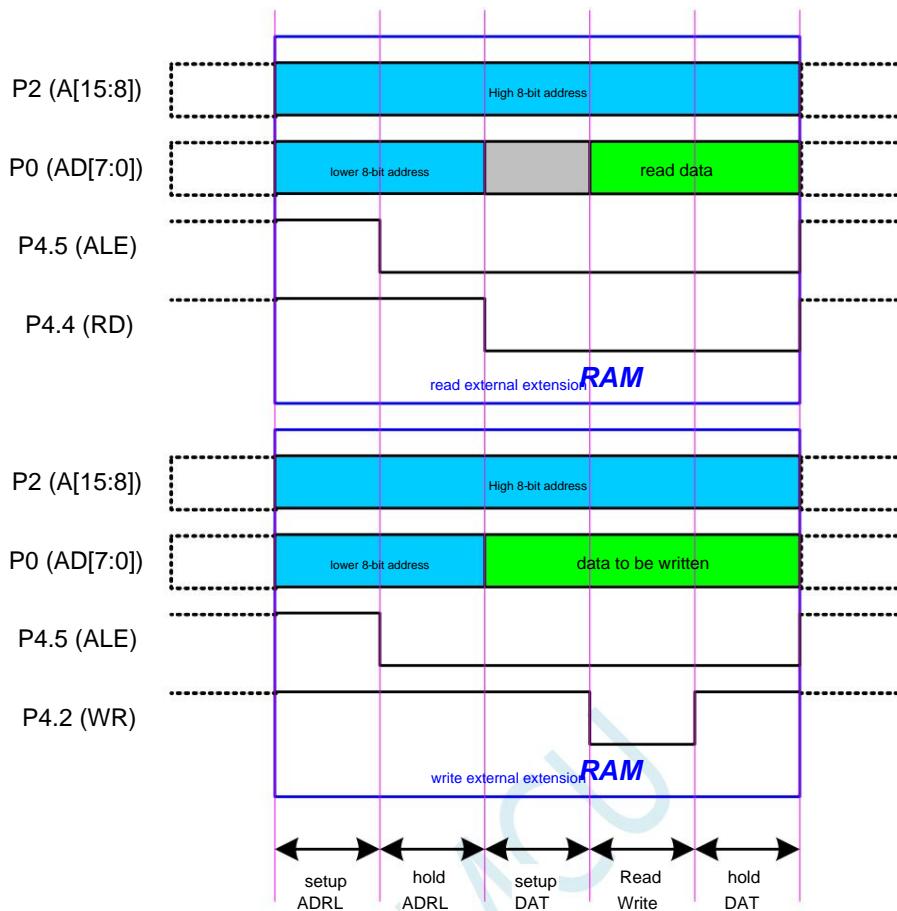
RW\_S[1:0]: RD/RW control line selection bits

00: P4.4 is RD, P4.2 is WR

x1: reserved

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

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



### 9.2.9 External Data Bus Clock Control Register (CKCON)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
CKCON	EAH	-	-	-	T1M	TOM		CKCON[2:0]

T1M: reserved

TOM: reserved

CKCON[2:0]: External data bus clock control register (power-on reset value is 7, it is recommended to set it to 0)

CKCON[2:0] wait clock number	
0	0 clocks
1	1 clock
...	...
7	7 clocks

## 9.2.10 Bit-addressable data memory in STC32G series microcontrollers

The internal bit-addressable data memory of STC32G series MCU includes two parts: the address range of the first part is the DATA area. 20H~7FH, the address range of the second part is the special function register SFR: 80H~FFH, and the two parts of the area are described below.

### DATA area

The 00H~1FH of the DATA area is the mapping area of the registers R0~R7, which is not bit addressable. The remaining 20H~7FH, a total of 96 bytes, each byte is all bit addressable.

The definition method of assembly code:

BVAR1	BIT	20H.0	;Define bit variable BVAR1
BVAR2	BIT	50H.1	;Define bit variable BVAR2
BVAR3	BIT	70H.2	;Define bit variable BVAR3

How to use assembly code:

SETB	BVAR1	;BVAR1 = 1
CLR	BVAR2	;BVAR2 = 0
CPL	BVAR3	;BVAR3 = ~BVAR3

C language code definition method:

char ebdata	flag;	//Define a byte variable in the bit addressable area
sbit bVar1 =	flag^0;	//declare the bit variable bVar1 in flag using sbit
sbit bVar2 =	flag^7;	//declare the bit variable bVar2 in flag using sbit
bit ebdata	bVar3;	//Use bit ebdata to directly declare the bit variable bVar3

How to use the C language code:

bVar1 = 1;	// set the bit variable to 1
bVar2 = 0;	// clear the bit variable to 0
bVar3 = ~bVar3;	//invert the bit variable

### Special Function Register (SFR) Area

80H~FFH of all SFR area, a total of 128 bytes, each byte can be bit addressable.

The definition method of assembly code:

BVAR1	BIT	80H.0	;Define bit variable BVAR1
BVAR2	BIT	87H.1	;Define bit variable BVAR2
BVAR3	BIT	FFH.2	;Define bit variable BVAR3

How to use assembly code:

SETB	BVAR1	;BVAR1 = 1
CLR	BVAR2	;BVAR2 = 0
CPL	BVAR3	;BVAR3 = ~BVAR3

C language code definition method:

sfr P0	=	0x80;	//Define SFR
sbit P00	=	P0^0;	//declare SFR bits in SFR using sbit
sfr PCON =		0x87;	//Define SFR
sbit PD	=	PCON^1;	//declare SFR bits in SFR using sbit
sfr ACC =		0xE0;	//Define SFR
sbit ACC7 =		ACC^7;	//declare SFR bits in SFR using sbit

How to use the C language code:

PD = 0;	// set the bit variable to 1
---------	------------------------------

```
P00 = 1;                                // clear the bit variable to 0  
ACC7 = ~ACC7                            //invert the bit variable
```

Note: Bit variables do not support the definition of array and pointer types

STCMCU

## 9.2.11 Instructions for using extended SFR enable register **EAXFR**

The addressing of the memory address of the STC32G series microcontroller is as follows:

Area	address range function	illustrate
data area	00:0000H - 00:FFFFH Data area (edata)	stack area
	01:0000H - 01:FFFFH Internal extended data area (xdata)	
	02:0000H - 7D:FFFFH Data reserved area	
	<b>7E:0000H - 7E:FFFFH Extended SFR area (XFR)</b> 7F:0000H - 7F:FFFFH	Need to enable <b>EAXFR (P_SW2.7)</b> to access
	External data area (xdata) 80:0000H - FD:FFFFH Code reserved area	Need to enable EXTRAM (AUXR.1) to access
code area		
	FE:0000H - FE:FFFFH Extended code area (ecode)	
	FF:0000H - FF:FFFFH Code area (code)	

To access the extended SFR in the XFR area, you need to set EAXFR (P\_SW2.7) to 1 first, because the addressing of the memory address of STC32G

The method is linear addressing, and the address of the XFR area is an independent address area, so **EAXFR** can be registered when the power-on system is initialized.

**Write 1 to the device (for example: **EAXFR = 1;**), keep it as 1 and do not need to modify it, you can access the XFR area normally.**

Note: Since the EAXFR control bit and S2\_S, S3\_S and other register bits share P\_SW2, when setting S2\_S, S3\_S in P\_SW2

, it is recommended to use the bit manipulation statement directly (eg S2\_S = 0; S3\_S = 1;) to avoid direct manipulation of the P\_SW2 register. If the code really needs

To modify P\_SW2 directly, be sure to also use the AND-OR instruction (eg P\_SW2 &= ~0x01; P\_SW2 |= 0x02;).

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

EAXFR: Extended RAM Area Special Function Register (XFR) Access Control Register

0: Disable access to XFR

1: Enable access to XFR.

When you need to access **XFR**, you must first set **EAXFR** to 1 to perform normal reading and writing on **XFR**. It is recommended to initialize at power-on

Set it directly to 1, do not modify it later

## 9.3 Unique ID numbers and important parameters stored in read-only special function registers (CHIPID)

product line	CHIPID
STC32G12K128 series	●
STC32G8K64 series	●
STC32F12K60 series	●

The read-only special function register CHIPID inside the STC32G series microcontroller stores some special parameters related to the chip, including

Including: global unique ID number, frequency of 32K power-down wake-up timer, internal 1.19V reference signal source value and IRC parameters. in the user program

In the sequence, only the content in CHIPID can be read and cannot be modified. Encrypting user programs with data in CHIPID is an official STC recommendation.

recommended optimal solution.

### Related registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID00	Hardware digital ID00	7EFDE0H	Globally unique ID number (0th byte)								nnnn,nnnn
CHIPID01	Hardware digital ID01	7EFDE1H	Globally unique ID number (1st byte)								nnnn,nnnn
CHIPID02	Hardware digital ID02	7EFDE2H	Globally unique ID number (2nd byte)								nnnn,nnnn
CHIPID03	Hardware Digital ID03	7EFDE3H	Globally unique ID number (3rd byte)								nnnn,nnnn
CHIPID04	Hardware Digital ID04	7EFDE4H	Globally unique ID number (4th byte)								nnnn,nnnn
CHIPID05	Hardware Digital ID05	7EFDE5H	Globally unique ID number (5th byte)								nnnn,nnnn
CHIPID06	Hardware Digital ID06	7EFDE6H	Globally Unique ID Number (6th byte)								nnnn,nnnn
CHIPID07	Hardware Digital ID07	7EFDE7H	Internal 1.19V reference signal source (high byte)								nnnn,nnnn
CHIPID08	Hardware digital ID08	7EFDE8H	Internal 1.19V reference signal source (low byte)								nnnn,nnnn
CHIPID09	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	22.1184MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID12	Hardware digital ID12	7EFDECH	24MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID13	Hardware digital ID13	7EFDEDH	27MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID14	Hardware digital ID14	7EFDEEH	30MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID15	Hardware digital ID15	7EFDEFH	33.1776MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID16	Hardware digital ID16	7EFDF0H	35MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID17	Hardware digital ID17	7EFDF1H	36.864MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID18	Hardware digital ID18	7EFDF2H	40MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID19	Hardware digital ID19	7EFDF3H	44.2368MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID20	Hardware digital ID20	7EFDF4H	48MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID21	Hardware digital ID21	7EFDF5H	VRTRIM parameters for the 6M band								nnnn,nnnn
CHIPID22	Hardware digital ID22	7EFDF6H	VRTRIM parameters for the 10M band								nnnn,nnnn
CHIPID23	Hardware digital 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 digital ID25	7EFDF9H	00H	nnnn,nnnn
CHIPID26	Hardware digital ID26	7EFDFAH	User program space end address (high byte)	nnnn,nnnn
CHIPID27	Hardware digital ID27	7EFDFBH	Chip test time (years)	nnnn,nnnn
CHIPID28	Hardware digital ID28	7EFDFCH	Chip test time (month)	nnnn,nnnn
CHIPID29	Hardware digital ID29	7EFDFDH	Chip test time (day)	nnnn,nnnn
CHIPID30	Hardware Digital ID30	7EFDFEH	Chip package form number	nnnn,nnnn
CHIPID31	Hardware digital ID31	7EFDFFFH	5AH	nnnn,nnnn

STCMCU

### 9.3.1 Interpretation of the globally unique ID number of CHIP

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID00	Hardware digital ID00	7EFDE0H	Globally unique ID number (0th byte)								nnnn,nnnn
CHIPID01	Hardware digital ID01	7EFDE1H	Globally unique ID number (1st byte)								nnnn,nnnn
CHIPID02	Hardware digital ID02	7EFDE2H	Globally unique ID number (2nd byte)								nnnn,nnnn
CHIPID03	Hardware Digital ID03	7EFDE3H	Globally unique ID number (3rd byte)								nnnn,nnnn
CHIPID04	Hardware Digital ID04	7EFDE4H	Globally unique ID number (4th byte)								nnnn,nnnn
CHIPID05	Hardware Digital ID05	7EFDE5H	Globally unique ID number (5th byte)								nnnn,nnnn
CHIPID06	Hardware Digital ID06	7EFDE6H	Globally Unique ID Number (6th byte)								nnnn,nnnn

[CHIPID0, CHIPID1]: 16-bit MCU ID, used to distinguish different MCU models (higher order first).

[CHIPID2, CHIPID3]: 16-bit test machine number (high order first).

[CHIPID4, CHIPID5, CHIPID6]: 24-bit test serial number (high order first).

### 9.3.2 Interpretation of the internal reference signal source of CHIP

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID07	Hardware Digital ID07	7EFDE7H	Internal 1.19V reference signal source (high byte)								nnnn,nnnn
CHIPID08	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-order first).

The standard value is 1190 (04A6H), and the unit is mV, which is 1.19V. But the actual chip is due to manufacturing errors. Internal reference signal source

The voltage value is not affected by the working voltage VCC, so the internal reference signal source can be combined with the ADC to calibrate the ADC,

It can also be combined with a comparator to detect operating voltage.

### 9.3.3 Interpretation of the internal 32K IRC oscillation frequency of CHIP

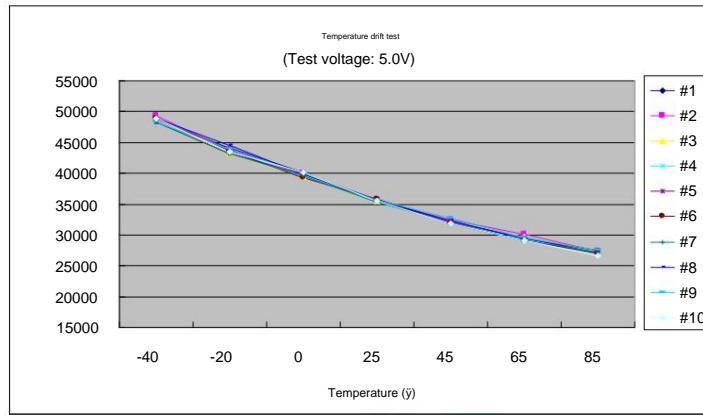
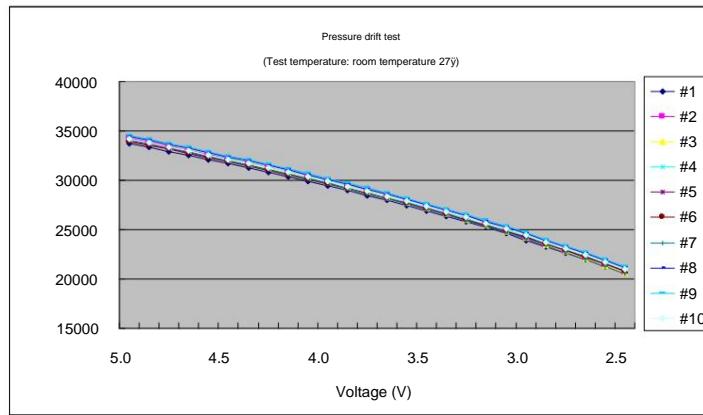
symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID09	Hardware Digital ID09	7EFDE9H	32K Power-down wake-up timer frequency (high byte)								nnnn,nnnn
CHIPID10	Hardware digital ID10	7EFDEAH	32K Power-down wake-up timer frequency (low byte)								nnnn,nnnn

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

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

The pressure drift is relatively large.

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



### 9.3.4 Interpretation of high-precision **IRC** parameters of CHIP

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID11	Hardware digital ID11	7EFDEBH	22.1184MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID12	Hardware digital ID12	7EFDECH	24MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID13	Hardware digital ID13	7EFDEDH	27MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID14	Hardware digital ID14	7EFDEEH	30MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID15	Hardware digital ID15	7EFDEFH	33.1776MHz IRC parameters (27M band)								nnnn,nnnn
CHIPID16	Hardware digital ID16	7EFDF0H	35MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID17	Hardware digital ID17	7EFDF1H	36.864MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID18	Hardware digital ID18	7EFDF2H	40MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID19	Hardware digital ID19	7EFDF3H	44.2368MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID20	Hardware digital ID20	7EFDF4H	48MHz IRC parameters (44M band)								nnnn,nnnn
CHIPID21	Hardware digital ID21	7EFDF5H	VRTRIM parameters for the 6M band								nnnn,nnnn
CHIPID22	Hardware digital ID22	7EFDF6H	VRTRIM parameters for the 10M band								nnnn,nnnn
CHIPID23	Hardware digital ID23	7EFDF7H	VRTRIM parameters for the 27M band								nnnn,nnnn
CHIPID24	Hardware digital ID24	7EFDF8H	VRTRIM parameters for the 44M band								nnnn,nnnn

The STC32G series single-chip microcomputer that supports the CHIPID function, the integrated high-precision IRC is divided into 4 frequency bands, and the reference voltage corresponding to each frequency band is

The voltage value has been calibrated at the factory. When selecting different frequency bands, you only need to fill in the voltage calibration value of the corresponding frequency band into VRTRIM

Register can be. The center frequencies of the 4 frequency bands are 6MHz, 10MHz, 27MHz and 44MHz. Due to manufacturing errors, the center

The frequency may generally have a deviation of  $\pm 5\%$ . In order to obtain the precise user frequency, the frequency can be fine-tuned and calibrated using IRTRIM. Make

When downloading the user program with the download software officially provided by STC, the system will automatically set VRTRIM and VRTRIM according to the frequency set by the user.

IRTRIM register. At the same time, the IRTRIM values of 10 commonly used frequencies and the reference of 4 frequency bands are also preset in CHIPID.

The voltage value calibration value allows the user to dynamically modify the operating frequency during program operation.

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

[CHIPID21 : CHIPID24]: Calibration value of reference voltage value for 4 frequency bands.

When the user modifies the frequency dynamically, he only needs to read and write a certain frequency calibration value in [CHIPID11 : CHIPID20] into the IRTRIM register,

At the same time, according to the frequency band corresponding to the frequency, read out a certain voltage calibration value in [CHIPID21 : CHIPID24] and write it into VRTRIM

Register can be. For detailed operations, please refer to the sample programs in the following chapters.

### 9.3.5 Interpretation of test time parameters of CHIP

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID27	Hardware digital ID27	7EFDFBH	Chip test time (years)								nnnn,nnnn
CHIPID28	Hardware digital ID28	7EFDFCH	Chip test time (month)								nnnn,nnnn
CHIPID29	Hardware digital ID29	7EFDFDH	Chip test time (day)								nnnn,nnnn

The year, month, and day parameters of the test time are all BCD codes. (For example: CHIPID27=0x21, CHIPID28=0x11, CHIPID29=0x18,

The production test date of the target chip is November 18, 2021)

### 9.3.6 Interpretation of Chip Package Type Number of CHIP

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
CHIPID30	Hardware Digital ID30	7EFDFEH	Chip package form number								nnnn,nnnn

Package Number	Package Type	Package Number	Package Type
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 Example program

### 9.4.1 Read the internal 1.19V reference signal source value

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT       (65536-FOSC/115200/4)

#define VREFH_ADDR CHIPID07
#define VREFL_ADDR     CHIPID08

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 XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest
}

```

```

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 internal 1.19V High byte of reference signal source
UartSend(VREF_ADDRL);           //read internal 1.19V Low byte of reference signal source

while (1);
}

```

#### 9.4.2 Read the global unique ID number

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"
#include "intrins.h"               // For header files, see Download Software

#define FOSC          11059200UL        //Define as an unsigned long integer to avoid calculation overflow
#define BRT           (65536-FOSC/115200/4)

#define ID_ADDR       (&CHIPID00)

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      XFR
    CKCON = 0x00;                            //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                                //assign to      CPU The speed of executing the program is set to the fastest

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

    char i;

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

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

    while (1);
}

```

#### 9.4.3 Read the frequency of 32K power-down wake-up timer

---

```

//The test frequency is 11.0592MHz

#ifndef "stc8h.h"
#define "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

```

```

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

#define F32K_ADDRH    CHIPID09
#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      XFR
    CKCON = 0x00;         //access to set external data bus speed to fastest
    WTST = 0x00;          //Set the program code to wait for parameters,
                           //assign to      CPU   The speed of executing the program is set to the fastest

    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(F32K_ADDRH);                                //read 32K high byte of frequency
UartSend(F32K_ADDRL);                                //read 32K low byte of frequency

while (1);
}

```

#### 9.4.4 User Defined Internal IRC Frequency

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#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 #define   //VRTRIM_10M
VRT27M_ADDR    CHIPID23 #define   //VRTRIM_27M
VRT44M_ADDR    CHIPID24          //VRTRIM_44M

void main()
{
    EAXFR = 1;                                         //Enable XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to CPU The speed of executing the program is set to the fastest

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

/// choose22.1184MHz

```

```

// CLKDIV = 0x04;
// IRTRIM = T22M_ADDR;
// VRTRIM = VRT27M_ADDR;
// IRCBAND = 0x02;
// CLKDIV = 0x00;

//choose24MHz
CLKDIV = 0x04;
IRTRIM = T24M_ADDR;
VRTRIM = VRT27M_ADDR;
IRCBAND = 0x02;
CLKDIV = 0x00;

/// choose27MHz
// CLKDIV = 0x04;
// IRTRIM = T27M_ADDR;
// VRTRIM = VRT27M_ADDR;
// IRCBAND = 0x02;
// CLKDIV = 0x00;

/// choose30MHz
// CLKDIV = 0x04;
// IRTRIM = T30M_ADDR;
// VRTRIM = VRT27M_ADDR;
// IRCBAND = 0x02;
// CLKDIV = 0x00;

/// choose33.1776MHz
// CLKDIV = 0x04;
// IRTRIM = T33M_ADDR;
// VRTRIM = VRT27M_ADDR;
// IRCBAND = 0x02;
// CLKDIV = 0x00;

/// choose35MHz
// CLKDIV = 0x04;
// IRTRIM = T35M_ADDR;
// VRTRIM = VRT44M_ADDR;
// IRCBAND = 0x03;
// CLKDIV = 0x00;

/// choose44.2368MHz
// CLKDIV = 0x04;
// IRTRIM = T44M_ADDR;
// VRTRIM = VRT44M_ADDR;
// IRCBAND = 0x03;
// CLKDIV = 0x00;

/// choose48MHz
// CLKDIV = 0x04;
// IRTRIM = T48M_ADDR;
// VRTRIM = VRT44M_ADDR;
// IRCBAND = 0x03;
// CLKDIV = 0x00;

while (1);
}

```

## 9.4.5 Read and write off-chip extended RAM

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"
#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      XFR
    CKCON = 0x00;        //access to set external data bus speed to fastest
    WTST = 0x00;         //Set the program code to wait for parameters,
                        //assign to 0      CPU The speed of executing the program is set to the fastest

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

    EXRAM = 1;           //Enable access to off-chip
    BUS_SPEED = 2;        //Set external bus access speed

    x8 = EXRAMB[0x0100]; //Read byte data from 0x0100 externally extended address to a x8
    x16 = EXRAMW[0x0200]; //variable Read byte data 0x0200 externally extended address x16
    x32 = EXRAMD[0x0300]; //          RAM   0x0C00      4      x32
                        //          Keil                  BE (big-endian)
                        //That is, the high byte is stored at a lower address, and the low byte is stored at a higher address

    EXRAMB[0x0101] = x8; //put variable x8 data written to external extension RAM of 0x0101 address
    EXRAMB[0x0205] = x16; //put variable x16 data written to external extension RAM the 0x040A address
    EXRAMB[0x030c] = x32; //put variable x32 data written to external extension RAM 0x0C30 address

    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	DPL1	DPH1	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_ADDR1 IAP_ADDR2 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	INTCKO
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_CCR8B 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			
7FEFA8	DCTIM		T3T4PIN	ADCEXCFG	CMPEXCFG			
7FEFA0	TM0PS	TM1PS	TM2PS	TM3PS	TM4PS			
7FEF98	SPFUNC	RSTFLAG	RSTCR0	RSTCR1	RSTCR2	RSTCR3	RSTCR4	RSTCR5
7FEF88	I2CMSAUX							
7FEF80	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
7FEF70	YEAR	MONTH	DAY	HOUR	MIN	SEC	SSEC	
7FEF68	INIYEAR	INIMONTH	INIDAY	INI HOUR	INI MIN	INI SEC	INI SSEC	
7FEF60	RTCCR	RTCCFG	RTC1EN	RTC1F	ALA HOUR	ALA MIN	ALA SEC	ALA SSEC
7FEF50	LCMIFCFG LCMIFCFG2	LCMIFCR	LCMIFSTA	LCMIFDATAL	LCMDATH			

7FEFE30	P0IE	P1IE	P2IE	P3IE	P4IE	P5IE	P6IE	P7IE
7FEFE28	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR	P6DR	P7DR
7FEFE20	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR	P6SR	P7SR
7FEFE18	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
7FEFE10	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
7FEFE08	X32KCR			HSCLKDIV				
7FEF00	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDDB	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	P0IMO	P1IMO	P2IMO	P3IMO	P4IMO	P5IMO	P6IMO	P7IMO
7EFD10	POINTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF	P6INTF	P7INTF
7EFD00	POINTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE	P6INTE	P7INTE
7EFBF8	HSSPI_CFG HSSPI_CFG2		HSSPI_STA					
7EFBF0	SPPWMA_CFG HSPPWMA_ADR HSPPWMA_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_RXAH DMA_UR4T_RXAL				
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_RXAH DMA_UR3T_RXAL				
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_RXAH DMA_UR2T_RXAL				
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_RXAH DMA_UR1T_RXAL				
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_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	DPL1	DPH1	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_ADDR 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	WK TCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRC BAND LIR TRIM		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_PWM PWMB_CCR5H PWMB_CCR5L PWMB_CCR6H								
7EEFEE PWMB_CCMR1 PWMB_CCMR2 PWMB_CCMR3 PWMB_CCMR4 PWMB_CCER1 PWMB_CCER2 PWMB_CNTRH PWMB_CNTROL								
7EEFE0 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_PWM PWMA_CCR1H PWMA_CCR1L PWMA_CCR2H								
7EFEC8 PWMA_CCMR1 PWMA_CCMR2 PWMA_CCMR3 PWMA_CCMR4 PWMA_CCER1 PWMA_CCER2 PWMA_CNTRH PWMA_CNTROL								
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		
7FEFA0 TM0PS TM1PS TM2PS TM3PS TM4PS								
7FEF98 SPFUNC RSTFLAG RSTCR0 RSTCR1 RSTCR2 RSTCR3 RSTCR4 RSTCR5								
7FEF88 I2CMSAUX								
7FEF80 I2CCFG I2CMSCR I2CMSST I2CSLCR I2CSLST I2CSLADR I2CTxD I2CRxD								
7FEF70 YEAR MONTH DAY HOUR MIN SEC SSEC								
7FEF68 INIYEAR INIMONTH INIDAY INIHOUR INIMIN INISEC INISSEC								
7FEF60 RTCCR RTCCFG RTCIEN RTCIF ALAHOUR ALAMIN ALASEC ALASSEC								
7FEF50 LCMIFCFG LCMIFCFG2 LCMIFCR LCMIFSTA LCMIFDATL LCMDATH								
7FEF40 P0PD P1PD P2PD P3PD P4PD P5PD P6PD P7PD								
7FEF30 P0IE P1IE P2IE P3IE P4IE P5IE P6IE P7IE								

7FEF08	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR	P6DR	P7DR
7FEF09	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR	P6SR	P7SR
7FEF0A	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
7FEF0B	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
7FEF0C	X32KCR			HSCLKDIV				
7FEF0D	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDDB	IRC48MCR
7EFDF8	CHIPID[24]	CHIPID[25]	CHIPID[26]	CHIPID[27]	CHIPID[28]	CHIPID[29]	CHIPID[30]	CHIPID[31]
7EFDFO	CHIPID[16]	CHIPID[17]	CHIPID[18]	CHIPID[19]	CHIPID[20]	CHIPID[21]	CHIPID[22]	CHIPID[23]
7EFD0E	CHIPID[8]	CHIPID[9]	CHIPID[10]	CHIPID[11]	CHIPID[12]	CHIPID[13]	CHIPID[14]	CHIPID[15]
7EFD0F	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	P0IMO	P1IMO	P2IMO	P3IMO	P4IMO	P5IMO	P6IMO	P7IMO
7EFD10	POINTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF	P6INTF	P7INTF
7EFD00	POINTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE	P6INTE	P7INTE
7EFBF8	HSSPI_CFG HSSPI_CFG2		HSSPI_STA					
7EFBF0	SPPWMA_CFG HSPPWMA_ADR HSPPWMA_DAT				HSPPWMB_CFG	HSPPWMB_ADR HSPPWMB_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_RXAH	DMA_UR4T_RXAL	
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_RXAH	DMA_UR3T_RXAL	
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_RXAH	DMA_UR2T_RXAL	
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_RXAH	DMA_UR1T_RXAL	
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_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	DPL1	DPH1	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	WK TCL	WK TCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRC BAND LIR TRIM		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 PWM_B_CCR7H PWMB_CCR7L PWMB_CCR8H PWMB_CCR8L PWMB_BKR PWMB_DTR PWMB_OISR								
7EFEF0 PWMB_PSCRH PWMB_PSCR1 PWMB_ARRH PWMB_ARRL PWMB_RCR PWMB_CCR5H PWMB_CCR5L PWMB_CCR6H								
7EEFEE8 PWMB_CCMR1 PWMB_CCMR2 PWMB_CCMR3 PWMB_CCMR4 PWMB_CCER1 PWMB_CCER2 PWMB_CNTRH PWMB_CNT RL								
7EEFEE0 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_PSCR1 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_CNT RL								
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		
7FEFA8 ADCTIM					T3T4PIN	ADCEXCFG	CMPEXCFG	
7FEFA0 TM0PS	TM1PS	TM2PS	TM3PS	TM4PS				
7FEF98 SPFUNC	RSTFLAG	RSTCR0	RSTCR1	RSTCR2	RSTCR3	RSTCR4	RSTCR5	
7FEF88 I2CMSAUX								
7FEF80 I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD	
7FEF70 YEAR	MONTH	DAY	HOUR	MIN	SEC	SSEC		
7FEF68 INIYEAR	INIMONTH	INIDAY	INI HOUR	INIMIN	INISEC	INISSEC		
7FEF60 RTCCR	RTCCFG	RTC IEN	RTC IF	ALA HOUR	ALA MIN	ALA SEC	ALA SEC	
7FEF50 LCMIFCFG LCMIFCFG2		LCMIFCR	LCMIFSTA	LCMIF DATL	LCMDATH			
7FEF40 P0PD	P1PD	P2PD	P3PD	P4PD	P5PD	P6PD	P7PD	

7FEF30	P0IE	P1IE	P2IE	P3IE	P4IE	P5IE	P6IE	P7IE
7FEF28	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR	P6DR	P7DR
7FEF20	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR	P6SR	P7SR
7FEF18	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS	P6NCS	P7NCS
7FEF10	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU	P6PU	P7PU
7FEF08	X32KCR			HSCLKDIV	HPLLCSR	HPLLSCR		
7FEF00	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_CFLG
7EFD60	PINIPL	PINIPH						
7EFD40	P0WKUE	P1WKUE	P2WKUE	P3WKUE	P4WKUE	P5WKUE	P6WKUE	P7WKUE
7EFD30	P0IM1	P1IM1	P2IM1	P3IM1	P4IM1	P5IM1	P6IM1	P7IM1
7EFD20	P0IMO	P1IMO	P2IMO	P3IMO	P4IMO	P5IMO	P6IMO	P7IMO
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	SPWMMA_CFG HSPWMMA_ADR HSPWMMA_DAT				HSPWMBA_CFG	HSPWMBA_ADR HSPWMBA_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_RXAH DMA_I2ST_RXAL			
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_RXAH DMA_I2CT_RXAL			
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_RXAH DMA_LCM_TXAH DMA_LCM_TKAL 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_RXAH DMA_UR4T_RXAL				
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_RXAH DMA_UR3T_RXAL				
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_RXAH DMA_UR2T_RXAL				
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_RXAL		
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_RXAL	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_RXAL DMA	M2M_RXAH	

## 10.4 List of Special Function Registers (SFR: 0x80-0xFF)

symbol	describe	address	Bit addresses and symbols								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	001,0000
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
TLO Timer 0 lower 8-bit register		8AH									0000,0000
TL1 Timer 1 lower 8-bit register		8BH									0000,0000
TH0 Timer 0 upper 8-bit register		8CH									0000,0000
TH1 Timer 1 upper 8-bit register		8DH									0000,0000
AUXR auxiliary register 1		8EH	10x12	T1x12 UART	T_M0x6 T2R		T2_C/T T2x12 EXTRAM	S1BRT	0000,0001		
INTCLKO Interrupt and Clock Output Control Register		8FH	-	EX4	EX3	EX2	-	T2CLKO T	CLKO T0 CLK0 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 Select Detection		9DH	USBCKS USBCK	S2	-	-	-	-	SEL[1:0]		10xx,xxnn
LIRTRIM IRC Frequency Trim Register		9EH	-	-	-	-	-	-	LIRTRIM[1:0]	xxxx,xxnn	
IRTRIM IRC Frequency Trim 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	Enable 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 WKTN									0111,1111
S3CON Serial	port 3 control register	ACH S3SM0	S3ST3	S3SM2	S3REN S3TB8 S3RB8 SBTI				S3RI	0000,0000	
S3BUF Serial	port 3 data register	ADH									0000,0000
TA	DPTR Timing Control Register	AEH									0000,0000
IE2 Interrupt	Enable Register 2	AFH FUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	0000,0000	
P3	P3 port	B0H F37	P36	P35	P34	P33	P32	P31	P30	1111,1111	
P3M1	P3 port configuration register 1	B1H F37M1	P36M1	P35M1	P34M1	P33M1 P32M1 P31M1	P30M1 1111,1100				
P3M0 P3 port	configuration register 0	B2H F37M0	P36M0	P35M0	P34M0	P33M0 P32M0 P31M0	P30M0 0000,0000				
P4M1	P4 port configuration register 1	B3H F47M1	P46M1	P45M1	P44M1	P43M1 P42M1 P41M1	P40M1 1111,1111				
P4M0 P4 port	configuration register 0	B4H F47M0	P46M0	P45M0	P44M0	P43M0 P42M0 P41M0	P40M0 0000,0000				
IP2 Interrupt	Priority Control Register 2	B5H FUSB	PI2C	PCMP	PX4	PPWMB PPWMA PSPI			PS2	0000,0000	
IP2H High	Interrupt Priority Control Register 2	B6H PUSB	PI2CH PCMPH		PX4H PPWMB PPWMAH PSPI	H P\$2H 0000,0000					
IPH High	Interrupt Priority Control Register B7H		-	PLVDH PAOCH	PSH	PT1H PX	H 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 FAXFR	=	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 F47	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]					
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]			
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 F67M1	P66M1	P65M1	P64M1	P63M1 P62M1 P61M1	P60M1 1111,1111				
P6M0 P6 port	configuration register 0	CCH F67M0	P66M0	P65M0	P64M0	P63M0 P62M0 P61M0	P60M0 0000,0000				
SPSTAT SPI	Status Register	CDH SPIF	WCOL	-	-	-	-	-	-	-	00xx,xxxx
SPCTL SPI	Control Register	CEH \$SIG	SPENDORD		MSTR CPOL CPHA			SPR[1:0]			
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			
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	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				
DPL1 The second group of data pointers (low byte)	E4H									0000,0000			
DPH1 Second group data pointer (high byte)	E5H									0000,0000			
CMPCR1 Comparator Control Register 1	E6H CMPEN CMPIF			PIE	NIE	-	-	-	CMPDE 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 read wait control register	E9H									0000,0111			
CKCON XRAM Control Register	EAH	-	-	-	T1M	TOM	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 PCAN2IF 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 000k,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 USY 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: 0x7EFE00-0x7EFF)

The following special function registers are extended SFR (XFR), the logical address is located in the XDATA area, P\_SW2 needs to be registered before accessing  
Set the most significant bit (EAXFR) of the device to 1, then use the MOV @DRk, Rm and MOV Rm, @DRk instructions to access, for example:

MOV A,#00H

MOV WR6,#WORD0 CLKSEL ; **CLKSEL**

Can be replaced with the register that needs to be accessed

MOV WR4, #WORD2 CLKSEL

MOV @DR4,R11

and

MOV WR6,#WORD0 CLKSEL ; **CLKSEL**

Can be replaced with the register that needs to be accessed

MOV WR4, #WORD2 CLKSEL

MOV R11,@DR4

symbol	describe	address	Bit addresses and symbols								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 00xx,00x0
IRC32KCR Internal 32K oscillator control register	7EFE04H ENIRC32K		-	-	-	-	-	-	-	-	IRC32KST 0xxx,xxx0
MCLKOCR Master Clock Output Control Register	7EFE05H MCLKO_S					MCLKODIV[6:0]					0000,0000
IRCDB Internal high-speed oscillator debounce control	7EFE06H										1000,0000
IRC48MCR Internal 48M oscillator control register	7EFE07H ENIRC48M										IRC48MST 1xxx,xxx0
X32KCR External 32K crystal oscillator control register	7FFE08H ENX32K GAIN32K			-	-	-	-	-	-	-	X32KST 00xx,xxx0
HSCLKDIV High Speed Clock Divider Register	7EFE0BH										0000,0010
HPLLCR High Speed PLL Control Register	7EFE0CH ENHPLL		-	-	-						0xxx,0000
HPLLPSCR High-speed PLL prescaler register	7EFE0DH		-	-	-	-					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 control register	7EFE14H									0000,0000
P5PU	P5 port pull-up resistor control register	7EFE15H	-	-	-						xxx0,0000
P6PU	P6 port pull-up resistor control register	7EFE16H									0000,0000
P7PU	P7 port pull-up resistor control register	7EFE17H									0000,0000
P0NCS	Port 0 Schmitt trigger control register	7EFE18H									0000,0000
P1NCS	Port 1 Schmitt trigger control register	7EFE19H									0000,0000
P2NCS	Port 2 Schmitt trigger control register	7EFE1AH									0000,0000
P3NCS	Port 3 Schmitt trigger control register	7EFE1BH									0000,0000

P4NCS	Port 4 Schmitt trigger control register	7EFE1CH							0000,0000
P5NCS	Port 5 Schmitt trigger control register	7EFE1DH	-	-	-	-			
P6NCS	Port 6 Schmitt trigger control register	7EFE1EH							0000,0000
P7NCS	Port 7 Schmitt trigger control register	7EFE1FH							0000,0000
P0SR	P0 port level conversion rate register	7EFE20H							1111,1111
P1SR	P1 port level conversion rate register	7EFE21H							1111,1111
P2SR	P2 port level conversion rate register	7EFE22H							1111,1111
P3SR	P3 port level conversion rate register	7EFE23H							1111,1111
P4SR	P4 port level conversion rate register	7EFE24H							1111,1111
P5SR	P5 port level conversion rate register	7EFE25H	-	-	-	-			
P6SR	P6 port level conversion rate register	7EFE26H							1111,1111
P7SR	P7 port level conversion rate register	7EFE27H							1111,1111
P0DR	Port 0 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	-	-	-	-			
P6DR	Port 6 drive current control register	7EFE2EH							1111,1111
P7DR	P7 port drive current control register	7EFE2FH							1111,1111
P0IE	Port 0 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	-	-	-	-			
P6IE	P6 port input enable control register	7EFE36H							1111,1111
P7IE	P7 port input enable control register	7EFE37H							1111,1111
P0PD	Port 0 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	-	-	-	-			
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_0	00,0000
LCMIFCFG2 LCM Interface Configuration Register 2		7EFE51H	-	LCMIFCPS[1:0]		SETUPT[2:0]			x000,0000
LCMIFCR LCM Interface Control Register		7EFE52H	ENLCMIF	-	-	-	-	CMD[2:0]	
LCMIFSTA LCM Interface Status Register		7EFE53H	-	-	-	-	-	-	LCMIFIF xxxx,xxx0
LCMIDDATL LCM interface low byte data		7EFE54H	LCMIFSAT[7:0]						0000,0000
LCMIDDATH LCM interface high byte data		7EFE55H	LCMIDAT [15:8]						0000,0000
RTCCR RTC Control Register		7EFE60H	-	-	-	-	-	-	RUNRTC xxxx,xxx0
RTCCFG RTC Configuration Register		7EFE61H	-	-	-	-	-	RTCKS SET	RTC xxxx,xx00

RTCIEN RTC	Interrupt Enable Register	7EFE62H	EALAI	EDAYI	E	HOURI	EMINI			ESECI	E	EC2I	ESEC8I	ESEC32I	0000,0000		
RTCIF	RTC Interrupt Request Register	7EFE63H	A	AIIF	DAYIF	HOURIF	MINIF			SECIF	SEC2IF	SEC8IF	SEC32IF	0000,0000			
ALAHOURL RTC	alarm hour value	7EFE64H	-	-	-	-	-									xxx,0,0000	
ALAMIN RTC	alarm minute value	7EFE65H	-	-	-											xx00,0000	
ALASEC RTC	alarm seconds value	7EFE66H	-	-	-											xx00,0000	
1/128 second value of ALASSEC RTC alarm		7EFE67H	-													x000,0000	
INIYEAR RTC	year initialization	7EFE68H	-													x000,0000	
INIMONTH RTC	monthly initialization	7EFE69H	-	-	-	-	-	-								xxxx,0,0000	
INIDAY RTC	day initialization	7EFE6AH	-	-	-	-	-									xxx0,0,0000	
INIHOUR RTC	hour initialization	7EFE6BH	-	-	-	-	-									xxx0,0,0000	
INIMIN RTC	minute initialization	7EFE6CH	-	-	-											xx00,0,0000	
INISEC RTC	seconds initialization	7EFE6DH	-	-	-											xx00,0,0000	
INISSEC	RTC1/128 seconds initialization	7EFE6EH	-													x000,0000	
Year count	value for YEAR RTC	7EFE70H	-													x000,0000	
Month count	value of MONTH RTC	7EFE71H	-	-	-	-	-	-								xxxx,0,0000	
DAY	Daily count value of RTC	7EFE72H	-	-	-	-	-									xxx0,0,0000	
HOUR RTC	Hour Count Value	7EFE73H	-	-	-	-	-									xxx0,0,0000	
MIN	Minute count value for RTC	7EFE74H	-	-	-											xx00,0,0000	
SEC	Second count value of RTC	7EFE75H	-	-	-											xx00,0,0000	
SSEC	1/128 second count value of RTC	7EFE76H	-													x000,0000	
I2CCFG I2C Configuration Register		7EFE80H	E	I2C	MSSL											0000,0000	
I2CMSCR I2C Master Control Register		7EFE81H	E	MSI		-	-	-								0xxx,0000	
I2CMSST I2C Master Status Register		7EFE82H	M	SBUSY	MSIF											MSACKI	
I2CSLCR I2C Slave Control Register		7EFE83H	-			ESTAI	ERXI	ETXI	ESTOI	-	-	-	-			MSACKO 00xx,xx10	
I2CSLST I2C Slave Status Register		7EFE84H	S	LBUSY	STAIF		RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000			SLRST x000,0xx0	
I2CSLADR I2C Slave Address Register		7EFE85H														MA 0000,0000	
I2CTXD I2C data transmission register		7EFE86H														0000,0000	
I2CRXD I2C Data Receive Register		7EFE87H														0000,0000	
I2CMSAUX I2C Master Auxiliary Control Register		7EFE88H	-	-	-	-	-	-	-	-	-	-	-	-		WDTA xxxx,xxx0	
SPFUNC Auxiliary Control Register		7EFE98H	-	-	-	-	-	-	-	-	-	-	-	-		BKSWR xxxx,xxx0	
RSTFLAG Reset Flag Register		7EFE99H	-	-	-	-	-	-	-	-	-	-	-	-		LVDRSTF WDTRSTF SWRSTF ROMOVF EXRSTF xxxx,0100	
RSTCR0 Reset Control Register 0		7EFE9AH	-	-	-	-	-	-	-	RSTTM34	T\$TTM2	-	-	-		RSTTM01 xxxx,00x0	
RSTCR1 Reset Control Register 1		7EFE9BH	-	-	-	-	-	-	-	RSTUR4	R\$TUR3	R\$TUR2	R\$TUR1	xxxx,0000			
RSTCR2 Reset Control Register 2		7EFE9CH	R\$TCAN2	R\$TCAN	R\$TLIN	R\$TRTC	R\$TPWMB	R\$TPVMA	R\$TI2C	R\$TSP1	0000,0000						
RSTCR3 Reset Control Register 3		7EFE9DH	R\$TFPU	R\$TDMA	R\$TLCM	R\$TLCD	R\$LED	R\$TTKS	R\$TCMP	R\$TADC	0000,0000						
RSTCR4 Reset Control Register 4		7EFE9EH	-	-	-	-	-	-	-	-	-	-	-	-		RSTMDU xxxx,xxx0	
RSTCR5 Reset Control Register 5		7EFE9FH	-	-	-	-	-	-	-	-	-	-	-	-		xxxx,xxxx	
TM0PS Timer 0 Clock Prescaler Register		7EFEA0H														0000,0000	
TM1PS Timer 1 Clock Prescaler Register		7EFEA1H														0000,0000	
TM2PS Timer 2 Clock Prescaler Register		7EFEA2H														0000,0000	
TM3PS Timer 3 Clock Prescaler Register		7EFEA3H														0000,0000	
TM4PS Timer 4 Clock Prescaler Register		7EFEA4H														0000,0000	
ADCTIM ADC Timing Control Register		7EFEA8H	C	SETUP	CSHOLD	[1:0]										SMPDUTY[4:0]	
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]				
PWMA_ETRPS PWMA ETR select register 7EFEB0H							BRKAPS	ETRAPS[1:0]				
PWMA_ENO PWMA output enable control	7EFEB1H ENO4N ENO4P ENO3N ENO3P ENO2N ENO2P ENO1N ENO1P 0000,0000											
PWMA_PS PWMA output pin selection register 7EFEB2H		C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]				
PWMA_IOAUX PWMA auxiliary register	7EFEB3H AUX4N AUX4P AUX3N AUX3P AUX2N AUX2P AUX1N AUX1P 0000,0000											
PWMB_ETRPS PWMB ETR select register 7EFEB4H							BRKBPS	ETRBPS[1:0]				
PWMB_ENO PWMB output enable control	7EFEB5H ENO8P ENO7P ENO6P ENO5P x0x0,x0x0	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P x0x0,x0x0			
PWMB_PS PWMB output pin selection register 7EFEB6H		C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]				
PWMB_IOAUX PWMB auxiliary register	7EFEB7H AUX8P AUX7P AUX6P AUX5P x0x0,x0x0	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P x0x0,x0x0			
CANAR CANBUS address register	7EFEBBH											
CANDR CANBUS Data Register	7EFEBCH											
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 IER	BIE	TIE	COMIE CC1IE	CC3IE CC2IE CC1IE			UIE	0000,0000				
PWMA_SR1 PWMA Status Register 1	7EFEC5H BIF	TIF	COMIF CC1IF	CC3IF CC2IF CC1IF	CC1IF	CC2IF	CC1IF	UIF 0000,0000				
PWMA_SR2 PWMA Status Register 2	7EFEC6H	-	-	CC4OF CC3OF CC2OF CC1OF			-	xxx0,000x				
PWMA_EGR PWMA event occurrence register 7EFEC7H EGR	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]									
PWMA_CNTRL	PWMA counter low byte	7EFECFH	CNT[7:0]									
PWMA_PSCRH	PWMA prescaler high byte	7EFED0H	PSC[15:8]									
PWMA_PSCRL	PWMA prescale low byte	7EFED1H	PSC[7:0]									
PWMA_ARRH	PWMA auto-reload register high byte	7EFED2H	ARR[15:8]									
PWMA_ARRL	PWMA auto-reload register low byte	7EFED3H	ARR[7:0]									
PWMA_RCR	PWMA Repeat Counter Register	7EFED4H	REP[7:0]									
PWMA_CCR1H	PWMA compare capture register 1 high bit	7EFED5H	CCR1[15:8]									
PWMA_CCR1L	PWMA compare capture register 1 low bit	7EFED6H	CCR1[7:0]									
PWMA_CCR2H	PWMA compare capture register 2 high bit	7EFED7H	CCR2[15:8]									
PWMA_CCR2L	PWMA compare capture register 2 low bit	7EFED8H	CCR2[7:0]									
PWMA_CCR3H	PWMA compare capture register 3 high bit	7EFED9H	CCR3[15:8]									

PWMA_CCR3L PWMA compare capture register 3 low bit	7EFEDAH	CCR3[7:0]							0000,0000	
PWMA_CCR4H PWMA compare capture register 4 high bit	7EFEDBH	CCR4[15:8]							0000,0000	
PWMA_CCR4L PWMA compare capture register 4 low bit	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 Dead Time Control Register	7EFEDDEH	DTG[7:0]							0000,0000	
PWMA_OISR PWMA output idle status register	7EFEDFH OIS4N		OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	
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 CC6IE		CC7IE CC6IE CC5IE		UIE	0000,0000	
PWMB_SR1 PWMB Status Register 1	7EFEE5H BIF		TIF	COMIF CC6IF		CC7IF	CC6IF	CC5IF	UIF	
PWMB_SR2 PWMB Status Register 2	7EFEE6H	-	-	-	CC8OF CC7OF CC6OF CC5OF			-	xxx0,000x	
PWMB_EGR PWMB event occurrence 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 prescale high byte	7EFEF0H	PSC[15:8]							0000,0000	
PWMB_PSCRL PWMB prescale low byte	7EFEF1H	PSC[7:0]							0000,0000	
PWMB_ARRH PWMB auto-reload register high byte	7EFEF2H	ARR[15:8]							0000,0000	
PWMB_ARRH PWMB auto-reload register low byte	7EFEF3H	ARR[7:0]							0000,0000	
PWMB_RCR PWMB Repeat Counter Register	7EFEF4H	REP[7:0]							0000,0000	
PWMB_CCR5H PWMB compare capture register 5 high bit	7EFEF5H	CCR1[15:8]							0000,0000	
PWMB_CCR5L PWMB compare capture register 5 low bit	7EFEF6H	CCR1[7:0]							0000,0000	
PWMB_CCR6H PWMB compare capture register 6 high bit	7EFEF7H	CCR2[15:8]							0000,0000	
PWMB_CCR6L PWMB compare capture register 6 low bit	7EFEF8H	CCR2[7:0]							0000,0000	
PWMB_CCR7H PWMB compare capture register 7 high bit	7EFEF9H	CCR3[15:8]							0000,0000	
PWMB_CCR7L PWMB compare capture register 7 low bit	7EFEFAH	CCR3[7:0]							0000,0000	
PWMB_CCR8H PWMB compare capture register 8 high bit	7EFEFBH	CCR4[15:8]							0000,0000	
PWMB_CCR8L PWMB compare capture register 8 low bit	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 Dead Time Control Register	7EFEEFH	DTG[7:0]							0000,0000	
PWMB_OISR PWMB output idle status register	7EFEEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5 x0x0,x0x0	

## 10.6 Extended Special Function Register List (XFR: 0x7EFD00-0x7EFDFF)

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	Port 0 Interrupt Enable Register	7EFDO0H P07INTE P06INTE P05INTE P04INTE P03INTE P02INTE P01INTE P00INTE 0000,0000									
P1INTE	Port 1 Interrupt Enable Register	7EFDO1H P17INTE P16INTE P15INTE P14INTE P13INTE P12INTE P11INTE P10INTE 0000,0000									
P2INTE	Port 2 Interrupt Enable Register	7EFDO2H P27INTE P26INTE P25INTE P24INTE P23INTE P22INTE P21INTE P20INTE 0000,0000									
P3INTE	Port 3 Interrupt Enable Register	7EFDO3H P37INTE P36INTE P35INTE P34INTE P33INTE P32INTE P31INTE P30INTE 0000,0000									
P4INTE	Port 4 Interrupt Enable Register	7EFDO4H P47INTE P46INTE P45INTE P44INTE P43INTE P42INTE P41INTE P40INTE 0000,0000									
P5INTE	Port 5 Interrupt Enable Register	7EFDO5H - - P55INTE P54INTE P53INTE P52INTE P51INTE P50INTE xx00,0000									
P6INTE	Port 6 Interrupt Enable Register	7EFDO6H P67INTE P66INTE P65INTE P64INTE P63INTE P62INTE P61INTE P60INTE 0000,0000									
P7INTE	Port 7 Interrupt Enable Register	7EFDO7H P77INTE P76INTE P75INTE P74INTE P73INTE P72INTE P71INTE P70INTE 0000,0000									
P0INTF	Port 0 Interrupt Flag Register	7EFDO10H P07INTF P06INTF P05INTF P04INTF P03INTF P02INTF P01INTF P00INTF 0000,0000									
P1INTF	Port 1 Interrupt Flag Register	7EFDO11H P17INTF P16INTF P15INTF P14INTF P13INTF P12INTF P11INTF P10INTF 0000,0000									
P2INTF	Port 2 Interrupt Flag Register	7EFDO12H P27INTF P26INTF P25INTF P24INTF P23INTF P22INTF P21INTF P20INTF 0000,0000									
P3INTF	Port 3 Interrupt Flag Register	7EFDO13H P37INTF P36INTF P35INTF P34INTF P33INTF P32INTF P31INTF P30INTF 0000,0000									
P4INTF	Port 4 Interrupt Flag Register	7EFDO14H P47INTF P46INTF P45INTF P44INTF P43INTF P42INTF P41INTF P40INTF 0000,0000									
P5INTF	Port 5 Interrupt Flag Register	7EFDO15H - - P55INTF P54INTF P53INTF P52INTF P51INTF P50INTF xx00,0000									
P6INTF	Port 6 Interrupt Flag Register	7EFDO16H P67INTF P66INTF P65INTF P64INTF P63INTF P62INTF P61INTF P60INTF 0000,0000									
P7INTF	Port 7 Interrupt Flag Register	7EFDO17H P77INTF P76INTF P75INTF P74INTF P73INTF P72INTF P71INTF P70INTF 0000,0000									
P0IM0	Port 0 Interrupt Mode Register 0	7EFDO20H P07IM0 P06IM0 P05IM0 P04IM0 P03IM0 P02IM0 P01IM0 P00IM0 0000,0000									
P1IM0	Port 1 Interrupt Mode Register 0	7EFDO21H P17IM0 P16IM0 P15IM0 P14IM0 P13IM0 P12IM0 P11IM0 P10IM0 0000,0000									
P2IM0	Port 2 Interrupt Mode Register 0	7EFDO22H P27IM0 P26IM0 P25IM0 P24IM0 P23IM0 P22IM0 P21IM0 P20IM0 0000,0000									
P3IM0	Port 3 Interrupt Mode Register 0	7EFDO23H P37IM0 P36IM0 P35IM0 P34IM0 P33IM0 P32IM0 P31IM0 P30IM0 0000,0000									
P4IM0	Port 4 Interrupt Mode Register 0	7EFDO24H P47IM0 P46IM0 P45IM0 P44IM0 P43IM0 P42IM0 P41IM0 P40IM0 0000,0000									
P5IM0	Port 5 Interrupt Mode Register 0	7EFDO25H - - P55IM0 P54IM0 P53IM0 P52IM0 P51IM0 P50IM0 xx00,0000									
P6IM0	Port 6 Interrupt Mode Register 0	7EFDO26H P67IM0 P66IM0 P65IM0 P64IM0 P63IM0 P62IM0 P61IM0 P60IM0 0000,0000									
P7IM0	Port 7 Interrupt Mode Register 0	7EFDO27H P77IM0 P76IM0 P75IM0 P74IM0 P73IM0 P72IM0 P71IM0 P70IM0 0000,0000									
P0IM1	Port 0 Interrupt Mode Register 1	7EFDO30H P07IM1 P06IM1 P05IM1 P04IM1 P03IM1 P02IM1 P01IM1 P00IM1 0000,0000									
P1IM1	Port 1 Interrupt Mode Register 1	7EFDO31H P17IM1 P16IM1 P15IM1 P14IM1 P13IM1 P12IM1 P11IM1 P10IM1 0000,0000									
P2IM1	Port 2 Interrupt Mode Register 1	7EFDO32H P27IM1 P26IM1 P25IM1 P24IM1 P23IM1 P22IM1 P21IM1 P20IM1 0000,0000									
P3IM1	Port 3 Interrupt Mode Register 1	7EFDO33H P37IM1 P36IM1 P35IM1 P34IM1 P33IM1 P32IM1 P31IM1 P30IM1 0000,0000									
P4IM1	Port 4 Interrupt Mode Register 1	7EFDO34H P47IM1 P46IM1 P45IM1 P44IM1 P43IM1 P42IM1 P41IM1 P40IM1 0000,0000									
P5IM1	Port 5 Interrupt Mode Register 1	7EFDO35H - - P55IM1 P54IM1 P53IM1 P52IM1 P51IM1 P50IM1 xx00,0000									
P6IM1	Port 6 Interrupt Mode Register 1	7EFDO36H P67IM1 P66IM1 P65IM1 P64IM1 P63IM1 P62IM1 P61IM1 P60IM1 0000,0000									
P7IM1	Port 7 Interrupt Mode Register 1	7EFDO37H P77IM1 P76IM1 P75IM1 P74IM1 P73IM1 P72IM1 P71IM1 P70IM1 0000,0000									
P0WKUE	P0 port interrupt wake-up enable	7EFDO40H P07WKUE P06WKUE P05WKUE P04WKUE P03WKUE P02WKUE P01WKUE P00WKUE 0000,0000									
P1WKUE	P1 port interrupt wake-up enable	7EFDO41H P17WKUE P16WKUE P15WKUE P14WKUE P13WKUE P12WKUE P11WKUE P10WKUE 0000,0000									
P2WKUE	P2 port interrupt wake-up enable	7EFDO42H P27WKUE P26WKUE P25WKUE P24WKUE P23WKUE P22WKUE P21WKUE P20WKUE 0000,0000									
P3WKUE	P3 port interrupt wake-up enable	7EFDO43H P37WKUE P36WKUE P35WKUE P34WKUE P33WKUE P32WKUE P31WKUE P30WKUE 0000,0000									
P4WKUE	P4 port interrupt wake-up enable	7EFDO44H P47WKUE P46WKUE P45WKUE P44WKUE P43WKUE P42WKUE P41WKUE P40WKUE 0000,0000									
P5WKUE	P5 port interrupt wake-up enable	7EFDO45H - - P55WKUE P54WKUE P53WKUE P52WKUE P51WKUE P50WKUE xx00,0000									
P6WKUE	P6 port interrupt wake-up enable	7EFDO46H P67WKUE P66WKUE P65WKUE P64WKUE P63WKUE P62WKUE P61WKUE P60WKUE 0000,0000									

P7WKUE P7 port	interrupt wake-up enable	7EFDA47H	P7WKUE P76WKUE P75WKUE P74WKUE	P73WKUE P72WKUE P71WKUE P70WKUE	0000,0000						
PINIPL	I/O port interrupt priority low register	7EFDA60H	P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP	0000,0000
PINIPH	I/O port interrupt priority high register	7EFDA61H	P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH	0000,0000
FSHWUPPRD FLS	ASH wake-up wait time register	7EFDA68H									0011,1100
UR1TOCR Serial	port 1 timeout control register	7EFDA70H	ENTO ENTOI SCALE			-	-	-	-	-	000x,xxxx
UR1TOSR Serial	port 1 timeout status register	7EFDA71H		-	-	-	-	-	-	TOIF xxxx	xxx0
UR1TOTH Serial	port 1 timeout length control register	7EFDA72H				TM[15:8]					0000,0000
UR1TOTL Serial	port 1 timeout length control register	7EFDA73H				TM[7:0]					0000,0000
UR2TOCR Serial	port 2 timeout control register	7EFDA74H	ENTO ENTOI SCALE			-	-	-	-	-	000x,xxxx
UR2TOSR Serial	port 2 timeout status register	7EFDA75H		-	-	-	-	-	-	TOIF xxxx	xxx0
UR2TOTH Serial	port 2 timeout length control register	7EFDA76H				TM[15:8]					0000,0000
UR2TOTL Serial	port 2 timeout length control register	7EFDA77H				TM[7:0]					0000,0000
UR3TOCR Serial	port 3 timeout control register	7EFDA78H	ENTO ENTOI SCALE			-	-	-	-	-	000x,xxxx
UR3TOSR Serial	port 3 timeout status register	7EFDA79H		-	-	-	-	-	-	TOIF xxxx	xxx0
UR3TOTH Serial	port 3 timeout length control register	7EFDA7AH				TM[15:8]					0000,0000
UR3TOTL Serial	port 3 timeout length control register	7EFDA7BH				TM[7:0]					0000,0000
UR4TOCR Serial	port 4 timeout control register	7EFDA7CH	ENTO ENTOI SCALE			-	-	-	-	-	000x,xxxx
UR4TOSR Serial	port 4 timeout status register	7EFDA7DH		-	-	-	-	-	-	TOIF xxxx	xxx0
UR4TOTH Serial	port 4 timeout length control register	7EFDA7EH				TM[15:8]					0000,0000
UR4TOTL Serial	port 4 timeout length control register	7EFDA7FH				TM[7:0]					0000,0000
SPITOCR SPI Timeout Control Register		7EFDA80H	ENTO ENTOI SCALE			-	-	-	-	-	000x,xxxx
SPITOSR SPI Timeout Status Register		7EFDA81H		-	-	-	-	-	-	TOIF xxxx	xxx0
SPITOTH SPI timeout length control register		7EFDA82H				TM[15:8]					0000,0000
SPITOTL SPI timeout length control register		7EFDA83H				TM[7:0]					0000,0000
I2CTOCR I2C Timeout Control Register		7EFDA84H	ENTO ENTOI SCALE			-	-	-	-	-	000x,xxxx
I2CTOSR I2C Timeout Status Register		7EFDA85H		-	-	-	-	-	-	TOIF xxxx	xxx0
I2CTOTH I2C timeout length control register		7EFDA86H				TM[15:8]					0000,0000
I2CTOTL I2C timeout length control register		7EFDA87H				TM[7:0]					0000,0000
I2SCR	I2S Control Register	7EFDA98H	TxEIE RXNEIE ERRIE			FRF	-	-		TXDMAEN RXDMAEN	0000,xx00
I2SSR	I2S Status Register	7EFDA99H	-	FRE	BUY	OVR	UDR CHSID		TXE	RXNE	x000,0000
I2SDRH	I2S data register high byte	7EFDA9AH					DR[15:8]				0000,0000
I2SDRL	I2S data register low byte	7EFDA9BH					DR[7:0]				0000,0000
I2SPRH	I2S frequency division register high byte	7EFDA9CH	-	-	-	-	-	-	MCKOE ODD	xxxx,xx00	
I2SPRL	I2S frequency divider register low byte	7EFDA9DH					DIV[7:0]				0000,0000
I2SCFGH	I2S configuration register high byte	7EFDA9EH	-	-	-	-	-	I2SE		I2SCFG[1:0]	xxxx,x000
I2SCFGL	I2S Configuration Register Low Byte	7EFDA9FH	PCMSYNC	-		STD[1:0]	CKPOL	DATLEN[1:0]		CHLEN	0x00,0000
I2SMD	I2S slave mode control register	7EFDA9OH					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 port 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	SPLSV	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 guard time register 7EFDC5H													0000,0000
USARTBRR Serial port 1 baud rate register	7EFDC6H								USARTBRR[15:8]				0000,0000
USARTBRL Serial port 1 baud rate register	7EFDC7H								USARTBRR[7:0]				0000,0000
USART2CR1 Serial port 2 control register 1	7EFDC8H	LINEN	DORD	CLKEN	SPMOD	SPIEN	SPLSV	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 guard 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
CHIPID00 Hardware digital ID00	7EFDE0H								Globally unique ID number (0th byte)				nnnn,nnnn
CHIPID01 Hardware digital ID01	7EFDE1H								Globally unique ID number (1st byte)				nnnn,nnnn
CHIPID02 Hardware digital ID02	7EFDE2H								Globally unique ID number (2nd byte)				nnnn,nnnn
CHIPID03 Hardware digital ID03	7EFDE3H								Globally unique ID number (3rd byte)				nnnn,nnnn
CHIPID04 Hardware digital ID04	7EFDE4H								Globally unique ID number (4th byte)				nnnn,nnnn
CHIPID05 Hardware digital ID05	7EFDE5H								Globally unique ID number (5th byte)				nnnn,nnnn
CHIPID06 Hardware digital ID06	7EFDE6H								Globally Unique ID Number (6th byte)				nnnn,nnnn
CHIPID07 Hardware digital ID07	7EFDE7H								Internal 1.19V reference signal source (high byte)				nnnn,nnnn
CHIPID08 Hardware digital ID08	7EFDE8H								Internal 1.19V reference signal source (low byte)				nnnn,nnnn
CHIPID09 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								22.1184MHz IRC parameters (27M band)				nnnn,nnnn
CHIPID12 Hardware digital ID12	7EFDECH								24MHz IRC parameters (27M band)				nnnn,nnnn
CHIPID13 Hardware digital ID13	7EFDEDH								27MHz IRC parameters (27M band)				nnnn,nnnn
CHIPID14 Hardware digital ID14	7EFDEEH								30MHz IRC parameters (27M band)				nnnn,nnnn
CHIPID15 Hardware digital ID15	7EFDEFH								33.1776MHz IRC parameters (27M band)				nnnn,nnnn
CHIPID16 Hardware digital ID16	7EFDF0H								35MHz IRC parameters (44M band)				nnnn,nnnn
CHIPID17 Hardware digital ID17	7EFDF1H								36.864MHz IRC parameters (44M band)				nnnn,nnnn
CHIPID18 Hardware digital ID18	7EFDF2H								40MHz IRC parameters (44M band)				nnnn,nnnn
CHIPID19 Hardware digital ID19	7EFDF3H								44.2368MHz IRC parameters (44M band)				nnnn,nnnn
CHIPID20 Hardware digital ID20	7EFDF4H								48MHz IRC parameters (44M band)				nnnn,nnnn
CHIPID21 Hardware digital ID21	7EFDF5H								VRTRIM parameters for the 6M band				nnnn,nnnn
CHIPID22 Hardware digital ID22	7EFDF6H								VRTRIM parameters for the 10M band				nnnn,nnnn
CHIPID23 Hardware digital 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 digital ID25	7EFDF9H								00H				nnnn,nnnn
CHIPID26 Hardware digital ID26	7EFDFAH								User program space end address (high byte)				nnnn,nnnn
CHIPID27 Hardware digital ID27	7EFDFBH								Chip test time (years)				nnnn,nnnn

CHIPID28	Hardware digital ID28	7EFDCH	Chip test time (month)								nnnn,nnnn
CHIPID29	Hardware digital ID29	7EFDFH	Chip test time (day)								nnnn,nnnn
CHIPID30	Hardware digital ID30	7EFDFEH	Chip package form number								nnnn,nnnn
CHIPID31	Hardware digital ID31	7EFDFFH	5AH								nnnn,nnnn

## 10.7 Extended Special Function Register List (XFR: 0x7EFB00-0x7EFBFF)

symbol	describe	address	Bit addresses and symbols								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									0000,0000
HSPWMA_DAT	High-speed PWMA data register	7EFBF2H										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									0000,0000
HSPWMB_DAT	High-speed PWMB data register	7EFBF6H										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)

symbol	describe	address	Bit addresses and symbols								reset value	
			B7	B6	B5	B4	B3	B2	B1	B0		
DMA_M2M_CFG	M2M_DMA configuration register	7EFA00H	M2MIE	-	TXACO	RXACO			M2MIP[1:0]		M2MPTY[1:0]	0x00,0000
DMA_M2M_CR	M2M_DMA Control Register	7EFA01H	ENM2M TRIG			-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA Status Register	7EFA02H		-	-	-	-	-	-	-	-	M2MIF xxxx,xxxx0
DMA_M2M_AMT	M2M_DMA transfer total bytes	7EFA03H										0000,0000
DMA_M2M_DONE	M2M_DMA transfer completed 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,xxxx0
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	-	-	-	-	-					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	SPII 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 SP	DMA Status Register	7EFA22H	-	-	-	-	-	TXOVW RXLOSS SPIIF xxxx,x000	
DMA_SPI_AMT SP	DMA transfer total bytes 7EFA23H								0000,0000
DMA_SPI_DONE SP	DMA transfer completed byte number	7EFA24H							0000,0000
DMA_SPI_TXAH SP	DMA transmit high address	7EFA25H							0000,0000
DMA_SPI_RXAL SP	DMA transmit low address	7EFA26H							0000,0000
DMA_SPI_RXAH SP	DMA receive high address	7EFA27H							0000,0000
DMA_SPI_RXAL SP	DMA receive low address	7EFA28H							0000,0000
DMA_SPI_CFG2 SP	DMA Configuration Register 2	7EFA29H	-	-	-	-	-	WRPSS SSS[1:0]	xxxx,x000
DMA_UR1T_CFG	UR1T_DMA Configuration Register 7EFA30H	UR1TIE		-	-	-	UR1TIP[1:0]	UR1TPTY[1:0]	0xxx,0000
DMA_UR1T_CR	UR1T_DMA Control Register 7EFA31H	ENUR1T TRIG			-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA Status Register 7EFA32H		-	-	-	-	TXOVW	-	UR1TIF xxxx,x0x0
DMA_UR1T_AMT	UR1T_DMA transfer total bytes 7EFA33H								0000,0000
DMA_UR1T_DONE	UR1T_DMA transfer completed bytes 7EFA34H								0000,0000
DMA_UR1T_TXAH	UR1T_DMA send high address 7EFA35H								0000,0000
DMA_UR1T_RXAL	UR1T_DMA transmit low address 7EFA36H								0000,0000
DMA_UR1R_CFG	UR1R_DMA Configuration Register 7EFA38H	UR1RIE		-	-	-	UR1RIP[1:0]	UR1RPTY[1:0]	0xxx,0000
DMA_UR1R_CR	UR1R_DMA Control Register 7EFA39H	ENUR1R		-	TRIG	-	-	-	CLRFIFO 0x0,xxx0
DMA_UR1R_STA	UR1R_DMA Status Register 7EFA3AH		-	-	-	-	-	-	RXLOSS UR1RIF xxxx,x0x0
DMA_UR1R_AMT	UR1R_DMA transfer total bytes 7EFA3BH								0000,0000
DMA_UR1R_DONE	UR1R_DMA transfer completed byte number 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 Register 7EFA40H	UR2TIE		-	-	-	UR2TIP[1:0]	UR2TPTY[1:0]	0xxx,0000
DMA_UR2T_CR	UR2T_DMA Control Register 7EFA41H	ENUR2T TRIG			-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA Status Register 7EFA42H		-	-	-	-	TXOVW	-	UR2TIF xxxx,x0x0
DMA_UR2T_AMT	UR2T_DMA transfer total bytes 7EFA43H								0000,0000
DMA_UR2T_DONE	UR2T_DMA transfer completed bytes 7EFA44H								0000,0000
DMA_UR2T_TXAH	UR2T_DMA send high address 7EFA45H								0000,0000
DMA_UR2T_RXAL	UR2T_DMA transmit low address 7EFA46H								0000,0000
DMA_UR2R_CFG	UR2R_DMA Configuration Register 7EFA48H	UR2RIE		-	-	-	UR2RIP[1:0]	UR2RPTY[1:0]	0xxx,0000
DMA_UR2R_CR	UR2R_DMA Control Register 7EFA49H	ENUR2R		-	TRIG	-	-	-	CLRFIFO 0x0,xxx0
DMA_UR2R_STA	UR2R_DMA Status Register 7EFA4AH		-	-	-	-	-	-	RXLOSS UR2RIF xxxx,x0x0
DMA_UR2R_AMT	UR2R_DMA transfer total bytes 7EFA4BH								0000,0000
DMA_UR2R_DONE	UR2R_DMA transfer completed 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 Register 7EFA50H	UR3TIE		-	-	-	UR3TIP[1:0]	UR3TPTY[1:0]	0xxx,0000
DMA_UR3T_CR	UR3T_DMA Control Register 7EFA51H	ENUR3T TRIG			-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA Status Register 7EFA52H		-	-	-	-	TXOVW	-	UR3TIF xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA transfer total bytes 7EFA53H								0000,0000
DMA_UR3T_DONE	UR3T_DMA transfer completed bytes 7EFA54H								0000,0000
DMA_UR3T_TXAH	UR3T_DMA send high address 7EFA55H								0000,0000
DMA_UR3T_RXAL	UR3T_DMA transmit low address 7EFA56H								0000,0000
DMA_UR3R_CFG	UR3R_DMA Configuration Register 7EFA58H	UR3RIE		-	-	-	UR3RIP[1:0]	UR3RPTY[1:0]	0xxx,0000

DMA_UR3R_CR	UR3R_DMA Control Register 7EFA59H	EN	UR3R		-	TRIG	-	-	-	-	CLRFIFO 0x0,xxx0
DMA_UR3R_STA	UR3R_DMA Status Register 7EFA5AH			-	-	-	-	-	-	RXLOSS UR3RIF xxxx,xx00	
DMA_UR3R_AMT	UR3R_DMA transfer total bytes 7EFA5BH										0000,0000
DMA_UR3R_DONE	UR3R_DMA transfer completed byte number 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 Register 7EFA60H	UR4TIE			-	-		UR4TIP[1:0]		UR4TPTY[1:0]	0xxx,0000
DMA_UR4T_CR	UR4T_DMA Control Register 7EFA61H	EN	UR4T	TRIG		-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA Status Register 7EFA62H			-	-	-	-	-	TXOVW	-	UR4TIF xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA transfer total bytes 7EFA63H										0000,0000
DMA_UR4T_DONE	UR4T_DMA transfer completed bytes 7EFA64H										0000,0000
DMA_UR4T_TXAH	UR4T_DMA send high address 7EFA65H										0000,0000
DMA_UR4T_RXAL	UR4T_DMA transmit low address 7EFA66H										0000,0000
DMA_UR4R_CFG	UR4R_DMA Configuration Register 7EFA68H	UR4RIE			-	-	-	UR4RIP[1:0]		UR4RPTY[1:0]	0xxx,0000
DMA_UR4R_CR	UR4R_DMA Control Register 7EFA69H	EN	UR4R		-	TRIG	-	-	-	-	CLRFIFO 0x0,xxx0
DMA_UR4R_STA	UR4R_DMA Status Register 7EFA6AH			-	-	-	-	-	-	RXLOSS UR4RIF xxxx,xx00	
DMA_UR4R_AMT	UR4R_DMA transfer total bytes 7EFA6BH										0000,0000
DMA_UR4R_DONE	UR4R_DMA transfer completed byte number 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	EN	LCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD				0000,0xxx
DMA_LCM_STA	LCM_DMA Status Register 7EFA72H			-	-	-	-	-	-	TXOVW LCMIF xxxx,x00	
DMA_LCM_AMT	LCM_DMA transfer total bytes 7EFA73H										0000,0000
DMA_LCM_DONE	LCM_DMA transfer completed bytes 7EFA74H										0000,0000
DMA_LCM_TXAH	LCM_DMA send high address 7EFA75H										0000,0000
DMA_LCM_TXAL	LCM_DMA transmit 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_M2M_AMTH	M2M_DMA transfer total bytes 7EFA80H										0000,0000
DMA_M2M_DONEH	M2M_DMA transfer completed bytes 7EFA81H										0000,0000
DMA_SPI_AMTH	SPI_DMA transfer total bytes 7EFA84H										0000,0000
DMA_SPI_DONEH	SPI_DMA transfer completed byte number 7EFA85H										0000,0000
DMA_LCM_AMTH	LCM_DMA transfer total bytes 7EFA86H										0000,0000
DMA_LCM_DONEH	LCM_DMA transfer completed bytes 7EFA87H										0000,0000
DMA_UR1T_AMTH	UR1T_DMA transfer total bytes 7EFA88H										0000,0000
DMA_UR1T_DONEH	UR1T_DMA transfer completed bytes 7EFA89H										0000,0000
DMA_UR1R_AMTH	UR1R_DMA transfer total bytes 7EFA8AH										0000,0000
DMA_UR1R_DONEH	UR1R_DMA transfer completed byte number 7EFA8BH										0000,0000
DMA_UR2T_AMTH	UR2T_DMA transfer total bytes 7EFA8CH										0000,0000
DMA_UR2T_DONEH	UR2T_DMA transfer completed byte number 7EFA8DH										0000,0000
DMA_UR2R_AMTH	UR2R_DMA transfer total bytes 7EFA8EH										0000,0000
DMA_UR2R_DONEH	UR2R_DMA transfer completed bytes 7EFA8FH										0000,0000
DMA_UR3T_AMTH	UR3T_DMA transfer total bytes 7EFA90H										0000,0000

DMA_UR3T_DONEH	UR3T_DMA transfer completed bytes 7EFA91H								0000,0000	
DMA_UR3R_AMTH	UR3R_DMA transfer total bytes 7EFA92H								0000,0000	
DMA_UR3R_DONEH	UR3R_DMA transfer completed bytes 7EFA93H								0000,0000	
DMA_UR4T_AMTH	UR4T_DMA transfer total bytes 7EFA94H								0000,0000	
DMA_UR4T_DONEH	UR4T_DMA transfer completed bytes 7EFA95H								0000,0000	
DMA_UR4R_AMTH	UR4R_DMA transfer total bytes 7EFA96H								0000,0000	
DMA_UR4R_DONEH	UR4R_DMA transfer completed bytes 7EFA97H								0000,0000	
DMA_I2CT_CFG	I2CT_DMA configuration register 7EFA98H I2CTIE	-	-	-	-	I2CTIP[1:0]		I2CTTPY[1:0]		0xxx,0000
DMA_I2CT_CR	I2CT_DMA Control Register 7EFA99H ENI2CT TRIG	-	-	-	-	-	-	-	-	00xx,xxxx
DMA_I2CT_STA	I2CT_DMA Status Register 7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF xxxx,x0x0	
DMA_I2CT_AMT	I2CT_DMA transfer total bytes 7EFA9BH								0000,0000	
DMA_I2CT_DONE	I2CT_DMA transfer completed byte number 7EFA9CH								0000,0000	
DMA_I2CT_TXAH	I2CT_DMA send high address 7EFA9DH								0000,0000	
DMA_I2CT_RXAL	I2CT_DMA transmit low address 7EFA9EH								0000,0000	
DMA_I2CR_CFG	I2CR_DMA Configuration Register 7EFAA0H I2CRIE	-	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0]		0xxx,0000
DMA_I2CR_CR	I2CR_DMA Control Register 7EFAA1H ENI2CR TRIG	-	-	-	-	-	-	-	-	CLRFIFO 00xx,xxx0
DMA_I2CR_STA	I2CR_DMA Status Register 7EFAA2H	-	-	-	-	-	-	-	RXLOSS I2CRIF xxxx,xx00	
DMA_I2CR_AMT	I2CR_DMA transfer total bytes 7EFAA3H								0000,0000	
DMA_I2CR_DONE	I2CR_DMA transfer completed byte number 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 transfer total bytes 7EFAA8H								0000,0000	
DMA_I2CT_DONEH	I2CT_DMA transfer completed byte number 7EFAA9H								0000,0000	
DMA_I2CR_AMTH	I2CR_DMA transfer total bytes 7EFAAAH								0000,0000	
DMA_I2CR_DONEH	I2CR_DMA transfer completed bytes 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 register 7EFAB0H I2STIE	-	-	-	-	I2STIP[1:0]		I2STPTY[1:0]		0xxx,0000
DMA_I2ST_CR	I2ST_DMA Control Register 7EFAB1H ENI2ST	-	TRIG	-	-	-	-	-	-	00xx,xxxx
DMA_I2ST_STA	I2ST_DMA Status Register 7EFAB2H	-	-	-	-	-	TXOVW	-	I2STIF xxxx, x0x0	
DMA_I2ST_AMT	I2ST_DMA transfer total bytes 7EFAB3H								0000,0000	
DMA_I2ST_DONE	I2ST_DMA transfer completed byte number 7EFAB4H								0000,0000	
DMA_I2ST_TXAH	I2ST_DMA send high address 7EFAB5H								0000,0000	
DMA_I2ST_RXAL	I2ST_DMA transmit low address 7EFAB6H								0000,0000	
DMA_I2SR_CFG	I2SR_DMA Configuration Register 7EFAB8H I2SRIE	-	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0]		0xxx,0000
DMA_I2SR_CR	I2SR_DMA Control Register 7EFAB9H ENI2SR	-	TRIG	-	-	-	-	-	-	CLRFIFO 00x0,xxx0
DMA_I2SR_STA	I2SR_DMA Status Register 7EFABAHH	-	-	-	-	-	-	-	RXLOSS I2SRIF xxxx,xx00	
DMA_I2SR_AMT	I2SR_DMA transfer total bytes 7EFABBH								0000,0000	
DMA_I2SR_DONE	I2SR_DMA transfer completed byte number 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 transfer total bytes 7EFAC0H								0000,0000	
DMA_I2ST_DONEH	I2ST_DMA transfer completed byte number 7EFAC1H								0000,0000	

DMA_I2SR_AMTH	I2SR_DMA transfer total bytes 7EFAC2H												0000,0000
DMA_I2SR_DONEH	I2SR_DMA transfer completed byte number 7EFAC3H												0000,0000
DMA_ARB_CFG	DMA President Configuration Register 7EFAF8H	WT	RREN	-	-	-	-	STASEL[3:0]-					0xxx,0000
DMA_ARB_STA	DMA President Status Register 7EFAF9H												0000,0000

STCMCU

# 11 I/O port

product line	Maximum number of I/O ports
STC32G12K128 series	<b>60</b>
STC32G8K64 series	<b>45</b>
STC32F12K60 series	<b>45</b>

All I/O ports of STC32G series microcontrollers have 4 working modes: quasi-bidirectional port/weak pull-up (standard 8051 output port mode),

Push-pull output/strong pull-up, high-impedance input (current can neither flow in nor out), open-drain output. The working mode of the I/O port can be controlled by software for easy configuration.

## Notes on I/O :

1. The state of P3.0 and P3.0 ports after power-on is weak pull-up/quasi-bidirectional port mode

2. Except for P3.0 and P3.1, all other IO ports are in the high-impedance input state after power-on, and the user is using the IO port.

The IO port mode must be set before

3. When the chip is powered on, if you do not need to use USB for ISP download, the three I/O ports of P3.0/P3.1/P3.2 cannot be used at the same time.

is low, otherwise it will enter USB download mode and cannot run 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 switch from the high-impedance input state to the dual input state for a short time.

To port mode, used to read the external state of the P3.2 port to determine whether to enter the USB download mode

5. When using P5.4 as the reset pin, the 4K pull-up resistor inside this port will always be turned on;

When I/O port, based on the special consideration that this I/O port shares the pin with the reset pin, the 4K pull-up resistor inside the port depends on the It will turn on for about 6.5 milliseconds, and then automatically turn off (when the user's circuit design needs to use the P5.4 port to drive the external

When using the external circuit, please be sure to consider the problem that there will be a high level of 6.5 milliseconds at the moment of power-on)

## 11.1 I/O port related registers

symbol	describe	address	Bit addresses and symbols								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	93H P07M1 P06M1 P05M1 P04M1 P03M1 P02M1 P01M1 P00M1	0000,0000								
P0M1	Port 0 configuration register 1	94H P07M0 P06M0 P05M0 P04M0 P03M0 P02M0 P01M0 P00M0	1111,1111								
P1M0 P1 port	configuration register 0	91H P17M1 P16M1 P15M1 P14M1 P13M1 P12M1 P11M1 P10M1	0000,0000								

P1M1	P1 port configuration register 1	92H P17M0 P16M0 P15M0 P14M0 P13M0 P12M0 P11M0 P10M0 1111,1111						
P2M0 P2 port	configuration register 0	95H P27M1 P26M1 P25M1 P24M1 P23M1 P22M1 P21M1 P20M1 0000,0000						
P2M1	P2 port configuration register 1	96H P27M0 P26M0 P25M0 P24M0 P23M0 P22M0 P21M0 P20M0 1111,1111						
P3M0 P3 port	configuration register 0	B1H P37M1 P36M1 P35M1 P34M1 P33M1 P32M1 P31M1 P30M1 0000,0000						
P3M1	P3 port configuration register 1	B2H P37M0 P36M0 P35M0 P34M0 P33M0 P32M0 P31M0 P30M0 1111,1100						
P4M0 P4 port	configuration register 0	B3H P47M1 P46M1 P45M1 P44M1 P43M1 P42M1 P41M1 P40M1 0000,0000						
P4M1	P4 port configuration register 1	B4H P47M0 P46M0 P45M0 P44M0 P43M0 P42M0 P41M0 P40M0 1111,1111						
P5M0 P5 port	configuration register 0	C9H - - P55M1 P54M1 P53M1 P52M1 P51M1 P50M1 xx00,0000						
P5M1	P5 port configuration register 1	CAH - - P55M0 P54M0 P53M0 P52M0 P51M0 P50M0 xx11,1111						
P6M0 P6 port	configuration register 0	CBH P67M1 P66M1 P65M1 P64M1 P63M1 P62M1 P61M1 P60M1 0000,0000						
P6M1	P6 port configuration register 1	CCH P67M0 P66M0 P65M0 P64M0 P63M0 P62M0 P61M0 P60M0 1111,1111						
P7M0 P7 port	configuration register 0	E1H P77M1 P76M1 P75M1 P74M1 P73M1 P72M1 P71M1 P70M1 0000,0000						
P7M1	P7 port configuration register 1	E2H P77M0 P76M0 P75M0 P74M0 P73M0 P72M0 P71M0 P70M0 1111,1111						

symbol	describe	address	Bit addresses and symbols								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	Port 0 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	Port 3 Schmitt trigger control register	7EFE1BH P37NCS P36NCS P35NCS P34NCS P33NCS P32NCS P31NCS P30NCS 0000,0000									
P4NCS	P4 port Schmitt trigger control register	7EFE1CH P47NCS P46NCS P45NCS P44NCS P43NCS P42NCS P41NCS P40NCS 0000,0000									
P5NCS	Port 5 Schmitt trigger control register	7EFE1DH - - P55NCS P54NCS P53NCS P52NCS P51NCS P50NCS xx00,0000									
P6NCS	Port 6 Schmitt trigger control register	7EFE1EH P67NCS P66NCS P65NCS P64NCS P63NCS P62NCS P61NCS P60NCS 0000,0000									
P7NCS	Port 7 Schmitt trigger control register	7EFE1FH P77NCS P76NCS P75NCS P74NCS P73NCS P72NCS P71NCS P70NCS 0000,0000									
P0SR	P0 port level conversion rate register	7EFE20H P07SR P06SR P05SR P04SR P03SR P02SR P01SR P00SR 1111,1111									
P1SR	P1 port level conversion rate register	7EFE21H P17SR P16SR P15SR P14SR P13SR P12SR P11SR P10SR 1111,1111									
P2SR	P2 port level conversion rate register	7EFE22H P27SR P26SR P25SR P24SR P23SR P22SR P21SR P20SR 1111,1111									
P3SR	P3 port level conversion rate register	7EFE23H P37SR P36SR P35SR P34SR P33SR P32SR P31SR P30SR 1111,1111									
P4SR	P4 port level conversion rate register	7EFE24H P47SR P46SR P45SR P44SR P43SR P42SR P41SR P40SR 1111,1111									
P5SR	P5 port level conversion rate register	7EFE25H - - P55SR P54SR P53SR P52SR P51SR P50SR xx11,1111									
P6SR	P6 port level conversion rate register	7EFE26H P67SR P66SR P65SR P64SR P63SR P62SR P61SR P60SR 1111,1111									
P7SR	P7 port level conversion 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	Port 0 pull-down resistor control register	7EFE40H	P07PD P06PD P05PD P04PD P03PD	P02PD P11PD P00PD	000,0000				
P1PD	P1 port pull-down resistance 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 P21PD P20PD	0000,0000				
P3PD	P3 port pull-down resistor control register	7EFE43H	P37PD P36PD P35PD P34PD P33PD	P32PD P31PD P30PD	0000,0000				
P4PD	P4 port pull-down resistor control register	7EFE44H	P47PD P46PD P45PD P44PD P43PD	P42PD P41PD P40PD	0000,0000				
P5PD	P5 port pull-down resistor control register	7EFE45H	-	-	P55PD P54PD P53PD P52PD	P51PD P50PD	xx00,0000		
P6PD	P6 port pull-down resistor control register	7EFE46H	P67PD P66PD P65PD P64PD P63PD	P62PD P61PD P60PD	0000,0000				
P7PD	P7 port pull-down resistor control register	7EFE47H	P77PD P76PD P75PD P74PD P73PD	P72PD P71PD P70PD	0000,0000				

### 11.1.1 Port Data Register (Px)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	A0H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	B0H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
P4	C0H	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5	C8H	-	-	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
P6	E8H	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
P7	F8H	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0

Read and write port status

Write 0: output low level to port buffer

Write 1: output high level to port buffer

Read: Directly read the level on the port pin

### 11.1.2 Port Mode Configuration Registers (PxM0, PxM1)

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

Configure the mode of the port

PnM1.x	PnM0.x	Pn.x port working mode
0	0	Quasi-bidirectional port
0	1	Push-pull output
1	0	High impedance input
1	1	Open drain output

### 11.1.3 Port Pull-Up Resistor Control Register (PxPU)

symbol	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							

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

0: Disable the 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)

symbol	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 bits

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

1: Disable the Schmitt trigger function of the port.

### 11.1.5 Port Level Transition Speed Control Register (PxSR)

symbol	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 translation

0: The level conversion speed is fast, and the corresponding upper and lower impulses will be relatively large

1: The level conversion speed is slow, and the corresponding upper and lower impulses are relatively small

### 11.1.6 Port Drive Current Control Register (PxDR)

symbol	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 the drive capability of the port

0: General drive capability

1: Enhance the driving ability

### 11.1.7 Port digital signal input enable control register (PxIE)

symbol	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 a comparator input port, ADC input port or touch key input port and other analog ports,

Before entering the clock stop mode, it must be set to 0, otherwise there will be extra power consumption.

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

### 11.1.8 Port Pull-Down Resistor Control Register (PxPD)

symbol	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 pull-down resistor inside the port

1: Enable the pull-down resistor inside the port

STCMCU

## 11.2 Configuring I/O Ports

The configuration of each I/O requires two registers to be set up.

Taking the P0 port as an example, the configuration of the P0 port needs to use two registers P0M0 and P0M1 for configuration, as shown in the following figure:

That is, the 0th bit of P0M0 and the 0th bit of P0M1 are combined to configure the mode of port P0.0

That is, the first bit of P0M0 and the first bit of P0M1 are combined to configure the mode of port P0.1

All other I/Os are configured similarly.

The combination of PnM0 and PnM1 is shown in the table below

PnM1	PnM0	I/O port working mode
0	0	Quasi-bidirectional port (traditional 8051 port mode, weak pull-up)  The sink current can reach 20mA, and the source current is 270~150µA (manufacturing error exists)
0	1	Push-pull output (strong pull-up output, up to 20mA, with a current limiting resistor)
1	0	High impedance input (current can neither flow in nor out)
1	1	Open-drain output (Open-Drain), internal pull-up resistor disconnected  Open-drain mode can both read external status and external output (high or low flat). To read the external state correctly or to output a high level externally, an external  Add a pull-up resistor, otherwise the external state will not be read, and the high level will not be output to the outside world.

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

### Notice:

Although each I/O port can withstand a sink current of 20mA in weak pull-up (quasi-bidirectional port) / strong push-pull output / open-drain mode (current limiting is still required) Resistors, such as 1K, 560Ω, 472Ω, etc., can output a current of 20mA in the case of strong push-pull output (also add a current limiting resistor), but the whole chip The recommended working current should not exceed 90mA, that is, the recommended current flowing from VCC should not exceed 90mA, and the recommended current flowing from GND should not exceed 90mA. 90mA, the overall inflow/outflow current is not recommended to exceed 90mA.

## 11.3 I/O structure diagram

### 11.3.1 Quasi-bidirectional port (weak pull-up)

The quasi-bidirectional port (weak pull-up) output type can be used as output and input function without reconfiguring the port output state. This is because the drive capability is weak when the port output is 1, allowing an external device to pull it low. When the pin output is low, it is very capable of driving and can sink a considerable amount of current. The quasi-bidirectional port has 3 pull-up transistors to suit different needs.

Of the 3 pull-up transistors, 1 pull-up transistor is called a "weak pull-up" and 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. If a pin output is 1 and is pulled low by an external device, the weak pull-up is turned off and the "very weak pull-up" remains on. In order to pull this pin low, the external device must have sufficient current sink capability Bring the voltage on the pin 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.

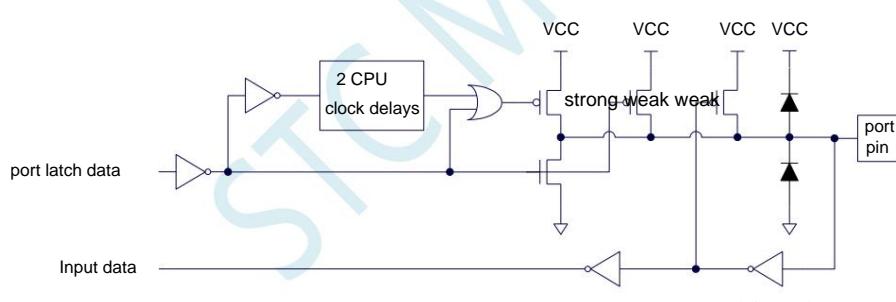
A second pull-up transistor, called a "very weak pull-up", turns on when the port latch is 1. When the pin is left floating, this very weak pull-up source generates a very weak pull-up current that pulls the pin high. For a 5V microcontroller, the current of the "extremely weak pull-up" transistor is about 18uA; for a 3.3V microcontroller, the current of the "extremely weak pull-up" transistor is about 5uA.

The third pull-up transistor is called "strong pull-up". This pull-up is used to speed up quasi-bidirectional port switching when the port latch transitions from 0 to 1.

Logic 0 to logic 1 transition. When this happens, the strong pull-up is turned on for about 2 clocks to allow the pin to be pulled high quickly.

Quasi-bidirectional port (weak pull-up) with a Schmitt trigger input and a glitch suppression circuit. Quasi-bidirectional port (weak pull-up) to read external Before the state, it must be latched as '1' before the correct external state can be read.

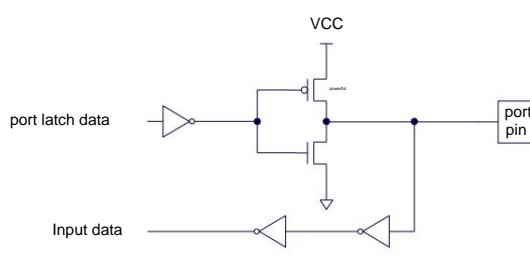
The output of the quasi-bidirectional port (weak pull-up) is shown in the figure below:



### 11.3.2 Push-Pull Output

The pull-down structure of the strong push-pull output configuration is the same as the pull-down structure of the open-drain output and the quasi-bidirectional port, but provides a continuous strong pull-up when the latch is 1. Push-Pull mode is generally used when a larger drive current is required.

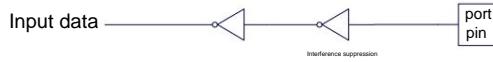
The strong push-pull pin configuration is shown in the following figure:



### 11.3.3 High- Impedance Inputs

Current can neither flow in nor out of the input port with  
a Schmitt trigger input and a glitch suppression circuit

The high-impedance input pin configuration is shown in the following figure:



### 11.3.4 Open-Drain Output

Open-drain mode can both read external status and output externally (high level or low level). If you want to read the external state correctly or need to output externally

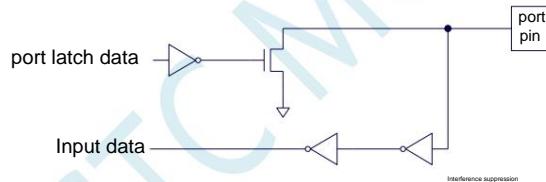
For high level, an external pull-up resistor is required.

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 side

The mode must have an external pull-up, generally connected to VCC through a resistor. If there is an external pull-up resistor, the open-drain I/O port can also read the external status, that is, the I/O port configured in open-drain mode can also be used as an input I/O port. The pull-down in this way is the same as the quasi-bidirectional port.

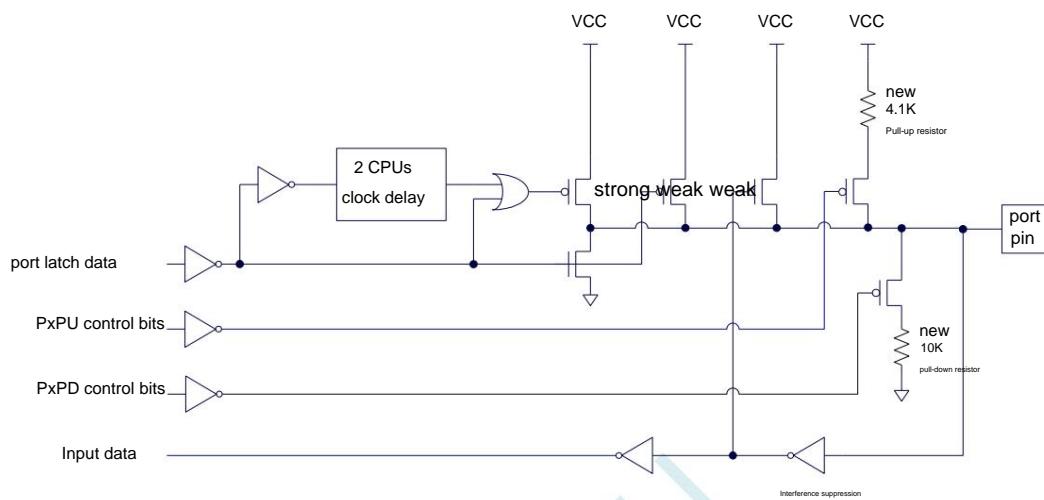
The open-drain port has a Schmitt trigger input and a glitch suppression circuit.

The output port configuration is shown in the following figure:



### 11.3.5 Added 4.1K pull-up resistor and 10K pull-down resistor

All I/O ports of the STC32G series can enable a pull-up resistor of about 4.1K (due to manufacturing errors, the range of the pull-up resistor is range may be 3K~5K) and a pull-down resistor of about 10K (due to manufacturing errors, the range of pull-down resistors may be 8K~12K)



Port Pull-Up Resistor Control Register

Symbol	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	

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

0: Disable the 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)

symbol	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 pull-down resistor inside the port

1: Enable the pull-down resistor inside the port

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

When the user needs the I/O port to output a faster frequency externally, it can increase the drive current of the I/O port and increase the level conversion speed of the I/O port.

In order to improve the external output speed of the I/O port

Set the PxSR register, which can be used to control the level conversion speed of the I/O port. When it is set to 0, the corresponding I/O port is flipped quickly, and it is set to

1 is a slow rollover.

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

When it is 0, it is a strong drive current

### 11.3.7 How to set the current drive capability of the I/O port

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

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

When it is 0, it is a strong drive current

### 11.3.8 How to reduce the external radiation of I/O ports

Due to the setting of the PxSR register, it can be used to control the level conversion speed of the I/O port. Setting the PxDR register can be used to control the I/O port driver.

Dynamic current size

When the external radiation of the I/O port needs to be reduced, the PxSR register needs to be set to 1 to reduce the level conversion speed of the I/O port.

Set the PxDR register to 1 to reduce the I/O drive current, and ultimately reduce the external radiation of the I/O port

## 11.4 Example program

### 11.4.1 Port mode settings (applicable to all I/Os)

---

```
//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to0 CPU The speed of executing the program is set to the fastest

    P0M0 = 0x00; //set upP0.0~P0.7 for bidirectional port mode
    P0M1 = 0x00;
    P1M0 = 0xff; //set upP1.0~P1.7 for push-pull output mode
    P1M1 = 0x00;
    P2M0 = 0x00; //set upP2.0~P2.7 for high-impedance input mode
    P2M1 = 0xff;

    P3M0 = 0xcc; //setting P3.0~P3.1 for bidirectional port mode
    P3M1 = 0xf0; //setting P3.2~P3.3 for push-pull output mode
                  //setting P3.4~P3.5 for high-impedance input mode
                  //setting P3.6~P3.7 open-drain mode

    while (1);
}
```

---

### 11.4.2 Bidirectional port read and write operations (applicable to all I/O)

---

```
//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to0 CPU The speed of executing the program is set to the fastest

    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 up P0.0~P0.7 for bidirectional port mode
P0M1 = 0x00;

P0 = 0xff;                                    //P0 All ports output high level
_nop_();
_nop_();
P0 = 0x00;                                    //P0 All ports 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 internal weak pull-up resistor before reading the port
_nop_();
_nop_();
CY = P00;                                     //wait two clocks
                                                //read port status

while (1);
}

```

#### 11.4.3 Turn on the internal pull-up resistor of the I/O port (applicable to all I/Os)

---

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                           //For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                               //Set the program code to wait for parameters,
                                                //assign to      CPU   The speed of executing the program is set to the fastest

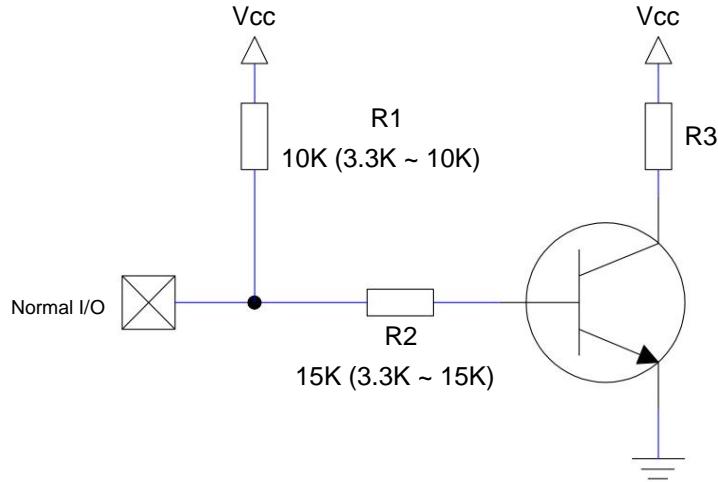
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;

```

```
P3M1 = 0x00;  
P4M0 = 0x00;  
P4M1 = 0x00;  
P5M0 = 0x00;  
P5M1 = 0x00;  
  
P0PU = 0x0f; //Open P0.0~P0.3 Internal pull-up resistor on port  
  
P1PU = 0xf0; //Open P1.4~P1.7 Internal pull-up resistor on port  
  
while (1);  
}
```

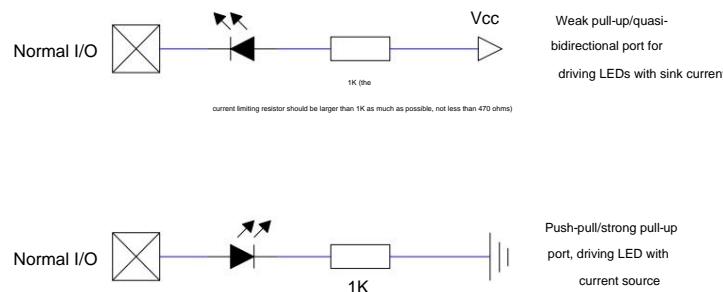
STCMCU

## 11.5 A typical triode control circuit



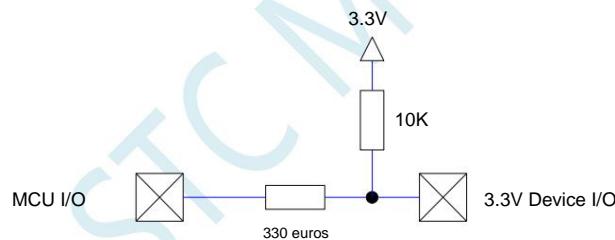
If a weak pull-up control is used, it is recommended to add a pull-up resistor R1 (3.3K~10K). If the pull-up resistor R1 (3.3K~10K) is not added, it is recommended that the value is above 15K, or use a strong push-pull output.

## 11.6 Typical LED Control Circuit



## 11.7 I/O port interconnection of 3V/5V devices in mixed voltage power supply system

When the STC series wide-voltage single-chip microcomputer works at 5V, if it is necessary to directly connect the 3.3V device, in order to prevent the 3.3V device from being unable to withstand 5V, a 330 $\mu$ A current limiting resistor can be connected to the I/O port of the 3.3V device first in series with the corresponding microcontroller I/O port. O port, when the program is initialized, 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 to the Vcc of the 3.3V device from the I/O port of the corresponding 3.3V device. The level is 3.3V, the low level is 0V, and the input and output are all normal.

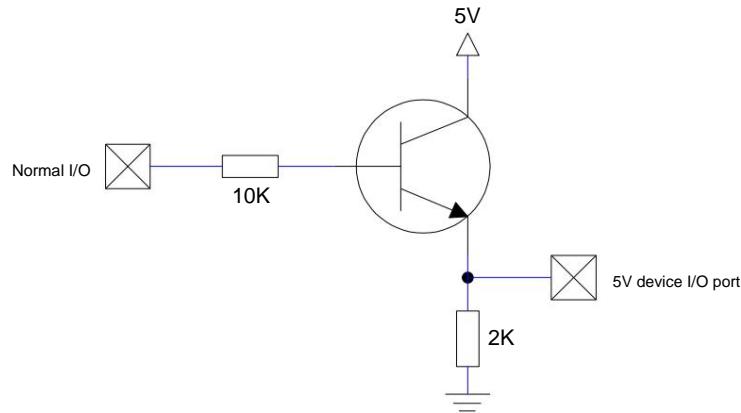


When the STC wide-voltage microcontroller works at 3V, if it is necessary to directly connect a 5V device, if the corresponding I/O port is an input, an isolation diode can be connected in series on the I/O port to isolate the high-voltage part. When the external signal voltage is higher than the working voltage of the microcontroller, it is turned off, and the I/O port is internally pulled up to a high level, so the state of the read I/O port is high level; when the external signal voltage is low, it is turned on, and the I/O port Clamped at 0.7V, read by microcontroller when less than 0.8V

The status of the I/O port is low.



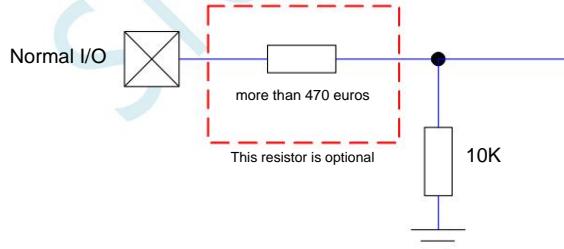
When the STC wide-voltage microcontroller works at 3V, if it needs to be directly connected to a 5V device, if the corresponding I/O port is an output, an NPN can be used. The triode is isolated, and the circuit is as follows:



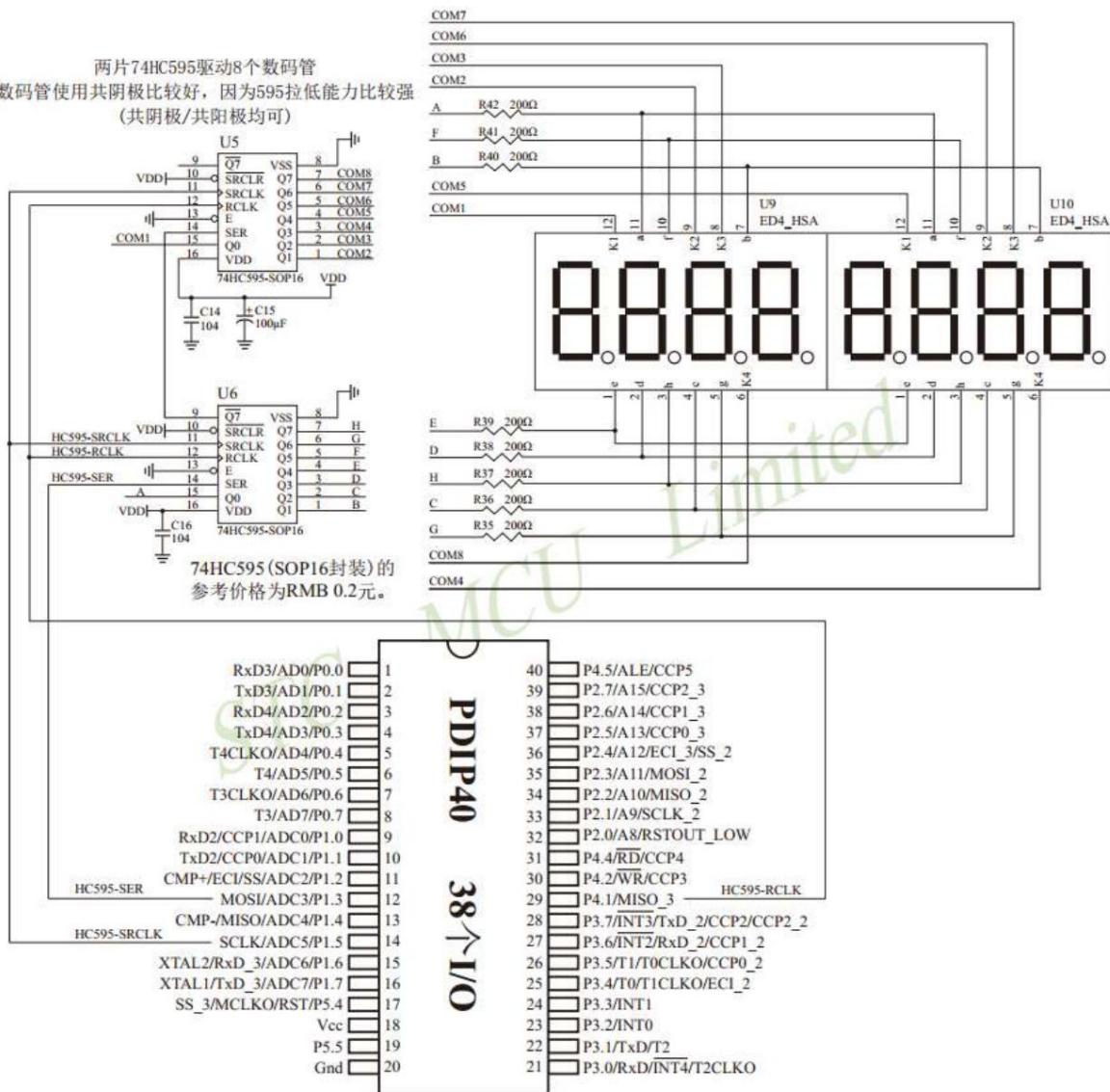
## 11.8 How to make the I/O port low level when power-on reset

When the ordinary 8051 microcontroller is powered on and reset, the ordinary I/O port is a weak pull-up (quasi-bidirectional port) high-level output, and many practical applications require a certain Some I/O ports are low-level output, otherwise the controlled system (such as a motor) will malfunction. The current STC microcontroller has both weak pull-up output and low-level output. Strong push-pull output can easily solve this problem.

Now a pull-down resistor (about 10K) can be added to the I/O port of the STC microcontroller, so that when power-on reset, in addition to the download ports P3.0 and P3.1 Except for weak pull-up (quasi-bidirectional port), other I/O ports are high-impedance input mode, and there are external pull-down resistors, so when the I/O port is powered on and reset, the external to low level. 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 the output is strong, the I/O port drives The current can reach 20mA, so this port can definitely be driven as a high-level output.



## 11.9 The circuit diagram of using 74HC595 to drive 8 digital tubes (serial expansion, 3 lines)



## 12 interrupt system

The interrupt system is set up to enable the CPU to have real-time processing capability for external emergencies.

When the central processing unit CPU is processing something, an emergency event request occurs from the outside world, requiring the CPU to suspend the current work, Turn to deal with this emergency. After dealing with it, go back to the place where it was interrupted and continue the original work. This process is called interrupt. The component that realizes this function is called the interrupt system, and the request source that asks the CPU to interrupt is called the interrupt source. Microcomputer's interrupt system is generally Multiple interrupt sources are allowed. When several interrupt sources request an interrupt from the CPU at the same time and request to service it, there is a CPU priority response. The question of which interrupt source is requested. Usually queue up according to the priority of the interrupt source, and prioritize the interrupt request source of the most urgent event, that is, the regular Each interrupt source has a priority level. The CPU always responds to the highest priority interrupt request first.

When the CPU is processing an interrupt source request (executing the corresponding interrupt service routine), another priority than it occurs also high interrupt source requests. If the CPU can suspend the service routine for the original interrupt source, and instead process the interrupt request source with higher priority, After processing, return to the original low-level interrupt service routine. This process is called interrupt nesting. Such an interrupt system is called a multi-level interrupt system System, the interrupt system without interrupt nesting function is called single-level interrupt system.

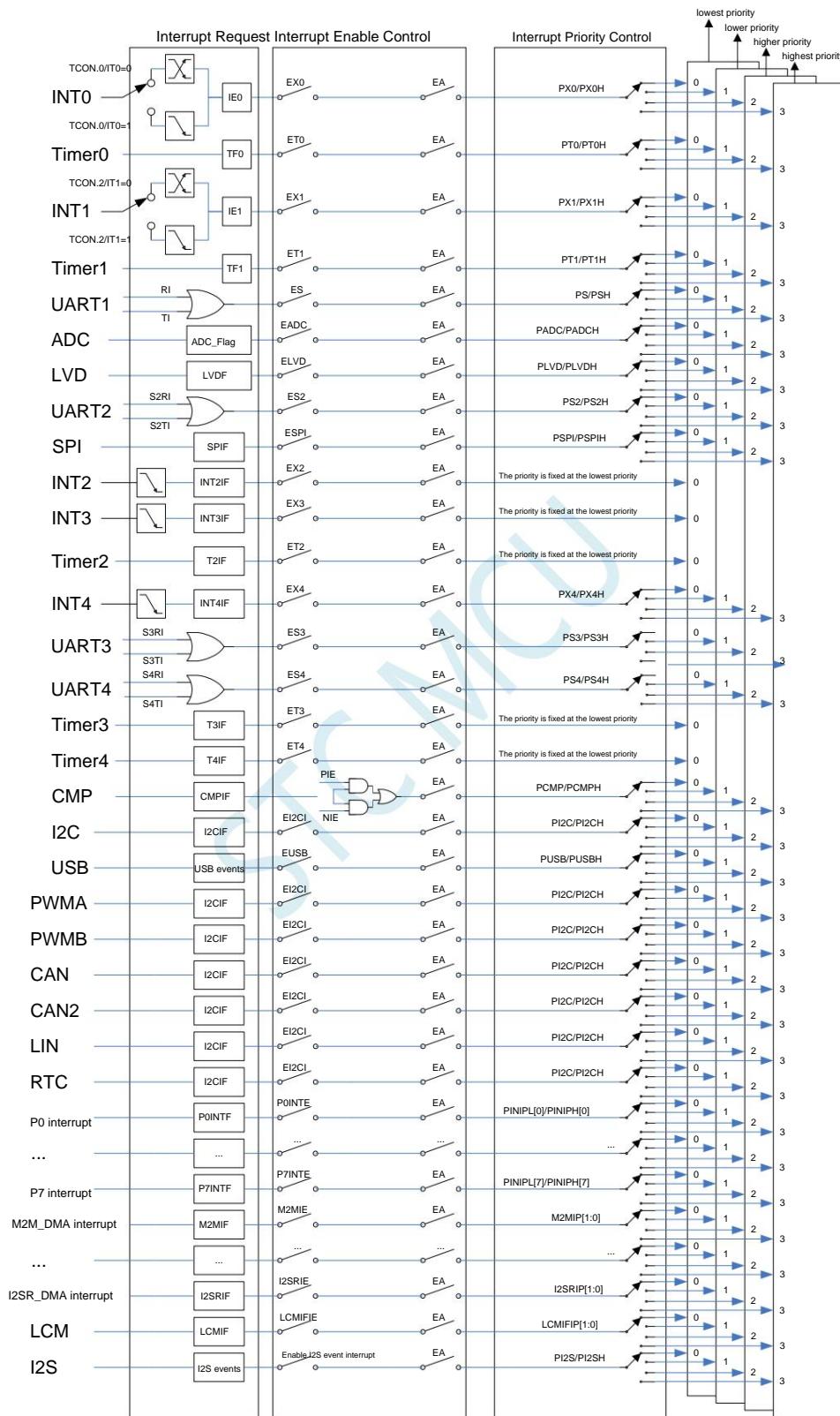
The user can mask the corresponding interrupt request by turning off the total interrupt enable bit (EA/IE.7) or the corresponding interrupt enable bit, or by turning on the The corresponding interrupt enable bit enables the CPU to respond to the corresponding interrupt request. Each interrupt source can be independently controlled by software to enable or disable interrupts. status, and the priority level of some interrupts can be set by software. High-priority interrupt requests can interrupt low-priority interrupts, and vice versa. Priority interrupt requests cannot interrupt higher priority interrupts. When two interrupts of the same priority are generated at the same time, the polling order will be Determines which interrupt the system responds to first.

### 12.1 STC32G series interrupt sources

interrupt source	STC32G12K128 series	STC32G8K64 series	STC32F12K60 series
External Interrupt 0 Interrupt (INT0)	ÿ	ÿ	ÿ
Timer 0 interrupt (Timer0)	ÿ	ÿ	ÿ
External Interrupt 1 Interrupt (INT1)	ÿ	ÿ	ÿ
Timer 1 Interrupt (Timer1)	ÿ	ÿ	ÿ
Serial port 1 interrupt (UART1)	ÿ	ÿ	ÿ
Analog-to-Digital Conversion Interrupt (ADC)	ÿ	ÿ	ÿ
Low Voltage Detect Interrupt (LVD)	ÿ	ÿ	ÿ
Serial port 2 interrupt (UART2)	ÿ	ÿ	ÿ
Serial Peripheral Interface Interrupt (SPI)	ÿ	ÿ	ÿ
External Interrupt 2 Interrupt (INT2)	ÿ	ÿ	ÿ
External Interrupt 3 Interrupt (INT3)	ÿ	ÿ	ÿ
Timer 2 Interrupt (Timer2)	ÿ	ÿ	ÿ
External Interrupt 4 Interrupt (INT4)	ÿ	ÿ	ÿ
Serial port 3 interrupt (UART3)	ÿ	ÿ	ÿ
Serial port 4 interrupt (UART4)	ÿ	ÿ	ÿ
Timer 3 Interrupt (Timer3)	ÿ	ÿ	ÿ

Timer 4 interrupt (Timer4)	ÿ	ÿ	ÿ
Comparator interrupt (CMP)	ÿ	ÿ	ÿ
I2C bus interrupt	ÿ	ÿ	ÿ
USB interrupt	ÿ		ÿ
PWMA	ÿ	ÿ	ÿ
PWMB	ÿ	ÿ	ÿ
CAN interrupt	ÿ	ÿ	ÿ
CAN2 interrupt	ÿ	ÿ	ÿ
LIN interrupt	ÿ	ÿ	ÿ
RTC interrupt	ÿ	ÿ	ÿ
P0 port interrupt	ÿ	ÿ	ÿ
P1 port interrupt	ÿ	ÿ	ÿ
P2 port interrupt	ÿ	ÿ	ÿ
P3 port interrupt	ÿ	ÿ	ÿ
P4 port interrupt	ÿ	ÿ	ÿ
P5 port interrupt	ÿ	ÿ	ÿ
P6 port interrupt	ÿ		
P7 port interrupt	ÿ		
M2M_DMA interrupt	ÿ	ÿ	ÿ
ADC_DMA interrupt	ÿ	ÿ	ÿ
SPI_DMA interrupt	ÿ	ÿ	ÿ
Serial port 1 sends DMA interrupt	ÿ	ÿ	ÿ
Serial port 1 receives DMA interrupt	ÿ	ÿ	ÿ
Serial port 2 sends DMA interrupt	ÿ	ÿ	ÿ
Serial port 2 receives DMA interrupt	ÿ	ÿ	ÿ
Serial port 3 sends DMA interrupt	ÿ	ÿ	ÿ
Serial port 3 receives DMA interrupt	ÿ	ÿ	ÿ
Serial port 4 sends DMA interrupt	ÿ	ÿ	ÿ
Serial port 4 receives DMA interrupt	ÿ	ÿ	ÿ
LCM_DMA interrupt	ÿ	ÿ	ÿ
LCM interrupt	ÿ	ÿ	ÿ
I2C transmit DMA interrupt	ÿ	ÿ	ÿ
I2C receive DMA interrupt	ÿ	ÿ	ÿ
I2S interrupt			ÿ
I2S send DMA interrupt			ÿ
I2S receive DMA interrupt			ÿ

## 12.2 STC32G interrupt structure diagram



## 12.3 STC32G series interrupt list

(Table 1)

interrupt source	interrupt vector		order	priority setting	Priority Interrupt Request Bit	Interrupt Enable 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		PCMP, PCMPH	0/1/2/3	CMPIF	PIE NIE
I2C	FF00C3H	00C3H	see the note	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)

interrupt source	interrupt vector		order	priority setting	Priority	Interrupt Request Bit	Interrupt Enable 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
						SEC1IF	ESECI
						SEC2IF	ESEC2I
						SEC8IF	ESEC8I
						SEC32IF	ESEC32I

(table 3)

interrupt source	interrupt vector		Sequence	Priority Setting	Priority Interrupt Request	Bit	Interrupt Enable	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 interrupt	FF015BH	015BH	43	PINIPL[6], PINIPH[6] 0/1/2/3		P6INTF	P6INTE	
P7 interrupt	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	
I2S interrupt	FF01F3H	01F3H	62	PI2S, PI2SH	0/1/2/3	TXE	TXEIE	
					0/1/2/3	RXNE	RXNEIE	
					0/1/2/3	FRE	ERRIE	
						OVR		
						UDR		
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	

Declare an Interrupt Service Routine in C

void INT0\_Routine(void) interrupt 0;

```
void TM0_Routine(void)           interrupt 1;  
void INT1_Routine(void)          interrupt 2;  
void TM1_Routine(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;
```

The C language interrupt service routine whose interrupt number exceeds 31 cannot be directly declared with interrupt. Please refer to "Development Environment

The processing method of the subsection "About the Interrupt Number Greater than 31 Compilation Errors in Keil" in the "Initial and ISP Download" chapter

Law. Assembly language is not affected

## 12.4 Interrupt Related Registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IE Interrupt	Enable Register	A8H EA		ELVD EADC		ES	ET1	EX1	ET0	EX0 000,0000	
IE2 Interrupt	Enable Register 2	AFH EUSB		ET4	ET3	ES4	ES3	ET2	ESPI	ES2 000,0000	
IP Interrupt	Priority Control Register	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0 x00,0000	
IPH High Interrupt Priority Control Register B7H			-	PLVDH PADCH		PSH	PT1H PX1H PT0H PX0H x000,0000				
IP2 Interrupt	Priority Control Register 2	B5H USB		PI2C	PCMP	PX4	PPWMB PPWMA PSPI			PS2 000,0000	
IP2H High Interrupt Priority Control Register 2 B6H		PUSBH		PI2CH PCMPH		PX4H PPWMBH PPWMAH PSPHI 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 001,0000	
TCON Timer	Control Register	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0 000,0000	
INTCLKO	Interrupt and Clock Output Control Register	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 SMO/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_EPWM					ADC_CHS[3:0]				0000,0000
SPSTAT	SPI Status Register	CDH SPIF		WCOL	-	-	-	-	-	-	0xx,xxxx
CMPCR1	Comparator Control Register 1	E6H CMPEN CMPIF			PIE	NIE	-	-	-	-	CMPDE 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				

symbol	describe	address	Bit addresses and symbols								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
RTCEN	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	I2C Master Control Register	7EFE81H EMSI	-	-	-		MSCMD[3:0]				0xxx,0000
I2CMSST	I2C Master Status Register	7EFE82H MSBUSY MSIF		-	-	-			MSACKI MSACKO 00xx,xx10		
I2CSLCR	I2C Slave Control Register	7EFE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	-	SLRST x000,0xxx
I2CSLST	I2C 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 BF		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 BF		TIF	COMIF CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000	
PWMB_SR2	PWMB Status Register 2	7EFEE6H	-	-	-	CC8OF CC7OF CC6OF CC5OF			-	-	xxx0,000x

P0INTE	Port 0 Interrupt Enable Register	7EF0D0H P07INTE P06INTE P05INTE P04INTE P03INTE P02INTE P01INTE P00INTE 0000,0000							
P1INTE	Port 1 Interrupt Enable Register	7EF0D01H P17INTE P16INTE P15INTE P14INTE P13INTE P12INTE P11INTE P10INTE 0000,0000							
P2INTE	Port 2 Interrupt Enable Register	7EF0D02H P27INTE P26INTE P25INTE P24INTE P23INTE P22INTE P21INTE P20INTE 0000,0000							
P3INTE	Port 3 Interrupt Enable Register	7EF0D03H P37INTE P36INTE P35INTE P34INTE P33INTE P32INTE P31INTE P30INTE 0000,0000							
P4INTE	Port 4 Interrupt Enable Register	7EF0D04H P47INTE P46INTE P45INTE P44INTE P43INTE P42INTE P41INTE P40INTE 0000,0000							
P5INTE	Port 5 Interrupt Enable Register	7EF0D05H - - P55INTE P54INTE P53INTE P52INTE P51INTE P50INTE xx00,0000							
P6INTE	Port 6 Interrupt Enable Register	7EF0D06H P67INTE P66INTE P65INTE P64INTE P63INTE P62INTE P61INTE P60INTE 0000,0000							
P7INTE	Port 7 Interrupt Enable Register	7EF0D07H P77INTE P76INTE P75INTE P74INTE P73INTE P72INTE P71INTE P70INTE 0000,0000							
P0INTF	Port 0 Interrupt Flag Register	7EF0D10H P07INTF P06INTF P05INTF P04INTF P03INTF P02INTF P01INTF P00INTF 0000,0000							
P1INTF	Port 1 Interrupt Flag Register	7EF0D11H P17INTF P16INTF P15INTF P14INTF P13INTF P12INTF P11INTF P10INTF 0000,0000							
P2INTF	Port 2 Interrupt Flag Register	7EF0D12H P27INTF P26INTF P25INTF P24INTF P23INTF P22INTF P21INTF P20INTF 0000,0000							
P3INTF	Port 3 Interrupt Flag Register	7EF0D13H P37INTF P36INTF P35INTF P34INTF P33INTF P32INTF P31INTF P30INTF 0000,0000							
P4INTF	Port 4 Interrupt Flag Register	7EF0D14H P47INTF P46INTF P45INTF P44INTF P43INTF P42INTF P41INTF P40INTF 0000,0000							
P5INTF	Port 5 Interrupt Flag Register	7EF0D15H - - P55INTF P54INTF P53INTF P52INTF P51INTF P50INTF xx00,0000							
P6INTF	Port 6 Interrupt Flag Register	7EF0D16H P67INTF P66INTF P65INTF P64INTF P63INTF P62INTF P61INTF P60INTF 0000,0000							
P7INTF	Port 7 Interrupt Flag Register	7EF0D17H P77INTF P76INTF P75INTF P74INTF P73INTF P72INTF P71INTF P70INTF 0000,0000							
PINIPL	I/O port interrupt priority low register	7EF0D60H P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP 0000,0000
PINIPH	I/O port interrupt priority high register	7EF0D61H P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH 0000,0000
UR1TOCR Serial	port 1 timeout control register	7EF0D70H ENTO ENTOI SCALE			-	-	-	-	- 000x,xxxx
UR1TOSR Serial	port 1 timeout status register	7EF0D71H	-	-	-	-	-	-	TOIF xxxx,xxx0
UR2TOCR Serial	port 2 timeout control register	7EF0D74H ENTO ENTOI SCALE			-	-	-	-	- 000x,xxxx
UR2TOSR Serial	port 2 timeout status register	7EF0D75H	-	-	-	-	-	-	TOIF xxxx,xxx0
UR3TOCR Serial	port 3 timeout control register	7EF0D78H ENTO ENTOI SCALE			-	-	-	-	- 000x,xxxx
UR3TOSR Serial	port 3 timeout status register	7EF0D79H	-	-	-	-	-	-	TOIF xxxx,xxx0
UR4TOCR Serial	port 4 timeout control register	7EF0D7CH ENTO ENTOI SCALE			-	-	-	-	- 000x,xxxx
UR4TOSR Serial	port 4 timeout status register	7EF0D7DH	-	-	-	-	-	-	TOIF xxxx,xxx0
SPITOCR SPI Timeout Control Register	SPITOCR SPI Timeout Control Register	7EF0D80H ENTO ENTOI SCALE			-	-	-	-	- 000x,xxxx
SPITOSR SPI Timeout Status Register	SPITOSR SPI Timeout Status Register	7EF0D81H	-	-	-	-	-	-	TOIF xxxx,xxx0
I2CTOCR I2C Timeout Control Register	I2CTOCR I2C Timeout Control Register	7EF0D84H ENTO ENTOI SCALE			-	-	-	-	- 000x,xxxx
I2CTOSR I2C Timeout Status Register	I2CTOSR I2C Timeout Status Register	7EF0D85H	-	-	-	-	-	-	TOIF xxxx,xxx0
I2SCR	I2S Control Register	7EF0D98H TXIE RXNEIE ERRIE		FRF	-	-	-	TXDMAEN RXDMAEN 0000,xx00	
I2SSR	I2S Status Register	7EF0D99H	-	FRE	BUY	OVR	UDR CHSID	TXE	RXNE x000,0000
DMA_M2M_CFG	M2M_DMA configuration register	7EF0A00H M2MIE		-	TXACO RXACO		M2MIP[1:0]		M2MPTY[1:0] 0x00,0000
DMA_M2M_STA	M2M_DMA Status Register	7EF0A02H		-	-	-	-	-	M2MIF xxxx,xxx0
DMA_ADC_CFG	ADC_DMA Configuration Register	7EF0A10H ADCIE		-	-	-	ADCMIP[1:0]		ADCPTY[1:0] 0xxx,0000
DMA_ADC_STA	ADC_DMA Status Register	7EF0A12H		-	-	-	-	-	ADCIF xxxx,xxx0
DMA_SPI_CFG	SPI_DMA configuration register	7EF0A20H SPIIE ACT_TX ACT_RX		-		SPIIP[1:0]		SPIPTY[1:0]	000x,0000
DMA_SPI_STA	SPI_DMA Status Register	7EF0A22H	-	-	-	-	-	TXOVW RXLOSS SPIIF xxxx,x000	
DMA_UR1T_CFG	UR1T_DMA Configuration Register	7EF0A30H UR1TIE		-	-	-	UR1TIP[1:0]	UR1TPTY[1:0]	0xxx,0000
DMA_UR1T_STA	UR1T_DMA Status Register	7EF0A32H		-	-	-	-	TXOVW	- UR1TIF xxxx,x0x0
DMA_UR1R_CFG	UR1R_DMA Configuration Register	7EF0A38H UR1RIE		-	-	-	UR1RIP[1:0]	UR1RPTY[1:0]	0xxx,0000
DMA_UR1R_STA	UR1R_DMA Status Register	7EF0A3AH		-	-	-	-	-	RXLOSS UR1RIF xxxx,xx00
DMA_UR2T_CFG	UR2T_DMA Configuration Register	7EF0A40H UR2TIE		-	-	-	UR2TIP[1:0]	UR2TPTY[1:0]	0xxx,0000
DMA_UR2T_STA	UR2T_DMA Status Register	7EF0A42H		-	-	-	-	TXOVW	- UR2TIF xxxx,x0x0

DMA_UR2R_CFG	UR2R_DMA Configuration Register 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 Register 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 Register 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 Register 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 Register 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 register 7EFA98H	I2CTIE	-	-	-	-	I2CTIP[1:0]	I2CTTPY[1:0]	0xxx,0000
DMA_I2CT_STA	I2CT_DMA Status Register 7EFA9AH	-	-	-	-	-	TXOVW	-	I2CTIF xxxx,x0x0
DMA_I2CR_CFG	I2CR_DMA Configuration Register 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 register 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 Register 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 Enable Bit)

IE (Interrupt Enable Register)

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

EA: global interrupt enable control bit. The role of the EA is to enable interrupts to form multi-level control. That is, each interrupt source is controlled by the EA first;

Each interrupt source is controlled by its own interrupt enable control bit.

0: CPU blocks all interrupt requests

1: CPU open interrupt

ELVD: Low-voltage detection interrupt enable bit.

0: Disable low-voltage detection interrupt

1: Enable low voltage detection interrupt

EADC: A/D conversion interrupt enable bit.

0: Disable A/D conversion interrupt

1: Enable A/D conversion interrupt

ES: Serial port 1 interrupt enable bit.

0: Disable serial port 1 interrupt

1: Enable serial port 1 interrupt

ET1: Timer/counter T1 overflow interrupt enable bit.

0: Disable T1 interrupt

1: Enable T1 interrupt

EX1: External interrupt 1 interrupt enable bit.

0: Disable INT1 interrupt

1: Enable INT1 interrupt

ET0: Timer/counter T0 overflow interrupt enable bit.

0: Disable T0 interrupt

1: Enable T0 interrupt

EX0: External interrupt 0 interrupt enable bit.

0: Disable INT0 interrupt

1: Enable INT0 interrupt

IE2 (Interrupt Enable Register 2)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
IE2	E7H	EUSB	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

EUSB: USB interrupt enable bit.

0: Disable USB interrupt

1: Enable USB interrupt

ET4: Timer/Counter T4 overflow interrupt enable bit.

0: Disable T4 interrupt

1: Enable T4 interrupt

ET3: Timer/Counter T3 overflow interrupt enable bit.

0: Disable T3 interrupt

1: Enable T3 interrupt

ES4: Serial port 4 interrupt enable bit.

0: Disable serial port 4 interrupt

1: Enable serial port 4 interrupt

ES3: Serial port 3 interrupt enable bit.

0: Disable serial port 3 interrupt

1: Enable serial port 3 interrupt

ET2: Timer/Counter T2 overflow interrupt enable bit.

0: Disable T2 interrupt

1: Enable T2 interrupt

ESPI: SPI interrupt enable bit.

0: Disable SPI interrupt

1: Enable SPI interrupt

ES2: Serial port 2 interrupt enable bit.

0: Disable serial port 2 interrupt

1: Enable serial port 2 interrupt

INTCLKO (External Interrupt and Clock Output Control Register)

Symbol address B7			B6	B5	B4	B3	B2	B1	B0
INTCLKO	95H	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

EX4: External interrupt 4 interrupt enable bit.

0: Disable INT4 interrupt

1: Enable INT4 interrupt

EX3: External interrupt 3 interrupt enable bit.

0: Disable INT3 interrupt

1: Enable INT3 interrupt

EX2: External interrupt 2 interrupt enable bit.

0: Disable INT2 interrupt

1: Enable INT2 interrupt

#### CMPCR1 (Comparator Control Register 1)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	B4H	CMPEN CMPIF	PIE	NIE	-	-	CMPOE CMPRES

PIE: Comparator rising edge interrupt enable bit.

0: Comparator rising edge interrupt disabled

1: Enable comparator rising edge interrupt

NIE: Comparator falling edge interrupt enable bit.

0: Disable comparator falling edge interrupt

1: Enable comparator falling edge interrupt

#### CANICR (CAN Bus Interrupt Control Register)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
CANICR	F1H	PCAN2H CAN2IF CAN2IE PCAN2L PCANH CANIF CANIE PCANL					

CANIE: CAN interrupt enable bit.

0: Disable CAN interrupt

1: Enable CAN interrupt

CAN2IE: CAN2 interrupt enable bit.

0: Disable CAN2 interrupt

1: Enable CAN2 interrupt

#### LINICR (LIN Bus Interrupt Control Register)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
LINICR	F9H	-	-	-	PLINH	LINIF	LINIE PLINL

LINIE: LIN interrupt enable bit.

0: Disable LIN interrupt

1: Enable LIN interrupt

#### LCM Interface Configuration Register

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG 7EFFE50H LCMIFIE	-	LCMIFIP[1:0]	LCMIFDPS[1:0]	D16_D8 M68_I80			

LCMIFIE: LCM interface interrupt enable bit.

0: Disable LCM interface interrupt

1: Enable LCM interface interrupt

#### RTC Interrupt Enable Register

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
RTCIEN	7EFE62H	EALAI EDAYI EHOURI EMINI				ESECI ESEC2I ESEC8I	ESEC32I		

EALAI: Alarm Interrupt Enable Bit

0: Disable the alarm interrupt

1: Enable alarm interrupt

EDAYI: One-day (24-hour) interrupt enable bit

0: Turn off the one-day interrupt

1: Enable one-day interrupt

EHOURI: One Hour (60 Minutes) Interrupt Enable Bit

0: Turn off the hour interrupt

1: Enable hour interrupt

EMINI: One minute (60 seconds) interrupt enable bit

0: Turn off the minute interrupt

1: Enable minute interrupt

ESECI: One-Second Interrupt Enable Bit

0: Disable seconds 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

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
I2CMSCR	7EFE81H	EMSI	-	-	-	-	MSCMD[2:0]		
I2CSLCR	7EFE83H	-	ESTAI ERXI		ETXI	ESTOI	-	-	SLRST

EMSI: I2C master mode interrupt enable bit.

0: Disable I2C master mode interrupt

1: Enable I2C master mode interrupt

ESTAI: I2C slave receive START event interrupt enable bit.

0: Disable I2C slave receiving START event interrupt

1: Enable I2C slave to receive START event interrupt

ERXI: I2C slave receive data completion event interrupt enable bit.

0: Disable I2C slave receive data completion event interrupt

1: Enable I2C slave receive data completion event interrupt

ETXI: I2C slave transmit data completion event interrupt enable bit.

- 0: Disable I2C slave send data completion event interrupt
- 1: Enable I2C slave to send data complete event interrupt

ESTOI: I2C slave receive STOP event interrupt enable bit.

- 0: Disable I2C slave receiving STOP event interrupt
- 1: Enable I2C slave to receive STOP event interrupt

#### PWMA Interrupt Enable Register

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0	
PWMA_IER	7EFFEC4H	BIE	TIE	COMIE	CC4IE		CC3IE	CC2IE	CC1IE	UIE

BIE: PWMA brake interrupt enable bit.

- 0: Disable PWMA brake interrupt
- 1: Enable PWMA brake interrupt

TIE: PWMA trigger interrupt enable bit.

- 0: Disable PWMA trigger interrupt
- 1: Enable PWMA to trigger interrupt

COMIE: PWMA compare interrupt enable bit.

- 0: Disable PWMA compare interrupt
- 1: Enable PWMA compare interrupt

CC4IE: PWMA capture compare channel 4 interrupt enable bit.

- 0: Disable PWMA capture compare channel 4 interrupt
- 1: Enable PWMA to capture compare channel 4 interrupt

CC3IE: PWMA capture compare channel 3 interrupt enable bit.

- 0: Disable PWMA capture compare channel 3 interrupt
- 1: Enable PWMA to capture compare channel 3 interrupt

CC2IE: PWMA capture compare channel 2 interrupt enable bit.

- 0: Disable PWMA capture compare channel 2 interrupt
- 1: Enable PWMA to capture compare channel 2 interrupt

CC1IE: PWMA capture compare channel 1 interrupt enable bit.

- 0: Disable PWMA capture compare channel 1 interrupt
- 1: Enable PWMA to capture compare channel 1 interrupt

UIE: PWMA update interrupt enable bit.

- 0: Disable PWMA update interrupt
- 1: Enable PWMA update interrupt

#### PWMB Interrupt Enable Register

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0	
PWMB_IER	7EFFEE4H	BIE	TIE	COMIE	CC8IE		CC7IE	CC6IE	CC5IE	UIE

BIE: PWMB brake interrupt enable bit.

- 0: Disable PWMB brake interrupt
- 1: Enable PWMB brake interrupt

TIE: PWMB trigger interrupt enable bit.

0: Disable PWMB trigger interrupt

1: Enable PWMB to trigger interrupt

COMIE: PWMB compare interrupt enable bit.

0: Disable PWMB compare interrupt

1: Enable PWMB compare interrupt

CC8IE: PWMB capture compare channel 8 interrupt enable bit.

0: Disable PWMB capture compare channel 8 interrupt

1: Enable PWMB to capture compare channel 8 interrupt

CC7IE: PWMB capture compare channel 7 interrupt enable bit.

0: Disable PWMB capture compare channel 7 interrupt

1: Enable PWMB to capture compare channel 7 interrupt

CC6IE: PWMB capture compare channel 6 interrupt enable bit.

0: Disable PWMB capture compare channel 6 interrupt

1: Enable PWMB to capture compare channel 6 interrupt

CC5IE: PWMB capture compare channel 5 interrupt enable bit.

0: Disable PWMB capture compare channel 5 interrupt

1: Enable PWMB to capture compare channel 5 interrupt

UIE: PWMB update interrupt enable bit.

0: Disable PWMB update interrupt

1: Enable PWMB update interrupt

Port Interrupt Enable Register

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
P0INTE	7EF00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	7EF01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	7EF02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	7EF03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P4INTE	7EF04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	7EF05H	-	-	P55INTE	P54INTE	P53INTE	P52INTE	P51INTE	P50INTE
P6INTE	7EF06H	P67INTE	P66INTE	P65INTE	P64INTE	P63INTE	P62INTE	P61INTE	P60INTE
P7INTE	7EF07H	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

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
UR1TOCR	7EF070H	ENTO	ENTOI	SCALE	-	-	-	-	-

ENTOI: Serial port 1 timeout interrupt enable bit.

0: Disable serial port 1 timeout interrupt

1: Allow serial port 1 timeout interrupt

Serial port 2 timeout control register

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
UR2TOCR 7EFD74H ENTO ENTOI SCALE				-	-	-	-	-

ENTOI: Serial port 2 timeout interrupt enable bit.

0: Disable serial port 2 timeout interrupt

1: Allow serial port 2 timeout interrupt

Serial port 3 timeout control register

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
UR3TOCR 7EFD78H ENTO ENTOI SCALE				-	-	-	-	-

ENTOI: Serial port 1 timeout interrupt enable bit.

0: Disable serial port 3 timeout interrupt

1: Allow serial port 3 timeout interrupt

Serial port 4 timeout control register

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
UR4TOCR 7EFD7CH ENTO ENTOI SCALE				-	-	-	-	-

ENTOI: Serial port 1 timeout interrupt enable bit.

0: Disable serial port 4 timeout interrupt

1: Allow serial port 4 timeout interrupt

**SPI** Timeout Control Register

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
SPITOCSR 7EFD80H ENTO ENTOI SCALE				-	-	-	-	-

ENTOI: SPI timeout interrupt enable bit.

0: Disable SPI timeout interrupt

1: Enable SPI timeout interrupt

**I2C** Timeout Control Register

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
I2CTOCSR 7EFD84H ENTO ENTOI SCALE				-	-	-	-	-

ENTOI: I2C timeout interrupt enable bit.

0: Disable I2C timeout interrupt

1: Enable I2C timeout interrupt

**I2S** Control Register

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
I2SCR	7EF98H	TXEIE RXNEIE	ERRIE		FRF	-	-	TXDMAEN RXDMAEN	

TXEIE: Output buffer empty interrupt enable bit.

0: Disable output buffer empty interrupt

1: Enable output buffer empty interrupt

RXNEIE: Input buffer not empty interrupt enable bit.

0: Disable input buffer non-empty interrupt

1: Enable input buffer non-empty interrupt

ERRIE: Error interrupt enable bit.

0: Disable error interrupt

1: Enable error interrupt

DMA_I2CT_CFG	I2CT_DMA configuration register	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]	I2CTPTY[1:0]	0xxx,0000
DMA_I2CR_CFG	I2CR_DMA Configuration Register	7EFAA0H	I2CRIE	-	-	-	I2CRIP[1:0]	I2CRPTY[1:0]	0xxx,0000
DMA_I2ST_CFG	I2ST_DMA configuration register	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]	I2STPTY[1:0]	0xxx,0000
DMA_I2SR_CFG	I2SR_DMA Configuration Register	7EFAB8H	I2SRIE	-	-	-	I2SRIP[1:0]	I2SRPTY[1:0]	0xxx,0000

#### DMA Interrupt Enable Register

symbol	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]		ADCPPTY[1:0]	
DMA_SPI_CFG	7EFA20H	SPIIIE ACT_RX			-	SPIIIP[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 enable bit.

0: Disable DMA\_M2M interrupt

1: Enable DMA\_M2M interrupt

ADCIE: DMA\_ADC (ADC DMA) interrupt enable bit.

0: Disable DMA\_ADC interrupt

1: Enable DMA\_ADC interrupt

SPIIIE: DMA\_SPI (SPI DMA) interrupt enable bit.

0: Disable DMA\_SPI interrupt

1: Enable DMA\_SPI interrupt

UR1TIE: DMA\_UR1T (serial port 1 transmit DMA) interrupt enable bit.

0: Disable DMA\_UR1T interrupt

1: Enable DMA\_UR1T interrupt

UR1RIE: DMA\_UR1R (serial port 1 receive DMA) interrupt enable bit.

0: Disable DMA\_UR1R interrupt

1: Enable DMA\_UR1R interrupt

UR2TIE: DMA\_UR2T (serial port 2 transmit DMA) interrupt enable bit.

0: Disable DMA\_UR2T interrupt

1: Enable DMA\_UR2T interrupt

UR2RIE: DMA\_UR2R (serial port 2 receive DMA) interrupt enable bit.

0: Disable DMA\_UR2R interrupt

1: Enable DMA\_UR2R interrupt

UR3TIE: DMA\_UR3T (serial port 3 transmit DMA) interrupt enable bit.

0: Disable DMA\_UR3T interrupt

1: Enable DMA\_UR3T interrupt

UR3RIE: DMA\_UR3R (serial port 3 receive DMA) interrupt enable bit.

0: Disable DMA\_UR3R interrupt

1: Enable DMA\_UR3R interrupt

UR4TIE: DMA\_UR4T (serial port 4 transmit DMA) interrupt enable bit.

0: Disable DMA\_UR4T interrupt

1: Enable DMA\_UR4T interrupt

UR4RIE: DMA\_UR4R (serial port 4 receive DMA) interrupt enable bit.

0: Disable DMA\_UR4R interrupt

1: Enable DMA\_UR4R interrupt

LCMIE: DMA\_LCM (LCM interface DMA) interrupt enable bit.

0: Disable DMA\_LCM interrupt

1: Enable DMA\_LCM interrupt

I2CTIE: DMA\_I2CT (I2C transmit DMA) interrupt enable bit.

0: Disable DMA\_I2CT interrupt

1: Enable DMA\_I2CT interrupt

I2CRIE: DMA\_I2CR (I2C receive DMA) interrupt enable bit.

0: Disable DMA\_I2CR interrupt

1: Enable DMA\_I2CR interrupt

I2STIE: DMA\_I2ST (I2S transmit DMA) interrupt enable bit.

0: Disable DMA\_I2ST interrupt

1: Enable DMA\_I2ST interrupt

I2SRIE: DMA\_I2SR (I2S receive DMA) interrupt enable bit.

0: Disable DMA\_I2SR interrupt

1: Enable DMA\_I2SR interrupt

## 12.4.2 Interrupt Request Register (Interrupt Flag Bit)

Timer Control Register

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

TF1: Timer 1 overflow interrupt flag. In the interrupt service routine, the hardware is automatically cleared.

TF0: Timer 0 overflow interrupt flag. In the interrupt service routine, the hardware is automatically cleared.

IE1: External interrupt 1 interrupt request flag. In the interrupt service routine, the hardware is automatically cleared.

IE0: External interrupt 0 interrupt request flag. In the interrupt service routine, the hardware is automatically cleared.

Interrupt Flag Auxiliary Register

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

INT4IF: External interrupt 4 interrupt request flag. In the interrupt service routine, the hardware is automatically cleared.

INT3IF: External interrupt 3 interrupt request flag. In the interrupt service routine, the hardware is automatically cleared.

INT2IF: External interrupt 2 interrupt request flag. In the interrupt service routine, the hardware is automatically cleared.

T4IF: Timer 4 overflow interrupt flag. In the interrupt service routine, the hardware is automatically cleared.

T3IF: Timer 3 overflow interrupt flag. In the interrupt service routine, the hardware is automatically cleared.

T2IF: Timer 2 overflow interrupt flag. In the interrupt service routine, the hardware is automatically cleared.

Serial Control Register

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI
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 sends complete interrupt request flag. Software reset is required.

RI: Serial port 1 receives complete interrupt request flag. Software reset is required.

S2TI: Serial port 2 sends completion interrupt request flag. Software reset is required.

S2RI: Serial port 2 receives complete interrupt request flag. Software reset is required.

S3TI: Serial port 3 sends complete interrupt request flag. Software reset is required.

S3RI: Serial port 3 receives complete interrupt request flag. Software reset is required.

S4TI: Serial port 4 sends complete interrupt request flag. Software reset is required.

S4RI: Serial port 4 receives complete interrupt request flag. Software reset is required.

## Power Management Register

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

LVDF: Low voltage detection interrupt request flag. Software reset is required.

## ADC Control Register

symbol	land site	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 complete interrupt request flag. Software reset is required.

## SPI Status Register

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

SPIF: SPI data transfer complete interrupt request flag. Need software to write "1" to clear.

## Comparator Control Register 1

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
CMPPCR1	E6H	CMPEN	CMPIF	PIE	NIE	-	-	CMPOE	CMPRES

CMPIF: Comparator Interrupt Request Flag. Software reset is required.

## CANICR (CAN Bus Interrupt Control Register)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
CANICR	F1H	PCAN2H	CAN2IF	CAN2IE	PCAN2L	PCANH	CANIF	CANIE	PCANL

CANIF: CAN interrupt request flag. Software reset is required.

CAN2IF: CAN2 interrupt request flag. Software reset is required.

## LINICR (LIN Bus Interrupt Control Register)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LINICR	F9H	-	-	-	-	PLINH	LINIF	LINIE	PLINL

LINIF: LIN interrupt request flag. Software reset is required.

## LCM Interface Status Register

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LCMIFSTA	7EFF53H	-	-	-	-	-	-	-	LCMIFIF

LCMIFIF: LCM interface interrupt request flag. Software reset is required.

#### RTC Interrupt Request Flag Register

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
RTCIF	7EFE63H	ALAIF DAYIF HOURIF MINIF				SECIF	SEC2IF SEC8IF SEC32IF		

ALAIF: Alarm interrupt request bit. Software reset is required.

DAYIF: One-day (24-hour) interrupt request bit. Software reset is required.

HOURIF: One hour (60 minutes) interrupt request bit. Software reset is required.

MINIF: One minute (60 seconds) interrupt request bit. Software reset is required.

SECIF: One second interrupt request bit. Software reset is required.

SEC2IF: 1/2 second interrupt request bit. Need to be cleared by software

SEC8IF: 1/8 second interrupt request bit. Software reset is required.

SEC32IF: 1/32 second interrupt request bit. Software reset is required.

#### I2C Status Register

Symbol	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: I2C Master Mode Interrupt Request Flag. Software reset is required.

ESTAI: I2C slave receive START event interrupt request flag. Software reset is required.

ERXI: I2C slave receive data completion event interrupt request flag. Software reset is required.

ETXI: I2C slave send data completion event interrupt request flag. Software reset is required.

ESTOI: I2C slave receives STOP event interrupt request flag. Software reset is required.

#### PWMA Status Register

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	7EFE C5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	
PWMA_SR2	7EFE C6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-

BIF: PWMA brake interrupt request flag. Software reset is required.

TIF: PWMA trigger interrupt request flag. Software reset is required.

COMIF: PWMA compare interrupt request flag. Software reset is required.

CC4IF: PWMA channel 4 capture compare interrupt request flag. Software reset is required.

CC3IF: PWMA channel 3 capture compare interrupt request flag. Software reset is required.

CC2IF: PWMA channel 2 capture compare interrupt request flag. Software reset is required.

CC1IF: PWMA channel 1 capture compare interrupt request flag. Software reset is required.

UIF: PWMA update interrupt request flag. Software reset is required.

CC4OF: Recapture interrupt request flag occurred on PWMA channel 4. Software reset is required.

CC3OF: Recapture interrupt request flag occurred on PWMA channel 3. Software reset is required.

CC2OF: Recapture interrupt request flag occurred on PWMA channel 2. Software reset is required.

CC1OF: Recapture interrupt request flag occurred on PWMA channel 1. Software reset is required.

#### PWMB Status Register

Symbol address B7	B7	B6	B5	B4	B3	B2	B1	B0
PWMB_SR1 7EFFE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF
PWMB_SR2 7EFFE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

BIF: PWMB brake interrupt request flag. Software reset is required.

TIF: PWMB trigger interrupt request flag. Software reset is required.

COMIF: PWMB compare interrupt request flag. Software reset is required.

CC8IF: PWMB channel 8 capture compare interrupt request flag. Software reset is required.

CC7IF: PWMB channel 7 capture compare interrupt request flag. Software reset is required.

CC6IF: PWMB channel 6 capture compare interrupt request flag. Software reset is required.

CC5IF: PWMB channel 5 capture compare interrupt request flag. Software reset is required.

UIF: PWMB update interrupt request flag. Software reset is required.

CC8OF: Recapture interrupt request flag occurred on PWMB channel 8. Software reset is required.

CC7OF: Recapture interrupt request flag occurred on PWMB channel 7. Software reset is required.

CC6OF: Recapture interrupt request flag occurred on PWMB channel 6. Software reset is required.

CC5OF: Recapture interrupt request flag occurred on PWMB channel 5. Software reset is required.

#### Port Interrupt Flag Register

Symbol address B7	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	7EF010H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF
P1INTF	7EF011H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF
P2INTF	7EF012H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF
P3INTF	7EF013H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF
P4INTF	7EF014H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF
P5INTF	7EF015H	-	-	P55INTF	P54INTF	P53INTF	P52INTF	P51INTF
P6INTF	7EF016H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF
P7INTF	7EF017H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF

PnINTF.x: port interrupt request flag (n=0~7, x=0~7)

0: Pn.x port has no interrupt request

1: The Pn.x port has an interrupt request. If the interrupt is enabled, the interrupt service routine will be entered. Software reset is required.

#### Serial port 1 timeout status register

Symbol address B7	B7	B6	B5	B4	B3	B2	B1	B0
UR1TOSR	7EF071H	-	-	-	-	-	-	TOIF

TOIF: Serial port 1 timeout interrupt request flag. Software reset is required.

## Serial port 2 timeout status register

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
UR2TOSR	7EFD75H	-	-	-	-	-	-	-	TOIF

TOIF: Serial port 2 timeout interrupt request flag. Software reset is required.

## Serial port 3 timeout status register

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
UR3TOSR	7EFD79H	-	-	-	-	-	-	-	TOIF

TOIF: Serial port 3 timeout interrupt request flag. Software reset is required.

## Serial port 4 timeout status register

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
UR4TOSR	7EFD7DH	-	-	-	-	-	-	-	TOIF

TOIF: Serial port 4 timeout interrupt request flag. Software reset is required.

**SPI Timeout Status Register**

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
SPITOSR	7EFD81H	-	-	-	-	-	-	-	TOIF

TOIF: SPI timeout interrupt request flag. Software reset is required.

**I2C Timeout Status Register**

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
I2CTOSR	7EFD75H	-	-	-	-	-	-	-	TOIF

TOIF: I2C timeout interrupt request flag. Software reset is required.

**I2S Timeout Status Register**

Symbol	Address B7		B6	B5	B4	B3	B2	B1	B0
I2STOSR	7EFD79H	-	-	-	-	-	-	-	TOIF

TOIF: I2S timeout interrupt request flag. Software reset is required.

**DMA Interrupt Flag Register**

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
DMA_M2M_STA	7EFA02H	-	-	-	-	-	-	-	M2MIF
DMA_ADC_STA	7EFA12H	-	-	-	-	-	-	-	ADCIF
DMA_SPI_STA	7EFA22H	-	-	-	-	-TXO/VW	RXLOSS\$	SPIIF	
DMA_UR1T_STA	7EFA32H	-	-	-	-	-TXO/VW		-	UR1TIF
DMA_UR1R_STA	7EFA3AH	-	-	-	-	-	-	-	RXLOSS UR1RIF
DMA_UR2T_STA	7EFA42H	-	-	-	-	-TXO/VW		-	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 reset is required.

ADCIF: DMA\_ADC (ADC DMA) interrupt request flag. Software reset is required.

SPIIF: DMA\_SPI (SPI DMA) interrupt request flag. Software reset is required. .

UR1TIF: DMA\_UR1T (serial port 1 sends DMA) interrupt request flag. Software reset is required.

UR1RIF: DMA\_UR1R (serial port 1 receive DMA) interrupt request flag. Software reset is required.

UR2TIF: DMA\_UR2T (serial port 2 send DMA) interrupt request flag. Software reset is required.

UR2RIF: DMA\_UR2R (serial port 2 receive DMA) interrupt request flag. Software reset is required.

UR3TIF: DMA\_UR3T (serial port 3 send DMA) interrupt request flag. Software reset is required.

UR3RIF: DMA\_UR3R (serial port 3 receive DMA) interrupt request flag. Software reset is required.

UR4TIF: DMA\_UR4T (serial port 4 send DMA) interrupt request flag. Software reset is required.

UR4RIF: DMA\_UR4R (serial port 4 receive DMA) interrupt request flag. Software reset is required.

LCMIF: DMA\_LCM (LCM interface DMA) interrupt request flag. Software reset is required.

I2CTIF: DMA\_I2CT (I2C transmit DMA) interrupt request flag. Software reset is required.

I2CRIF: DMA\_I2CR (I2C receive DMA) interrupt request flag. Software reset is required.

I2STIF: DMA\_I2ST (I2S transmit DMA) interrupt request flag. Software reset is required.

I2SRIF: DMA\_I2SR (I2S receive DMA) interrupt request flag. Software reset is required.

### 12.4.3 Interrupt Priority Register

Interrupt Priority Control Register

Symbol	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 bits

00: INT0 interrupt priority is level 0 (the lowest level)

01: INT0 interrupt priority level 1 (lower level)

10: INT0 interrupt priority level 2 (higher)

11: INTO interrupt priority is level 3 (highest)

PT0H, PT0: Timer 0 interrupt priority control bits

00: Timer 0 interrupt priority level 0 (lowest level) 01: Timer 0 interrupt priority level 1 (lower level) 10: Timer 0 interrupt priority level 2 (higher level) 11: Timer 0 Interrupt priority level 3 (highest)

PX1H, PX1: External interrupt 1 interrupt priority control bits

00: INT1 interrupt priority is level 0 (the lowest level)  
01: INT1 interrupt priority level 1 (lower level)  
10: INT1 interrupt priority level 2 (higher)  
11: INT1 interrupt priority level 3 (highest)

PT1H, PT1: Timer 1 interrupt priority control bits

00: Timer 1 interrupt priority is level 0 (lowest level) 01: Timer 1 interrupt priority level is level 1 (lower level) 10: Timer 1 interrupt priority level is level 2 (higher level) 11: Timer 1 Interrupt priority is level 3 (highest)

PSH, PS: Serial port 1 interrupt priority control bit

00: Serial port 1 interrupt priority is level 0 (the lowest level)  
01: Serial port 1 interrupt priority is level 1 (lower level) 10:  
Serial port 1 interrupt priority is level 2 (higher level) 11:  
Serial port 1 interrupt priority is level 3 (highest level)

PADCH, PADC: ADC interrupt priority control bits

00: ADC interrupt priority is level 0 (lowest level)  
01: ADC interrupt priority is level 1 (lower level)  
10: ADC interrupt priority level 2 (higher)  
11: ADC interrupt priority is level 3 (highest)

PLVDH, PLVD: Low-voltage detection interrupt priority control bits

00: LVD interrupt priority is level 0 (lowest level)  
01: LVD interrupt priority is level 1 (lower level)  
10: LVD interrupt priority level 2 (higher)  
11: LVD interrupt priority is level 3 (highest)

PS2H, PS2: Serial port 2 interrupt priority control bits

00: Serial port 2 interrupt priority is level 0 (the lowest level)  
01: Serial port 2 interrupt priority is level 1 (lower level) 10:  
Serial port 2 interrupt priority is level 2 (higher level) 11:  
Serial port 2 interrupt priority is level 3 (the highest level)

PS3H, PS3: Serial port 3 interrupt priority control bits

00: Serial port 3 interrupt priority is level 0 (lowest) 01:  
Serial port 3 interrupt priority is level 1 (lower level) 10:  
Serial port 3 interrupt priority is level 2 (higher) 11: Serial  
port 3 interrupt priority is level 3 (the highest level)

PS4H, PS4: Serial port 4 interrupt priority control bits

00: Serial port 4 interrupt priority is level 0 (the lowest level)  
01: Serial port 4 interrupt priority is level 1 (lower level)

10: Serial port 4 interrupt priority is level 2 (higher) 11:

Serial port 4 interrupt priority is level 3 (highest)

PSPIH, PSPI: SPI interrupt priority control bits

00: SPI interrupt priority is level 0 (lowest level)

01: SPI interrupt priority level 1 (lower level)

10: SPI interrupt priority level 2 (higher)

11: SPI interrupt priority is level 3 (highest)

PPWMAH,PPWMA: Advanced PWMA interrupt priority control bit 00:

Advanced PWMA interrupt priority is 0 (lowest) 01: Advanced

PWMA interrupt priority is 1 (lower) 10: Advanced PWMA interrupt priority is 2 (higher level) 11: Advanced PWMA interrupt priority 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 is level 1 (lower level) 10:

Advanced PWMB interrupt priority is level 2 (higher level) 11:

Advanced PWMB interrupt priority is level 3 (highest level)

PX4H, PX4: External interrupt 4 interrupt priority control bits

00: INT4 interrupt priority is level 0 (the lowest level)

01: INT4 interrupt priority level 1 (lower level)

10: INT4 interrupt priority level 2 (higher)

11: INT4 interrupt priority is level 3 (highest)

PCMPH, PCMP: comparator interrupt priority control bits

00: CMP interrupt priority is level 0 (lowest level)

01: CMP interrupt priority level 1 (lower level)

10: CMP interrupt priority level 2 (higher)

11: CMP interrupt priority level 3 (highest)

PI2CH, PI2C: I2C interrupt priority control bits

00: I2C interrupt priority is level 0 (lowest level)

01: I2C interrupt priority level 1 (lower level)

10: I2C interrupt priority level 2 (higher)

11: I2C interrupt priority level 3 (highest)

PUSBH, PUSB: USB interrupt priority control bits

00: USB interrupt priority is level 0 (lowest level)

01: USB interrupt priority level 1 (lower level)

10: USB interrupt priority level 2 (higher)

11: USB interrupt priority level 3 (highest)

PRTCH, PRTC: RTC interrupt priority control bits

00: RTC interrupt priority is level 0 (lowest level)

01: RTC interrupt priority level 1 (lower level)

10: RTC interrupt priority is level 2 (higher level)

11: RTC interrupt priority is level 3 (highest)

PI2SH, PI2S: I2S interrupt priority control bits

00: I2S interrupt priority is level 0 (the lowest level)

- 01: I2S interrupt priority is level 1 (lower level)
- 10: I2S interrupt priority is level 2 (higher level)
- 11: I2S interrupt priority is level 3 (highest)

**CANICR (CAN Bus Interrupt Control Register)**

Symbol 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 bits

- 00: CAN interrupt priority is 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)

PCAN2H, PCAN2L: CAN2 interrupt priority control bits

- 00: CAN2 interrupt priority is level 0 (the lowest level)
- 01: CAN2 interrupt priority is level 1 (lower level)
- 10: CAN2 interrupt priority is level 2 (higher level)
- 11: CAN2 interrupt priority is level 3 (highest)

**LINICR (LIN Bus Interrupt Control Register)**

Symbol address B7		B6	B5	B4	B3	B2	B1	B0	
LINICR	F9H	-	-	-	-	PLINH	LINIF	LINIE	PLINL

PLINH, PLINL: LIN interrupt priority control bits

- 00: LIN interrupt priority is level 0 (lowest level)
- 01: LIN interrupt priority level 1 (lower level)
- 10: LIN interrupt priority level 2 (higher)
- 11: LIN interrupt priority is level 3 (highest)

**LCM Interface Configuration Register**

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFFE50H 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 (the lowest level)
- 01: LCM interface interrupt priority is level 1 (lower level)
- 10: LCM interface interrupt priority is level 2 (higher level)
- 11: LCM interface interrupt priority is level 3 (highest)

**Port Interrupt Priority Control Register**

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
PINIPL	7EF60H P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP
PINIPH	7EF61H P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH

P0IPH, P0IP: P0 port interrupt priority control bits

- 00: P0 port interrupt priority is level 0 (the lowest level)
- 01: P0 port interrupt priority is level 1 (lower level)
- 10: P0 port interrupt priority is level 2 (higher level)
- 11: P0 port interrupt priority is level 3 (the highest level)

P1IPH, P1IP: P1 port interrupt priority control bits

- 00: P1 port interrupt priority is level 0 (the lowest level)
- 01: P1 port interrupt priority is level 1 (lower level)
- 10: P1 port interrupt priority is level 2 (higher level)
- 11: P1 port interrupt priority is level 3 (highest level)

P2IPH, P2IP: P2 port interrupt priority control bits

- 00: P2 port interrupt priority is level 0 (the lowest level)
- 01: P2 port interrupt priority is level 1 (lower level)
- 10: P2 port interrupt priority is level 2 (higher level)
- 11: P2 port interrupt priority is level 3 (highest level)

P3IPH, P3IP: P3 port interrupt priority control bits

- 00: P3 port interrupt priority is level 0 (the lowest level)
- 01: P3 port interrupt priority is level 1 (lower level)
- 10: P3 port interrupt priority is 2 (higher)
- 11: P3 port interrupt priority is level 3 (highest level)

P4IPH, P4IP: P4 port interrupt priority control bits

- 00: P4 port interrupt priority is level 0 (the lowest level)
- 01: P4 port interrupt priority is level 1 (lower level)
- 10: P4 port interrupt priority is 2 (higher)
- 11: P4 port interrupt priority is level 3 (highest level)

P5IPH, P5IP: P5 port interrupt priority control bits

- 00: P5 port interrupt priority is level 0 (the lowest level)
- 01: P5 port interrupt priority is level 1 (lower level)
- 10: P5 port interrupt priority is level 2 (higher level)
- 11: P5 port interrupt priority is level 3 (highest level)

P6IPH, P6IP: P6 port interrupt priority control bits

- 00: P6 port interrupt priority is level 0 (the lowest level)
- 01: P6 port interrupt priority is level 1 (lower level)
- 10: P6 port interrupt priority is level 2 (higher level)
- 11: P6 port interrupt priority is level 3 (highest level)

P7IPH, P7IP: P7 port interrupt priority control bits

- 00: P7 port interrupt priority is level 0 (the lowest level)
- 01: P7 port interrupt priority is level 1 (lower level)
- 10: P7 port interrupt priority is 2 (higher)
- 11: P7 port interrupt priority is level 3 (highest level)

#### DMA Interrupt Priority Control Register

symbol	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]
DMA_SPI_CFG_7EFA20H_SPIIE_ACT	TX ACT_RX		-	SPIIP[1:0]	SPIPTY[1:0]
DMA_UR1T_CFG_7EFA30H_UR1TIE		-	-	-	UR1TIP[1:0]
DMA_UR1R_CFG_7EFA38H_UR1RIE		-	-	-	UR1RIP[1:0]
DMA_UR2T_CFG_7EFA40H_UR2TIE		-	-	-	UR2TIP[1:0]
DMA_UR2R_CFG_7EFA48H_UR2RIE		-	-	-	UR2RIP[1:0]
DMA_UR3T_CFG_7EFA50H_UR3TIE		-	-	-	UR3TIP[1:0]
DMA_UR3R_CFG_7EFA58H_UR3RIE		-	-	-	UR3RIP[1:0]
DMA_UR4T_CFG_7EFA60H_UR4TIE		-	-	-	UR4TIP[1:0]
DMA_UR4R_CFG_7EFA68H_UR4RIE		-	-	-	UR4RIP[1:0]
DMA_LCM_CFG_7EFA70H_LCMIE		-	-	-	LCMIP[1:0]
DMA_I2CT_CFG_7EFA98H_I2CTIE		-	-	-	I2CTIP[1:0]
DMA_I2CR_CFG_7EFAA0H_I2CRIE		-	-	-	I2CRIP[1:0]
DMA_I2ST_CFG_7EFAB0H_I2STIE		-	-	-	I2STIP[1:0]
DMA_I2SR_CFG_7EFAB8H_I2SRIE		-	-	-	I2SRIP[1:0]

M2MIP: DMA\_M2M (memory-to-memory DMA) interrupt priority control bits

- 00: DMA\_M2M interrupt priority is level 0 (the lowest level)
- 01: DMA\_M2M interrupt priority is level 1 (lower level)
- 10: DMA\_M2M interrupt priority level 2 (higher level)
- 11: DMA\_M2M interrupt priority is level 3 (highest)

ADCIP: DMA\_ADC (ADC DMA) interrupt priority control bits

- 00: DMA\_ADC interrupt priority is level 0 (the lowest level)
- 01: DMA\_ADC interrupt priority is level 1 (lower level)
- 10: DMA\_ADC interrupt priority level 2 (higher level)
- 11: DMA\_ADC interrupt priority is level 3 (highest)

SPIIP: DMA\_SPI (SPI DMA) interrupt priority control bits

- 00: DMA\_SPI interrupt priority is level 0 (the lowest level)
- 01: DMA\_SPI interrupt priority level 1 (lower level)
- 10: DMA\_SPI interrupt priority level 2 (higher level)
- 11: DMA\_SPI interrupt priority level 3 (highest)

UR1TIP: DMA\_UR1T (serial port 1 sends DMA) interrupt priority control bit

- 00: DMA\_UR1T interrupt priority is level 0 (the lowest level)
- 01: DMA\_UR1T interrupt priority is level 1 (lower level)
- 10: DMA\_UR1T interrupt priority level 2 (higher level)
- 11: DMA\_UR1T interrupt priority is level 3 (highest)

UR1RIP: DMA\_UR1R (serial port 1 receive DMA) interrupt priority control bit

- 00: DMA\_UR1R interrupt priority is level 0 (the lowest level)
- 01: DMA\_UR1R interrupt priority is level 1 (lower level)
- 10: DMA\_UR1R interrupt priority level 2 (higher level)
- 11: DMA\_UR1R interrupt priority is level 3 (highest)

UR2TIP: DMA\_UR2T (serial port 2 send DMA) interrupt priority control bit

- 00: DMA\_UR2T interrupt priority is level 0 (the lowest level)
- 01: DMA\_UR2T interrupt priority is level 1 (lower level)

10: DMA\_UR2T interrupt priority level 2 (higher level)

11: DMA\_UR2T interrupt priority is level 3 (highest)

UR2RIP: DMA\_UR2R (serial port 2 receive DMA) interrupt priority control bit

00: DMA\_UR2R interrupt priority is level 0 (the lowest level)

01: DMA\_UR2R interrupt priority is level 1 (lower level)

10: DMA\_UR2R interrupt priority level 2 (higher)

11: DMA\_UR2R interrupt priority is level 3 (highest)

UR3TIP: DMA\_UR3T (serial port 3 send DMA) interrupt priority control bit

00: DMA\_UR3T interrupt priority is level 0 (the lowest level)

01: DMA\_UR3T interrupt priority is level 1 (lower level)

10: DMA\_UR3T interrupt priority level 2 (higher level)

11: DMA\_UR3T interrupt priority is level 3 (highest)

UR3RIP: DMA\_UR3R (serial port 3 receive DMA) interrupt priority control bit

00: DMA\_UR3R interrupt priority is level 0 (the lowest level)

01: DMA\_UR3R interrupt priority is level 1 (lower level)

10: DMA\_UR3R interrupt priority level 2 (higher level)

11: DMA\_UR3R interrupt priority is level 3 (highest)

UR4TIP: DMA\_UR4T (serial port 4 transmit DMA) interrupt priority control bit

00: DMA\_UR4T interrupt priority is level 0 (the 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 is level 3 (highest)

UR4RIP: DMA\_UR4R (serial port 4 receive DMA) interrupt priority control bit

00: DMA\_UR4R interrupt priority is level 0 (the 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 is level 3 (highest)

LCMIP: DMA\_LCM (LCM interface DMA) interrupt priority control bit

00: DMA\_LCM interrupt priority is level 0 (the lowest level)

01: DMA\_LCM interrupt priority level 1 (lower level)

10: DMA\_LCM interrupt priority level 2 (higher)

11: DMA\_LCM interrupt priority is level 3 (highest)

I2CTIP: DMA\_I2CT (I2C transmit DMA) interrupt priority control bit

00: DMA\_I2CT interrupt priority is level 0 (the 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 is level 3 (highest)

I2CRIP: DMA\_I2CR (I2C receive DMA) interrupt priority control bit

00: DMA\_I2CR interrupt priority is level 0 (the 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 is level 3 (highest)

I2STIP: DMA\_I2ST (I2S transmit DMA) interrupt priority control bit

00: DMA\_I2ST interrupt priority is level 0 (the 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 is level 3 (highest)

I2SRIP: DMA\_I2SR (I2S receive DMA) interrupt priority control bit

00: DMA\_I2SR interrupt priority is level 0 (the lowest level)

01: DMA\_I2SR interrupt priority is level 1 (lower level)

10: DMA\_I2SR interrupt priority level 2 (higher level)

11: DMA\_I2SR interrupt priority is level 3 (highest)

## 12.5 Example Program

### 12.5.1 INT0 interrupt (rising and falling), which can support both rising and falling

along

---

//The test frequency is **11.0592MHz**

```

//#include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

void INT0_Isr() interrupt 0
{
    if (INT0)                                     // Determine rising and falling edges
    {
        P10 = !P10;                               //test port
    }
    else
    {
        P11 = !P11;                               //test port
    }
}

void main()
{
    EA=1;                                         //Enable      XFR
    CKCON=0x00;                                    //access to set external data bus speed to fastest
    WTST=0x00;                                     //Set the program code to wait for parameters,
                                                    //assign to      CPU   The speed of executing the program is set to the fastest

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

    IT0 = 0;                                       //enable INT0  Rising and falling edge interrupts
    EX0 = 1;                                       //enable INT0  interrupt
    EA = 1;

    while (1);
}

```

---

## 12.5.2 INT0 interrupt (falling edge)

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void INT0_Isr() interrupt 0
{
    P10 = !P10;                                         //test port
}

void main()
{
    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to      CPU The speed of executing the program is set to the fastest

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

    IT0 = 1;                                            //enable INT0 falling edge interrupt
    EX0 = 1;                                            //enable INT0 interrupt
    EA = 1;

    while (1);
}

```

---

## 12.5.3 INT1 interrupt (rising and falling), which can support both rising and falling

along

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void INT1_Isr() interrupt
2 {

```

```

if (INT1)                                //Determine rising and falling edges
{
    P10 = !P10;                           //test port
}
else
{
    P11 = !P11;                           //test port
}

void main()
{
    EAXFR = 1;                          //Enable      XFR
    CKCON = 0x00;                      //access to set external data bus speed to fastest
    WTST = 0x00;                        //Set the program code to wait for parameters,
                                         //assign to      CPU The speed of executing the program is set to the fastest

    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)

---

```

//The test frequency is 11.0592MHz

##include "stc8h.h"
#include "stc32g.h"                      //For header files, see Download Software
#include "intrins.h"

void INT1_Isr() interrupt 2
{
    P10 = !P10;                           //test port
}

void main()
{
    EAXFR = 1;                          //Enable      XFR
    CKCON = 0x00;                      //access to set external data bus speed to fastest
    WTST = 0x00;                        //Set the program code to wait for parameters,
                                         //assign to      CPU The speed of executing the program is set to the fastest
}

```

---

//assign as 0 can be **CPU** The speed of executing the program is set to the fastest

```

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 supports falling edge interrupt

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void INT2_Isr() interrupt 10
{
    P10 = !P10; //test port
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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 = ; //EnableINT2 interrupt
EA = 1;

while (1);
}

```

## 12.5.6 INT3 interrupt (falling edge), only supports falling edge interrupt

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void INT3_Isr() interrupt 11
{
    P10 = !P10; //test port
}

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    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; //EnableINT3 interrupt
    EA = 1;

    while (1);
}

```

## 12.5.7 INT4 interrupt (falling edge), only supports falling edge interrupt

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software

```

```

#include "intrins.h"

void INT4_Isr() interrupt 16
{
    P10 = !P10;                                //test port

}

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                              //Set the program code to wait for parameters,
                                              //assign to      CPU The speed of executing the program is set to the fastest

    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;                                  //EnableINT4 interrupt
    EA = 1;

    while (1);
}

```

## 12.5.8 Timer 0 Interrupt

---

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                           //For header files, see Download Software
#include "intrins.h"

void TMO_Isr() interrupt
1 {
    P10 = !P10;                                //test port

}

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                              //Set the program code to wait for parameters,
                                              //assign to      CPU The speed of executing the program is set to the fastest

    P0M0 = 0x00;
    P0M1 = 0x00;

```

```

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

TMOD = 0x00;
TL0 = 0x66;                                //65536-11.0592M/12/1000
TH0 = 0xfc;
TR0 = 1;                                     //start timer
ET0 = 1;                                     //Enable timer interrupt
EA = 1;

while (1);
}

```

## 12.5.9 Timer 1 Interrupt

---

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h"                           // For header files, see Download Software
#include "intrins.h"

void TM1_Isr() interrupt
3 {
    P10 = !P10;                               //test port
}

void main()
{
    EAXFR = 1;                             //Enable      XFR
    CKCON = 0x00;                          //access to set external data bus speed to fastest
    WTST = 0x00;                           //Set the program code to wait for parameters,
                                         //assign to      CPU The speed of executing the program is set to the fastest

    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

---

//The test frequency is **11.0592MHz**

```

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void TM2_Isr() interrupt 12
{
    P10 = !P10; //test port
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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 timer
    ET2 = 1; //Enable timer interrupt
    EA = 1;

    while (1);
}

```

---

## 12.5.11 Timer 3 Interrupts

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void TM3_Isr() interrupt 19
{
    P10 = !P10;                                         //test port
}

void main()
{
    EAXFR = 1;                                         //enable access XFR
    CKCON = 0x00;                                       //Set the external data bus speed to the fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to CPU The speed of executing the program is set to the fastest

    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 timer
    ET3 = 1;                                            //Enable timer interrupt
    EA = 1;

    while (1);
}

```

---

## 12.5.12 Timer 4 Interrupts

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void TM4_Isr() interrupt 20
{
    P10 = !P10;                                         //test port
}

```

---

```

}

void main()
{
    EAXFR = 1;                                //enable access XFR
    CKCON = 0x00;                             //Set the external data bus speed to the fastest
    WTST = 0x00;                             //Set the program code to wait for parameters,
                                                //assign to CPU The speed of executing the program is set to the fastest

    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;                               //start timer
    T4R = 1;                                  //Enable timer interrupt
    ET4 = 1;
    EA = 1;

    while (1);
}

```

## 12.5.13 UART1 interrupt

---

```

//The test frequency is 11.0592MHz

##include "stc8h.h"
#include "stc32g.h"                           //For header files, see Download Software
#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; //Enable      XFR
CKCON = 0x00; //access to set external data bus speed to fastest
WTST = 0x00; //Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

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;
S1BRT = 1;
T2x12 = 1; //start timer
T2R = 1; //Enable serial interrupt
ES = 1;
EA = 1;
SBUF = 0x5a; //Send test data

while (1);
}

```

## 12.5.14 UART2 Interrupt

---

//The test frequency is 11.0592MHz

```

##include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void UART2_Isr() interrupt
8 {
    if (S2TI)
    {
        S2TI = 0; //clear interrupt flag
        P12 = !P12; //test port
    }
    if (S2RI)
    {
        S2RI = 0; //clear interrupt flag
        P13 = !P13; //test port
    }
}

void main()
{
}

```

```

EAXFR = 1; //Enable      XFR
CKCON = 0x00; //access to set external data bus speed to fastest
WTST = 0x00; //Set the program code to wait for parameters,
//assign to      CPU The speed of executing the program is set to the fastest

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 interrupt
EA = 1; //Send test data
S2BUF = 0x5a;

while (1);
}

```

## 12.5.15 UART3 Interrupt

---

//The test frequency is 11.0592MHz

```

//##include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void UART3_Isr() interrupt 17
{
    if (S3TI)
    {
        S3TI = 0; //clear interrupt flag
        P12 = !P12; //test port
    }
    if (S3RI)
    {
        S3RI = 0; //clear interrupt flag
        P13 = !P13; //test port
    }
}

void main()
{
    EAXFR = 1; //Enable      XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
}

```

```
//assign as 0 can be CPU The speed of executing the program is set to the fastest

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; //65536-11059200/115200/4=0FFE8H
T2L = 0xe8;
T2H = 0xff;
T2x12 = 1; T2R = 1; //start timer
ES3 = 1; //Enable serial interrupt
EA = 1; //Send test data
S3BUF = 0x5a;

while (1);
}
```

---

## 12.5.16 UART4 interrupt

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void UART4_Isr() interrupt 18
{
    if (S4TI)
    {
        S4TI = 0; //clear interrupt flag
        P12 = !P12; //test port
    }
    if (S4RI)
    {
        S4RI = 0; //clear interrupt flag
        P13 = !P13; //test port
    }
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    P0M0 = 0x00;
```

---

```

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

S4CON = 0x10;                                // 65536-11059200/115200/4=0FFE8H
T2L = 0xe8;
T2H = 0xff;
T2x12 = 1; T2R = 1;                          // start timer
ES4 = 1;                                      // Enable serial interrupt
EA = 1;                                       // Send test data
S4BUF = 0x5a;

while (1);
}

```

## 12.5.17 ADC Interrupts

```

//The test frequency is 11.0592MHz

##include "stc8h.h"
#include "stc32g.h"                            // For header files, see Download Software
#include "intrins.h"

void ADC_Isr() interrupt
{
    ADC_FLAG = 0;                            // clear interrupt flag
    P0 = ADC_RES;                           // test port
    P2 = ADC_RESL;                          // test port
}

void main()
{
    EAXFR = 1;                             // Enable XFR
    CKCON = 0x00;                           // access to set external data bus speed to fastest
    WTST = 0x00;                            // Set the program code to wait for parameters,
                                            // assign to CPU The speed of executing the program is set to the fastest

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

```

```

P5M0 = 0x00;
P5M1 = 0x00;

ADCCFG = 0x00;
ADC_CONTR = 0xc0;           //Enable and start ADC module
EADC = 1;                  //enable interrupt
EA = 1;

while (1);
}

```

## 12.5.18 LVD Interrupts

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"           //For header files, see Download Software
#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      XFR
    CKCON = 0x00;           //access to set external data bus speed to fastest
    WTST = 0x00;             //Set the program code to wait for parameters,
                           //assign to      CPU The speed of executing the program is set to the fastest

    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;               //The interrupt flag needs to be cleared at power-on
    RSTCFG = LVD3V0;         //Set voltage to enable 3.0V
    ELVD = 1;                //interrupt LVD
    EA = 1;
}

```

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

---

### 12.5.19 Comparator Interrupt

---

//The test frequency is 11.0592MHz

```
///#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void CMP_Isr() interrupt 21 {
    CMPIF = 0; //clear interrupt flag
    P10 = !P10; //test port
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    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 Interrupts

---

//The test frequency is 11.0592MHz

```
///#include "stc8h.h"
```

```

#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void SPI_Isr() interrupt
{
    SPIF = 1;                                         //clear interrupt flag
    P10 = !P10;                                       //test port
}

void main()
{
    EAXFR = 1;                                       //Enable XFR
    CKCON = 0x00;                                     //access to set external data bus speed to fastest
    WTST = 0x00;                                      //Set the program code to wait for parameters,
                                                       //assign to CPU The speed of executing the program is set to the fastest

    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 mode
    SPSTAT = 0xc0;                                     //clear interrupt flag
    ESPI = 1;                                         //enable SPI
    EA = 1;                                           //Send test data
    SPDAT = 0x5a;

    while (1);
}

```

## 12.5.21 I2C Interrupts

---

```

//The test frequency is 11.0592MHz

##include "stc8.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void I2C_Isr() interrupt
{
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                            //clear interrupt flag
        P10 = !P10;                                  //test port
    }
}

```

```
void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                             //Set the program code to wait for parameters,
                                              //assign to      CPU The speed of executing the program is set to the fastest

    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 interrupt

    EA = 1;                                  //send start command

    I2CMSCR = 0x81;                            //send start command

    while (1);
}
```

## 13 Ordinary I/O ports can be interrupted, not traditional external interrupts

product line	I/O interrupt I/O interrupt priority I/O interrupt wake-up function		
STC32G12K128 series	●	Level 4	●
STC32G8K64 series	●	Level 4	●
STC32F12K60 series	●	Level 4	●

The STC32G series supports all I/O interrupts and supports 4 interrupt modes: falling edge interrupt, rising edge interrupt, low level interrupt, High level interrupt. Each group of I/O ports has an independent interrupt entry address, and each I/O can independently set the interrupt mode.

Note: The falling edge interrupt and rising edge interrupt of the ordinary I/O port of the **STC32G12K128-Beta** chip should not be used temporarily

### 13.1 I/O port interrupt related registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE Port 0	Interrupt Enable Register 7EFD00H	P07INTE P06INTE P05INTE P04INTE P03INTE P02INTE P01INTE P00INTE	0000,0000								
P1INTE Port 1	Interrupt Enable Register 7EFD01H	P17INTE P16INTE P15INTE P14INTE P13INTE P12INTE P11INTE P10INTE	0000,0000								
P2INTE Port 2	Interrupt Enable Register 7EFD02H	P27INTE P26INTE P25INTE P24INTE P23INTE P22INTE P21INTE P20INTE	0000,0000								
P3INTE Port 3	Interrupt Enable Register 7EFD03H	P37INTE P36INTE P35INTE P34INTE P33INTE P32INTE P31INTE P30INTE	0000,0000								
P4INTE Port 4	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 Port 6	Interrupt Enable Register 7EFD06H	P67INTE P66INTE P65INTE P64INTE P63INTE P62INTE P61INTE P60INTE	0000,0000								
P7INTE Port 7	Interrupt Enable Register 7EFD07H	P77INTE P76INTE P75INTE P74INTE P73INTE P72INTE P71INTE P70INTE	0000,0000								
P0INTF Port 0	interrupt flag register 7EFD10H	P07INTF P06INTF P05INTF P04INTF P03INTF P02INTF P01INTF P00INTF	0000,0000								
P1INTF Port 1	Interrupt Flag Register 7EFD11H	P17INTF P16INTF P15INTF P14INTF P13INTF P12INTF P11INTF P10INTF	0000,0000								
P2INTF Port 2	Interrupt Flag Register 7EFD12H	P27INTF P26INTF P25INTF P24INTF P23INTF P22INTF P21INTF P20INTF	0000,0000								
P3INTF Port 3	Interrupt Flag Register 7EFD13H	P37INTF P36INTF P35INTF P34INTF P33INTF P32INTF P31INTF P30INTF	0000,0000								
P4INTF Port 4	Interrupt Flag Register 7EFD14H	P47INTF P46INTF P45INTF P44INTF P43INTF P42INTF P41INTF P40INTF	0000,0000								
P5INTF Port 5	interrupt flag register 7EFD15H	- - -	P55INTF P54INTF P53INTF P52INTF P51INTF P50INTF	xx00,0000							
P6INTF Port 6	Interrupt Flag Register 7EFD16H	P67INTF P66INTF P65INTF P64INTF P63INTF P62INTF P61INTF P60INTF	0000,0000								
P7INTF Port 7	Interrupt Flag Register 7EFD17H	P77INTF P76INTF P75INTF P74INTF P73INTF P72INTF P71INTF P70INTF	0000,0000								
P0IM0 Port 0	Interrupt Mode Register 0	7EFD20H P07IM0 P06IM0 P05IM0 P04IM0 P03IM0 P02IM0 P01IM0 P00IM0	0000,0000								
P1IM0 Port 1	Interrupt Mode Register 0	7EFD21H P17IM0 P16IM0 P15IM0 P14IM0 P13IM0 P12IM0 P11IM0 P10IM0	0000,0000								
P2IM0 Port 2	Interrupt Mode Register 0	7EFD22H P27IM0 P26IM0 P25IM0 P24IM0 P23IM0 P22IM0 P21IM0 P20IM0	0000,0000								
P3IM0 Port 3	Interrupt Mode Register 0	7EFD23H P37IM0 P36IM0 P35IM0 P34IM0 P33IM0 P32IM0 P31IM0 P30IM0	0000,0000								
P4IM0 Port 4	Interrupt Mode Register 0	7EFD24H P47IM0 P46IM0 P45IM0 P44IM0 P43IM0 P42IM0 P41IM0 P40IM0	0000,0000								
P5IM0 Port 5	Interrupt Mode Register 0	7EFD25H - - -	P55IM0 P54IM0 P53IM0 P52IM0 P51IM0 P50IM0	xx00,0000							
P6IM0 Port 6	Interrupt Mode Register 0	7EFD26H P67IM0 P66IM0 P65IM0 P64IM0 P63IM0 P62IM0 P61IM0 P60IM0	0000,0000								
P7IM0 Port 7	Interrupt Mode Register 0	7EFD27H P77IM0 P76IM0 P75IM0 P74IM0 P73IM0 P72IM0 P71IM0 P70IM0	0000,0000								
P0IM1	Port 0 Interrupt Mode Register 1	7EFD30H P07IM1 P06IM1 P05IM1 P04IM1 P03IM1 P02IM1 P01IM1 P00IM1	0000,0000								

P1IM1	Port 1 Interrupt Mode Register 1	7EF31H P17IM1 P16IM1 P15IM1 P14IM1 P13IM1 P12IM1 P11IM1 P10IM1 0000,0000							
P2IM1	Port 2 Interrupt Mode Register 1	7EF32H P27IM1 P26IM1 P25IM1 P24IM1 P23IM1 P22IM1 P21IM1 P20IM1 0000,0000							
P3IM1	Port 3 Interrupt Mode Register 1	7EF33H P37IM1 P36IM1 P35IM1 P34IM1 P33IM1 P32IM1 P31IM1 P30IM1 0000,0000							
P4IM1	Port 4 Interrupt Mode Register 1	7EF34H P47IM1 P46IM1 P45IM1 P44IM1 P43IM1 P42IM1 P41IM1 P40IM1 0000,0000							
P5IM1	Port 5 Interrupt Mode Register 1	7EF35H - - P55IM1 P54IM1 P53IM1 P52IM1 P51IM1 P50IM1 xx00,0000							
P6IM1	Port 6 Interrupt Mode Register 1	7EF36H P67IM1 P66IM1 P65IM1 P64IM1 P63IM1 P62IM1 P61IM1 P60IM1 0000,0000							
P7IM1	Port 7 Interrupt Mode Register 1	7EF37H P77IM1 P76IM1 P75IM1 P74IM1 P73IM1 P72IM1 P71IM1 P70IM1 0000,0000							
PINIPR I/O port interrupt priority low register	7EF40H P7IP	P6IP P5IP P4IP P3IP P2IP P1IP P0IP 0000,0000							
PINIPH I/O port interrupt priority high register	7EF41H P7IPH	P6IPH P5IPH P4IPH P3IPH P2IPH P1IPH P0IPH 0000,0000							
P0WKUE P0 port interrupt wake-up enable register	7EF42H P07WKUE P06WKUE P05WKUE P04WKUE P03WKUE P02WKUE P01WKUE P00WKUE 0000,0000								
P1WKUE P1 port interrupt wake-up enable register	7EF43H P17WKUE P16WKUE P15WKUE P14WKUE P13WKUE P12WKUE P11WKUE P10WKUE 0000,0000								
P2WKUE P2 port interrupt wake-up enable register	7EF44H P27WKUE P26WKUE P25WKUE P24WKUE P23WKUE P22WKUE P21WKUE P20WKUE 0000,0000								
P3WKUE P3 port interrupt wake-up enable register	7EF45H P37WKUE P36WKUE P35WKUE P34WKUE P33WKUE P32WKUE P31WKUE P30WKUE 0000,0000								
P4WKUE P4 port interrupt wake-up enable register	7EF46H P47WKUE P46WKUE P45WKUE P44WKUE P43WKUE P42WKUE P41WKUE P40WKUE 0000,0000								
P5WKUE P5 port interrupt wake-up enable register	7EF47H - - P55WKUE P54WKUE P53WKUE P52WKUE P51WKUE P50WKUE xx00,0000								
P6WKUE P6 port interrupt wake-up enable register	7EF60H P67WKUE P66WKUE P65WKUE P64WKUE P63WKUE P62WKUE P61WKUE P60WKUE 0000,0000								
P7WKUE P7 port interrupt wake-up enable register	7EF61H P77WKUE P76WKUE P75WKUE P74WKUE P73WKUE P72WKUE P71WKUE P70WKUE 0000,0000								

### 13.1.1 PORT INTERRUPT ENABLE REGISTER (PxINTE)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
P0INTE	7EF00H P07INTE P06INTE P05INTE P04INTE P03INTE P02INTE P01INTE P00INTE								
P1INTE	7EF01H P17INTE P16INTE P15INTE P14INTE P13INTE P12INTE P11INTE P10INTE								
P2INTE	7EF02H P27INTE P26INTE P25INTE P24INTE P23INTE P22INTE P21INTE P20INTE								
P3INTE	7EF03H P37INTE P36INTE P35INTE P34INTE P33INTE P32INTE P31INTE P30INTE								
P4INTE	7EF04H P47INTE P46INTE P45INTE P44INTE P43INTE P42INTE P41INTE P40INTE								
P5INTE	7EF05H - - P55INTE P54INTE P53INTE P52INTE P51INTE P50INTE								
P6INTE	7EF06H P67INTE P66INTE P65INTE P64INTE P63INTE P62INTE P61INTE P60INTE								
P7INTE	7EF07H 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)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
P0INTF	7EF10H P07INTF P06INTF P05INTF P04INTF P03INTF P02INTF P01INTF P00INTF								
P1INTF	7EF11H P17INTF P16INTF P15INTF P14INTF P13INTF P12INTF P11INTF P10INTF								
P2INTF	7EF12H P27INTF P26INTF P25INTF P24INTF P23INTF P22INTF P21INTF P20INTF								
P3INTF	7EF13H P37INTF P36INTF P35INTF P34INTF P33INTF P32INTF P31INTF P30INTF								
P4INTF	7EF14H P47INTF P46INTF P45INTF P44INTF P43INTF P42INTF P41INTF P40INTF								
P5INTF	7EF15H - - P55INTF P54INTF P53INTF P52INTF P51INTF P50INTF								
P6INTF	7EF16H P67INTF P66INTF P65INTF P64INTF P63INTF P62INTF P61INTF P60INTF								

P7INTF	7EFDFD17H P7INTF P76INTF P75INTF P74INTF P73INTF P72INTF P71INTF P70INTF			
--------	--	--	--	--

PnINTF.x: port interrupt request flag (n=0~7, x=0~7)

0: Pn.x port has no interrupt request

1: The Pn.x port has an interrupt request. If the interrupt is enabled, the interrupt service routine will be entered. The flag bit needs to be cleared by software.

### 13.1.3 PORT INTERRUPT MODE CONFIGURATION REGISTER (PxIM0, PxIM1)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
P0IM0	7EFDFD20H P07IM0 P06IM0 P05IM0 P04IM0 P03IM0 P02IM0 P01IM0 P00IM0							
P0IM1	7EFDFD30H P07IM1 P06IM1 P05IM1 P04IM1 P03IM1 P02IM1 P01IM1 P00IM1							
P1IM0	7EFDFD21H P17IM0 P16IM0 P15IM0 P14IM0 P13IM0 P12IM0 P11IM0 P10IM0							
P1IM1	7EFDFD31H P17IM1 P16IM1 P15IM1 P14IM1 P13IM1 P12IM1 P11IM1 P10IM1							
P2IM0	7EFDFD22H P27IM0 P26IM0 P25IM0 P24IM0 P23IM0 P22IM0 P21IM0 P20IM0							
P2IM1	7EFDFD32H P27IM1 P26IM1 P25IM1 P24IM1 P23IM1 P22IM1 P21IM1 P20IM1							
P3IM0	7EFDFD23H P37IM0 P36IM0 P35IM0 P34IM0 P33IM0 P32IM0 P31IM0 P30IM0							
P3IM1	7EFDFD33H P37IM1 P36IM1 P35IM1 P34IM1 P33IM1 P32IM1 P31IM1 P30IM1							
P4IM0	7EFDFD24H P47IM0 P46IM0 P45IM0 P44IM0 P43IM0 P42IM0 P41IM0 P40IM0							
P4IM1	7EFDFD34H P47IM1 P46IM1 P45IM1 P44IM1 P43IM1 P42IM1 P41IM1 P40IM1							
P5IM0	7EFDFD25H	-	-	P55IM0 P54IM0 P53IM0 P52IM0 P51IM0 P50IM0				
P5IM1	7EFDFD35H	-	-	P55IM1 P54IM1 P53IM1 P52IM1 P51IM1 P50IM1				
P6IM0	7EFDFD26H P67IM0 P66IM0 P65IM0 P64IM0 P63IM0 P62IM0 P61IM0 P60IM0							
P6IM1	7EFDFD36H P67IM1 P66IM1 P65IM1 P64IM1 P63IM1 P62IM1 P61IM1 P60IM1							
P7IM0	7EFDFD27H P77IM0 P76IM0 P75IM0 P74IM0 P73IM0 P72IM0 P71IM0 P70IM0							
P7IM1	7EFDFD37H 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
1	1	high level interrupt

### 13.1.4 Port Interrupt Priority Control Registers (PINIPL, PINIPH)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0	
PINIPL	7EFDFD60H P7IP	P6IP	P5IP	P4IP	P3IP	P2IP	P1IP	P0IP
PINIPH	7EFDFD61H P7IPH	P6IPH	P5IPH	P4IPH	P3IPH	P2IPH	P1IPH	P0IPH

PxIP, PxIPH: Px port interrupt priority control bits

00: Px port interrupt priority is level 0 (the lowest level)

01: Px port interrupt priority is level 1 (lower level)

10: Px port interrupt priority is 2 (higher)

11: Px port interrupt priority is level 3 (highest level)

### 13.1.5 Port Interrupt Power-down Wake-Up Enable Register (PxWKUE)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
P0WKUE	7EF0D40H P07	WKUE P06WKUE P05WKUE P04	WKUE P03WKUE P02WKUE P01	WKUE P00WKUE					
P1WKUE	7EF0D41H P17	WKUE P16WKUE P15WKUE P14	WKUE P13WKUE P12WKUE P11	WKUE P10WKUE					
P2WKUE	7EF0D42H P27	WKUE P26WKUE P25WKUE P24	WKUE P23WKUE P22WKUE P21	WKUE P20WKUE					
P3WKUE	7EF0D43H P37	WKUE P36WKUE P35WKUE P34	WKUE P33WKUE P32WKUE P31	WKUE P30WKUE					
P4WKUE	7EF0D44H P47	WKUE P46WKUE P45WKUE P44	WKUE P43WKUE P42WKUE P41	WKUE P40WKUE					
P5WKUE	7EF0D45H	-	-	P55WKUE P54WKUE P53WKUE P52WKUE P51	WKUE P50WKUE				
P6WKUE	7EF0D46H P67	WKUE P66WKUE P65WKUE P64	WKUE P63WKUE P62WKUE P61	WKUE P60WKUE					
P7WKUE	7EF0D47H P77	WKUE P76WKUE P75WKUE P74	WKUE P73WKUE P72WKUE P71	WKUE 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 Pn.x port interrupt power-down wake-up function

## 13.2 Example program

### 13.2.1 P0 port falling edge interrupt

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXF = 1;                                     //Enable      XFR
    CKCON = 0x00;                                  //access to set external data bus speed to fastest
    WTST = 0x00;                                   //Set the program code to wait for parameters,
                                                    //assign to      CPU The speed of executing the program is set to the fastest

    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 port interrupt

    EA = 1;

    while (1);
}

// Can't compile directly in because the interrupt vector is greater than
// must borrow interrupt entry address No.
void common_isr() interrupt 13
{
    unsigned char intf;

    intf = P0INTF;
    if (intf)
    {
        P0INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P0.0 port interrupt
        }
        if (intf & 0x02)
        {
    }
}

```

```

        }
        if (intf & 0x04)
        {
            //P0.1 port interrupt
        }
        if (intf & 0x08)
        {
            //P0.2 port interrupt
        }
        if (intf & 0x10)
        {
            //P0.3 port interrupt
        }
        if (intf & 0x20)
        {
            //P0.4 port interrupt
        }
        if (intf & 0x40)
        {
            //P0.5 port interrupt
        }
        if (intf & 0x80)
        {
            //P0.6 port interrupt
        }
    }
}

// ISR.ASM
//Save the code below as ISP.ASM , and then add the file to the project

```

<b>CSEG</b> <b>JMP</b> <b>POINT_ISR:</b>	<b>AT 012BH</b> <b>POINT_ISR</b>	; <b>P0</b> port interrupt entry address
	<b>JMP</b> <b>END</b>	; <b>borrow 13</b> The entry address of the interrupt

### 13.2.2 P1 port rising edge interrupt

//The test frequency is 11.0592MHz

```

##include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1;                                     //Enable      XFR
    CKCON = 0x00;                                    //access to set external data bus speed to fastest
    WTST = 0x00;                                     //Set the program code to wait for parameters,
                                                    //assign to      CPU The speed of executing the program is set to the fastest
    P0MO = 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 port interrupt

EA = 1;

while (1);

}

// Can't compile directly because the interrupt vector is greater than
// must borrow interrupt entry address No.
// 31      KEIL
// 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 code below as **ISP.ASM** , and then add the file to the project

<b>CSEG</b>	<b>AT 0133H</b>	<b>;P1</b> port interrupt entry address
<b>JMP</b>	<b>P1INT_ISR</b>	
<b>P1INT_ISR:</b>		
<b>JMP</b>	<b>006BH</b>	,borrow <b>13</b> The entry address of the interrupt
<b>END</b>		

---

### 13.2.3 P2 port low level interrupt

---

//The test frequency is **11.0592MHz**

```

##include "stc8h.h"
##include "stc32g.h"                                //For header files, see Download Software
##include "intrins.h"

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    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;

    EA = 1;

    while (1);
}

// Since the interrupt vector is greater than 31, exist KEIL cannot be compiled directly in

```

```

//must borrow the 13 No. interrupt entry address
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 code below as ISP.ASM, and then add the file to the project

```

<b>CSEG</b> <b>JMP</b>	<b>AT 013BH</b> <b>P2INT_ISR</b>	; <b>P2</b> port interrupt entry address
<b>P2INT_ISR:</b>		; <b>borrow 13</b> The entry address of the interrupt
<b>JMP</b> <b>END</b>	<b>006BH</b>	

### 13.2.4 P3 port high level interrupt

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXF = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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

    EA = 1;

    while (1);
}

// Can't compile directly in because the interrupt vector is greater than
// must borrow interrupt entry address No.
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)
        {
    
```

```

        }
        if (intf & 0x08)
        {
            //P3.2 port interrupt
        }
        if (intf & 0x10)
        {
            //P3.3 port interrupt
        }
        if (intf & 0x20)
        {
            //P3.4 port interrupt
        }
        if (intf & 0x40)
        {
            //P3.5 port interrupt
        }
        if (intf & 0x80)
        {
            //P3.6 port interrupt
        }
    }
}

// ISR.ASM
//Save the code below as ISP.ASM, and then add the file to the project

```

<b>CSEG</b> <b>JMP</b> <b>P3INT_ISR:</b> <b>JMP</b> <b>END</b>	<b>AT 0143H</b> <b>P3INT_ISR</b> <b>006BH</b>	; <b>P3</b> port interrupt entry address ;borrow <b>13</b> The entry address of the interrupt
--	---	--

---

# 14 timer/counter (24-bit timer, 8-bit prescaler + 16-bit auto-reload)

STC32G series microcontrollers have 5 [24-bit timers/counters \(8-bit prescaler + 16-bit count\)](#). Five 16-bit timers

The devices T0, T1, T2, T3 and T4 all have two working modes: counting mode and timing mode. For timer/counter T0 and T1, use it

They use the corresponding control bit C/T in the special function register TMOD to select whether T0 or T1 is a timer or a counter. on timer/meter

T2, use the control bit T2\_C/T in the special function register AUXR to select whether T2 is a timer or a counter. to timer/count

T3, use the control bit T3\_C/T in the special function register T4T3M to select whether T3 is a timer or a counter. for timer/counter

T4, use the control bit T4\_C/T in the special function register T4T3M to select whether T4 is a timer or a counter. Timer/Counter Core

The heart part is an up-counter, which essentially counts the pulses. Only the source of the count pulses is different: if the count pulses come from the system

Clock, it is timing mode, at this time, the timer/counter gets a count pulse every 12 clocks or every 1 clock, and the count value is incremented by 1;

If the counting pulse comes from the external pin of the microcontroller, it is the counting mode, and 1 is added for each pulse.

When timer/counter T0, T1 and T2 work in timing mode, T0x12, T1x12 and T1x12 in special function register AUXR

T2x12 respectively decides whether the system clock/12 or the system clock/1 (without frequency division) and then allows T0, T1 and T2 to count. when timer/count

When T3 and T4 work in timing mode, T3x12 and T4x12 in special function register T4T3M respectively determine the system clock/12

Or let T3 and T4 count after the system clock/1 (not divided). When the timer/counter works in the counting mode, the external pulse count is

Numbers are not divided.

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), Mode 3 (16-bit auto-reload mode with non-maskable interrupts). Timer/Counter 1 Divide Modulo

Except Equation 3, other working modes are the same as Timer/Counter 0. T1 is invalid in mode 3 and stops counting. The working mode of timer T2

Fixed to 16-bit auto-reload mode. T2 can be used as a timer, or as a serial port baud rate generator and programmable clock output.

Timer 3 and Timer 4 are the same as Timer T2, and their working mode is fixed to 16-bit auto-reload mode. T3/T4 can be set

It can also be used as a serial port baud rate generator and programmable clock output.

## 14.1 Timer related registers

symbol	describe	address	Bit addresses and symbols								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 lower 8-bit register 8AH										0000,0000
TL1	Timer 1 lower 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	T_M0x6 T2R	T2_C/T T2x12	EXTRAM S1	BRT 0000,0001				
INTCLKO	Interrupt and Clock Output Control Register 8FH		-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	Power-down wake-up timer low byte AAH										1111,1111
WKTCH	Power-down wake-up timer high byte ABH WKTEN										0111,1111
T4T3M	Timer 4/3 Control Register 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

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
TM0PS	Timer 0 Clock Prescaler Register	7EFFEA0H									0000,0000
TM1PS	Timer 1 Clock Prescaler Register	7EFFEA1H									0000,0000
TM2PS	Timer 2 Clock Prescaler Register	7EFFEA2H									0000,0000
TM3PS	Timer 3 Clock Prescaler Register	7EFFEA3H									0000,0000
TM4PS	Timer 4 Clock Prescaler Register	7EFFEA4H									0000,0000

## 14.2 Timer 0/1

### 14.2.1 Timer 0/1 Control Register (TCON)

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

TF1: T1 overflow interrupt flag. After T1 is allowed to count, it starts counting by 1 from the initial value. When an overflow occurs, the TF1 bit is set to "1" by hardware,

And request an interrupt to the CPU, and keep it until the CPU responds to the interrupt, then it is cleared to "0" by the hardware (it can also be cleared to "0" by the query software).

TR1: The running control bit of timer T1. This bit is set and cleared by software. When GATE (TMOD.7)=0, TR1=1, T1 is allowed to open

Start counting, when TR1=0, T1 counting is prohibited. T1 is only allowed when GATE (TMOD.7)=1, TR1=1 and INT1 input high level count.

TF0: T0 overflow interrupt flag. After T0 is allowed to count, it starts counting by 1 from the initial value. When overflow occurs, TF0 is set to "1" by hardware.

Request an interrupt from the CPU and keep the CPU responding to the interrupt until it is cleared by hardware (it can also be cleared by query software).

TR0: The running control bit of timer T0. This bit is set and cleared by software. When GATE (TMOD.3)=0, TR0=1, T0 is allowed to open

Start counting, when TR0=0, T0 counting is prohibited. When GATE (TMOD.3)=1, TR0=1 and INT0 input high level, T0 is allowed Counting, T0 counting is prohibited when TR0=0.

IE1: External interrupt 1 request source (INT1/P3.3) flag. IE1=1, the external interrupt requests an interrupt to the CPU, when the CPU responds to the interrupt

Hardware clears "0" IE1.

IT1: External interrupt source 1 trigger control bit. IT1=0, external interrupt 1 can be triggered by rising edge or falling edge. IT1=1, the external interrupt 1 is programmed as

Falling edge trigger mode.

IE0: External interrupt 0 request source (INT0/P3.2) flag. IE0=1 External interrupt 0 requests an interrupt from the CPU. When the CPU responds to an external interrupt,

Cleared by hardware to "0" IE0 (edge-triggered).

IT0: External interrupt source 0 trigger control bit. IT0=0, external interrupt 0 can be triggered by rising edge or falling edge. IT0=1, external interrupt 0 is programmed as

Falling edge trigger mode.

### 14.2.2 Timer 0/1 Mode Register (TMOD)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1\_GATE: Control timer 1. When set to 1, timer/counter 1 can be turned on only when the INT1 pin is high and the TR1 control bit is 1.

T0\_GATE: Control timer 0. When set to 1, timer/counter 0 can be turned on only when the INT0 pin is high and the TR0 control bit is 1.

T1\_C/T: Control timer 1 to be used as a timer or counter, clear to 0 to use as a timer (counting the internal system clock), set 1 to be used as a timer Counter (counts external pulses on pin T1/P3.5).

T0\_C/T: Control timer 0 to be used as a timer or counter, clear to 0 to use as a timer (counting the internal system clock), set 1 to be used as a timer Counter (counts external pulses on pin T0/P3.4).

T1\_M1/T1\_M0: Timer timer/counter 1 mode selection

		Timer/Counter 1 working mode
T1_M1	T1_M0	16-bit auto-reload mode When the 16-bit count value in [TH1,TL1] overflows, the system will automatically convert the internal 16-bit

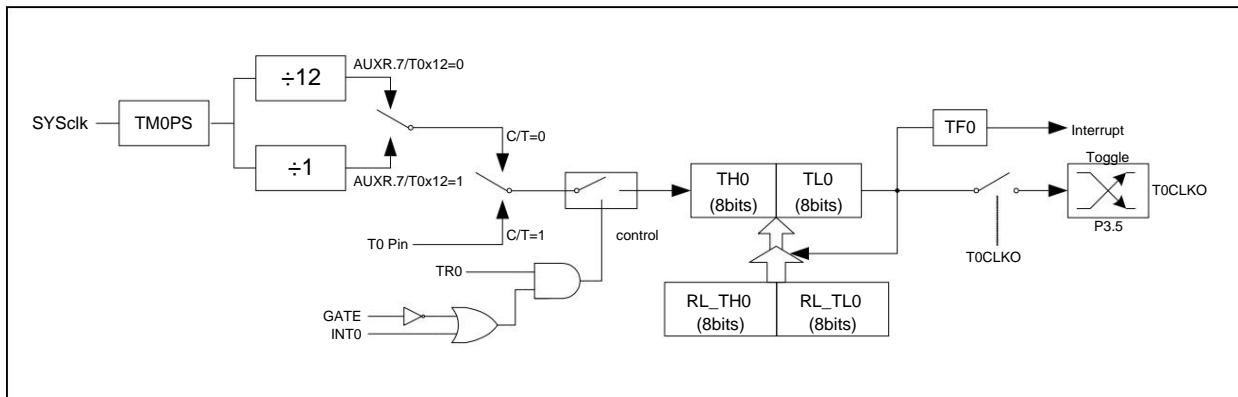
		The reload value in the reload register is loaded into [TH1,TL1].
0	1	16-bit no auto-reload mode When the 16-bit count value in [TH1,TL1] overflows, timer 1 will start counting from 0
1	0	8-bit auto-reload mode When the 8-bit count value in TL1 overflows, the system will automatically reload the value in TH1 Loaded into TL1.
1	1 T1 stopped working	

T0\_M1/T0\_M0: Timer timer/counter 0 mode selection

T0_M1 T0_M0		Timer/Counter 0 working mode
0	0	16-bit auto-reload mode When the 16-bit count value in [TH0,TL0] overflows, the system will automatically convert the internal 16-bit The reload value in the reload register is loaded into [TH0,TL0].
0	1	16-bit no auto-reload mode When the 16-bit count value in [TH0,TL0] overflows, timer 0 will start counting from 0 in 8-
1	0	bit auto-reload mode When the 8-bit count value in TL0 overflows, the system will automatically reload the value in TH0 Loaded into TL0.
1	1	16-bit auto-reload mode for non-maskable interrupts Same as mode 0, non-maskable interrupt, interrupt priority is the highest, higher than other Interrupt has priority and cannot be closed, and can be used as the system tick of the operating system timer, or system monitoring timer.

### 14.2.3 Timer 0 Mode 0 (16-bit auto-reload mode)

In this mode, Timer/Counter 0 acts as an auto-reloadable 16-bit counter, as shown in the following figure:



Timer/Counter 0 Mode 0: 16-Bit Auto-Rewload Mode

When GATE=0 (TMOD.3), such as TR0=1, the timer counts. When GATE=1, timer 0 can be controlled by external input INT0, which can realize pulse width measurement. TR0 is a control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0 , the multiplexer is connected to the frequency division output of the system clock, T0 counts the internal system clock, and T0 works in the timing mode. when When C/T=1 , the multiplexer is connected to the external pulse input P3.4/T0, that is, T0 works in counting mode.

The timer 0 of the STC microcontroller has two counting rates: one is the 12T mode, which increases by 1 every 12 clocks, which is the same as the traditional 8051 microcontroller; the other is the 1T mode, which adds 1 to each clock, and the speed is the speed of the traditional 8051 microcontroller. 12 times. 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

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, that is, timer/counter 0 is disabled, the content written to TL0 will be written to RL\_TL0 at the same time, and the content written to TH0 will also be written to RL\_TH0 at the same time. When TR0=1, that is, timer/counter 0 is allowed to work, the content written to TL0 is not actually written to the current register TL0, but is written to the hidden register RL\_TL0, and the content written to TH0 is actually written to the hidden register RL\_TH0. Instead of writing to the current register TH0, it writes to the hidden register RL\_TH0, which neatly implements a 16-bit reload timer. When reading the content of TH0 and TL0, the content read is the content of TH0 and TL0, not the content of RL\_TH0 and RL\_TL0.

When timer 0 works in mode 0 (TMOD[1:0]/[M1,M0]=00B), the overflow of [TH0,TL0] not only sets TF0, but also automatically Reload the contents of [RL\_TH0,RL\_TL0] into [TH0,TL0].

When T0CLKO/INT\_CLKO.0=1, P3.5/T1 pin is configured as timer 0 clock output T0CLKO. The output clock frequency is T0 overflow rate/2.

If C/T=0, timer/counter T0 counts the internal system clock, then:

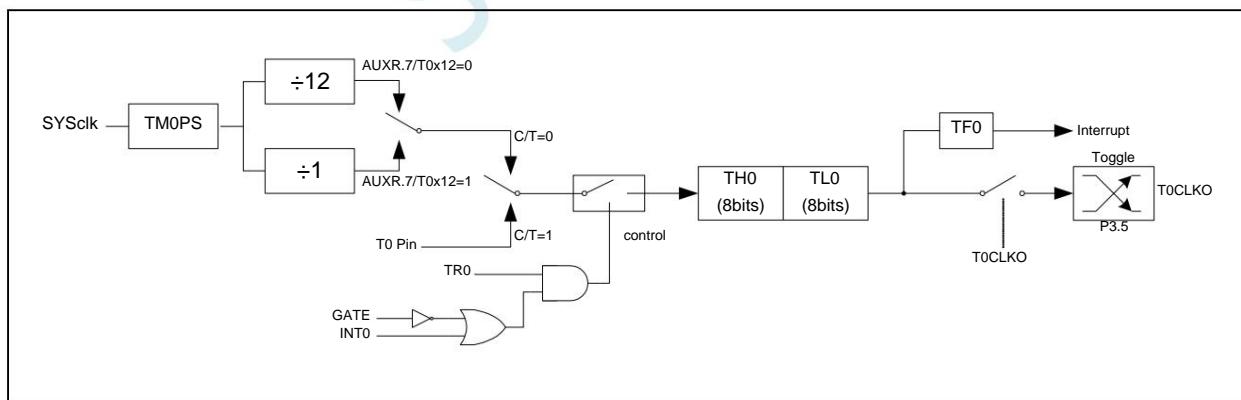
Output clock frequency when T0 works in 1T mode (AUXR.7/T0x12=1) =  $(\text{SYSclk}) / (\text{TM0PS} + 1) / (65536 - [\text{RL\_TH0}, \text{RL\_TL0}]) / 2$   
T0 works in 12T mode (AUXR.7 /T0x12=0) output clock frequency =  $(\text{SYSclk}) / (\text{TM0PS} + 1) / 12 / (65536 - [\text{RL\_TH0}, \text{RL\_TL0}]) / 2$

If C/T=1, timer/counter T0 counts external pulse input (P3.4/T0), then: output clock

frequency =  $(\text{T0\_Pin\_CLK}) / (65536 - [\text{RL\_TH0}, \text{RL\_TL0}]) / 2$

#### 14.2.4 Timer 0 Mode 1 (16-bit non-reloadable mode)

In this mode, timer/counter 0 works in 16-bit non-reloadable mode, as shown in the following figure:



Mode 1 of Timer/Counter 0: 16-bit non-reloadable mode

In this mode, timer/counter 0 is configured as 16-bit non-reloadable mode, which consists of 8 bits of TL0 and 8 bits of TH0. TL0's The 8-bit overflow will carry over to TH0, and the overflow flag bit TF0 in TCON will be set when TH0 counts overflow.

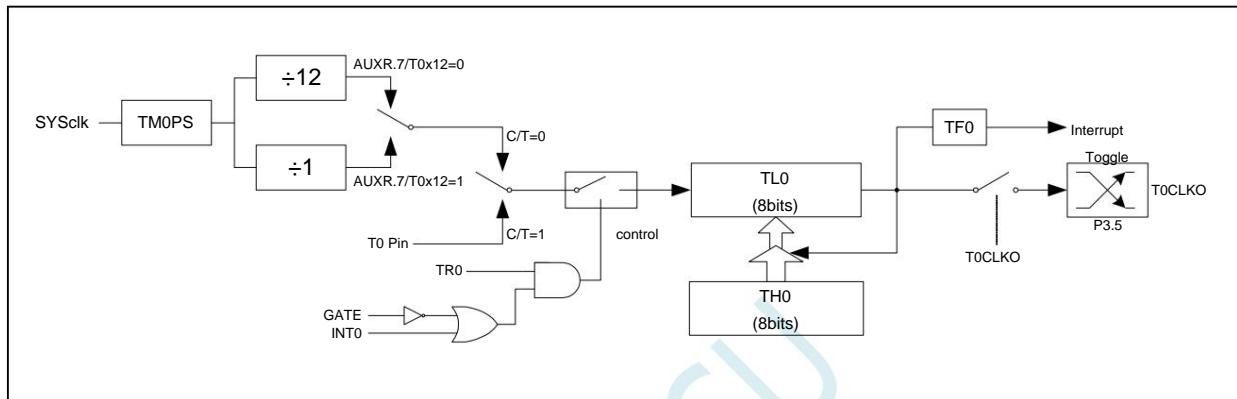
When GATE=0 (TMOD.3), such as TR0=1, the timer counts. When GATE=1, timer 0 can be controlled by external input INT0, which can realize pulse width measurement. TR0 is a control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0 , the multiplexer is connected to the frequency division output of the system clock, T0 counts the internal system clock, and T0 works in the timing mode. when When C/T=1 , the multiplexer is connected to the external pulse input P3.4/T0, that is, T0 works in counting mode.

The timer 0 of the STC microcontroller has two counting rates: one is the 12T mode, which increases by 1 every 12 clocks, which is the same as the traditional 8051 microcontroller; the other is the 1T mode, which adds 1 to each clock, and the speed is the speed of the traditional 8051 microcontroller. 12 times. 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.

## 14.2.5 Timer 0 Mode 2 (8 -Bit Auto-Reload Mode)

In this mode, Timer/Counter 0 acts as an auto-reloadable 8-bit counter, as shown in the following figure:



Timer/Counter 0 Mode 2: 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 software. When reloading, the content of TH0 is not changed. Change.

When T0CLKO/INT\_CLKO.0=1, P3.5/T1 pin is configured as timer 0 clock output T0CLKO. The output clock frequency is T0 overflow rate/2.

If C/T=0, timer/counter T0 counts the internal system clock, then:

$$\text{Output clock frequency when T0 works in 1T mode (AUXR.7/T0x12=1)} = (\text{SYSclk}) / (\text{TM0PS} + 1) / (256 - \text{TH0}) / 2$$

$$\text{T0 works in 12T mode (AUXR.7/T0x12=0)} \text{ when the output clock frequency} = (\text{SYSclk}) / (\text{TM0PS} + 1) / 12 / (256 - \text{TH0}) / 2$$

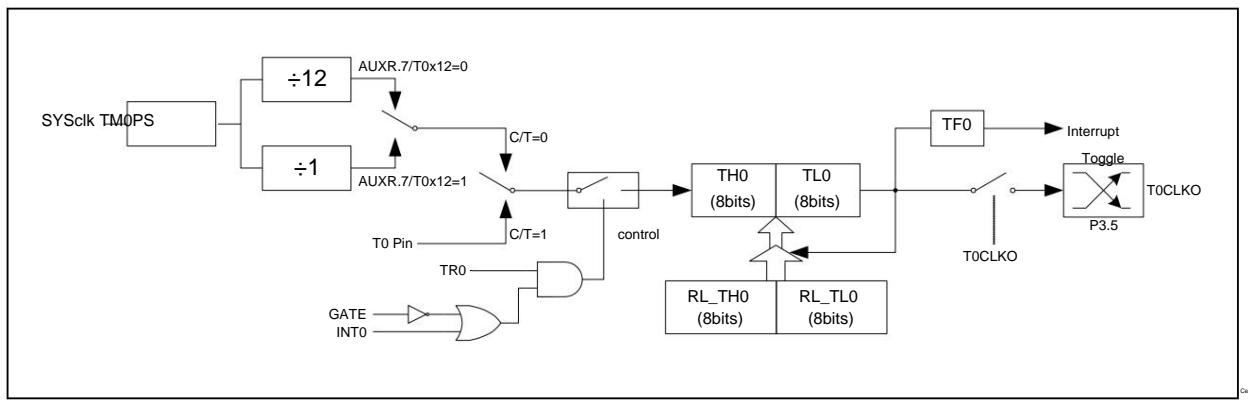
If C/T=1, timer/counter T0 counts external pulse input (P3.4/T0), then: output clock

$$\text{frequency} = (\text{T0_Pin_CLK}) / (256 - \text{TH0}) / 2$$

## 14.2.6 Timer 0 mode 3 (non-maskable interrupt 16 -bit auto-reload, real-time operation

as a system metronome)

For timer/counter 0, its working mode mode 3 is the same as working mode 0 (the schematic diagram of timer mode 3 in the figure below is the same as working mode 0). The only difference is: when timer/counter 0 works in mode 3, only ET0/IE.1 (timer/counter 0 interrupt enable bit) is required, and EA/IE.7 (total interrupt enable bit) is not required. ) can turn on the timer/counter 0 interrupt, the timer/counter 0 interrupt in this mode has nothing to do with the total interrupt enable bit EA, once the timer/counter 0 interrupt working in mode 3 is turned on (ET0=1 ), then the interrupt is not maskable, the priority of the interrupt is the highest, that is, the interrupt cannot be interrupted by any interrupt, and the interrupt is neither controlled by EA/IE.7 nor controlled by ET0 after it is turned on . , this interrupt cannot be masked when EA=0 or ET0=0. Therefore, this mode is called the 16-bit auto-reload mode of non-maskable interrupts .

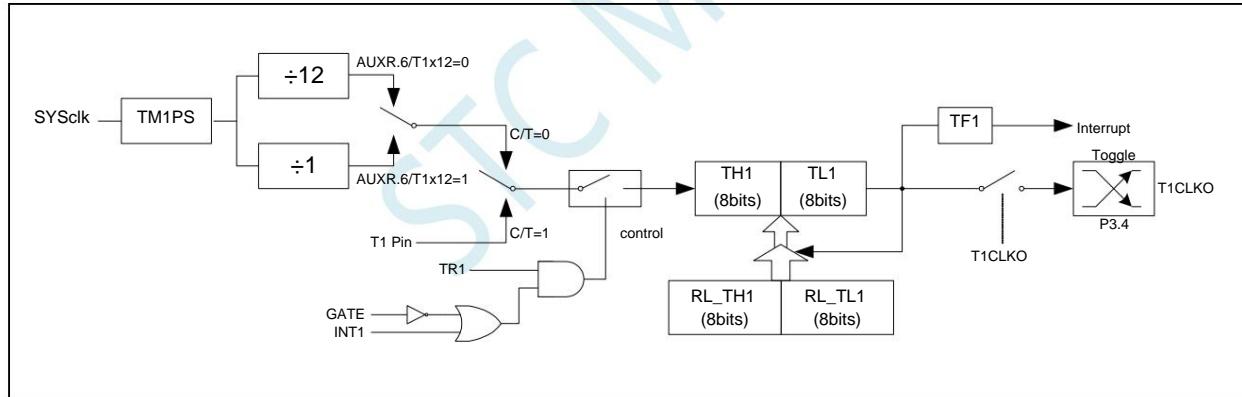


Timer/Counter 0 Mode 3: 16-Bit Auto-Reload Mode with Non-Maskable Interrupts

Note: When timer/counter 0 is working in mode 3 (16-bit auto-reload mode with non-maskable interrupts), it is not necessary to enable EA/IE.7 (total interrupt enable bit), just enable ET0/IE.1 (Timer/Counter 0 Interrupt Enable Bit) can turn on the timer/counter 0 interrupt. The timer/counter 0 interrupt in this mode has nothing to do with the global interrupt enable bit EA. Once the timer/counter 0 interrupt in this mode is turned on, the timer/counter 0 interrupt priority is the highest, and it cannot be interrupted by any other interrupt (no matter it is lower than the timer/counter 0 interrupt priority). The interrupt is still an interrupt with a higher priority than its priority, and neither can interrupt the timer/counter 0 interrupt at this time), and the interrupt is neither controlled by EA/IE.7 nor controlled by ET0 after it is turned on. Clear Neither the EA nor ET0 can turn off this interrupt.

## 14.2.7 Timer 1 Mode 0 (16 -Bit Auto-Reload Mode)

In this mode, Timer/Counter 1 acts as an auto-reloadable 16-bit counter, as shown in the following figure:



Timer/Counter 1 Mode 0: 16-Bit Auto-Reload Mode

When GATE=0 (TMOD.7), such as TR1=1, the timer counts. When GATE=1 , timer 1 can be controlled by external input INT1, which can realize pulse width measurement. TR1 is a control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0 , the multiplexer is connected to the frequency division output of the system clock, T1 counts the internal system clock, and T1 works in the timing mode. when When C/T=1 , the multiplexer is connected to the external pulse input P3.5/T1, that is, T1 works in counting mode.

Timer 1 of STC microcontroller has two counting rates: one is 12T mode, which increases by 1 every 12 clocks, which is the same as that of traditional 8051 microcontroller; the other is 1T mode, which increases by 1 for each clock, and the speed is the speed of traditional 8051 microcontroller. 12 times. The rate of T1 is determined by T1x12 in the special function register AUXR. 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 and TH1 share the same address, and RL\_TL1 and TL1 share the same address. When TR1=0, that is, timer/counter 1 is disabled, the content written to TL1 will be written to RL\_TL1 at the same time.

What is written in TH1 is also written in RL\_TH1 at the same time. When TR1=1, that is, timer/counter 1 is allowed to work, the content written to TL1 is not actually written to the current register TL1, but is written to the hidden register RL\_TL1, and the content written to TH1 is actually written into the hidden register RL\_TL1. Instead of writing to the current register TH1, it writes to the hidden register RL\_TH1, which neatly implements a 16-bit reload timer. When reading the content of TH1 and TL1, the content read is the content of TH1 and TL1, not the content of RL\_TH1 and RL\_TL1.

When timer 1 works in mode 1 (TMOD[5:4]/[M1,M0]=00B), the overflow of [TH1,TL1] not only sets TF1, but also automatically Reload the contents of [RL\_TH1,RL\_TL1] into [TH1,TL1].

When T1CLKO/INT\_CLKO.1=1, P3.4/T0 pin is configured as timer 1 clock output T1CLKO. The output clock frequency is  $T_1$  overflow rate/2.

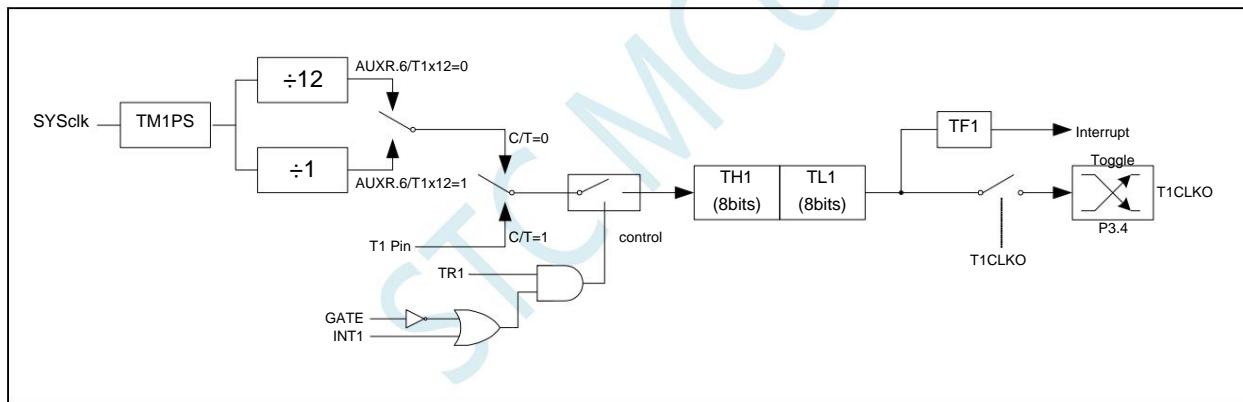
If C/T=0, timer/counter T1 counts the internal system clock, then:

Output clock frequency when T1 works in 1T mode ( $AUXR.6/T1x12=1$ ) =  $(SYSclk)/(TM1PS+1)/(65536-[RL_TH1, RL_TL1])/2$   
T1 works in 12T mode ( $AUXR.6/T1x12=0$ ) output clock frequency =  $(SYSclk)/(TM1PS+1)/12/(65536-[RL_TH1, RL_TL1])/2$

If C/T=1, timer/counter T1 counts external pulse input (P3.5/T1), then: 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 works in 16-bit non-reloadable mode, as shown in the following figure:



Timer/Counter 1 Mode 1: 16-bit non-reloadable mode

In this mode, timer/counter 1 is configured as 16-bit non-reloadable mode, which consists of 8 bits of TL1 and 8 bits of TH1. TL1's The 8-bit overflow will carry over to TH1, and the overflow flag bit TF1 in TCON will be set when TH1 overflows.

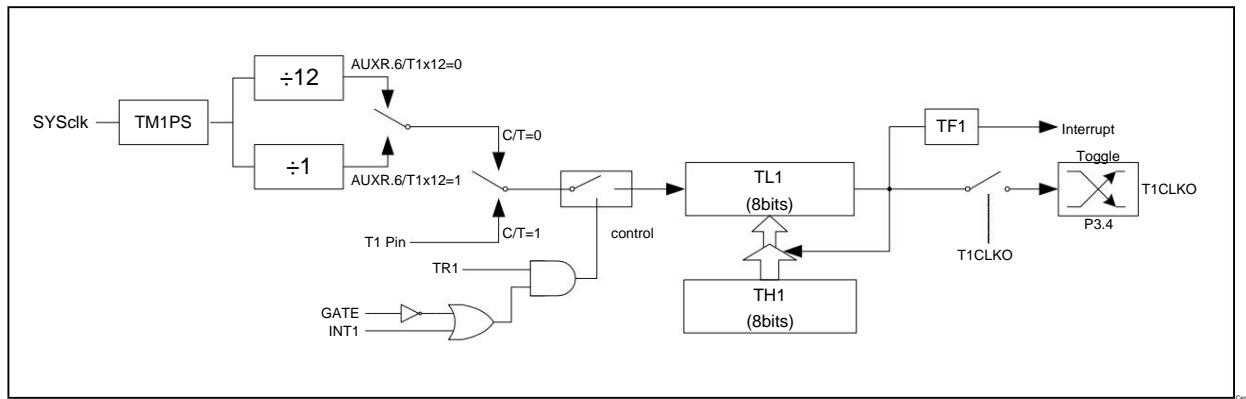
When GATE=0 (TMOD.7), such as TR1=1, the timer counts. When GATE=1, timer 1 can be controlled by external input INT1, which can realize pulse width measurement. TR1 is a control bit in the TCON register. For the specific function description of each bit of the TCON register, see the introduction of the TCON register in the previous section.

When C/T=0, the multiplexer is connected to the frequency division output of the system clock, T1 counts the internal system clock, and T1 works in the timing mode. When C/T=1, the multiplexer is connected to the external pulse input P3.5/T1, that is, T1 works in counting mode.

Timer 1 of STC microcontroller has two counting rates: one is 12T mode, which increases by 1 every 12 clocks, which is the same as that of traditional 8051 microcontroller; the other is 1T mode, which increases by 1 for each clock, and the speed is the speed of traditional 8051 microcontroller. 12 times. The rate of T1 is determined by T1x12 in the special function register AUXR. 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)

In this mode, Timer/Counter 1 acts as an auto-reloadable 8-bit counter, as shown in the following figure:



Timer/Counter 1 Mode 2: 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 software, and the content of TH1 is not changed when reloading.

Change.

When T1CLKO/INT\_CLKO.1=1, P3.4/T0 pin is configured as timer 1 clock output T1CLKO. The output clock frequency is  $T_1$  Overflow rate/2.

If C/T=0, timer/counter T1 counts the internal system clock, then:

$$\text{Output clock frequency when } T_1 \text{ works in } 1T \text{ mode (AUXR.6/T1x12=1)} = (\text{SYSclk}) / (\text{TM1PS}+1) / (256-\text{TH1}) / 2$$

$$\text{Output clock frequency when } T_1 \text{ works in } 12T \text{ mode (AUXR.6/T1x12=0)} = (\text{SYSclk}) / (\text{TM1PS}+1) / 12 / (256-\text{TH1}) / 2$$

If C/T=1, the timer/counter T1 counts the external pulse input (P3.5/T1), then:

$$\text{Output Clock Frequency} = (T_1 \text{ Pin CLK}) / (256-\text{TH1}) / 2$$

## 14.2.10 Timer 0 count register (TL0, TH0)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

When timer/counter 0 works in 16-bit mode (mode 0, mode 1, mode 3), TL0 and TH0 are combined into a 16-bit register,

TL0 is the low byte and TH0 is the high byte. In 8-bit mode (mode 2), TL0 and TH0 are two independent 8-bit registers.

## 14.2.11 Timer 1 count register (TL1, TH1)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

When timer/counter 1 works in 16-bit mode (mode 0, mode 1), TL1 and TH1 are combined into a 16-bit register, and TL1 is low byte, TH1 is the high byte. In 8-bit mode (mode 2), TL1 and TH1 are two independent 8-bit registers.

### 14.2.12 Auxiliary Register 1 (AUXR)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6 T2R	T2_C/T T2x12 EXTRAM S1BRT				

T0x12: Timer 0 speed control bit

0: 12T mode, that is, the CPU clock is divided by 12 (FOSC/12)

1: 1T mode, that is, the CPU clock is not divided by frequency (FOSC/1)

T1x12: Timer 1 Speed Control Bit

0: 12T mode, that is, the CPU clock is divided by 12 (FOSC/12)

1: 1T mode, that is, the CPU clock is not divided by frequency (FOSC/1)

### 14.2.13 Interrupt and Clock Output Control Register (INTCLKO)

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

T0CLKO: Timer 0 clock output control

0: turn off the clock output

1: Enable P3.5 port is timer 0 clock output function

When timer 0 overflows, the level of port 3.5 automatically flips.

T1CLKO: Timer 1 clock output control

0: turn off the clock output

1: Enable P3.4 port is timer 1 clock output function

When timer 1 overflows, the level of port 3.4 automatically flips.

### 14.2.14 Timer 0 8 - bit Prescaler Register (TM0PS)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
TM0PS	7EFEA0H								

Timer 0 clock = system clock SYSclk ÷ ( TM0PS + 1 )

### 14.2.15 Timer 1 8 - bit Prescaler Register (TM1PS)

symbol	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	Period calculation formula	
Mode 0/3 (16-bit auto-reload)	1T	timer period =	$\frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)}$ (auto-reload)
	12T	timer period =	$\frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)} \times 12$ (auto-reload)
Mode 1 (16-bit does not auto-reload)	1T	timer period =	$\frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)}$ (Requires software loading)
	12T	timer period =	$\frac{65536 - [TH0, TL0]}{SYSclk/(TM0PS+1)} \times 12$ (Requires software loading)
Mode 2 (8-bit auto-reload)	1T	timer period =	$\frac{256 - TH0}{SYSclk/(TM0PS+1)}$ (auto-reload)
	12T	timer period =	$\frac{256 - TH0}{SYSclk/(TM0PS+1)} \times 12$ (auto-reload)

### 14.2.17 Timer 1 calculation formula

Timer Mode	Timer Speed	Period calculation formula	
Mode 0 (16-bit auto-reload)	1T	timer period =	$\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)}$ (auto-reload)
	12T	timer period =	$\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)} \times 12$ (auto-reload)
Mode 1 (16-bit does not auto-reload)	1T	timer period =	$\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)}$ (Requires software loading)
	12T	timer period =	$\frac{65536 - [TH1, TL1]}{SYSclk/(TM1PS+1)} \times 12$ (Requires software loading)
Mode 2 (8-bit auto-reload)	1T	timer period =	$\frac{256 - TH1}{SYSclk/(TM1PS+1)}$ (auto-reload)
	12T	timer period =	$\frac{256 - TH1}{SYSclk/(TM1PS+1)} \times 12$ (auto-reload)

## 14.3 Timer 2

### 14.3.1 Auxiliary Register 1 (AUXR)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6 T2R	T2_C/T	T2x12 EXTRAM	S1BRT		

T2R: Timer 2 running control bit

0: Timer 2 stops counting

1: Timer 2 starts counting

T2\_C/T: Control timer 0 to be used as a timer or counter, clear to 0 to use as a timer (counting the internal system clock), set 1 to be used as a timer

Counter (counts external pulses on pin T2/P1.2).

T2x12: Timer 2 Speed Control Bit

0: 12T mode, that is, the CPU clock is divided by 12 (FOSC/12)

1: 1T mode, that is, the CPU clock is not divided by frequency (FOSC/1)

### 14.3.2 Interrupt and Clock Output Control Register (INTCLKO)

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

T2CLKO: Timer 2 clock output control

0: Turn off the clock output

1: Enable P1.3 port is timer 2 clock output function

When timer 2 overflows, the level of port P1.3 automatically flips.

### 14.3.3 Timer 2 Count Registers (T2L, T2H)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

The working mode of timer/counter 2 is fixed to 16-bit reload mode, T2L and T2H are combined into a 16-bit register, T2L is the low byte,

T2H is the high byte. When the 16-bit count value in [T2H, T2L] overflows, the system will automatically reload the internal 16-bit reload register

The value is loaded into [T2H, T2L].

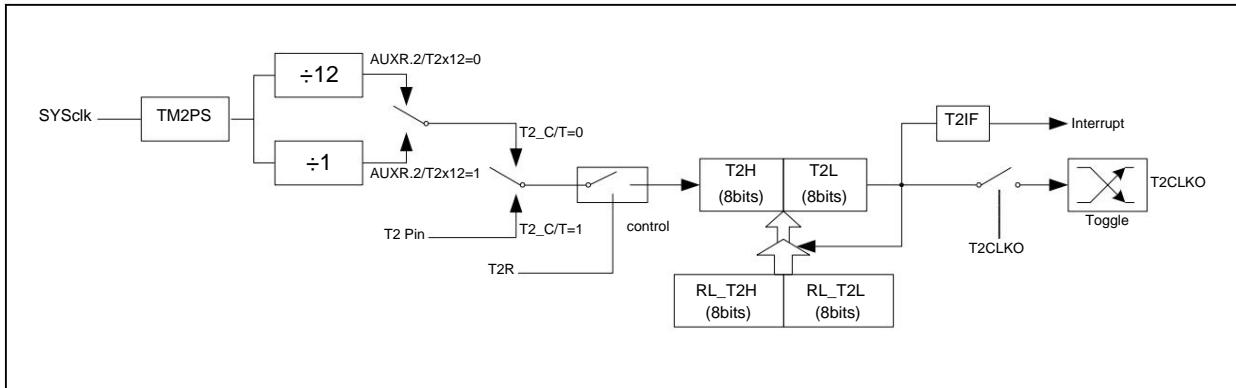
### 14.3.4 Timer 2 8 - bit Prescaler Register (TM2PS)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
TM2PS	FEA2H								

Timer 2 clock = system clock SYSclk  $\div$  ( TM2PS + 1 )

### 14.3.5 Timer 2 working mode

The functional block diagram of Timer/Counter 2 is as follows:



Timer/Counter 2 operating mode: 16-bit auto-reload mode

T2R/AUXR.4 is the control bit in the AUXR register. For the specific function description of each bit of the AUXR register, see the introduction of the AUXR register in the previous section.

Shaw.

When T2\_C/T=0, the multiplexer is connected to the system clock output, T2 counts the internal system clock, and T2 works in timing mode. When T2\_C/T=1, the multiplexer is connected to the external pulse input T2, that is, T2 works in counting mode.

The timer 2 of the STC microcontroller has two counting rates: one is the 12T mode, which increases by 1 every 12 clocks, which is the same as that of the traditional 8051 microcontroller; the other is the 1T mode, which adds 1 to each clock, and the speed is the speed of the traditional 8051 microcontroller. 12 times. The rate of T2 is determined by T2x12 in the special function register AUXR. 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 and T2H share the same address, RL\_T2L and T2L share the same address. When T2R=0, that is, timer/counter 2 is disabled, the content written to T2L will be written to RL\_T2L at the same time, and the content written to T2H will also be written to RL\_T2H at the same time. When T2R=1, that is, timer/counter 2 is allowed to work, the content written to T2L is not actually written to the current register T2L, but is written to the hidden register RL\_T2L, and the content written to T2H is actually written to RL\_T2H instead of writing to the current register T2H, it writes to the hidden register RL\_T2H, which cleverly implements a 16-bit reload timer. When reading the content of T2H and T2L, the content read is the content of T2H and T2L, not the content of RL\_T2H and RL\_T2L.

The overflow of [T2H, T2L] not only sets the interrupt request flag (T2IF) to make the CPU go to execute the interrupt program of timer 2, but also automatically automatically reloads the contents of [RL\_T2H, RL\_T2L] into [T2H, T2L].

### 14.3.6 Timer 2 calculation formula

timer speed	Period calculation formula
1T	timer period = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)}$ (auto-reload)
12T	timer period = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)} \times 12$ (auto-reload)

## 14.4 Timer 3/4

### 14.4.1 Timer 3/4 function pin switching

Symbol address B7			B6	B5	B4	B3	B2	B1	B0
T3T4PIN 7EFFEACH		-	-	-	-	-	-	-	T3T4SEL

T3T4SEL: T3/T3CLKO/T4/T4CLKO pin selection 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)

Symbol 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 run control bit

- 0: Timer 4 stops counting
- 1: Timer 4 starts counting

T4\_C/T: Control timer 4 to be used as a timer or counter, clear to 0 to use as a timer (counting the internal system clock), set 1 to be used as a timer

Counter (counts external pulses on pin T4/P0.6).

T4x12: Timer 4 Speed Control Bit

- 0: 12T mode, that is, the CPU clock is divided by 12 (FOSC/12)
- 1: 1T mode, that is, the CPU clock is not divided by frequency (FOSC/1)

T4CLKO: Timer 4 clock output control

- 0: Turn off the clock output
  - 1: Enable P0.7 port is timer 4 clock output function
- When timer 4 overflows, the level of port P0.7 automatically flips.

TR3: Timer 3 run control bit

- 0: Timer 3 stops counting
- 1: Timer 3 starts counting

T3\_C/T: Control timer 3 to be used as a timer or counter, clear to 0 to use as a timer (counting the internal system clock), set 1 to be used as a timer

Counter (counts external pulses on pin T3/P0.4).

T3x12: Timer 3 Speed Control Bit

- 0: 12T mode, that is, the CPU clock is divided by 12 (FOSC/12)
- 1: 1T mode, that is, the CPU clock is not divided by frequency (FOSC/1)

T3CLKO: Timer 3 clock output control

- 0: Turn off the clock output
  - 1: Enable P0.5 port is timer 3 clock output function
- When timer 3 overflows, the level of port P0.5 will automatically flip.

### 14.4.3 Timer 3 Count Registers (T3L, T3H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

The working mode of timer/counter 3 is fixed to 16-bit reload mode, T3L and T3H are combined into a 16-bit register, T3L is the low byte, T3H is the high byte. When the 16-bit count value in [T3H, T3L] overflows, the system will automatically reload the internal 16-bit reload register

The value is loaded into [T3H, T3L].

### 14.4.4 Timer 4 Count Registers (T4L, T4H)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

The working mode of timer/counter 4 is fixed to 16-bit reload mode, T4L and T4H are combined into a 16-bit register, T4L is the low byte, T4H is the high byte. When the 16-bit count value in [T4H, T4L] overflows, the system will automatically reload the internal 16-bit reload register

The value is loaded into [T4H, T4L].

### 14.4.5 Timer 3 8 - bit Prescaler Register (TM3PS)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TM3PS	FEA3H								

Timer 3 clock = system clock SYSclk  $\div$  ( TM3PS + 1 )

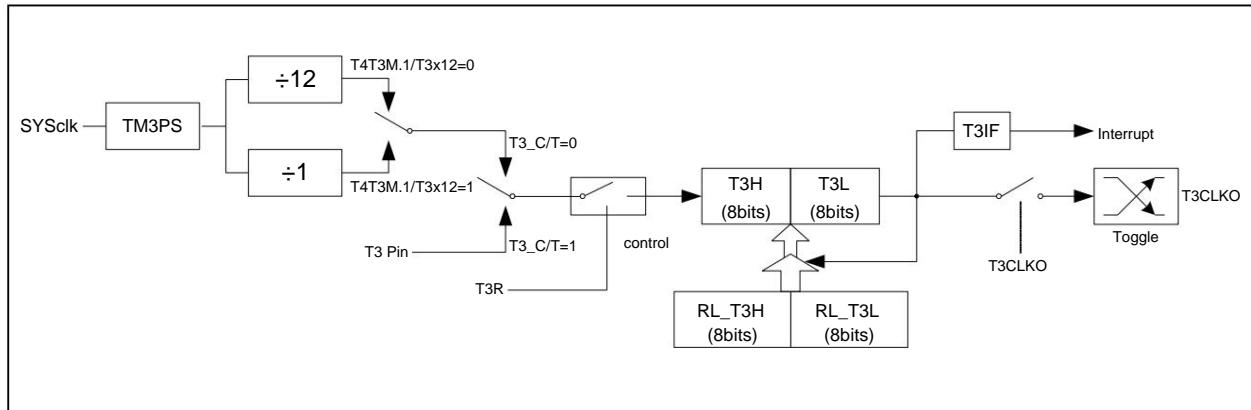
### 14.4.6 Timer 4 8 - bit Prescaler Register (TM4PS)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
TM4PS	FEA4H								

Timer 4 clock = system clock SYSclk  $\div$  ( TM4PS + 1 )

### 14.4.7 Timer 3 working mode

The functional block diagram of Timer/Counter 3 is as follows:



### Timer/Counter 3 operating mode: 16-bit auto-reload mode

T3R/T4T3M.3 is the control bit in the T4T3M register. For the specific function description of each bit of the T4T3M register, see the introduction of the T4T3M register in the previous section.

When T3\_C/T=0, the multiplexer is connected to the system clock output, T3 counts the internal system clock, and T3 works in timing mode. When T3\_C/T=1, the multiplexer is connected to the external pulse input T3, that is, T3 works in counting mode.

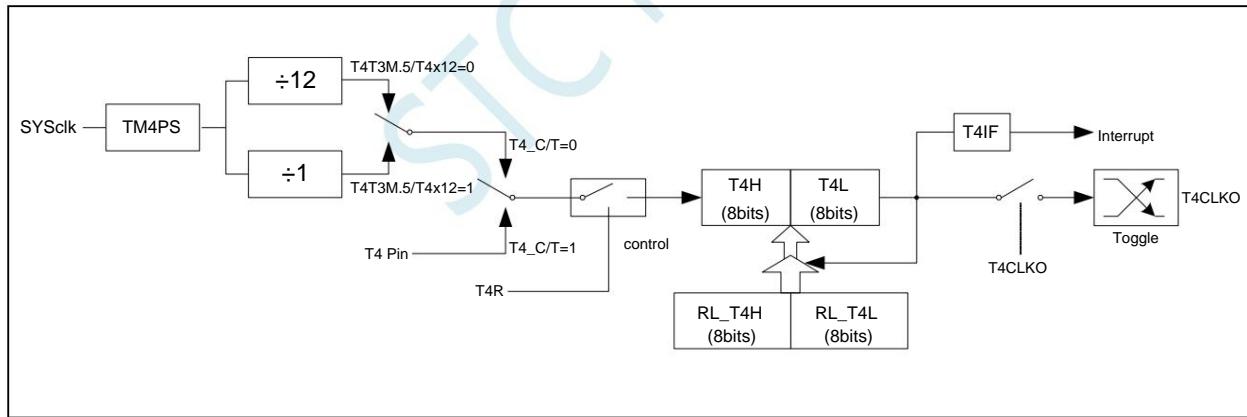
The timer 3 of STC microcontroller has two counting rates: one is 12T mode, which increases by 1 every 12 clocks, which is the same as that of traditional 8051 microcontroller; the other is 1T mode, which increases by 1 for each clock, and the speed is the speed of traditional 8051 microcontroller. 12 times. The rate of T3 is determined by T3x12 in the special function register T4T3M. 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 and T3H share the same address, RL\_T3L and T3L share the same address. When T3R=0, that is, timer/counter 3 is disabled, the content written to T3L will be written to RL\_T3L at the same time, and the content written to T3H will also be written to RL\_T3H at the same time. When T3R=1, that is, 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 the hidden register RL\_T3L, and the content written to T3H is actually written. Instead of writing to the current register T3H, it writes to the hidden register RL\_T3H, which cleverly implements a 16-bit reload timer. When reading the content of T3H and T3L, the content read is the content of T3H and T3L, not the content of RL\_T3H and RL\_T3L.

The overflow of [T3H, T3L] not only sets the interrupt request flag (T3IF) to make the CPU go to execute the interrupt program of timer 3, but also automatically automatically reload the contents of [RL\_T3H, RL\_T3L] into [T3H, T3L].

### 14.4.8 Timer 4 working mode

The functional block diagram of Timer/Counter 4 is as follows:



### Timer/Counter 4 operating mode: 16-bit auto-reload mode

T4R/T4T3M.7 is the control bit in the T4T3M register. For the specific function description of each bit of the T4T3M register, see the introduction of the T4T3M register in the previous section.

When T4\_C/T=0, the multiplexer is connected to the system clock output, T4 counts the internal system clock, and T4 works in timing mode. When T4\_C/T=1, the multiplexer is connected to the external pulse input T4, that is, T4 works in counting mode.

Timer 4 of STC microcontroller has two counting rates: one is 12T mode, which increases by 1 every 12 clocks, which is the same as that of traditional 8051 microcontroller; the other is 1T mode, which adds 1 to each clock, and the speed is the speed of traditional 8051 microcontroller. 12 times. The rate of T4 is determined by T4x12 in the special function register T4T3M. If T4x12=0, T4 works in 12T mode; if T4x12=1, T4 works in 1T mode

Timer 4 has two hidden registers RL\_T4H and RL\_T4L. RL\_T4H and T4H share the same address, RL\_T4L and T4L share the same address.

share the same address. When T4R=0, that is, timer/counter 4 is disabled, the content written to T4L will be written to RL\_T4L at the same time. The content written to T4H is also written to RL\_T4H at the same time. When T4R=1, that is, timer/counter 4 is allowed to work, write content to T4L, the actual In fact, it is not written into the current register T4L, but into the hidden register RL\_T4L, and the content written to T4H is not actually written. In the current register T4H, the hidden register RL\_T4H is written instead, so that the 16-bit reload timer can be implemented cleverly. When reading T4H When reading the content of T4L and T4L, the content read is the content of T4H and T4L, not the content of RL\_T4H and RL\_T4L.

The overflow of [T4H, T4L] not only sets the interrupt request flag (T4IF) to make the CPU go to execute the interrupt program of timer 4, but also automatically Automatically reload the contents of [RL\_T4H, RL\_T4L] into [T4H, T4L].

#### 14.4.9 Timer 3 calculation formula

timer speed	Period calculation formula
1T	timer period = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)}$ (auto-reload)
12T	timer period = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)} \times 12$ (auto-reload)

#### 14.4.10 Timer 4 calculation formula

timer speed	Period calculation formula
1T	timer period = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)}$ (auto-reload)
12T	timer period = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)} \times 12$ (auto-reload)

## 14.5 Example Program

### 14.5.1 Timer 0 (mode 0 - 16-bit auto-reload), used as a timer

---

```
//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10; //test port
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

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

    TMOD = 0x00; //mode|0
    TL0 = 0x66; //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1; //start timer
    ET0 = 1; //Enable timer interrupt
    EA = 1;

    while (1);
}
```

---

### 14.5.2 Timer 0 (mode 1 - 16 bits are not automatically reloaded), used as a timer

---

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

```
#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
```

```

#include "intrins.h"

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                                //parameters at reset
    TH0 = 0xfc;
    P10 = !P10;                                 //test port

}

void main()
{
    EAXFR = 1;                                  //Enable      XFR
    CKCON = 0x00;                               //access to set external data bus speed to fastest
    WTST = 0x00;                                //Set the program code to wait for parameters,
                                                //assign to      CPU The speed of executing the program is set to the fastest

    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
                                                //65536-11.0592M/12/1000
    TL0 = 0x66;
    TH0 = 0xfc;
    TR0 = 1;                                    //start timer
    ET0 = 1;                                    //Enable timer interrupt
    EA = 1;

    while (1);
}

```

### 14.5.3 Timer 0 (mode 2 - 8-bit auto-reload), used as a timer

---

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //test port
}

void main()
{
    EAXFR = 1;                                //enable access XFR

```

```

CKCON = 0x00; //Set the external data bus speed to the fastest
WTST = 0x00; //Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

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; //model2
TL0 = 0xf4; //256-11.0592M/12/76K
TH0 = 0xf4;
TR0 = 1; //start timer
ET0 = 1; //Enable timer interrupt
EA = 1;

while (1);
}

```

#### 14.5.4 Timer 0 (mode 3 - 16-bit auto-reload non-maskable interrupt), used as a timer

Time

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10; //test port
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

    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;                                //mode3
TL0 = 0x66;                                  //65536-11.0592M/ 12/1000
TH0 = 0xfc;
TR0 = 1;                                     //start timer
ET0 = 1;                                     //Enable timer interrupt
//EA = 1;                                     //out of control

while (1);
}

```

#### 14.5.5 Timer 0 (external counting - extended T0 is external falling edge interrupt)

```

//The test frequency is 11.0592MHz

#ifndef "stc8h.h"
#include "stc32g.h"                            //For header files, see Download Software
#include "intrins.h"

void TM0_Isr() interrupt 1
{
    P10 = !P10;                               //test port
}

void main()
{
    EAXFR = 1;                             //Enable XFR
    CKCON = 0x00;                           //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                            //assign to CPU The speed of executing the program is set to the fastest

    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 count mode
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                                //start timer
    ET0 = 1;                                //Enable timer interrupt
}

```

```

EA = 1;
while (1);
}

```

#### 14.5.6 Timer 0 (measurement pulse width - INT0 high level width)

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void INT0_Isr() interrupt 0 {

    P0 = TL0; //TL0 low byte of measured value
    P1 = TH0; //TH0 is the high byte of the measured value
    TL0 = 0x00;
    TH0 = 0x00;
}

void main()
{
    EA,XFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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 model
    TMOD = 0x08; //Enable GATE,INT0 time enable timer
    TL0 = 0x00;
    TH0 = 0x00;
    while (INT0); //wait for INT0
    TR0 = 1; //start timer
    IT0 = 1; //Enable falling edge interrupt
    EX0 = 1;
    EA = 1;

    while (1);
}

```

---

## 14.5.7 Timer 0 (Mode 0), Clock Divided Output

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                          //assign to      CPU   The speed of executing the program is set to the fastest

    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;                                         //model0
    TL0 = 0x66;                                         //65536-11.0592M/ 12/1000
    TH0 = 0xfc;
    TR0 = 1;                                            //start timer
    T0CLKO = 1;                                         //enable clock output

    while (1);
}

```

## 14.5.8 Timer 1 (mode 0 - 16-bit auto-reload), used as a timer

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                         //test port
}

void main()
{
    EAXFR = 1;                                         //enable access XFR

```

```

CKCON = 0x00; //Set the external data bus speed to the fastest
WTST = 0x00; //Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

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; //model0
TL1 = 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 are not automatically reloaded), used as a timer

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    TL1 = 0x66; //parameters at reset
    TH1 = 0xfc;
    P10 = !P10; //test port
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00;
    WTST = 0x00;

    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;                                //mode1
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), used as a timer

---

//The test frequency is 11.0592MHz

*For header files, see Download Software*

```

#ifndef "stc8h.h"
#include "stc32g.h"
#include "intrins.h"

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                //test port
}

void main()
{
    EAXFR = 1;                                 //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                               //Set the program code to wait for parameters,
                                                //assign to CPU   The speed of executing the program is set to the fastest

    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;                                //mode2
    TL1 = 0xf4;                                //256-11.0592M/ 12/76K
    TH1 = 0xf4;
    TR1 = 1;                                   //start timer
    ET1 = 1;                                   //Enable timer interrupt
}

```

```

EA = 1;

while (1);

}

```

#### 14.5.11 Timer 1 (external counting - extended T1 is external falling edge interrupt)

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void TM1_Isr() interrupt
{
    P10 = !P10; //test port
}

void main()
{
    EAXF = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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 count mode
    TL1 = 0xff;
    TH1 = 0xff;
    TR1 = 1; //start timer
    ET1 = 1; //Enable timer interrupt
    EA = 1;

    while (1);
}

```

#### 14.5.12 Timer 1 (measurement pulse width - INT1 high level width)

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void INT1_Isr() interrupt 2 {

    P0 = TL1;                                         // TL1 low byte of measured value
    P1 = TH1;                                         // TH1 is the high byte of the measured value
    TL1 = 0x00;
    TH1= 0x00;
}

void main()
{
    EAXFR = 1;                                       // Enable XFR
    CKCON = 0x00;                                     // access to set external data bus speed to fastest
    WTST = 0x00;                                      // Set the program code to wait for parameters,
                                                       // assign to CPU The speed of executing the program is set to the fastest

    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 model
    TMOD = 0x80;                                     // Enable GATE,INT1 for 1 enable timing
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);
    TR1 = 1;                                         // start timer
    IT1 = 1;                                         // Enable falling edge interrupt
    EX1 = 1;
    EA = 1;

    while (1);
}

```

### 14.5.13 Timer 1 (Mode 0), Clock Divided Output

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void main()

```

```

{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                            //assign to CPU The speed of executing the program is set to the fastest

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

    TMOD = 0x00;                                //mode|0
    TL1 = 0x66;                                //65536-11.0592M/ 12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //start timer
    T1CLKO = 1;                                //enable clock output

    while (1);
}

```

#### 14.5.14 Timer 1 (mode 0) as serial port 1 baud rate generator

---

```

//The test frequency is 11.0592MHz

#ifndef "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

#define FOSC          11059200UL                  //Define as an unsigned long integer to avoid calculation overflow
#define BRT           (65536-FOSC/115200/4)

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      XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to      CPU  The speed of executing the program is set to the fastest

    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

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT       (256-FOSC/115200/32)

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      XFR
    CKCON = 0x00;                            //access to set external data bus speed to fastest
    WTST = 0x00;                             //Set the program code to wait for parameters,
                                                //assign to      CPU The speed of executing the program is set to the fastest

    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), used as timer

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

```

```

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                //test port
}

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                            //assign to      CPU The speed of executing the program is set to the fastest

    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 timer
    ET2 = 1;                                 //Enable timer interrupt
    EA = 1;

    while (1);
}

```

#### 14.5.17 Timer 2 (external counting - extended T2 is external falling edge interrupt)

---

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                //test port
}

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                            //assign to      CPU The speed of executing the program is set to the fastest
}

```

```

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 the external count mode and start the timer
ET2 = 1;                                         //Enable timer interrupt
EA = 1;

while (1);
}

```

### 14.5.18 Timer 2, clock divider output

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     //For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to      CPU The speed of executing the program is set to the fastest

    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;                                         //start timer
    T2R = 1;                                            //enable clock output
    T2CLKO = 1;
}

```

```

    while (1);
}

```

### 14.5.19 Timer 2 as serial port 1 baud rate generator

---

//The test frequency is 11.0592MHz

```

#ifndef "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL                      //Define as an unsigned long integer to avoid calculation overflow
#define BRT       (65536-FOSC/115200/4)

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()
{
    EAXF = 1;           // enable access XFR
    CKCON = 0x00;
    WTST = 0x00;
    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 as serial port 2 baud rate generator

---

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL
#define BRT      (65536-FOSC/115200/4)                //Define as an unsigned long integer to avoid calculation overflow

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 = 0x10;
    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 XFR
    CKCON = 0x00;   // Set the external data bus speed to the fastest
    WTST = 0x00;    // Set the program code to wait for parameters,
                    // assign to CPU. The speed of executing the program is set to the fastest
    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 as serial port 3 baud rate generator

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL
#define BRT       (65536-FOSC/115200/4) //Define as an unsigned long integer to avoid calculation overflow

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 XFR
    CKCON = 0x00;   //Set the external data bus speed to the fastest
    WTST = 0x00;    //Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    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 as serial port 4 baud rate generator

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT       (65536-FOSC/115200/4)

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 XFR
    CKCON = 0x00;
    WTST = 0x00;
    //Set the external data bus speed to the fastest
    //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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), used as timer

---

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#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 XFR
    CKCON = 0x00;                            //Set the external data bus speed to the fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                                //assign to CPU The speed of executing the program is set to the fastest

    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 Counting - Extended T3 for External Falling Edge Interrupt)

---

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#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 XFR
    CKCON = 0x00;                            //Set the external data bus speed to the fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                                //assign to CPU The speed of executing the program is set to the fastest
}

```

```
//assign as 0 can be CPU The speed of executing the program is set to the fastest

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 the external count mode and start the timer
IE2 = ET3;              //Enable timer interrupt
EA = 1;

while (1);
}
```

---

## 14.5.25 Timer 3, clock divider output

```
//The test frequency is 11.0592MHz

##include "stc8h.h"
##include "stc32g.h"          //For header files, see Download Software
##include "intrins.h"

void main()
{
    EAXFR = 1;                //Enable XFR
    CKCON = 0x00;              //access to set external data bus speed to fastest
    WTST = 0x00;               //Set the program code to wait for parameters,
                                //assign to CPU The speed of executing the program is set to the fastest

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

T3L = 0x66;                 //65536-11.0592M/12/1000
T3H = 0xfc;
T4T3M = 0x09;               //Enable clock output and start timer
```

```

    while (1);
}

```

#### 14.5.26 Timer 3 as serial port 3 baud rate generator

//The test frequency is 11.0592MHz

```

#ifndef "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT      (65536-FOSC/115200/4)

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 XFR
    CKCON = 0x00;        //Set the external data bus speed to the fastest
    WTST = 0x00;         //Set the program code to wait for parameters,
                        //assign to CPU The speed of executing the program is set to the fastest

    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), used as timer

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"          //For header files, see Download Software
#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 XFR
    CKCON = 0x00; //Set the external data bus speed to the fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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; //start timer
    T4T3M = 0x80; //Enable timer interrupt
    IE2 = ET4;
    EA = 1;

    while (1);
}

```

#### 14.5.28 Timer 4 (External Counting - Extended T4 for External Falling Edge Interrupt)

---

//The test frequency is 11.0592MHz

```

##include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#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 XFR
    CKCON = 0x00; //Set the external data bus speed to the fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    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 count mode and start the timer
    IE2 = ET4; //Enable timer interrupt
    EA = 1;

    while (1);
}

```

### 14.5.29 Timer 4, Clock Divided Output

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    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 as serial port 4 baud rate generator

---

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h"                           //For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL                  //Define as an unsigned long integer to avoid calculation overflow
#define BRT       (65536-FOSC/115200/4)

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 XFR
    CKCON = 0x00;
    WTST = 0x00;
    //Set the external data bus speed to the fastest
    //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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 communication (USART1, USART2)

The STC32G series microcontrollers have 2 full-duplex synchronous/asynchronous serial communication interfaces (USART1 and USART2). each serial port It consists of 2 data buffers, a shift register, a serial control register and a baud rate generator. of each serial port The data buffer consists of two independent receiving and sending buffers, which can send and receive data at the same time.

Serial port 1 and serial port 2 of STC32G series microcontrollers have 4 working modes, of which the baud rate of two modes is variable, and the other two species are fixed for selection in different applications. Users can use software to set different baud rates and choose different working modes. The host can Receive/transmit program processing by polling or interrupt mode, which is very flexible to use.

The communication ports of serial port 1 and serial port 2 can be switched to multiple groups of ports through the switching function of the function pins, so that one communication port can be Time-division multiplexing into multiple communication ports.

### 15.1 Serial port function pin switch

Symbol 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 selection bit

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

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

S2\_S: Serial port 2 function pin selection bit

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

Symbol 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 selection bit of 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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

S1SPI\_S[1:0]: SPI function pin selection bit of 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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

## 15.2 Serial port related registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	Serial port 1 control register 98H	\$M0/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	\$S2M0/FE \$2SM1			S2SM2 S2REN S2TB8 S2RB8 S2TI					S2RI	0000,0000
S2BUF	Serial port 2 data register 9BH										0000,0000
PCON	Power Control Register	87H \$MOD 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

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
S2CFG	Serial Port 2 Configuration Register	7EFDB4H	-	S2MOD0 S2M0x6	-	-	-	-	-	-	W1 000xxxx0
S2ADDR	Serial port 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 SPSEL 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 guard 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 SPSEL 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 guard 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 port USART)

### 15.3.1 Serial Port 1 Control Register (SCON)

Symbol 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. When the UART detects during reception

When an invalid stop bit is reached, this bit is set by the UART receiver and must be cleared by software. When SMOD0 in the PCON register

When the bit is 0, this bit and SM1 together specify the communication working mode of serial port 1, as shown in the following table:

SM0	SM1	Serial port 1 working mode	Function Description
0	0	Mode 0	Synchronous shift serial method
0	1	Mode 1 Variable baud rate 8-bit data mode	
1	0	Mode 2 Fixed baud rate 9-bit data mode	
1	1	Mode 3 Variable baud rate 9-bit data mode	

SM2: Enable mode 2 or mode 3 multi-computer communication control bit. When serial port 1 uses mode 2 or mode 3, if the SM2 bit is 1 and REN

When the bit is 1, the receiver is in the address frame screening state. At this time, the received ninth bit (ie RB8) can be used to filter the address frame,

If RB8=1, it means that the frame is an address frame, the address information can enter SBUF, and RI is set to 1, and then in the interrupt service routine

Then compare the address numbers; if RB8=0, it means that the frame is not an address frame, it should be discarded and RI=0. in mode 2 or mode 3

, if the SM2 bit is 0 and the REN bit is 1, the receiver is in the address frame filtering disabled state, regardless of the received RB8

0 or 1, can make the received information enter SBUF, and make RI=1, at this time, RB8 is usually the parity bit. Mode 1 and Mode 0

For non-multi-machine communication mode, in these two modes, SM2 should be set to 0.

REN: enable/disable serial port receive control bit

0: Forbid the serial port to receive data

1: Allow the serial port to receive data

TB8: When serial port 1 uses mode 2 or mode 3, TB8 is the ninth bit of data to be sent, which can be set or cleared by software as required. in mold

In Mode 0 and Mode 1, this bit is not used.

RB8: When serial port 1 uses mode 2 or mode 3, RB8 is the ninth bit of data received, which is generally used as check bit or address frame/data

frame flags. In Mode 0 and Mode 1, this bit is not used.

TI: Serial port 1 sends interrupt request flag. In mode 0, when the 8th bit of the data sent by the serial port ends, the TI is automatically set to 1 by the hardware.

An interrupt is requested from the host, and TI must be cleared by software after the interrupt is responded to. In other modes, it is set by hardware when the stop bit begins to transmit

Automatically set TI to 1, and send a request to the CPU to interrupt. After responding to the interrupt, TI must be cleared by software.

RI: Serial port 1 receive interrupt request flag. In mode 0, when the serial port receives the 8th bit of data, the hardware will automatically set RI to 1.

Request an interrupt from the host, and RI must be cleared by software after responding to the interrupt. In other modes, the intermediate moment when the stop bit is received by the serial

RI is automatically set to 1 by hardware, and an interrupt request is sent to the CPU. After responding to the interrupt, RI must be cleared by software.

### 15.3.2 Serial port 1 data register (SBUF)

Symbol address B7			B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF: Serial port 1 data receive/transmit buffer. SBUF is actually two buffers, read buffer and write buffer, the two operations are respectively

Corresponding to two different registers, one is a write-only register (write buffer), and the other is a read-only register (read buffer). right

SBUF read operation is actually to read the serial port receive buffer, and write operation to SBUF is to trigger the serial port to start sending data.

according to.

### 15.3.3 Power Management Register (PCON)

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

SMOD: Serial port 1 baud rate control bit

0: The baud rate of each mode of serial port 1 is not doubled

1: Serial port 1 mode 1 (valid when using timer 1 of mode 2 as a baud rate generator), mode 2, mode 3 (using mode 2

(active when Timer 1 is used as a baud rate generator) doubles the baud rate

SMOD0: Frame Error Detection Control Bit

0: No frame error detection function

1: Enable frame error detection function. At this time, SM0/FE of SCON is the FE function, which is the frame error detection flag bit.

### 15.3.4 Auxiliary Register 1 (AUXR)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6 T2R T2_C/T	T2x12 EXTRAM	S1BRT			

UART\_M0x6: Communication speed control of serial port 1 mode 0

0: The baud rate of serial port 1 mode 0 is not doubled, fixed to Fosc/12

1: The baud rate of serial port 1 mode 0 is 6 times faster, that is, it is fixed to Fosc/12\*6 = Fosc/2

S1BRT: Serial port 1 baud rate transmitter selection bit

0: select Timer 1 as the 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 timer 1 is not recommended. Timer 2 can be shared at the same time

Baud rate generator for serial port 1, serial port 2, serial port 3 and serial port 4

### 15.3.5 Serial port 1 mode 0, mode 0 baud rate calculation formula

When the serial port 1 selects the working mode as mode 0, the serial communication interface works in the synchronous shift register mode.

When the communication speed setting bit UART\_M0x6 is 0, its baud rate is fixed to the 12 frequency division of the system clock (SYSclk/12); when set

When UART\_M0x6 is 1, its baud rate is fixed to 2 divided by the system clock frequency (SYSclk/2). RxD is the data port of serial communication,

TxD is the synchronous shift pulse output pin, which transmits and receives 8-bit data, low-order first.

Transmission process of mode 0: When the host executes the command to write data into the transmission buffer SBUF, the transmission is started, and the serial port is about 8 digits.

According to the baud rate of SYSclk/12 or SYSclk/2 (12 or 2 determined by UART\_M0x6) output from RxD pin (from

Low bit to high bit), the interrupt flag TI is set to 1 after sending, and the TxD pin outputs a synchronous shift pulse signal. When the write signal is valid, it is separated by one

Clock, the sending control terminal SEND is valid (high level), allowing RxD to send data, and allowing TxD to output synchronous shift pulses. One frame (8

Bit) When the data transmission is completed, each control terminal returns to the original state, only TI maintains a high level, which is in the state of interrupt application. sending data again

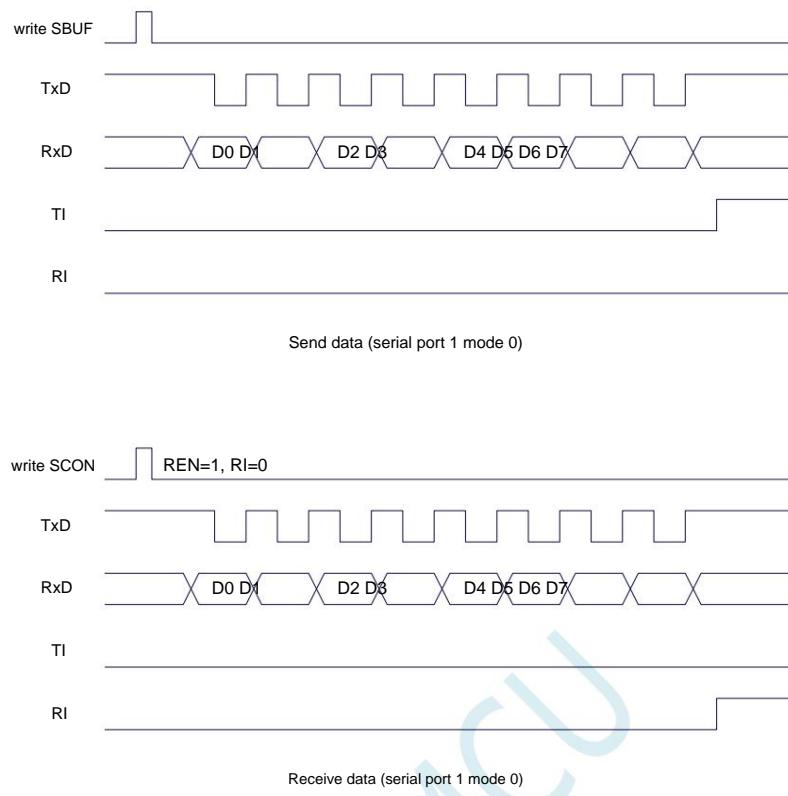
Before, TI must be cleared by software.

The receiving process of mode 0: first, clear the receive interrupt request flag RI and set the enable receive control bit REN to start the mode 0 connection.

collection process. After starting the receiving process, RxD is the serial data input terminal, and TxD is the synchronization pulse output terminal. The baud rate for serial reception is

SYSclk/12 or SYSclk/2 (divide by 12 or 2 determined by UART\_M0x6). After receiving a frame of data (8 bits),

The control signal is reset, the interrupt flag RI is set to 1, and it is in the state of interrupt request. When receiving again, RI must be cleared to 0 by software



When working in mode 0, the multi-computer communication control bit SM2 must be cleared so that it does not affect the TB8 and RB8 bits. Since the baud rate is fixed

For SYScclk/12 or SYScclk/2, no timer is required, and the clock of the microcontroller is directly used as a synchronous shift pulse.

The baud rate calculation formula of serial port 1 mode 0 is shown in the following table (SYScclk is the system operating frequency):

UART_M0x6 baud rate calculation formula	
0	baud rate = $\frac{\text{SYScclk}}{12}$
1	baud rate = $\frac{\text{SYScclk}}{2}$

### 15.3.6 Serial port 1 mode 1, mode 1 baud rate calculation formula

When software sets SM0 and SM1 of SCON to "01", serial port 1 works in mode 1. This mode is 8-bit UART format, and one frame of information is 10 bits: 1 start bit, 8 data bits (low-order first) and 1 stop bit. The baud rate is variable, you can set the baud rate according to your needs. TxD is the data transmitting port, RxD is the data receiving port, and the serial port is full-duplex receiving/transmitting.

Transmission process of mode 1: When transmitting in serial communication mode, the data is output by the serial transmitter TxD. When the host executes an instruction to write SBUF, it starts the transmission of serial communication, and the write "SBUF" signal also loads "1" into the ninth bit of the transmit shift register, and informs the TX control unit to start transmitting. The shift register continuously shifts the data to the right to the TxD port for transmission, and continuously shifts in "0" on the left side of the data for supplementation. When the highest displacement of the data is to the output position of the shift register, the ninth bit is "1", and the bits on the left of it are all "0".

This state condition makes the TX control unit make the last shift output, and then invalidates the allow-to-send signal "SEND" to complete a frame of information. information is sent, and the interrupt request bit TI is set, that is, TI=1, to request interrupt processing from the host.

The receiving process of mode 1: when the software sets the receiving enable flag bit REN, that is, when REN=1, the receiver will respond to the signal of the RxD port. Perform detection, when it is detected that the RxD port sends a falling edge transition from "1" → "0", the receiver is started to prepare to receive data, and immediately Resets the receive counter of the baud rate generator and loads the shift register with 1FFH. Received data is shifted in from the right side of the receive shift register, The loaded 1FFH is shifted to the left. When the start bit "0" is moved to the leftmost of the shift register, the RX controller is shifted for the last time.

Complete the reception of one frame. If both of the following conditions are met:

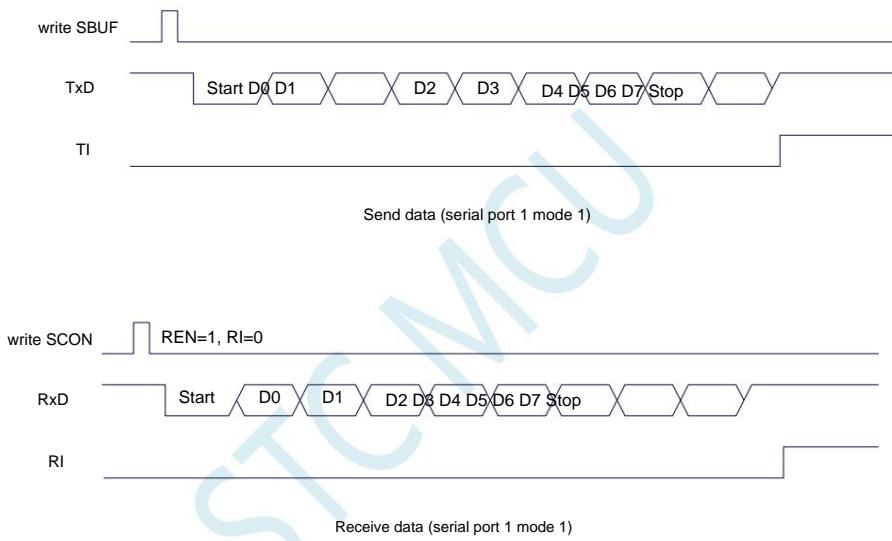
- RI=0;
- SM2=0 or the received stop bit is 1.

Then the received data is valid, it is loaded into SBUF, the stop bit is entered into RB8, the RI flag is set to 1, and an interrupt is requested from the host.

If the above two conditions cannot be met at the same time, the received data will be invalid and lost. Regardless of whether the conditions are met or not, the receiver will detect the RxD terminal again.

The transition of "1" → "0" on the port continues to receive the next frame. Receive is valid, after responding to the interrupt, the RI flag must be cleared by software. Pass

Normally, when serial communication works in mode 1, SM2 is set to "0".



The baud rate of serial port 1 is variable, and its baud rate can be generated by timer 1 or timer 2. When the timer is in 1T mode (12 double speed), the corresponding baud rate speed will be increased by 12 times accordingly.

The baud rate calculation formula of serial port 1 mode 1 is shown in the following table: (SYSclk is the system operating frequency)

choose timer	timer speed	Reload value calculation formula	baud rate
timer 2	1T	timer 2 reload value = 65536 - $\frac{SYSclk}{4 \times \text{baud rate}}$	baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$
	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 1 Mode 0	1T	timer 1 reload value = 65536 - $\frac{SYSclk}{4 \times \text{baud rate}}$	baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$
	12T	timer 1 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 1 Mode 2	1T	timer 1 reload value = 256 - $\frac{2 \text{SMOD} \times SYSclk}{32 \times \text{baud rate}}$	baud rate = $\frac{2 \text{SMOD} \times SYSclk}{32 \times (256 - \text{number of timer reloads})}$
	12T	timer 1 reload value = 256 - $\frac{2 \text{SMOD} \times SYSclk}{12 \times 32 \times \text{baud rate}}$	baud rate = $\frac{2 \text{SMOD} \times SYSclk}{12 \times 32 \times (256 - \text{number of timer reloads})}$

The following is the reload value of the timer corresponding to the common frequency and common baud rate

frequency (MHz)	baud rate	Timer 2		Timer 1 Mode 0		Timer 1 Mode 2			
		1T mode	12T mode	1T mode	12T mode	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 FF00H	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 the two bits of SM0 and SM1 are 10, serial port 1 works in mode 2. Serial port 1 working mode 2 is 9-bit data asynchronous communication UART mode, the information of one frame consists of 11 bits: 1 start bit, 8 data bits (low order first), 1 programmable bit (9th bit data) and 1 stop bit. The programmable bit (the 9th bit of data) is provided by TB8 in SCON during transmission, and can be set to 1 or 0 by software, or the odd/even parity bit P value in PSW can be loaded into TB8 (TB8 can be used as a multi-computer The address data flag bit in communication can also be used as the parity bit of the data). On reception, the ninth bit of data is loaded into RB8 of SCON. TxD is the transmit port, RxD is the receive port, and receives/transmits in full duplex mode.

The baud rate of mode 2 is fixed to the system clock divided by 64 or 32 (depending on the value of SMOD in PCON). The baud rate calculation formula of serial port 1 mode 2 is shown in the following table (SYSclk is the system operating frequency):

SMOD baud rate calculation formula	
0	baud rate = $\frac{\text{SYSclk}}{64}$
1	baud rate = $\frac{\text{SYSclk}}{32}$

Compared with mode 1, except that the baud rate generation source is slightly different, and the ninth data bit provided to the shift register by TB8 is different during transmission, the rest of the functional structure is basically the same, and the receiving/transmitting operation process and timing are also different. basically the same.

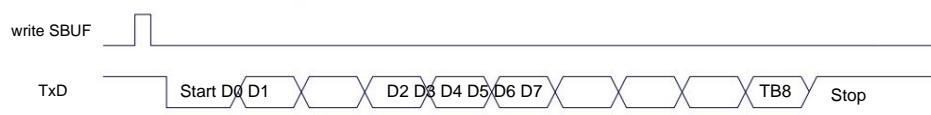
When the receiver receives a frame of information, the following conditions must be met at the same time:

- RI=0

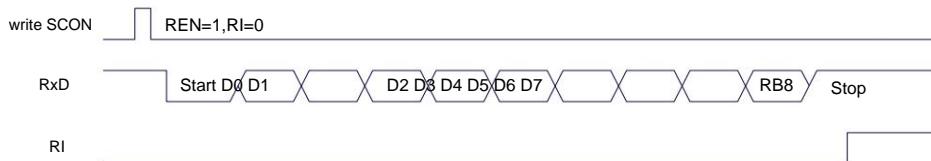
- SM2=0 or SM2=1 and the received ninth data bit RB8=1. When the above two

conditions are satisfied at the same time, the received data of the shift register will be loaded into SBUF and RB8, the RI flag will be set to 1, and the host will be requested to interrupt processing. If one of the above conditions is not satisfied, the data just received in the shift register is invalid and lost, and RI is not set. Regardless of whether the above conditions are satisfied or not, the receiver starts to detect the transition information of the RxD input port again, and receives the input information of the next frame. In Mode 2, the received stop bits are independent of SBUF, RB8 and RI.

The setting of SM2 and TB8 in SCON and the agreement of communication protocol through the software provide convenience for 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 the two bits of SM0 and SM1 are 11, serial port 1 works in mode 3. Serial communication mode 3 is 9-bit data asynchronous communication UART mode, the information of one frame consists of 11 bits: 1 start bit, 8 data bits (low order first), 1 programmable bit (9th bit data) and 1 stop bit. A programmable bit (9th data bit) on transmit is provided by TB8 in SCON and can be set to 1 or 0 by software, or can be Load the odd/even parity bit P value in PSW into TB8 (TB8 can be used as both the address data flag bit in multi-machine communication and the parity bit). On reception, the ninth bit of data is loaded into RB8 of SCON. TxD is the transmit port, RxD is the receive port, in full duplex mode to receive/transmit.

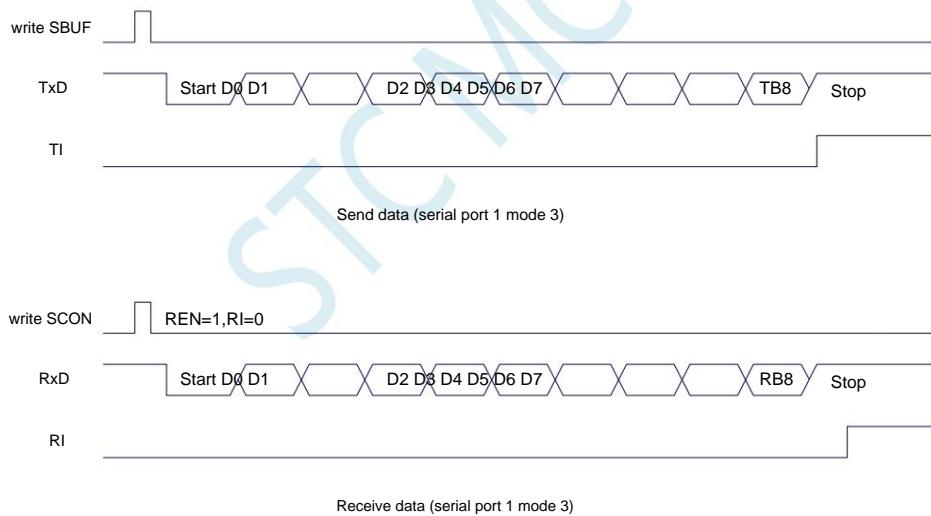
Compared with mode 3, mode 3 is basically the same as mode 1, except that the 9th data bit provided to the shift register by TB8 is different during transmission. The same, its receiving and sending operation process and timing are basically the same.

When the receiver receives a frame of information, the following conditions must be met at the same time:

- RI=0
- SM2=0 or SM2=1 and the received ninth data bit RB8=1.

When the above two conditions are satisfied at the same time, the received data of the shift register is loaded into SBUF and RB8, the RI flag is set to 1, and request interrupt processing from the host. If one of the above conditions is not satisfied, the data in the shift register just received is invalid and lost, and RI is not set. Regardless of whether the above conditions are satisfied or not, the receiver starts to detect the transition information of the RxD input port again, and receives the next frame. Enter information. In Mode 3, the received stop bits are independent of SBUF, RB8 and RI.

The setting of SM2 and TB8 in SCON and the agreement of the communication protocol by the software provide convenience for multi-machine communication.



The baud rate calculation formula of serial port 1 mode 3 is exactly the same as that of mode 1. Please refer to the baud rate calculation formula of Mode 1.

### 15.3.9 Automatic address recognition

#### 15.3.10 Serial port 1 slave address control register (SADDR, SADEN)

Symbol 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-machine communication. The main principle is that the slave system uses the hardware comparison function to identify the address information in the serial port data stream of the host, the slave address of the local machine is set by the registers SADDR and SADEN, and the hardware automatically matches the address information. The slave address is filtered. When the slave address information from the host matches the slave address set by the machine, 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 address will wake up from idle mode, which can greatly reduce the power consumption of the slave MCU, even if the slave is working normally. Only the slave with matching address will wake up from idle mode, which can greatly reduce the power consumption of the slave MCU, even if the slave is working normally.

The state can also avoid the continuous entry of serial port interruption and reduce the system execution efficiency.

To use the automatic address recognition function of the serial port, you first need to set the serial port communication mode of the MCU participating in the communication to mode 2 or Mode 3 (usually select mode 3 with a variable baud rate, because the baud rate of mode 2 is fixed and inconvenient to adjust), and turn on the slave's SM2 bit of SCON. For serial port mode 2 or mode 3 of 9 data bits, the 9th data (stored in RB8) is the address/Data flag bit, when the ninth bit of data is 1, it means that the previous 8-bit data (stored in SBUF) is address information. When SM2 When set to 1, the slave MCU will automatically filter out non-address data (data whose ninth bit is 0), while the address data in SBUF will be filtered out. (Data whose 9th bit is 1) is automatically compared with the local address set by SADDR and SADEN. RI is set to "1" and an interrupt is generated, otherwise the serial port data received this time will not be processed.

The setting of the slave address is set through the SADDR and SADEN registers. SADDR is the slave address register, in which stores the slave address of the machine. SADEN is the slave address mask bit register, which is used to set the ignore bit in the address information. The setting method as follows:

E.g

SADDR = 11001010

SADEN = 10000001

Then the matching address is 1xxxxxx0

That is, as long as bit0 in the address data sent by the host is 0 and bit7 is 1, it can match the local address

Another example

SADDR = 11001010

SADEN = 00001111

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, and the upper 4 bits are ignored, and you can to any value.

The master can use the broadcast address (FFH) to select all slaves at the same time for communication.

### 15.3.11 Serial Port 1 Synchronous Mode Control Register 1 (USARTCR1)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
USARTCR1 7	EFDC0H	LINEN	DORD	CLKEN	SPMOD	SPIEN			SPSLV CPOL CPHA

LINEN: LIN mode enable bit

0: Disable LIN mode

1: Enable LIN mode

DORD: The order in which data bits are sent/received in SPI mode

0: Send/receive the high-order bit (MSB) of the data first

1: Send/receive the low-order bit (LSB) of the data first

CLKEN: SmartCard mode clock output control bit

0: Disable clock output

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 in master mode

1: SPI is in slave mode

CPOL: SPI Clock Polarity Control

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

1: High level when SCLK is idle, the front clock edge of SCLK is a falling edge, and the rear clock edge is a rising edge

CPHA: SPI Clock Phase Control

0: The data SS pin drives the first bit of data at a low level and changes the data on the trailing edge of SCLK, and samples the data on the leading edge of the clock

1: Data is driven on the leading edge of SCLK and sampled on the trailing edge

### 15.3.12 Serial Port 1 Synchronous Mode Control Register 2 (USARTCR2)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0	
USARTCR2 7EF	DC1H	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 the output of NACK signal

1: Enable output NACK signal

HDSEL: single-wire half-duplex mode enable bit

0: Disable single-line half-duplex mode

1: Enable single-wire half-duplex mode

PCEN: Check Bit Control Enable

0: Disable parity bit control function

1: Enable parity bit control function

PS: check digit mode selection

0: even test

1: odd parity

PE: Check bit error flag (must be cleared by software)

0: No check error

1: There is a verification error

### 15.3.13 Serial Port 1 Synchronous Mode Control Register 3 (USARTCR3)

Symbol 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 Synchronous Mode Control Register 4 (USARTCR4)

Symbol 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 Synchronous Mode Control Register 5 (USARTCR5)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
USARTCR5 7EFDC4H	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC

BRKDET: LIN slave mode interval field (BREAK) detection flag

0: No LIN gap field detected

1: LIN interval field detected, need to be cleared by software

HDRER: LIN slave mode header (HEADER) error

0: No LIN header error detected

1: LIN message header error detected, need to be cleared by software

SLVEN: LIN Slave Mode Enable Bit

0: LIN is master mode

1: LIN is in slave mode

ASYNC: LIN automatic synchronization function enable bit

0: Disable automatic synchronization

1: Enable automatic synchronization

**TXCF:** Transmission collision flag. Single-wire half-duplex mode, LIN mode and SmartCard mode are valid

0: No transmission collision detected (sent data same as received data)

1: Transmission collision detected (data sent is different from data received)

**SENDBK:** Send interval field. The software writes 1 to trigger the transmission interval field, and the hardware automatically clears it to 0 after the transmission is completed.

**HDRDET:** LIN slave mode header (HEADER) detection flag

0: LIN header not detected

1: LIN header is detected and needs to be cleared by software

**SYNC:** LIN slave mode sync field detection flag.

After correct analysis of the sync field, the hardware sets the SYNC flag to 1. SYNC is automatically cleared by hardware when receiving a marker field

### 15.3.16 Serial Port 1 Synchronous Mode Guard Time Register (USARTGTR)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
USARTGTR	7E	FDC5H							

The USARTGTR register is only valid in SmartCard mode. This register gives the guard time value according to the number of baud rate clocks. The TI logo is in

Set to 1 by hardware when the data bit sent by the SmartCard is equal to the guard 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 (USARTBRR)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
USARTBRH	7E	FDC6H				USARTBRR[15:8]			
USARTBRL	7E	FDC7H				USARTBRR[7:0]			

Baud rate calculation formula in LIN mode, IrDA mode and SmartCard mode

$$\text{Sync Baud Rate} = \frac{\text{SYSclk}}{16^* \text{ USARTBRR}[15:0]}$$

## 15.4 Serial port 2 (synchronous/asynchronous serial port USART2)

### 15.4.1 Serial Port 2 Control Register (S2CON)

Symbol 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 UART2 is receiving

This bit is set by the UART2 receiver when an invalid stop bit is detected and must be cleared by software. When the S2CFG register is

When the S2MOD0 bit is 0, this bit and S2SM1 together specify the communication working mode of serial port 2, as shown in the following table:

S2SM0	S2SM1	Serial port 2 working mode	Function Description
0	0	Mode 0	Synchronous shift serial method
0	1	Mode 1 Variable baud rate 8-bit data mode	
1	0	Mode 2 Fixed baud rate 9-bit data mode	
1	1	Mode 3 Variable baud rate 9-bit data mode	

S2SM2: Enable mode 2 or mode 3 multi-computer 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 ninth bit (ie S2RB8) can be used to

Filter the address frame, if S2RB8=1, it means the frame is an address frame, the address information can enter S2BUF, and make S2RI 1, enter

And then compare the address number in the interrupt service routine; if S2RB8=0, it means that the frame is not an address frame, it should be discarded and kept

S2RI=0. In Mode 2 or Mode 3, if the S2SM2 bit is 0 and the S2REN bit is 1, the receiver is in address frame filtering

In the disabled state, no matter the received S2RB8 is 0 or 1, the received information can be entered into S2BUF, and S2RI=1, this

When S2RB8 is usually the parity bit. Mode 1 and Mode 0 are non-multi-machine communication modes, in these two modes, S2SM2 should be set  
is 0.

S2REN: enable/disable serial port receive control bit

0: Forbid the serial port to receive data

1: Allow the serial port to receive data

S2TB8: When serial port 2 uses mode 2 or mode 3, S2TB8 is the ninth bit of data to be sent, which is set or cleared by software as needed.

In Mode 0 and Mode 1, this bit is not used.

S2RB8: When serial port 2 uses mode 2 or mode 3, S2RB8 is the ninth bit of data received, which is generally used as a check digit or address frame  
/Data frame flag. In Mode 0 and Mode 1, this bit is not used.

S2TI: Serial port 2 sends interrupt request flag. In mode 0, when the 8th bit of the data sent by the serial port ends, the S2TI is automatically converted by the hardware.

Set to 1 to request an interrupt from the host. After responding to the interrupt, S2TI must be cleared by software. In other modes, the transmission starts at the stop bit

S2TI is automatically set to 1 by the hardware, and an interrupt request is sent to the CPU. After responding to the interrupt, S2TI must be cleared by software.

S2RI: Serial port 2 receive interrupt request flag. In mode 0, when the serial port receives the 8th bit data, the hardware will automatically convert the S2RI

Set to 1 to request an interrupt from the host. After responding to the interrupt, S2RI must be cleared by software. In other modes, the serial received stop bit is

At the middle moment, the hardware will automatically set S2RI to 1, and send an interrupt request to the CPU. After responding to the interrupt, S2RI must be cleared by software.

### 15.4.2 Serial port 2 data register (S2BUF)

Symbol address B7			B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

S2BUF: Serial port 1 data receive/transmit buffer. S2BUF is actually two buffers, read buffer and write buffer, the two operations are divided into

Do not correspond to two different registers, one is a write-only register (write buffer), and the other is a read-only register (read buffer). right

When S2BUF is read, it actually reads the serial port receive buffer, and when S2BUF is written, it triggers the serial port to start sending data.

### 15.4.3 Serial port 2 configuration register (S2CFG)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
S2CFG	7EFDB4H	-	S2MOD0	S2M0x6					W1

S2MOD0: Framing Error Detection Control Bit

0: No frame error detection function

1: Enable frame error detection function. At this time, S2SM0/FE of S2CON is the FE function, which is the frame error detection flag bit.

S2M0x6: Communication speed control of serial port 2 mode 0

0: The baud rate of serial port 2 mode 0 is not doubled, fixed to Fosc/12

1: The baud rate of serial port 2 mode 0 is 6 times faster, that is, it is fixed to Fosc/12\*6 = Fosc/2

**W1:** When serial port 2 needs to be used, this bit must be set to "1", otherwise unexpected errors may occur. If you do not need to use a serial port 2, you do not need to set W1 specially.

### 15.4.4 Serial port 2 mode 0, mode 0 baud rate calculation formula

When the serial port 2 selects the working mode as mode 0, the serial communication interface works in the synchronous shift register mode. When the communication speed setting bit S2M0x6 is 0, its baud rate is fixed at 12 frequency division of the system clock (SYSclk/12); when S2M0x6 is set When 1, the baud rate is fixed at the system clock frequency divided by 2 (SYSclk/2). RxD2 is the serial communication data port, TxD2 is the same Step shift pulse output pin, send and receive 8-bit data, low-order first.

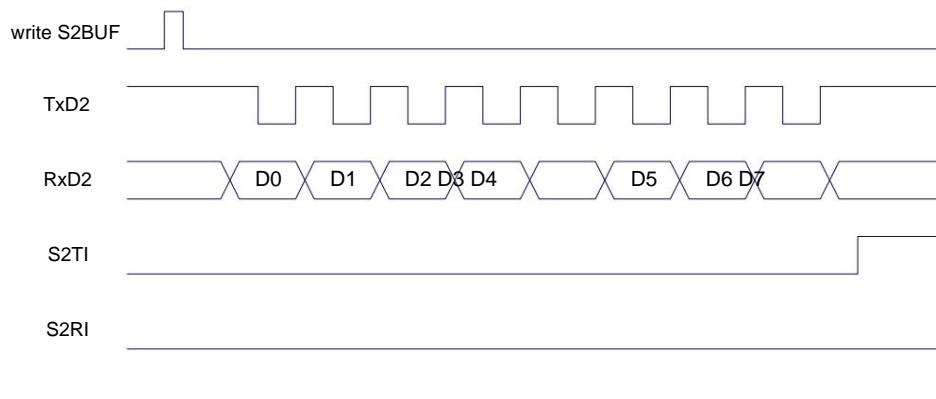
The transmission process of mode 0: when the host executes the command to write data into the transmission buffer S2BUF, the transmission is started, and the serial port is about 8 digits.

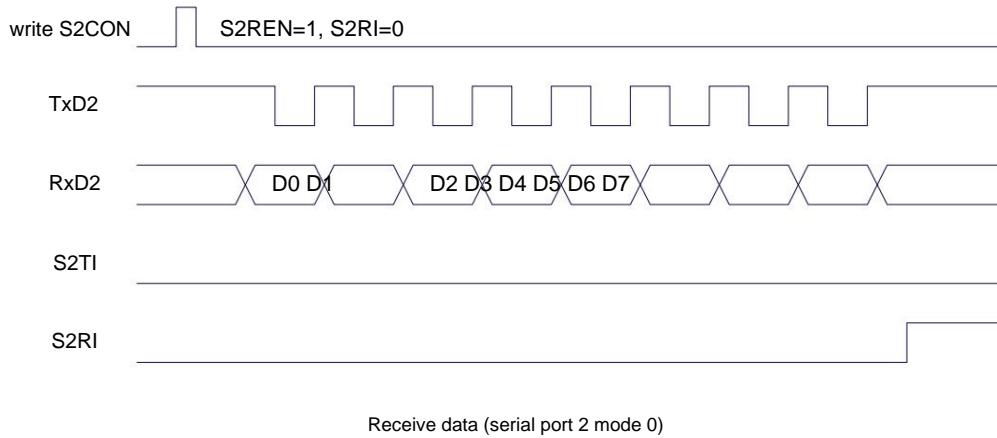
Output from RxD2 pin (from low

bit to high), the interrupt flag S2TI is set to 1 after sending, and the TxD2 pin outputs a synchronous shift pulse signal. When the write signal is valid, it is separated by one Clock, the sending control terminal SEND is valid (high level), allowing RxD2 to send data, and allowing TxD2 to output synchronous shift pulses. one frame (8 bits) When the data transmission is completed, each control terminal returns to the original state, and only S2TI keeps the high level, which is in the state of interrupt application. sending again Before data, S2TI must be cleared to 0 by software.

The receiving process of mode 0: first clear the receiving interrupt request flag S2RI and set the enable receiving control bit S2REN to start the mode 0 Receive process. After starting the receiving process, RxD2 is the serial data input terminal, and TxD2 is the synchronization pulse output terminal. Baud rate for serial reception Either SYSclk/12 or SYSclk/2 (divide-by-12 or divide-by-2 determined by S2M0x6). After receiving a frame of data (8 bits),

The control signal is reset, the interrupt flag S2RI is set to 1, and it is in the state of interrupt application. When receiving again, S2RI must be cleared to 0 by software





When working in mode 0, the multi-machine communication control bit S2SM2 must be cleared to 0 so that it does not affect the S2TB8 and S2RB8 bits. Thanks Porter  
The rate is fixed as SYScclk/12 or SYScclk/2, no timer is needed, and the clock of the microcontroller is directly used as the synchronous shift pulse.

The baud rate calculation formula of serial port 2 mode 0 is shown in the following table (SYScclk is the system operating frequency):

S2M0x6	Baud rate calculation formula
0	baud rate = $\frac{\text{SYScclk}}{12}$
1	baud rate = $\frac{\text{SYScclk}}{2}$

#### 15.4.5 Serial port 2 mode 1, mode 1 baud rate calculation formula

When software sets S2SM0 and S2SM1 of S2CON to "01", serial port 2 works in mode 1. This mode is 8-bit UART format, and one frame of information is 10 bits: 1 start bit, 8 data bits (low-order first) and 1 stop bit. The baud rate is variable, you can set the baud rate according to your needs. TxD2 is the data transmitting port, RxD2 is the data receiving port, and the serial port is full-duplex receiving/transmitting.

Transmission process in mode 1: When transmitting in serial communication mode, the data is output by the serial transmitter TxD2. When the host executes an instruction to write S2BUF, it starts the transmission of serial communication, and the write "S2BUF" signal also loads "1" into the ninth bit of the transmit shift register, and informs the TX control unit to start transmission. The shift register continuously shifts the data to the right to the TxD port for transmission, and continuously shifts in "0" on the left side of the data for supplementation. When the highest shift of the data reaches the output position of the shift register, the ninth bit is followed by "1", and all bits on the left side of it are "0". This state condition causes the TX control unit to make the last shift. Output, and then invalidate the allow-to-send signal "SEND" to complete the transmission of a frame of information, and set the interrupt request bit S2TI, that is, S2TI=1, to request interrupt processing from the host.

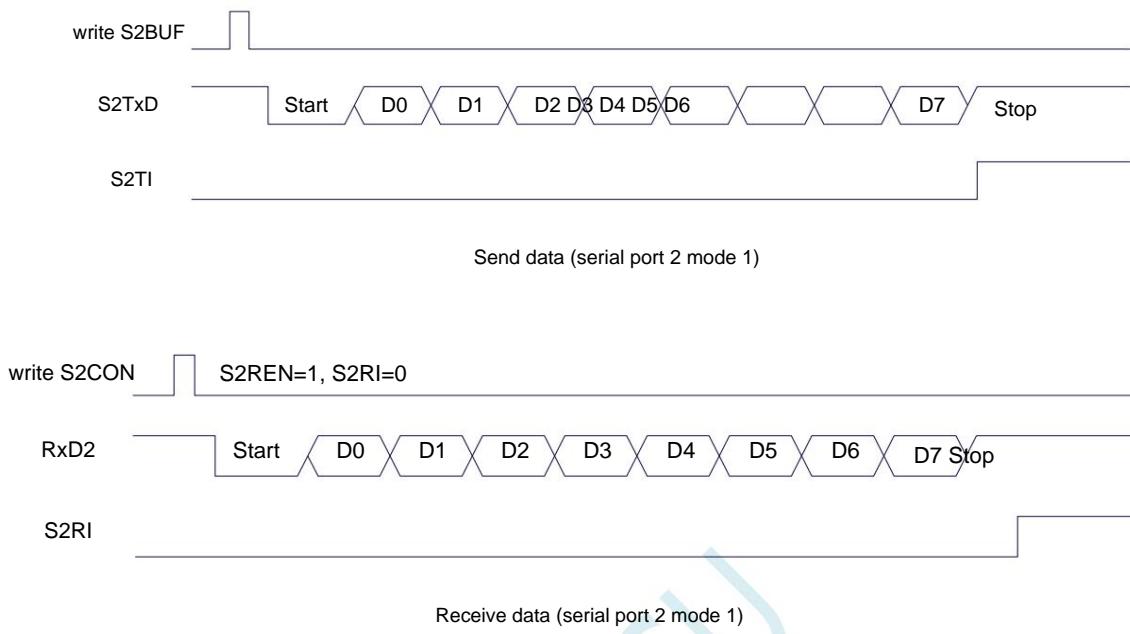
The receiving process of mode 1: when the software sets the receiving enable flag bit S2REN, that is, when S2REN=1, the receiver detects the signal of the RxD2 port. When it is detected that the RxD2 port sends a falling edge from "1" to "0" When the transition occurs, the receiver is started to prepare to receive data, and the receive counter of the baud rate generator is reset immediately, and 1FFH is loaded 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. 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 a shift. frame reception. If both of the following conditions are met:

-S2RI=0;

-S2SM2=0 or the received stop bit is 1. Then the received data

is valid, and it is loaded into S2BUF, the stop bit is entered into S2RB8, the S2RI flag is set to 1, and the request is made to the host.

If the above two conditions cannot be met at the same time, the received data will be invalid and lost. Regardless of whether the conditions are met or not, the receiver will check the RxD2 again. The transition of "1"~"0" on the port continues to receive the next frame. The reception is valid. After the interrupt is responded, the S2RI flag must be cleared by software. Normally, when serial communication works in mode 1, S2SM2 is set to "0".



The baud rate of serial port 2 is variable, and its baud rate is fixed by timer 2. When the timer adopts 1T mode (12 times speed), the corresponding baud rate is also 12 times faster.

The baud rate calculation formula of serial port 2 mode 1 is shown in the following table: (SYSclk is the system operating frequency)

select timing device	timer speed Spend	Reload value calculation formula	baud rate
timer 2	1T	timer 2 reload value = 65536 - $\frac{SYSclk}{4 \times \text{baud rate}}$	baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$
	12T	timer 2 reload value = 65536 - $\frac{SYSclk}{12 \times 4 \times \text{baud rate}}$	baud rate = $\frac{SYSclk}{12 \times 4 \times (65536 - \text{number of timer reloads})}$

The following is the reload value of the timer corresponding to the common frequency and common baud rate

frequency (MHz)	baud rate	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 port 2 mode 2, mode 2 baud rate calculation formula

When the two bits of S2SM0 and S2SM1 are 10, the serial port 2 works in mode 2. Serial port 2 working mode 2 is 9-bit data asynchronous communication

In UART mode, the information of one frame consists of 11 bits: 1 start bit, 8 data bits (low order first), 1 programmable bit (the first

9 bits of data) and 1 stop bit. The programmable bit (the 9th data bit) during transmission is provided by S2TB8 in S2CON and can be set to 1 by software

Or 0, or the odd/even parity bit P value in PSW can be loaded into S2TB8 (S2TB8 can be used as the address data flag bit in multi-machine communication,

It can also be used as the parity bit of the data). The ninth bit of data is loaded into S2RB8 of S2CON when received. TxD2 is the transmit port, RxD2 is

Receive port, receive/transmit in full duplex mode.

The baud rate of mode 2 is fixed to the system clock divided by 64 or 32 (depending on the value of S2MOD in S2CFG)

The baud rate calculation formula of serial port 2 mode 2 is shown in the following table (SYSclk is the system operating frequency):

SMOD baud rate calculation formula
------------------------------------

0	$\text{baud rate} = \frac{\text{SYSclk}}{64}$
1	$\text{baud rate} = \frac{\text{SYSclk}}{32}$

Compared with mode 1, mode 2 is slightly different except for the baud rate generation source, which is provided by S2TB8 to the ninth data bit of the shift register during transmission.

Except for the same, other functional structures are basically the same, and the receiving/transmitting operation process and timing are basically the same.

When the receiver receives a frame of information, the following conditions must be met at the same time:

-S2RI=0

-S2SM2=0 or S2SM2=1 and the ninth received data bit S2RB8=1.

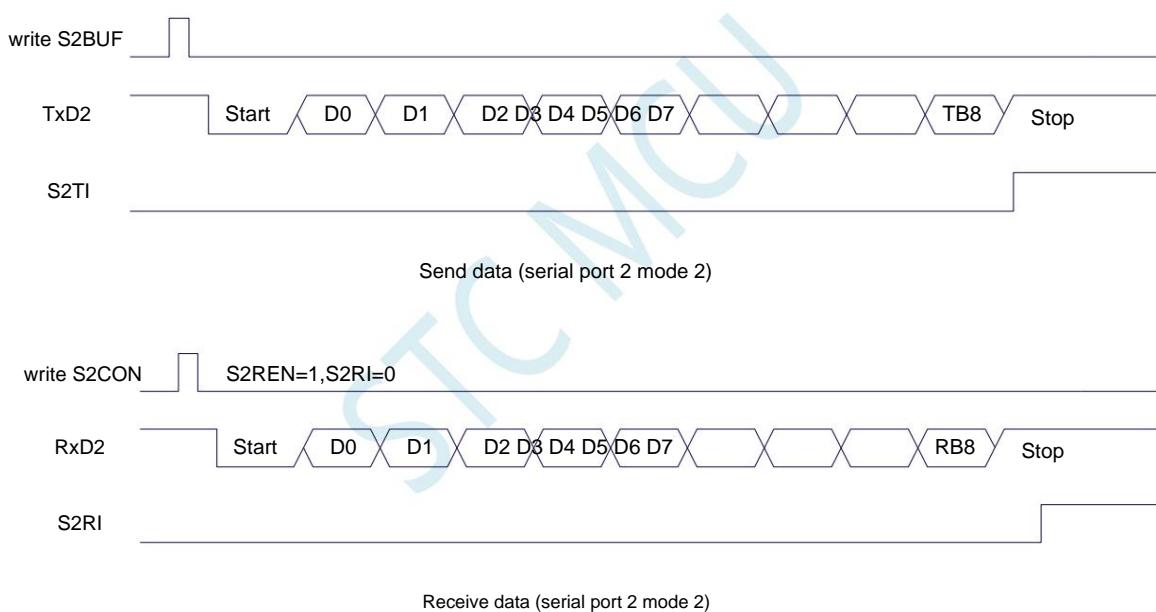
When the above two conditions are satisfied at the same time, the received data of the shift register is loaded into S2BUF and S2RB8, and the S2RI flag bit is

Set to 1 and requests interrupt handling from the host. If one of the above conditions is not satisfied, the data just received in the shift register is invalid and

Lost and S2RI is not asserted. Regardless of whether the above conditions are met or not, the receiver starts to detect the transition information of the RxD2 input port again.

Receive the input information of the next frame. In Mode 2, the received stop bits are independent of S2BUF, S2RB8 and S2RI.

The setting of S2SM2 and S2TB8 in S2CON and the agreement of communication protocol by the software provide convenience for multi-computer communication.



### 15.4.7 Serial port 2 mode 3, mode 3 baud rate calculation formula

When the two bits of S2SM0 and S2SM1 are 11, the serial port 2 works in mode 3. Serial communication mode 3 is 9-bit data asynchronous communication UART mode, the information of one frame consists of 11 bits: 1 start bit, 8 data bits (low order first), 1 programmable bit (9th bit data) and 1 stop bit. Programmable bit (9th data bit) when transmitting is provided by S2TB8 in S2CON and can be set to 1 or 0 by software, or The user can load the odd/even parity bit P value in the PSW into S2TB8 (S2TB8 can be used as the address data flag bit in multi-machine communication, and can also be used as the parity bit of the data). The ninth bit of data is loaded into S2RB8 of S2CON when received. TxD2 is the sending port, RxD2 is the receiving end port, receive/transmit in full duplex mode.

Compared with mode 1, except for the 9th data bit provided to the shift register by S2TB8 during transmission, the rest of the functional structure is based on This is the same, and its receiving and sending operation process and timing are basically the same.

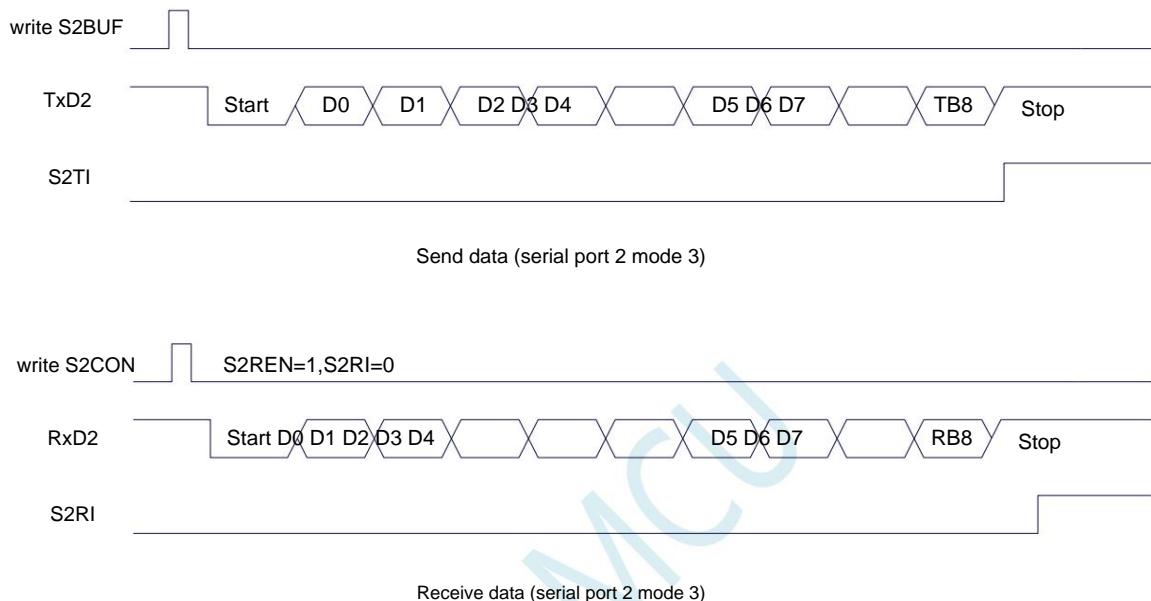
When the receiver receives a frame of information, the following conditions must be met at the same time:

-S2RI=0

-S2SM2=0 or S2SM2=1 and the ninth received data bit S2RB8=1.

When the above two conditions are satisfied at the same time, the received data of the shift register is loaded into S2BUF and S2RB8, and the S2RI flag bit is Set to 1 and requests interrupt handling from the host. If one of the above conditions is not satisfied, the data just received in the shift register is invalid and Lost and S2RI is not asserted. Regardless of whether the above conditions are met or not, the receiver starts to detect the transition information of the RxD2 input port again. Receive the input information of the next frame. In Mode 3, the received stop bits are independent of S2BUF, S2RB8 and S2RI.

The setting of S2SM2 and S2TB8 in S2CON and the stipulation of the communication protocol by the software provide convenience for multi-machine communication.



The baud rate calculation formula of serial port 2 mode 3 is exactly the same as that of mode 1. Please refer to the baud rate calculation formula of Mode 1.

#### 15.4.8 Serial port 2 automatic address recognition

#### 15.4.9 Serial port 2 slave address control registers (S2ADDR, S2ADEN)

Symbol 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 the field of multi-machine communication. The main principle is that the slave system uses the hardware comparison function to identify the Based on the address information in the data stream of the host serial port 2, the slave address of the local machine set by the registers S2ADDR and S2ADEN, the hardware automatically Automatically filter the slave address, when the slave address information from the host matches the slave address set by the local machine, the hardware generates a string of Port 2 is interrupted; otherwise, the hardware automatically discards the serial port 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, even if the slave is in The normal working state can also avoid the continuous interruption of serial port 2 and reduce the system execution efficiency.

To use the automatic address recognition function of serial port 2, firstly, you need to set the serial port 2 communication mode of the MCU involved in communication to mode 1, And turn on the S2SM2 bit of the S2CON of the slave. For serial port 2 mode 1 of 9 data bits, the 9th data (stored in S2RB8 ) is the address/data flag bit, when the ninth bit of data is 1, it means that the previous 8-bit data (stored in S2BUF) is the address signal.

interest. When S2SM2 is set to 1, the slave MCU will automatically filter out non-address data (data whose ninth bit is 0), while the S2BUF The address data (data whose 9th bit is 1) is automatically compared with the local address set by S2ADDR and S2ADEN, if the address If they match, S2RI will be set to "1" and an interrupt will be generated, otherwise the serial port 2 data received this time will not be processed.

The setting of the slave address is set through the two registers S2ADDR and S2ADEN. S2ADDR is the slave address register, It stores the slave address of the machine. S2ADEN is the slave address mask bit register, which is used to set the ignore bit in the address information.

The law is as follows:

E.g

S2ADDR = 11001010

S2ADEN = 10000001

Then the matching address is 1xxxxxx0

That is, as long as bit0 in the address data sent by the host is 0 and bit7 is 1, it can match the local address

Another example

S2ADDR = 11001010

S2ADEN = 00001111

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, and the upper 4 bits are ignored, and you can to any value.

The master can use the broadcast address (FFH) to select all slaves at the same time for communication.

#### 15.4.10 Serial Port 2 Synchronous Mode Control Register 1 (USART2CR1)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
USART2CR1 7	EFDC8H	LINEN	DORD	CLKEN	SPMOD	SPIEN			SPSLV CPOL CPHA

LINEN: LIN mode enable bit

0: Disable LIN mode

1: Enable LIN mode

DORD: The order in which data bits are sent/received in SPI mode

0: Send/receive the high-order bit (MSB) of the data first

1: Send/receive the low-order bit (LSB) of the data first

CLKEN: SmartCard mode clock output control bit

0: Disable clock output

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 in master mode

1: SPI is in slave mode

CPOL: SPI Clock Polarity Control

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

1: High level when SCLK is idle, the front clock edge of SCLK is a falling edge, and the rear clock edge is a rising edge

#### CPHA: SPI Clock Phase Control

0: The data SS pin drives the first bit of data at a low level and changes the data on the trailing edge of SCLK, and samples the data on the leading edge of the clock

1: Data is driven on the leading edge of SCLK and sampled on the trailing edge

### 15.4.11 Serial Port 2 Synchronous Mode Control Register 2 (USART2CR2)

Symbol 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 the output of NACK signal

1: Enable output NACK signal

HDSEL: single-wire half-duplex mode enable bit

0: Disable single-line half-duplex mode

1: Enable single-wire half-duplex mode

PCEN: Check Bit Control Enable

0: Disable parity bit control function

1: Enable parity bit control function

PS: check digit mode selection

0: even test

1: odd parity

PE: Check bit error flag (must be cleared by software)

0: No check error

1: There is a verification error

### 15.4.12 Serial Port 2 Synchronous Mode Control Register 3 (USART2CR3)

Symbol 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 Synchronous Mode Control Register 4 (USART2CR4)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
USART2CR4 7EF	DCBH	-	-	-	-	SCCKS[1:0]		SPICKS[1:0]	

SCCKS[1:0]: SmartCard mode clock selection

SCCKS[1:0]	SmartCard mode clock
00	SYScclk/64
01	SYScclk/32
10	SYScclk/16
11	SYScclk/8

SPICKS[1:0]: SPI mode clock selection

SPICKS[1:0]	SPI mode clock
00	SYScclk/4
01	SYScclk/8
10	SYScclk/16
11	SYScclk/2

### 15.4.14 Serial Port 2 Synchronous Mode Control Register 5 (USART2CR5)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
USART2CR5 7EF	DCCH	BRKDET	HDRER	SLVEN	ASYNC	TXCF	SENDBK	HDRDET	SYNC

BRKDET: LIN slave mode interval field (BREAK) detection flag

0: No LIN gap field detected

1: LIN interval field detected, need to be cleared by software

HDRER: LIN slave mode header (HEADER) error

0: No LIN header error detected

1: LIN message header error detected, need to be cleared by software

SLVEN: LIN Slave Mode Enable Bit

0: LIN is master mode

1: LIN is in slave mode

ASYNC: LIN automatic synchronization function enable bit

0: Disable automatic synchronization

1: Enable automatic synchronization

TXCF: Transmission collision flag. Single-wire half-duplex mode, LIN mode and SmartCard mode are valid

0: No transmission collision detected (sent data same as received data)

1: Transmission collision detected (data sent is different from data received)

SENDBK: Send interval field. The software writes 1 to trigger the transmission interval field, and the hardware automatically clears it to 0 after the transmission is completed.

HDRDET: LIN slave mode header (HEADER) detection flag

0: LIN header not detected

1: LIN header is detected and needs to be cleared by software

SYNC: LIN slave mode sync field detection flag.

After correct analysis of the sync field, the hardware sets the SYNC flag to 1. SYNC is automatically cleared by hardware when receiving a marker field

### 15.4.15 Serial 2 Synchronous Mode Guard Time Register (USART2GTR)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
USART2GTR7EFDCDH								

The USARTGTR register is only valid in SmartCard mode. This register gives the guard time value according to the number of baud rate clocks. S2TI logo

It is set to 1 by hardware when the data bit sent by the SmartCard is equal to the guard 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)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
USART2BRH 7EFDCEH					USART2BR[15:8]			
USART2BRL 7EFDCFH					USART2BR[7:0]			

Baud rate calculation formula in LIN mode, IrDA mode and SmartCard mode

$$\text{Sync Baud Rate} = \frac{\text{SYSclk}}{16^* \text{ USART2BR}[15:0]}$$

## 15.5 Example Program

### 15.5.1 Serial port 1 uses timer 2 as a baud rate generator

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT       (65536-FOSC/115200/4)

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 XFR
    CKCON = 0x00;         //Set the external data bus speed to the fastest
    WTST = 0x00;          //Set the program code to wait for parameters,
                           //assign to CPU The speed of executing the program is set to the fastest

    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 a baud rate generator

---

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL
#define BRT      (65536-FOSC/115200/4)                //Define as an unsigned long integer to avoid calculation overflow

bit      busy;
char     wptr;
char     rptr;

```

```

char      buffer[16];

void UartIsr() interrupt
{
    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      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                               //Set the program code to wait for parameters,
                                                //assign to      CPU   The speed of executing the program is set to the fastest

    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 a baud rate generator

---

//The test frequency is 11.0592MHz

```

#ifndef "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT      (256-FOSC/115200/32)

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      XFR
    CKCON = 0x00;                            //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                                //assign to      CPU The speed of executing the program is set to the fastest

    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 port 2 uses timer 2 as a baud rate generator

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     //For header files, see Download Software
#include "intrins.h"

#define FOSC          11059200UL                         //Define as an unsigned long integer to avoid calculation overflow
#define BRT           (65536-FOSC/115200/4)

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 = 0x10;
    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 XFR
    CKCON = 0x00;   // Set the external data bus speed to the fastest
    WTST = 0x00;    // Set the program code to wait for parameters,
                    // assign to CPU The speed of executing the program is set to the fastest

    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;
        }
    }
}
```

## 16 Asynchronous serial communication (UART3, UART4)

The STC32G series microcontrollers have 2 full-duplex asynchronous serial communication interfaces (UART3 and UART4). Each serial port consists of 2 It consists of data buffer, a shift register, a serial control register and a baud rate generator. Data buffer for each serial port

The buffer is composed 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, and the baud rates of these two modes are variable. User can Use software to set different baud rates and choose different working modes. The host can program receive/transmit by polling or interrupt It is very flexible to use.

The communication ports of serial port 3 and serial port 4 can be switched to multiple groups of ports through the switching function of the function pins, so that one communication port can be switched.

Time-division multiplexing into multiple communication ports.

### 16.1 Serial port function pin switch

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

S4\_S: Serial port 4 function pin selection bit

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

S3\_S: Serial port 3 function pin selection bit

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

### 16.2 Serial port related registers

symbol	describe	address	Bit addresses and symbols								reset value	
			B7	B6	B5	B4	B3	B2	B1	B0		
S3CON	Serial port 3 control register	A8H S3SM0	S3ST3	S3SM2 S3REN S3TB8 S3RB8 S3T							S3RI 0000,0000	
S3BUF	Serial port 3 data register	A9H										0000,0000
S4CON	Serial port 4 control register	F8H S4SM0	S4ST4	S4SM2 S4REN S4TB8 S4RB8 S4T							S4RI 0000,0000	
S4BUF	Serial port 4 data register	F9H										0000,0000

## 16.3 Serial port 3 (asynchronous serial port **UART3**)

### 16.3.1 Serial port 3 control register (**S3CON**)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
S3CON	ACH S3SM0		S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0: Specify the communication working mode of serial port 3, as shown in the following table:

S3SM0	serial port 3 working mode	Function Description
0	Mode 0 Variable	baud rate 8-bit data mode
1	Mode 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 of serial port 3

1: select timer 3 as the baud rate generator of serial port 3

S3SM2: Allow 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 screening state. At this point, the received 9th bit (ie S3RB8) can be used to filter the address frame:

If S3RB8=1, it means that the frame is an address frame, the address information can enter S3BUF, and make S3RI 1, and then in the interrupt service

Then compare the address numbers in the program; if S3RB8=0, it means that the frame is not an address frame, it should be discarded and S3RI=0 should be kept. in mode 1

, if the S3SM2 bit is 0 and the S3REN bit is 1, the receiver is in the address frame filtering disabled state. Regardless of received S3RB8

If it is 0 or 1, the received information can be entered into S3BUF, and S3RI=1. At this time, S3RB8 is usually the parity bit. Mode 0

For non-multi-machine communication mode, in this mode, S3SM2 should be set to 0.

S3REN: enable/disable serial port receive control bit

0: Forbid the serial port to receive data

1: Allow the serial port to receive data

S3TB8: When serial port 3 uses mode 1, S3TB8 is the ninth bit of data to be sent, generally used as a check bit or address frame/data frame mark

Bit, 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 ninth bit of data received, generally used as a check digit or address frame/data frame mark

Chi bit. In Mode 0, this bit is not used.

S3TI: Serial port 3 sends interrupt request flag. When the stop bit starts to transmit, the hardware will automatically set S3TI to 1, and send a request to the CPU to interrupt,

S3TI must be cleared in software after responding to an interrupt.

S3RI: Serial port 3 receive interrupt request flag. The hardware automatically sets S3RI to 1 at the middle moment of serial reception of the stop bit, and sends a message to the CPU.

Interrupt request, S3RI must be cleared by software after responding to the interrupt.

### 16.3.2 Serial port 3 data register (**S3BUF**)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

S3BUF: Serial port 1 data receive/transmit buffer. S3BUF is actually 2 buffers, read buffer and write buffer, the two operations are divided into

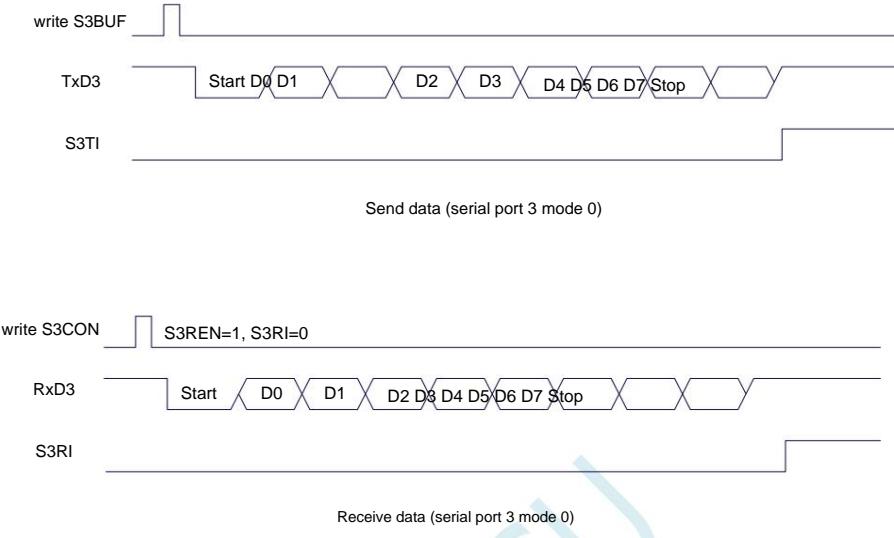
Do not correspond to two different registers, one is a write-only register (write buffer), and the other is a read-only register (read buffer). right

S3BUF read operation is actually to read the serial port receive buffer, and write operation to S3BUF is to trigger 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 variable baud rate UART operating mode. In this mode, one frame of information is 10 bits: 1 start bit, 8 data bits (low order first) and 1 stop bit.

The baud rate is variable, and the baud rate can be set as required. TxD3 is the data transmitting port, RxD3 is the data receiving port, and the serial port is full-duplex receiving/transmitting.



The baud rate of serial port 3 is variable, and its baud rate can be generated by timer 2 or timer 3. When the timer is in 1T mode (12 double speed), the corresponding baud rate speed will be increased by 12 times accordingly.

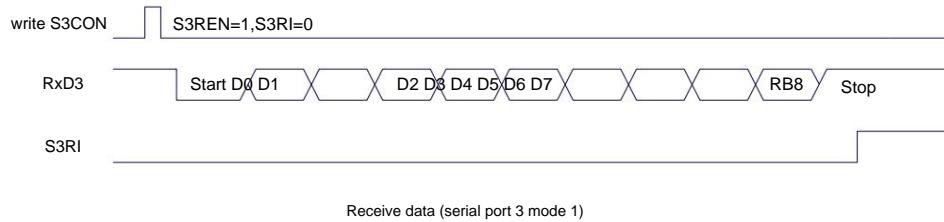
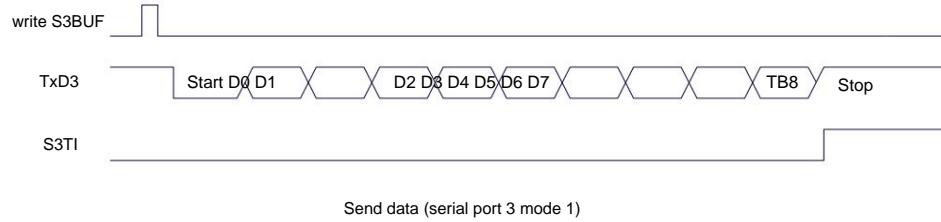
The baud rate calculation formula of serial port 3 mode 0 is shown in the following table: (SYSclk is the system operating frequency)

select timing device	timer speed Spend	Reload value calculation formula	baud rate
timer 2	1T	timer 2 reload value = $65536 - \frac{SYSclk}{4 \times \text{baud rate}}$	baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$
	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 = $4 \times \frac{SYSclk}{(65536 - \text{number of timer reloads})}$
	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 variable baud rate UART operating mode. In this mode, one frame of information is 11 bits: 1 start bit, 9 data bits (low order first) and 1 stop bit.

The baud rate is variable, and the baud rate can be set as required. TxD3 is the data transmitting port, RxD3 is the data receiving port, and the serial port is full-duplex receiving/transmitting.



The baud rate calculation formula of serial port 3 mode 1 is exactly the same as that of mode 0. Please refer to the baud rate calculation formula of mode 0.

## 16.4 Serial port 4 (asynchronous serial port **UART4**)

### 16.4.1 Serial port 4 control register (**S4CON**)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
S4CON	FDH	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0: Specify the communication working mode of serial port 4, as shown in the following table:

S4SM0	serial port 4 working mode	Function Description
0	Mode 0 Variable	baud rate 8-bit data mode
1	Mode 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 of serial port 4

1: select timer 4 as the baud rate generator of serial port 4

S4SM2: Allow 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 point, the received 9th bit (ie S4RB8) can be used to filter the address frame:

If S4RB8=1, it means that the frame is an address frame, the address information can enter S4BUF, and make S4RI 1, and then in the interrupt service

Then compare the address numbers in the program; if S4RB8=0, it means that the frame is not an address frame, it should be discarded and S4RI=0 should be kept. in mode 1

, if the S4SM2 bit is 0 and the S4REN bit is 1, the receiver is in the address frame filtering disabled state. Regardless of received S4RB8

If it is 0 or 1, the received information can be entered into S4BUF, and S4RI=1. At this time, S4RB8 is usually the check digit. Mode 0

For non-multi-machine communication mode, in this mode, S4SM2 should be set to 0.

S4REN: enable/disable serial port receive control bit

0: Forbid the serial port to receive data

1: Allow the serial port to receive data

S4TB8: When serial port 4 uses mode 1, S4TB8 is the ninth bit of data to be sent, generally used as a parity bit or address frame/data frame mark

Bit, set or cleared by software as needed. In Mode 0, this bit is not used.

S4RB8: When serial port 4 uses mode 1, S4RB8 is the ninth bit of data received, generally used as a check bit or address frame/data frame mark

Chi bit. In Mode 0, this bit is not used.

S4TI: Serial port 4 sends interrupt request flag. When the stop bit starts to transmit, the hardware will automatically set S4TI to 1, and send a request to the CPU to interrupt,

S4TI must be cleared in software after responding to an interrupt.

S4RI: Serial port 4 receive interrupt request flag. The hardware automatically sets S4RI to 1 at the middle moment of serial reception of the stop bit, and sends a message to the CPU.

Interrupt request, S4RI must be cleared by software after responding to the interrupt.

### 16.4.2 Serial port 4 data register (**S4BUF**)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
S4BUF	FEH								

S4BUF: Serial port 1 data receive/transmit buffer. S4BUF is actually two buffers, read buffer and write buffer, the two operations are divided into

Do not correspond to two different registers, one is a write-only register (write buffer), and the other is a read-only register (read buffer). right

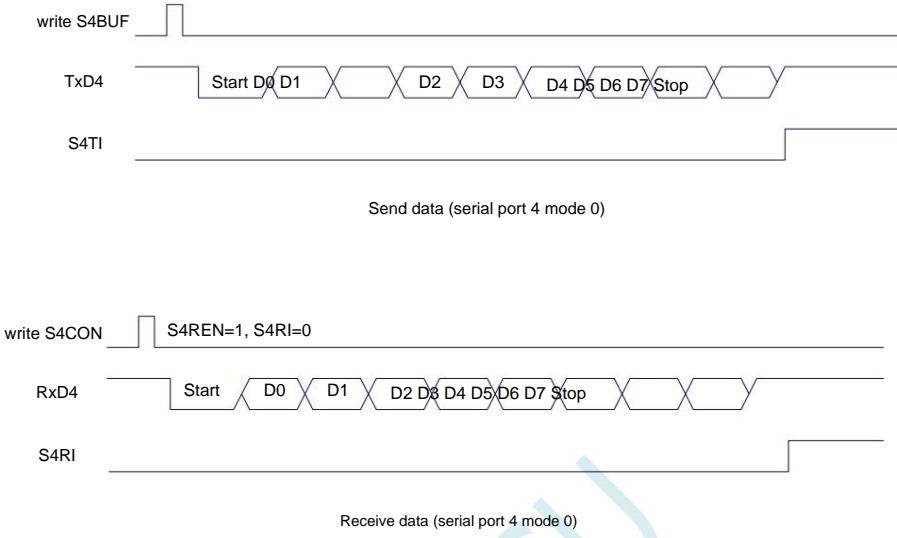
S4BUF reads, actually reads the serial port receive buffer, and writes to S4BUF triggers the serial port to start sending

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 variable baud rate UART operating mode. In this mode, one frame of information is 10 bits: 1 start bit, 8 data bits (low order first) and 1 stop bit.

The baud rate is variable, and the baud rate can be set as required. TxD4 is the data transmitting port, RxD4 is the data receiving port, and the serial port is full-duplex receiving/transmitting.



The baud rate of serial port 4 is variable, and its baud rate can be generated by timer 2 or timer 4. When the timer is in 1T mode (12 double speed), the corresponding baud rate speed will be increased by 12 times accordingly.

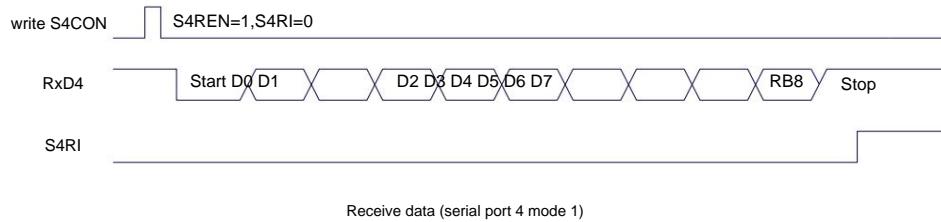
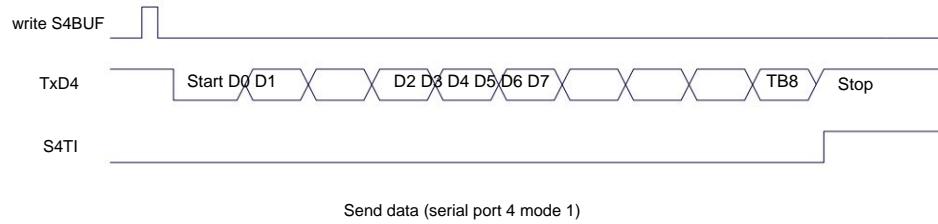
The baud rate calculation formula of serial port 4 mode 0 is shown in the following table: (SYSclk is the system operating frequency)

select timing device	timer speed Spend	Reload value calculation formula	baud rate
timer 2	1T	timer 2 reload value = $65536 - \frac{SYSclk}{4 \times \text{baud rate}}$	baud rate = $\frac{SYSclk}{4 \times (65536 - \text{number of timer reloads})}$
	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 = $4 \times \frac{SYSclk}{(65536 - \text{number of timer reloads})}$
	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 variable baud rate UART operating mode. In this mode, one frame of information is 11 bits: 1 start bit, 9 data bits (low order first) and 1 stop bit.

The baud rate is variable, and the baud rate can be set as required. TxD4 is the data transmitting port, RxD4 is the data receiving port, and the serial port is full-duplex receiving/transmitting.

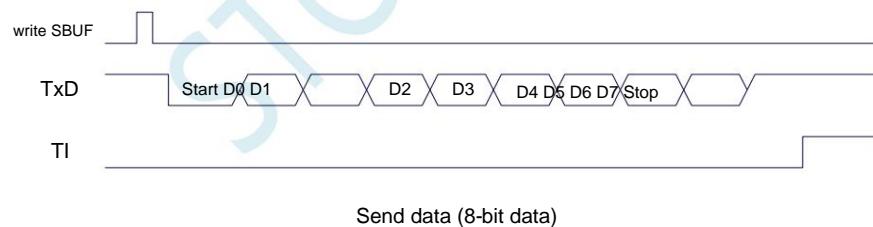


The baud rate calculation formula of serial port 4 mode 1 is exactly the same as that of mode 0. Please refer to the baud rate calculation formula of mode 0.

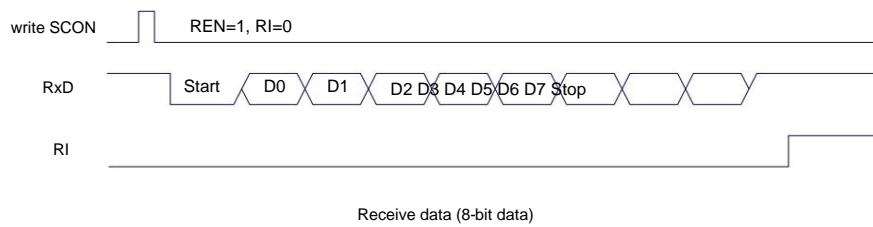
## 16.5 Notes on Serial Port

Regarding the serial port interrupt request, there are the following issues that need to be noted: (Serial port 1, serial port 2, serial port 3, and serial port 4 are all similar, the following is the serial port 1 as an example)

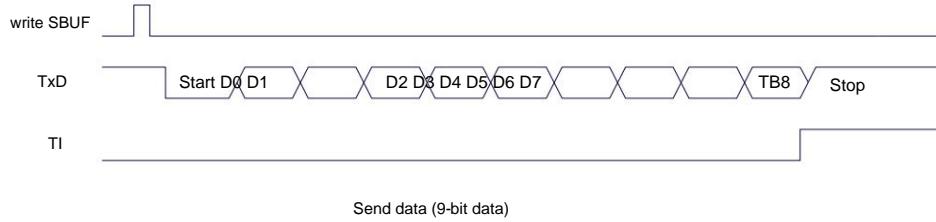
In 8-bit data mode, a TI interrupt request is generated after the entire stop bit is transmitted, as shown in the following figure:



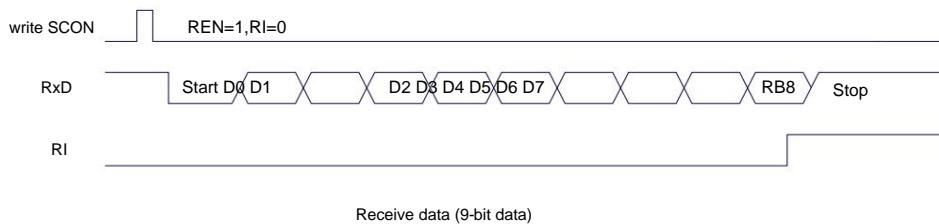
In 8-bit data mode, an RI interrupt request is generated after receiving half of the stop bits, as shown in the following figure:



In 9-bit data mode, a TI interrupt request is generated after the entire 9th data bit is transmitted, as shown in the following figure:



In 9-bit data mode, an RI interrupt request is generated after half of the 9th data bit is received, as shown in the following figure:



## 16.6 Example Program

### 16.6.1 Serial port 3 uses timer 2 as a baud rate generator

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT       (65536-FOSC/115200/4)

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 XFR
    CKCON = 0x00;
    WTST = 0x00;
    POM0 = 0x00;
    POM1 = 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 uses timer 3 as a baud rate generator

---

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

#define FOSC          11059200UL
#define BRT           (65536-FOSC/115200/4)           //Define as an unsigned long integer to avoid calculation overflow

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 XFR
    CKCON = 0x00;   //Set the external data bus speed to the fastest
    WTST = 0x00;    //Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    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 uses timer 2 as a baud rate generator

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT      (65536-FOSC/115200/4)

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 XFR
    CKCON = 0x00;
    WTST = 0x00;
    //Set the external data bus speed to the fastest
    //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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 uses timer 4 as baud rate generator

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

#define FOSC      11059200UL //Define as an unsigned long integer to avoid calculation overflow
#define BRT      (65536-FOSC/115200/4)

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 XFR
    CKCON = 0x00;
    WTST = 0x00;
    //Set the external data bus speed to the fastest
    //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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");

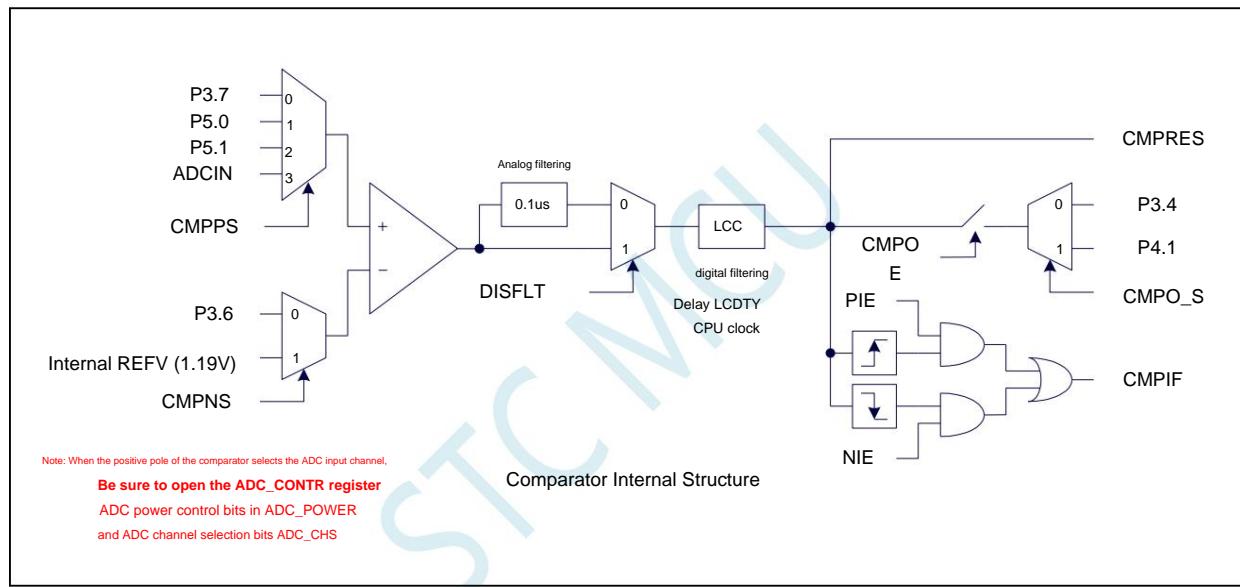
    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

## 17 comparators, power-down detection, internal fixed comparison voltage

A comparator is integrated inside the STC32G series microcontroller. The positive pole of the comparator can be P3.7 port, P5.0 port, P5.1 port or the analog input channel of the ADC, and the negative pole can be the P3.6 port or the REFV voltage after the internal BandGap passes through the OP (internal fixed comparison voltage). The application of multiple comparators can be realized through multiplexers and time-division multiplexing.

There are two levels of programmable filtering inside the comparator: analog filtering and digital filtering. Analog filtering can filter out the comparison input signal. The glitch signal, digital filtering can wait for the input signal to be more stable before comparing. The comparison result can be read directly by reading the internal register Bit acquisition, and the comparator result can also be output to an external port in the forward or reverse direction. Outputting comparison results to external ports can be used as Trigger signal and feedback signal, can expand the application scope of comparison.

### 17.1 Comparator Internal Structure Diagram



### 17.2 Comparator function pin switching

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
P_SW2	BAH EAXFR		=	I2C_S[1:0]	<b>CMPO_S</b>	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

symbol	describe	address	Bit addresses and symbols								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	7EFEEAH	CHYS[1:0]		-	-	-	- CMPNS CMPPPS[1:0] 00xx,x000			

### 17.3.1 Comparator Control Register 1 (CMPCR1)

Symbol address	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN CMPIF	PIE	NIE	-	-	CMPOE CMPRES	

CMPEN: Comparator Module Enable bit

0: Turn off the compare function

1: Enable compare function

CMPIF: Comparator interrupt flag. When PIE or NIE is enabled, if the corresponding interrupt signal is generated, the hardware will automatically set CMPIF to 1,

And make an interrupt request to the CPU. This flag must be cleared by user software.

PIE: Comparator rising edge interrupt enable bit.

0: Comparator rising edge interrupt is 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 is disabled.

1: Enable comparator falling edge interrupt. An interrupt request is generated when the comparison result of the enabled comparator changes from 1 to 0.

CMPOE: Comparator Result Output Control Bit

0: Disable comparator result output

1: Enable comparator result output. The comparator result is output to P3.4 or P4.1 (set by CMPO\_S in P\_SW2)

CMPRES: Comparison result of the comparator. This bit is read-only.

0: Indicates that the level of CMP+ is lower than the level of CMP-

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 of the comparator.

### 17.3.2 Comparator Control Register 2 (CMPCR2)

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

INVCMPO: Comparator Result Output Control

0: The comparator result is output in the positive direction. If CMPRES is 0, P3.4/P4.1 output low level, otherwise, output high level.

1: Invert the output of the comparator result. If CMPRES is 0, P3.4/P4.1 output high level, otherwise, output low level.

DISPLT: Analog Filter Function Control

0: Enable 0.1us analog filter function

1: Disable the 0.1us analog filter function, which can slightly increase the comparison speed of the comparator.

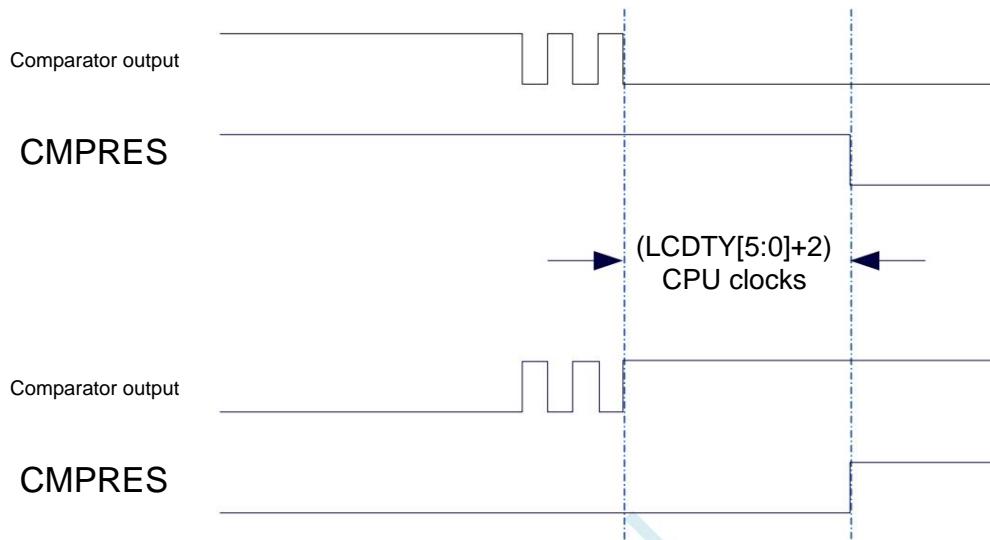
LCDTY[5:0]: Digital filter function control

The digital filtering function is the digital signal debounce function. When a rising or falling edge of the comparison result changes, the comparator detects a change

The converted signal must keep the number of CPU clocks set by LCDTY unchanged before the data change is considered valid; no

It will be treated as if there is no change in the signal.

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



### 17.3.3 Comparator Extended Configuration Register (CMPEXCFG)

Symbol address	B7		B6	B5	B4 B3		B2	B1	B0
CMPEXCFG 7EFEEAH		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 voltage REFV of the internal BandGap after passing through OP as the negative input source of the comparator (when the chip is shipped from the factory, the internal reference

Voltage adjusted to 1.19V)

CMPPS[1:0]: Comparator positive input selection

CMPPS[1:0] Comparator positive terminal	
00	P3.7
01	P5.0
10	P5.1
11	ADCIN

(Note 1: When the positive pole of the comparator selects the ADC input channel, please be sure to turn on the ADC power in the ADC\_CONTR register.

Source control bit **ADC\_POWER** and ADC channel selection bit **ADC\_CHS**)

## 17.4 Example Program

### 17.4.1 Use of comparator (interrupt method)

---

```
//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void CMP_Isr() interrupt 21 {
    CMPIF = 0; //clear interrupt flag
    if (CMPRES)
    {
        P10 = !P10; //Falling edge interrupt test port
    }
    else
    {
        P11 = !P11; //Rising edge interrupt test port
    }
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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; // Comparators DC Hysteresis input selection
    // 0:0mV; 0x40:10mV; 0x80:20mV; 0xc0:30mV

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

    CMPCR1 = 0x00;
}
```

```

CMPCR2=0x00; // Comparator positive output
INVCMPO = 0; // Comparator Inverted Output
// INVCMPO = 1; // Enable 0x1MS
DISFLT = 0; // Disable 0x1MS
// DISFLT = 1; // Comparator result is output directly
// CMPCR2 &= ~0x3f; // Comparator result via 16 output after debounced clocks
CMPCR2 |= 0x10; // Enable Comparator Edge Interrupt
PIE = NIE = 1; // Comparator rising edge interrupt disabled
// PIE = 0; // 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 Comparator (Query Mode)

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; // Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    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; // Comparators DC Hysteresis input selection
// 0:0mV; 0x40:10mV; 0x80:20mV; 0xc0:30mV

    CMPEXCFG &= ~0x03; // P3.7 As CMP+ input pin
// CMPEXCFG |= 0x01; // P5.0 for the CMP+ input pin

```

```

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

CMPCR1 = 0x00;                // Comparator positive output
CMPCR2= 0x00;                 // Comparator Inverted Output
INVCMPO = 0;                  // Enable A/DMS
// INVCMPO = 1;                // Disable A/DMS
DISFLT = 0;                   // Comparator result is output directly
// DISFLT = 1;                 // Comparator result via 16 output after debounced clocks
// CMPCR2 &= ~0x3f;           // Enable Comparator Edge Interrupt
CMPCR2 |= 0x10;               // Comparator rising edge interrupt disabled
PIE = NIE = 1;                // Enable Comparator Rising Edge Interrupt
// PIE = 0;                    // Disable Comparator Falling Edge Interrupt
// NIE = 0;                    // Enable Comparator Falling Edge Interrupt
// NIE = 1;                    // Disable comparator output
// CMPOE = 0;                  // enable comparator output
CMPOE = 1;                   // Enable Comparator Module
CMPEN = 1;

while (1)
{
    P10 = CMPRES;             // Read the comparator comparison result
}

```

### 17.4.3 Multiplexing Application of Comparator (Comparator + ADC Input Channel)

Since the positive pole of the comparator can select the analog input channel of the ADC, multiple ratios can be achieved through multiplexers and time-division multiplexing.

Comparator application.

Note: When the positive pole of the comparator selects the ADC input channel, be sure to turn on the ADC power control bit in the ADC\_CONTR register ADC\_POWER and ADC channel selection bits ADC\_CHS

```

//The test frequency is 11.0592MHz

##include "stc8h.h"
##include "stc32g.h"           // For header files, see Download Software
##include "intrins.h"

void main()
{
    EAXFR = 1;                // Enable XFR
    CKCON = 0x00;              // access to set external data bus speed to fastest
    WTST = 0x00;               // Set the program code to wait for parameters,
                                // assign to CPU The speed of executing the program is set to the fastest

    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;
P1M1 |= 0x01;
ADC_CONTR = 0x80;

CMPEXCFG = 0x00;
// CMPEXCFG &= ~0x03;
// CMPEXCFG |= 0x01;
// CMPEXCFG |= 0x02;
CMPEXCFG |= 0x03;
CMPEXCFG &= ~0x04;
// CMPEXCFG |= 0x04;

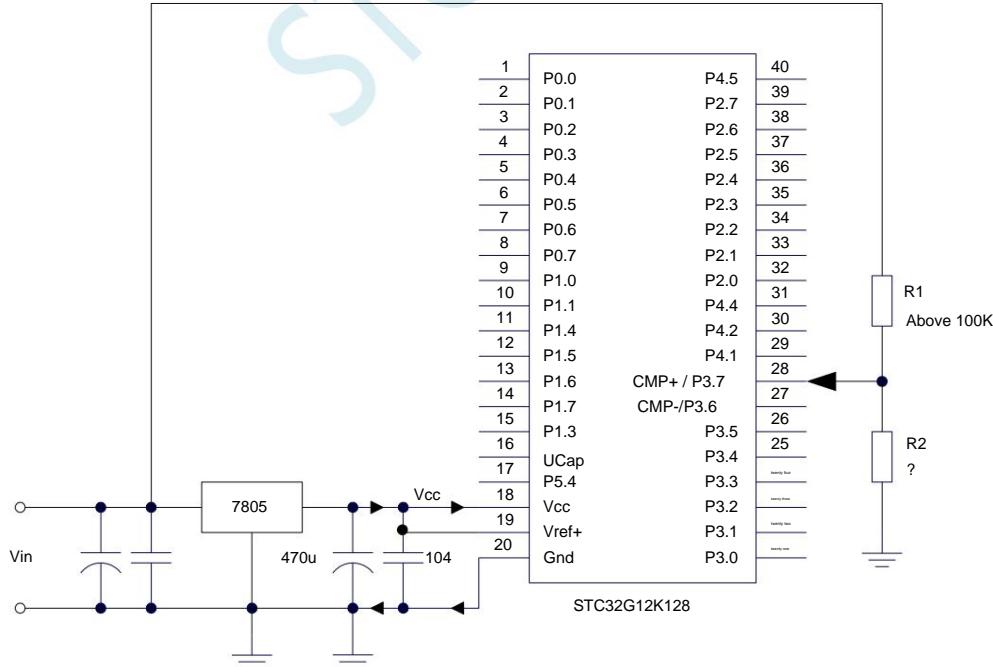
CMPCR2 = 0x00;
CMPCR1 = 0x00;
CMPOE = 1;
CM PEN = 1;

while (1);
}

```

17.4.4 The comparator is used for external power-down detection (the user data should be saved in time to

## EEPROM )

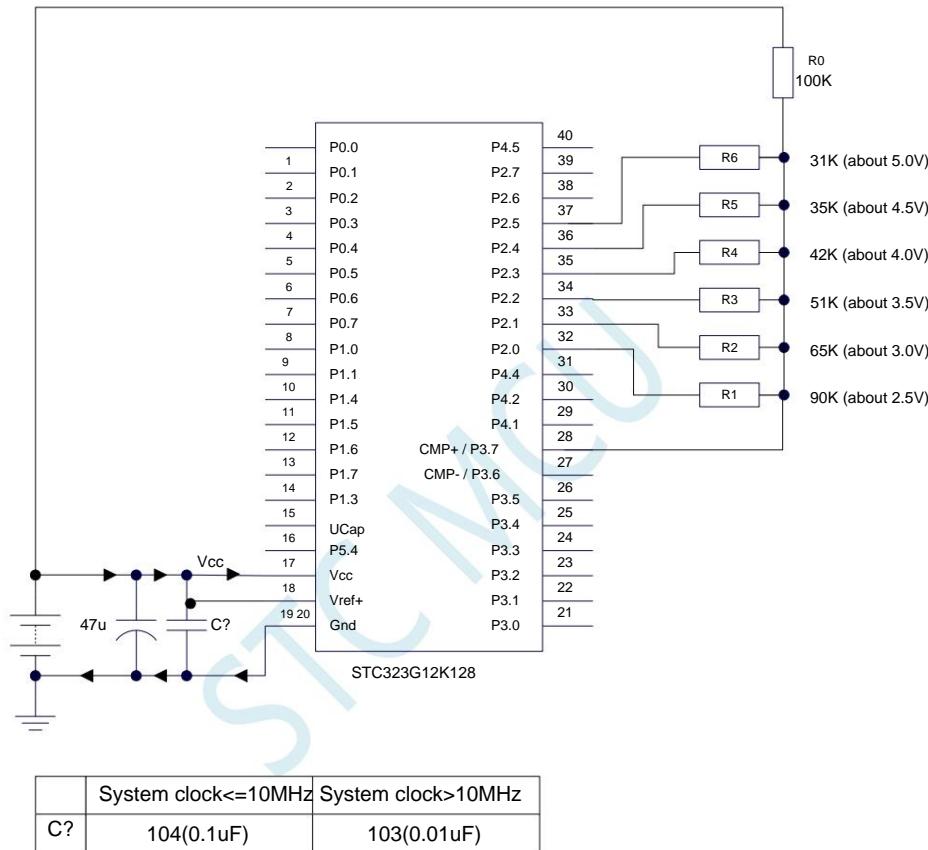


In the above figure, the resistors R1 and R2 divide the voltage at the front end of the voltage regulator block 7805, and the divided voltage is used as the external voltage of the comparator CMP+.

The input is compared to an internal 1.19V reference signal source.

Generally, when the AC power is 220V, the DC voltage at the front end of the voltage regulator block 7805 is 11V, but when the AC voltage drops to 160V, the DC voltage at the front end of the voltage regulator block 7805 is 8.5V. When the DC voltage at the front end of the voltage regulator block 7805 is lower than or equal to 8.5V, the DC voltage input by the front end is divided by resistors R1 and R2 to the positive input terminal CMP+ of the comparator, and the input voltage at the CMP+ terminal is lower than the internal 1.19V reference signal source. A comparator interrupt can be generated at this time to allow sufficient time to save the data to the EEPROM during brownout detection. When the DC voltage at the front end of the voltage regulator block 7805 is higher than 8.5V, the DC voltage input by the front end is divided by resistors R1 and R2 to the positive input terminal CMP+ of the comparator, and the input voltage at the CMP+ terminal is higher than the internal 1.19V reference signal source. The CPU can continue to work normally.

### 17.4.5 Comparator Detection Operating Voltage (Battery Voltage)



In the above figure, the working voltage of the MCU can be approximately measured by using the principle of resistive voltage divider (the gated channel, the IO port of the MCU outputs a low level, the port voltage value is close to Gnd, the channel that is not gated, the IO port of the MCU High output in open-drain mode, does not affect other channels).

The negative terminal of the comparator selects the internal 1.19V reference signal source, and the positive terminal selects the voltage value input to the CMP+ pin after dividing the voltage by a resistor.

During initialization, ports P2.5~P2.0 are set to open-drain mode and output high. First, the P2.0 port outputs a low level. At this time, if the Vcc voltage is low

If the voltage is lower than 2.5V, the comparison value of the comparator is 0; otherwise, if the Vcc voltage is higher than 2.5V, the comparison value of the comparator is 1;

If it is determined that Vcc is higher than 2.5V, then the P2.0 port will output a high level, and the P2.1 port will output a low level. At this time, if the Vcc voltage is lower than 3.0V, then compare the

The comparison value of the comparator is 0, otherwise, if the Vcc voltage is higher than 3.0V, the comparison value of the comparator is 1;

If it is determined that Vcc is higher than 3.0V, then the P2.1 port will output a high level, and the P2.2 port will output a low level. At this time, if the Vcc voltage is lower than 3.5V, then compare the

The comparison value of the comparator is 0, otherwise, if the Vcc voltage is higher than 3.5V, the comparison value of the comparator is 1;

If it is determined that Vcc is higher than 3.5V, the P2.2 port will output a high level, and the P2.3 port will output a low level. At this time, if the Vcc voltage is lower than 4.0V, then compare the

The comparison value of the comparator is 0, otherwise, if the Vcc voltage is higher than 4.0V, the comparison value of the comparator is 1;

If it is determined that Vcc is higher than 4.0V, then the P2.3 port will output a high level, and the P2.4 port will output a low level. At this time, if the Vcc voltage is lower than 4.5V, then compare the

The comparison value of the comparator is 0, otherwise, if the Vcc voltage is higher than 4.5V, the comparison value of the comparator is 1;

If it is determined that Vcc is higher than 4.5V, then the P2.4 port will output a high level, and the P2.5 port will output a low level. At this time, if the Vcc voltage is lower than 5.0V, then compare the

The comparison value of the comparator is 0, otherwise, if the Vcc voltage is higher than 5.0V, the comparison value of the comparator is 1.

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

void delay()
{
    char i;

    for (i=0; i<20; i++);
}

void main()
{
    unsigned char v;

    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                       //access to set external data bus speed to fastest
    WTST = 0x00;                                         //Set the program code to wait for parameters,
                                                       //assign to      CPU   The speed of executing the program is set to the fastest

    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 Initialize in open-drain mode
    P2M1 = 0x3f;
    P2 = 0xff;

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

    CMPCR2 = 0x10;
    CMPCR1 = 0x00;
    CMPPEN = 1;                                         //The comparator result goes through 16 output after debounced clocks
                                                       //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 microcontroller integrates a large-capacity EEPROM inside. Using ISP/IAP technology, the internal Data Flash can be EEPROM, the number of erasing and writing is more than 100,000 times. EEPROM can be divided into several sectors, each sector contains 512 bytes.

Note: The write operation of EEPROM can only write 1 in the byte as 0. When it is necessary to write 0 in the byte as 1, the sector must be executed.

Erase operation. EEPROM read/write operations are performed in 1-byte units, and EEPROM erase operations are performed in 1-sector (512 bytes)

When performing an erase operation, if there is data that needs to be retained in the target sector, these data must be read to RAM in advance

It is temporarily stored in the EEPROM, and after the erasing is completed, the saved data and the data to be updated are written back to the EEPROM/DATA-FLASH.

Therefore, when using EEPROM, it is recommended that the data modified at the same time be placed in the same sector, and the data that are not modified at the same time should be placed in different sectors.

The same sector does not have to be full. Data memory erase operations are performed in sectors (512 bytes per sector).

EEPROM can be used to save some parameter data that needs to be modified in the application process and will not be lost when power off. In the user program, you can

Byte read/byte program/sector erase operations are performed on EEPROM. When the working voltage is low, it is recommended not to operate the EEPROM.

In order to avoid sending data loss.

### 18.1 EEPROM Operation Time

ÿ Read 1 byte: 4 system clocks (use MOVC instruction to read more convenient and faster)

ÿ Programming 1 byte: about 30~40us (the actual programming time is 6~7.5us, but it is necessary to add the state transition time and various control

SETUP and HOLD time of the control signal)

ÿ Erase 1 sector (512 bytes): about 4~6ms

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

**IAP\_TPS=system operating frequency/1000000 (the decimal part is rounded to the nearest integer)**

For example: the 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

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP data register	C2H									0000,0000
IAP_DATA1	IAP data register	D9H									0000,0000
IAP_DATA2	IAP data register	DAH									0000,0000
IAP_DATA3	IAP data register	DBH									0000,0000
IAP_ADDRE	IAP extended address register	F6H									1111,1111
IAP_ADDRH	IAP high address register	C3H									0000,0000
IAP_ADDRL	IAP low address register	C4H									0000,0000
IAP_CMD	IAP Command Register	C5H	-	-	-	-	-		CMD[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)

symbol	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]			

When the EEPROM read operation is performed, the EEPROM data read out after the command is executed is stored in the IAP\_DATA register.

When performing the EEPROM write operation, before executing the write command, the data to be written must be stored in the IAP\_DATA register, and then send a write command. The Erase EEPROM command has nothing to do with the IAP\_DATA register.

Note: STC32G12K128 series and STC32G8K64 series only support single-byte read and write operations, and the data register only has IAP\_DATA efficient. STC32F12K60 series supports 4-byte read and write operations, data registers IAP\_DATA, IAP\_DATA1, IAP\_DATA2, IAP\_DATA3 are valid

## 18.2.2 EEPROM Address Register (IAP\_ADDR)

symbol	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]			

The target address register for EEPROM read, write, and erase operations. IAP\_ADDRH holds the high byte of the address, IAP\_ADDRL save the low byte of the address

## 18.2.3 EEPROM Command Register (IAP\_CMD)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[2:0]

CMD[2:0]: Send EEPROM operation command

000 (CMD0): No operation

001 (CMD1): read EEPROM byte command. Read the 1 byte where the target address ADDR[23:0] is located.

010 (CMD2): Write EEPROM byte command. Write the 1 byte where the target address ADDR[23:0] is located.

011 (CMD3): Erase EEPROM sector. Erase 1 sector (512 bytes) where target address ADDR[23:9] is located.

100 (CMD4): No operation

101 (CMD5): Write EEPROM word (4 bytes) command. Write the 4 bytes where the target address ADDR[23:2] is located.

110 (CMD6): Erase EEPROM half sector (256 bytes). Erase the 0.5 sector where the target address ADDR[23:8] is located (256 byte).

111 (CMD7): Erase EEPROM block. Erase the 32 sectors (16K bytes) where the target address ADDR[23:15] is located.

Note: STC32G12K128 series and STC32G8K64 series only support CMD0, CMD1, CMD2, CMD3. STC32F12K60

Series supports all commands.

## 18.2.4 EEPROM Trigger Register (IAP\_TRIG)

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

IAP\_TRIG is set to complete the command register, address register, data register and control register of EEPROM read, write and erase, it needs to be To trigger the corresponding read, write,

Erase operation. After the operation is completed, the EEPROM address registers IAP\_ADDRH, IAP\_ADDRL and the EEPROM command register The contents of IAP\_CMD remain unchanged. If you want to operate on the data of the next address next, you need to manually update the address register

The value of IAP\_ADDRH and register IAP\_ADDRL.

Note: In each EEPROM operation, write 5AH to IAP\_TRIG first, and then write A5H, the corresponding command will take effect.

After writing the trigger command, the CPU will be in the IDLE waiting state, and the CPU will not be released from the IDLE state until the corresponding IAP operation is completed.

Return to normal state and continue to execute CPU instructions.

## 18.2.5 EEPROM Control Register (IAP\_CONTR)

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

IAPEN: EEPROM operation enable control bit

0: Disable EEPROM operation

1: Enable EEPROM operation

SWBS: software reset selection control bit, (need to be used in conjunction with SWRST)

0: Execute program from user code after software reset

1: Start program execution from the system ISP monitoring code area after software reset

SWRST: Software Reset Control Bit

0: no action

1: Generate software reset

CMD\_FAIL: EEPROM operation failure status bit, needs to be cleared by software

0: EEPROM operation is correct

1: EEPROM operation failed

## 18.2.6 EEPROM Erase Wait Time Control Register (IAP\_TPS)

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

Need to be set according to the operating frequency

If the working frequency is 12MHz, you need to set IAP\_TPS to 12; if the working frequency is 24MHz, you need to set IAP\_TPS to 24,

And so on for other frequencies.

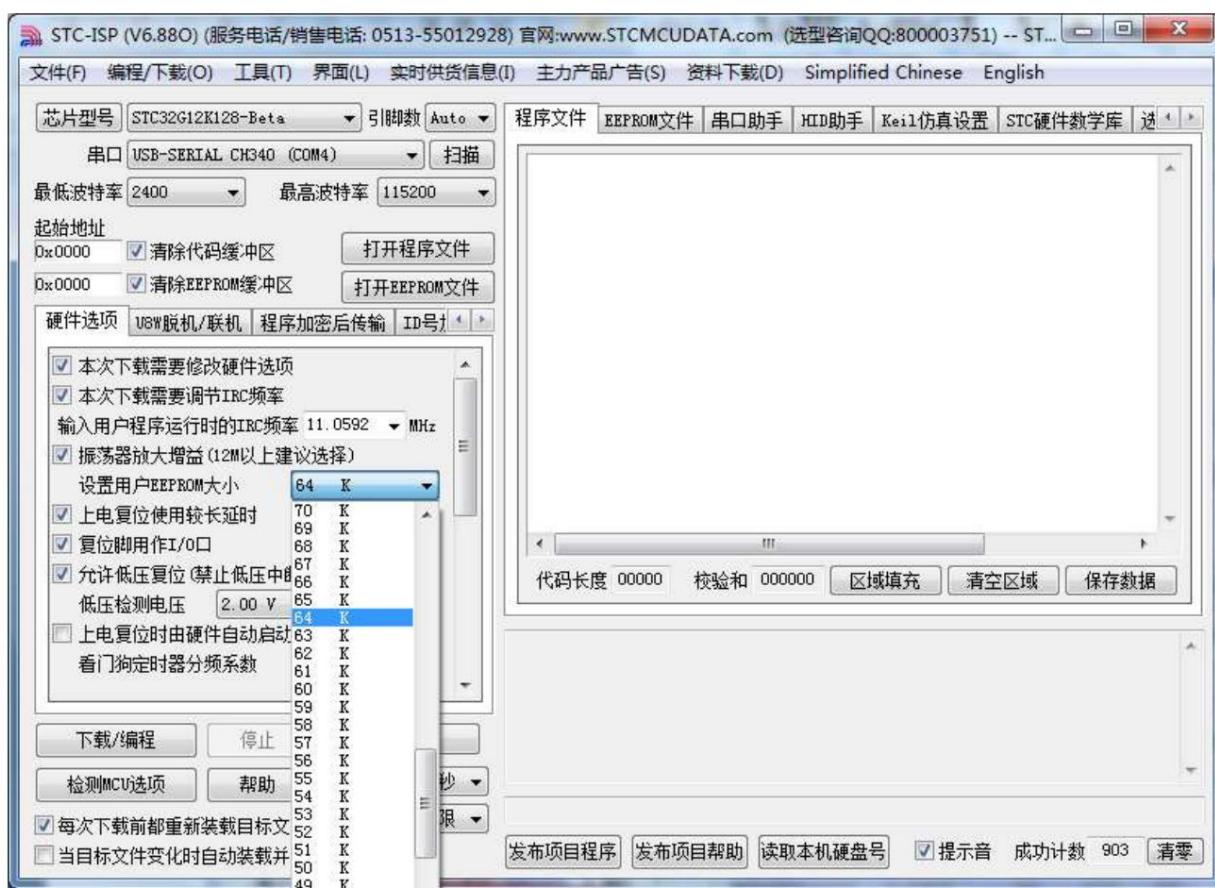
## 18.3 EEPROM Size and Address

STC32G series microcontrollers all have EEPROM used to save user data. The internal EEPROM has 3 operation modes: read, write and erase, in which the erase operation is performed in units of sectors, each sector is 512 bytes, that is, the erase command will be erased every time the erase command is executed. Except for one sector, read data and write data are operated in units of bytes, that is, each time a read or write command is executed, it can only be read out or write a byte.

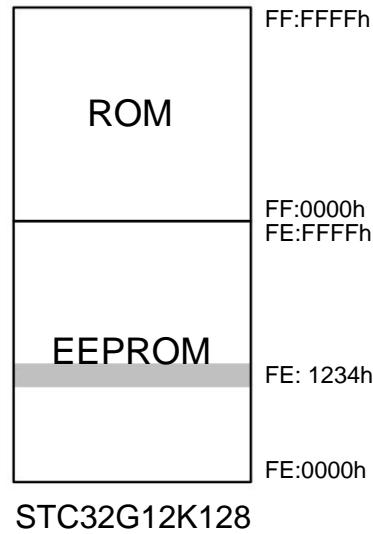
There are two ways to access the EEPROM inside the STC32G series microcontroller: IAP mode and MOV mode. IAP method can EEPROM performs read, write, and erase operations, but MOV can only perform read operations on EEPROM, but cannot perform write and erase operations. Whether using IAP or MOV to access the EEPROM, the correct target address needs to be set first. Using the IAP side In the mode, the address data is the target address of the EEPROM, and the address starts from 0000. 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 takes the model STC32G12K128 as an example.

The standard address is explained in detail:

The EEPROM size of STC32G12K128 needs to be set during ISP download, as shown in the figure below, set the EEPROM for 64K



Then the EEPROM is located in FE:0000h~FE:FFFFh in the 128K Flash storage space (Note: No matter how much EEPROM is set, EEPROM always starts from FE:0000h in Flash memory space), as shown in the following figure:



STC32G12K128

Now, when you need to read, write, and erase the unit of EEPROM physical address 1234h, if you use the IAP method to access,

The set target address is 1234h, that is, IAP\_ADDRE is set to 00h, IAP\_ADDRH is set to 12h, IAP\_ADDRL is set to 34h,

Then set the corresponding trigger command to operate the 1234h unit correctly. But if you use MOV to read the EEPROM

1234h unit, the target address is the base address FE:0000h plus 1234h, that is, the 32-bit register DRx must be set to FE:1234h,

The MOV instruction can be used to read correctly (Note: STC32G series is different from STC8 series, MOVC cannot be used to read EEPROM).

The following table lists the situation of accessing EEPROM in IAP mode and MOV mode when setting different EEPROM sizes

model	Number of size sectors	IAP mode read/write/erase		MOV read	
		start address	end address	start 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 Example Program

### 18.4.1 EEPROM Basic Operation

---

```
//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void lapIdle()
{
    IAP CONTR = 0; //turn off IAP
    IAP CMD = 0; //clear command register
    IAP TRIG = 0; //clear trigger register
    IAP ADDR = 0x00;
    IAP ADDRH = 0x00;
    IAP ADDRL = 0x00;
}

char lapRead(unsigned long addr)
{
    char dat;

    IAP CONTR = 0x80; //Enable IAP
    IAP TPS = 12; //setting waits for parameter 12MHz
    IAP CMD = 1; //setting read command
    IAP ADDR = addr; //set low address
    IAP ADDRH = addr >> 8; //set high address
    IAP ADDRE = addr >> 16; //set the highest address
    IAP TRIG = 0x5a; //Write Trigger Command(0x5a)
    IAP TRIG = 0xa5; //Write Trigger Command(0xa5)

    _nop_();
    _nop_();
    _nop_();
    _nop_();

    dat = IAP DATA; // IAP data
    lapIdle(); //read IAP Function

    return dat;
}

void lapProgram(unsigned long addr, char dat)
{
    IAP CONTR = 0x80; //Enable IAP
    IAP TPS = 12; //setting wait for parameter 12MHz
    IAP CMD = 2; //setting write command
    IAP ADDR = addr; //set low address
    IAP ADDRH = addr >> 8; //set high address
    IAP ADDRE = addr >> 16; //set the highest address
    IAP DATA = dat; //write IAP
    IAP TRIG = 0x5a; //Write Trigger Command(0x5a)
    IAP TRIG = 0xa5; //Write Trigger Command(0xa5)

    _nop_();
    _nop_();
}
```

```

_nop_();
_nop_();
lapIdle();                                     //closure IAP Function
}

void lapErase(unsigned long addr)
{
    IAP_CONTR = 0x80;                           //Enable IAP
    IAP_TPS = 12;                                //Set Wait Parameter 12MHz
    IAP_CMD = 3;                                 //Set Erase Command
    IAP_ADDRL = addr;                            //set low IAP address
    IAP_ADDRH = addr >> 8;                      //set high IAP address
    IAP_ADDRE = addr >> 16;                     //set the highest address
    IAP_TRIG = 0x5a;                            //Write Trigger Command(0x5a)
    IAP_TRIG = 0xa5;                            //Write Trigger Command(0xa5)

    _nop_();
    _nop_();
    _nop_();
    _nop_();                                     //
    lapIdle();                                     //closure IAP Function
}

void main()
{
    EAXFR = 1;                                  //Enable XFR
    CKCON = 0x00;                               //access to set external data bus speed to fastest
    WTST = 0x00;                                //Set the program code to wait for parameters,
                                                //assign to CPU The speed of executing the program is set to the fastest

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

    lapErase(0x0400);
    P0 = lapRead(0x0400);                         //P0=0xff
    lapProgram(0x0400, 0x12);
    P1 = lapRead(0x0400);                         //P1=0x12

    while (1);
}

```

## 18.4.2 Reading EEPROM with MOV

---

//The test frequency is 11.0592MHz

```

##include "stc8.h"
#include "stc32g.h"                                // For header files, see Download Software

```

```

#include "intrins.h"

#define IAP_BASE          0xfe0000

void lapIdle()
{
    IAP CONTR = 0;                                //turn off IAP
    IAP CMD = 0;                                  //clear command register
    IAP TRIG = 0;                                 //clear trigger register
    IAP_ADDRE = 0x00;
    IAP_ADDRH = 0x00;
    IAP_ADDRL = 0x00;
}

char lapRead(unsigned long addr)
{
    Addr = (addr & 0xffff) | IAP_BASE;           //use MOV read EEPROM Need to add base address
    return *(char far *)(addr);                  //use MOV read data
}

void lapProgram(unsigned long addr, char dat)
{
    IAP CONTR = 0x80;                            //Enable IAP
    IAP TPS = 12;                               //setting wait for parameter 12MHz
    IAP CMD = 2;                                //setting write command
    IAP_ADDRL = addr;                           //set low address
    IAP_ADDRH = addr >> 8;                      //set high address
    IAP_ADDRE = addr >> 16;                     //set the highest address
    IAP DATA = dat;                            //write data
    IAP TRIG = 0x5a;                            //Write Trigger Command(0x5a)
    IAP TRIG = 0xa5;                            //Write Trigger Command(0xa5)

    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    lapIdle();                                //closureIAP Function
}

void lapErase(unsigned long addr)
{
    IAP CONTR = 0x80;                            //Enable IAP
    IAP TPS = 12;                               //Set Wait Parameter 12MHz
    IAP CMD = 3;                                //Set Erase Command
    IAP_ADDRL = addr;                           //set low address
    IAP_ADDRH = addr >> 8;                      //set high address
    IAP_ADDRE = addr >> 16;                     //set the highest address
    IAP TRIG = 0x5a;                            //Write Trigger Command(0x5a)
    IAP TRIG = 0xa5;                            //Write Trigger Command(0xa5)

    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    lapIdle();                                //closureIAP Function
}

void main()
{
    EAXFR = 1;                                //Enable XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
}

```

```

WTST = 0x00;
// Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

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

lapErase(0x0400);
P0 = lapRead(0x0400); // P0=0xff
lapProgram(0x0400, 0x12);
P1 = lapRead(0x0400); // P1=0x12

while (1);
}

```

### 18.4.3 Send EEPROM data using serial port

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#ifndef include "stc32g.h"
#ifndef include "intrins.h"
// For header files, see Download Software

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

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; // turn off IAP
    IAP_CMD = 0; // clear command register
}

```

```

IAP_TRIG = 0;                                //clear trigger register
IAP_ADDRE = 0x00;
IAP_ADDRH = 0x00;
IAP_ADDRL = 0x00;
}

char lapRead(unsigned long addr)
{
    char dat;

    IAP CONTR = 0x80;                           //Enable IAP
    IAP TPS = 12;                             //setting waits for parameter 12MHz
    IAP CMD = 1;                            //setting read command
    IAP ADDRL = addr;                         //set low IAP address
    IAP ADDRH = addr >> 8;                  //set high IAP address
    IAP ADDRE = addr >> 16;                 //set the highest address
    IAP TRIG = 0x5a;                          //Write Trigger Command(0x5a)
    IAP TRIG = 0xa5;                          //Write Trigger Command(0xa5)

    _nop_();
    _nop_();
    _nop_();
    _nop_();
    dat = IAP DATA;                          // IAP data
    lapIdle();                               //read IAP Function

    return dat;
}

void lapProgram(unsigned long addr, char dat)
{
    IAP CONTR = 0x80;                           //Enable IAP
    IAP TPS = 12;                             //setting wait for parameter 12MHz
    IAP CMD = 2;                            //setting write command
    IAP ADDRL = addr;                         //set low IAP address
    IAP ADDRH = addr >> 8;                  //set high IAP address
    IAP ADDRE = addr >> 16;                 //set the 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_();
    lapIdle();                               //closure IAP Function
}

void lapErase(unsigned long addr)
{
    IAP CONTR = 0x80;                           //Enable IAP
    IAP TPS = 12;                             //Set Wait Parameter 12MHz
    IAP CMD = 3;                            //Set Erase Command
    IAP ADDRL = addr;                         //set low IAP address
    IAP ADDRH = addr >> 8;                  //set high IAP address
    IAP ADDRE = addr >> 16;                 //set the highest address
    IAP TRIG = 0x5a;                          //Write Trigger Command(0x5a)
    IAP TRIG = 0xa5;                          //Write Trigger Command(0xa5)

    _nop_();
    _nop_();
    _nop_();
}

```

```

_nop_();
lapIdle();
}

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;
    //Enable      XFR
    //access to set external data bus speed to fastest
    //Set the program code to wait for parameters,
    //assign to      CPU  The speed of executing the program is set to the fastest

    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();
    lapErase(0x0400);
    UartSend(lapRead(0x0400));
    lapProgram(0x0400, 0x12);
    UartSend(lapRead(0x0400));

    while (1);
}

```

#### 18.4.4 Serial port 1 read and write EEPROM - read with MOV

(main.c)

//The test frequency is 11.0592MHz

```

/*
 * This program has been tested and is completely normal. Telephone technical support is not provided. If you do not understand, please supplement the relevant basis by yourself.
 */
***** The function of this program *****
STC32G describes the series of general EEPROM programs to modify the
program and directly download the test. When using, pay attention to the erase and writing.
11.0592MHZ
PC : 115200,8,n,1.
EEPROM          64byte, read   64byte operations.

```

Command :

**E 0** **EEPROM** Perform sector erasing operation, **E** indicating that the erasing number **0** is sector decimal For , **0~126**, , see detail **IC**.  
**W 0** example **EEPROM** writing operation, indicating that the writing number is sector decimal **0** ( **0~126**, see detail **IC**). Consecutive from the start address of the sector  
Write **64** to read.  
**R 0** byte **EEPROM** conduct **IAP** read operation, **R** Indicates reading numbers **0** for **0** sector(decimal, **0~126**, see specific **IC**). start address from sector  
continuous reading **64** to-byte.  
**M 0** pairs consecutively from **MOVC** Read operation(operation address is **\*512+Offset address**, the number is sector decimal **0**, **0~126**, see specific **IC**).  
the start address of the sector **64** sector byte

\* 2019-6-10  
Note: For general purpose, the program does not recognize whether the sector is valid or not, and the user decides according to the specific model, date  
\*\*\*\*\*\*/

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

#define Baudrate1          115200L
#define UART1_BUF_LENGTH   10
#define EEADDR_OFFSET #define 0xfe0000      //definition EEPROM use MOV Base address added when accessing
TimeOutSet1           5

/ *****
Local constant statement last      *****/
u8 code T_Strings[]={"  year in this door today, the faces of the peach blossoms are red. I don't know where the face is going, but the peach blossoms are still smiling in the spring breeze. "};

/ *****
local variable declaration      *****/
u8 xdatatmp[70];
u8 xdataRX1_Buffer[UART1_BUF_LENGTH];
u8 RX1_Cnt;
u8 RX1_TimeOut;
bit B_TX1_Busy;

/ *****
local function declaration      *****/
void UART1_config(void);
void TX1_write2buff(u8 dat);
void PrintString1(u8 *puts);

/ *****
External function and variable declarations      *****/
/ *****
u8 CheckData(u8 dat)
{
    if((dat >= '0') && (dat <= '9')) return (dat-'0');
    if((dat >= 'A') && (dat <= 'F')) return (dat-'A'+10);
    if((dat >= 'a') && (dat <= 'f')) return (dat-'a'+10);
    return 0xff;
}

u16 GetAddress(void)
{
    u16 address;
    u8 i;

    address = 0;
    if(RX1_Cnt < 3)      return 65535;          //error
    if(RX1_Cnt <= 5)      //5 Within bytes is sector operation, decimal
                           //supports command E0, E 12, E 120
                           //          W 0, W 12, W 120
                           //          R 0, R 12, R 120
    {
        for(i=2; i<RX1_Cnt; i++)
        {
            if(CheckData(RX1_Buffer[i]) > 9)
                return 65535;          //error
            address = address * 10 + CheckData(RX1_Buffer[i]);
        }
        if(address < 124)         //limited in 0~123 sector
    }
}
```

```

    {
        address <= 9;
        return (address);
    }
}
else if(RX1_Cnt == 8) //8 Byte direct address manipulation, hexadecimal ,
//support command E 0x1234, W 0x12b3, R 0xA00
{
    if((RX1_Buffer[2] == '0') && ((RX1_Buffer[3] == 'x') || (RX1_Buffer[3] == 'X')))
    {
        for(i=4; i<8; i++)
        {
            if(CheckData(RX1_Buffer[i]) > 0x0F)
                return 65535; //error
            address = (address << 4) + CheckData(RX1_Buffer[i]);
        }
        if(address < 63488)
            return (address); //limited in 0~123 sector
    }
}
return 65535; //error
}

//=====
// The : void delay_ms(u8 ms)
// function describes the delay function.
// Parameter ms, version ms Count here only supports 1~255ms. Automatically adapt to master clock
// date remains to be delayed
// : VER1.0
// : 2013-4-1
// :
//=====

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

//use MOV readEEPROM
void EEPROM_MOV_read_n(u16 EE_address, u8 *DataAddress, u16 number)
{
    u8 far *pc;

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

/******************************************** main function *****/
void main(void)

```

```

{
    u8 i;
    u16 addr;

    UART1_config(); // Select other value, 2: use do baud rate do baud ,
    //of baud rate to enable Timer1 rate .
    EA = 1; //total interrupt

    PrintString1("STC8 seriesMCU use serial port 1 test EEPROM program\r\n"); //UART1 send a string

    while(1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0) //timeout count
        {
            if(--RX1_TimeOut == 0)
            {
                if(RX1_Buffer[1] == ' ')
                {
                    addr = GetAddress();
                    if(addr < 63488) //limited in 0~123 sector
                    {
                        if(RX1_Buffer[0] == 'E') //PC request to erase a sector
                        {
                            EEPROM_SectorErase(addr);
                            PrintString1("Erase\r\n");
                        }
                    }
                    else if(RX1_Buffer[0] == 'W') //PC { request to write EEPROM 64 byte data
                        EEPROM_write_n(addr,T_Strings,64);
                        PrintString1("Write\r\n");
                    }
                    else if(RX1_Buffer[0] == 'R') //PC { request return 64 byte EEPROM data
                        PrintString1("IAP The read data is as follows: \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(0xd);
                        TX1_write2buff(0xa);
                    }
                    else if(RX1_Buffer[0] == 'M') //PC { request return 64 byte EEPROM data
                        PrintString1("MOVC \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(0xd);
                        TX1_write2buff(0xa);
                    }
                    else PrintString1(" command error!\r\n");
                }
                else PrintString1(" command error!\r\n");
            }
            RX1_Cnt = 0;
        }
    }
}

```

```

        }

}

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

//=====
// Function void PrintString1(u8 *puts)
// description Serial port send string function.
// Parameter puts: pointer returns
// version date remarks
//      : VER1.0
//      : 2014-11-28
//      :
//=====

void PrintString1(u8
*puts) { //send a string
    for (; *puts != 0; puts++) //end with stop 0
    {
        TX1_write2buff(*puts);
    }
}

//=====
// The   : void UART1_config(void)
// function : UART1 initialization function.
// description none.
// parameter none.
// returns : VER1.0
// the   : 2014-11-28
// version date remarks
//=====

void UART1_config(void)
{
    TR1 = 0; //S1 BRT Use Timer1;
    AUXR &= ~0x01; //Timer1 set as 1T mode
    AUXR |= (1<<6); //Timer1 set As Timer
    TMOD &= ~(1<<6); //Timer1_16bitAutoReload;
    TMOD &= ~0x30;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0; // Disable Timer1
    INT_CLKO &= ~0x02; // Timer1 Does not output high-speed clock
    TR1 = 1; // 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 Sync Shift Output Bit Data , ,
                                //      0x40: 8 Variable P1n Standard Bit Data
                                //      0x80: 9
}

```

```

//      0xc0: 9 Bit data variable baud rate
//PS = 1;           //high priority interrupt
//ES = 1;           //enable interrupt
//REN = 1;          //Allow to receive

B_TX1_Busy = 0;
RX1_Cnt = 0;
}

//=====
// The   : void UART1_int (void) interrupt UART1_VECTOR
// function : UART1    interrupt function.
// description: nine.
// parameter: none.
// returns  : VER1.0
// the     : 2014-11-28
// version date remarks
//=====

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUFL_LENGTH)
            RX1_Cnt = 0;                                //Spill proof
        RX1_Buffer[RX1_Cnt++] = SBUF;
        RX1_TimeOut = TimeOutSet1;
    }

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

```

**(EEPROM.c)**

```

//The test frequency is 11.0592MHz

// This program is STC series built-in EEPROM  Read and write programs.

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

//=====
// function: void ISP_Disable(void)
// description: forbidden access ISP/IAP.
// parameter: none
//      : none.
//      : V1.0, 2012-10-22
//=====

void DisableEEPROM(void)
{
    ISP_CONTR = 0;                                //prohibit ISP/IAP operate
    IAP_TPS = 0;                                  //remove command
    ISP_CMD = 0;                                  //Prevent command from being triggered by mistake
    ISP_TRIG = 0;                                 //clear address high byte
    ISP_ADDRE = 0xff;
}

```

```

ISP_ADDRH = 0xff; // clear address high byte
ISP_ADDRL = 0xff; // Clear the low byte of the address, pointing to a non-EEPROM area to prevent misoperation
}

//=====
// Function: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// description starts from the specified EEPROM. The first address reads a byte and puts it into the specified buffer. The *
// parameters : EE_address: EEPROM first address of the read. Read length is the first address of
// DataAddress:
// number:
// Back : non.
// to version V1.0, 2012-10-22
//=====

void EEPROM_read_n(u32 EE_address, u8 *DataAddress, u16 number)
{
    EA = 0; //Disable interrupts
    ISP_CONTR = ISP_EN; //allow operation
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //Working frequency setting
    ISP_READ(); //Send byte read command, when the command does not need to be changed, there is no need to send the command again

    do
    {
        ISP_ADDRE = EE_address / 65536; //Send the high byte of the address (the address needs to be re-sent when the address needs to be changed)
        ISP_ADDRH = (EE_address / 256) % 256; //Send the high byte of the address (the address needs to be re-sent when the address needs to be changed)
        ISP_ADDRL = EE_address % 256; //send address low byte
        ISP_TRIG(); //Send 5AH, then send A5H arrive ISP/IAP trigger register, it every time
        // After sending, the command is triggered to start immediately
        // CPU IAP After waiting for completion, the program execution will continue.

        _nop_();
        _nop_();
        _nop_();
        _nop_();
        *DataAddress = ISP_DATA; //The read data is sent to
        EE_address++; //it every time
        DataAddress++; //After sending, the command is triggered to start immediately
    }while(--number);

    DisableEEPROM(); //CPU IAP After waiting for completion, the program execution will continue.
    EA = 1; //Re-enable interrupts
}

/**************************************** Sector Erase Function *****/
//=====
// Function: void EEPROM_SectorErase(u16 EE_address)
// description Returns the version of the sector to be erased by the sector erase
// parameters : EE_address: EEPROM the address of
// : non.
// : V1.0, 2013-5-10
//=====

void EEPROM_SectorErase(u32 EE_address)
{
    EA = 0; //Disable interrupts
    ISP_ADDRE = EE_address / 65536; //Only sector erase, no byte erase, byte 512 sector. /
    ISP_ADDRH = (EE_address / 256) % 256; //Any byte address in a sector is a sector address.
    ISP_ADDRL = EE_address % 256; //Send the high byte of the address (the address needs to be re-sent when the address needs to be changed)
    ISP_CONTR = ISP_EN; //Send the high byte of the address (the address needs to be re-sent when the address needs to be changed)
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L); //send address low byte
    //allow operation
    //Working frequency setting
}

```

```

ISP_ERASE();
ISP_TRIG();
_nop_();
_nop_();
_nop_();
_nop_();
DisableEEPROM();
EA = 1; //Re-enable interrupts
}

=====

// Function: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// description Write the buffered bytes to the parameter of the specified first
// address. Write the contents of the buffer memory locations of the source
// DataAddress:
// number:
// Back : non.
// to version V1.0, 2012-10-22
=====

void EEPROM_write_n(u32 EE_address, u8 *DataAddress, u16 number)
{
    EA = 0; //Disable interrupts

    ISP_CONTR = ISP_EN;
    IAP_TPS = (u8)(MAIN_Fosc / 1000000L);
    ISP_WRITE();
    do
    {
        ISP_ADDRE = EE_address / 65536;
        ISP_ADDRH = (EE_address / 256) % 256;
        ISP_ADDRL = EE_address % 256;
        ISP_DATA = *DataAddress;
        ISP_TRIG();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        EE_address++;
        DataAddress++;
    }while(--number);

    DisableEEPROM();
    EA = 1; //Re-enable interrupts
}

```

#### 18.4.5 Password Erase Write - Multi-sector Backup - Serial Port 1 Operation

##### (main.c)

//The test frequency is 11.0592MHz

```

/*
    This program has been tested and is completely normal. Telephone technical support is not provided. If you do not understand, please supplement the relevant basis by yourself.
    */
***** This program function *****
STC32G description series of general programs demonstrate, multi-sector backup, write with correct sector data if there is a sector error, all sector errors such as the first time ( )
Run the program and write the default value

```

3 Write one sector each time, that is, redundant backup. Write one record per sector. After writing, read the value data and check the source data and check value, and prompt each record from the serial port or error. Not all self-checking data are written to the wrong sector if there is a sector error, the data of the correct sector will be written to the wrong sector. If all sectors are wrong, the default value will be written to the wrong sector. Before performing the operation, you need to set the password to the operation. If the password is incorrect and clears the password each time it exits

1 (P3.0 P3.1) The return result is correct

is to ensure that the write number

incorrect and clears the password each time it exits

Please do not modify the program directly to download the  
frequently 11.0592MHZ.

"03- Password erase and write in multi-sector backup serial port operation

"UART-EEPROM.hex"

Select main when testing download

PC Serial port setting baud 115200,8,n,1.  
EEPROM rate for sector erasing and writing 64 byte, read 64 byte operations.

Command example:

Use the serial port assistant to send a single character in upper or lower case .

send E or e: right EEPROM perform a sector erase operation, E Indicates that erasing will erase the sector 0 , 1 , 2.

send W or w: Perform a write operation on the pair, indicating that the write will be written to the sector sector write 0 , 1 , 2, Sequential write per sector 64 byte sector 0 write 0x0000~0x003f, sector write 1 0x0200~0x023f, 0 0x0400~0x043f.

send R or r: The read operation is performed on the pair, indicating that the read will read out the sector sector read out the sector 0 , 1 , 2, Continuous read per sector 64 byte sector 0 read out 0x0000~0x003f, read out 1 0x0200~0x023f, 0 0x0400~0x043f.

Note: For general purposes, the program does not recognize whether the sector is valid or not, and the user decides according to the specific model, date  
\*\*\*\*\*

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

#define Baudrate1 115200L

/***** Local constant statement last *****/
u8 code T_StringD[]={"u8 year in this door today, the faces of the peach blossoms are red. I don't know where the face is going, but the peach blossoms are still smiling in the spring
code T_StringW[]=""; //breeze. It is seen as a ridge and a peak on the side, with different heights and heights. I do not know the true face of Mount Lu, only because I am in this mountain"};
//General data
//array to write
```

```
bit B_TX1_Busy;
u8 cmd; //Serial single character command
```

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

```
/***** External function and variable declarations *****/
/***** read EEPROM Record, and verify and return the verification result , 0 to be correct, 1 for error *****/
u8 ReadRecord(u16 addr)
{
```

```



for(i=0; i<66; i++) tmp[i] = 0;
PassWord = D_PASSWORD;
EEPROM_read_n(addr,tmp,66);
for(ChckSum=0, i=0; i<64; i++)
    ChckSum += tmp[i];
j = ((u16)tmp[64]<<8) + (u16)tmp[65]; j
^= 0x5555; if(ChckSum != j) return 1; 0;
return
}

/ **** write EEPROM Record and verify and return the verification result , 0 to be correct 1 for error ****/
u8 SaveRecord(u16 addr)
{
    u8 i;
    u16 ChckSum;

    for(ChckSum=0, i=0; i<64; i++)
        ChckSum += SaveTmp[i];
    ChckSum ^= 0x5555;
    SaveTmp[64] = (u8)(ChckSum >> 8);
    SaveTmp[65] = (u8)ChckSum;

    PassWord = D_PASSWORD;
    EEPROM_SectorErase(addr);
    PassWord = D_PASSWORD;
    EEPROM_write_n(addr, SaveTmp, 66);

    for(i=0; i<66; i++)
        tmp[i] = 0;
    PassWord = D_PASSWORD;
    EEPROM_read_n(addr,tmp,66);
    for(i=0; i<66; i++) {
        if(SaveTmp[i] != tmp[i])
            return 1;
    }
    return 0;
}

/ **** main function ****/
void main(void)
{
    u8 i;
    u8 status;

    UART1_config();
    EA = 1;

    PrintString1("STC8G-8H-8C seriesMCU Test with serial port EEPROM program\r\n"); //UART1 send a string

    // Power on to read a sector and verify that if there is a sector error, it will be correct
    // Sector write error area, If every sector is wrong, write the default value
    status = 0;
}

```

```

if(ReadRecord(0x0000) == 0) //read sector 0
{
    status |= 0x01; //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; //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; //correct mark status.2=1
    for(i=0; i<64; i++)
        SaveTmp[i] = tmp[i]; //save in write buffer
}

if(status == 0) //if all sectors are wrong, write default value
{
    for(i=0; i<64; i++)
        SaveTmp[i] = T_StringD[i]; //read default value
}
else PrintString1("The data of each sector is correct when powered on !\r\n"); //UART1 send a string prompt

if((status & 0x01) == 0) //Write default value for sector error
{
    if(SaveRecord(0x0000) == 0)
        PrintString1(" write sector is 0correct !\r\n");
    else
        PrintString1(" write sector0error !\r\n"); //Write the recording sector correctly
}
if((status & 0x02) == 0) //Write default value for sector error
{
    if(SaveRecord(0x0200) == 0)
        PrintString1(" write sector is 1correct !\r\n");
    else
        PrintString1(" write sector1error !\r\n"); //write record 1sector error
}
if((status & 0x04) == 0) //Write default value for sector error
{
    if(SaveRecord(0x0400) == 0)
        PrintString1(" write sector is 2correct !\r\n");
    else
        PrintString1(" write sector2error !\r\n"); //write record 2sector error
}

while(1)
{
    if(cmd != 0) //There are 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("Sector0 erase complete");
}

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

else if(cmd == 'R')                   //PC request return 64 byte EEPROM data
{
    if(ReadRecord(0x0000) == 0)          //read sector0 The data
    {
        The data of the read sector is as follows: !\r\n;
        PrintString1("for(i=0; i<64; i++)");
        TX1_write2buff(tmp[i]);          //Return data to serial port
        TX1_write2buff(0xd);             // carriage return line feed
        TX1_write2buff(0xa);
    }
    else PrintString1(" Data error i0 read sector !\r\n");

    if(ReadRecord(0x0200) == 0)          //read sector1 The data
    {
        The data of the read sector is as follows: !\r\n;
        PrintString1("for(i=0; i<64; i++)");
        TX1_write2buff(tmp[i]);          //Return data to serial port
        TX1_write2buff(0xd);             // carriage return line feed
        TX1_write2buff(0xa);
    }
    else PrintString1(" Data error i1 read sector !\r\n");

    if(ReadRecord(0x0400) == 0)          //read sector2 The data
    {
        The data of the read sector is as follows: !\r\n;
        PrintString1("for(i=0; i<64; i++)");
        TX1_write2buff(tmp[i]);          //Return data to serial port
        TX1_write2buff(0xd);             // carriage return line feed
        TX1_write2buff(0xa);
    }
    else PrintString1(" Data error i2 read sector !\r\n");
}

else PrintString1(" command error!\r\n");

```

```

        cmd = 0;
    }

}

// **** send a byte ****
void TX1_write2buff(u8 dat) { //write send buffer

    B_TX1_Busy = 1; //flag send busy
    SBUF = dat; //send a byte
    while(B_TX1_Busy); //wait for sending
}

//=====
// Function void PrintString1(u8 *puts)
// description Serial port send string function.
// Parameter puts: pointer returns
// version date remarks
//      : VER1.0
//      : 2014-11-28
//      :
//=====

void PrintString1(u8 *puts) //send a string
{
    for (; *puts != 0; puts++) //end with stop 0
    {
        TX1_write2buff(*puts);
    }
}

//=====
// The : void UART1_config(void)
// function : UART1 initialization function.
// description: none.
// parameter: none.
// returns : VER1.0
// the : 2014-11-28
// version date remarks
//=====

void UART1_config(void)
{
    TR1 = 0; //S1 BRT Use Timer1;
    AUXR &= ~0x01; //Timer1 set as 1T mode
    AUXR |= (1<<6); //Timer1 set As Timer
    TMOD &= ~(1<<6); //Timer1_16bitAutoReload;
    TMOD &= ~0x30;
    TH1 = (u8)((65536L-(MAIN_Fosc / 4) / Baudrate1) >> 8);
    TL1 = (u8)(65536L-(MAIN_Fosc / 4) / Baudrate1);
    ET1 = 0; // Disable Timer1
    INT_CLKO &= ~0x02; // Timer1 Does not output high-speed clock
    TR1 = 1; // 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 ,
                                //          0x40: 8 output bit data/variable baud,
}

```

```

//          // 0x80: 9 Bit Data Fixed Baud Rate ,
//          // 0xc0: 9 Bit Data Variable Baud Rate
// PS = 1;           //high priority interrupt
// ES = 1;           //enable interrupt
// REN = 1;           //Allow to receive

B_TX1_Busy = 0;
}

//=====
// The : void UART1_int (void) interrupt UART1_VECTOR
// function : UART1 interrupt function.
// description : nine.
// parameter : none.
// returns : VER1.0
// the : 2014-11-28
// version date remarks
//=====

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        cmd = SBUF;
    }

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

```

**(EEPROM.c)**


---

```

//The test frequency is 11.0592MHz

// This program is STC series built-in EEPROM Read and write programs.

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

u32 PassWord;           //Password required for erasing and writing

//=====
// function: void ISP_Disable(void)
// description: forbidden access ISP/IAP.
// parameter: none
// : none.
// : V1.0, 2012-10-22
//=====

void DisableEEPROM(void)
{
    ISP_CONTR = 0;           //prohibit ISP/IAP operate
    IAP_TPS = 0;             //remove command
    ISP_CMD = 0;              //Prevent commands from being triggered by mistake
    ISP_TRIG = 0;             //clear Q address high byte
    ISP_ADDRE = 0xff;         //clear Q address high byte
    ISP_ADDRH = 0xff;
}
```

```

ISP_ADDR = 0xff; // Clear the low byte of the address, pointing to a non-EEPROM area to prevent misoperation
}

//=====
// The function: void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
// description starts from the specified EEPROM. The first address reads a byte and puts it into the specified buffer. The *
// parameters : EE_address: first address of the read. read the data and length into the first address of
// DataAddress:
// number:
// Back : non.
// to version V1.0, 2012-10-22
//=====

void EEPROM_read_n(u32 EE_address, u8 *DataAddress, u16 number)
{
    if(PassWord == D_PASSWORD) { // The password is correct to operate EEPROM

        EA = 0; // Disable interrupts
        ISP_CONTR = ISP_EN; // allow operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); // Working frequency setting
        ISP_READ(); // Send byte read command, when the command does not need to be changed, there is no need to send the command again
        do
        {
            ISP_ADDRE = EE_address / 65536; // Send the high byte of the address (the address needs to be re-sent when the address needs to be changed)
            ISP_ADDRH = (EE_address / 256) % 256; // Send the high byte of the address (the address needs to be re-sent when the address needs to be changed)
            ISP_ADDRDL = EE_address % 256; // send address low byte
            if(PassWord == D_PASSWORD) { // The password is correct to trigger the operation

                ISP_TRIG = 0xA5; //Send first, then send A5H arrive ISP/IAP trigger register,
                ISP_TRIG = 0xA5; //it every time
                // After sending the command is triggered to start immediately
                // CPU After waiting for completion, the program execution will continue.

                _nop_();
                _nop_();
                _nop_();
                _nop_();
                *DataAddress = ISP_DATA; //The read data is sent to
                EE_address++; //DataAddress++;
            }
        }while(--number);

        DisableEEPROM(); //Re-enable interrupts
        EA = 1;
    }
    PassWord = 0; //clear password
}

***** Sector Erase Function *****/
//=====

// Function: void EEPROM_SectorErase(u16 EE_address)
// description Returns the version of the sector to be erased by the sector erase
// parameters : EE_address: EEPROM the address of
// : non.
// : V1.0, 2013-5-10
//=====

void EEPROM_SectorErase(u32 EE_address)
{
    if(PassWord == D_PASSWORD) { // The password is correct to operate EEPROM

        EA = 0; // Disable interrupts
    }
}

```

```

ISP_ADDRESS = EE_address / 65536;
ISP_ADDRH = (EE_address / 256) % 256;
ISP_ADDRL = EE_address % 256;
ISP_CONTR = ISP_EN;
IAP_TPS = (u8)(MAIN_Fosc / 1000000L);
ISP_ERASE();
if(PassWord == D_PASSWORD) {
    ISP_TRIG = 0x5A;
    ISP_TRIG = 0xA5;
} _nop_();
_nop_();
_nop_();
_nop_();
DisableEEPROM();
EA = 1;
}
PassWord = 0;
}

//=====
// Function: void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
// description Write the buffered bytes to the parameter of the specified first
// address. When the command is triggered, it triggers the reading of the source
// DataAddress:
// number:
// Back : non.
// to version V1.0, 2012-10-22
//=====

void EEPROM_write_n(u32 EE_address, u8 *DataAddress, u16 number)
{
    if(PassWord == D_PASSWORD) { // The password is correct to operate EEPROM
        EA = 0; // Disable interrupts
        ISP_CONTR = ISP_EN; // allow operation
        IAP_TPS = (u8)(MAIN_Fosc / 1000000L); // Working frequency setting
        ISP_WRITE(); // Send byte write command, when the command does not need to be changed, no need to send the command again
        do
        {
            ISP_ADDRESS = EE_address / 65536; // Send the high byte of the address (the address needs to be re-sent when the address needs to be changed)
            ISP_ADDRH = (EE_address / 256) % 256; // Send the high byte of the address (the address needs to be re-sent when the address needs to be changed)
            ISP_ADDRL = EE_address % 256; // send address low byte
            ISP_DATA = *DataAddress; // Send data to, ISP_DATA -> send when the data changes
            if(PassWord == D_PASSWORD) { // The password is correct to trigger the operation
                ISP_TRIG = 0x5A; // Send 5AH, then send A5H arrive ISP/IAP trigger register,
                ISP_TRIG = 0xA5; // it every time
            } _nop_();
            _nop_();
            _nop_();
            _nop_();
            EE_address++;
        }
    }
}

```

```
    DataAddress++;
}while(~number);

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

STCMCU

## 19 ADC analog-to-digital conversion, traditional DAC implementation

The STC32G series microcontroller integrates a 12-bit high-speed A/D converter. The clock frequency of the ADC is the system frequency divided by 2 and then The frequency is divided again by the frequency division factor set by the user (the ADC clock frequency range is SYSclk/2/1~SYSclk/2/16).

There are two data formats for ADC conversion results: left-justified and right-justified. It can be easily read and referenced by the user program.

**Note:** The 15th channel of the ADC is a channel dedicated to measuring the internal 1.19V reference signal source. The reference signal source value is calibrated as 1.19V, due to manufacturing errors and measurement errors, the actual internal reference signal source has an error of about  $\pm 1\%$  compared to 1.19V. If the user needs to know the accurate internal reference signal source value of each chip, the accurate reference signal source can be externally connected, and then use the 15th ADC value. **channel for measurement calibration. When the ADC\_VRef+ pin is connected to a reference power supply, the 15th channel of the ADC can be used to reverse the ADC\_VRef +** The voltage of the external reference power supply is connected to the pin; if **ADC\_VREF+ is shorted to** MCU-VCC, the voltage of **MCU-VCC can be reversed**.

If the chip has an external reference power pin **ADC\_VRef+** of the **ADC**, it must not be floating, and must be connected to an external reference power supply or Connect directly to **VCC**.

### 19.1 ADC related registers

symbol	describe	address	Bit addresses and symbols								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

symbol	describe	address	Bit addresses and symbols								reset value	
			B7	B6	B5	B4	B3	B2	B1	B0		
ADCTIM	ADC timing control register	7EFFEA8H	CSSETUP	CSHOLD	[1:0]			SMPDUTY[4:0]				0010,1010

#### 19.1.1 ADC control register (ADC\_CONTR), PWM triggers ADC control

system												
Symbol address	B7		B6	B5	B4	B3	B2	B1		B0		
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_POWER: ADC power control bit			ADC_CHS[3:0]			

0: Turn off ADC power

1: Power on the ADC.

It is recommended to power down the ADC before entering idle mode and power-down mode to reduce power consumption

pay attention:

1. After powering on the internal ADC module of the MCU, wait for about 1ms, and wait for the ADC power supply inside the MCU to stabilize

Then let the ADC work;

2. Properly lengthening the sampling time of the external signal is the charging or discharging time of the internal sampling and holding capacitor of the ADC.

The internal and external potentials are equal.

ADC\_START: ADC conversion start control bit. ADC conversion starts after writing 1, this bit is automatically cleared by hardware after conversion is completed.

0: No effect. Writing 0 will not stop the A/D conversion even if the ADC has started converting.

1: Start ADC conversion, hardware will automatically clear this bit after conversion is complete.

**ADC\_FLAG:** ADC conversion end flag. When the ADC completes a conversion, the hardware will automatically set this bit to 1 and present it to the CPU interrupt request. This flag must be cleared by software.

**ADC\_EPWMT:** Enable PWM real-time trigger ADC function. For details, please refer to the **16-bit Advanced PWM Timer chapter**

**ADC\_CHS[3:0]:** ADC analog channel selection bits

(Note: The I/O port selected as the ADC input channel must set the **PxM0/PxM1** register to set the I/O port mode to high-impedance output.

enter mode. In addition, if the ADC channel still needs to be enabled after the MCU enters the power-down mode/clock stop mode, you need to set **PxE**

The register closes the digital input channel to prevent the external analog input signal from being high and low and causing extra 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

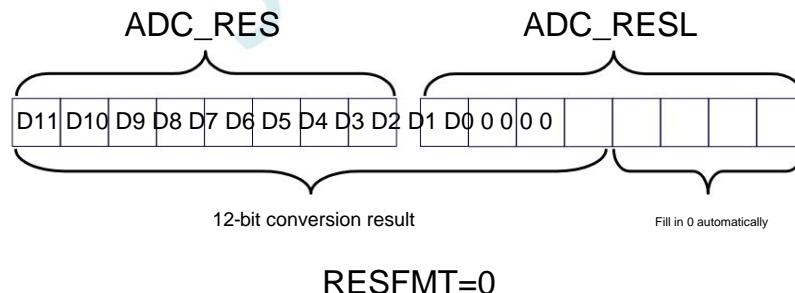
[1] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

### 19.1.2 ADC CONFIGURATION REGISTER (ADCCFG)

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

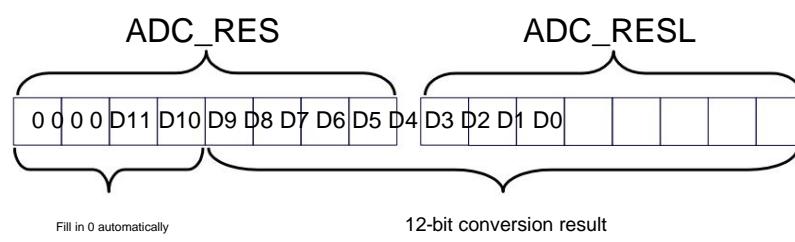
**RESFMT:** ADC conversion result format control bit

0: The conversion result is left-aligned. ADC\_RES holds the upper 8 bits of the result, and ADC\_RESL holds the lower 4 bits of the result. The format is as follows:



**RESFMT=0**

1: The conversion result is right-aligned. ADC\_RES holds the upper 4 bits of the result, and ADC\_RESL holds the lower 8 bits of the result. The format is as follows:



**RESFMT=1**

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)

Symbol 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 will be automatically saved in ADC\_RES and ADC\_RESL. Please save the data format of the results Refer to the RESFMT setting in the ADC\_CFG register.

### 19.1.4 ADC Timing Control Register (ADCTIM)

Symbol 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 Clocks	
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

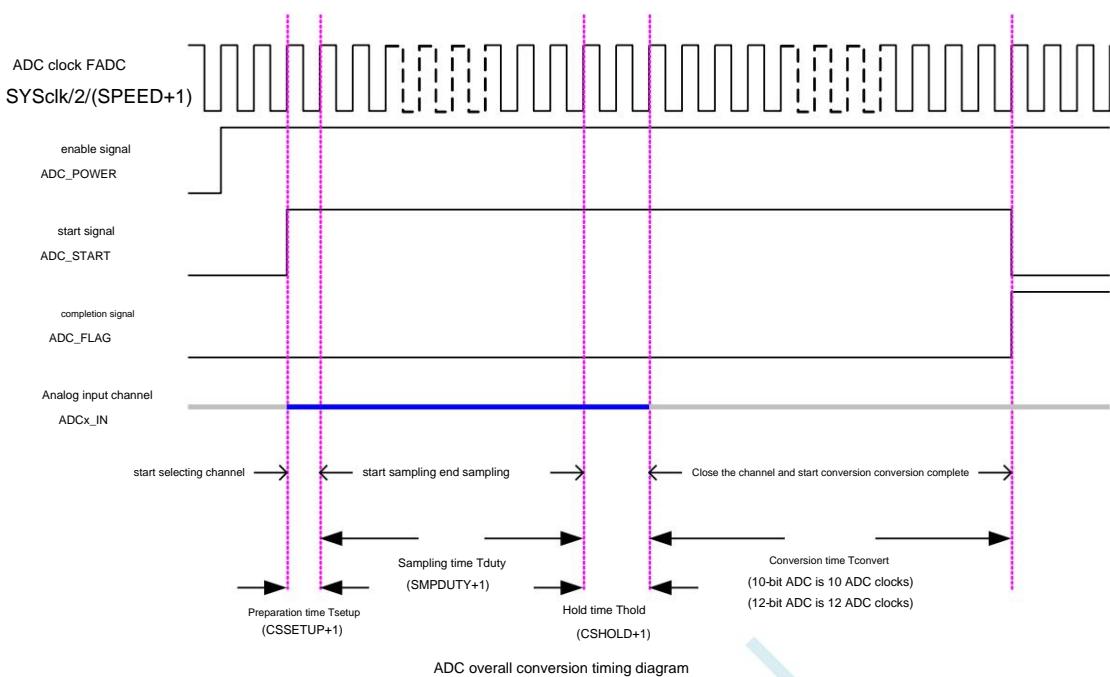
SMPDUTY[4:0]: ADC analog signal sampling time control Tduty (Note: SMPDUTY must not be set less than 01010B)

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

ADC digital-to-analog conversion time: Tconvert

12-bit ADC conversion time is fixed to 12 ADC operating clocks

A complete ADC conversion time is: Tsetup + Tduty + Thold + Tconvert, as shown in the figure below



## 19.2 ADC Static Characteristics

symbol	describe	Min	Typ	Max		unit
RES resolution		-		12	-	Bits
ET overall error		-		0.5	1	LSB
EO offset error		-		-0.1	1	LSB
EG gain error		-		0	1	LSB
ED Differential Nonlinear Error		-		0.7	1.5	LSB
EI integral nonlinear error		-		1	2	LSB
The resistance of the RAIN channel equivalent resistance		-		ÿ	-	ohm
RESD	connected in series before the sampling and holding capacitor Electrostatic resistance	-		700	-	ohm
CADC internal	sample and hold capacitor	-		16.5	-	pF

## 19.3 ADC related calculation formula

### 19.3.1 ADC Speed Calculation Formula

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

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

#### Notice:

- ÿ The speed of 12-bit ADC cannot be higher than 800KHz
- ÿ The value of SMPDUTY cannot be less than 10, it is recommended to set it to 15
- ÿ CSSETUP can use the power-on default value of 0
- ÿ CHOLD can use the power-on default value of 1 (ADCTIM is recommended to be set to 3FH)

### 19.3.2 ADC conversion result calculation formula

$$\text{The input voltage Vin of the ADC converted channel} = \frac{\text{12-bit ADC conversion result} = 4096 \times \text{Voltage of ADC external reference source}}{\text{(with independent ADC_Vref+ pin)}}$$

### 19.3.3 Reverse ADC input voltage calculation formula

$$\text{The input voltage Vin of the ADC converted channel} = \frac{\text{12-bit ADC conversion result}}{\text{4096}} = \frac{\text{Voltage of the ADC external reference source}}{\text{(with independent ADC_Vref+ pin)}}$$

#### 19.3.4 Calculation formula of reverse working voltage

When it is necessary to use the ADC input voltage and the ADC conversion result to reverse the working voltage, if the target chip does not have an independent ADC\_Vref+ pin, then The following formula can be directly measured and used. If the target chip has an independent ADC\_Vref+ pin, the ADC\_Vref+ pin must be connected to Vcc pin.

$$\text{The input voltage } V_{in} \text{ of the ADC converted channel}$$

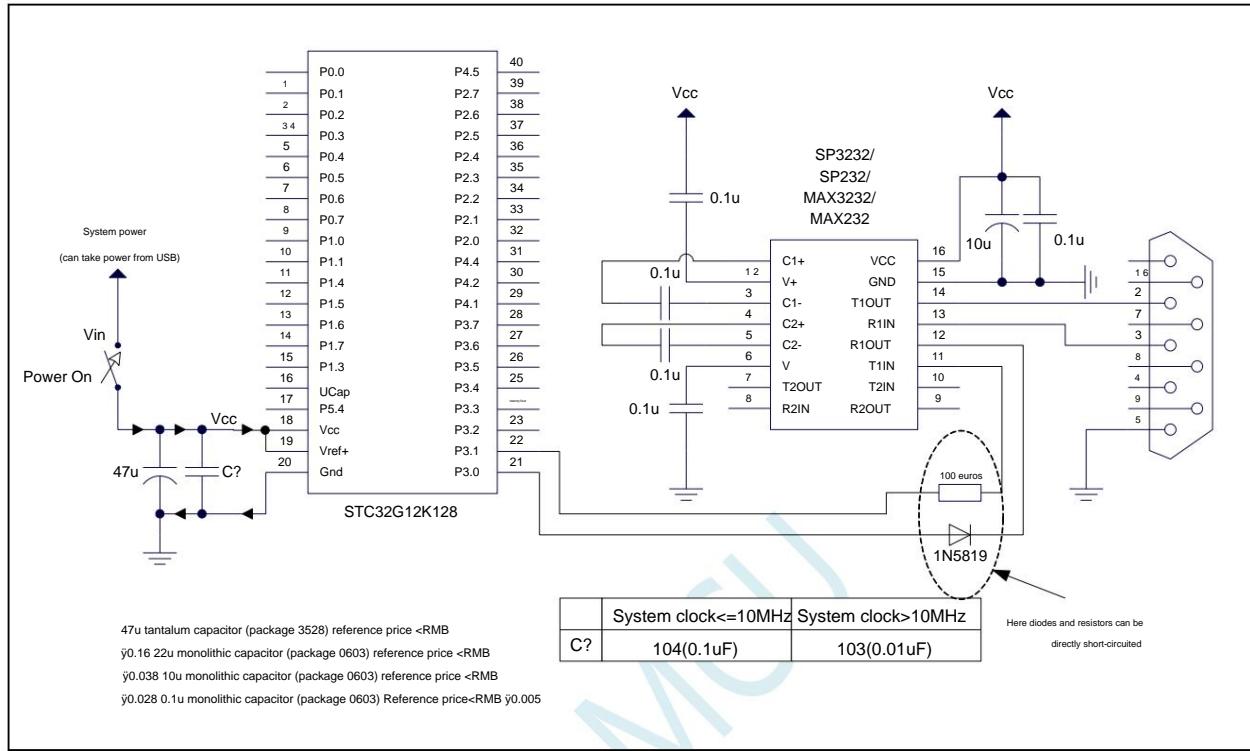
---

MCU working voltage  $V_{cc} = 4096 \times$  \_\_\_\_\_  
12-bit ADC conversion result

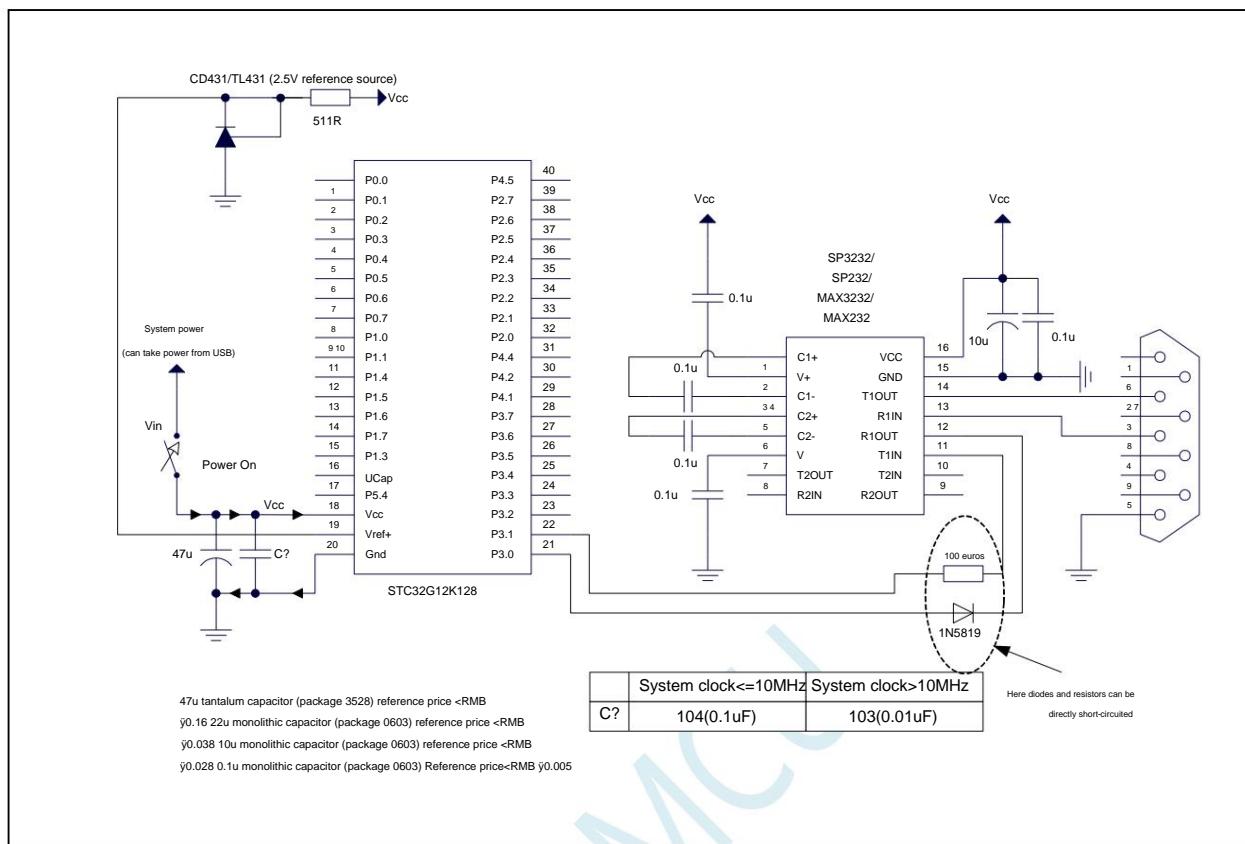
STCMCU

## 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 Example Program

#### 19.5.1 ADC Basic Operation (Query Mode)

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

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

    //Enable access XFR
    //to set the external data bus speed to the fastest.
    //Set the program code to wait for the parameter,
    //assign it to 0      CPU The speed of executing the program is set to the fastest
}

```

```

P4M0 = 0x00;
P4M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

P1M0 = 0x00;                                // set up P1.0 for ADC_mouth
P1M1 = 0x01;

ADCTIM = 0x3f;                               // set ADC Internal timing
ADCCFG = 0x0f;                               // set ADC The clock is the system /2/16/16
ADC_POWER = 1;                                // enable ADC clock module

while (1)
{
    ADC_START = 1;                            // start up AD convert
    _nop_();
    _nop_();
    while (!ADC_FLAG);                      // query ADC flag
    ADC_FLAG = 0; P2 = ADC_RES;              // Clear completion sign
                                            // read ADC
}

```

---

## 19.5.2 ADC Basic Operation (Interrupt Mode)

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                           // For header files, see Download Software
#include "intrins.h"

void ADC_Isr() interrupt 5 {
    ADC_FLAG = 0;                            clear interrupt flag
    P2 = ADC_RES;                           read ADC
    ADC_START = 1;                          ///// continue to convert
}

void main()
{
    EAXFR = 1;                             Enable XFR
    CKCON = 0x00;                           access to set external data bus speed to fastest
    WTST = 0x00;                            Set the program code to wait for parameters,
                                            ///// assign to CPU The speed of executing the program is set to the fastest

    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 up P1.0 for ADC mouth
P1M1 = 0x01;

ADCTIM = 0x3f;                               // set ADC Internal timing
ADCCFG = 0x0f;                               // set ADC The clock is the system /2/16/16
ADC_POWER = 1;                                // enable ADC clock module
EADC = 1;                                    // ADC interrupt
EA = 1;                                      // start up AD convert
ADC_START = 1;

while (1);
}

```

### 19.5.3 Formatting ADC Conversion Results

---

//The test frequency is 11.0592MHz

```

##include "stc8h.h"
#include "stc32g.h"                           // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    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 up P1.0 for ADC mouth
    P1M1 = 0x01;

    ADCTIM = 0x3f;                               // set ADC Internal timing
    ADCCFG = 0x0f;                               // set ADC The clock is the system /2/16/16
    ADC_POWER = 1;                                // enable ADC clock module
    ADC_START = 1;                                // start AD convert
    _nop_();
    _nop_();
    while (!ADC_FLAG);                          // query ADC completion flag
    ADC_FLAG = 0;                                // clear completion sign

    ADCCFG = 0x00;                                // Set the result to left-align
}

```



```

ACC = ADC_RES;
B = ADC_RESL;

// ADCCFG = 0x20;
// ACC = ADC_RES;
// B = ADC_RESL;

while (1);

}

```

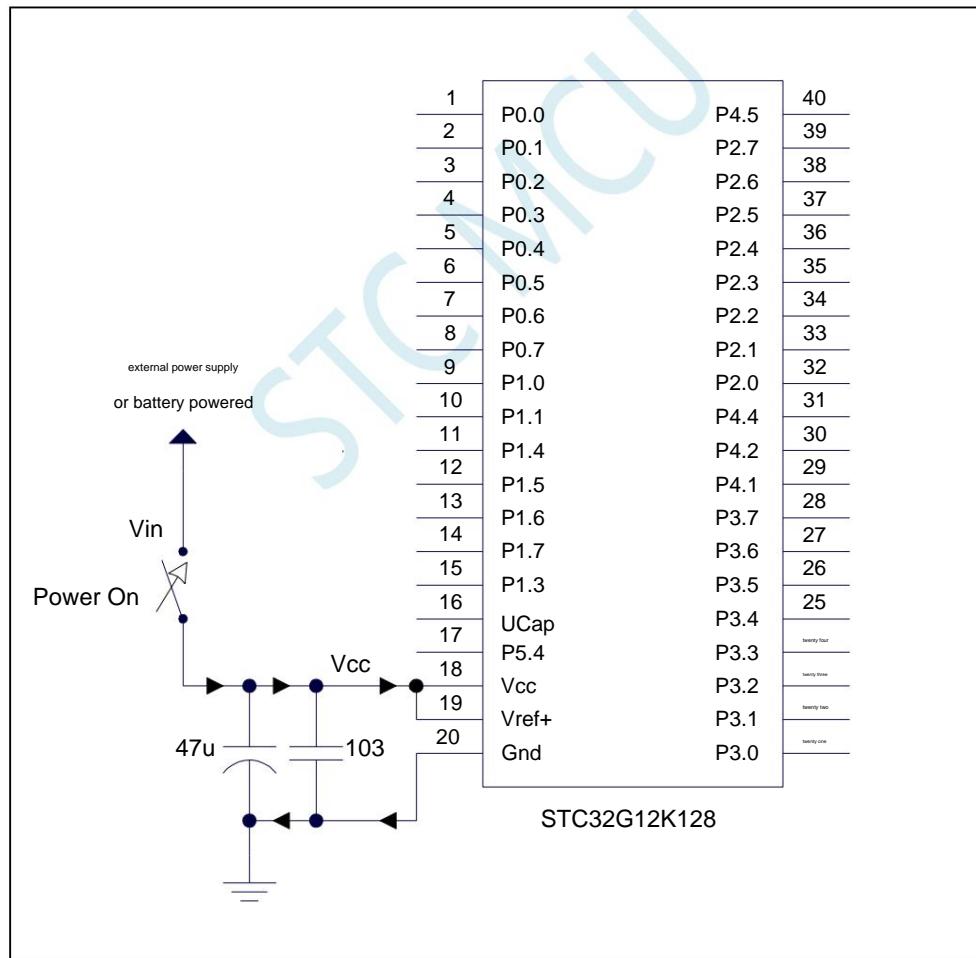
// A storage ADC of 12 high bit of the result 8  
 // B[7:4] storage ADC low 12 the bit result 4 bit, B[3:0] for 0  
 // Set the result to right-align  
 // A[3:0] Store the upper bits of the result and store the ADC of 12 4 bit, A[7:4] for 0  
 // B lower bits of the result ADC of 12 8

#### 19.5.4 Use ADC channel 15 (internal 1.19V reference signal source) to measure external power

voltage or battery voltage

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 it will not change with the change of the working voltage of the chip, so you can measure the internal 1.19V reference signal source, and then pass the ADC's value to invert the external voltage or external battery voltage.

The following figure is a reference circuit diagram:



//The test frequency is 11.0592MHz

```

##include "stc8h.h"
#include "stc32g.h"
#include "intris.h"

```

// For header files, see Download Software

```

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

#define VREFH_ADDR CHIPID07
#define VREFL_ADDR     CHIPID08

bit      busy;

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 up ADC Internal timing

    ADCCFG = 0x2f;                                 //set ADC The clock is the system clock/2/16
    ADC_CONTR = 0x8f;                             //enable ADC module and select the first 15 aisle
}

int ADCRead()
{
    int res;

    ADC_START = 1;                                //start up AD convert
    _nop_();
    _nop_();
    while (!ADC_FLAG);                           //query completion flag
    ADC_FLAG = 0;                                //Clear completion sign

    res = (ADC_RES << 8) | ADC_RESL;           //read ADC result
}

```

```

    return res;
}

void main()
{
    int res;
    int vcc;
    int i;

    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                            //assign to      CPU The speed of executing the program is set to the fastest

    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;           //from CHIPID   Read the internal reference voltage value in

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

    ES = 1;
    EA = 1;

//ADCRead();
//ADCRead();                                     //The first two data are recommended to be discarded

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

    vcc = (int)(4096L * BGV / res);             //Bit arithmetic calculation)      VREF   The pin voltage is the battery voltage
                                                //12//      (12 bit output digital signal level serial port (mV)
    UartSend(vcc >> 8);
    UartSend(vcc);

    while (1);
}

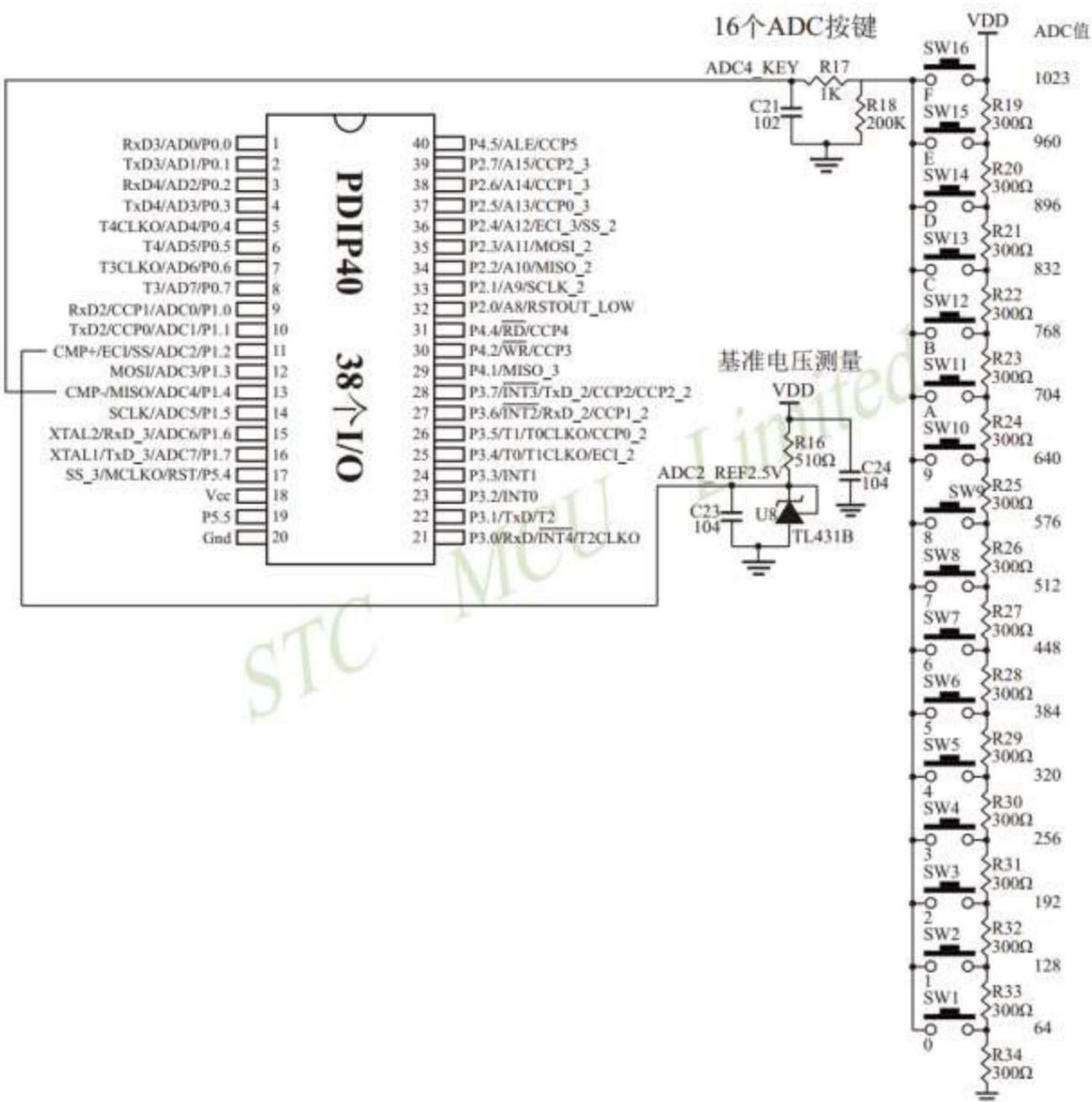
```

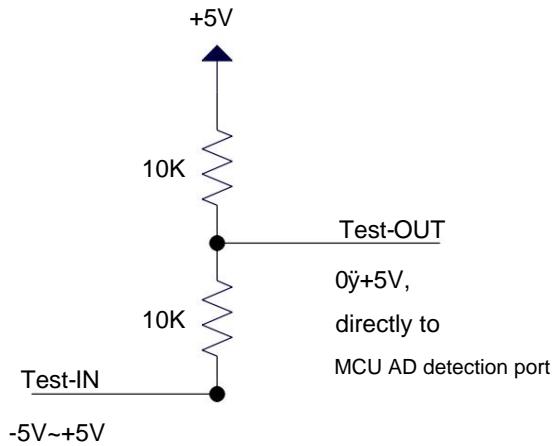
The above method uses the 15th channel of the ADC to invert the external battery voltage. Within the ADC measurement range, the external measurement of the ADC voltage is proportional to the measured value of the ADC, so the 15th channel of the ADC can also be used to invert the input voltage of the external channel, assuming currently, the internal reference signal source voltage is BGV, the ADC measurement value of the internal reference signal source is resbg, and the external channel input voltage is The ADC measurement value is resx, then the external channel input voltage  $V_x = BGV / resbg \cdot resx;$

## 19.5.5 ADC as key scan application circuit diagram

The method of reading the ADC key: read the ADC value every 10ms or so, and save the last 3 readings, and then judge the key when the change is relatively small.

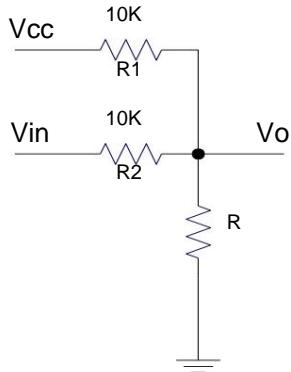
When the judgment key is valid, a certain deviation is allowed, such as a deviation of  $\pm 16$  characters.



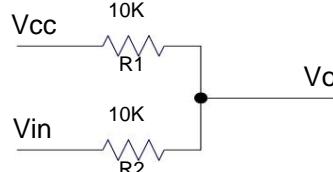
**19.5.6 Detecting Negative Voltage Reference Circuit Diagram**

Negative pressure conversion circuit

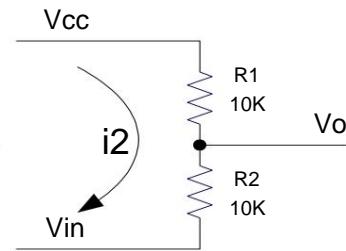
### 19.5.7 Application of Commonly Used Adding Circuits in ADCs



Commonly used adding circuits



Simplified addition circuit



Deformed into the form of a voltage divider circuit

Refer to the voltage divider circuit to get Equation 1

$$\text{Equation 1: } V_o = V_{in} + i_2 * R_2$$

$$\text{Equation 2: } i_2 = (V_{cc} - V_{in}) / (R_1 + R_2) \{ \text{Condition: Current to } V_o \neq 0 \}$$

Substitute  $R_1=R_2$  into Equation 2 to get Equation 3

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

Substitute Equation 3 into Equation 1 to get Equation 4

$$\text{Equation 4: } V_o = (V_{cc} + V_{in}) / 2$$

According to Equation 4, the above circuit can be regarded as an adding circuit.

In the analog-to-digital conversion measurement of the microcontroller, the measured voltage is required to be greater than 0 and less than VCC. If the measured voltage is less than 0V, you can use the

The method circuit will raise the measured voltage to above 0V. At this time, there are certain requirements for the variation range of the measured voltage:

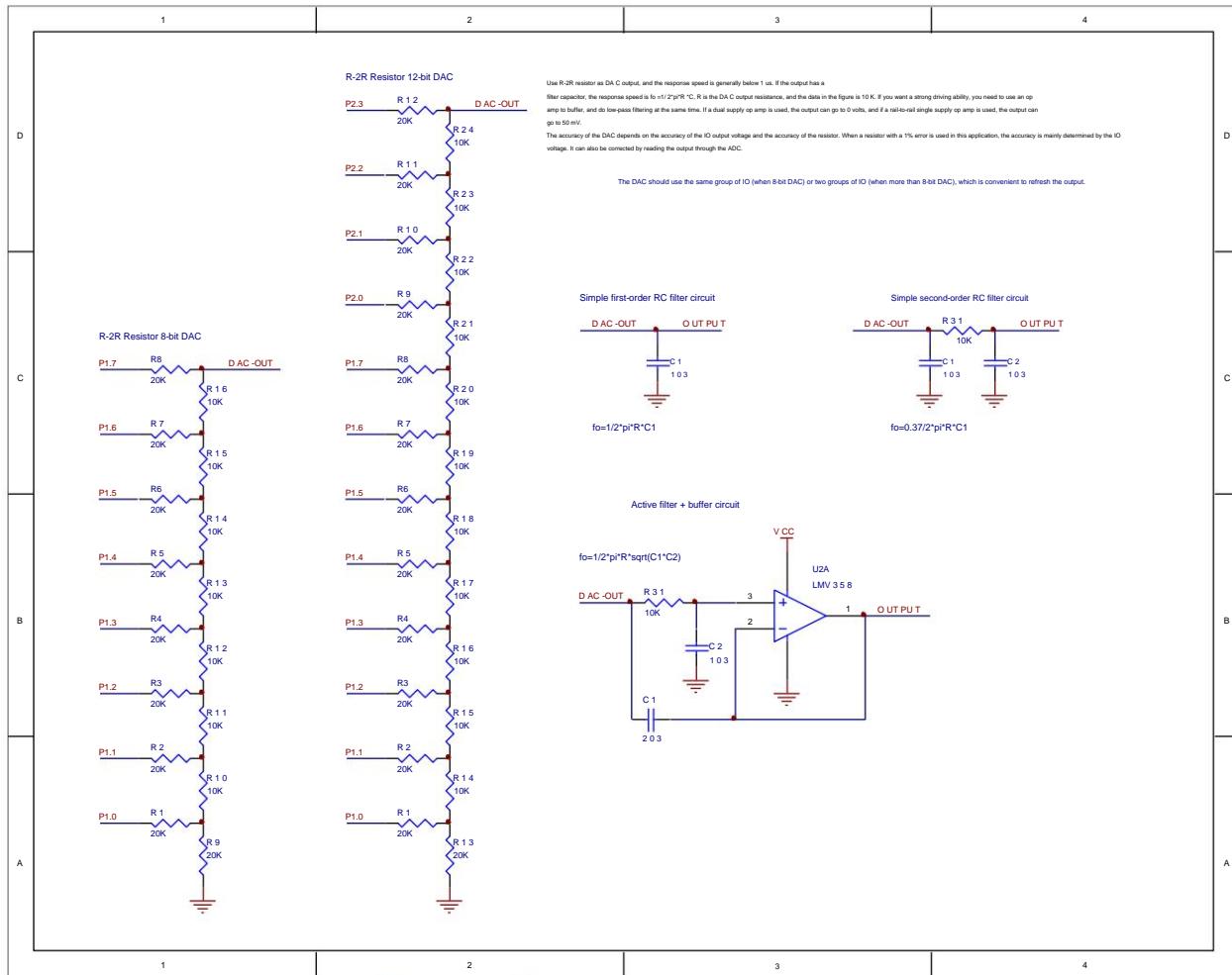
Substituting the above conditions into Equation 4, the following 2 equations can be obtained

$$(V_{cc} + V_{in}) / 2 > 0 \quad \text{i.e. } V_{in} > -V_{cc}$$

$$(V_{cc} + V_{in}) / 2 < V_{cc} \quad \text{i.e. } V_{in} < V_{cc}$$

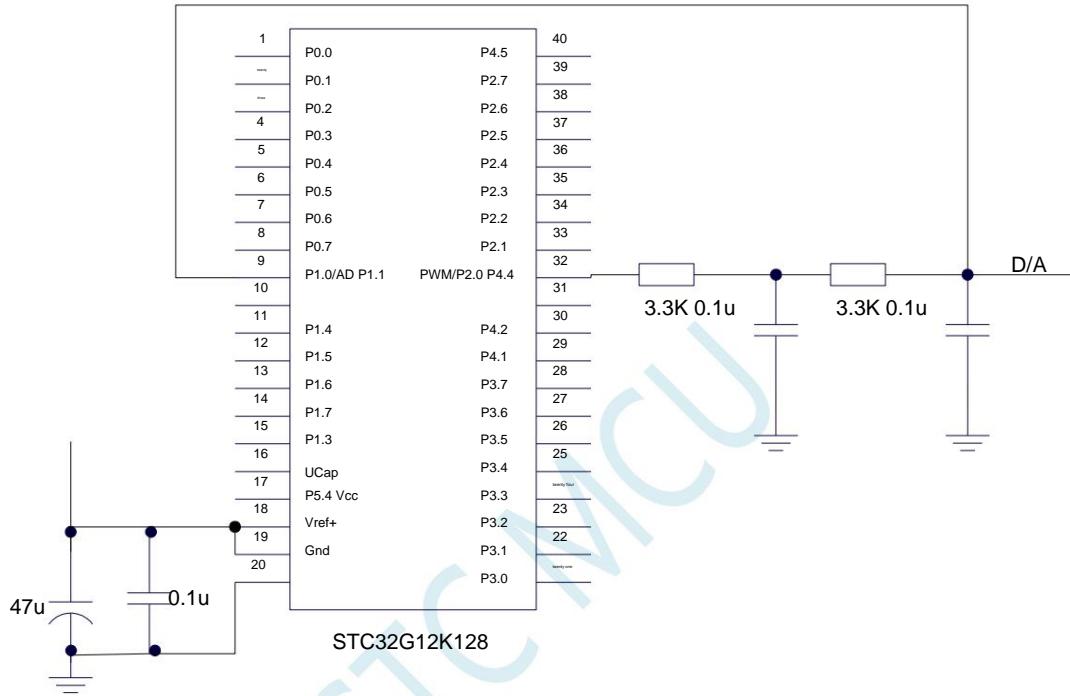
The above 2 equations can be combined: **-Vcc < Vin < Vcc**

## 19.6 Classical Circuit Diagram for DAC Using I/O and R-2R Resistor Divider



## 19.7 Reference circuit diagram for 16-bit DAC using PWM

The advanced PWM timer of the STC32G series microcontroller can output a 16-bit PWM waveform, and then through two-stage low-pass filtering, a 16-bit DAC signal can be generated, and the DAC signal can be changed by adjusting the high-level duty cycle of the PWM waveform. . The application circuit diagram is shown in the figure below, and the output DAC signal can be input to the ADC of the MCU for feedback measurement.



## 20 Synchronous Serial Peripheral Interface (SPI)

STC32G series microcontroller integrates a high-speed serial communication interface - SPI interface. SPI is a full-duplex high-speed synchronous communication bus. The integrated SPI interface of the STC32G series provides two operation modes: master mode and slave mode.

Note: The SPI mode of USART1 and USART2 can support two complete sets of SPI. For details, please refer to the USART chapter

### 20.1 SPI function pin switch

Symbol 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 selection bit

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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

Symbol 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 selection bit of 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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

S1SPI\_S[1:0]: SPI function pin selection bit of 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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

### 20.2 SPI related registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI Status Register	CDH SPIF	WCOL	-	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI Control Register	CEH SSIG	SPEN	DORD MSTR CPOL CPHA					SPR[1:0]		0000,0100

SPDAT	SPI data register	CFH							0000,0000
-------	-------------------	-----	--	--	--	--	--	--	-----------

## 20.2.1 SPI Status Register (SPSTAT)

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

SPIF: SPI interrupt flag bit.

When 1 byte of data is sent/received, the hardware will automatically set this bit to 1 and issue an interrupt request to the CPU. When the SSIG bit is set

When set to 0, when the master/slave mode of the device changes due to the change of the SS pin level, this flag will also be automatically set by the hardware.

Set to 1 to signal a device mode change.

Note: This flag bit must be cleared by the user by writing 1 to this bit through software.

WCOL: SPI write conflict flag.

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

Note: This flag bit must be cleared by the user by writing 1 to this bit through software.

## 20.2.2 SPI Control Register (SPCTL), SPI Speed Control

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

SSIG: SS pin function control bit

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

1: Ignore SS pin function, use MSTR to determine whether the device is a master or a slave

SPEN: SPI enable control bit

0: Disable SPI function

1: Enable SPI function

DORD: The order in which SPI data bits are sent/received

0: Send/receive the high-order bit (MSB) of the data first

1: Send/receive the low-order bit (LSB) of the data first

MSTR: Device Master/Slave Mode Select bit

Set host mode:

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

If SSIG=1, only need to set MSTR to 1 (ignore the level of SS pin)

Set slave mode:

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

If SSIG=1, only need to set MSTR to 0 (ignore the level of SS pin)

CPOL: SPI Clock Polarity Control

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

1: High level when SCLK is idle, the front clock edge of SCLK is a falling edge, and the rear clock edge is a rising edge

CPHA: SPI Clock Phase Control

0: The data SS pin drives the first bit of data at a low level and changes the data on the trailing edge of SCLK, and samples the data on the leading edge (must be SSIG = 0)

1: Data is driven on the leading edge of SCLK and sampled on the trailing edge

SPR[1:0]: SPI clock frequency selection

SPR[1:0] SCLK frequency
00 SYSclk/4

01	SYSclk/8
10	SYSclk/16
11	SYSclk/2

## 20.2.3 SPI Data Register (SPDAT)

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

SPI transmit/receive data buffer.

STCMCU

## 20.3 SPI communication method

There are usually three types of SPI communication methods: single master and single slave (one host device is connected to a slave device), mutual master and slave (two devices are connected, and each other are master and slave), single master and multiple slaves (one The master device is connected to multiple slave devices)

### 20.3.1 Single master and single slave

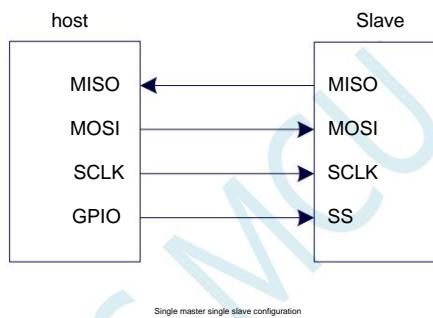
Two devices are connected, one of which is fixed as the master and the other is fixed as the slave.

Host settings: SSIG is set to 1, MSTR is set to 1, fixed to host mode. The master can use any port to connect to the slave's

SS pin, pull down the SS pin of the slave to enable the slave

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

The single-master single-slave connection configuration diagram is as follows:



### 20.3.2 Mutual master and slave

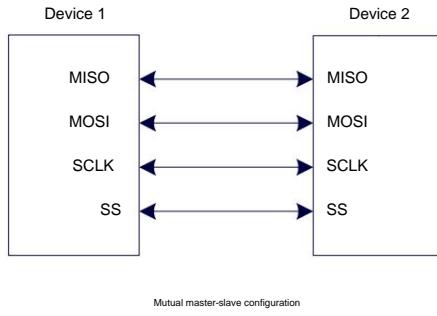
The two devices are connected, and the master and slave are not fixed.

Setting method 1: When both devices are initialized, set SSIG to 0, MSTR to 1, and set the SS pin to output high level in bidirectional port mode. At this point both devices are in host mode that does not ignore SS. When one of the devices needs to start transmission, it can set its own SS pin to output mode and output a low level, and pull down the other party's SS pin, so that the other device is forcibly set to slave mode.

Setting method 2: When both devices are initialized, they set themselves to ignore SS slave mode, that is, set SSIG to 1, MSTR

Set to 0. When one of the devices needs to start transmission, first check the level of the SS pin, if it is high, it will set itself to ignore the SS master mode, and then the data transmission can be performed.

The configuration diagram of the mutual master-slave connection is as follows:



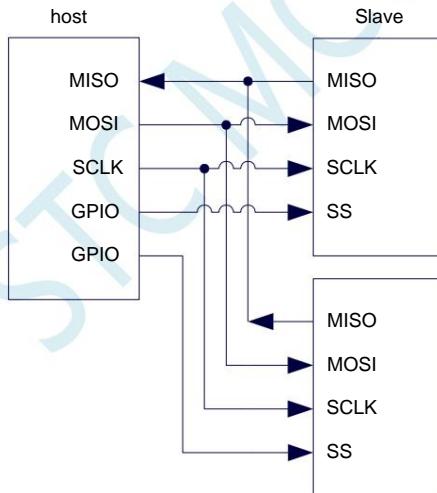
Mutual master-slave configuration

### 20.3.3 Single master and multiple slaves

Multiple devices are connected, one device is fixed as the master, and the other devices are fixed as slaves.

Host settings: SSIG is set to 1, MSTR is set to 1, fixed to host mode. The host can use any port to connect to each SS pins of two slaves, pull down the SS pin of one of the slaves to enable the corresponding slave device slave settings: SSIG is set to 0, and the SS pin is used as the chip select signal of the slave.

The single-master multi-slave connection configuration diagram is as follows:



Single-master multi-slave configuration

## 20.4 Configuring SPI

control bit			communication port			illustrate	
SPEN	SSIG	MSTR	SS	MISO	MOSI	SCLK	
0	x	x	x	input	input	input	close SPI function, SS/MOSI/MISO/SCLK are common IO
1	0	0	0	output	input	input	slave mode, and is selected
1	0	0	1	Hi-Z input	in slave mode,	but not selected	
1	0	1	0	1	0	output	Slave mode, master mode where SS is not ignored and MSTR is 1, When the SS pin is pulled low, MSTR will be automatically cleared by hardware, The working mode will be passively set to slave mode
1	0	1	1	input	Hi-Z	master mode, idle state	
1	1	0	x	output	input	input	slave mode
1	1	1	x	I/O	master mode		

### Notes on slave mode:

When CPHA=0, SSIG must be 0 (ie SS pin cannot be ignored). The SS pin must be pulled before each serial byte is sent low and must be reset high after the serial byte has been sent. The SPDAT register cannot be written while the SS pin is low operation, otherwise a write conflict error will result. Action when CPHA=0 and SSIG=1 is undefined.

When CPHA=1, SSIG can be set to 1 (ie the pin can be ignored). If SSIG=0, the SS pin can be held low between consecutive transfers Active (ie, always fixed low). This method is suitable for systems with fixed single master and single slave.

### Notes on host mode:

In SPI, transfers are always initiated by the host. If the SPI is enabled (SPEN=1) and selected as the master, the master will A write to register SPDAT will start the SPI clock generator and data transfer. Half to one SPI after data is written to SPDAT After the bit time, the data will appear on the MOSI pin. Data written to the master's SPDAT register is shifted from the MOSI pin to the slave's MOSI foot. At the same time, the data in the SPDAT register of the slave is shifted from the MISO pin to the MISO pin of the master.

After a byte is transferred, the SPI clock generator is stopped, the transfer complete flag (SPIF) is set, and if the SPI interrupt is enabled, it will be generated. Generate an SPI interrupt. The two shift registers of the master and slave CPUs can be viewed as a 16-bit circular shift register. when data from When the master is shifted to the slave, the data is also shifted in in the opposite direction. This means that in one shift cycle, the number of master and slave exchanged with each other.

### Change mode via SS

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

User software must always check the MSTR bit if the bit is passively cleared by a slave select action and the user wants to continue To use the SPI as a master, the MSTR bit must be reset, otherwise it will remain in slave mode.

write conflict

SPI is single-buffered when transmitting and double-buffered when receiving. In this way, new data cannot be written before the previous transmission has not been completed into the shift register. When the data register SPDAT is written during a transmission, the WCOL bit will be set to 1 to indicate that data has occurred.

Write conflict error. In this case, the currently sent data continues to be sent, and the newly written data will be lost.

When writing collision detection for master or slave, it is rare for the master to have a write collision because the master has the full control. But it is possible for the slave to have a write conflict because when the master initiates the transfer, the slave has no control.

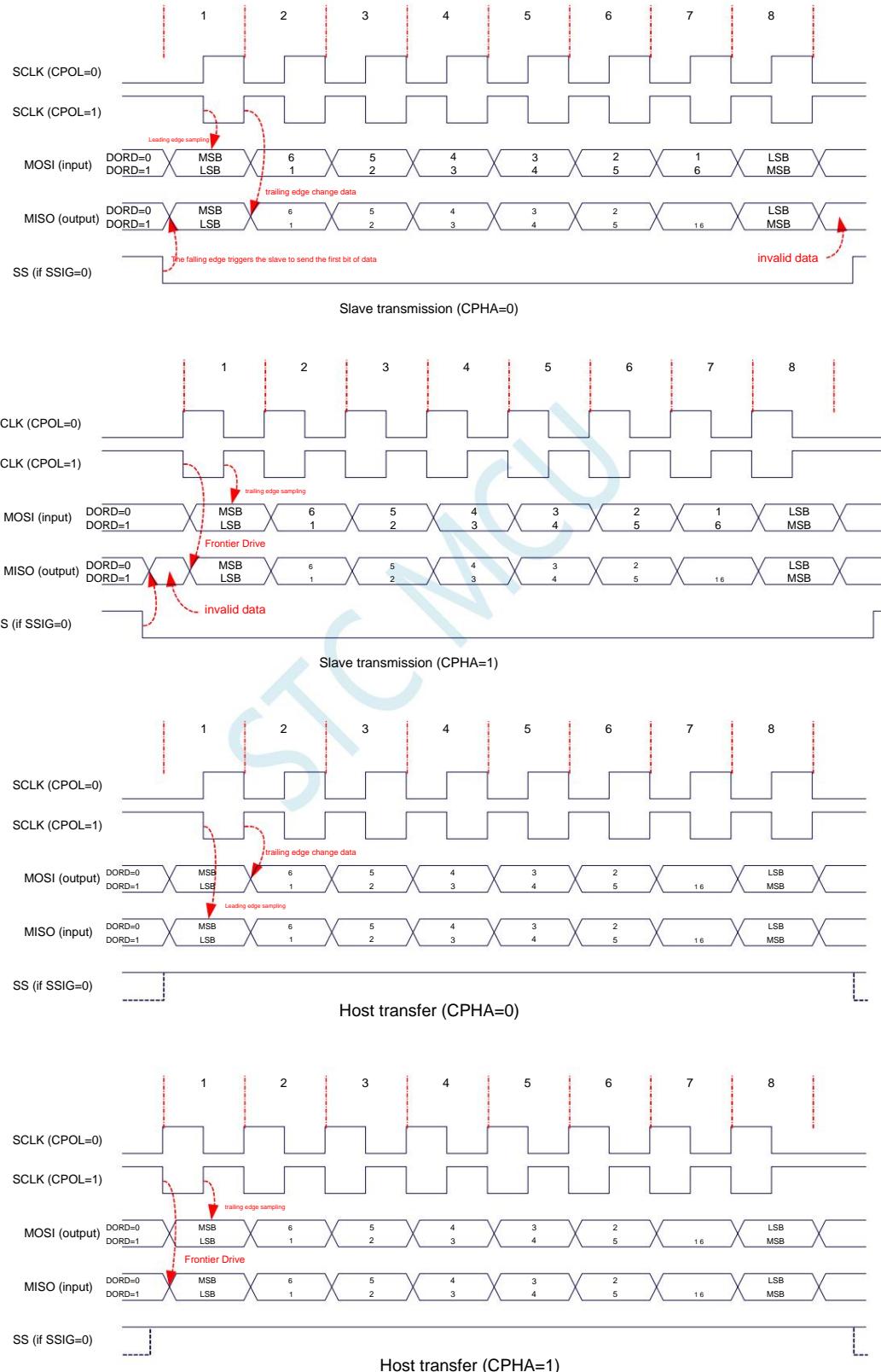
When data is received, the received data is transferred to a parallel read data buffer, which frees the shift register for the next count. receipt of data. But the received data must be read from the data register before the next character is completely shifted in, otherwise, the previous received data will be lost.

WCOL can be cleared by software by writing a '1' to it.

## 20.5 Data Mode

The SPI's clock phase control bit CPHA allows the user to set the clock edge at which data is sampled and changed. The clock polarity bit CPOL can

Let the user set the clock polarity. The following illustration shows the SPI communication timing under different clock phase and polarity settings.



## 20.6 Example Program

### 20.6.1 SPI single-master single-slave system host program (interrupt mode)

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#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 SS pin
    busy = 0; //test port
    LED = !LED;
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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 mode
    SPSTAT = 0xc0; //clear interrupt flag
    ESPI = 1; //enable SPI
    EA = 1;

    while (1)
    {
        while (busy);
    }
}
```

```

busy = 1;
SS = 0;
SPDAT = 0x5a;
    //Pull down the SS pin
    //Send test data
}
}

```

---

## 20.6.2 SPI single master single slave system slave program (interrupt mode)

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                     //For header files, see Download Software
#include "intrins.h"

sbit led      = P1^1;

void SPI_Isr() interrupt 9 {
    SPIF = 1;                                         //clear interrupt flag
    SPDAT = SPDAT;                                    //Send the received data back to the host
    LED = !LED;                                       //test port
}

void main()
{
    EAXFR = 1;                                         //Enable      XFR
    CKCON = 0x00;                                      //access to set external data bus speed to fastest
    WTST = 0x00;                                       //Set the program code to wait for parameters,
                                                       //assign to      CPU   The speed of executing the program is set to the fastest

    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 mode
    SPSTAT = 0xc0;                                      //clear interrupt flag
    ESPI = 1;                                           //enable SPI
    EA = 1;

    while (1);
}

```

---

### 20.6.3 SPI single-master single-slave system host program (query method)

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

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

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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 mode
    SPSTAT = 0xc0; //clear interrupt flag

    while (1)
    {
        SS = 0; SPDAT = 0xa5; while (!SPIF); SPIF = 1; SS = 1; LED = !LED;
        //Pull down the slave pin
        //Send test data
        //query completion flag
        //clear interrupt flag
        //Pull up the test SS pin
        //port of the slave
    }
}

```

---

### 20.6.4 SPI single master single slave system slave program (query method)

---

//The test frequency is 11.0592MHz

```
//#include "stc8h.h"
```

```

#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

sbit led      = P1^1;

void main()
{
    EAXFR = 1;                                     //Enable XFR
    CKCON = 0x00;                                   //access to set external data bus speed to fastest
    WTST = 0x00;                                    //Set the program code to wait for parameters,
                                                    //assign to CPU The speed of executing the program is set to the fastest

    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 mode
    SPSTAT = 0xc0;                                   //clear interrupt flag

    while (1)
    {
        while (!SPIF);                            //query completion flag
        SPIF = 1;                                 //clear interrupt flag
        SPDAT = SPDAT;                           //Send the received data back to the host
        LED = !LED;                               //test port
    }
}

```

## 20.6.5 SPI mutual master-slave system program (interrupt mode)

---

```

//The test frequency is 11.0592MHz

##include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#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 SS pin
    }
}

```

```

        SPCTL = 0x40;                                //Reset to slave standby
    }
    else
    {
        SPDAT = SPDAT;                            //slave mode
        //Send the received data back to the host
    }
    LED = !LED;                                  //test port
}

void main()
{
    EAXFR = 1;          //enable access XFR
    CKCON = 0x00;      //Set the external data bus speed to the fastest
    WTST = 0x00;       //Set the program code to wait for parameters,
                      //assign to CPU The speed of executing the program is set to the fastest

    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 Mode for Standby
    SPSTAT = 0xc0;                                //clear interrupt flag
    ESPI = 1;                                     //enable SPI
    EA = 1;

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

```

## 20.6.6 SPI mutual master-slave system program (query method)

//The test frequency is 11.0592MHz

//#include "stc8h.h"

```

#include "stc32g.h"
#include "intrins.h"

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

void main()
{
    EAXFR = 1;      //enable access XFR
    CKCON = 0x00;
    WTST = 0x00;
    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;
    SPSTAT = 0xc0;

    while (1)
    {
        if (!KEY)
        {
            SPCTL = 0x50;
            SS = 0; SPDAT
            = 0x5a; while (!
            KEY);
        }
        if (SPIF)
        {
            SPIF = 1;           //clear interrupt flag
            if (SPCTL & 0x10)
            {
                SS = 1;           //host mode
                SPCTL = 0x40;     //Pull up the pin of the SS
                //Reset to slave standby
            }
            else
            {
                SPDAT = SPDAT;   //slave mode
                //Send the received data back to the host
            }
            LED = !LED;         //test port
        }
    }
}

```

// For header files, see Download Software

// Set the external data bus speed to the fastest

// Set the program code to wait for parameters,

// assign to **CPU** The speed of executing the program is set to the fastest

//Enable SPI Mode for Standby

//clear interrupt flag

//wait for a key to trigger

//Enable SPI mode

//Pull down the **SS** pin

//Send test data

//Wait for the key to be released

//clear interrupt flag

//host mode

//Pull up the pin of the **SS**

//Reset to slave standby

//slave mode

//Send the received data back to the host

//test port

STCMCU

# 21 High Speed SPI (HSSPI)

STC32G series microcontrollers provide high-speed mode (HSPSI) for SPI. High-speed SPI is based on ordinary SPI, adding high-speed model.

When the system runs at a lower operating frequency, the high-speed SPI can work at frequencies up to 144M. thereby reducing the kernel power

The purpose of improving the performance of peripherals

## 21.1 Related registers

symbol	describe	address	Bit addresses and symbols								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	TXEEMPT	RXFULL	RXEEMPT xxxx	0000

### 21.1.1 High Speed SPI Configuration Register (HSSPI\_CFG)

Symbol 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)

Symbol 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: Disable high-speed mode.

1: Enable high speed mode.

When the SPI speed is the system clock/2 (SPCTL.SPR=3), the high-speed mode must be enabled

FIFOEN: FIFO mode enable bit for high-speed SPI

0: Disable FIFO mode.

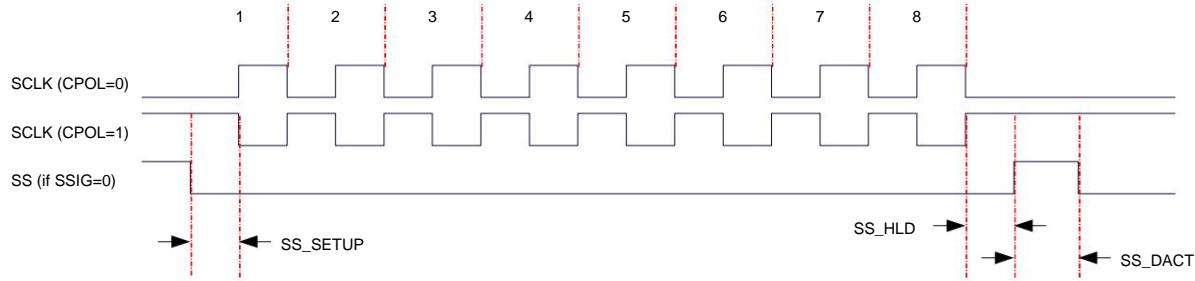
1: Enable FIFO mode.

It is meaningful only when the SPI is in DMA operation, and the normal SPI mode is meaningless. SPI transmit and receive FIFO depth is 4 byte.

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 meaningful only in DMA in SPI master mode, only in SPI

In host mode, the hardware automatically outputs the SS control signal only when DMA data transfer is performed. When 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)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
HSSPI_STA 7EFBFAH	-	-	-	-	TXFULL TXEMPT RXFULL RXEMPT			

TXFULL: transmit data FIFO full flag (read only)

TXEMPT: Transmit Data FIFO Empty Flag (read only)

RXFULL: Receive data FIFO full flag (read only)

RXEMPT: Receive data FIFO empty flag (read only)

## 21.2 Example program

### 21.2.1 Enabling high-speed mode of SPI

---

//The test frequency is 12MHz

```

//#include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#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 XFR
    CKCON = 0x00;// Set the external data bus speed to the fastest
    WTST = 0x00;

    //choosePLL output clock
}

```

```

#ifndef (PLL_SEL == PLL_96M)
    CLKSEL &= ~CKMS;
#endif (PLL_SEL == PLL_144M)
    CLKSEL |= CKMS;
#else
    CLKSEL &= ~CKMS;
#endif

//choosePLL The input clock frequency division ensures that the input clock is 12M
USBCLK &= ~PCKI_MSK;
#if (FOSC == 12000000UL)
    USBCLK |= PCKI_D1;
#elif (FOSC == 24000000UL)
    USBCLK |= PCKI_D2;
#elif (FOSC == 48000000UL)
    USBCLK |= PCKI_D4;
#elif (FOSC == 96000000UL)
    USBCLK |= PCKI_D8;
#else
    USBCLK |= PCKI_D1;
#endif

//start upPLL
USBCLK |= ENCKM;

delay();

//chooseHSPWM/HSSPI clock
#if (HSCK_SEL == HSCK_MCLK)
    CLKSEL &= ~HSIOCK;
#elif (HSCK_SEL == HSCK_PLL)
    CLKSEL |= HSIOCK;
#else
    CLKSEL &= ~HSIOCK;
#endif

HSCLKDIV = 0;

SPCTL = 0xd0;
HSSPI_CFG2 |= 0x20;

P1M0 = 0x28;
P1M1 = 0x00;

while (1)
{
    SPSTAT = 0xc0;
    SPDAT = 0x5a;
    while (!SPIF);
}

```

*STCMCUDATA*

---

//choosePLL of 96M as PLL the output clock of

//choosePLL of 144M as PLL the output clock of

//Default selectionPLL of 96M as PLL the output clock of

//choosePLL The input clock 1 frequency division

//PLL input clock 2 frequency division

//PLL input clock 4 frequency division

//PLL input clock 8 frequency division

//defaultPLL input clock 1 frequency division

//EnablePLL frequency doubling

//wait PLL frequency lock

//HSPWM/HSSPI Select the main clock as the clock source

//HSPWM/HSSPI choosePLL The output clock is the clock source

//defaultHSPWM/HSSPI Select the main clock as the clock source

//HSPWM/HSSPI The clock source is not divided

//set SPI for host mode speed for SPI clock/4

//enableSPI high speed mode

# 22 I2C bus

The microcontroller of the STC32G series integrates an I2C serial bus controller. I2C is a high-speed synchronous communication bus that enables communication Use SCL (clock line) and SDA (data line) for synchronous communication. For port assignment of SCL and SDA, the STC32G series

The single-chip microcomputer provides a switching mode, which can switch SCL and SDA to different I/O ports, so that users can use a group of I2C buses as multiple I/O ports. Groups are time-multiplexed.

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

- ÿ No arbitration after sending start signal (START)

- ÿ No timeout detection is performed when the clock signal (SCL) stays at a low level

The I2C bus of the STC32G series provides two operating modes: master mode (SCL is an output port, sending a synchronous clock signal) and Slave mode (SCL is the input port, receiving synchronous clock signal)

## 22.1 I2C function pin switching

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

I2C\_S[1:0]: I2C function pin selection bit

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

## 22.2 I2C related registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
I2CCFG	I2C Configuration Register	7EFE80H ENI2C	MSSL				MSSPEED[5:0]				0000,0000
I2CMSCR	I2C host control register	7EFE81H EMSI		-	-	-			MSCMD[3:0]		0xxx,0000
I2CMSST	I2C Master Status Register	7EFE82H MSBUSY MSIF		-	-	-		-	-MSACKI	MSACKO	00xx,xx10
I2CSLCR	I2C slave control register	7EFE83H	-	ESTAI	ERXI	ETXI ESTOI		-	-	SLRST	x000,0xx0
I2CSLST	I2C Slave Status Register	7EFE84H SLBUSY STAIF			RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I2C slave address register	7EFE85H				SLADR[6:0]				MA	0000,0000
I2CTXD	I2C data transmission register	7EFE86H									0000,0000
I2CRXD	I2C data receive register	7EFE87H									0000,0000
I2CMSAUX	I2C Master Auxiliary Control Register	7EFE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

## 22.3 I2C master mode

### 22.3.1 I2C CONFIGURATION REGISTER (I2CCFG), BUS SPEED CONTROL

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	7FEF80H	ENI2C	MSSL				MSSPEED[5:0]

ENI2C: I2C function enable control bit

0: Disable I2C function

1: Enable I2C function

MSSL: I2C operating mode select bit

0: slave mode

1: host mode

MSSPEED[5:0]: I2C bus speed (number of wait clocks) control, I2C **bus speed = SYSCLK / 2 / (MSSPEED \* 2 + 4)**

(The fastest speed of I2C is **SYSCLK/8**)

The number of clocks corresponding to MSSPEED[5:0]	
0	4
1	6
2	8
...	...
x	2x+4
...	...
62	128
63	130

The wait parameter set by the MSSPEED parameter is valid only when the I2C module is operating in master mode. This wait parameter is mainly used for

The following signals of the host mode:

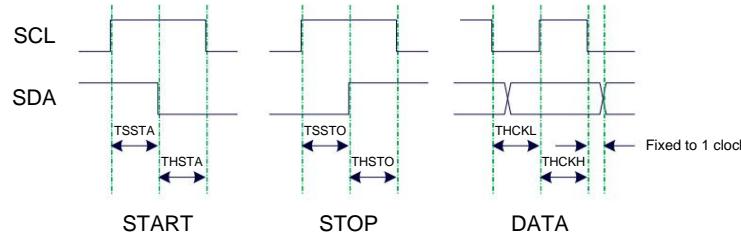
TSSTA: Setup Time of START

THSTA: Hold Time of START

TSSTO: Setup Time of STOP

THSTO: Hold Time of STOP

THCKL: Hold Time of SCL Low for the clock signal



Example 1: When **MSSPEED=10**, TSSTA=THSTA=TSSTO=THSTO=THCKL=24/FOSC

Example 2: When 400K I2C bus speed is required at 24MHz operating frequency ,

$$\text{MSSPEED}=(24\text{M} / 400\text{K} / 2 - 4) / 2 = 13$$

## 22.3.2 I2C MASTER CONTROL REGISTER (I2CMSCR)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
I2CMSCR	7EFE81H	EMSI	-	-	-	-	-	-	MSCMD[2:0]

EMSI: Host Mode Interrupt Enable Control Bit

0: Disable interrupts in host mode

1: Enable interrupts in master mode

MSCMD[3:0]: host command

0000: Standby, no action.

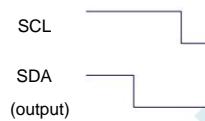
0001: Start command.

Send a START signal. If the current I2C controller is idle, that is, when MSBUSY (I2CMSST.7) is 0,

Writing this command will make the controller enter the busy state, the hardware will automatically set the MSBUSY state bit to 1, and start sending the START signal.

number; if the current I2C controller is in a busy state, writing this command can trigger the sending of a START signal. that sends the START signal

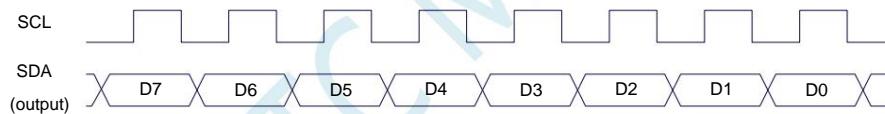
The waveform is shown in the following figure:



0010: Send data command.

After writing this command, the I2C bus controller will generate 8 clocks on the SCL pin and transfer the data in the I2CTXD register

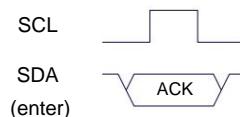
It is sent to the SDA pin by bit (the high-order data is sent first). The waveform of the transmitted data is shown in the following figure:



0011: Receive ACK command.

After writing this command, the I2C bus controller will generate 1 clock on the SCL pin and will read the data from the SDA port

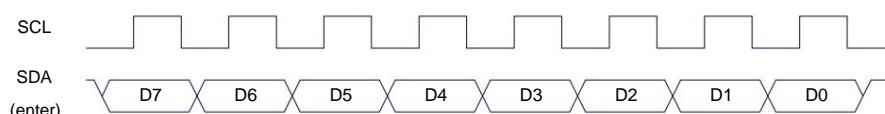
Save to MSACKI (I2CMSST.1). The waveform of receiving ACK is shown in the following figure:



0100: Receive data command.

After writing this command, the I2C bus controller will generate 8 clocks on the SCL pin and will read the data from the SDA port

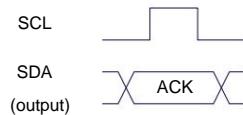
Shift left to the I2CRXD register in turn (receive high-order data first). The waveform of the received data is shown in the following figure:



0101: Send ACK command.

After writing this command, the I2C bus controller will generate 1 clock on the SCL pin and send MSACKO (I2CMSST.0)

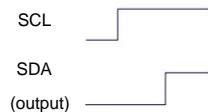
data is sent to the SDA port. The waveform of sending ACK is shown in the following figure:



0110: Stop command.

Send a STOP signal. After writing this command, the I2C bus controller starts sending STOP signals. After the signal is sent, the hardware

Automatically clears the MSBUSY status bit. The waveform of the STOP signal is shown in the following figure:



0111: Reserved.

1000: Reserved.

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

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

1010: Send data command + receive ACK command.

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

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

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

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

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

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

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

### 22.3.3 I2C Master Auxiliary Control Register (I2CMSAUX)

Symbol address	B7		B6	B5	B4 B3		B2	B1	B0
I2CMSAUX 7EFE88H		-	-	-	-	-	-	-	WDTA

WDTA: I2C data automatic transmission enable bit in master mode

0: Disable automatic sending

1: Enable automatic transmission

If the automatic transmission function is enabled, when the MCU completes the write operation to the I2CTXD data register, the I2C controller will automatically trigger the

Send "1010" command, that is, automatically send data and receive ACK signal.

### 22.3.4 I2C MASTER STATUS REGISTER (I2CMSST)

Symbol address	B7		B6	B5	B4 B3		B2	B1	B0
I2CMSST 7EFE82H	MSBUSY	MSIF		-	-	-	-	-	MSACKI MSACKO

MSBUSY: I2C controller status bit in master mode (read-only bit)

0: The controller is idle

1: The controller is busy

When the I2C controller is in the master mode, in the idle state, after sending the START signal, the controller enters the busy state,

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: Master mode interrupt request bit (interrupt flag bit). When the I2C controller in master mode executes the complete register I2CMSCR

After the MSCMD command, an interrupt signal is generated, the hardware automatically sets this bit to 1, and sends an interrupt request to the CPU. After responding to the interrupt, the MSIF bit must be set to 1.

It must be cleared by software.

MSACKI: In master mode, the ACK data received after sending "0011" command to the MSCMD bit of I2CMSCR. (read only bits)

MSACKO: In master mode, prepare the ACK signal to be sent out. When sending "0101" command to MSCMD of I2CMSCR

After the bit, the controller will automatically read the data of this bit and send it to SDA as ACK.

## 22.4 I2C slave mode

### 22.4.1 I2C Slave Control Register (I2CSLCR)

Symbol address B7	B6	B5	B4 B3	B2	B1	B0		
I2CSLCR	7EFE83H	-	ESTAI	ERXI	ETXI ESTOI	-	-	SLRST

ESTAI: START signal received interrupt enable bit in slave mode

0: Interrupt occurs when the START signal is received when the slave mode is disabled

1: Interrupt occurs when START signal is received when slave mode is enabled

ERXI: Interrupt enable bit after receiving 1 byte of data in slave mode

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

1: Interrupt occurs after 1 byte of data is received when slave mode is enabled

ETXI: Interrupt enable bit after sending 1 byte of data in slave mode

0: Interrupt occurs after sending completion data when slave mode is disabled

1: When the slave mode is enabled, an interrupt occurs after the completion of 1 byte of data transmission

ESTOI: STOP signal received in slave mode interrupt enable bit

0: Interrupt occurs when STOP signal is received when slave mode is disabled

1: Interrupt occurs when STOP signal is received when slave mode is enabled

SLRST: Reset Slave Mode

### 22.4.2 I2C Slave Status Register (I2CSLST)

Symbol address B7	B6	B5	B4 B3	B2	B1	B0
I2CSLST	7EFE84H SLBUSY	STAIF	RXIF	TXIF STOIF	-	SLACKI SLACKO

SLBUSY: I2C controller status bit in slave mode (read-only bit)

0: The controller is idle

1: The controller is busy

When the I2C controller is in slave mode, in the idle state, after receiving the START signal sent by the master, the controller will continue to detect

After the device address data, if the device address matches the slave address image set in the current I2CSLADR register, the control

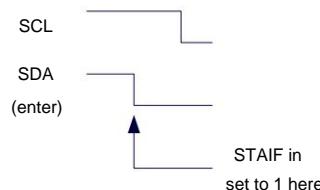
The controller enters the busy state, and the busy state will be maintained until the STOP signal sent by the host is successfully received, and then the state will resume.

return to idle state.

STAIF: Interrupt request bit after receiving START signal in slave mode. After the I2C controller in slave mode receives the START signal,

The hardware will automatically set this bit to 1 and send an interrupt request to the CPU. After responding to the interrupt, the STAIF bit must be cleared by software. STAIF is set.

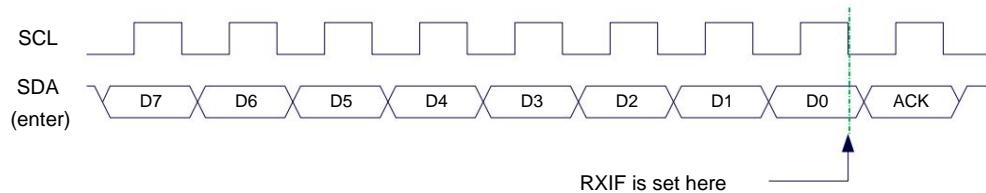
The time point of 1 is shown in the following figure:



RXIF: Interrupt request bit after receiving 1 byte of data in slave mode. After the I2C controller in slave mode receives 1 byte of data,

At the falling edge of the 8th clock, the hardware will automatically set this bit to 1 and send a request to the CPU to interrupt. After responding to the interrupt, the RXIF bit must be set.

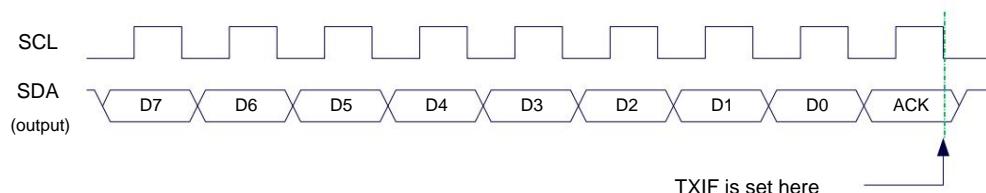
Cleared by software. The time point when RXIF is set to 1 is shown in the following figure:



**TXIF:** Interrupt request bit after sending 1 byte of data in slave mode. The I2C controller in slave mode sends a complete 1-byte count

After receiving the 1-bit ACK signal successfully, the hardware will automatically set this bit to 1 at the falling edge of the ninth clock, and send a message to the CPU.

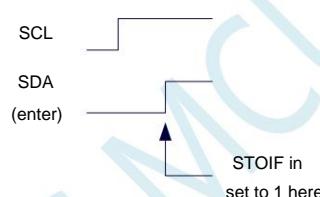
An interrupt is requested, and the TXIF bit must be cleared in software after the interrupt is serviced. The time point when TXIF is set to 1 is shown in the following figure:



**STOIF:** Interrupt request bit after receiving STOP signal in slave mode. After the I2C controller in slave mode receives the STOP signal, the hard

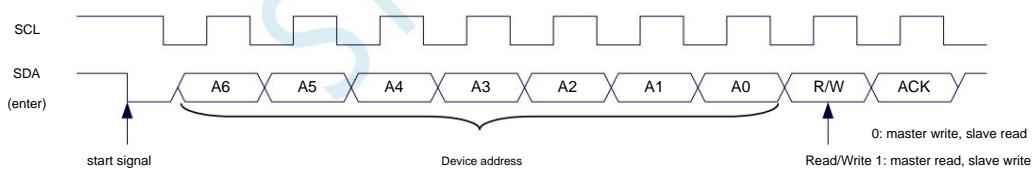
The software will automatically set this bit to 1 and send an interrupt request to the CPU. After responding to the interrupt, the STOIF bit must be cleared by software. STOIF is set to 1

The time points are shown in the following figure:



**SLACKI:** ACK data received in slave mode.

**SLACKO:** In slave mode, prepare the ACK signal to be sent out.



### 22.4.3 I2C Slave Address Register (I2CSLADR)

Symbol address B7		B6	B5	B4 B3		B2	B1	B0
I2CSLADR 7EFE85H	SLADR[6:0]							MA

SLADR[6:0]: slave device address

When the I2C controller is in slave mode, after receiving the START signal, the controller will continue to detect the next device sent by the host.

address data and read/write signals. When the device address sent by the host is the same as the slave device address set in SLADR[6:0]

When matching, the controller will send an interrupt request to the CPU to request the CPU to process the I2C event; otherwise, if the device address does not match, the I2C

The controller continues to monitor, waits for the next start signal, and continues to match the next device address.

MA: slave device address matching control

0: Device address must continue to match SLADR[6:0]

1: Ignore the settings in SLADR and match all device addresses

## 22.4.4 I2C Data Registers (I2CTXD, I2CRXD)

Symbol	address B7		B6	B5	B4 B3	B2	B1	B0
I2CTXD	7EFE86H							
I2CRXD	7EFE87H							

I2CTXD is the I2C transmit data register, which stores the I2C data to be transmitted

I2CRXD is the I2C receive data register, which stores the received I2C data

## 22.5 Example Program

### 22.5.1 I2C host mode access AT24C256 (interrupt mode)

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#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 Order
    while (busy);
}

void SendData(char dat)
{
    I2CTXD = dat; //write data to data buffer
    busy = 1;
    I2CMSCR = 0x82; //send SEND Order
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83; //send readACK Order
    while (busy);
}

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

void SendACK()
```

```

{
    I2CMSST = 0x00;                                //set upACK Signal
    busy = 1;
    I2CMSCR = 0x85;                                //send ACK Order
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                                //set upNAK Signal
    busy = 1;
    I2CMSCR = 0x85;                                //send ACK Order
    while (busy);
}

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

void Delay()
{
    int i;

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

void main()
{
    EAXFR = 1;                                     //Enable XFR
    CKCON = 0x00;                                   //access to set external data bus speed to fastest
    WTST = 0x00;                                    //Set the program code to wait for parameters,
                                                    //assign to CPU The speed of executing the program is set to the fastest

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

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

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

while (1);
}

```

## 22.5.2 I2C host mode access to AT24C256 (query method)

---

```

//The test frequency is 11.0592MHz

##include "stc8h.h"
#include "stc32g.h"                                         // For header files, see Download Software
#include "intrins.h"

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

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

void Start()
{
    I2CMSCR = 0x01;                                       //send START Order
    Wait();
}

```

```

}

void SendData(char dat)
{
    I2CTXD = dat;                                //write data to data buffer
    I2CMSCR = 0x02;                               //send command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                                //send readACK Order
    Wait();
}

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

void SendACK()
{
    I2CMSST = 0x00;                                //set ACK Signal
    I2CMSCR = 0x05;                               //send ACK Order
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                                //set NAK Signal
    I2CMSCR = 0x05;                               //send ACK Order
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                                //send STOP Order
    Wait();
}

void Delay()
{
    int i;

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

void main()
{
    EAXFR = 1;                                     //Enable XFR
    CKCON = 0x00;                                 //access to set external data bus speed to fastest
}

```

```

WTST = 0x00;                                // Set the program code to wait for parameters,
                                                // assign to CPU The speed of executing the program is set to the fastest

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 start command
SendData(0xa0);                                // Send device address write command

RecvACK();
SendData(0x00);                                // Send storage address high byte

RecvACK();
SendData(0x00);                                // Send storage address low byte

RecvACK();
SendData(0x12);                                // write test data 1

RecvACK();
SendData(0x78);                                // write test data 2

RecvACK();
Stop();                                         // send stop command

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

Start();                                         // send start command
SendData(0xa0);                                // Send device address write command

RecvACK();
SendData(0x00);                                // Send storage address high byte

RecvACK();
SendData(0x00);                                // Send storage address low byte

RecvACK();
Start();                                         // send start command
SendData(0xa1);                                // Send device address read command

RecvACK();
P0 = RecvData();                                //read data 1

SendACK();
P2 = RecvData();                                //read data 2

SendNAK();
Stop();                                         // send stop command

while (1);
}

```

## 22.5.3 I2C host mode access to PCF8563

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#include "intrins.h"

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

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

void Start()
{
    I2CMSCR = 0x01;                                //send START Order
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                                 //write data to data buffer
    I2CMSCR = 0x02;                               //send command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                               //send readACK Order
    Wait();
}

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

void SendACK()
{
    I2CMSST = 0x00;                               //set ACK Signal
    I2CMSCR = 0x05;                               //send ACK Order
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                               //set NAK Signal
    I2CMSCR = 0x05;                               //send ACK Order
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                               //send STOP Order
    Wait();
}

```

```

}

void Delay()
{
    int i;

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

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                               //Set the program code to wait for parameters,
                                                //assign to      CPU   The speed of executing the program is set to the fastest

    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;                            //EnableI2C host mode
    I2CMSST = 0x00;

    Start();                                    //send start command
    SendData(0xa2);                           //Send device address write command

    RecvACK();
    SendData(0x02);                           //send storage address

    RecvACK();
    SendData(0x00);                           //set seconds value

    RecvACK();
    SendData(0x00);                           //set minute value

    RecvACK();
    SendData(0x12);                           //set hour value

    RecvACK();
    Stop();                                    //send stop command

    while (1)
    {
        Start();                                //send start command
        SendData(0xa2);                         //Send device address write command

        RecvACK();
        SendData(0x02);                         //send storage address

        RecvACK();
        Start();                                //send start command
    }
}

```

```

SendData(0xa3);                                // Send device address read command
RecvACK();
P0 = RecvData();                                // read seconds value
SendACK();
P2 = RecvData();                                // read minute value
SendACK();
P3 = RecvData();                                // read hour value
SendNAK();
Stop();                                         // send stop command

Delay();
}

}

```

## 22.5.4 I2C Slave Mode (Interrupt Mode)

//The test frequency is 11.0592MHz

```

#ifndef "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

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

bit     isda;                                // Device address flag
bit     isma;                                // storage address flag
unsigned char      addr;
unsigned char edata     buffer[32];

void I2C_Isr() interrupt 24 {

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                         // deal with START event
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                         // deal with RECV event
        if (isda)
        {
            isda = 0;                            // deal with RECV event( RECV DEVICE ADDR )
        }
        else if (isma)
        {
            isma = 0;                            // deal with RECV event( RECV MEMORY ADDR )
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;           // deal with RECV event( RECV DATA )
        }
    }
    else if (I2CSLST & 0x10)

```

```

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

void main()
{
    EAXFR = 1;                                     // Enable      XFR
    CKCON = 0x00;                                   // access to set external data bus speed to fastest
    WTST = 0x00;                                    // Set the program code to wait for parameters,
                                                    // assign to      CPU The speed of executing the program is set to the fastest

    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 mode
    I2CSLADR = 0x5a;                               // Setting the slave device address      I2CSLADR=0101_1010B
                                                    // register      I2CSLADR[7:1]=010_1101B, MA=0B0, .
                                                    //           MA for      The device address sent by the host must match the
                                                    //           I2CSLADR[7:1] same to access this slave device.
                                                    // If the host needs to write data, it needs to send 5AH(0101_1010B)
                                                    // it. If the host needs to read data, it needs to send 5BH(0101_1011B)

    I2CSLST = 0x00;
    I2CSLCR = 0x78;                               // Enable Slave Mode Interrupt
    EA = 1;

    isda = 1;
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}

```

## 22.5.5 I2C Slave Mode (Query Mode)

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

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

bit isda; //Device address flag
bit isma; //storage address flag

unsigned char addr;
unsigned char edata buffer[32];

void main()
{
    EAXFR = 1; //enable access XFR
    CKCON = 0x00; //Set the external data bus speed to the fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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 mode
    I2CSLADR = 0x5a; //Setting the slave device address I2CSLADR=0101_1010B
                      //register I2CSLADR[7:1]=010_1101B, MA=0B0, ·
                      //MA for The device address sent by the host must match the
                      //I2CSLADR[7:1]/same to access this I2C device.
                      //If the host needs to write data, it needs to send 5AH(0101_1010B)
                      //it. If the host needs to read data, it needs to send 5BH(0101_1011B)

    I2CSLST = 0x00; //Disable Slave Mode Interrupts
    I2CSLCR = 0x00;

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

    while (1)
    {
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40; //deal with START event
        }
    }
}

```

```

else if (I2CSLST & 0x20)
{
    I2CSLST &= ~0x20;                                // deal with RECV event
    if (isda)
    {
        isda = 0;                                    // deal with RECV event( RECV DEVICE ADDR )
    }
    else if (isma)
    {
        isma = 0;                                    // deal with RECV event( RECV MEMORY ADDR )
        0; addr = I2CRXD;
        I2CTXD = buffer[addr];
    }
    else
    {
        buffer[addr++] = I2CRXD;                    // deal with RECV event( RECV DATA )
    }
}
else if (I2CSLST & 0x10)
{
    I2CSLST &= ~0x10;                                // deal with SEND event
    if (I2CSLST & 0x02)
    {
        I2CTXD = 0xff;                            // received NAK stop reading data
    }
    else
    {
        I2CTXD = buffer[++addr];                  // received ACK continue to read data
    }
}
else if (I2CSLST & 0x08)
{
    I2CSLST &= ~0x08;                                // deal with STOP event
    isda = 1;
    isma = 1;
}
}
}
}

```

## 22.5.6 Master Code for Testing I2C Slave Mode Code

---

```

//The test frequency is 11.0592MHz

##include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

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

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

```

```

void Start()
{
    I2CMSCR = 0x01;                                //send START Order
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                                  //write data to data buffer
    I2CMSCR = 0x02;                                //send command
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                                //send readACK Order
    Wait();
}

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

void SendACK()
{
    I2CMSST = 0x00;                                //set ACK Signal
    I2CMSCR = 0x05;                                //send ACK Order
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                                //set NAK Signal
    I2CMSCR = 0x05;                                //send ACK Order
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                                //send STOP Order
    Wait();
}

void main()
{
    EAXFR = 1;                                     //Enable      XFR
    CKCON = 0x00;                                   //access to set external data bus speed to fastest
    WTST = 0x00;                                    //Set the program code to wait for parameters,
                                                    //assign to      CPU The speed of executing the program is set to the fastest

    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 start command  
SendData(0x5a); //Send device address(010_1101B)+ write command(0B)  
RecvACK();  
SendData(0x00); //send storage address  
RecvACK();  
SendData(0x12); //write test data 1  
RecvACK();  
SendData(0x78); //write test data 2  
RecvACK();  
Stop(); //send stop command  
  
Start(); //send start command  
SendData(0x5a); //Send device address(010_1101B)+ write command(0B)  
RecvACK();  
SendData(0x00); //Send storage address high byte  
RecvACK();  
Start(); //send start command  
SendData(0x5b); //Send device address(010_1101B)+ read command(1B)  
RecvACK();  
P0 = RecvData(); //read data 1  
SendACK();  
P2 = RecvData(); //read data 2  
SendNAK();  
Stop(); //send stop command  
  
while (1);  
}
```

## 23 Advanced PWM

The MCU of STC32G series integrates an 8-channel 16-bit advanced PWM timer, which is divided into two groups of PWMs with different periods, named PWMA and PWMB, which can be set separately. The first group of PWM/PWMA can be configured as 4 groups of complementary/symmetrical/dead-time controlled PWM or capture external signals, and the second group of PWM/PWMB can be configured as 4-channel PWM output or capture external signals.

The clock frequency of the first group of PWM/PWMA can be the clock divided by the system clock through the registers [PWMA\\_PSCRH](#) and [PWMA\\_PSCRL](#), and the frequency division value can be any value between 1~65535. The clock frequency of the second group of PWM/PWMB can be the clock divided by the system clock through the registers [PWMB\\_PSCRH](#) and [PWMB\\_PSCRL](#), and the frequency division value can be any value between 1~65535. The clock frequencies of the two groups of PWMs can be set independently.

The first group of PWM timer/PWMA has 4 channels (PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N), each channel can independently realize PWM output (complementary symmetrical PWM output with dead time can be set), capture and compare functions; the second group of PWM timer/PWMB has 4 channels (PWM5, PWM6, PWM7, PWM8), each channel can also independently implement 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 dead zone, while the second group can only output single-ended PWM, and other functions are exactly the same. The following description of advanced PWM timers only takes the first group as an example.

When using the first group of PWM timers to output PWM waveforms, PWM1P/PWM2P/PWM3P/PWM4P outputs can be enabled individually, and PWM1N/PWM2N/PWM3N/PWM4N outputs can also be enabled individually. For example: If PWM1P output is enabled alone, PWM1N can no longer output independently, unless PWM1P and PWM1N form a set of complementary symmetrical outputs. The four outputs of PWMA can be set independently, for example: PWM1P and PWM2N outputs can be enabled independently, and PWM2N and PWM3N outputs can also be enabled independently. If you need to use the first group of PWM timers for capture function or pulse width measurement, the input signal can only be input from the positive terminal of each channel, that is, only PWM1P/PWM2P/PWM3P/PWM4P have capture function and pulse width measurement function.

When two sets of advanced PWM timers capture external signals, you can select rising edge capture or falling edge capture. If you need to capture the rising edge and falling edge at the same time, you can connect the input signal to two PWM channels at the same time, and enable one of them to capture the rising edge and the other to [capture the falling edge](#). [What's more powerful is that when the external input signal is connected to two PWM channels at the same time, the period value and duty cycle value of the signal can be captured at the same time.](#)

**STC three hardware PWM comparison: PCA/**

[CCP/PWM compatible with traditional 8051: can output](#) PWM waveform, capture external input signal and output high-speed pulse. 6-bit/7-bit/8-bit/10-bit PWM waveform can be output externally, the frequency of 6-bit PWM waveform is the frequency of PCA module clock source/64; the frequency of 7-bit PWM waveform is the frequency of PCA module clock source/128; 8-bit The frequency of the PWM waveform is the frequency of the PCA module clock source/256; the frequency of the 10-bit PWM waveform is the frequency of the PCA module clock source/1024. Capture external input signal, can capture rising edge, falling edge or both.

[The 15-bit enhanced PWM of the STC8G series:](#) can only output the PWM waveform to the outside, and has no input capture function. The frequency and duty cycle of external output PWM can be set arbitrarily. Through software intervention, multiple complementary/symmetrical/dead-time PWM waveforms can be realized. There are external anomaly detection function and real-time trigger ADC conversion function.

[16-bit advanced PWM timer of STC32G/STC8H series:](#) It is the most powerful PWM of STC at present, which can output PWM waveform of any frequency and any duty cycle to the outside world. Complementary/symmetrical/dead-time PWM waveforms can be output without software intervention. Can capture external input signal, can capture rising edge, falling edge or both rising edge and falling edge. When measuring external waveform, it can measure the period value and duty cycle value of the waveform at the same time. There are quadrature encoding function, external abnormal detection function and real-time trigger ADC conversion function.

In the following description, PWMA represents the first group of **PWM timers**, and PWMB represents the second group of **PWM timers**

Group 1 Advanced **PWM Timer/PWMA** Internal Signal Description

TI1: External clock input signal 1 (PWM1P pin signal or PWM1P/PWM2P/PWM3P XOR 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: The reference waveform (intermediate waveform) output by **output channel 1**

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:** edge signal of TI2F

**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:** Reference waveform output by output channel 2 (intermediate waveform)

OC2: Main output signal of channel 2 (OC2REF signal after CC2P polarity processing)

**OC2N:** Complementary output signal of channel 2 (OC2REF signal after CC2NP polarity processing)

TI3: External clock input signal 3 (PWM3P pin signal)

**TI3F:** TI3 signal after IC3F digital filtering

**TI3F\_ED:** 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:** Reference waveform output by output channel 3 (intermediate waveform)

OC3: Main output signal of channel 3 (OC3REF signal after CC3P polarity processing)

**OC3N:** Complementary output signal of channel 3 (OC3REF signal after CC3NP polarity processing)

TI4: External clock input signal 4 (PWM4P pin signal)

**TI4F:** TI4 signal after IC4F digital filtering

**TI4F\_ED:** 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:** Reference waveform output by output channel 4 (intermediate waveform)

OC4: Main output signal of channel 4 (OC4REF signal after CC4P polarity processing)

**OC4N:** Complementary output signal of channel 4 (OC4REF signal after CC4NP polarity processing)

ITR1: Internal trigger input signal 1

ITR2: Internal trigger input signal 2

TRC: fixed to TI1\_ED

**TRGI:** Trigger input signal after TS multiplexer

**TRGO:** Trigger output signal after MMS multiplexer

ETR: External trigger input signal (PWMETI1 pin signal)

**ETRP:** ETR signal after ETP edge detector and ETPS divider

**ETRF:** ETRP signal after ETF digital filtering

BRK: Brake input signal (PWMF1T)

CK\_PSC: prescaler 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 PWM5/PWM6/PWM7 XOR signal)

**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:** Reference waveform output by output channel 5 (intermediate waveform)

OC5: Main output signal of channel 5 (OC5REF signal after CC5P polarity processing)

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

**TI6F:** TI6 signal after IC6F digital filtering

**TI6F\_ED:** 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:** Reference waveform output by output channel 6 (intermediate waveform)

OC6: Main output signal of channel 6 (OC6REF signal after CC6P polarity processing)

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

**TI7F:** TI7 signal after IC7F digital filtering

**TI7F\_ED:** 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:** Reference waveform output by output channel 7 (intermediate waveform)

OC7: Main output signal of channel 7 (OC7REF signal after CC7P polarity processing)

TI8: External clock input signal 8 (PWM8 pin signal)

**TI8F:** TI8 signal after IC8F digital filtering

**TI8F\_ED:** 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:** The reference waveform output by output channel 8 (intermediate waveform)

OC8: Main output signal of channel 8 (OC8REF signal after CC8P polarity processing)

## 23.1 Introduction

The only difference between PWMB and PWMA is that PWMA can output complementary symmetrical PWM with dead time, while PWMB can only output single-ended PWM, other functions are exactly the same. The following introduction about advanced PWM only takes PWMA as an example.

PWMA consists of a 16-bit auto-reload counter driven by a programmable prescaler.

PWMA is suitable for many different purposes:

ÿ Basic timing

Measure pulse width of input signal (input capture) ÿ Generate output waveform (output compare, PWM and single-pulse mode) ÿ Correspond to different events (capture, compare, overflow, brake, trigger) ÿ Interrupt with PWMB Or external signal (external clock, reset signal, trigger and enable signal) synchronization

PWMA is suitable for a wide variety of control applications, including those requiring mid-aligned mode PWM, which supports complementary output and dead time control.

The clock source of PWMA can be an internal clock or an external signal, which can be selected through the configuration register.

## 23.2 Main Features

Features of PWMA include:

ÿ 16-bit up, down, up/down auto-load counter ÿ Repeating counter

that allows updating of timer registers after a specified number of counter cycles ÿ 16-bit programmable  
(can be modified in real time) prescaler, divider The frequency coefficient is any number between 1 and 65535

value

ÿ Synchronization circuit for controlling timers and timer interconnection using external signals ÿ Up to 4 independent channels can be configured as: - Input capture - Output compare

- PWM output (edge or center-aligned mode) - Six-step

PWM output - Single-pulse mode output - Supports 4

complementary outputs on channel with programmable  
dead time

ÿ The brake input signal (PWMDLT) can put the timer output signal into a reset state or a certain state ÿ External trigger input pin  
(PWMDTI)

Events that generate interrupts include:

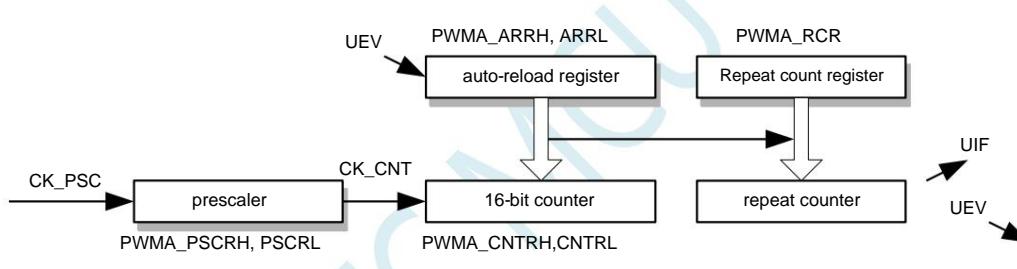
- Update: Counter overflow/underflow, Counter initialization (by software or internal/external trigger) - Trigger event (counter start, stop, initialization or count by internal/external trigger) - Input capture - Output compare - Brake signal input

## 23.3 Time base unit

The time base unit of the PWMA contains:

ÿ 16-bit up/down counter  
ÿ 16-bit auto-reload register  
ÿ Repeat counter  
Prescaler

PWMA Time Base Unit



The 16-bit counter, prescaler, auto-reload register and repeat counter register are all readable and writable by software.

Auto-reload registers consist of preload registers and shadow registers.

The auto-reload register can be written in two modes:

ÿ Auto preload is enabled (ARPE bit of PWMA\_CR1 register is 1).

In this mode, the data written to the auto-reload register will be saved in the preload register and will be stored on the next update event (UEV) is transferred to the shadow register.

ÿ Auto preload is disabled (ARPE bit of PWMA\_CR1 register is 0).

In this mode, data written to the auto-reload registers will be written to the shadow registers immediately.

The generation conditions of the update event: ÿ The counter overflows up or down. ÿ Software sets the UG bit in the PWMA\_EGR register. ÿ The clock/trigger controller generates the trigger event.

When preload is enabled (ARPE=1), if an update event occurs, the value in the preload register (PWMA\_ARR) will be written into the shadow register and the value in the PWMA\_PSCR register will be written into the prescaler .

Setting the UDIS bit in the PWMA\_CR1 register will disable the update event (UEV).

The output of the prescaler, CK\_CNT, drives the counter, and CK\_CNT is only in the counter enable bit (CEN) of the IM1\_CR1 register.

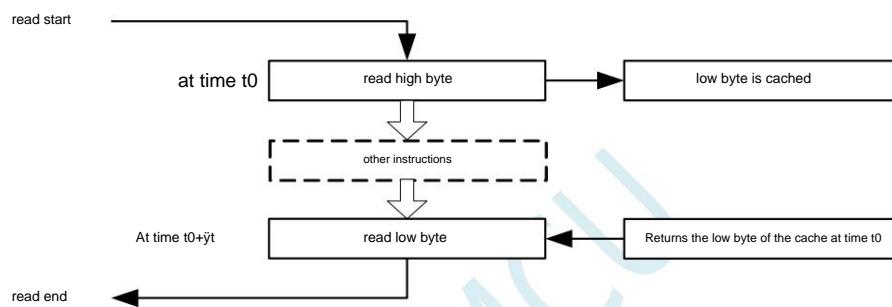
Only valid when set.

Note: The actual counter does not start counting until one clock cycle after the CEN bit is enabled.

### 23.3.1 Reading and writing 16-bit counters

The operation of writing to the counter is not buffered, and the PWMA\_CNTRH and PWMA\_CNTL registers can be written at any time, so To avoid writing wrong values, it is generally recommended not to write new values while the counter is running.

The read counter operation has an 8-bit buffer. The user must read the high byte of the timer first, after the user reads the high byte, the low byte will be automatically cached, the cached data will remain 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 preload register will be written to the 16-bit PWMA\_ARR register. This operation is done by two instructions, each instruction writes 1 byte. The high byte must be written first, followed by the low byte.

Shadow registers are locked when the upper byte is written and held until the lower byte is written.

### 23.3.3 Prescaler

Implementation of the prescaler:

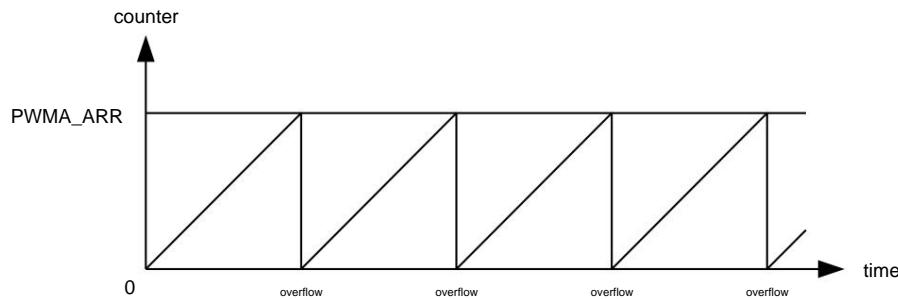
The prescaler for PWMA is based on a 16-bit counter controlled by a 16-bit register (PWMA\_PSCR). Due to this control The register is buffered so it can be changed at runtime. The prescaler scales the counter's clock frequency between 1 and 65536 Divide by any value. The value of the prescaler is written from the preload register, and the shadow register that holds the currently used value is loaded when the low byte is written. enter. Since two separate write operations are required to write to the 16-bit register, the high byte must be written first. The new prescaler value is the next Used when the next update event arrives. The read operation of the PWMA\_PSCR register is done through the preload register.

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

## 23.3.4 Count Up Mode

In up-counting mode, the counter counts from 0 to the user-defined compare value (the value of the PWMA\_ARR register), and then restarts counting from 0 and generating a counter overflow event; if the UDIS bit of the PWMA\_CR1 register is 0, a counter overflow event will be generated.

Counter in up-counting mode



An update can also be generated by software or by setting the UG bit in the PWMA\_EGR register using the trigger controller event.

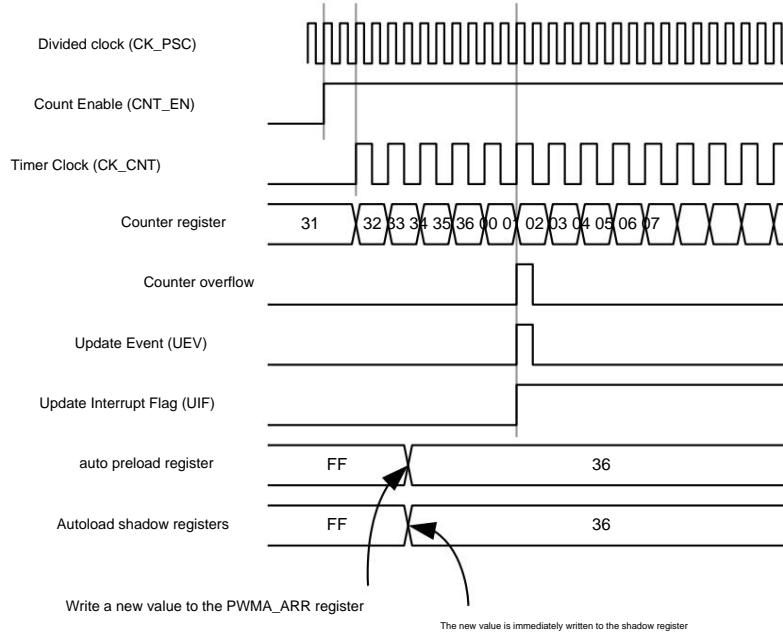
Update events can be disabled using software to set the UDIS bit in the PWMA\_CR1 register, which avoids the need to update the preload register. The shadow registers are updated when the device is executed. No update event will be generated until the UDIS bit is cleared. But when an update event should be generated, the count The prescaler will still be cleared to 0, and the prescaler count will also be cleared to 0 (but the prescaler value will remain unchanged). Also, if PWMA\_CR1 is set The URS bit in the register (select update request), setting the UG bit will generate an update event UEV, but the hardware does not set the UIF flag (ie no interrupt request is generated). This is to avoid simultaneous update and capture interrupts when the counter is cleared in capture mode.

When an update event occurs, all registers are updated, and the hardware simultaneously sets the update flag (PWMA\_SR) according to the URS bit. register UIF bits):

- ÿ The autoload shadow register is reloaded with the value of the preload register (PWMA\_ARR).
- ÿ The prescaler buffer is loaded with the value of the preload register (the contents of the PWMA\_PSC register).

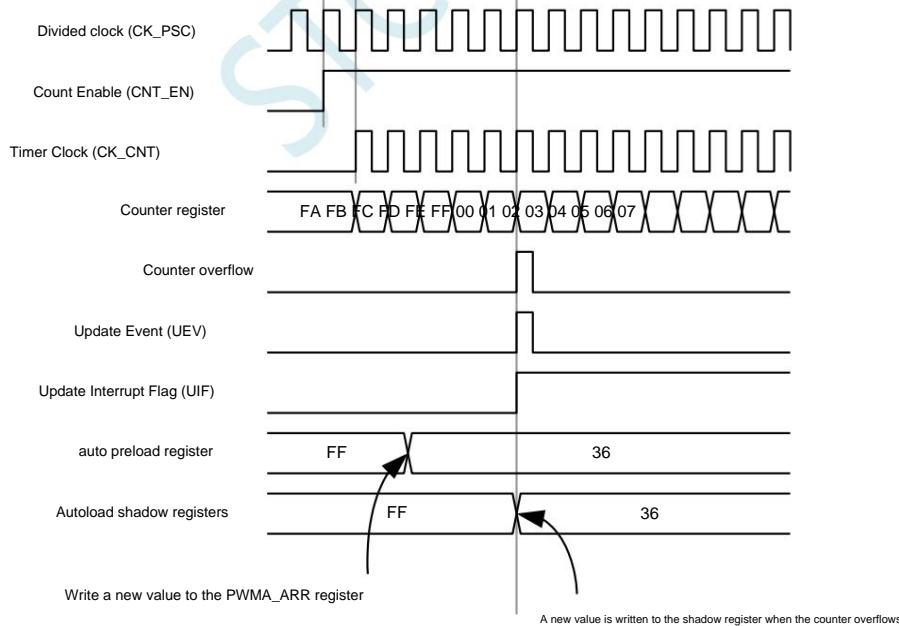
The following figure gives some examples to illustrate the action of the counter at different clock frequencies when PWMA\_ARR=0x36. The prescaler in the figure is 2, so the counter's clock (CK\_CNT) frequency is half the frequency of the prescaled clock (CK\_PSC). The autoload function is disabled in the figure can (ARPE=0), so when the counter reaches 0x36, the counter overflows, the shadow register is updated immediately, and an update is generated at the same time. event.

When ARPE=0 (ARR is not preloaded), the counter update when the prescaler is 2:



The prescaler in the figure below is 1, so the frequency of CK\_CNT is the same as CK\_PSC. In the figure, auto-reload is enabled (ARPE=1), so An overflow occurs when the counter reaches 0xFF. 0x36 will be written on overflow, generating an update event.

ARPE=1 (PWMA\_ARR preload) Counter update when prescaler is 1:

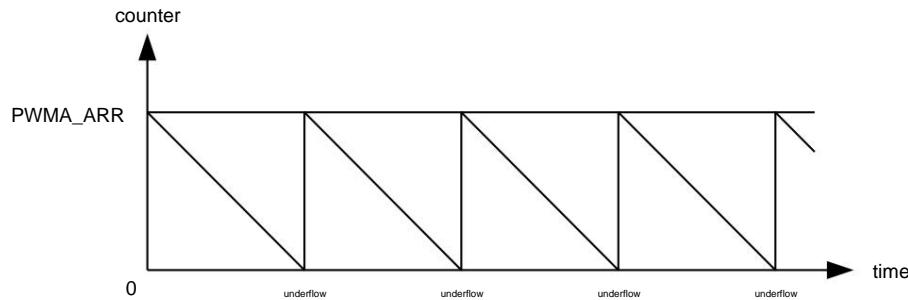


### 23.3.5 Down Counting Mode

In down mode, the counter starts from the auto-loaded value (the value of the PWMA\_ARR register) and counts down to 0, and then starts from the

Counting restarts with the automatically loaded value and generates a counter underflow event. If the UDIS bit of the PWMA\_CR1 register is cleared In addition, an update event (UEV) is also generated.

Counter in countdown mode



An update can also be generated by software or by setting the UG bit in the PWMA\_EGR register using the trigger controller. event.

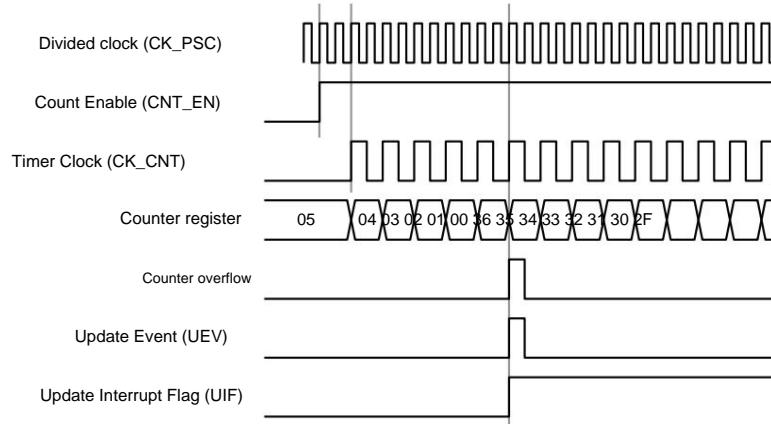
UEV events can be disabled by setting the UDIS bit in the PWMA\_CR1 register. This avoids updating the preload registers when updating shadow register. Therefore no update event will be generated until the UDIS bit is cleared. However, the counter will still restart from the current autoload value counts, and the prescaler counter restarts from 0 (but the prescaler cannot be modified). Also, if PWMA\_CR1 is set The URS bit in the register (select update request), setting the UG bit will generate an update event UEV but not set the UIF flag (so no interrupts), this is to avoid both update and capture interrupts being generated when a capture event occurs and the counter is cleared.

When an update event occurs, all registers are updated, and the hardware sets the update flag at the same time according to the URS bit (PWMA\_SR register UIF bits):

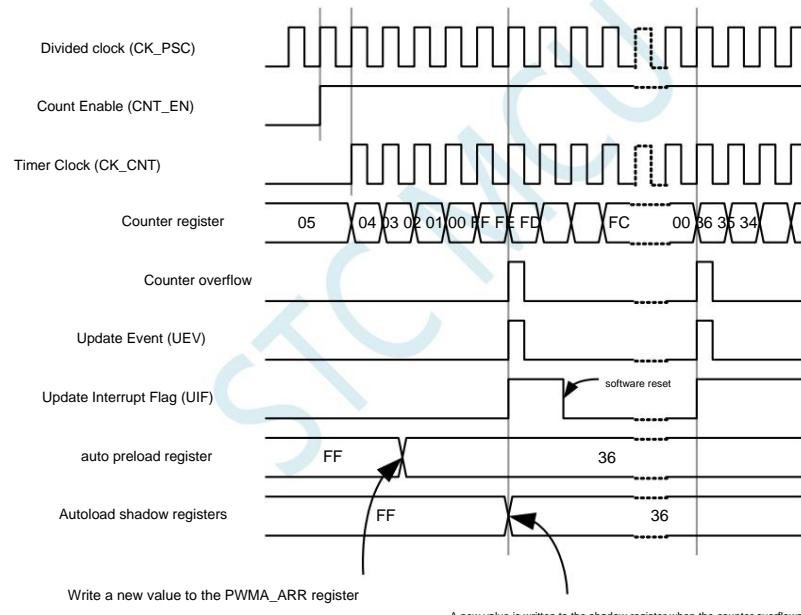
- ÿ The autoload shadow register is reloaded with the value of the preload register (PWMA\_ARR).
- ÿ The prescaler buffer is loaded with the value of the preload register (the contents of the PWMA\_PSC register).

Below are some graphs of the counter at different clock frequencies when PWMA\_ARR=0x36. The figure below depicts the countdown mode in Next, the new value is written on the next cycle when preload is not enabled.

ARPE=0 (ARR not preloaded), counter update when prescaler is 2:



ARPE=1 (ARR preload), counter update when prescaler is 1

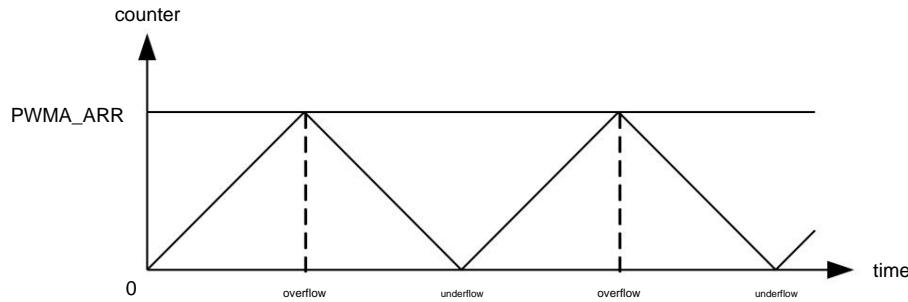


### 23.3.6 Center -aligned mode (count up/down)

In center-aligned mode, the counter starts counting from 0 to the value in the PWMA\_ARR register, generating a counter overflow event, then count down from the value in the PWMA\_ARR register to 0 and generate a counter underflow event; then start counting again from 0.

In this mode, the DIR direction bit in PWMA\_CR1 cannot be written. It is updated by hardware and indicates the current counting direction.

Counter in center-aligned mode



If the timer has a repeating counter, it will be generated after repeating the specified number of times (the value of PWMA\_RCR) to overflow and underflow.

Generated Update Event (UEV). Otherwise, an update event will be generated for each overflow and underflow.

An update can also be generated by software or by setting the UG bit in the PWMA\_EGR register using the trigger controller.

event. At this time, the counter starts counting from 0 again, and the prescaler also starts counting from 0 again.

UEV events can be disabled by setting the UDIS bit in the PWMA\_CR1 register. This avoids updating the preload register New shadow register. Therefore, no update event will be generated until the UDIS bit is cleared to 0. However, the counter will still reload automatically based on the current value, continue counting up or down. If the timer has a repeat counter, since the repeat register is not double buffered, the new repeat The value will take effect immediately, so you need to be careful when modifying it. Additionally, if the URS bit in the PWMA\_CR1 register is set (select more new request), setting the UG bit will generate an update event UEV but not set the UIF flag (and therefore not generate an interrupt), this is to avoid This avoids simultaneous update and capture interrupts when a capture event occurs and the counter is cleared.

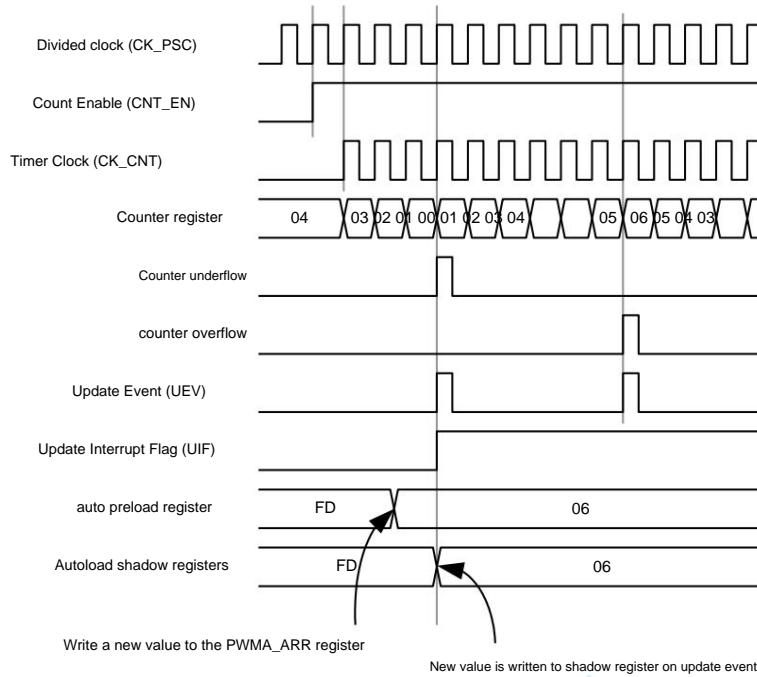
When an update event occurs, all registers are updated, and the hardware updates the flags (bits in the PWMA\_SR register according to the URS bit UIF bits):

- ÿ The prescaler register is loaded with the preload (PWMA\_PSC register) value.
- ÿ The current autoload register is updated with the preload value (the contents of the PWMA\_ARR register).

Note that if an update occurs due to 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:

The internal clock frequency division factor is 1, PWMA\_ARR=0x6, ARPE=1



Tips for using center alignment mode:

When the center-aligned mode is activated, the counter will count according to the original up/down configuration. That is, the DIR bit in the PWMA\_CR1 register will determine whether the counter will count up or down. Also, software cannot modify the value of the DIR bit and the CMS bit at the same time. It is not recommended to write the value of the counter while the counter is counting in the center-aligned mode, which will lead to unpredictable consequences. specific say:

When a value larger than the auto-reload value is written to the counter (PWMA\_CNT>PWMA\_ARR), the counting direction of the counter does not change. For example the counter has overflowed, but the counter is still counting up.  
0 or the value of PWMA\_ARR is written to the counter, but the update event does not occur.  
The safe way to use the counter in center-aligned mode is to generate an update event in software (setting the UG bit in the PWMA\_EGR register) before starting the counter, and not to modify the value of the counter while the counter is counting.

### 23.3.7 Repeat Counter

The time base unit explains how the update event (UEV) is generated when the counter overflows/underflows, however in fact it can only be repeated Generated when the value of the counter reaches 0. This feature is very useful for generating PWM signals.

This means that on every N count overflow or underflow, data is transferred from the preload register to the shadow register (PWMA\_ARR auto-reload register, PWMA\_PSC preload register, and in compare mode the capture/compare register PWMA\_CCRx), N is the value in the PWMA\_RCR repeat count register.

The repetition counter is decremented when any of the following conditions are true:

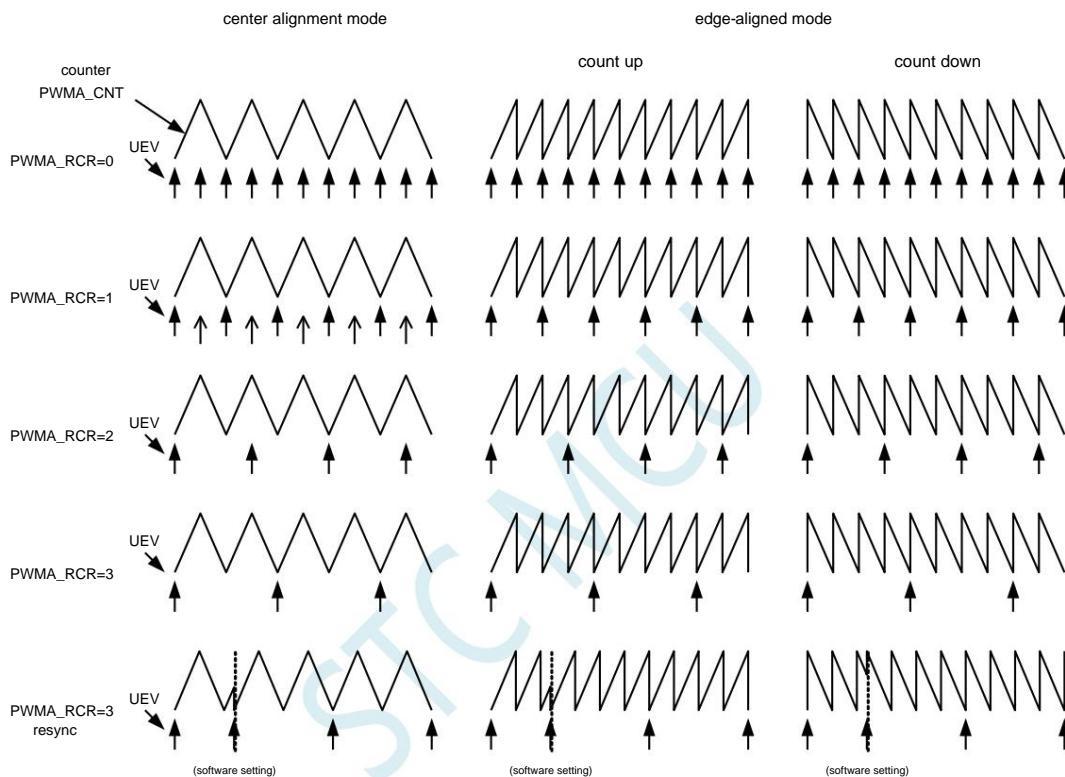
Every time the counter overflows in count-up mode. Every time the counter overflows in count-down mode. Every time it overflows and every time it underflows in center-aligned mode.

Although this limits the maximum cycle period of the PWM to 128, it can update the duty cycle 2 times per PWM period. in the center

In aligned mode, because the waveform is symmetrical, if the compare register is refreshed only once per PWM cycle, the maximum resolution is  $2 \times t_{CK\_PSC}$ .

The repeat counter is automatically loaded and the repeat rate is defined by the value of the PWMA\_RCR register. When an update event is generated by software (via By setting the UG bit in PWMA\_EGR) or generated by the hardware's clock/trigger controller, no matter what the value of the repeat counter is, An update event occurs immediately and the contents of the PWMA\_RCR register are reloaded into the repeat counter.

Examples of update rates in different modes and register settings for PWMA\_RCR



## 23.4 Clock/Trigger Controller

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

### 23.4.1 Prescaled Clock (CK\_PSC)

The prescaled clock (CK\_PSC) for the time base unit can be provided by the following sources:

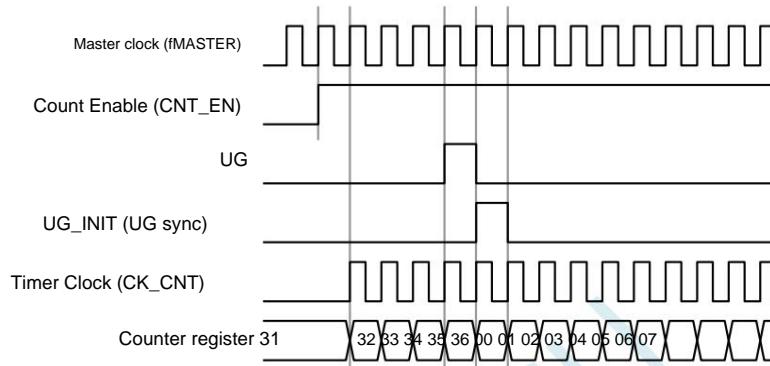
- ÿ Internal clock (fMASTER)
- ÿ External clock mode 1: external clock input (Tlx)
- ÿ External clock mode 2: external trigger input ETR
- ÿ Internal trigger input (ITRx): Use one timer as the prescaled clock of another timer.

## 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, ECE=0 in the PWMA\_ETR register), the CEN, DIR and UG bits are the actual control bits and can only be modified by software (The UG bit is still automatically cleared). Once the CEN bit is written to 1, the prescaler is clocked from the internal clock.

The figure below describes the operation of the control circuit and the up-counter in normal mode without prescaler.

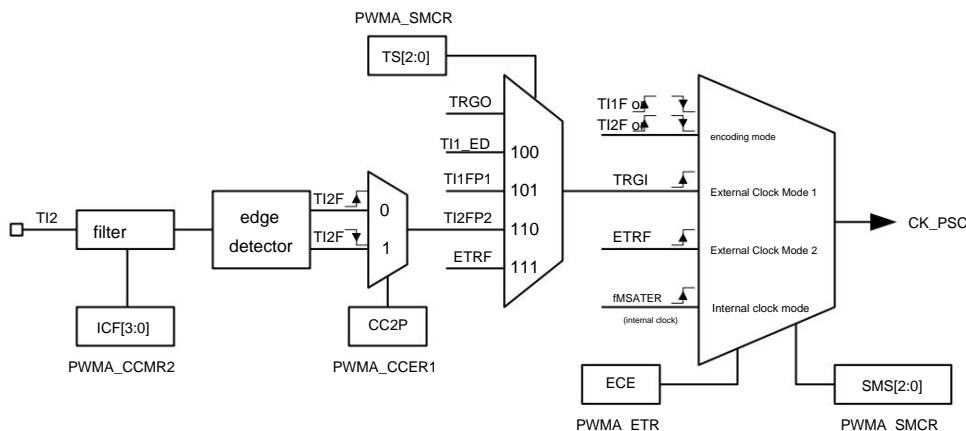
Control circuit in normal mode, fMASTER frequency division factor is 1



## 23.4.3 External Clock Source Mode 1

This mode is selected when SMS=111 in the PWMA\_SMCR register. The counter can count on every 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 steps:

1. Configure CC2S=01 in the PWMA\_CCMR2 register, use channel 2 to detect the rising edge of TI2 input

2. Configure the IC2F[3:0] bits of the PWMA\_CCMR2 register to select the input filter bandwidth (if no filter is required, keep IC2F=0000)

Note: The capture prescaler is not used as a trigger, so it does not need to be configured, nor does the TI2S bit, they are only used to select the input capture source.

3. Configure CC2P=0 in PWMA\_CCER1 register, select rising edge polarity 4. Configure SMS=111 in

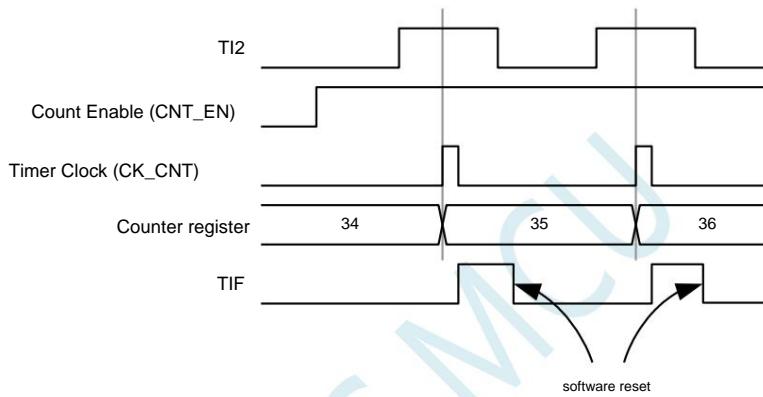
PWMA\_SMCR register, configure counter to use external clock mode 1 5. Configure TS=110 in PWMA\_SMCR register, select TI2 as input source 6 . Set CEN=1 in the PWMA\_CR1 register to start the counter

When the rising edge occurs on TI2, the counter counts once, and the trigger flag bit (TIF bit in the PWMA\_SR1 register) is set to 1, such as

An interrupt request is generated if the interrupt is enabled (configured in the PWMA\_IER register).

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

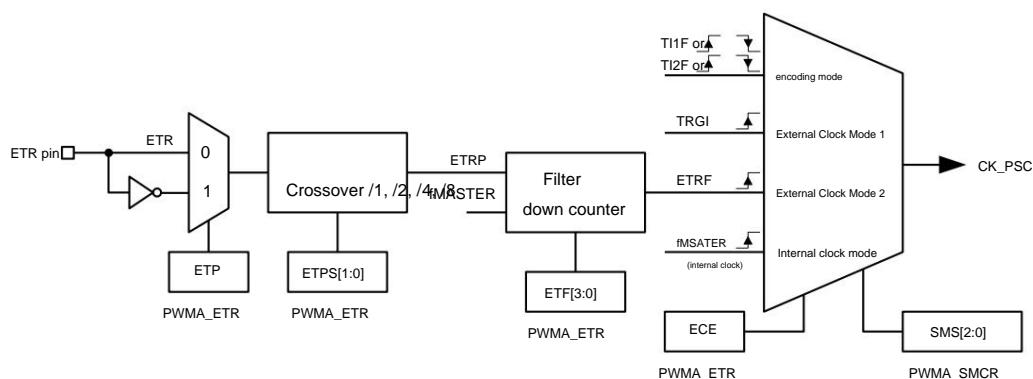
Control Circuit in External Clock Mode 1



#### 23.4.4 External Clock Source Mode 2

The counter can count on every rising or falling edge of the external trigger input ETR signal. Set the ECE of the PWMA\_ETR register to This mode is selected by writing a 1 to the bit.

Overall block diagram of external trigger input:

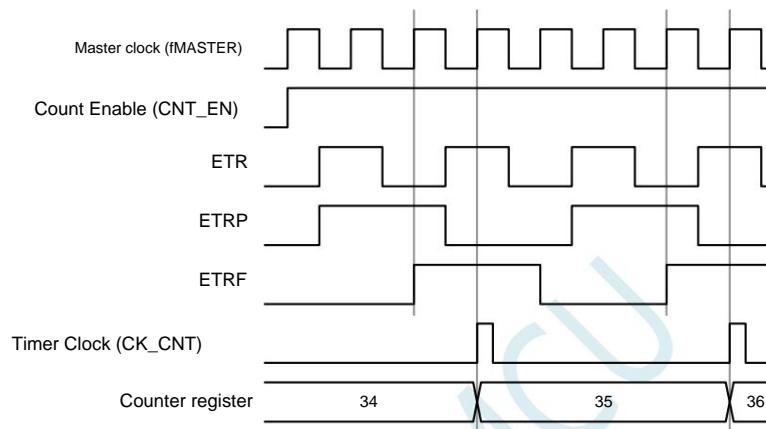


For example, to configure the counter to count up every 2 rising edges of the ETR signal, use the following steps:

1. No filter is needed in this example, configure ETF[3:0]=0000 in the PWMA\_ETR register
2. Set the prescaler and configure ETPS[1:0]=01 in the PWMA\_ETR register
3. Select the rising edge detection of ETR and configure ETP=0 in the PWMA\_ETR register
4. Enable external clock mode 2, configure ECE=1 in the PWMA\_ETR register
5. Start the counter and write CEN=1 in the PWMA\_CR1 register

The counter counts every 2 rising edges of ETR.

Control Circuit in External Clock Mode 2



### 23.4.5 Trigger synchronization

The PWMA's counter is synchronized with an external trigger signal using three modes:

- ÿ Standard trigger mode
- ÿ Reset trigger mode
- ÿ Gated trigger mode

Standard trigger mode

The enable (CEN) of the counter depends on the event on the selected input.

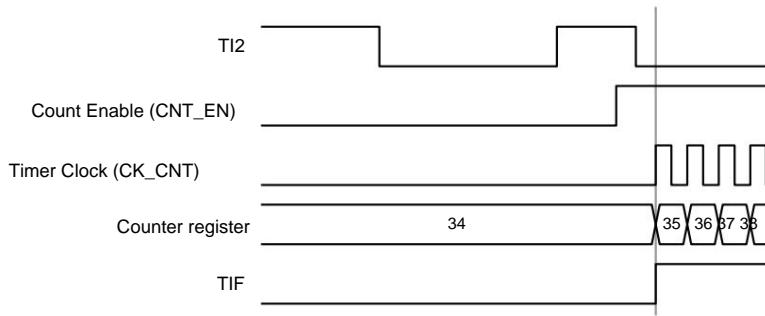
In the following example, the counter starts counting up on the rising edge of the TI2 input:

1. Configure CC2P=0 in the PWMA\_CCER1 register, and select the rising edge of TI2 as the trigger condition.
2. Configure SMS=110 in the PWMA\_SMCR register to select the counter as trigger mode. Configure the PWMA\_SMCR register TS=110, select TI2 as the input source.

When a rising edge occurs on TI2, the counter starts to count under the drive of the internal clock, and the TIF flag is set at the same time. TI2 rising edge and count

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

Standard trigger mode control circuit



#### reset trigger mode

The counter and its prescaler can be reinitialized when a trigger input event occurs. At the same time, if PWMA\_CR1 TS=101 of the device, an update event UEV is also generated, then all preload registers (PWMA\_ARR, PWMA\_CCRx) will be updated.

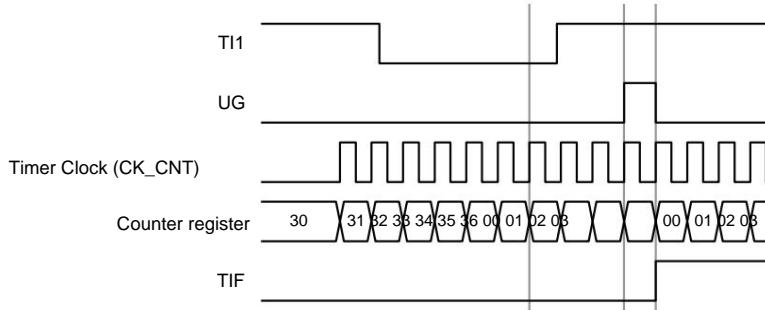
In the following example, a rising edge at the TI1 input causes the up-counter to be cleared:

1. Configure CC1P=0 in the PWMA\_CCER1 register to select the polarity of TI1 (only detect the rising edge of TI1).
2. Configure SMS=100 in the PWMA\_SMCR register to select the timer as reset trigger mode. Configure the PWMA\_SMCR register TS=101 of the device, and select TI1 as the input source.
3. Configure CEN=1 in the PWMA\_CR1 register to start the counter.

The counter starts counting according to the internal clock, and then counts normally until a rising edge of TI1 occurs. At this point, the counter is cleared and then restarts counting from 0. At the same time, the trigger flag (TIF bit in the PWMA\_SR1 register) is set, if the interrupt is enabled (PWMA\_IER TIE bit of the register), an interrupt request is generated.

The figure below shows the behavior when the auto-reload register PWMA\_ARR=0x36. Between the rising edge of TI1 and the actual reset of the counter The delay depends on the resynchronization circuit at the TI1 input.

#### Control circuit in reset trigger mode



#### Gated trigger mode

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

In the following example, the counter only counts up when TI1 is low:

1. Configure CC1P=1 in the PWMA\_CCER1 register to determine the polarity of TI1 (only detect low level on TI1).

2. Configure SMS=101 in the PWMA\_SMCR register, select the timer as the gated trigger mode, configure the PWMA\_SMCR register

TS=101 in the controller, select TI1 as the input source.

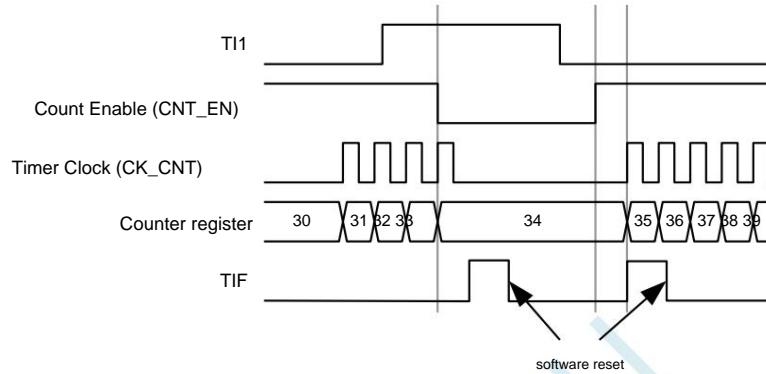
3. Configure the CEN=1 of the PWMA\_CR1 register to start the counter (in gated mode, if CEN=0, the counter cannot be started).

move, 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 will be set. The delay between the rising edge of TI1 and the actual stop of the counter depends on the resynchronization circuit at the TI1 input.

Control circuit in gated trigger mode



#### External Clock Mode 2 Joint Trigger Mode

External clock mode 2 can be used with trigger mode of another input signal. For example, the ETR signal is used as an input for an external clock input, another input signal can be used as trigger input (standard trigger mode, reset trigger mode and gated trigger mode are supported). Be careful not to ETR is configured as TRGI through the TS bit in the PWMA\_SMCR register.

In the example below, once a rising edge occurs on TI1, the counter counts up once on each rising edge of ETR:

1. Configure the external trigger input circuit through the PWMA\_ETR register. Configure ETPS=0 to disable prescaler, configure ETP=0 to monitor

On the rising edge of ETR signal, configure ECE=1 to enable external clock mode 2.

2. Configure CC1P=0 in the PWMA\_CCER1 register to select the rising edge trigger of TI1.

3. Configure SMS=110 in the PWMA\_SMCR register to select the timer as trigger mode. Configure the PWMA\_SMCR register

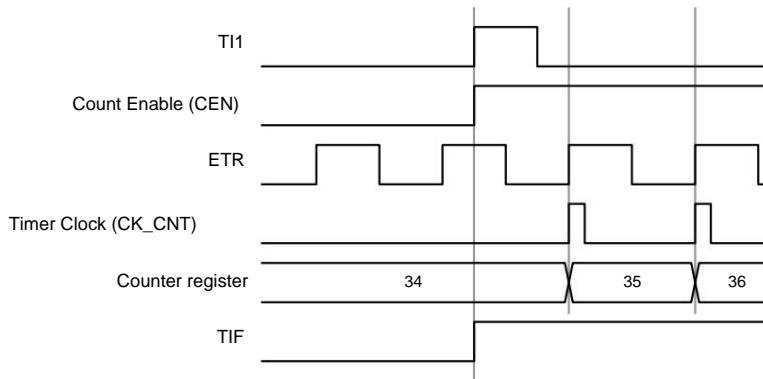
TS=101 to select TI1 as the input source.

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 resynchronization circuit at the TI1 input.

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

Control circuit in external clock mode 2+ trigger mode



### 23.4.6 Synchronization with PWMB

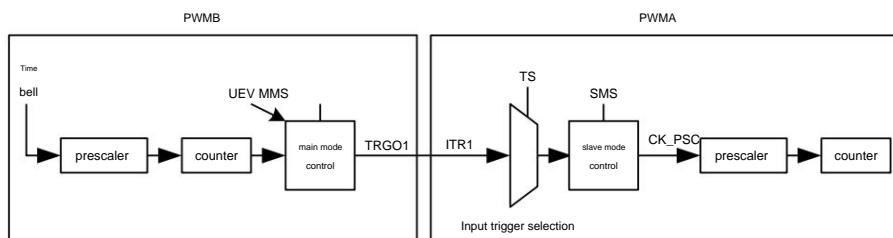
In the chip, the timers are interconnected internally for synchronization or linking of the timers. When a timer is configured in master mode, it can complete the reset operation, start operation, stop operation or operation by outputting the trigger signal (TRGO) to those timers configured as slave mode. The driver clock for those timers.

Use **TRGO** of **PWMB** as the prescaled clock of **PWMA**

For example, the user can configure PWMB as the prescaled clock of PWMA, the following configuration is required:

1. Configure PWMB in master mode so that a periodic trigger signal is output on each update event (UEV). Configure PWMB\_CR2 Register MMS=010, so that TRGO can output a rising edge at each update event.
2. The TRGO signal output by PWMB is linked to PWMA. PWMA needs to be configured to trigger slave mode, using ITR2 as input trigger signal. The above operations can be achieved by configuring TS=010 in the PWMA\_SMCR register.
3. Set the clock/trigger controller to external clock mode 1 by configuring SMS=111 in the PWMA\_SMCR register. This action will make the rising edge of the periodic trigger signal TRGO output by PWMB drives the clock of PWMA.
4. Finally, set the CEN bit of PWMB (in the PWMB\_CR1 register) to enable both PWMs.

Timer example in master/triggered slave mode



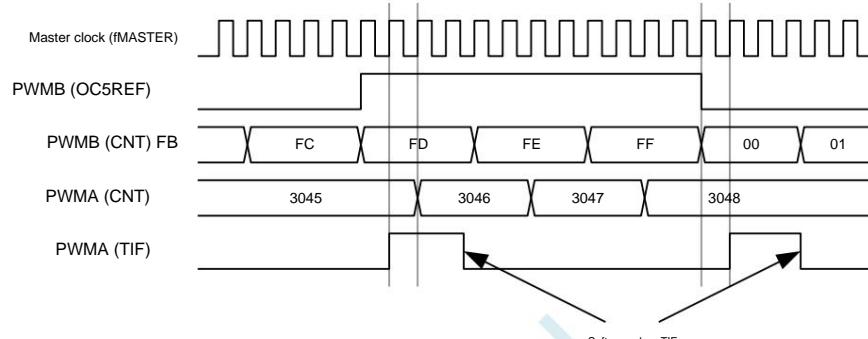
#### Enable PWMA with PWMB

- In this example, we enable PWMA with the compare output of PWMB. PWMA is only pressed when the OC1REF signal of PWMB is high. Counts according to its own drive clock. Both PWMs are clocked by fMASTER divided by 4 (fCK\_CNT = fMASTER/4).
1. Configure PWMB as master mode and output the compare output signal (OC1REF) as a trigger signal. (Configure the PWMB\_CR2 register MMS=100 of the device).

2. Configure the waveform of the OC5REF signal of PWMB (PWMB\_CCMR1 register).
3. Configure PWMA to use the output of PWMB as its own trigger input signal (configure TS=010 in the PWMA\_SMCR register).
4. Configure PWMA for gated trigger mode (configure SMS=101 in PWMA\_SMCR register).
5. Set the CEN bit (PWMA\_CR1 register) to enable PWMA.
6. Set the CEN bit (PWMB\_CR1 register) to enable PWMB.

Note: The clocks of the two PWMs are not synchronized, but only affect the enable signal of PWMA.

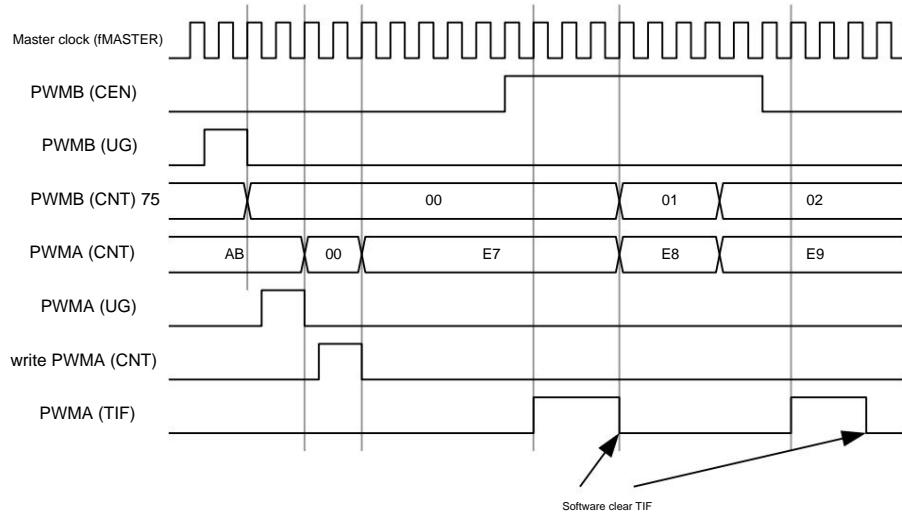
PWMB's output gate triggers PWMA



In the above figure, the counter and prescaler of PWMA are not initialized before startup, so they all start counting from the existing value. If By resetting both timers before starting PWMB, the user can write the desired value to the PWMA counter to start it from the specified value. start counting. The reset operation of PWMA can be realized by software writing the UG bit of the PWMA\_EGR register.

In the example below, we synchronize PWMB and PWMA. PWMB is master mode and starts counting from 0. PWMA is Trigger slave mode and start counting from 0xE7. Both PWMs use the same division factor. When the CEN of the PWMB\_CR1 register is cleared bit, PWMB is disabled and PWMA stops counting.

1. Configure PWMB as master mode and output the compare output signal (OC5REF) as a trigger signal. (Configure the PWMB\_CR2 register MMS=100 of the device).
2. Configure the waveform of the OC5REF signal of PWMB (PWMB\_CCMR1 register).
3. Configure PWMA to use the output of PWMB as its own trigger input signal (configure TS=010 in the PWMA\_SMCR register).
4. Configure PWMA for gated trigger mode (configure SMS=101 in PWMA\_SMCR register).
5. Reset PWMB by writing 1 to the UG bit (PWMB\_EGR register).
6. Reset PWMA by writing 1 to the UG bit (PWMA\_EGR register).
7. Write 0xE7 into the PWMA counter (PWMA\_CNTRL) to initialize PWMA.
8. Enable PWMA by writing 1 to the CEN bit (PWMA\_CR1 register).
9. Start PWMB by writing 1 to the CEN bit (PWMB\_CR1 register).
10. Stop PWMB by writing 0 to the CEN bit (PWMB\_CR1 register).



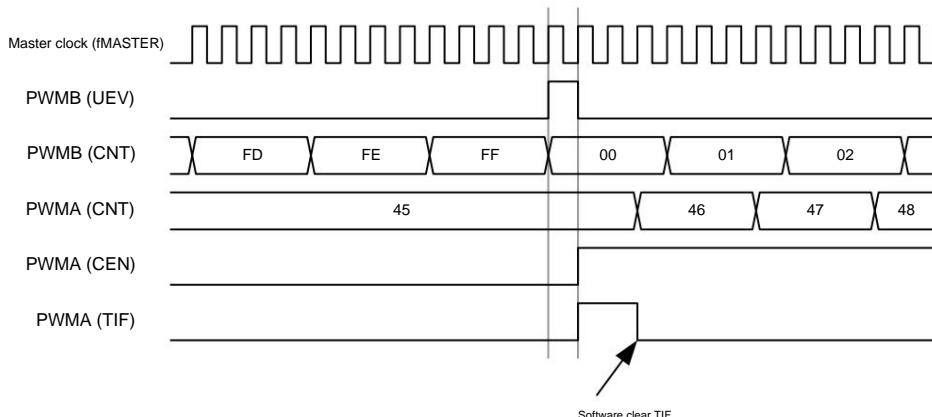
#### Start PWMA with PWMB

In this example, we start PWMA with an update event of PWMB.

PWMA starts counting from its existing value (can be non-zero) according to PWMA's own drive clock when an update event occurs in PWMB value). PWMA automatically enables the CEN bit after receiving the trigger signal and starts counting until the user registers PWMA\_CR1 Write 0 to the CEN bit of the device. Both PWMs use fMASTER divided by 4 as the drive clock ( $fCK\_CNT = fMASTER/4$ ).

1. Configure PWMB as master mode and output update signal (UEV). (Configure MMS=010 in PWMB\_CR2 register).
2. Configure the period of PWMB (PWMB\_ARR register).
3. Configure PWMA to use the output of PWMB as the input trigger signal (configure TS=010 in the PWMA\_SMCR register).
4. Configure PWMA to trigger mode (configure SMS=110 in PWMA\_SMCR register).
5. Set the CEN bit (PWMB\_CR1 register) to start PWMB.

PWMB's update event (PWMB-UEV) triggers PWMA



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

Trigger two PWMs synchronously with an external signal

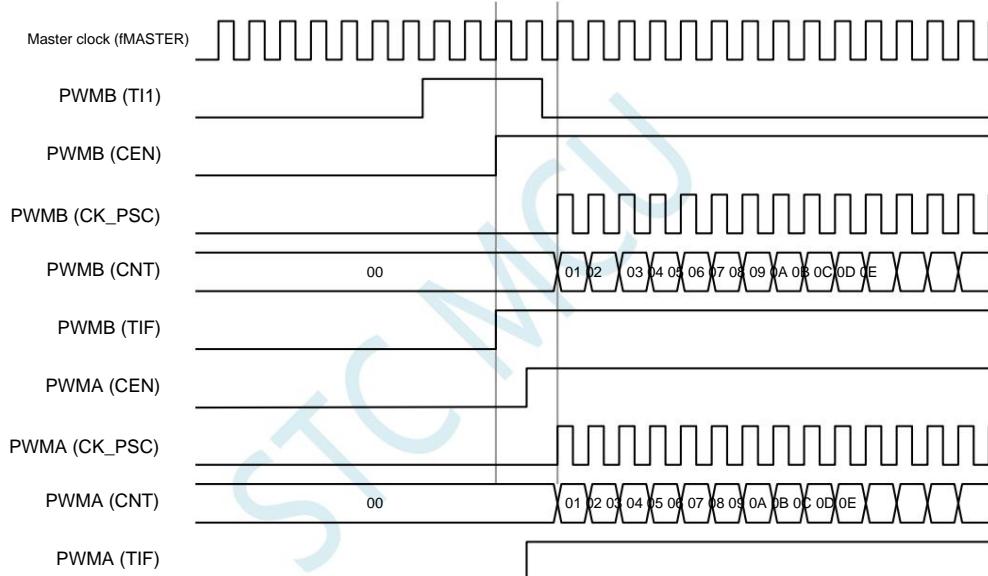
In this example, use the rising edge of TI1 to enable PWMB and simultaneously enable PWMA. To keep the timers aligned, PWMB It needs to be configured in master/slave mode (slave mode for TI1 signal, master mode for PWMA). 1.

Configure PWMB as master mode to output enable signal as the trigger of PWMA (configure MMS=001 in PWMB\_CR2 register). 2. Configure PWMB as slave mode, and use the TI1 signal as the input trigger signal (configure TS=100 in the PWMB\_SMCR register). 3. Configure the trigger mode of PWMB (configure SMS=110 in the PWMB\_SMCR register). 4. Configure PWMB for master/slave mode (configure MSM=1 in PWMB\_SMCR register). 5. Configure PWMA to use the output of PWMB as the input trigger signal (configure TS=010 in the PWMA\_SMCR register). 6. Configure the trigger mode of the PWMA (configure SMS=110 in the PWMA\_SMCR register).

When a rising edge occurs on TI1, both timers start counting synchronously and both the TIF bits are set.

Note: In this example, both timers are initialized (UG bit set) before starting, so they both start counting from 0, but the user can also insert an offset by modifying the counter register (PWMA\_CNT), In this case, a delay is inserted between the CK\_PSC signal and the CNT\_EN signal of PWMB.

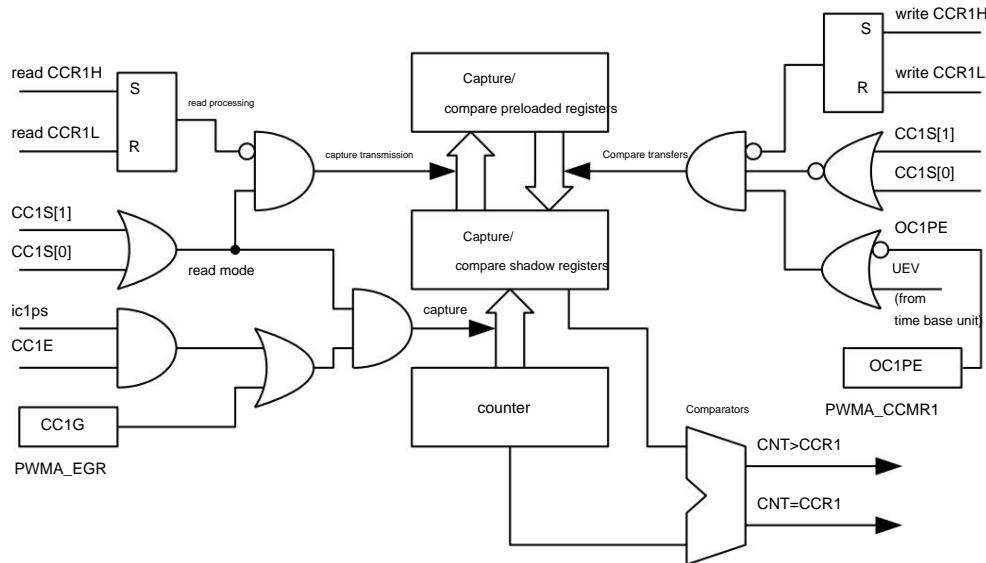
TI1 signal of PWMB triggers PWMB and PWMA



## 23.5 Capture/Compare Channels

PWM1P, PWM2P, PWM3P, PWM4P can be used as input capture, PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N can output compare. This function can be realized by configuring the CCIS channel selection bit of the capture/compare channel mode register (PWMA\_CCMR<sub>i</sub>), where *i* represents the number of channels from 1 to 4. Each capture/compare channel is built around a capture/compare register (including shadow registers), including the input portion of the capture (digital filter, multiplexer, and prescaler) and output portion (comparator and output control).

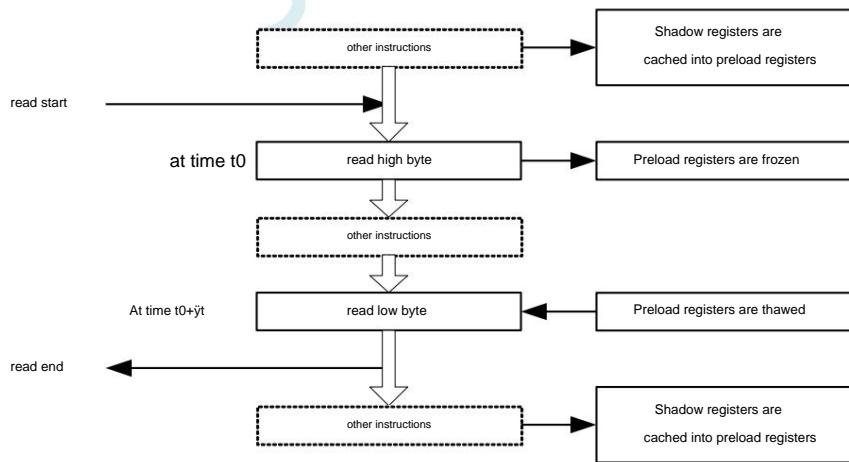
Main circuit for capture/compare channel 1 (similar for other channels)



The capture/compare module consists of a preload register and a shadow register. Read and write procedures only operate on preload registers. In capture mode, the capture takes place on the shadow register before being copied into the preload register. In compare mode, the content of the preload register is copied to the shadow register, and then the shadow register content is compared with the counter.

When the channel is configured in output mode (CCIS=0 in the PWMA\_CCMRi register), the PWMA\_CCRi register can be accessed at any time.

When the channel is configured in input mode, the read operation of the PWMA\_CCRi register is similar to the read operation of the counter. When a capture occurs, the contents of the counter are captured into the PWMA\_CCRi shadow register and then copied into the preload register. During a read operation, the preload register is frozen.



The above figure describes the read operation flow of the 16-bit CCRi register. The buffered data will remain unchanged until the end of the read flow.

After the entire read process is complete, if only the PWMA\_CCRiL register is read, the lower bit (LS) of the counter value is returned.

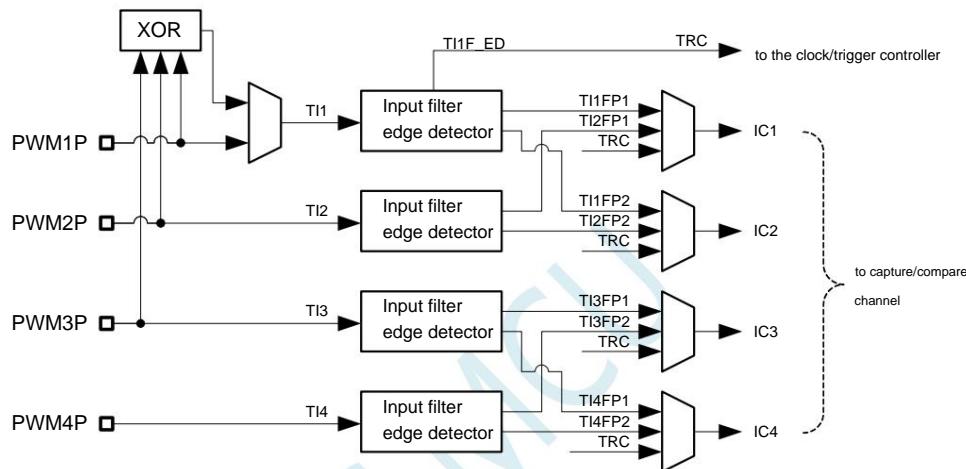
If the high-order (MS) data is read after the low-order (LS) data is read, the same low-order data will not be returned.

### 23.5.1 16-bit PWMA\_CCRi register write process

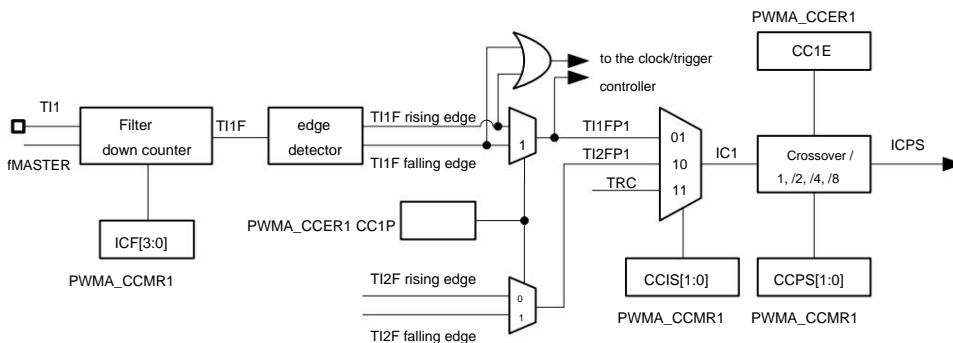
Writing to the 16-bit PWMA\_CCRi register is done through the preload register. Two instructions must be used to complete the entire process, one instruction corresponding to one byte. The upper byte (MS) must be written first. While writing the upper byte (MS), updates to the shadow registers are inhibited until the lower byte (LS) write operation is complete.

### 23.5.2 Input Module

Block Diagram of Input Module



As shown, the input section samples the corresponding TIx input signal and produces a filtered signal TIxF. Then, a polarized selector The selected edge monitor generates a signal (TIxFP<sub>x</sub>) which can be used as an input trigger for the trigger mode controller or as a capture control. This signal is prescaled into the capture register (ICxPS).



### 23.5.3 Input Capture Mode

In input capture mode, when the corresponding edge on the IC<sub>i</sub> signal is detected, the current value of the counter is latched into the capture/compare register

(PWMA\_CCRx). When a capture event occurs, the corresponding CCiIF flag (PWMA\_SR register) is set.

An interrupt request will be generated if the CCiIE bit in the PWMA\_IER register is set, ie the interrupt is enabled. If the CCiIF flag is already high when a capture event occurs, the recapture flag CCiOF (PWMA\_SR2 register) is set. CCiIF is cleared by writing CCiIF=0 or reading the captured data stored in the PWMA\_CCRI register. Write CCiOF=0 to clear CCiOF.

#### Capture on rising edge of **PWM** input signal

The following example shows how to capture the counter value into the PWMA\_CCR1 register on the rising edge of the T11 input. The steps are as follows:

1. Select a valid input: eg PWMA\_CCR1 is connected to the T11 input, so write CC1S=01 in the PWMA\_CCR1 register, the channel is configured as an input, and the PWMA\_CCR1 register becomes read-only.
2. According to the characteristics of the input signal T1i, the filter time of the corresponding input filter can be set by configuring the ICiF bit in the PWMA\_CCMRi register. Assuming the input signal is jittering for a maximum of 5 clock cycles, we have to configure the filter bandwidth to be longer than 5 clock cycles; so we can sample 8 consecutive times to confirm a true edge transition on T11, i.e. in the TIMi\_CCMR1 register Write IC1F=0011 in , at this time, the signal is valid only when 8 identical T11 signals are continuously sampled (sampling frequency is fMASTER).

3. Select the valid conversion edge of the T11 channel and write CC1P=0 (rising edge) in the PWMA\_CCER1 register.
4. Configure the input prescaler.

In this example, we want the capture to occur at every valid level transition, so the prescaler

Disabled (write IC1PS=00 in PWMA\_CCMR1 register).

5. Set CC1E=1 in the PWMA\_CCER1 register to allow the capture of the counter value into the capture register.
6. If required, enable the relevant interrupt request by setting the CC1IE bit in the PWMA\_IER register.

When an input capture occurs:

- ÿ When a valid level transition occurs, the value of the counter is transferred to the PWMA\_CCR1 register.
- The CC1IF flag is set. When at least 2 consecutive captures occur and CC1IF has not been cleared, CC1OF is also cleared.

Set to 1.

- ÿ An interrupt will be generated if the CC1IE bit is set.

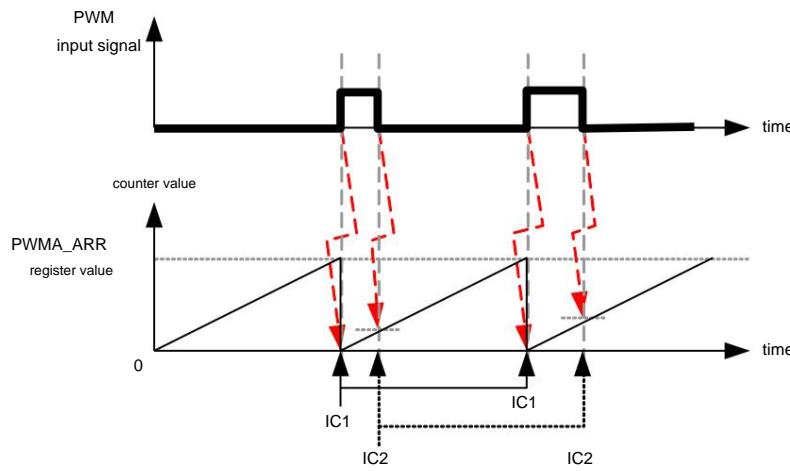
In order to handle capture overflow event (CC1OF bit), it is recommended to read data before reading out the recapture flag, this is to avoid loss Recapture information that may be generated after reading the capture overflow flag and before reading the data.

Note: An input capture interrupt can be generated by software by setting the corresponding CCiG bit in the PWMA\_EGR register.

#### **PWM** input signal measurement

This mode is a special case of input capture mode and operates the same as input capture mode with the following differences:

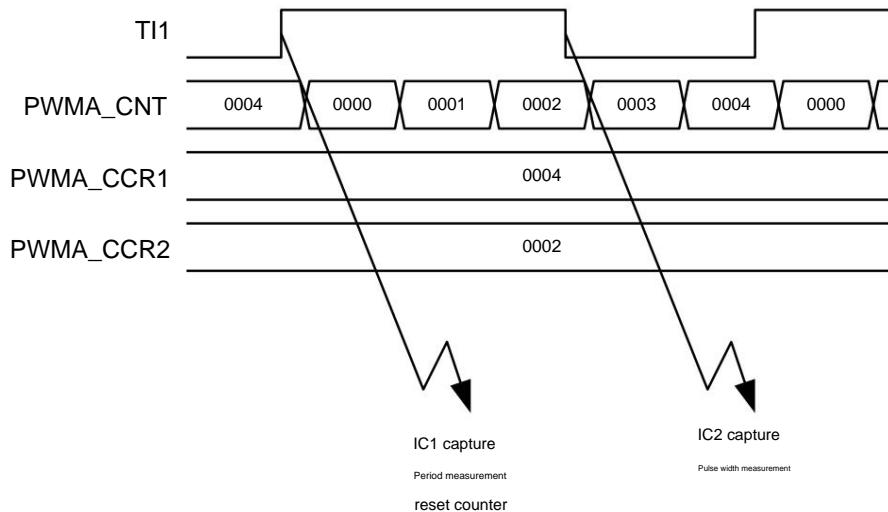
- ÿ Both ICi signals are mapped to the same T1i input.
- ÿ The polarities of the active edges of the two ICi signals are opposite.
- ÿ One of the T1iFP signals is used as the trigger input signal, and the trigger mode controller is configured to reset the trigger mode.



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 fMASTER and the value of the prescaler)

1. Select the valid input of PWMA\_CCR1: set CC1S=01 in the PWMA\_CCMR1 register (TI1 is selected).
2. Select the active polarity of TI1FP1 (used to capture data into PWMA\_CCR1 and clear the counter): set CC1P=0 (rising edge efficient).
3. Select valid input of PWMA\_CCR2: set CC2S=10 in PWMA\_CCMR2 register (select TI1FP2).
4. Select the valid polarity of TI1FP2 (capture data to PWMA\_CCR2): set CC2P=1 (falling edge valid).
5. Select a valid trigger input signal: Set TS=101 in the PWMA\_SMCR register (TI1FP1 is selected).
6. Configure the trigger mode controller to reset trigger mode: set SMS=100 in PWMA\_SMCR.
7. Enable capture: set CC1E=1 and CC2E=1 in the PWMA\_CCER1 register.

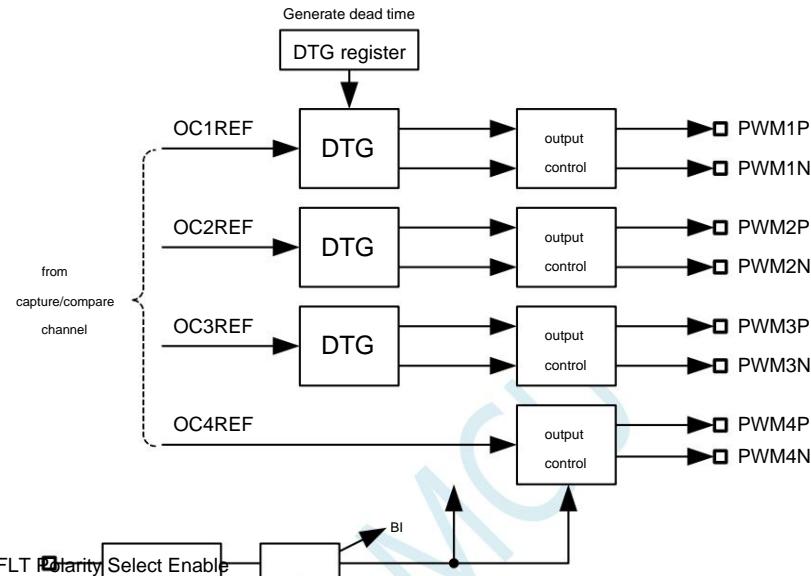
PWM input signal measurement example



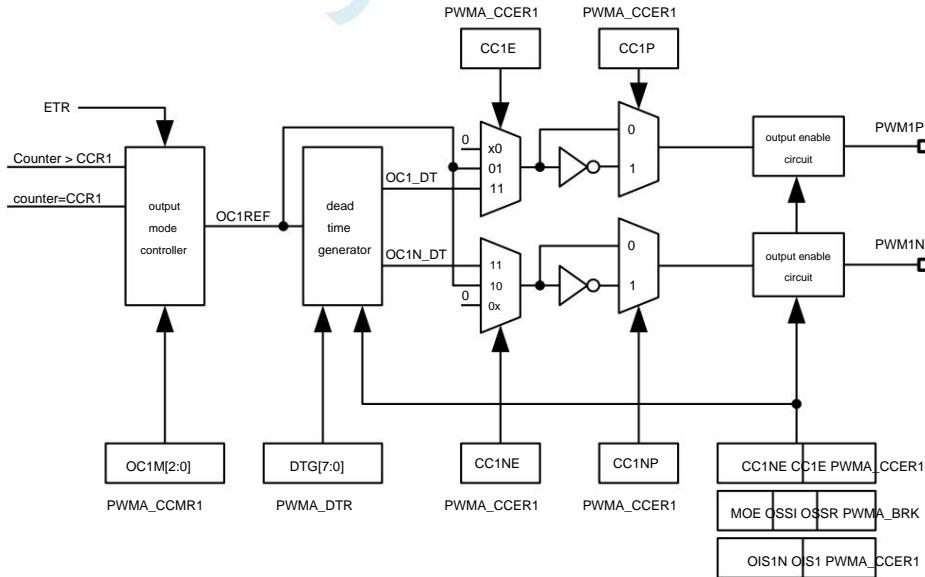
## 23.5.4 Output module

The output module generates an intermediate waveform for reference, called OCiREF (active high). The brake function and polarity are handled at the end of the module.

#### Block diagram of output module



**Channel 1 Detailed block diagram of output module with complementary outputs (similar to 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.

The OCiREF signal can be forced low by setting OCiM=101 in the PWMA\_CCMRi register. Set OCiM=100 in the PWMA\_CCMRi register to force the OCiREF signal low.

The output of OCi/OCiN is high or low depending on the CCiP/CCiNP polarity flags.

In this mode, the comparison between the PWMA\_CCRi shadow register and the counter is still in progress, and the corresponding flags are also modified.

The corresponding interrupt will still be generated.

## 23.5.6 Output Compare Mode

This mode is used to control an output waveform or to indicate that a given period of time has elapsed.

When the counter matches the contents of the capture/compare register, the following operations are performed:

- ÿ According to different output comparison modes, the corresponding OCi output signal:
  - ÿ keep unchanged (OCiM=000) ÿ set to active level (OCiM=001) ÿ set to inactive level (OCiM=010) ÿ flipped (OCiM=011) ÿ Set the flag bit in the interrupt status register (PWMA\_SR1 register CCiF bit in . ÿ An interrupt is generated

if the corresponding interrupt enable bit (bit CCiE in the PWMA\_IER register) is set.

The OCiM bits of the PWMA\_CCMRi register are used to select the output compare mode, while the CCiP bits of the PWMA\_CCMRi register are used to select the active and inactive level polarity. The OCiPE bit of the PWMA\_CCMRi register is used to select whether the PWMA\_CCRi register needs to use a preload register. In output compare mode, the update event UEV has no effect on the OCiREF and OCi outputs. The time precision is one count period of the counter. The output compare mode can also be used to output a single pulse.

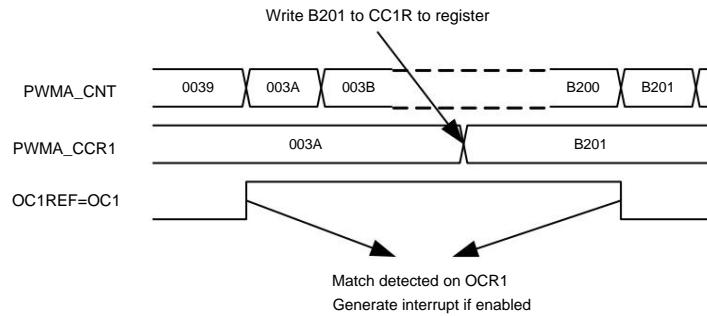
Configuration steps for output compare mode:

1. Select the counter clock (internal, external or prescaler). 2. Write the appropriate data to the PWMA\_ARR and PWMA\_CCRi registers. 3. If an interrupt request is to be generated, set the CCiE bit. 4.

Select the output mode steps: 1. Set OCiM=011, toggle the output of the OCiM pin when the counter matches CCRi 2. Set OCiPE = 0, disable the preload register 3. Set CCiP = 0, select the high level as Active level 4. Set CCiE = 1 to enable the output 5. Set the CEN bit of the PWMA\_CR1 register to start the counter

The PWMA\_CCRi register can be updated by software at any time to control the output waveform, provided the preload register is not used (OCiPE=0), otherwise the shadow register of PWMA\_CCRi can only be updated on the next update event.

Output compare mode, flips 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 a PWM. The OCiPE bit in the PWMA\_CCMRi register must be set to enable the corresponding preload register, or the ARPE bit in the PWMA\_CR1 register can be set to enable the auto-reload preload register (in up-counting mode or central symmetric mode).

Since the preload register can only be transferred to the shadow register when an update event occurs, the counter starts counting. All registers must be initialized by setting the UG bit of the PWMA\_EGR register before the data can be counted.

The polarity of OCi can be set by software in the CCiP bit in the PWMA\_CCERi register, which can be set to active high or active low. The output enable of OCi is controlled by a combination of 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 comparing, (depending on the count count direction of the counter) to determine whether PWMA\_CCRi>PWMA\_CNT or PWMA\_CNT>PWMA\_CCRi is satisfied.

Depending on the state of the CMS bit field in the PWMA\_CR1 register, the timer can generate either an edge-aligned PWM signal or a center-aligned PWM signal.

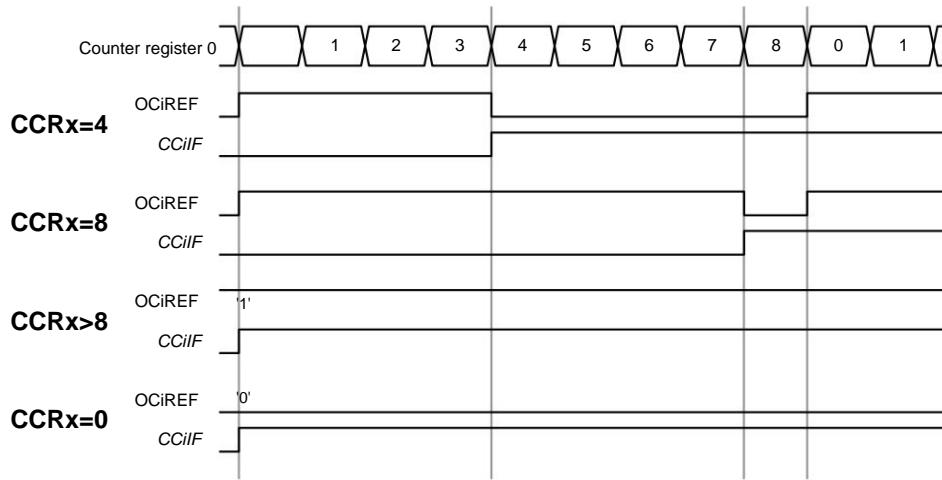
### PWM edge-aligned mode

count up configuration

Count up is performed when the DIR bit in the PWMA\_CR1 register is 0.

Below is an example of PWM Mode 1. When PWMA\_CNT<PWMA\_CCRi, the PWM reference signal OCiREF is high, otherwise it is low. OCiREF remains at '1' if the compare value in PWMA\_CCRi is greater than the auto-reload value (PWMA\_ARR). If the compare value is 0, OCiREF remains at '0'.

Edge-aligned, waveform in PWM mode 1 (ARR=8)



#### Countdown configuration

When the DIR bit of the PWMA\_CR1 register is 1, count down is performed.

In PWM mode 1, the reference signal OCiREF is low when PWMA\_CNT>PWMA\_CCRI, and high otherwise. if

The compare value in PWMA\_CCRI is greater than the auto-reload value in PWMA\_ARR, then OCiREF remains at '1'. cannot be produced in this mode Generate 0% PWM waveform.

#### PWM center-aligned mode

Center-aligned mode when the CMS bit in the PWMA\_CR1 register is not '00' (all other configurations numbers have the same effect).

The compare flag can be set when the counter counts up, counts down, or counts up and down, depending on the setting of the CMS bits.

1. The count direction bit (DIR) in the PWMA\_CR1 register is updated by hardware, do not modify it by software.

Some examples of center-aligned PWM waveforms are given below:

- ÿ PWMA\_ARR=8

- ÿ PWM mode 1

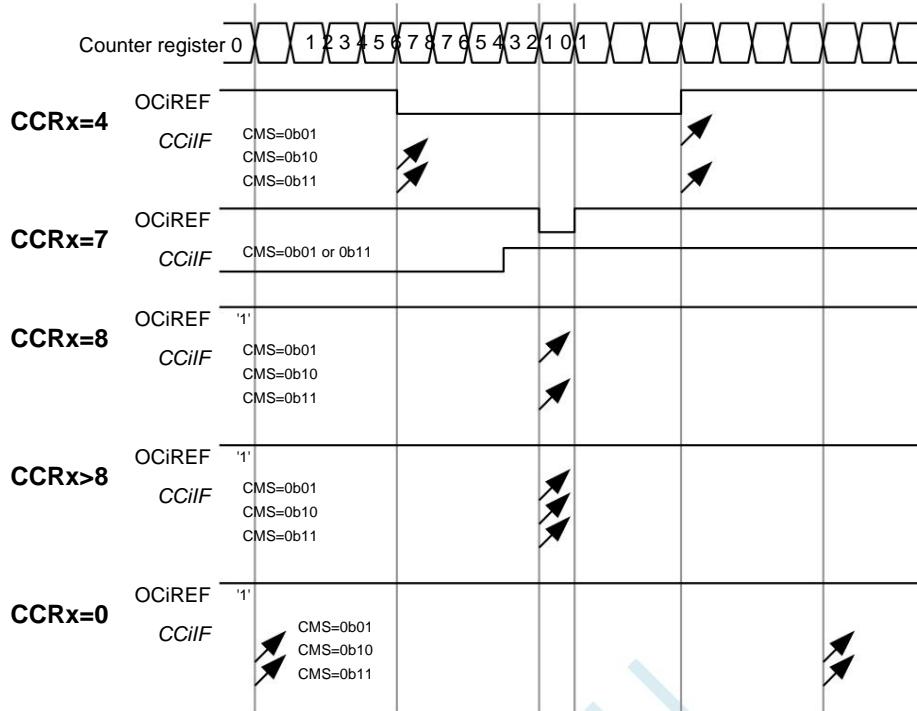
- ÿ The flag bit is set in the following three cases:

  - ÿ Only when the counter counts down (CMS=01)

  - ÿ only when the counter counts up (CMS=10)

  - ÿ when the counter counts up and down (CMS=11)

  - ÿ Center-aligned PWM waveform (ARR=8)



#### Single pulse mode

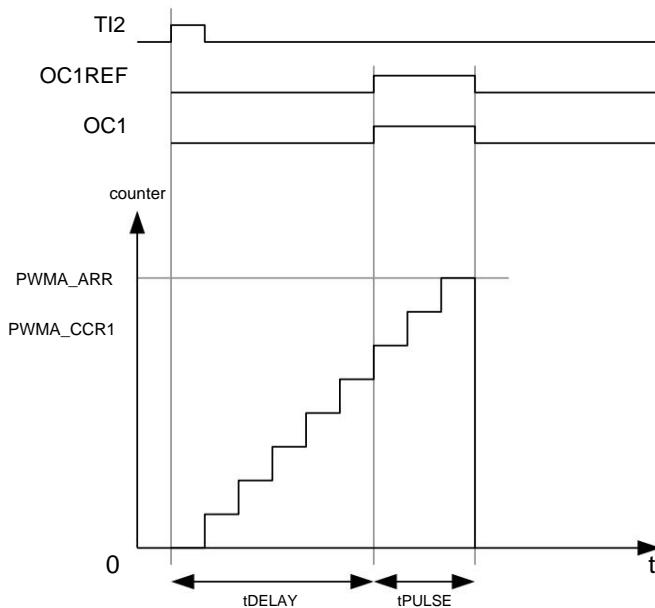
One-shot mode (OPM) is a special case of the aforementioned modes. This mode allows the counter to respond to a stimulus and after a sequence-controlled delay, a pulse with a controllable pulse width is generated.

The counter can be started by the clock/trigger controller to generate waveforms in output compare mode or PWM mode, set up. The OPM bit of the PWMA\_CR1 register will select the one-shot mode, where the counter automatically stops on the next update event UEV. A pulse can only be generated if the comparison value is different from the initial value of the counter. Before starting (when the timer is waiting to fire), it must be configured as follows:

Up counting mode: Counter CNT < CCRi  
Down counting mode: Counter

CNT > CCRi.

#### Single Pulse Mode Legend



For example, a delay of  $t_{DELAY}$  after the detection of a rising edge on the TI2 input generates a  $t_{PULSE}$  wide pulse on OC1.

Positive pulse: (Assume IC2 as trigger source for trigger 1 channel)

ÿ Set CC2S=01 in the PWMA\_CCMR2 register to map IC2 to TI2. ÿ Set CC2P=0 in the PWMA\_CCER1 register to enable IC2 to detect the rising edge. ÿ Set TS=110 in the PWMA\_SMCR register to make IC2 the trigger source (TRGI) of the clock/trigger controller. ÿ Set SMS=110 in PWMA\_SMCR register (trigger mode), IC2 is used to start the counter. The waveform of the OPM is determined by the value written to the compare register (taking into account the clock frequency and the counter prescaler). ÿ  $t_{DELAY}$  is defined by the value in the PWMA\_CCR1 register. •  $t_{PULSE}$  is defined by the difference between the autoload value and the compare value ( $PWMA\_ARR - PWMA\_CCR1$ ). ÿ Assuming that a waveform from 0 to 1 is to be generated when a compare 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 the PWMA\_CCMR1 register to enter PWM mode 2, according to It is necessary to selectively set OC1PE=1 in the PWMA\_CCMR1 register, set ARPE in the PWMA\_CR1 register, enable the preload register, then fill in the comparison value in the PWMA\_CCR1 register, fill in the autoload value in the PWMA\_ARR register, and set the UG bit to generate an update event and then wait for an external trigger event on the TI2.

In this example, the DIR and CMS bits in the PWMA\_CR1 register should be set low.

Since only one pulse is required, set OPM=1 in the PWMA\_CR1 register, at the next update event (when the Counting stops when the autoload value rolls over to 0).

#### OCx fast enable (special case)

In one-shot mode, edge detection on the Tli input sets the CEN bit to start the counter, and a comparison between the counter and the compare value produces a one-shot output. But these operations require a certain number of clock cycles, so it limits the minimum delay achievable  $t_{DELAY}$ .

If you want to output the waveform with the minimum delay, you can set the OCIFE bit in the PWMA\_CCMRi register. At this time, the OCREF (and OCx) is forced to respond directly to the excitation without relying on the comparison result, and the output waveform is the same as the waveform when the comparison matches. OCIFE is only available in channel configuration

Takes effect when set to PWMA and PWMB modes.

Complementary outputs and dead-time insertion

The PWMA can output two complementary signals, and can manage the instantaneous turn-off and turn-on of the output. This time is often called dead time. switch delay, etc.) to adjust the dead time.

Polarity can be independently selected for each output (main output OCi or complementary output OCiN) by configuring the CCiP and CCiNP bits in the PWMA\_CCERi register.

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, dead-band control is activated when transitioning to the IDLE state (MOE falls to 0).

Setting the CCiE and CCiNE bits at the same time will insert dead time, and if there is a braking circuit, also set the MOE bit. Each channel has an 8-bit dead-time generator. The reference signal OCiREF can generate two outputs OCi and OCiN.

Active if OCi and OCiN are high:

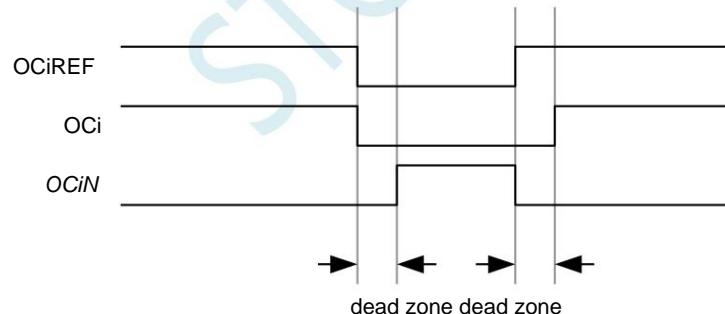
• The OCi output signal is the same as the reference signal, but its rising edge has a delay relative to the rising edge of the reference signal.

• The OCiN output signal is opposite to the reference signal, but its rising edge has a delay relative to the falling edge of the reference signal.

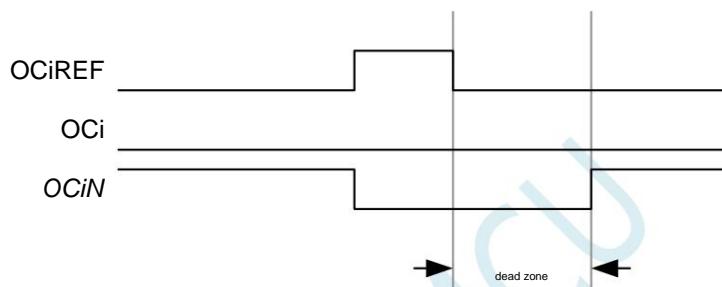
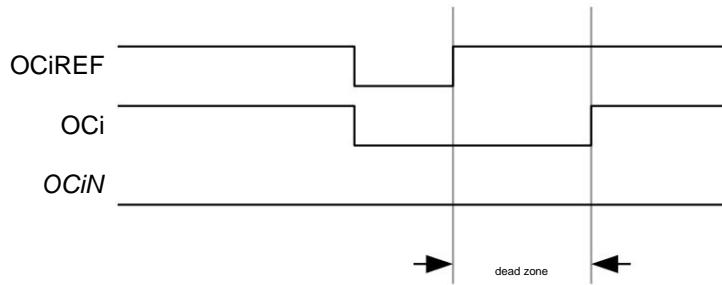
If the delay is greater than the currently active output width (OCi or OCiN), the corresponding pulse will not be generated.

The following figures show the relationship between the output signal of the dead-time generator and the current reference signal OCiREF. (Assuming CCiP=0, CCiNP=0, MOE=1, CCiE=1 and CCiNE=1)

Complementary outputs with dead-time insertion



Dead zone waveform delay greater than negative pulse



The dead-time delay is the same for each channel and is programmed by the DTG bits in the PWMA\_DTR register.

#### Redirect OCiREF to OCi or OCiN

In output mode (force output, output compare or PWM output), by configuring CCiE and CCiNE of the PWMA\_CCERi register CCiNE bit, OCiREF can be redirected to OCi or OCiN output.

This function can send a special waveform (eg PWM or static active level) on an output when the complementary output is at an inactive level. Another effect is to keep both outputs at inactive levels at the same time, or active levels at the same time (still complementary outputs with dead time).

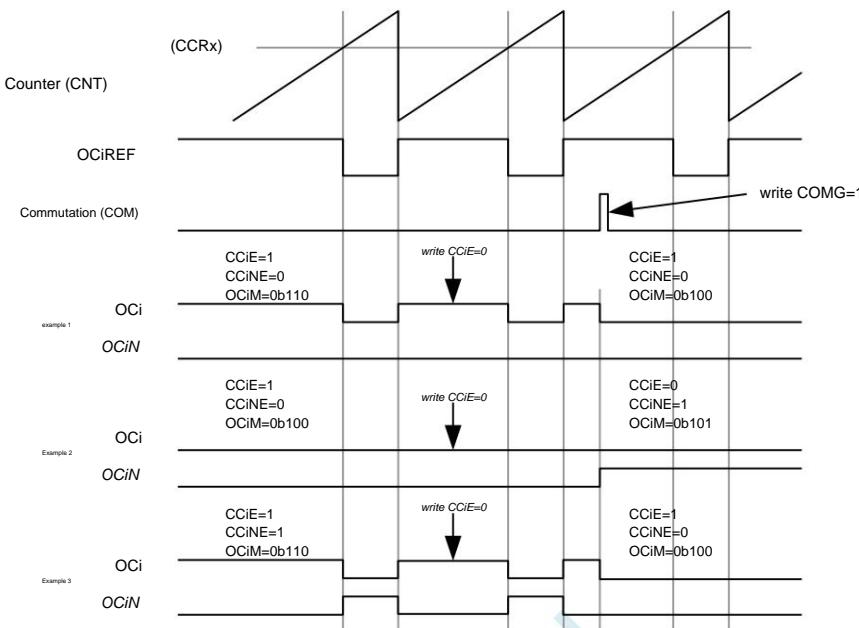
Note: When only OCiN is enabled (CCiE=0, CCiNE=1), it will not invert, but will take effect immediately when OCiREF goes high. For example, if CCiNP=0, then OCiN=OCiREF. On the other hand, when both OCi and OCiN are enabled (CCiE=CCiNE=1), OCi is valid when OCiREF is high; on the contrary, OCiN becomes valid when OCiREF is low.

#### Six-step PWM output for motor control

The preloaded bits are OCiM, CCiE and CCiNE when complementary outputs are required on a channel. These preload bits are transferred to shadow register bits when a COM commutation event occurs. This way you can pre-set the next step configuration and modify the configuration of all channels at the same time. COM can be generated by software by setting the COMG bit in the PWMA\_EGR register, or by hardware on the rising edge of TRGI.

The figure below shows the OCx and OCxN outputs in three different configurations when a COM event occurs.

Generate six-step PWM, use COM example (OSSR=1)



## 23.5.8 Using the Brake Function (PWMAFLT)

The brake function is often used in motor control. When using the brake function, according to the corresponding control bits (MOE, OSSI and OSSR bits), output enable signals and inactive levels are modified.

After system reset, the brake circuit is disabled and the MOE bit is low. The brake function can be enabled by setting the BKE bit in the PWMA\_BKR register. The polarity of the brake input signal can be selected by configuring the BKP bits in the same register. BKE and BKP can be modified simultaneously.

The falling edge of MOE can be asynchronous with respect to the clock block, so a resynchronization circuit is set up between the actual signal (applied on the output) and the synchronization control bits (in the PWMA\_BKR register). This resynchronization circuit creates a delay between the asynchronous signal and the synchronous signal. In particular, if MOE=1 is written when it is low, a delay (null instruction) must be inserted before reading it to read the correct value. This is because writing is asynchronous and reading is synchronous.

When a brake occurs (the selected level appears at the brake input), the following actions occur:

- The MOE bit is cleared asynchronously, placing the output in the inactive state, idle state, or reset state (selected by the OSSI bit). This feature is still valid when the MCU's oscillator is turned off. Once MOE=0, each output channel outputs the level set by the OISi bit in the PWMA\_OISR register. If OSSI=0,

Then the timer no longer controls the output enable signal, otherwise the output enable signal is always high.

When using complementary outputs:

The outputs are first placed in the reset state, ie inactive state (depending on polarity). This is an asynchronous operation, even if the timer does not

This function is also valid when the clock is turned off.

If the timer's clock still exists, the dead-time generator will be reactivated to drive the output port according to the level indicated by the OISi and OISiN bits after the dead-time. Even in this case, OCi and OCiN cannot be driven to active levels simultaneously. Note: Because of the resynchronization of the MOE, the dead time is a little longer than usual (about 2 clock cycles).

If the BIE bit in the PWMA\_IER register is set, an interrupt will be generated when the brake status flag (BIF bit in the PWMA\_SR1 register) is '1'.

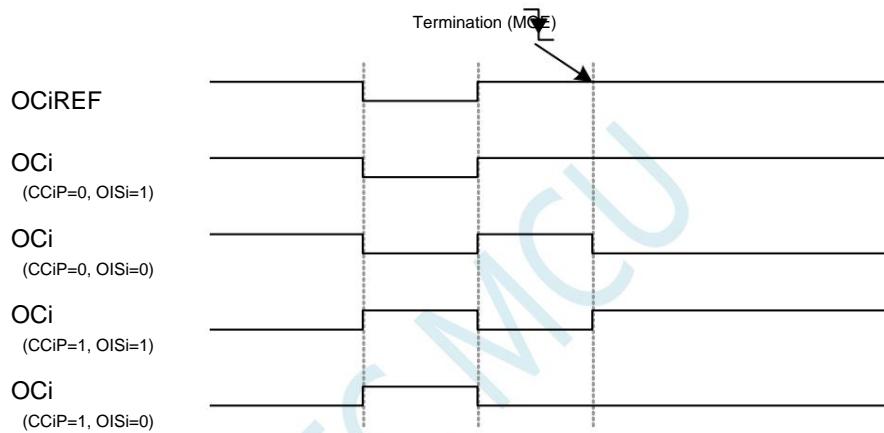
If the AOE bit in the PWMA\_BKR register is set, the MOE bit is automatically set at the next update event UEV. This can be used for example for waveform control, otherwise, MOE remains low until set to '1' again. This feature can be used for safety purposes, where you can connect the brake input to a power-driven alarm output, thermal sensor, or other safety device.

Note: Brake input is active level. Therefore, MOE cannot be set at the same time (either automatically or by software) when the brake input is active.

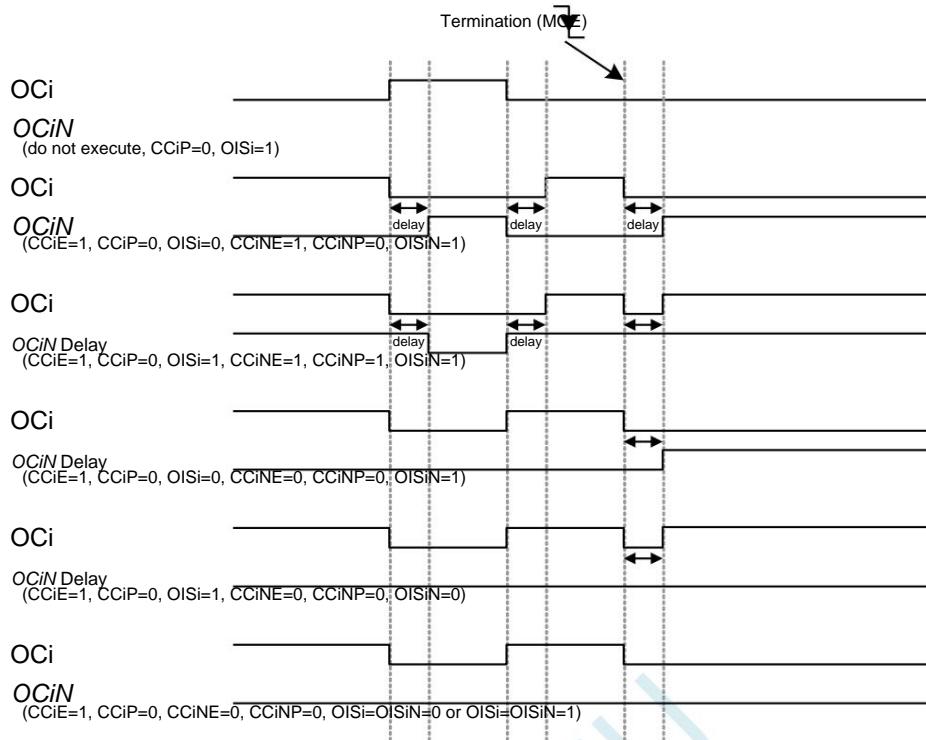
At the same time, the status flag BIF cannot be cleared.

Brake is generated by the BRK input (BKIN) whose active polarity is programmable and enabled or disabled by the BKE bit in the PWMA\_BKR register. In addition to brake input and output management, write protection is implemented in the brake circuit for application security. It allows the user to freeze several configuration parameters (OCi polarity and state when disabled, OCiM configuration, brake enable and polarity). The user can select one of three levels of protection through the LOCK bit in the PWMA\_BKR register. The LOCK bit field can only be modified once after MCU reset.

Brake response output (channel without complementary output)



Brake response output with complementary output (PWMA complementary output)



### 23.5.9 CLEARING OCiREF SIGNAL ON EXTERNAL EVENT

For a given channel, at the ETRF input (set the corresponding OCiCE bit in the PWMA\_CCMRi register to '1')

A high level can pull the OCiREF signal low, and the OCiREF signal will remain low until the next update event UEV occurs. This function only can be used in output compare mode and PWM mode, but not in forced mode.

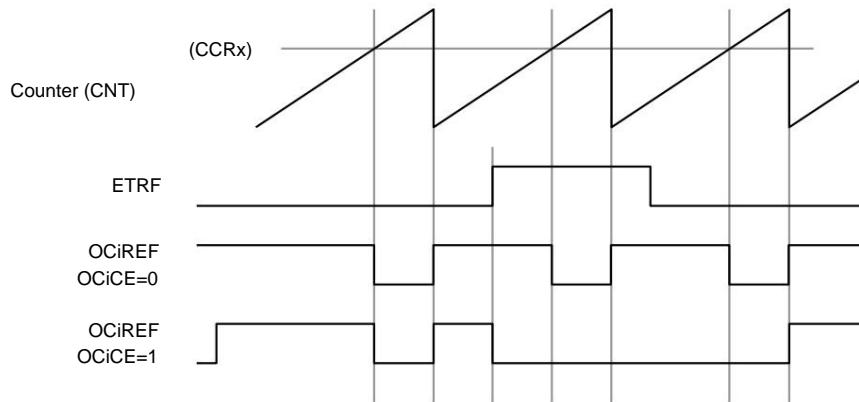
For example, the OCiREF signal can be connected to the output of a comparator for current control. At this time, the ETR must be configured as follows:

1. The external trigger 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. External trigger polarity (ETP) and external trigger filter (ETF) can be configured as required.

The figure below shows the behavior of the OCiREF signal for different OCiCE values when the ETRF input goes high.

In this example, the timer PWMA is placed in PWM mode.

ETR clears OCiREF of PWMA



### 23.5.10 Encoder Interface Mode

The encoder interface mode is generally used for motor control.

The way to select the encoder interface mode is:

- ÿ If the counter only counts on the edge of TI2, set SMS=001 in the PWMA\_SMCR register; ÿ If it only counts on the edge of TI1, set SMS=010; ÿ If the counter counts on the edge of TI1 and TI2 at the same time, set SMS=011 .

TI1 and TI2 polarity can be selected by setting the CC1P and CC2P bits in the PWMA\_CCER1 register;

Input filters can be programmed.

Two inputs TI1 and TI2 are used to interface the incremental encoder. Assuming the counter is already started (CEN=1 in the PWMA\_CR1 register), the counter counts on each valid transition on TI1FP1 or TI2FP2. TI1FP1 and TI2FP2 are the signals of TI1 and TI2 after passing through the input filter and polarity control. If there is no filtering and polarity conversion, then TI1FP1=TI1, TI2FP2=TI2. According to the transition sequence of the two input signals, a count pulse and a direction signal are generated. According to the transition sequence of the two input signals, the counter counts up or down, and the hardware sets the DIR bit of the PWMA\_CR1 register accordingly. A transition on either input (TI1 or TI2) recalculates the DIR bit whether the counter counts on TI1, on TI2, or on both TI1 and TI2.

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 autoload value of the PWMA\_ARR register (either 0 to ARR or ARR to 0, depending on the direction). So PWMA\_ARR must be configured before starting counting. In this mode the capture, comparator, prescaler, repeat counter, trigger output features, etc. still work as usual. Encoder mode and external clock mode 2 are not compatible and therefore cannot operate simultaneously.

In encoder interface mode, the counter is automatically modified according to the speed and direction of the incremental encoder, so the content of the counter always refers to the position of the encoder, and the count direction corresponds to the direction of rotation of the connected sensor.

The following table lists all possible combinations (assuming that TI1 and TI2 are not switched at the same time).

Relationship between counting direction and encoder signal

active edge	Relative signal level (TI1FP1 corresponds to TI2, TI2FP2 corresponds to TI1)	TI1FP1 signal		TI2FP2 signal	
		up, down	up, down		
Counts only on TI1	high	Count down, count up, don't count, don't count	count up,		
	and low	count down, don't count, don't count			

Counts only at TI2		don't count, don't count, count up, count down			
		don't count, don't count, count down, count up			
Count on TI1 and TI2		count down	count up	count up	count down
	high and low	count up	count down	count down	count up

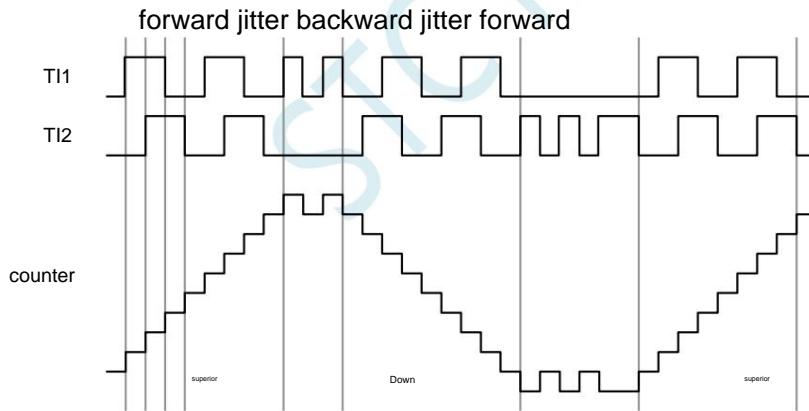
An external incremental encoder can be connected directly to the MCU without external interface logic. However, generally using a comparator will encode the differential output of the device is converted into a digital signal, which greatly increases the noise immunity. The third signal output by the encoder represents the mechanical zero point, it can be connected to an external interrupt input and trigger a counter reset.

The following is an example of the operation of a counter, showing the generation and direction control of the count signal. It also shows that when both edges are selected, how input jitter is suppressed; jitter may occur when the sensor is positioned close to a transition point. In this example, we take

The configuration is as follows:

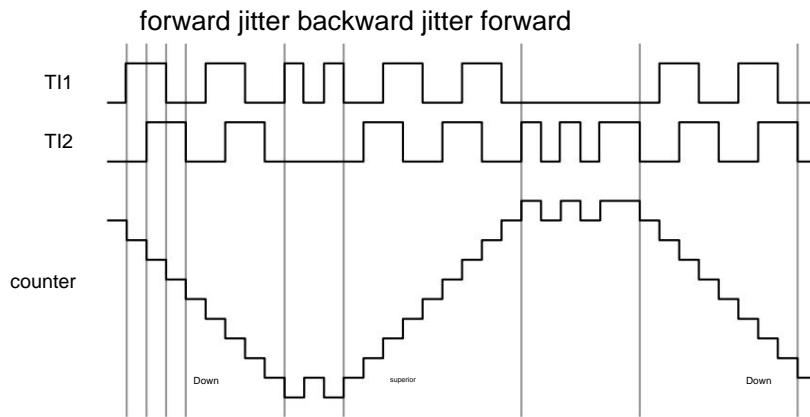
- ÿ CC1S=01 (PWMA\_CCMR1 register, IC1FP1 is mapped to TI1)
- ÿ CC2S=01 (PWMA\_CCMR2 register, IC2FP2 is mapped to TI2)
- ÿ CC1P=0 (PWMA\_CCER1 register, IC1 is not inverted, IC1=TI1)
- ÿ CC2P=0 (PWMA\_CCER1 register, IC2 is not inverted, IC2=TI2)
- ÿ SMS=011 (PWMA\_SMCR register, all inputs are valid on rising and falling edges).
- ÿ CEN=1 (PWMA\_CR1 register, counter enable)

#### Example of Counter Operation in Encoder Mode



The following figure is an example of the operation of the counter when the polarity of IC1 is reversed (CC1P=1, other configurations are the same as the above example)

#### IC1 Inverting Encoder Interface Mode Example



Provides information on the current position of the sensor when the timer is configured in encoder interface mode. Use another configuration in capture mode. The timer measures the interval between two encoder events and can obtain dynamic information (speed, acceleration, deceleration). Indicates mechanical zero. The encoder output can be used for this purpose. Depending on the interval between two events, the counter can be read out at certain time intervals. If possible, if you can, you can latch the value of the counter into the third input capture register (the capture signal must be periodic and can be controlled by another timer generated).

## 23.6 Interrupts

PWMA/PWMB each have 8 interrupt request sources:

- ÿ Brake interruption
- ÿ Trigger interrupt
- ÿ COM event interrupt
- ÿ Input capture/output compare 4 interrupt
- ÿ Input capture/output compare 3 interrupt
- ÿ Input capture/output compare 2 interrupt
- ÿ Input capture/output compare 1 interrupt
- ÿ Update event interrupt (eg: counter overflow, underflow and initialization)

To use the interrupt feature, for each interrupt channel used, set the corresponding interrupt in the PWMA\_IER/PWMB\_IER register

Enable bit: BIE, TIE, COMIE, CCIE, UIE bit. By setting the corresponding

The above interrupt sources can also be generated by software.

## 23.7 PWMA/PWMB register description

### 23.7.1 Function pin switching (PWMx\_PS)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0	
PWMA_PS	7E	FEB2H	C4PS[1:0]			C3PS[1:0]	C2PS[1:0]			C1PS[1:0]
PWMB_PS	7E	FEB6H	C8PS[1:0]			C7PS[1:0]	C6PS[1:0]			C5PS[1:0]

C1PS[1:0]: Advanced PWM channel 1 output pin selection 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 selection 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] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

C3PS[1:0]: Advanced PWM channel 3 output pin selection 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 selection 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 selection 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 selection 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 selection 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 selection 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 (PWMrx\_ETRPS)

symbol	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 select bits 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 of advanced PWMB

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

BRKAPS: Brake foot PWMFLT selection bit for advanced PWMA

BRKAPS	PWMFLT
0	P3.5
1	Comparator output

BRKBPS: Brake foot PWMFLT2 selection bit of advanced PWMB

BRKBPS	PWMFLT2
0	P3.5
1	Comparator output

### 23.7.3 Output Enable Register (PWMrx\_ENO)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0	
PWMA_ENO	7EFEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P

PWMB_ENO 7EFFEB5H		-	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 (PWMy\_IOAUX)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX 7EFFEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P	
PWMB_IOAUX 7EFFEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P	

AUX8P: PWM8 output additional control bit

0: The output of PWM8 is directly controlled by ENO8P

1: The output of PWM8 is jointly controlled by ENO8P and PWMB\_BKR

AUX7P: PWM7 output additional control bit

0: The output of PWM7 is directly controlled by ENO7P

1: The output of PWM7 is jointly controlled by ENO7P and PWMB\_BKR

AUX6P: PWM6 output additional control bit

0: The output of PWM6 is directly controlled by ENO6P

1: The output of PWM6 is jointly controlled by ENO6P and PWMB\_BKR

AUX5P: PWM5 output additional control bit

0: The output of PWM5 is directly controlled by ENO5P

1: The output of PWM5 is jointly controlled by ENO5P and PWMB\_BKR

AUX4N: PWM4N output additional control bit

0: The output of PWM4N is directly controlled by ENO4N

1: The output of PWM4N is jointly controlled by ENO4N and PWMA\_BKR

AUX4P: PWM4P output additional control bit

0: The output of PWM4P is directly controlled by ENO4P

1: The output of PWM4P is jointly controlled by ENO4P and PWMA\_BKR

AUX3N: PWM3N output additional control bit

0: The output of PWM3N is directly controlled by ENO3N

1: The output of PWM3N is jointly controlled by ENO3N and PWMA\_BKR

AUX3P: PWM3P output additional control bit

0: The output of PWM3P is directly controlled by ENO3P

1: The output of PWM3P is jointly controlled by ENO3P and PWMA\_BKR

AUX2N: PWM2N output additional control bit

0: The output of PWM2N is directly controlled by ENO2N

1: The output of PWM2N is jointly controlled by ENO2N and PWMA\_BKR

AUX2P: PWM2P output additional control bit

0: The output of PWM2P is directly controlled by ENO2P

1: The output of PWM2P is jointly controlled by ENO2P and PWMA\_BKR

AUX1N: PWM1N output additional control bit

0: The output of PWM1N is directly controlled by ENO1N

1: The output of PWM1N is controlled by ENO1N and PWMA\_BKR

AUX1P: PWM1P output additional control bit

0: The output of PWM1P is directly controlled by ENO1P

1: The output of PWM1P is jointly controlled by ENO1P and PWMA\_BKR

## 23.7.5 Control Register 1 (PWMy\_CR1)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR1	7EFEC0H ARPEA	CMSA[1:0]	DIRA OPMA URSA UDISA CENA					
PWMB_CR1	7EE0H ARPEB	CMSB[1:0]	DIRB OPMB URSB UDISB CENB					

ARPEn: auto preload enable bit (n=A,B)

0: PWMy\_ARR register is not buffered, it can be written directly

1: PWMy\_ARR register is buffered by preload buffer

CMSn[1:0]: select alignment mode (n=A,B) alignment

CMSn[1:0]	mode	
00	Description Edge-aligned mode	counter counts up or down depending on the direction bit (DIR)
01	Center Align Mode 1	The counter counts up and down alternately. The output compare interrupt flag of a channel configured as an output is set only when the counter is counting down.
10	Center Alignment Mode 2	The counter counts up and down alternately. The output compare interrupt flag of a channel configured as an output is set only when the counter is counting up. The counter counts up and
11	Center Alignment Mode 3	down alternately. The Output Compare Interrupt Flag bit of a channel configured as an output is set both when the counter counts up and when it counts down.

Note 1: When the counter is on (CEN=1), transition from edge-aligned mode to center-aligned mode is not allowed. Note 2:

In center-aligned mode, encoder mode (SMS=001, 010, 011) must be disabled.

DIRn: Counting direction of the counter (n= A, B) 0:

The counter counts up; 1: The counter counts down. Note: This bit is read only when the counter is configured in center-aligned mode or encoder mode.

OPMn: One-shot mode (n= A,B) 0: The

counter does not stop when an update event occurs; 1:

When the next update event occurs, clear the CEN bit and the counter stops.

URSn: Update request source (n= A,B) 0: If

UDIS enables generation of update events, one of the following events generates an update interrupt:

- ÿ Register is updated (counter overflow/underflow) ÿ UG bit is set by software ÿ Update generated by clock/trigger controller

1: If UDIS enables the generation of update events, the update interrupt is generated and UIF is set to 1 only when the following events occur: ÿ Registers are updated (counter overflow/underflow)

UDISn: Update disabled (n= A,B) 0:

Generate update (UEV) event once the following events occur: ÿ Counter

- overflow/underflow ÿ Generate software update event ÿ Clock/trigger mode controller generated hardware reset is buffered

Registers are loaded with their preloaded values. 1: No update event is generated, shadow registers (ARR, PSC, CCRx) retain their values. If the UG bit is set or the clock/trigger controller issues a hardware reset, the counter and prescaler are reinitialized.

CENn: Enable counter (n= A, B) 0: Disable

counter; 1: Enable counter. Note:

External clock, gated mode and

encoder mode can only work after the CEN bit is set by software. However, the trigger mode can automatically

The CEN bit is set by hardware.

## 23.7.6 Control register 2 (PWMAx\_CR2), and real-time trigger ADC

symbol	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 group of PWM/PWMA

0: PWM1P input pin is connected to TI1 (input of digital filter);

1: The PWM1P, PWM2P and PWM3P pins are XORed and connected to TI1 of the first group of PWMs.

TI5S: TI5 selection for the second group of PWM/PWMB

0: PWM5 input pin is connected to TI5 (input of digital filter);

1: PWM5, PWM6 and PWM7 pins are XORed and connected to TI5 of the second group of PWMs.

MMSA[2:0]: Master mode selection

MMSA[2:0] Master mode		illustrate
000	reset	The UG bit of the PWMA_EGR register is used as a trigger output (TRGO). like If the trigger input (the clock/trigger controller is configured in reset mode) generates a reset bit, the signal on TRGO will have a delay relative to the actual reset
001	Enable	The counter enable signal is used as a trigger output (TRGO). It is used to start Move the ADC so that the control enables the ADC for a period of time. Counter enable signal is the logical OR of the trigger input signal via the CEN control bit and gated mode produce. Unless master/slave mode is selected, when the counter enable signal is controlled by There is a delay on the TRGO when the input is triggered.  Note: When you need to use PWM to trigger ADC conversion, you need to set ADC_CONTR first ADC_POWER, ADC_CHS and ADC_EPWMT in the register, when PWM When the TRGO internal signal is generated, the system will automatically set ADC_START to start AD conversion. For details, please refer to the sample program "Starting with CEN of PWM PWMA timer, trigger ADC in real time"
010	Update	Update event is selected as trigger output (TRGO)
011	compare pulse	Once a capture or a compare is successful, when the CC1IF flag is set to 1 , the trigger output sends a positive pulse (TRGO)
100	Compare OC1REF	signal is used as trigger output (TRGO)
101	Compare OC2REF	signal is used as trigger output (TRGO)
110	Compare OC3REF	signal is used as trigger output (TRGO)
111	Compare OC4REF	signal is used as trigger output (TRGO)

MMSB[2:0]: Master mode selection

MMSB[2:0] Master mode		illustrate
000	reset	The UG bit of the PWMB_EGR register is used as a trigger output (TRGO). like If the trigger input (the clock/trigger controller is configured in reset mode) generates a reset bit, the signal on TRGO will have a delay relative to the actual reset
001	Enable	The counter enable signal is used as a trigger output (TRGO). It is used to start to drive multiple PWMs so that the control enables slave PWMs for a period of time. counter make

		The enable signal is through the CEN control bit and the trigger input signal in gated mode Logical OR produces. Unless master/slave mode is selected, when the counter enable signal When controlled by the trigger input, there is a delay on the TRGO.
010	Update	Update event is selected as trigger output (TRGO)
011	compare pulse	Once a capture or a comparison succeeds, when the CC5IF flag is set to 1 , the trigger output sends a positive pulse (TRGO)
100	Compare	OC5REF signal is used as trigger output (TRGO)
101	Compare	OC6REF signal is used as trigger output (TRGO)
110	Compare	OC7REF signal is used as trigger output (TRGO)
111	Compare	OC8REF signal is used as trigger output (TRGO)

Note: Only the **TRGO** of the first group of **PWM** can be used to trigger the **ADC**

Note: Only the **TRGO** of the second group of **PWM** can be used for the **ITR2** of the first group of **PWM**

COMSn: Update control selection for capture/compare control bits (n=A,B)

0: When CCPCn=1, these control bits are only updated when the COMG bit is set

1: When CCPCn=1, these control bits are updated only when the COMG bit is set or a rising edge occurs on TRGI

CCPCn: Capture/Compare Preload Control Bits (n= A,B)

0: CCIE, CCINE, CCIP, CCINP and OCIM bits are not preloaded

1: CCIE, CCINE, CCIP, CCINP, and OCIM bits are preloaded; when this bit is set, they only work when COMG is set bit is updated.

Note: This bit only works on channels with complementary outputs.

### 23.7.7 Slave Mode Control Register (PWMy\_SMCR)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SMCR 7EFFEC2H MSMA		TSA[2:0]		-		SMSA[2:0]		
PWMB_SMCR 7EFFEE2H MSMB		TSB[2:0]		-		SMSB[2:0]		

MSMn: Master/Slave mode (n= A,B)

0: No effect

1: Events on trigger input (TRGI) are delayed to allow perfect synchronization between PWMy 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 when SMS=000 to avoid false edge detection when changing.

#### SMSA[2:0]: Clock/Trigger/Slave Mode Select

SMSA[2:0]	Function	illustrate
000 Internal	clock mode If CEN=1, the prescaler is directly driven by the internal clock	
001	Encoder mode 1 According to the level of TI1FP1, the counter counts up/down on the edge of TI2FP2	
010	Encoder mode 2 According to the level of TI2FP2, the counter counts up/down on the edge of TI1FP1	
011	Encoder Mode 3	Depending on the level of the other input, the counter is on the edge of TI1FP1 and TI2FP2 Count up/down
100	reset mode	Reinitialize the counter on the rising edge of the selected trigger input (TRGI), and generate a signal to update the register
101	gated mode	When the trigger input (TRGI) is high, the clock to the counter is turned on. once touched If the send input goes low, the counter is stopped (but not reset). start of the counter Movement and stop are controlled
110	trigger mode	The counter is started (but not reset) on the rising edge of trigger input TRGI, only The start of the counter is controlled
111 External	clock mode 1	The rising edge of the selected trigger input (TRGI) drives the counter. Note: Do not use if TI1F_ED is selected as trigger input (TS=100) Gated mode. This is because TI1F_ED only outputs a pulses, however gated mode is to check the level of the trigger input

#### SMSB[2:0]: Clock/Trigger/Slave Mode Select

SMSB[2:0] function		illustrate
000 Internal	clock mode If CEN=1, the prescaler is driven directly by the internal clock	
001 Encoder	mode 1 According to the level of TI5FP5, the counter counts up/down on the edge of TI6FP6	
010 Encoder	mode 2 According to the level of TI6FP6, the counter counts up/down on the edge of TI5FP5	
011 Encoder	mode 3	Depending on the level of the other input, the counter counts towards the edge of the TI5FP5 and TI6FP6 up/down count
100	reset mode	Reinitialize the counter on the rising edge of the selected trigger input (TRGI), and generate a signal to update the register
101	gated mode	When the trigger input (TRGI) is high, the clock to the counter is turned on. once triggered The input goes low, the counter is stopped (but not reset). start of the counter

		and stops are controlled
110	trigger mode	The counter is started (but not reset) on the rising edge of trigger input TRGI, only the start of the counter is controlled
111 External clock mode 1		The rising edge of the selected trigger input (TRGI) drives the counter. Note: Do not use if TI5F_ED is selected as trigger input (TS=100) Gated mode. This is because TI5F_ED only outputs a pulses, however gated mode is to check the level of the trigger input

### 23.7.8 External Trigger Register (PWMx\_ETR)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETR 7Efec3H ETP1	ECEA			ETPSA[1:0]	ETFA[3:0]			
PWMB_ETR 7Efef3H ETP2	ETPn: Polarity	ECEB		ETPSB[1:0]	ETFB[3:0]			

of external trigger ETR (n= A,B)

0: Active at high level or rising edge

1: Active at low level or falling edge

ECEn: External clock enable (n= A,B)

0: Disable external clock mode 2

1: Enable external clock mode 2, the clock of the counter is the valid edge of ETRF.

Note 1: Setting ECE to 1 has the same effect as selecting external clock mode 1 connecting TRGI to ETRF (PWMn\_SMCR register , SMS=111, TS=111).

Note 2: External Clock Mode 2 can be used simultaneously with the following modes: Trigger Standard Mode; Trigger Reset Mode; Trigger Gated Mode. but

Yes, TRGI must never be connected to ETRF at this time (TS cannot be 111 in the 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 maximum frequency of the external trigger signal EPRP cannot exceed fMASTER/4. A prescaler can be used to reduce

The frequency of ETRP, which is useful when the frequency of EPRP is 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 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
0110	twenty four	1110	192
0111	32	1111	256

### 23.7.9 Interrupt Enable Register (PWMx\_IER)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
PWMA_IER	7EF	EC4H	BIEA		TIEA COMIEA CC4IE	CC3IE CC2IE			CC1IE UIEA
PWMB_IER	7EE	EE4H	BIEB		TIEB COMIEB CC8IE	CC7IE CC6IE			CC5IE UIEB

BIE<sub>n</sub>: Enable brake interrupt (n= A,B)

0: Disable brake interruption;

1: Enable brake interruption.

TIE: trigger interrupt enable (n= A, B)

0: Disable trigger interrupt;

1: Enable trigger interrupt.

COMIE: Enable COM Interrupt (n= A,B)

0: disable COM interrupt;

1: Enable COM interrupt.

CCnIE: Enable capture/compare n interrupts (n=1,2,3,4,5,6,7,8)

0: disable capture/compare n interrupt;

1: Enable capture/compare n interrupts.

UIEn: Allow update interrupts (n=A,B)

0: Disable update interrupt;

1: Allow update interruption.

### 23.7.10 Status Register 1 (PWMx\_SR1)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0		
PWMA_SR1	7EF	E5H	BIFA		TIFA COMIF	IFA CC4IF		CC3IF	CC2IF	CC1IF	UIFA
PWMB_SR1	7EE	EE5H	BIFB		TIFB COMIFB	CC8IF		CC7IF	CC6IF	CC5IF	UIFB

BIF<sub>n</sub>: Brake interrupt flag. This bit is set by hardware once the brake input is active. This bit can be cleared by software if the brake input is inactive

0. (n=A,B)

0: No brake event is generated

1: Active level detected on brake input

TIF<sub>n</sub>: Trigger interrupt flag. This bit is set by hardware when a trigger event occurs. Cleared by software. (n=A,B)

0: No trigger event is generated

1: Trigger an interrupt and wait for a response

COMIF<sub>n</sub>: COM Interrupt Flag. This bit is set by hardware once a COM event occurs. Cleared by software. (n=A,B)

0: No COM event is generated

1: COM interrupt waiting for response

CC8IF: capture/compare 8 interrupt flags, refer to CC1IF description

CC7IF: capture/compare 7 interrupt flag, refer to CC1IF description

CC6IF: capture/compare 6 interrupt flag, refer to CC1IF description

CC5IF: capture/compare 5 interrupt flag, refer to CC1IF description

CC4IF: capture/compare 4 interrupt flag, refer to CC1IF description

CC3IF: capture/compare 3 interrupt flag, refer to CC1IF description

CC2IF: capture/compare 2 interrupt flag, refer to CC1IF description

CC1IF: Capture/Compare 1 Interrupt Flag.

If channel CC1 is configured in output mode:

This bit is set by hardware when the counter value matches the compare value, except in centrosymmetric mode. It is cleared by software.

0: no match occurs;

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 starts from 0

Count up to ARR-1, then count down to 1 by ARR). Therefore, for all SMS bit values, both values are unmarked

remember. However, if CCR1>ARR, CC1IF is set when CNT reaches the ARR value.

If channel CC1 is configured in input mode:

This bit is set by hardware when a capture event occurs, it is cleared by software or by reading PWMA\_CCR1L.

0: No input capture is generated

1: Counter value has been captured to PWMA\_CCR1

UIFn: Update Interrupt Flag This bit is set by hardware when an update event occurs. It is cleared by software. (n=A,B)

0: No update event is generated

1: The update event is waiting for a response. This bit is set by hardware when the register is updated

ÿ If UDIS=0 in PWMn\_CR1 register, when the counter overflows or underflows

ÿ If UDIS=0 and URS=0 in the PWMn\_CR1 register, when the UG bit in the PWMn\_EGR register is set, the software counts

When the device CNT is reinitialized

ÿ If UDIS=0, URS=0 in PWMn\_CR1 register, when the counter CNT is reinitialized by the trigger event

### 23.7.11 Status Register 2 (PWMr\_SR2)

symbol	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 Flags. See CC1OF description.

CC7OF: Capture/Compare 7 Repeat Capture Flags. See CC1OF description.

CC6OF: Capture/Compare 6 Repeat Capture Flags. See CC1OF description.

CC5OF: Capture/Compare 5 Repeat Capture Flags. See CC1OF description.

CC4OF: Capture/Compare 4 Repeat Capture Flags. See CC1OF description.

CC3OF: Capture/Compare 3 Repeat Capture Flags. See CC1OF description.

CC2OF: Capture/Compare 2 Repeat Capture Flags. See CC1OF description.

CC1OF: Capture/Compare 1 Repeat Capture Flag. This flag can be set by hardware only when the corresponding channel is configured for input capture. write 0 to

Clear this bit.

0: No duplicate capture is 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 (PWMr\_EGR)

symbol	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 by software to generate a brake event and is automatically cleared by hardware (n=A,B)

0: no action

1: Generate a brake event. At this time MOE=0, BIF=1, if the corresponding interrupt is enabled (BIE=1), the corresponding interrupt will be generated

TGn: Generate trigger event. This bit is set by software to generate a trigger event and is automatically cleared by hardware (n= A,B)

0: no action

1: TIF=1, if the corresponding interrupt is enabled (TIE=1), the corresponding interrupt will be generated

COMGn: Capture/compare event, generate control update. This bit is set by software and cleared by hardware automatically (n=A,B)

0: no action

1: CCPC=1, allows to update CCIE, CCINE, CCiP, CCiNP, OCIM bits.

Note: This bit is only valid for channels with complementary outputs

CC8G: Generate capture/compare 8 events. Refer to CC1G description

CC7G: Generate capture/compare 7 event. Refer to CC1G description

CC6G: Generate capture/compare 6 events. Refer to CC1G description

CC5G: Generate capture/compare 5 events. Refer to CC1G description

CC4G: Generate capture/compare 4 events. Refer to CC1G description

CC3G: Generate capture/compare 3 events. Refer to CC1G description

CC2G: Generate capture/compare 2 event. Refer to CC1G description

CC1G: Generate capture/compare 1 event. Generate a capture/compare 1 event. This bit is set by software to generate a capture/compare event,

Automatically cleared to 0 by hardware.

0: no action;

1: Generate a capture/compare event on channel CC1.

If channel CC1 is configured as output: set CC1IF=1, if the corresponding interrupt is enabled, the corresponding interrupt will be generated.

If channel CC1 is configured as input: the current counter value is captured to the PWMA\_CCR1 register, set CC1IF=1, if enabled

Corresponding interrupts will generate corresponding interrupts. If CC1IF is already 1, set CC1OF=1.

UGn: An update event is generated. This bit is set to 1 by software and automatically cleared to 0 by hardware. (n=A,B)

0: no action;

1: Reinitialize the counter and generate an update event.

Note that the prescaler counter is also cleared to 0 (but the prescaler factor does not change). If in centrosymmetric mode or DIR=0 (count up

number), the counter is cleared to 0; if DIR=1 (counting down), the counter takes the value of PWMn\_ARR.

### 23.7.13 Capture/Compare Mode Register 1 (PWMy\_CCMR1)

A channel can be used for input (capture mode) or output (compare mode), the direction of the channel is defined by the corresponding CCnS bits. This register other

The roles of bits differ in input and output modes. OCxx describes the function of the channel in output mode, ICxx describes the function of the channel in input mode function under the formula. Therefore, it must be noted that the function of the same bit in output mode and input mode is different.

The channel is configured in compare output mode

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1 7	EFEC8H	OC1CE	OC1M[2:0]				OC1PE	OC1FE	CC1S[1:0]
PWMB_CCMR1 7	EFEE8H	OC5CE	OC5M[2:0]				OC5PE	OC5FE	CC5S[1:0]

OCnCE: Output compare n clear enable. This bit enables 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 ETRF input;

1: OCnREF=0 once ETRF input high level is detected.

OCnM[2:0]: Output compare n mode. The 3 bits define the action of the output reference signal OCnREF, and OCnREF determines the OCn

value of . OCnREF is active high, and the active level of OCn depends on the CCnP bit. (n=1,5)

OCnM[2:0]	Mode	illustrate
000	Freeze	Comparison between PWMn_CCR1 and PWMn_CNT has no effect on OCnREF
001	Set channel n on match When PWMn_CCR1=PWMn_CNT, OCnREF output is high	

	Output is active level	
010	When matching, set channel n's output is invalid level	When PWMn_CCR1=PWMn_CNT, OCnREF output is low
011	flip	When PWMn_CCR1=PWMn_CNT, flip OCnREF
100	Forced to inactive level Forces OCnREF low	
101	Force Active Level Force OCnREF High	
110	PWM mode 1	When counting up, when PWMn_CNT<PWMn_CCR1 OCnREF output is high, otherwise OCnREF output is low When counting down, when PWMn_CNT>PWMn_CCR1 OCnREF output is low, otherwise OCnREF output is high
111	PWM mode 2	When counting up, when PWMn_CNT<PWMn_CCR1 OCnREF output is low, otherwise OCnREF output is high When counting down, when PWMn_CNT>PWMn_CCR1 OCnREF output is high, otherwise OCnREF output is low

Note 1: Once the LOCK level is set to 3 (LOCK bit in the PWMn\_BKR register) and CCnS=00 (the channel is configured as output), this bit cannot be modified.

Note 2: In PWM Mode 1 or PWM Mode 2, only when the compare result changes or from Freeze mode in Output Compare mode The OCnREF level changes only when switching to PWM mode.

Note 3: On channels with complementary outputs, these bits are preloaded. If CCPC=1 in the PWMn\_CR2 register, the OCM bit New values are taken from the preload bits only when a COM event occurs.

OCnPE: output compare n preload enable (n=1,5)

0: The preload function of the PWMn\_CCR1 register is disabled, the PWMn\_CCR1 register can be written at any time, and the newly written value Works immediately.

1: Enable the preload function of the PWMn\_CCR1 register, read and write operations are only performed on the preload register, and the preload function of PWMn\_CCR1 is The load value is loaded into the current register when the update event arrives.

Note 1: Once the LOCK level is set to 3 (LOCK bit in the PWMn\_BKR register) and CCnS=00 (the channel is configured as output), this bit cannot be modified.

Note 2: The preload feature must be enabled in PWM mode for proper operation. But in single pulse mode (PWMn\_CR1 register OPM=1 of the controller), it is not required.

OCnFE: Output Compare n Fast Enable. This bit is used to speed up the CC output's response to a trigger input event. (n=1,5)

0: CCn operates normally according to the value of the counter and CCRn, even if the flip-flop is on. When the trigger input has a valid edge, the minimum delay to activate the CCn output is 5 clock cycles.

1: The active edge input to the flip-flop acts as if a compare match had occurred. Therefore, OC is set to the compare level and compared with Results are irrelevant. The delay between the active edge of the sampling flip-flop and the CC1 output is reduced to 3 clock cycles. OCFE only on channel Takes effect when 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 pin

CC1S[1:0]	Direction	input pin
00	output	
01	input	IC1 is mapped on T11FP1
10	input	IC1 is mapped on T12FP1
11	input	IC1 is mapped on TRC. This mode only works when the internal trigger input is

		When checked (selected by the TS bits 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 pin		
CC5S[1:0]	Direction	input pin
00	output	
01	input	IC5 is mapped on TI5FP5
10	input	IC5 is mapped on TI6FP5
11	enter	IC5 is mapped on TRC. This mode only works when the internal trigger input is When checked (selected by the TS bits of the PWM5_SMCR register)

**Note:** CC1S is writable **only when the channel is off** ( CC1E=0 in the PWMA\_CCER1 register ).

**Note:** CC5S is writable **only when the channel is off** ( CC5E=0 in the PWM5\_CCER1 register ).

The channel is configured in capture input mode

Symbol address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1_7	EFEC8H	IC1F[3:0]				IC1PSC[1:0]	CC1S[1:0]	
PWMB_CCMR1_7	EFEE8H	IC5PSC[1:0]	IC5F[3:0]				CC5S[1:0]	

ICnF[3:0]: Input capture n filter selection, this bit field defines the sampling frequency and digital filter length of TIn. (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	Twenty four	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 CCPC (PWMMn\_CR2 register device) value

ICnPSC[1:0]: Input/Capture n Prescaler. These two bits define the prescaler factor for the CCn input (IC1). (n=1,5)

00: No prescaler, each edge detected on the capture input triggers a capture

01: Trigger a capture every 2 events

10: trigger a capture every 4 events

11: trigger a capture every 8 events

CC1S[1:0]: Capture/Compare 1 selection. These two bits define the direction of the channel (input/output), and the selection of the input pin

CC1S[1:0]	Direction	input pin
00	output	
01	input	IC1 is mapped on TI1FP1
10	input	IC1 is mapped on TI2FP1
11	enter	IC1 is mapped on TRC. This mode only works when the internal trigger input is When checked (selected by the TS bits 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 pin

CC5S[1:0]	Direction	input pin
00	output	
01	input	IC5 is mapped on TI5FP5
10	input	IC5 is mapped on TI6FP5
11	enter	IC5 is mapped on TRC. This mode only works when the internal trigger input is checked (selected by the TS bits of the PWM5_SMCR register)

**Note:** CC1S is writable **only when the channel is off** ( CC1E=0 in the PWMA\_CCER1 register ).

**Note:** CC5S is writable **only when the channel is off** ( CC5E=0 in the PWM5\_CCER1 register ).

### 23.7.14 Capture/Compare Mode Register 2 (PWMx\_CCMR2)

The channel is configured in compare output mode

Symbol	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 enables the use of an external event on the PWMETI pin to clear the output signal of channel n (OCnREF) (n=2,6)

0: OCnREF is not affected by ETRF input;

1: OCnREF=0 once ETRF input high level 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 pin

CC2S[1:0]	Direction	input pin
00	output	
01	input	IC2 is mapped on TI2FP2
10	input	IC2 is mapped on TI1FP2
11	input	IC2 is mapped on TRC.

CC6S[1:0]: Capture/Compare 6 selection. These two bits define the direction of the channel (input/output), and the selection of the input pin

CC6S[1:0]	Direction	input pin
00	output	
01	input	IC6 is mapped on TI6FP6
10	input	IC6 is mapped on TI5FP6
11	input	IC6 is mapped on TRC.

The channel is configured in capture input mode

Symbol	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 pin

CC2S[1:0]	Direction	input pin
00	output	
01	input	IC2 is mapped on TI2FP2
10	input	IC2 is mapped on TI1FP2
11	input	IC2 is mapped on TRC.

CC6S[1:0]: Capture/Compare 6 selection. These two bits define the direction of the channel (input/output), and the selection of the input pin

CC6S[1:0]	Direction	input pin
00	output	
01	input	IC6 is mapped on TI6FP6
10	input	IC6 is mapped on TI5FP6
11	input	IC6 is mapped on TRC.

### 23.7.15 Capture/Compare Mode Register 3 (PWMx\_CCMR3)

The channel is configured in compare output mode

Symbol address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3 7	EFECAH	OC3CE	OC3M[2:0]	OC3PE	OC3FE		CC3S[1:0]	
PWMB_CCMR3 7	EEFEEAH	OC7CE	OCnCE:	OC7M[2:0]	OC7PE	OC7FE		CC7S[1:0]

Output compare n clear enable. This bit enables the use of an external event on the PWMETI pin to clear the output signal of channel n

(OCnREF) (n=3,7)

0: OCnREF is not affected by ETRF input;

1: OCnREF=0 once ETRF input high level 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 pin

CC3S[1:0]	Direction	input pin
00	output	
01	input	IC3 is mapped on TI3FP3
10	input	IC3 is mapped on TI4FP3
11	input	IC3 is mapped on TRC.

CC7S[1:0]: Capture/Compare 7 selection. These two bits define the direction of the channel (input/output), and the selection of the input pin

CC7S[1:0]	Direction	input pin
00	output	
01	input	IC7 is mapped on TI7FP7
10	input	IC7 is mapped on TI8FP7
11	input	IC7 is mapped on TRC.

The channel is configured in capture input mode

Symbol address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3 7	EFECAH	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 pin

CC3S[1:0]	Direction	input pin
00	output	
01	input	IC3 is mapped on TI3FP3
10	input	IC3 is mapped on TI4FP3
11	input	IC3 is mapped on TRC.

CC7S[1:0]: Capture/Compare 7 selection. These two bits define the direction of the channel (input/output), and the selection of the input pin

CC7S[1:0]	direction	input pin
00	output	
01	input	IC7 is mapped on TI7FP7
10	input	IC7 is mapped on TI8FP7
11	input	IC7 is mapped on TRC.

### 23.7.16 Capture/Compare Mode Register 4 (PWMy\_CCMR4)

The channel is configured in compare output mode

Symbol	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 OCnCE:		OC8M[2:0]		OC8PE	OC8FE		CC8S[1:0]	

Output compare n clear enable. This bit enables the use of an external event on the PWMETI pin to clear the output signal of channel n

(OCnREF) (n=4,8)

0: OCnREF is not affected by ETRF input;

1: OCnREF=0 once ETRF input high level 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 pin

CC4S[1:0]	Direction	input pin
00	output	
01	input	IC4 is mapped on TI4FP4
10	input	IC4 is mapped on TI3FP4
11	input	IC4 is mapped on TRC.

CC8S[1:0]: Capture/Compare 8 selection. These two bits define the direction of the channel (input/output), and the selection of the input pin

CC8S[1:0]	Direction	input pin
00	output	
01	input	IC8 is mapped on TI8FP8
10	input	IC8 is mapped on TI7FP8
11	input	IC8 is mapped on TRC.

The channel is configured in capture input mode

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4 7EFFECBH			IC4F[3:0]			IC4PSC[1:0]		CC4S[1:0]	
PWMB_CCMR4 7EFFEEBH 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 pin

CC4S[1:0]	Direction	input pin
00	output	
01	input	IC4 is mapped on TI4FP4
10	input	IC4 is mapped on TI3FP4
11	input	IC4 is mapped on TRC.

CC8S[1:0]: Capture/Compare 8 selection. These two bits define the direction of the channel (input/output), and the selection of the input pin

CC8S[1:0]	Direction	input pin
00	output	
01	input	IC8 is mapped on TI8FP8
10	input	IC8 is mapped on TI7FP8
11	input	IC8 is mapped on TRC.

### 23.7.17 Capture/Compare Enable Register 1 (PWMA\_CCER1)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
PWMA_CCER1 7EFFECCH CC2NP CC2NE CC2P:					CC2E	CC1NP	CC1NE	CC1P	
PWMB_CCER1 7EFFEECH CC6P:	-	-	CC6P	CC6E	-	-	CC5P	CC5E	

OC6 input capture/compare output polarity. Reference CC1P

CC6E: OC6 input capture/compare output enable. Reference CC1E

CC5P: OC5 input capture/compare output polarity. Reference CC1P

CC5E: OC5 input capture/compare output enable. Reference CC1E

CC2NP: OC2N compare output polarity. Reference CC1NP

CC2NE: OC2N compare output enable. Reference CC1NE

CC2P: OC2 input capture/compare output polarity. Reference CC1P

CC2E: OC2 input capture/compare output enable. Reference CC1E

CC1NP: OC1N compare output polarity

0: Active high level;

1: Active low level.

Note 1: Once the LOCK level (LOCK bit in the PWMA\_BKR register) is set to 3 or 2 and CC1S=00 (channel configured as output), this bit cannot be modified.

Note 2: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA\_CR2 register), only

When a COM event occurs, the CC1NP bit takes a new value from the preload bits.

CC1NE: OC1N compare output enable

0: Turn off the compare output.

1: Enable compare output, whose output level depends on the value of MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

Note: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA\_CR2 register), only in COM

When an event occurs, the CC1NE bit takes a new value from the preload bits.

#### CC1P: OC1 input capture/compare output polarity

The CC1 channel is configured as an output:

0: Active high

1: Active low

CC1 channel 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: Once the LOCK level (LOCK bit in the PWMA\_BKR register) is set to 3 or 2, this bit cannot be modified.

Note 2: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA\_CR2 register), only

When a COM event occurs, the CC1P bit takes a new value from the preload bits.

#### CC1E: OC1 input capture/compare output enable

0: close input capture/compare output;

1: Enable input capture/compare output.

Note: For channels with complementary outputs, this bit is preloaded. If CCPC=1 (PWMA\_CR2 register), only in COM

When an event occurs, the CC1E bit takes a new value from the preload bits.

Control bits for complementary output channels OCi and OCiN with brake function

		control bit			output status				
MOE	OSSIO	SRCCi	ECCiNE		OCi output status	OCiN output status			
1 X		0	0	0	Output		output disabled		
		0	0	1	Disable		OCiREF with polarity		
		0	1	0	Output Disable OCiREF		output disabled		
		0	1	with Polarity 1	OCiREF with Polarity and		Inverting OCiREF with Polarity and Dead Time		
		1	0	0	Deadband Output		output disabled		
		1	0	1	Disable OFF state (output enabled and inactive level) OCi=CCiP		OCiREF with polarity		
		1	1	0	OCiREF with polarity		Disabled (output enabled and inactive level) OCiN=CCiNP		
		1	1	1	OCiREF with polarity and dead time		Inverting OCiREF with Polarity and Dead Time		
0	0	XXX		output disabled					
	1			Off state (output enabled and inactive level) asynchronously: OCi=CCiP, OCiN=CCiNP; Then, if the clock exists: OCi=OISi after a dead time, OCiN=OISiN, suppose OISi and OISiN do not both correspond to the active levels of OCi and OCiN.					

Note: The state of the external I/O pins connected to the complementary OCi and OCiN channels depends on the OCi and OCiN channel states and the GPIO register.

### 23.7.18 Capture/Compare Enable Register 2 (PWMA\_CCER2)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER2	7FECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P

PWMB_CCER2 7EFEEDH CC8P:	-	-	CC8P	CC8E	-	-	CC7P	CC7E
--------------------------	---	---	------	------	---	---	------	------

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)

Symbol address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRH 7EFECEH					CNT1[15:8]			
PWMB_CNTRH 7EFEEEH					CNT2[15:8]			

CNTn[15:8]: The upper 8-bit value of the counter (n= A,B)

### 23.7.20 Counter Low 8 Bits (PWMx\_CNTRL)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRL 7EFEFCFH					CNT1[7:0]				
PWMB_CNTRL 7EFEEFH					CNT2[7:0]				

CNTn[7:0]: The lower 8-bit value of the counter (n= A,B)

### 23.7.21 Prescaler high 8 bits (PWMx\_PSCRH), output frequency calculation formula

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRH 7EFFED0H					PSC1[15:8]				
PWMB_PSCRH 7EFFEF0H					PSC2[15:8]				

PSCn[15:8]: The upper 8-bit value of the prescaler. (n=A,B)

The prescaler is used to divide CK\_PSC. The clock frequency of the counter (fCK\_CNT) is equal to fCK\_PSC/(PSCR[15:0]+1).

The PSCR contains the value loaded into the current prescaler register when the update event occurs (the update event includes the counter being UG of the TIM\_EGR bit is cleared to 0 or is cleared to 0 by a slave controller operating in reset mode). This means that in order for the new value to work, an update event must be generated.

#### PWM output frequency calculation formula

PWMA and PWMB two groups of PWM output frequency calculation formula is the same, and each group can be set to a different frequency.

alignment mode	PWM output frequency calculation formula
edge alignment	PWM output frequency = $\frac{\text{System operating frequency SYSclk}}{(\text{PWMx_PSCR} + 1) \times (\text{PWMx_AAR} + 1)}$

center alignment	System operating frequency SYSclk PWM output frequency = $\frac{\text{System operating frequency SYSclk}}{(\text{PWMx_PSCR} + 1) \times \text{PWMx_AAR} \times 2}$
------------------	---

### 23.7.22 Prescaler Lower 8 Bits (PWMx\_PSCRL)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRL	7EFED1H								PSC1[7:0]
PWMB_PSCRL	7EFEF1H								PSC2[7:0]

PSCn[7:0]: The lower 8-bit value of the prescaler. (n=A,B)

### 23.7.23 Auto-reload register upper 8 bits (PWMx\_ARRH)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRH	7EFED2H								ARR1[15:8]
PWMB_ARRH	7EFEF2H								ARR2[15:8]

ARRn[15:8]: Auto-reload high 8-bit value (n= A,B)

ARR contains the value to be loaded into the actual auto-reload register. When the value of auto-reload is 0, the counter does not work.

### 23.7.24 Auto-reload register lower 8 bits (PWMx\_ARRL)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRL	7EFED3H								ARR1[7:0]
PWMB_ARRL	7EFEF3H								ARR2[7:0]

ARRn[7:0]: Auto-reload lower 8-bit value (n= A,B)

### 23.7.25 Repeat Counter Register (PWMx\_RCR)

symbol	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)

When the preload function is enabled, these bits allow the user to set the update rate of the compare register (that is, periodically transferred from the preload register).

output to the current register); if an update interrupt is enabled, it will also affect the rate at which the update interrupt is generated. count down each time

When the counter REP\_CNT reaches 0, an update event is generated and the counter REP\_CNT starts counting again from the REP value. Depend on

Since REP\_CNT is only reloaded with the REP value when the periodic update event U\_RC occurs, writing to the PWMn\_RCR register

The new value entered will only take effect when the next periodic update occurs. This means that in PWM mode, (REP+1) corresponds to:

- in edge-aligned mode, the number of PWM cycles;

- In centrosymmetric mode, the number of PWM half cycles.

### 23.7.26 Capture/Compare Register 1/5 Upper 8 Bits (PWMx\_CCR1H)

symbol	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: CCRn contains the current compare value loaded (preload value). If registered in PWMn\_CCMR1

If the preload function is not selected in the register (OCnPE bit), the written value is immediately transferred to the current register. Otherwise, only when the

This preload value is transferred into the current capture/compare n register only when a new event occurs. The current comparison value is the same as the counter PWMn\_CNT

The value is compared and an output signal is generated on the OCn port.

If the CCn channel is configured as an input: CCRn contains the counter value at the time of the last input capture event (this register is only read).

### 23.7.27 Capture/Compare Register 1/5 Lower 8 Bits (PWMx\_CCR1L)

symbol	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-bit value of n (n=1,5)

### 23.7.28 Capture/Compare Register 2/6 Upper 8 Bits (PWMx\_CCR2H)

symbol	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 Lower 8 Bits (PWMx\_CCR2L)

symbol	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-bit value of n (n=2,6)

### 23.7.30 Capture/Compare Register 3/7 High 8 Bits (PWMx\_CCR3H)

symbol	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 upper 8-bit value of n (n=3,7)

### 23.7.31 Capture/Compare Register 3/7 Lower 8 Bits (PWMx\_CCR3L)

symbol	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-bit value of n (n=3,7)

### 23.7.32 Capture/Compare Register 4/8 Upper 8 Bits (PWMx\_CCR4H)

symbol	address B7	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 upper 8-bit value of n (n=4,8)

### 23.7.33 Capture/Compare Register 4/8 Lower 8 Bits (PWMx\_CCR4L)

symbol	address B7	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-bit value of n (n=4,8)

### 23.7.34 Brake Register (PWMx\_BKR)

symbol	address B7	B7	B6	B5	B4	B3	B2	B1	B0	
PWMA_BKR	7EFEDDH	MOEA AOEA BKPA BKEA OSSRA OSSIA								
PWMB_BKR	7EFEFDH	MOEB AOEB BKPB BKEB OSSRB OSSIB								

MOEn: Main output enable. Once the brake input is active, this bit is asynchronously cleared by hardware. Depending on the setting of the AOE bit, this bit can be

is set to 1 or is automatically set to 1. It is only valid for channels configured as outputs. (n=A,B)

0: OC and OCN outputs disabled or forced to idle state

1: The OC and OCN outputs are enabled if the corresponding enable bit (CCIE bit in the PWMn\_CCERX register) is set.

AOEn: Automatic output enable (n= A, B)

0: MOE can only be set to 1 by software;

1: MOE can be set by software or automatically on the next update event (if the brake input is inactive).

Note: Once the LOCK level (LOCK bit in the PWMn\_BKR register) is set to 1, this bit cannot be modified

BKPN: Brake input polarity (n= A, B)

0: The brake input is active at low level

1: Brake input active high

Note: Once the LOCK level (LOCK bit in the PWMn\_BKR register) is set to 1, this bit cannot be modified

BKEN: Brake function enable (n= A, B)

0: Brake input prohibited (BRK)

1: Turn on the brake input (BRK)

Note: Once the LOCK level (LOCK bit in the PWMn\_BKR register) is set to 1, this bit cannot be modified.

OSSRN: "Off state" selection in run mode. This bit is valid when MOE=1 and the channel is set to output (n=A,B)

0: When PWM is not working, disable OC/OCN output (OC/OCN enable output signal = 0);

1: When PWM is not working, once CCIE=1 or CCINE=1, first turn on OC/OCN and output an inactive level, and then set 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.

OSSI: "Off state" selection in idle mode. This bit is valid when MOE=0 and the channel is set to output. (n=A,B)

0: When PWM is not working, disable OC/OCN output (OC/OCN enable output signal = 0);

1: When PWM is not working, 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 settings. Write protection provided by this bit against software errors (n= A,B)**

LOCKn[1:0] protection level	Protect content
00	No protection registers No write protection
01	Lock level 1 cannot write to the BKE, BKP, AOE bits and OISI bit of PWMn_OISR register
10	Lock level 2 cannot write to bits in lock level 1, The CC polarity bits and the OSSR/OSSI bits also cannot be written
11	lock level 3 cannot write to bits in lock level 2, also cannot write to the CC control bits

Note: Since the BKE, BKP, AOE, OSSR, OSSI bits can be locked (depending on the LOCK bit), the first write to PWMn\_BKR They must be set when registering.

### 23.7.35 DEAD-TIME REGISTER (PWMx\_DTR)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DTR	7EFEDEH								DTGA[7:0]
PWMB_DTR	7EFFFEH								DTGB[7:0]

DTGn[7:0]: Dead time generator settings. (n=A,B)

These bits define the dead-time duration between inserted complementary outputs. (tCK\_PSC is the clock pulse of PWMn)

DTGn[7:5]	dead time
000	DTGn[7:0] * tCK_PSC
001	
010	
011	
100	(64 + DTGn[6:0]) * 2 * tCK_PSC
101	
110	(32 + DTGn[5:0]) * 8 * tCK_PSC
111	(32 + DTGn[4:0]) * 16 * tCK_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)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_OISR	7EFEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1
PWMB_OISR	7EFFFFH	-	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 in idle state

OIS3: OC3 output level in idle state

OIS2N: OC2N output level in idle state

OIS2: OC2 output level in idle state

OIS1N: OC1N output level in idle state

0: When MOE=0, after a dead time, OC1N=0;

1: When MOE=0, OC1N=1 after a 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

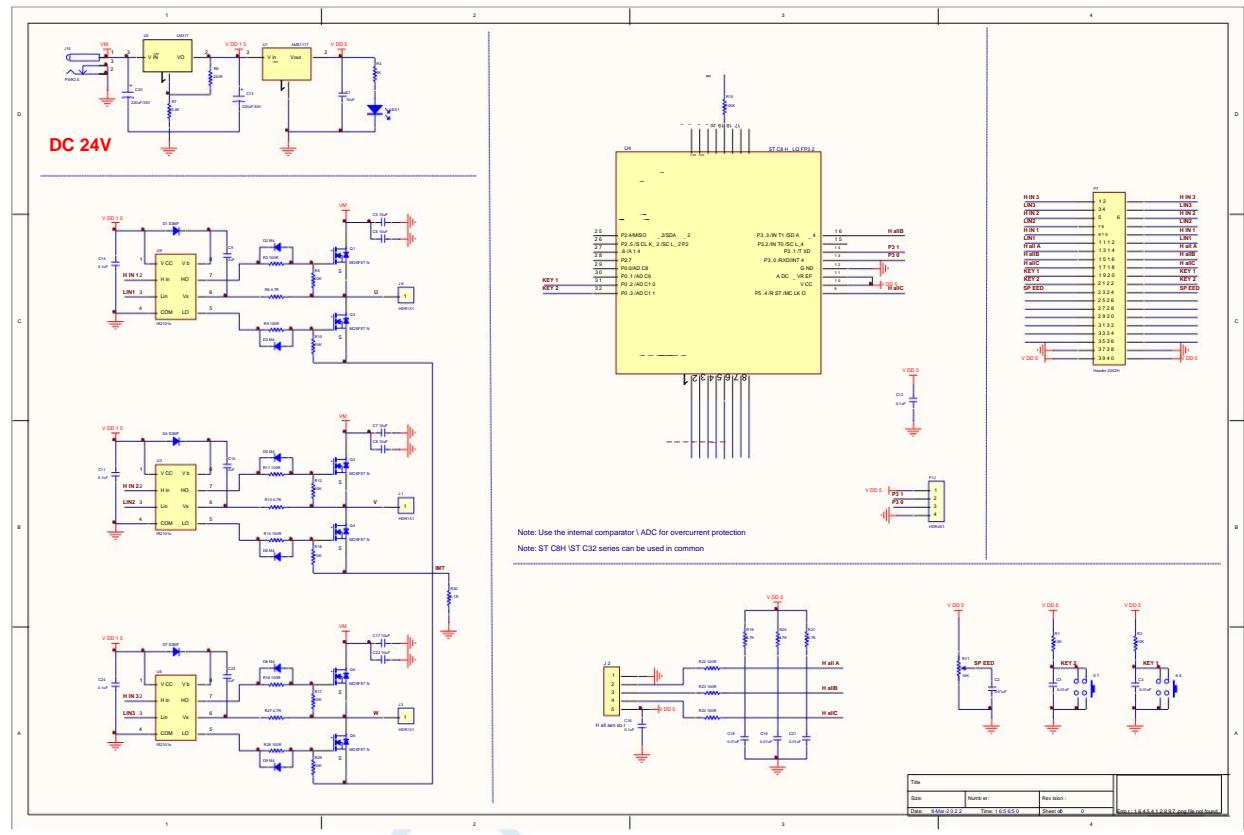
0: When MOE=0, if OC1N is enabled, after a dead zone, OC1=0;

1: When MOE=0, if OC1N is enabled, after a dead time, OC1=1.

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 Example Program

### 23.8.1 BLDC Brushless DC Motor (with HALL)



//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h"
#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 0xA
#define RUN 0xB
#define STOP 0xC
#define IDLE 0xD

#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 LED_OUT(u8 X); //LED Single-byte serial shift function

unsigned char code LED_OF[] = {

    0xC0,0xF9,0xA4,0xB0,
    0x99,0x92,0x82,0xF8,
    0x80, 0x90, 0x8C, 0xBF,
    0xC6, 0xA1, 0x86, 0xFF,
    0xbf
};

#define DIO P23 // Serial data input
#define RCLK P24 // Clock pulse signal - valid on rising edge
#define SCLK P25 // Input signal — Valid on rising edge

void DelayXus(unsigned char delayTime);
void DelayXms( unsigned char delayTime);

```

```

unsigned int ADC_Convert(u8 ch);
void PWM_Init(void);
void SPEED_ADJ();
unsigned char RD_HALL();
void MOTOR_START();
void MOTOR_STOP();
unsigned char KEY_detect();
void LED4_Display (unsigned int dat, unsigned char num);

unsigned char Display_num=1;
unsigned int Display_dat=0;
unsigned int Motor_speed;
unsigned char Motor_sta = IDLE;
unsigned char BRK_occur=0;
unsigned int 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      XFR
    CKCON = 0x00;                            //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                                //assign to      CPU The speed of executing the program is set to the fastest

    P_SW2 = 0x80;

    P1 = 0x00;
    P0M1 = 0x0C;
    P0M0 = 0x01;
    P1M1 = 0xc0;
    P1M0 = 0x3F;
    P2M1 = 0x00;
    P2M0 = 0x38;
    P3M1 = 0x28;
    P3M0 = 0x00;

    ET0=1;
    TR0=1;

    ADCCFG = 0x0f;
    ADC_CONTR = 0x80;

    PWMA_ENO = 0x3F;                      //PWMA output enable
    PWMB_ENO = 0x00;                      //PWMB output enable
    PWMA_PS = 0x00;                      //PWMA pin choose
    PWMB_PS = 0xd5;                      //PWMB pin choose

//*****output compare mode*****  

PWMx_duty = [CCRx/(ARR + 1)]*100  

//*****time base unit*****  

PWMB      catch hall sensor  

//**** time base unit ****/  

PWMB_PSCRL = 15;
PWMB_ARRH = 0xff;                      // Auto-reload registers, counters overflow point
PWMB_ARRL = 0xff;
PWMB_CCR8H = 0x00;
PWMB_CCR8L = 0x05;

```

```

////////// Channel configuration/////////
PWMB_CCMR1 = 0x43; // 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 commutation ****
////////// time base unit/////////
PWMA_PSCRH = 0x00; // Prescaler register
PWMA_PSCRL = 0x00;
PWMA_ARRH = (u8)(PWMA_Period >> 8);
PWMA_ARRL = (u8)(PWMA_Period);

////////// Channel configuration/////////
PWMA_CCMR1 = 0x70; // Channel Mode Configuration
PWMA_CCMR2 = 0x70;
PWMA_CCMR3 = 0x70;
PWMA_CCER1 = 0x11;
PWMA_CCER2 = 0x01;
PWMA_OISR = 0xAA;

////////// Mode configuration/////////
PWMA_CR1 = 0xA0;
PWMA_CR2 = 0x24;
PWMA_SMCR = 0x20;

////////// Enable interrupt configuration //////////
PWMA_BKR = 0x1c; // enable counter
PWMA_CR1 |= 0x01;

EA = 1;
while (1)
{
    P22=-P22;
    Display_dat = Motor_speed; // Motor_speed

    switch(Motor_st)
    {
        case START:
            MOTOR_START();
            Motor_st = RUN;
            break;
        case RUN:
            SPEED_ADJ();
            if((KEY_detect() == 2) || (BRK_occur == TRUE))
                Motor_st = 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 comparison 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);

    return Hall_sta;
}

void MOTOR_START()
{
    u16 temp;
    u16 ADC_result;

    PWMA_CCR1H = (u8)(PWMA_STPulse >> 8); // Counter comparison 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; // The main output enable is equivalent to the main 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_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 |= CC1N_POLARITY_HIGH;
PWMA_CCER1 &= CC2N_POLARITY_LOW;
PWMA_CCER2 &= CC3N_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_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 comparison 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--)
    {

```

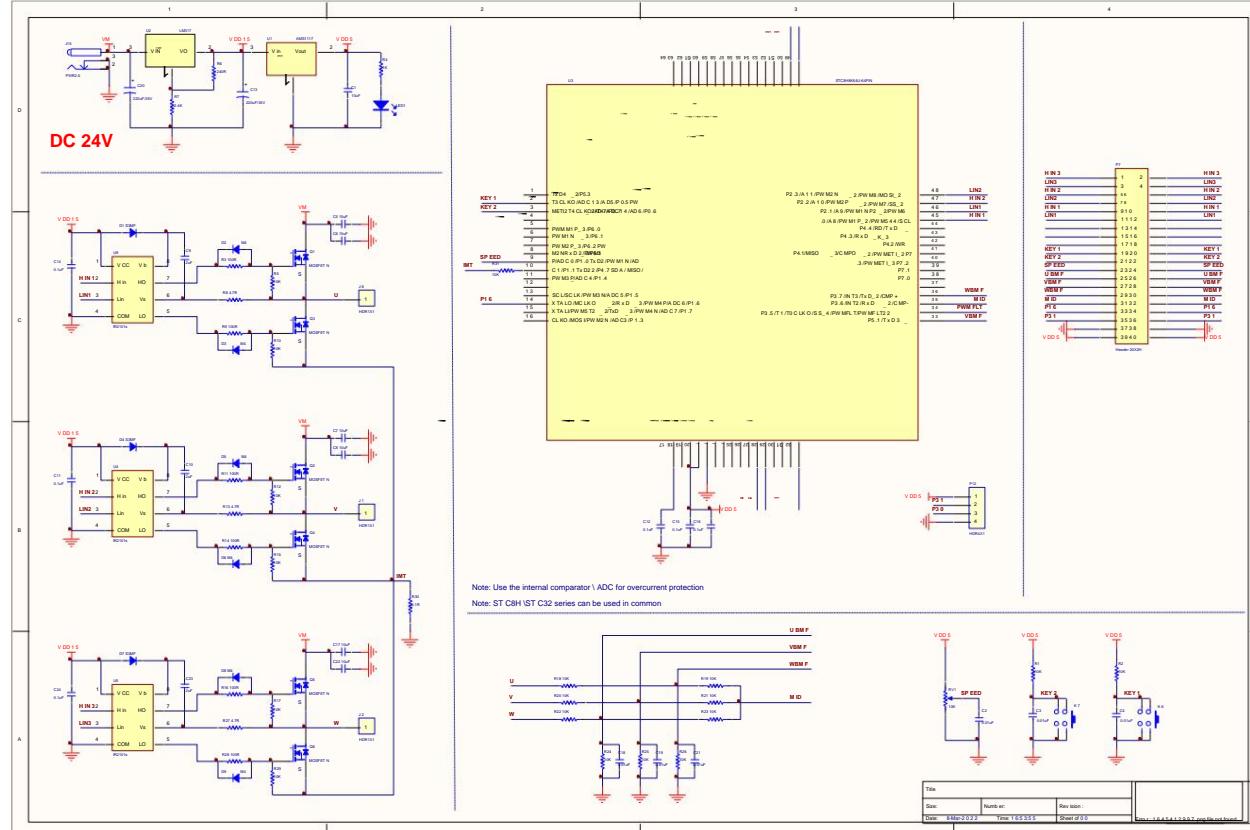
```
if (X&0x80) DIO=1;
else DIO=0;
X<<=1;
SCLK = 0;
SCLK = 1;
}

}

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 (No HALL)

—It can realize high-speed operation of 100,000 rpm without Hall



// The test working frequency is **11.0592MHz**

// the test working frequency. **11.0592MHz**

// This routine implements the following functions. This **3 PWM** Channel-controlled operation without Hall motor

// routine is only applicable to the motor in **57BL02**. **24V** Demonstration under no-load condition

```

##include "stc8h.h"
##include "stc32g.h"
##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)245)

#define START 0xA
#define RUN 0xB

```

```

#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=0xb0;

```

```

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;

void main(void)
{
    unsigned int temp=0;
    unsigned int ADC_result=0;

    CKCON = 0x00; //Set the external data bus speed to the fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    P_SW2= 0x80;
    P1 = 0x00;
    P0M1 = 0x0C;
    P0M0 = 0x01;
    P1M1 = 0xc0;
    P1M0 = 0x3F;
    P2M1 = 0x00;
    P2M0 = 0x38;
    P3M1 = 0x88;
    P3M0 = 0x02;

    ET0=1;
    TR0=0;
    ADCCFG = 0x0f;
    ADC_CONTR = 0x80;

    PWMA_ENO = 0x3F; //PWMA output enable
    PWMB_ENO = 0x00; //PWMB output enable
    PWMA_PS = 0x00; //PWMA pin choose
    PWMB_PS = 0xD5; //PWMB pin choose

    /*****output compare mode PWMx_duty = [CCRx/(ARR + 1)]*100 *****/
    //*****PWMB BMF      enter *****/
    //***** time base unit *****/
    PWMB_PSCRL = 15; //Auto-reload registers, counters overflow point
    PWMB_ARRH = 0xff;
    PWMB_ARRL = 0xff;
    PWMB_CCR8H = 0x00;
    PWMB_CCR8L = 0x05;
    //**** Channel configuration
    PWMB_CCMR1 = 0xf3; //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;
//////////////// & Enable interrupt configuration ///////////////////
PWMB_BKR = 0x80; // Main output enable
PWMB_IER = 0x02; // enable interrupt
/******PWMA Control motor commutation ******/
//////////////// time base unit ///////////////////
PWMA_PSCRH = 0x00; // Prescaler register
PWMA_PSCRL = 0x00;
PWMA_ARRH = (u8)(PWMA_Period >> 8);
PWMA_ARRL = (u8)(PWMA_Period);
////////////////// channel configuration
PWMA_CCMR1 = 0x70; // Channel Mode Configuration
PWMA_CCMR2 = 0x70;
PWMA_CCMR3 = 0x70;
PWMA_CCER1 = 0x11; // Configure Channel Output Enable and Polarity
PWMA_CCER2 = 0x01; // Configure Channel Output Enable and Polarity
PWMA_OISR = 0xAA; // Output level of each channel during configuration
////////////////// channel configuration ///////////////////
PWMA_CR1 = 0xA0;
PWMA_CR2 = 0x24;
PWMA_SMCR = 0x20;
PWMA_BKR = 0x0c; //////////////////& enable interrupt configuration ///////////////////
PWMA_CR1 |= 0x01; // enable counter
EA = 1;

UART_INIT();

while (1)
{
    switch(Motor_st)
    {
        case START:
            MOTOR_START();
            Motor_st = 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
            if((BRK_occur == TRUE))
            Motor_st = STOP;
            break;
        case STOP:
            MOTOR_STOP();
            Motor_st = IDLE;
            break;
        case IDLE:
            if(KEY_detect() == 1) // start the motor
            Motor_st = START;
    }
}

```

```

    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 filtering
    {
        PWMB_CAP1_v = ccr_tmp;
        PWMB_CAP1_v = (PWMB_CAP1_v<<8) + PWMB_CCR5L;
        if(Motor_sta == RUN) delay {                  // commutation      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 mode 1T
    TMOD = 0x20;                                      // Timer auto-reloads for 16 bits
    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);
    PWMA_CCR1H = (u8)(ADC_result >> 8); //Speed knob sampling ADC
    PWMA_CCR1L = (u8)(ADC_result); //Counter comparison value
    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;

    DelayXus(40);
    (P17)? (Hall_sta|=0x01) : (Hall_sta&=~0x01);
    (P54)? (Hall_sta|=0x02) : (Hall_sta&=~0x02);
    (P33)? (Hall_sta|=0x04) : (Hall_sta&=~0x04);

    return Hall_sta;
}

void MOTOR_START()
{
    PWMA_CCR1H = (u8)(PWMA_STPulse >> 8); //Counter comparison 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; //The main output enable is equivalent to the main 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(IP03)
    {
        DelayXms(10);
        if(IP03)
        {
            return 2;
        }
        else return 0;
    }

    } else return 0;
}

```

### 23.8.3 Implementing an Encoder Using Advanced PWM

---

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software

#define MAIN_Fosc 11059200L// define the master clock
***** Function Description *****

PWMA Module works in encoder mode PWMA The module can only connect one encoder *
serial 1 (RXD-->P3.0 TXD-->P3.1)      to return the reading result Serial port settings 115200,8,n,1;
encoder A phase input: PWM1P (P1.0)
encoder B phase input: PWM2P (P1.2)

encoder mode
    pattern 1: Two edges per pulse plus and minus two 2.
    pattern 2: edges per pulse plus and minus two edges 2.
    pattern 3: per pulse plus and minus two edges per pulse 4.
*****
```

```

unsigned int  pulse;
bit          B_Change;                           encoder pulse
                                                //Encoder count change

bit          B_TX1_Busy;                         // Send busy flag

void         PWMA_config(void);
void         UART1_config(unsigned long brt);     // brt: Communication baud rate
void         UART1_TxByte(unsigned char dat);

```

```

void main(void)
{
    unsigned int j;

    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    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
            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(0xd);
            UART1_TxByte(0xa);
        }
    }

//=====
// Function: void PWMA_config(void)
// description: PWM configure function.
// parameter: none.
// returns : none.
// version : V1.0, 2021-5-10
// remarks :
//=====

void PWMA_config(void)
{
    PWMA_PSCR = 0; //Prescaler Fck_cnt = Fck_psc/(PSCR[15:0]+1),
    //Edge-aligned frequency PWM = Sysclk/((PSCR+1)*(AAR+1)),
    //Center-aligned frequency PWM = Sysclk/((PSCR+1)*(AAR+1)*2).
    PWMA_ARR = 0xffff; //Auto-reload register contrgClear PWM cycle
    PWMA_CNTR = 0; //encoder counter value
    PWMA_ENO = 0; //IO Disable output PWM

    PWMA_CCMR1 = 0x01+(10<<4); //The channel 1 mode configuration is configured as the input channel ,,
    //0~15 corresponds 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); //The channel 2 mode configuration is configured as the input channel ,,
    //0~15 corresponds to the number of input filter clocks
}

```

```

// 1 2 4 8 12 16 24 32 48 64 80 96 128 160 192 256
//Encoder mode mode 1 mode two edges per pulse plus and minus mode four 2.
//edges per pulse plus and minus mode four 2.
// 3: pulse plus and minus mode four 4.
// 4: pulse plus and minus mode four 4.
// channel input enable and
//,
// IO select P1.0 P1.2
//enable interrupt
//Enabling the counter allows the auto-reload register buffer ,
//Edge-aligned mode to control the writeable auto-reload register
//bit7=1:Directly write the auto-reload register (this is addressed up
//=0: ( )
//)

}

//=====
// function: void PWMA_ISR(void) interrupt PWMA_VECTOR
//   : PWMA interrupt handler
//   : None
//   : none.
// description 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 has captured value
    }
    PWMA_SR1 = 0;
}

//=====
// The : void          UART1_config(u32 brt)
// function : UART1 initialization function.
// description:brt: Communication baud rate
// parameter none.
// returns : VER1.0
// the   : 2018-4-2
// version date remarks
//=====

void UART1_config(unsigned long brt) {                  //brt: Communication baud rate
    Brt = 65536UL - (MAIN_Fosc / 4) / brt;
    AUXR &= ~0x01;
    AUXR |= (1<<6);
    TMOD &= 0x0f;
    TH1 = (unsigned char)(brt >> 8);
    TL1 = (unsigned char)brt;
    TR1 = 1;                                         // Running Timer1
    P_SW1 &= ~0xc0;
    SCON = (SCON & 0x3f) | (1<<6);                //serial port switch P3.0 P3.1
    ES = 1;                                         //8 bit data bit start enable , 1 Bit stop bit without parity
    REN = 1;                                         //interrupt
                                                //Allow to receive
}

//=====
// Function void UART1_TxByte(u8 dat)
// description Serial port query Send a byte Function
// parameter dat: Return version of byte data to be sent
//   : none.
//   : VER1.0

```

```

// date : 2018-4-2
// note :
//=====
void UART1_TxByte(unsigned char dat)
{
    B_TX1_Busy = 1;                                //flag send busy
    SBUF = dat;                                    //send a byte
    while(B_TX1_Busy);                            //wait for sending to complete
}

//=====
// Function: void UART1_int (void) interrupt UART1_VECTOR
// description Serial 1 interrupt function
// port parameters: none
// Return value: none
// Remarks: VER1.0
//      : 2018-4-2
//      :
//=====

void UART1_int (void) interrupt 4
{
    if(RI)
        RI = 0;

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

```

### 23.8.4 Quadrature Encoder Mode

---

//The test frequency is 11.0592MHz

```

##include "stc8h.h"
#include "stc32g.h"                                     //For header files, see Download Software
#include "intrins.h"

unsigned char cnt_H, cnt_L;

void main(void)
{
    EAXFR = 1;                                //Enable XFR
    CKCON = 0x00;                            //access to set external data bus speed to fastest
    WTST = 0x00;                             //Set the program code to wait for parameters,
                                                //assign to CPU The speed of executing the program is set to the fastest

    P1M1 = 0x0f;
    P1M0 = 0x00;

    PWMA_ENO = 0x00;                         //configured as TRGI of pin need to be turned off ENO correspond bit and dubbed input
    PWMA_PS = 0x00;                           //00: PWM at P1

    PWMA_PSCRH = 0x00;                        //Prescaler register


```

```

PWMA_PSCRL = 0x00;

PWMA_CCMR1 = 0x21; //The channel mode is configured as input and connected to the encoder filter. 4 clock
PWMA_CCMR2 = 0x21; //The channel mode is configured as input and connected to the encoder filter. 4 clock

PWMA_SMCR = 0x03; //encoder mode3

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

EA = 1;

while (1);
}

//***** PWM ***** Interrupt to 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)

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void main(void)
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                    //assign to CPU The speed of executing the program is set to the fastest

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x0c;
    P1M0 = 0xF3;

    PWMA_ENO = 0xF3; //IO output PWM
    PWMA_PS = 0x00; //00:PWM at P1

    //***** *****
    PWMx_duty = [CCRx/(ARR + 1)]*100
    //*****
}
```

```

//configured as TRGI of pin need to be turned off ENO correspond bit and dubbed input
PWMA_PSCRH = 0x00; //Prescaler register
PWMA_PSCRL = 0x00; //Dead time configuration
PWMA_DTR = 0x00;

PWMA_CCMR1 = 0x68; //Channel Mode Configuration
PWMA_CCMR2 = 0x01; //Configured as an input channel
PWMA_CCMR3 = 0x68;
PWMA_CCMR4 = 0x68;

PWMA_SMCR = 0x66;

PWMA_ARRH = 0x08; //Auto-reload registers, counters 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; //The main output enable is equivalent to the main switch
PWMA_IER = 0x02; //enable interrupt
PWMA_CR1 = 0x08; //Single pulse mode
PWMA_CR1 |= 0x01; //enable counter

EA = 1;
while (1);
}

void PWMA_ISR() interrupt 26
{
    if (PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        PWMA_SR1 &= ~0X02;
    }
}

```

### 23.8.6 GATE MODE (INPUT LEVEL ENABLE COUNTER)

---

```

//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void main(void)
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest

```

```

WTST = 0x00;                                // Set the program code to wait for parameters,
                                                // assign to CPU The speed of executing the program is set to the fastest

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
/******************************************** */

// configured as TRGI of pin need to be turned off ENO correspond bit and dubbed input
PWMA_PSCRH = 0x00;                          // Prescaler register
PWMA_PSCRL = 0x00;                          // Dead time configuration
PWMA_DTR = 0x00;

PWMA_CCMR1 = 0x68;                          // Channel Mode Configuration
PWMA_CCMR2 = 0x68;                          // Configured as an input channel
PWMA_CCMR3 = 0x68;
PWMA_CCMR4 = 0x68;

PWMA_SMCR = 0x75;                           // Gated trigger mode ETRF enter

PWMA_ARRH = 0x08;                           // Auto-reload registers, counters 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;                           // The main output enable is equivalent to the main 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;
    }
}

```

}

## 23.8.7 External Clock Mode

//The test frequency is 11.0592MHz

```

//#include "stc0h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

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

    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
    *****//*
    configured as TRGI pin# need to be turned on ENO input on bit and dubbed
    PWMA_PSCRH = 0x00;   //Prescaler register
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x00;    //Dead time configuration

    PWMA_CCMR1 = 0x68;  Channel Mode Configuration
    PWMA_CCMR2 = 0x68;
    PWMA_CCMR3 = 0x68;
    PWMA_CCMR4 = 0x68;
    //Configured as an input channel

    PWMA_SMCR = 0x77;  //ETRF enter

    PWMA_ARRH = 0x08;  // Auto-reload registers, counters 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; //The main output enable is equivalent to the main 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;
    }
}
```

STCMCU

### 23.8.8 Input Capture Mode Measurement Pulse Period (Capture Rising Edge to Rising Edge or Falling

edge to falling edge)

Principle: Use the capture module CCx of a certain channel inside the advanced PWM to capture the rising edge or falling edge of the external port, and the two upper

The time between rising edges or two falling edges is the period of the pulse, that is to say, the difference between the two capture count values is the cycle.  
period value.

Example: Use the first group capture module CC1 capture function of PWMA to capture the rising edge on the PWM1P (P1.0) pin, in an interrupt

Subtract the two captured values before and after to get the period

Note: Only PWM1P, PWM2P, PWM3P, PWM4P, PWM5, PWM6, PWM7, PWM8 these pins and phase

The pin should be switched to have the capture function

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

int      cap;
int      cap_new;
int      cap_old;

void main(void)
{
    EA_XFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    P0M1 = 0x00;
    P0M0 = 0x00;
    P1M1 = 0x00;
    P1M0 = 0x00;

/* configured as TRGI of pin need to be turned off ENO correspond bit and dubbed input */
    PWMA_ENO = 0x00; //IO output PWM
    PWMA_PS = 0x00; //00: PWM at P1

    PWMA_CCMR1 = 0x01; //Configured as an input channel
    PWMA_SMCR = 0x56; //Configure Channel Output Enable and Polarity
    PWMA_CCER1 = 0x01;

    PWMA_IER = 0x02; //enable interrupt
    PWMA_CR1 |= 0x01; //enable counter

    EA = 1;
    while (1);
}

/* aisle 1 input, capture data via PWMA_CCR1H / PWMA_CCR1L read */
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;
}
}
```

---

STCMCU

### 23.8.9 Input Capture Mode Measurement Pulse High Width (Capture Rising Edge to Falling Edge)

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, then subtract the captured value of CCx from the captured value of CCx+1, the

The difference is the width of the high level of the pulse.

Example: Use the first group of capture module CC1 and the second group of capture module CC2 of PWMA to capture the PWM1P pin (P1.0) at the same time,

Among them, CC1 captures the rising edge of PWM1P, CC2 captures the falling edge of PWM1P, and subtracts the captured value of CC2 in the interrupt.

To the captured value of CC1, the difference is the width of the pulse high level.

Note: 1. The two-way capture module inside the chip is used to capture the same external pin at the same time, so there is no need to connect multiple external pins connected.

2. Only four combinations of CC1+CC2, CC3+CC4, CC5+CC6, CC7+CC8 can complete the above functions. CC1+CC2

Combination can capture PWM1P pin and PWM2P pin simultaneously; CC3+CC4 combination can capture simultaneously

PWM3P pin, can also capture PWM4P pin at the same time; CC5+CC6 combination can capture PWM5 pin at the same time, also can

Capture PWM6 pin at the same time; CC7+CC8 combination can capture PWM7 pin and PWM8 pin at the same time

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMA_CCER1 = 0x00; //CC1 capture TI1 rising edge, CC2 capture TI1 falling edge
    PWMA_CCMR1 = 0x01;
    PWMA_CCMR2 = 0x02;
    PWMA_CCER1 = 0x11;
    PWMA_CCER1 |= 0x00;
    PWMA_CCER1 |= 0x20;
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x04; //Enable CC2 catch 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;  
    cnt = cnt2 - cnt1;  
}  
}  
  
// The difference is the high level width
```

STCMCU

### 23.8.10 Input capture mode measurement pulse low width (capture falling edge to rising edge)

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 falling edge of this pin, CCx+1 captures the rising edge of this pin, then subtract the captured value of CCx from the captured value of CCx+1, the

The difference is the width of the pulse low level.

Example: Use the first group of capture module CC1 and the second group of capture module CC2 of PWMA to capture the PWM1P pin (P1.0) at the same time,

Among them, CC1 captures the falling edge of PWM1P, CC2 captures the rising edge of PWM1P, and subtracts the captured value of CC2 in the interrupt.

To the capture value of CC1, the difference is the width of the low level of the pulse.

Note: 1. The two-way capture module inside the chip is used to capture the same external pin at the same time, so there is no need to connect multiple external pins connected.

2. Only four combinations of CC1+CC2, CC3+CC4, CC5+CC6, CC7+CC8 can complete the above functions. CC1+CC2

Combination can capture PWM1P pin and PWM2P pin simultaneously; CC3+CC4 combination can capture simultaneously

PWM3P pin, can also capture PWM4P pin at the same time; CC5+CC6 combination can capture PWM5 pin at the same time, also can

Capture PWM6 pin at the same time; CC7+CC8 combination can capture PWM7 pin and PWM8 pin at the same time

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h"
#include "intrins.h"

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;
    PWMA_CCMR2 = 0x02;
    PWMA_CCER1 = 0x11;
    PWMA_CCER1 |= 0x00;
    PWMA_CCER1 |= 0x20;
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x02;
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;
    unsigned int cnt1;
    unsigned int cnt2;
}

```

*STCMCU*

// For header files, see Download Software

Enable XFR  
access to set external data bus speed to fastest

Set the program code to wait for parameters,  
assign to CPU The speed of executing the program is set to the fastest

//(CC1 capture TI1 rising edge CC2 capture TI1 falling edge)  
// CC1 is an input mode and is mapped to TI1FP1 superior  
// a capture function mapped to the input TI1FP2 superior  
mapped CC1/CC2  
Set capture polarity to rising edge  
CC1  
Set capture polarity to falling edge  
CC2

//Enable CC1 catch interrupt

```
if (PWMA_SR1 & 0x02)
{
    PWMA_SR1 &= ~0x02;

    cnt1 = (PWMA_CCR1H << 8) + PWMA_CCR1L;
    cnt2 = (PWMA_CCR2H << 8) + PWMA_CCR2L;
    cnt = cnt1 - cnt2;
}

// The difference is the low level width
```

STCMCU

### 23.8.11 Input Capture Mode 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 a reset trigger signal,

The captured value of CCx is the period, and the captured value of CCx+1 is the duty cycle.

Example: Use the first group of capture module CC1 and the second group of capture module CC2 of PWMA to capture the PWM1P pin (P1.0) at the same time,

Among them, CC1 captures the rising edge of PWM1P, CC2 captures the falling edge of PWM1P, and sets the rising edge signal of PWM1P to

Reset the trigger signal, the capture value of CC1 is the period, and the capture value of CC2 is the duty cycle.

Note: 1. The two-way capture module inside the chip is used to capture the same external pin at the same time, so there is no need to connect multiple external pins connected.

2. Only the two combinations of CC1+CC2 and CC5+CC6 can complete the above functions. CC1+CC2 combination can capture simultaneously

PWM1P pin, can also capture PWM2P pin at the same time; CC5+CC6 combination can capture PWM5 pin at the same time, also can

to capture the PWM6 pin simultaneously

3. Since the reset trigger signal is set, the capture value is the period value and the duty cycle value, and there is no need to subtract the previous capture value.

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1; // XFR
    CKCON = 0x00; // access to set external data bus speed to fastest
    WTST = 0x00; // Set the program code to wait for parameters,
    // assign to CPU The speed of executing the program is set to the fastest

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    //((CC1 Capture T11 edge ,CC2 Capture T11 edge ) )
    //CC1 capture period width,CC2 capture high level width

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;
    PWMA_CCMR2 = 0x02;
    PWMA_CCER1 = 0x11;
    PWMA_CCER1 |= 0x00;
    PWMA_CCER1 |= 0x20;
    PWMA_SMCR = 0x54;
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x06;
    EA = 1; //Enable CC1/CC2 catch interrupt

    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 period width  
}  
if (PWMA_SR1 & 0x04)  
{  
    PWMA_SR1 &= ~0x04;  
  
    cnt = (PWMA_CCR2H << 8) + PWMA_CCR2L;           // CC2 Capture duty cycle (high width)  
}  
}
```

### 23.8.12 Simultaneous Capture of Period and Duty Cycle of 4 Input Signals

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, the difference between the two capture values of CCx is the period, and the capture value of CCx+1

The difference between the captured value and the previous capture value of CCx is the duty cycle.

Example: Use the first group of capture module CC1 and the second group of capture module CC2 of PWMA to capture the PWM1P pin (P1.0) at the same time,

Among them, CC1 captures the rising edge of PWM1P, CC2 captures the falling edge of PWM1P, and the capture value of CC1 minus the previous capture value

That is the period, the capture value of CC2 minus the previous capture value of CC1 is the duty cycle. Simultaneous capture of CC5 and CC6 of PWMB

PWM5 (P2.0), CC7 and CC8 of PWMB simultaneously capture PWM7 (P2.2), CC3 and CC4 of PWMA simultaneously

Capture PWM3P (P1.4). In addition, use timer 0 to generate waveforms on P1.0, timer 1 to generate waveforms on P1.4, and

Timer 3 generates a waveform on P2.0, and Timer 4 generates a waveform on P2.2. The captured value is sent to the PC through the serial port.

Note: 1. The two-way capture module inside the chip is used to capture the same external pin at the same time, so there is no need to connect multiple external pins

connected.

2. Since the reset trigger signal is not set, the period value and the duty cycle value need to be subtracted accordingly.

---

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"
#include "stdio.h"

#define FOSC      12000000UL
#define BRT       (65536-FOSC/115200/4)
#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;

bit f1, f2, f3, f4;

void main()
{
    EAXFR = 1;           Enable      XFR
    CKCON = 0x00;        access to set external data bus speed to fastest
    WTST = 0x00;         Set the program code to wait for parameters,
                        //assign to CPU The speed of executing the program is set to the fastest
    POM0 = 0x00;
}

```

```

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

AUXR |= 0x80;                                //timer 0 use 1T model
AUXR |= 0x40;                                //timer 1 usage1T model
TMOD = 0x00;                                 //timer 0/1 cycle 16 Bit auto-reload mode
TL0 = T10K;                                  //timer 0 10K
TH0 = T10K >> 8;                            //timer 1 cycle 11K
TL1 = T11K;                                  //timer starts counting
TH1 = T11K >> 8;                            //timer starts counting
TR0 = 1;                                     //Enable timer interrupt
TR1 = 1;                                     //Enable timer interrupt
ET0 = 1;                                     //Enable timer interrupt
ET1 = 1;                                     //Enable timer interrupt

T3L = T12K;                                  //timer 3 cycle 12K
T3H = T12K >> 8;                            //timer 4 cycle 13K
T4L = T13K;                                  //Timer usage mode1T
T4H = T13K >> 8;                            //Enable timer interrupt
T4T3M = 0xaa;                               //Enable timer interrupt
IE2 |= 0x20;                                 //Enable timer interrupt
IE2 |= 0x40;

SCON = 0x52;
T2L = BRT;
T2H = BRT >> 8;
AUXR |= 0x15;

printf("PWM Test .\n");

PWMA_CCER1 = 0x00;                           //CC1 is input mode and is mapped TI1FP1 superior
PWMA_CCMR1 = 0x01;                           //CC2 to input mode and is mapped TI1FP2 superior
PWMA_CCMR2 = 0x02;                           //capture CC1 polarity to Rising edge CC2 capture function on
PWMA_CCER1 = 0x11;                           //CC1
PWMA_CCER1 |= 0x00;                           //Set capture polarity to Falling edge
PWMA_CCER1 |= 0x20;

PWMA_CCER2 = 0x00;                           //CC3 is input mode and is mapped TI3FP3 superior
PWMA_CCMR3 = 0x01;                           //CC4 to input mode and is mapped TI3FP4 superior
PWMA_CCMR4 = 0x02;                           //capture CC3 polarity to Rising edge CC4 capture function on
PWMA_CCER2 = 0x11;                           //CC3
PWMA_CCER2 |= 0x00;                           //Set capture polarity to Falling edge
PWMA_CCER2 |= 0x20;
PWMA_CR1 = 0x01;

PWMA_IER = 0x1e;                             //Enable CC1/CC2/CC3/CC4 catch interrupt

PWMB_CCER1 = 0x00;                           //CC5 is input mode and is mapped TI5FP5 superior
PWMB_CCMR1 = 0x01;                           //CC6 to input mode and is mapped TI5FP6 superior
PWMB_CCMR2 = 0x02;                           //capture CC5 polarity to Rising edge CC6 capture function on
PWMB_CCER1 = 0x11;                           //CC5
PWMB_CCER1 |= 0x00;

```

```

PWMB_CCER1 |= 0x20; //Set capture polarity to CC6 falling edge of

PWMB_CCER2 = 0x00; //CC7 is input mode and is mapped TI7FP8 superior
PWMB_CCMR3 = 0x01; //CC8 to input mode and is mapped TI7FP8 superior
PWMB_CCMR4 = 0x02; //capture CC7 polarity to the rising edge CC8 capture function on
PWMB_CCER2 = 0x11; //CC7
PWMB_CCER2 |= 0x00; //Set capture polarity to falling edge
PWMB_CCER2 |= 0x20; //CC8
PWMB_CR1 = 0x01;

PWMB_IER = 0x1e; //Enable CC5/CC6/CC7/CC8 catch 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 //produce CC1 waveform to P1.0 mouth
{
    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 //produce CC3 waveform to P1.4 mouth
{
    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
{
    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
{
    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 catch interrupt
    {
        PWMA_SR1 &= ~0x02;

        ccr = (PWMA_CCR1H << 8) + PWMA_CCR1L;          //read capture value
        cycle1 = ccr - ccr1; ccr1 = ccr; f1 = 1;          //calculation cycle
        //Save the current capture value
        //The cycle 1 and duty cycle of the waveform is captured, and the serial port is triggered to send
    }

    if (PWMA_SR1 & 0x04)                                //CC2 catch interrupt
    {
}

```

```

PWMA_SR1 &= ~0x04;

    ccr = (PWMA_CCR2H << 8) + PWMA_CCR2L;           // read capture value
    duty1 = ccr - ccr1;                                // Calculate the duty cycle
}

if (PWMA_SR1 & 0x08)                                // CC3 catch interrupt
{
    PWMA_SR1 &= ~0x08;

    ccr = (PWMA_CCR3H << 8) + PWMA_CCR3L;           // read capture value
    cycle2 = ccr - ccr3; ccr3 = ccr; f2 = 1;          // calculation cycle
    // Save the current capture value
    // The cycle 2nd duty cycle of the waveform is captured, and the serial port is triggered to send
}

if (PWMA_SR1 & 0x10)                                // CC4 catch interrupt
{
    PWMA_SR1 &= ~0x10;

    ccr = (PWMA_CCR4H << 8) + PWMA_CCR4L;           // read capture value
    duty2 = ccr - ccr3;                                // Calculate the duty cycle
}

void PWMB_ISR() interrupt PWMB_VECTOR
{
    unsigned int ccr;

    if (PWMB_SR1 & 0x02)                                // CC5 catch interrupt
    {
        PWMB_SR1 &= ~0x02;

        ccr = (PWMB_CCR5H << 8) + PWMB_CCR5L;           // read capture value
        cycle3 = ccr - ccr5; ccr5 = ccr; f3 = 1;          // calculation cycle
        // Save the current capture value
        // The cycle 3rd duty cycle of the waveform is captured, and the serial port is triggered to send
    }

    if (PWMB_SR1 & 0x04)                                // CC6 catch interrupt
    {
        PWMB_SR1 &= ~0x04;

        ccr = (PWMB_CCR6H << 8) + PWMB_CCR6L;           // read capture value
        duty3 = ccr - ccr5;                                // Calculate the duty cycle
    }

    if (PWMB_SR1 & 0x08)                                // CC7 catch interrupt
    {
        PWMB_SR1 &= ~0x08;

        ccr = (PWMB_CCR7H << 8) + PWMB_CCR7L;           // read capture value
        cycle4 = ccr - ccr7; ccr7 = ccr; f4 = 1;          // calculation cycle
        // Save the current capture value
        // The cycle 4th duty cycle of the waveform is captured, and the serial port is triggered to send
    }

    if (PWMB_SR1 & 0x10)                                // CC8 catch interrupt
    {
        PWMB_SR1 &= ~0x10;

        ccr = (PWMB_CCR8H << 8) + PWMB_CCR8L;           // read capture value
    }
}

```

```
duty4 = ccr - ccr7; // Calculate the duty cycle
}
}
```

STCMCU

### 23.8.13 Method of outputting PWM waveforms with duty cycles of 100% and 0% (in PWM1P)

example)

#### 23.8.13.1 Method 1: Set **PWMx\_ENO** to disable output **PWM**

---

```
//The test frequency is 11.0592MHz
#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMA_ENO &= ~0x01; //Prohibit PWM1P port output
                        //direct PWM1P port is, through GPO
                        //operation at this time VO The port register is forced to output a high level or a low level

    while (1);
}
```

---

#### 23.8.13.2 Method 2: Set **PWMx\_CCMRn** register to force output active/inactive level

C language code

---

```
//The test frequency is 11.0592MHz
#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    P1M0 = 0x00;
    P1M1 = 0x00;
```

---

```

P3M0 = 0x00;
P3M1 = 0x00;
P5M0 = 0x00;
P5M1 = 0x00;

PWMA_CCER1 = 0x00;                                //CC1 for output mode
PWMA_CCMR1 &= ~0x03;                             //CC1 Force output inactive level (duty      0%)
PWMA_CCMR1 |= 0x40;                            //CC1 cycle force output active level (duty 100%)
// PWMA_CCMR1 |= 0x50;                           //CC1 cycle force output active level (duty 100%) set high level as
                                                 //cycle enable active level
                                                 //Enable PWM1P
PWMA_ENO = 0x01;                                //enable main output
PWMA_BKR = 0x80;                                //start the timer
PWMA_CR1 = 0x01;

while (1);
}

```

### 23.8.14 PWM Complementary Output with Dead-Time Control

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"
#include "intrins.h"

void main(void)
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;

    PWMA_ENO = 0xFF;                                //IO output PWM
    PWMA_PS = 0x00;                                //00: PWM at P1

    *****
    PWMA_duty = [CCRx/(ARR + 1)]*100
    *****
    PWMA_PSCRH = 0x00;                            //Prescaler 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 registers, counters 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;
PWMA_CCER2 = 0x55;

PWMA_BKR = 0x80;
PWMA_IER = 0x02;
PWMA_CR1 = 0x01;

EA = 1;
while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        PWMA_SR1 &= ~0X02;
    }
}

```

### 23.8.15 PWM port for external interrupt (falling edge interrupt or rising edge interrupt)

---

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

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

    P1M1 = 0x00;
    P1M0 = 0x00;
    P3M1 = 0x00;
    P3M0 = 0x00;

    PWMA_CCER1 = 0x00;                             // Capture function input      TI1FP1 superior
    PWMA_CCMR1 = 0x01;                            mode and mapped to enable
    PWMA_CCER1 = 0x01;                            Set capture polarity to CC1 rising edge
    PWMA_CCER1 |= 0x00;                           CC1 //Set capture polarity to CC1 falling edge
// PWMA_CCER1 |= 0x02;
    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 Waveforms with Arbitrary Period and Arbitrary Duty Cycle

//The test frequency is 11.0592MHz

```

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void main()
{
    EAXFR = 1;
    CKCON = 0x00;
    WTST = 0x00;

    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;
    PWMA_CCMR1 = 0x60;
    PWMA_CCER1 = 0x01;
    PWMA_CCR1H = 0x01;
    PWMA_CCR1L = 0x00;
    PWMA_ARRH = 0x02;
    PWMA_ARRL = 0x00;
    PWMA_ENO = 0x01;
    PWMA_BKR = 0x80;
    PWMA_CR1 = 0x01;

    while (1);
}

```



**XFR**

Enable XFR  
access to set external data bus speed to fastest

Set the program code to wait for parameters,  
assign to CPU The speed of executing the program is set to the fastest

**CCMRx**  
Must be cleared to set to output mode CCERx close the channel  
before writing PWMA

**CC1**  
enable channel  
//Set duty cycle time

**PWM1P** port output  
enable main output  
//start the timer

### 23.8.17 Use PWM's CEN to start the PWMA timer to trigger the ADC in real time

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
    P3M1 = 0x00;

    ADC_CONTR = 0; //choose P1.0 for ADC input channel
    ADC_POWER = 1;
    ADC_EPWMT = 1;
    delay(); //wait ADC stable power
    EADC = 1;

    PWMA_CR2 = 0x10;
    PWMA_ARRH = 0x13;
    PWMA_ARRL = 0x38;
    PWMA_IER = 0x01;
    PWMA_CR1 = 0x01; //set up CEN start up PWMA Timer, real-time trigger ADC
    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 Period Repeated Triggering of ADC

//The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" //For header files, see Download Software
#include "intrins.h"

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    EA邢R = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
    P3M1 = 0x00;

    ADC_CONTR = 0; //choose P1.0 for ADC input channel
    ADC_POWER = 1;
    ADC_EPWMT = 1;
    delay();
    EA邢C = 1;

    PWMA_CR2 = 0x20; //The periodic update event is TRGO, for periodic triggering ADC
    PWMA_ARRH = 0x13;
    PWMA_ARRL = 0x38;
    PWMA_IER = 0x01;
    PWMA_CR1 = 0x01; //set up CEN start up PWMA timer
    EA = 1;

    while (1);
}

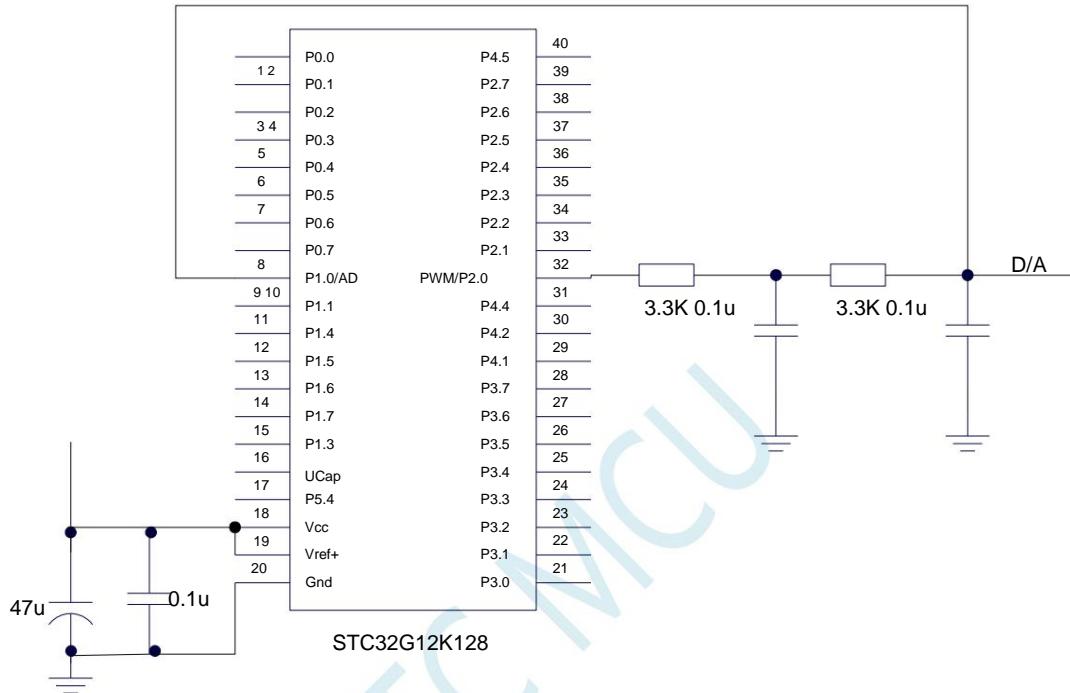
void ADC_ISR() interrupt 5
{
    ADC_FLAG = 0;
}

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 the STC32G series microcontroller can output a 16-bit PWM waveform, and then through two-stage low-pass filtering, a 16-bit DAC signal can be generated, and the DAC signal can be changed by adjusting the high-level duty cycle of the PWM waveform. The application circuit diagram is shown in the figure below, and the output DAC signal can be input to the ADC of the MCU for feedback measurement.



### **23.8.20 Complementary SPWM Using PWM**

Advanced PWM timers PWM1P/PWM1N, PWM2P/PWM2N, PWM3P/PWM3N, PWM4P/PWM4N Each channel can independently realize PWM output, or two complementary symmetrical outputs. The demonstration uses PWM1P, PWM1N to generate complementary SPWM. The main clock is 24MHZ, the PWM clock is 1T, the PWM period is 2400, the dead zone is 12 clocks (0.5us), the sine wave table uses 200 points, and the output sine wave frequency =  $24000000 / 2400 / 200 = 50$  Hz. This program is just a demonstration program of SPWM, the user can modify the PWM period and the number and amplitude of the sine wave through the above calculation method. The output frequency of this program is fixed. If frequency conversion is required, please design the frequency conversion scheme by yourself.

//The test frequency is **11.0592MHz**

```
//#include "stc8h.h"  
//#include "stc32g.h" // For header files, see Download Software  
//#include "intrins.h"  
  
#define MAIN_Fosc 24000000L // define the master clock  
  
typedef unsigned char u8;  
typedef unsigned int u16;
```

```

typedef unsigned long          u32;

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

/* **** 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, 2261, 2275, 2289,
    2302, 2314, 2324, 2342, 2356, 2365, 2368,
    2369, 2365, 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, 407, 382, 357, 334, 311, 290, 269,
    249, 230, 212, 195, 179, 165, 151, 138, 126, 116,
    106, 98, 71, 70, 71, 98, 106, 116, 126, 138, 151,
    90,     84,     79,     75,     72,     165, 129, 125, 10,
    72,     75,     79,     84,     90,     249, 269, 290,
    311, 334, 357, 382, 407, 433, 459, 487, 515, 544,
    574, 604, 635, 666, 698, 730, 763, 797, 830, 865,
    899, 934, 969, 1005, 1040, 1076, 1112, 1148, 1184,
}

```

```

};

u16 PWMA_Duty;
u8 PWM_Index;

//SPWM table lookup index

/* ***** main function ***** */
main function *****
{
    EAXFR = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                     //assign to CPU The speed of executing the program is set to the fastest

    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

    PWMA_Duty = 1220;

    PWMA_CCER1 = 0x00; // Write CCMRx must be cleared before CCxE close the channel
    PWMA_CCER2 = 0x00; // Channel Mode Configuration
    PWMA_CCMR1 = 0x60; // Configure Channel Output Enable and Polarity

// PWMA_CCMR2 = 0x60;
// PWMA_CCMR3 = 0x60;
// PWMA_CCMR4 = 0x60;
    PWMA_CCER1 = 0x05; //Set cycle time
// PWMA_CCER2 = 0x55;

    PWMA_ARRH = 0x09; //Set duty cycle time
    PWMA_ARRL = 0x60;

    PWMA_CCR1H = (u8)(PWMA_Duty >> 8);
    PWMA_CCR1L = (u8)(PWMA_Duty);

    PWMA_DTR = 0x0C; //Set dead time

    PWMA_ENO = 0x00; //enable output
    PWMA_ENO |= ENO1P; //enable output
    PWMA_ENO |= ENO1N; //enable output
// PWMA_ENO |= ENO2P;
// PWMA_ENO |= ENO2N;
// PWMA_ENO |= ENO3P;
// PWMA_ENO |= ENO3N;
// PWMA_ENO |= ENO4P;
// PWMA_ENO |= ENO4N;
//enable output

    PWMA_PS = 0x00; // Advanced PWM Channel output pin selection bit
    PWMA_PS |= PWMA_3; //selection PWMA_3 aisle
// PWMA_PS |= PWMB_3;
// PWMA_PS |= PWM3_3;
// PWMA_PS |= PWM4_3;
//selection PWMB_3 aisle
//selection PWM3_3 aisle
//selection PWM4_3 aisle

    PWMA_BKR = 0x80; //enable main output
    PWMA_IER = 0x01; //enable interrupt

```

```

PWMA_CR1 |= 0x01;                                // start the timer

EA = 1;                                         // Turn on total 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 - Frequency Adjustable - Pulse Count

///The test working 24MHz  
\*\*\*\*\* frequency \*\*\*\*\*  
is the PWM mode option. It realizes the high-speed adjustable output pulse width and output frequency.  
P6 , 10ms PWM, 10 Stop output after pulses  
1ms PWM .  
, 24MHZ ).  
(\*\*\*\*\*

---

```

#ifndef include "stc8.h"
#include "stc32g.h"                                // For header files, see Download Software
#include "intrins.h"

#define MAIN_Fosc      24000000L

typedef unsigned char    u8;
typedef unsigned int     u16;
typedef unsigned long    u32;

/********************************************* user-defined macro *****/
#define Timer0_Reload  (65536UL-(MAIN_Fosc / 1000))          // Timer0 Interrupt frequency 1000 sub-second

/*********************************************
```

---

```

#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

/* ***** local variable declaration *****/
bit B_1ms;
bit PWM1_Flag;

u16 Period;
u8 Counter;
u8 msSecond;

void UpdatePwm(void);
void TxPulse(void);

/* ***** main function *****/
main(void)
{
    EAXFR = 1;                                Enable      XFR
                                                access to set external data bus speed to fastest
    CKCON = 0x00;                             Set the program code to wait for parameters,
    WTST = 0x00;                               //assign to CPU The speed of executing the program is set to the fastest

    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 initialization
    AUXR = 0x80;                            // Timer0 set as 1T,16 bits timer auto-reload,
    TH0 = (u8)(Timer0_Reload / 256);

```

```

TL0 = (u8)(Timer0_Reload % 256); //Timer0 interrupt enable
ET0 = 1; //Tiner0 run
TR0 = 1;

PWMA_ENO = 0x00; //enable output
PWMA_ENO |= ENO1P;

PWMA_PS = 0x00; //Advanced PWM Channel output pin selection bit
PWMA_PS |= PWM1_3; //options PWM1_3 aisle

UpdatePwm();
PWMA_BKR = 0x80; //enable main output
PWMA_CR1 |= 0x01; //start the timer

P40 = 0; //to supply power
EA = 1; //Turn on total interrupt

while (1)
{
    if(B_1ms)
    {
        B_1ms = 0;
        msSecond++;
        if(msSecond >= 10)
        {
            msSecond = 0;
            TxPulse(); //10ms start once PWM output
        }
    }
}

// **** send pulse function ****/
void TxPulse(void)
{
    PWMA_CCER1 = 0x00; //CCMRx mode output must be CCxE close the channel
    PWMA_CCMR1 = 0x60; //Write PWM1 cleared before
    PWMA_CCER1 = 0x01; //Set CC1E channel active high
    PWMA_SR1 = 0; //Enable Clear Flag
    PWMA_CNTR = 0; //Clear the counter
    PWMA_IER = 0x02; //Enable capture comparison 1 interrupt
}

// **** Timer0 1ms interrupt function ****/
void timer0(void) interrupt 1
{
    B_1ms = 1;
    if(PWM1_Flag)
    {
        Period++; //Period increment
        if(Period >= 0x1000) PWM1_Flag = 0;
    }
    else
    {
        Period--; //period decrement
        if(Period <= 0x0100) PWM1_Flag = 1;
    }
    UpdatePwm(); //Set period, duty cycle
}

```

```
/***************************************** PWM
interrupt function *****/
void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        PWMA_SR1 &= ~0X02;                                //clear flag

        Counter++;
        if(Counter >= 10)                                //count 10 off after pulses  PWM counter
        {
            Counter = 0;
            PWMA_CCER1 = 0x00;                            //      CCMRx Must be cleared to CCxE close the channel
            PWMA_CCMR1 = 0x40;                            //Write PWM1 force to inactive level before
            PWMA_CCER1 = 0x01;                            //Set CC1E channel active high
            PWMA_IER = 0x00;                            //Enable Disable Interrupt
        }
    }
}

//=====
function: UpdatePwm(void)
description update PWM cycle duty cycle
parameter : none.
version : none.
////////////////: V1.0, 2012-11-22
//=====

void UpdatePwm(void)
{
    PWMA_ARR = Period;
    PWMA_CCR1 = (Period >> 1);                      //Set duty cycle time Period/2
}
```

## 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, and adds high-speed mode.

When the system runs at a lower operating frequency, the high-speed advanced PWM can work at frequencies up to 144M. so as to reduce core power consumption, the purpose of improving peripheral performance

### 24.1 Related registers

symbol	describe	address	Bit addresses and symbols								reset value	
			B7	B6	B5	B4	B3	B2	B1	B0		
HSPWMA_CFG	High Speed PWMA Configuration Register	7EFBF0H	-	-	-	-	-	INTEN	ASYNCE	CEN	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	ASYNCE	CEN	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)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
HSPWMA_CFG 7EFBF0H	-	-	-	-	-	-	INTEN	ASYNCE	CEN
HSPWMB_CFG 7EFBF4H	-	-	-	-	-	-	INTEN	ASYNCE	CEN

ASYNCE: Asynchronous Control Mode Enable Bit

0: Disable asynchronous control.

1: Enable asynchronous control mode.

Note: When the asynchronous control is turned off, the advanced PWMA/Advanced PWMB is the traditional mode, at this time the advanced PWM will automatically select the system working mode.

The operating frequency of the PWM is the same as the operating frequency of the system; if the PWM works in the high-speed mode, it is necessary to enable the

Step control mode, at this time, the PWM clock can choose the main clock (MCLK) or the 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, if you need to respond to the interrupt of Advanced PWMA/Advanced PWMB, you must enable the INTEN bit

#### 24.1.2 HSPWM Address Register (HSPWMn\_AD)

symbol	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]: The lower 7 bits of the special function register address of the advanced PWMA/PWMB

0: Disable asynchronous control.

1: Enable asynchronous control mode.

RW/BUSY: read and write control bits, status bits

Write 0: write the special function register of PWMA/PWMB asynchronously.

Write 1: read the special function register of PWMA/PWMB asynchronously.

Read 0: Asynchronous read and write have completed

Read 1: Asynchronous read and write in progress, in busy state

## 24.1.3 HSPWM Data Register (HSPWMn\_DAT)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0		
HSPWMn_DAT	7	EFBF2H	DATA[7:0]								
HSPWMn_DAT	7	EFBF6H	DATA[7:0]								

DATA[7:0]: Special function register data for advanced PWMA/PWMB

Write: Write data to the special function registers of the advanced PWMA/PWMB.

Read: Read data from the Special Function Registers of Advanced PWMA/PWMB.

Asynchronously read the special function register steps of **PWMA** : (PWMB is similar)

1. Read **HSPWMn\_ADR**, wait for **BUSY** to be 0, and confirm that the previous asynchronous read and write has been completed
2. Write the lower 7 bits of the special function register of **PWMA** into **HSPWMn\_ADR**, and set "1" **HSPWMn\_ADR.7** at the same time
3. Read **HSPWMn\_ADR** and wait for **BUSY** to be 0
4. Read **HSPWMn\_DAT**

Asynchronously write the special function register steps of **PWMA** : (PWMB is similar)

1. Read **HSPWMn\_ADR**, wait for **BUSY** to be 0, and confirm that the previous asynchronous read and write has been completed
2. Write the data that needs to be written to the special function register of **PWMA** into **HSPWMn\_DAT**
3. Write the lower 7 bits of the special function register of **PWMA** into **HSPWMn\_ADR**, and clear "0" **HSPWMn\_ADR.7** at the same time
4. Read **HSPWMn\_ADR** and wait for **BUSY** to be 0. (You can skip this step and continue to execute other codes to improve system efficiency)

Special attention: Special function registers **PWMA\_PS** and **PWMB\_PS** belong to port control registers, not **PWMA** and **PWMB**

Register group, so regardless of whether the asynchronous control mode of **PWM** is enabled, the **PWMA\_PS** and **PWMB\_PS** registers can only be used

Read and write in normal synchronous mode

## 24.2 Example Program

### 24.2.1 Enabling High-Speed Mode with Advanced PWM (Asynchronous Mode)

//The test frequency is 12MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#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);
    HSPWMA_ADR = addr | 0x80;
    // Wait for the previous asynchronous read and write to complete
    // Setting the indirect access address only needs to set the low-order bits of the original address
    // HSPWMA_ADR The highest bit of the register is written to indicate read data
}

```

```

while (HSPWMA_ADR & 0x80); dat
= HSPWMA_DAT;

return dat;
}

void WritePWMA(char addr, char dat)
{
    while (HSPWMA_ADR & 0x80);
    HSPWMA_DAT = dat;
    HSPWMA_ADR = addr & 0x7f;
}

void main()
{
    EAXFR = 1; // enable access XFR
    CKCON = 0x00; // Set the external data bus speed to the fastest
    WTST = 0x00;

    // choosePLL output clock
    #if (PLL_SEL == PLL_96M)
        CLKSEL &= ~CKMS;
    #elif (PLL_SEL == PLL_144M)
        CLKSEL |= CKMS;
    #else
        CLKSEL &= ~CKMS;
    #endif

    // choosePLL The input clock frequency division ensures that the input clock is 12M
    USBCLK &= ~PCKI_MSK;
    #if (FOSC == 12000000UL)
        USBCLK |= PCKI_D1;
    #elif (FOSC == 24000000UL)
        USBCLK |= PCKI_D2;
    #elif (FOSC == 48000000UL)
        USBCLK |= PCKI_D4;
    #elif (FOSC == 96000000UL)
        USBCLK |= PCKI_D8;
    #else
        USBCLK |= PCKI_D1;
    #endif

    // start upPLL
    USBCLK |= ENCKM;

    delay();

    // chooseHSPWM/HSSPI clock
    #if (HSCK_SEL == HSCK_MCLK)
        CLKSEL &= ~HSIOCK;
    #elif (HSCK_SEL == HSCK_PLL)
        CLKSEL |= HSIOCK;
    #else
        CLKSEL &= ~HSIOCK;
    #endif

    HSCLKDIV = 0;
}

// Wait for the current asynchronous read to complete
// read asynchronous data

// Wait for the previous asynchronous read and write to complete
// Prepare data to be written
// To set the indirect access address, you only need to set the XFR lower bits of address
// HSPWMA_ADR highest bit of the original register to 0 indicates write data

// enable access XFR
// choosePLL of 96M as PLL the output clock of
// choosePLL of 144M as PLL the output clock of
// Default selectionPLL of 96M as PLL the output clock of
// PLL input clock 1 frequency division
// PLL input clock 2 frequency division
// PLL input clock 4 frequency division
// PLL input clock 8 frequency division
// defaultPLL input clock 1 frequency division
// EnablePLL frequency doubling
// wait PLL frequency lock
// HSPWM/HSSPI Select the main clock as the clock source
// HSPWM/HSSPI choosePLL The output clock is the clock source
// defaultHSPWM/HSSPI Select the main clock as the clock source
// HSPWM/HSSPI The clock source is not divided

```

```

HSPWMA_CFG = 0x03; //Enable PWMA Asynchronous access to related registers

//set asynchronously PWMA related registers
WritePWMA((char)&PWMA_CCER1, 0x00); //CC1 for output mode
WritePWMA((char)&PWMA_CCMR1, 0x00); //OC1REF output PWM1(CNT<CCR output active level 1)
WritePWMA((char)&PWMA_CCMR1, 0x60); //Enable the output function on PWM1
WritePWMA((char)&PWMA_CCER1, 0x05); //Enable signal output to port
WritePWMA((char)&PWMA_ENO, 0x03); //enable main output
WritePWMA((char)&PWMA_BKR, 0x80); //set output PWM duty cycle
WritePWMA((char)&PWMA_CCR1H, 200 >> 8); //set output PWM cycle
WritePWMA((char)&PWMA_CCR1L, 200);
WritePWMA((char)&PWMA_ARRH, 1000 >> 8);
WritePWMA((char)&PWMA_ARRL, 1000); //Set Complementary Symmetrical PWM dead zone
WritePWMA((char)&PWMA_DTR, 10); //Output to start counting
WritePWMA((char)&PWMA_CR1, 0x01);

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 integrate USB2.0/USB1.1 compatible full-speed USB, 6 bidirectional endpoints, and support 4 endpoint transmission modes. mode (control transfer, interrupt transfer, bulk transfer and isochronous transfer), each endpoint has a 64-byte buffer.

The USB module has a total of 1280 bytes of FIFO, the structure is as follows:

Control	Endp 0-IN	64 Bytes	64 Bytes	Endp 0-OUT Control
INT	Endp 1-IN	64 Bytes	64 Bytes	Endp 1-OUT INT
BULK	Endp 2-IN	64 Bytes	64 Bytes	Endp 2-OUT BULK
ISO	Endp 3-IN	64 Bytes	64 Bytes	Endp 3-OUT ISO
INT	Endp 4-IN	128 Bytes	256 Bytes	Endp 4-OUT INT
BULK	Endp 5-IN	128 Bytes	256 Bytes	Endp 5-OUT BULK
ISO				ISO

## 25.1 USB related registers

symbol	describe	address	Bit addresses and symbols								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	YST[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)

Symbol	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: Enable USB function and USB clock

ENRST: USB Reset Set Control Bit

0: Disable USB reset settings

1: Enable USB reset

PS2M: PS2 mode function control bit

0: Disable PS2 mode function

1: Enable PS2 mode function

PUEN: 1.5K pull-up resistor control bit on DP/DM port

0: disable pull-up resistor

1: Enable pull-up resistor

PDEN: 500K pull-down resistor control bit on DP/DM port

0: Disable pull-down resistor

1: Enable pull-down resistor

DFREC: Differential Receive Status Bit (read only)

0: The current 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 and write when PS2 is 1)

0: The current D+ is logic 0 level

1: The current D+ is logic 1 level

DM: D-port status (read-only when PS2 is 0, read-write when PS2 is 1)

0: The current D- is logic 0 level

1: The current D- is logic 1 level

## 25.1.2 USB Clock Control Register (USBCLK)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
USBCLK	DCH	ENCKM	PCKI[1:0]		CRE	TST	_USB	TST	_PHY

ENCKM: PLL Multiplier Control

0: Disable PLL frequency multiplication

1: Enable PLL frequency multiplication

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: Disable clock tracking

1: Enable clock frequency tracking

TST\_USB: USB test mode

0: Disable USB test mode

1: Enable USB test mode

TST\_PHY: PHY test mode

0: Disable PHY test mode

1: Enable PHY test mode

PHYTST[1:0]: USB PHY test

PHYTST[1:0]	Way	DP DM
-------------	-----	-------

00 Mode 0: Normal 01	X	X
Mode 1: Force "1" 10 Mode 2:	1	0
Force "0" 11 Mode 3: Force	0	1
single-ended "0" 0		0

### 25.1.3 USB ADDRESS REGISTER (USBADR)

Symbol address B7	B6	B5	B4 B3	B2	B1	B0
USBADR	FCH	BUSY AUTORD		UADR[5:0]		

BUSY: USB register read busy flag

write 0: meaningless

Write 1: start the read operation of the USB indirect register, the address is set by USBADR

Read 0: Data in USBDAT register is valid

Read 1: Invalid data in USBDAT register, USB is reading indirect register

AUTORD: USB register automatic read flag, used for block read of USB FIFO

Write 0: The BUSY flag must be written every time the indirect USB register is read

Write 1: When software reads USBDAT, the next USB indirect register read will start automatically (USBADR unchanged)

UADR[5:0]: Address of USB indirect register

### 25.1.4 USB ADDRESS DATA REGISTER (USBDAT)

Symbol address B7	B6	B5	B4 B3	B2	B1	B0
USBDAT	ECH		UDAT[7:0]			

UDAT[7:0]: used to indirectly read and write 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	

symbol	describe	address	Bit addresses and symbols								reset value	
			B7	B6	B5	B4	B3	B2	B1	B0		
FADDR USB	Function Address Register	00H	UPDATE	UADDR[6:0]								0000,0000
POWER USB	Power Management Register	01H	ISOUD	-	-	-	USBRST	USBRSL	USBSS	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	-		xx00,0000x

INTRUSB USB	power interrupt flag	06H	-	-	-	-	SOFIF	RSTIF	RSUIF	SUSIF xxxx	0000
INTRIN1E USB	endpoint IN interrupt enable bit 07H		-	-	EP5INIE EP4INIE EP3INIE EP2INIE EP1INIE					EPOIE xx1,1111	
INTROUT1E USB	endpoint OUT interrupt enable bit 09H		-	-	EP5OUTIE EP4OUTIE EP3OUTIE EP2OUTIE EP1OUTIE					xx11,111x	
INTRUSBE USB	Power Interrupt Enable Bit	0BH	-	-	-	-	SOFIE	RSTIE	RSUIE	SUSIE xxxx	0110
FRAME1 USB	data frame number low byte	0CH	FRAME[7:0]							0000,0000	
FRAME2 USB	data frame number high byte	0DH	-	-	-	-	-	FRAME[10:8]			xxxx,x000
INDEX USB	Endpoint Index Register	0EH	-	-	-	-	-	INDEX[2:0]			xxxx,x000
INMAXP IN	endpoint maximum packet size 10H		INMAXP[7:0]							0000,0000	
CSR0 Endpoint 0 Control Status Register 11H	\$SUEND 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 END DMA FCDT				-	-	-		0010,0xxx
OUTMAXP OUT	endpoint maximum packet size 13H		OUTMAXP[7:0]							0000,0000	
OUTCSR1 OUT	Endpoint Control Status Register 1	14H	CLRDT STSTL SDSTL	FLUSH DATERR	OVRUN FIFO	FUL OPRDY	0000,0000				
OUTCSR2 OUT	endpoint control status register 2	15H	AUTOCLR ISO END DMA DMAMD			-	-	-	-		0000,xxxx
COUNT0 OUT	length of endpoint 0	16H		OUTCNT[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 Endpoint 0 FIFO access register 20H			FIFO0[7:0]							0000,0000	
FIFO1 Endpoint 1 FIFO access register 21H			FIFO1[7:0]							0000,0000	
FIFO2 Endpoint 2 FIFO access register 22H			FIFO2[7:0]							0000,0000	
FIFO3 Endpoint 3 FIFO access register 23H			FIFO3[7:0]							0000,0000	
FIFO4 Endpoint 4 FIFO access register 24H			FIFO4[7:0]							0000,0000	
FIFO5 Endpoint 5 FIFO access register 25H			FIFO5[7:0]							0000,0000	
UTRKCTL USB	Trace Control Register	30H	FTM1 FTM0		INTV[1:0]	ENST5	RES[2:0]			1011,1011	
UTRKSTS USB	Trace Status Register	31H	INTVCNT[3:0]			STS[1:0]	TST_UTRK	UTRK_RDY	1111	00x0	

### 25.2.1 USB Function Address Register (FADDR)

Symbol address B7		B6	B5	B4 B3	B2	B1	B0	
FADDR	00H	UPDATE	UADDR[6:0]					

UPDATE: Update USB function address

0: The last UADDR address has taken effect

1: The last UADDR address has not yet taken effect

UADDR[6:0]: Save the 7-bit function address of the USB

### 25.2.2 USB Power Control Register (POWER)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
POWER	01H	ISOUD	-	-	- USBRST USBRST	USBSUS ENSUS		

ISOUD (ISO Update): ISO Update

0: When software writes "1" to IPRDY, USB sends data packet after receiving next IN token

1: When the software writes "1" to IPRDY, the USB sends a data packet after receiving the SOF token, if it receives IN before the SOF token token, 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 reset status information on the current bus

0: No reset signal detected on the bus

1: A reset signal is detected on the bus

#### USBRSU (USB Resume): USB Resume Control Bit

Software forces a resume signal on the bus to remotely wake a USB device from suspend. when the USB is in

In suspend mode (USBSUS=1), writing "1" to this bit will force a resume signal to be generated on the USB bus, the software should be within 10-15ms

Write "0" to this bit to end the recovery signal. After software writes "0" to USBRSU, a USB recovery interrupt will be generated.

The file will automatically clear USBSUS to "0"

#### USBSUS (USB Suspend): USB Suspend Control Bit

This bit is set by hardware when the USB enters suspend mode. When the resume signal is forced on the bus in software or after

Hardware automatically clears this bit to '0' when a resume signal is detected on the bus and after reading the INTRUSB register.

#### ENSUS (Enable Suspend Detection): Enable USB Suspend Detection

0: Disable suspend detection, USB will ignore suspend signals on the bus

1: Enable suspend detection, when a suspend signal on the bus is detected, the USB will enter the suspend mode

### 25.2.3 USB Endpoint IN Interrupt Flag (INTRIN1)

Symbol	address	B7		B6	B5	B4	B3	B2	B1	B0
INTRIN1	02H	-	-	EP5INIF EP4	NIF EP3INIF	EP2INIF EP1	NIF EP0IF			

EP5INIF: IN Interrupt Flag for Endpoint 5

0: IN interrupt of endpoint 5 is invalid

1: IN interrupt of endpoint 5 is valid

EP4INIF: IN Interrupt Flag for Endpoint 4

0: IN interrupt of endpoint 4 is invalid

1: IN interrupt of endpoint 4 is valid

EP3INIF: IN interrupt flag for endpoint 3

0: IN interrupt of endpoint 3 is invalid

1: IN interrupt of endpoint 3 is valid

EP2INIF: IN Interrupt Flag for Endpoint 2

0: IN interrupt of endpoint 2 is invalid

1: IN interrupt of endpoint 2 is valid

EP1INIF: IN Interrupt Flag for Endpoint 1

0: IN interrupt of endpoint 1 is invalid

1: IN interrupt of endpoint 1 is valid

EP0IF: IN/OUT interrupt flag for endpoint 0

0: IN/OUT interrupt of endpoint 0 is invalid

1: IN/OUT interrupt active for endpoint 0

After software reads INTRIN1 register, hardware will automatically clear all interrupt flags in INTRIN1

### 25.2.4 USB Endpoint OUT Interrupt Flag (INTROUT1)

Symbol	address	B7		B6	B5	B4	B3	B2	B1	B0
INTROUT1	04H	-	-	EP5OUTIF EP4OUTIF EP3OUTIF EP2OUTIF	EP1OUTIF					-

EP5OUTIF: OUT interrupt flag for endpoint 5

0: OUT interrupt of endpoint 5 is invalid

1: OUT interrupt of endpoint 5 is valid

EP4OUTIF: OUT interrupt flag for endpoint 4

- 0: OUT interrupt of endpoint 4 is invalid
- 1: OUT interrupt of endpoint 4 is valid

EP3OUTIF: OUT interrupt flag for endpoint 3

- 0: OUT interrupt of endpoint 3 is invalid
- 1: OUT interrupt of endpoint 3 is valid

EP2OUTIF: OUT interrupt flag for endpoint 2

- 0: OUT interrupt of endpoint 2 is invalid
- 1: OUT interrupt of endpoint 2 is valid

EP1OUTIF: OUT interrupt flag for endpoint 1

- 0: OUT interrupt for endpoint 1 is invalid
- 1: OUT interrupt of endpoint 1 is valid

After software reads INTROUT1 register, hardware will automatically clear all interrupt flags in INTROUT1

## 25.2.5 USB Power Interrupt Flag (INTRUSB)

Symbol address	B7		B6	B5	B4 B3		B2	B1	B0
INTRUSB	06H	-	-	-	-	SOFIF R\$TIF RSUIF			SUSIF

SOFIF: USB start of frame interrupt flag

- 0: USB frame start signal interrupt is invalid
- 1: USB frame start signal interrupt is valid

RSTIF: USB reset signal interrupt flag

- 0: USB reset signal interrupt is invalid
- 1: USB reset signal interrupt is valid

RSUIF: USB Resume Signal Interrupt Flag

- 0: USB resume signal interrupt is invalid
- 1: USB resume signal interrupt valid

SUSIF: USB Suspend Signal Interrupt Flag

- 0: USB suspend signal interrupt is invalid
- 1: USB suspend signal interrupt is valid

After software reads INTRUSB register, hardware will automatically clear all interrupt flags in INTRUSB

## 25.2.6 USB Endpoint IN Interrupt Enable Register (INTRIN1E)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
INTRIN1E	07H	-	-	EP5INIE EP4NIE EP3INIE	EP2INIE EP1NIE EPOIE				

EP5INIE: IN Interrupt Control Bit for Endpoint 5

- 0: Disable IN interrupt for endpoint 5
- 1: Enable IN interrupt for endpoint 5

EP4INIE: IN Interrupt Control Bit for Endpoint 4

- 0: Disable IN interrupt for endpoint 4
- 1: Enable IN interrupt for endpoint 4

EP3INIE: IN Interrupt Control Bit for Endpoint 3

- 0: Disable IN interrupt of endpoint 3
- 1: Enable IN interrupt for endpoint 3

**EP2INIE:** IN Interrupt Control Bit for Endpoint 2

0: Disable IN interrupt for endpoint 2

1: Enable IN interrupt for endpoint 2

**EP1INIE:** IN Interrupt Control Bit for Endpoint 1

0: Disable IN interrupt for endpoint 1

1: Enable IN interrupt for endpoint 1

**EP0IE:** IN/OUT Interrupt Control Bit for Endpoint 0

0: Disable IN/OUT interrupt for endpoint 0

1: Enable IN/OUT interrupt for endpoint 0

## 25.2.7 USB Endpoint OUT Interrupt Enable Register (INTROUT1E)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
INTROUT1E 09H		-	-	EP5OUTIE	EP4OUTIE	EP3OUTIE	EP2OUTIE	EP1OUTIE	-

**EP5OUTIE:** OUT interrupt control bit for endpoint 5

0: Disable OUT interrupt for endpoint 5

1: Enable OUT interrupt for endpoint 5

**EP4OUTIE:** OUT interrupt control bit for endpoint 4

0: Disable the OUT interrupt of endpoint 4

1: Enable OUT interrupt for endpoint 4

**EP3OUTIE:** OUT interrupt control bit for endpoint 3

0: Disable OUT interrupt of endpoint 3

1: Enable OUT interrupt for endpoint 3

**EP2OUTIE:** OUT interrupt control bit for endpoint 2

0: Disable the OUT interrupt of endpoint 2

1: Enable OUT interrupt for endpoint 2

**EP1OUTIE:** OUT interrupt control bit for endpoint 1

0: Disable OUT interrupt for endpoint 1

1: Enable OUT interrupt for endpoint 1

## 25.2.8 USB Power Interrupt Enable Register (INTRUSB)

Symbol address	B7		B6	B5	B4 B3		B2	B1	B0
INTRUSB	0BH	-	-	-	-	SOFIE R\$TIE RSUIE			SUSIE

**SOFIE:** USB Start of Frame Signal Interrupt Control Bit

0: Disable USB frame start signal interrupt

1: Enable USB frame start signal interrupt

**RSTIE:** USB reset signal interrupt control bit

0: Disable USB reset signal interrupt

1: Enable USB reset signal interrupt

**RSUIE:** USB Resume Signal Interrupt Control Bit

0: Disable USB resume signal interrupt

1: Enable USB resume signal interrupt

**SUSIE:** USB Suspend Signal Interrupt Control Bit

0: Disable USB suspend signal interrupt

1: Enable USB suspend signal interrupt

### 25.2.9 USB Data Frame Number Register (FRAMEn)

Symbol address B7	B6	B5	B4 B3 B2	B1	B0
FRAME1	0CH	FRAME[7:0]			
FRAME2	0DH	-	-	-	FRAME[10:8]

FRAME[10:0]: 11-bit frame number used to save the last received data frame

### 25.2.10 USB Endpoint Index Register (INDEX)

Symbol address B7	B6	B5	B4 B3 B2	B1	B0
INDEX	0EH	-	-	INDEX[2:0]	

INDEX[2:0]: select USB endpoint

INDEX[2:0] target endpoint
000 endpoint 0
001 Endpoint 1
010 Endpoint 2
011 Endpoint 3
100 endpoint 4
101 Endpoint 5

### 25.2.11 Maximum packet size for IN endpoints (INMAXP)

Symbol address B7	B6	B5	B4 B3 B2	B1	B0
INMAXP	10H	INMAXP[7:0]			

INMAXP[7:0]: Set the size of the maximum data packet of the IN endpoint of the USB (Note: The data packet is in units of 8 bytes. For example, if necessary

To set the packet 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)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0		
CSR0	11H	SSUEND	SOPRDY	SDSTL	SUEND	DATEND	STSTL	IPRDY	OPRDY

#### SSUEND (Serviced Setup End)

SETUP End event processing completion flag. After processing the SETUP end event (SUEND), the software needs to set SSUEND Flag bit, the hardware will automatically clear the SUEND bit to "0" when it detects that SSUEND is written to "1".

#### SOPRDY (Serviced OPRDY)

OPRDY event processing completion flag. After processing the packet received from endpoint 0, software needs to set the SOPRDY flag bit, the hardware will automatically clear 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 After the signal is sent, the hardware automatically clears this bit to "0".

#### SUEND (Setup End)

SETUP Installation package end flag. When a control transfer ends before software writes a '1' to the DATAEND bit, the hardware

Read bit "1". When software writes '1' to SSUEND, hardware clears this bit to '0'.

#### DATEND (Data End)

Data ends. Software should write '1' to this bit under the following conditions:

1. When the firmware writes "1" to IPRDY after sending the last data packet;
2. After sending a zero-length packet, the firmware writes "1" to IPRDY;
3. When the firmware writes "1" to SOPRDY after receiving the last data packet;

This bit will be automatically cleared to "0" by hardware

#### STSTL (Sent Stall)

STALL signal transmission completion flag. After sending the STALL signal, the hardware sets this bit to "1". This bit must be cleared to '0' by software.

#### IPRDY (IN Packet Ready)

IN Packet preparation complete flag. Software should set this bit to '1' after packing a data to be transmitted into the FIFO of endpoint 0.

Hardware clears this bit to '0' when one of the following conditions occurs:

1. When the data packet has been sent;
2. When the data packet is overwritten by a SETUP packet;
3. When the data packet is covered by an OUT packet;

#### OPRDY (OUT Packet Ready)

OUT Packet preparation complete flag. When an OUT packet is received, the hardware sets this read-only bit to "1" and generates an interrupt. Should

The bit is cleared to '0' only when software writes a '1' to the SOPRDY bit.

When you need to get/set this information, you first have to use INDEX to select the target endpoint 0

## 25.2.13 IN Endpoint Control Status Register 1 (INCSR1)

Symbol address B7			B6	B5	B4	B3	B2	B1	B0
INCSR1	11H	-	CLRDT	STSTL	SDSTL	FLUSH	UNDRUN	FIFONE	IPRDY

CLRDT (Clear Data Toggle): Reset the IN data toggle bit.

When the IN endpoint needs to reset the data toggle bit to "0" due to being reconfigured or STALL, the software needs to send this data bit write "1".

STSTL (Sent Stall): STALL signal transmission completion flag.

When the STALL signal is sent, the hardware will set this bit to "1" (the FIFO is cleared and the IPRDY bit is cleared to "0").

This flag must be cleared to '0' by software.

SDSTL (Send Stall): STALL signal send request bit.

Software should write a '1' to this bit to generate a STALL signal in response to an IN token. Software should write a '0' to this bit to end STALL signal. This bit has no effect on ISO mode.

FLUSH (FIFO Flush): Clears the next packet of the FIFO of the IN endpoint.

Writing a '1' to this bit will clear the IN endpoint FIFO for the next packet to be transmitted. FIFO pointer is reset, IPRDY bit is cleared "0". If the FIFO contains multiple packets, software must write '1' to FLUSH for each packet. When the FIFO is emptied After completion, the hardware will clear the FLUSH bit to "0".

UNDRUN (Data Underrun): Insufficient data.

The function of this bit depends on how the IN endpoint is:

ISO mode: This bit is set to "1" when IPRDY is "0" and a zero-length packet is sent after receiving an IN token.

Interrupt/Bulk Mode: This bit is set to '1' when a NAK is used as a response to an IN token.

This bit must be cleared to '0' by software.

FIFONE (FIFO Not Empty): FIFO not empty flag of IN endpoint

0: The FIFO of the IN endpoint is empty

1: The FIFO of the IN endpoint contains one or more packets

**IPRDY (IN Packet Ready):** The IN data packet is ready to complete flag.

Software should set this bit to '1' after packing a data to be transmitted into the endpoint's FIFO. When one of the following conditions occurs,

Hardware clears this bit to "0":

1. When the data packet has been sent;
2. Automatic setting is enabled (AUTOSET = '1') and the FIFO data packet of endpoint IN reaches the value set by INMAXP;
3. If the endpoint is in synchronous mode and ISOUD is "1", the read value of IPRDY is always 0 before the next SOF is received.

When you need to get/set this information, you must first use INDEX to select target endpoints 1~5

### 25.2.14 IN Endpoint Control Status Register 2 (INCSR2)

Symbol address B7			B6	B5	B4	B3 B2		B1	B0
INCSR2	12H	AUTOSET	ISO	MODE ENDMA	FCDT -			-	-

**AUTOSET:** Automatically set the IPRDY flag control bit.

0: Disable automatic setting of the IPRDY flag

1: Enable automatic setting of IPRDY (the data loaded into the IN FIFO must reach the value set by INMAXP, otherwise IPRDY flags must be set manually)

**ISO:** Isochronous transfer enabled.

0: Endpoint is configured for bulk/interrupt transfer mode

1: The endpoint is configured for isochronous transport

**MODE:** Endpoint direction selection bit.

0: Select the endpoint direction as OUT

1: Select the endpoint direction as IN

**ENDMA:** DMA control for IN endpoints

0: Disable DMA requests for IN endpoints

1: Enable DMA request for IN endpoint

**FCDT:** Force DATA0/DATA1 data switching setting.

0: Endpoint data toggles only after sending a packet and receiving an ACK.

1: Endpoint data is forced to switch after each packet is sent, regardless of whether an ACK is received.

When you need to get/set this information, you must first use INDEX to select target endpoints 1~5

### 25.2.15 Maximum packet size for OUT endpoints (OUTMAXP)

Symbol address B7			B6	B5	B4 B3 B2			B1	B0
OUTMAXP	13H				OUTMAXP[7:0]				

**OUTMAXP[7:0]:** Set the size of the maximum data packet of the OUT endpoint of the USB (Note: The data packet is in units of 8 bytes. For example, if

Need to set the packet 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 target endpoints 1~5

### 25.2.16 OUT Endpoint Control Status Register 1 (OUTCSR1)

Symbol address B7			B6	B5	B4	B3	B2	B1	B0
OUTCSR1	14H	CLRD STSTL		SDSTL FLUSH DATERR	OVRRUN FIROFUL OPRDY				

**CLRD (Clear Data Toggle):** Reset the OUT data toggle bit.

When the OUT endpoint needs to reset the data toggle bit to "0" due to being reconfigured or STALL, the software needs to send the data to this number.

Write "1" according to the bit.

**STSTL (Sent Stall):** STALL signal transmission completion flag.

When the STALL signal is sent, the hardware will set this bit to "1". This flag must be cleared to '0' by software.

**SDSTL (Send Stall):** STALL signal send request bit.

Software should 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 Beam STALL signal. This bit has no effect on ISO mode.

**FLUSH (FIFO Flush):** Clears the next packet of the FIFO of 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 multiple packets, software must write '1' to FLUSH for each packet. When the FIFO is empty,

Hardware clears the FLUSH bit to '0'.

**OVRRUN (Data Overrun):** Data overflow.

This bit is set by hardware when an incoming 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 lost due to FIFO full since this flag was last cleared

**FIFOFUL (FIFO Full):** The FIFO data full flag of the OUT endpoint.

0: FIFO of OUT endpoint is not full

1: The FIFO of the OUT endpoint is full

**OPRDY (OUT Packet Ready):** OUT packet reception completion flag.

The hardware sets this bit to "1" when a packet is available. Software should clear this bit after unloading each packet from the OUT endpoint FIFO '0'.

When you need to get/set this information, you must first use INDEX to select target endpoints 1~5

## 25.2.17 OUT Endpoint Control Status Register 2 (OUTCSR2)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
OUTCSR2	15H AUTOCLR	ISO	ENDMA DMAMD -		-	-	-

**AUTOCLR:** Automatically clear the OPRDY flag control bit.

0: Disable automatic clearing of the OPRDY flag

1: Enable automatic clearing of OPRDY (the data downloaded from the OUT FIFO must reach the value set by OUTMAXP, otherwise OPRDY flags must be cleared manually)

**ISO:** Isochronous transfer enabled.

0: Endpoint is configured for bulk/interrupt transfer mode

1: The endpoint is configured for isochronous transport

**ENDMA:** DMA control for OUT endpoints

0: Disable DMA requests for OUT endpoints

1: Enable DMA request for OUT endpoint

**DMAMD:** Set the DMA mode of the OUT endpoint

When you need to get/set this information, you must first use INDEX to select target endpoints 1~5

## 25.2.18 OUT Length of USB Endpoint 0 (COUNT0)

Symbol address B7	B6	B5	B4 B3 B2	B1	B0
COUNT0	16H	-	OUTCNT0[6:0]		

**OUTCNT0[6:0]:** OUT byte length of endpoint 0

COUNT0 is dedicated to saving the data length of the last OUT packet received by endpoint 0 (because the endpoint 0 packet can only be up to

64 bytes, so only 7 bits are needed). This length value is only valid when the OPRDY bit of endpoint 0 is "1".

When you need to get this length information, you must first use INDEX to select the target endpoint 0

### 25.2.19 OUT Length of USB Endpoint (OUTCOUNTn)

Symbol address	B7		B6	B5	B4 B3	B2		B1	B0
OUTCOUNT1	16H						OUTCNT[7:0]		
OUTCOUNT2	17H	-	-	-	-	-		OUTCNT[10:8]	

OUTCNT[10:0]: OUT byte length of the endpoint

OUTCOUNT1 and OUTCOUNT2 are combined to form an 11-bit number, which saves the data length of the last OUT data packet.

at endpoints 1~5. This length value is only valid when the OPRDY bits of endpoints 1~5 are "1".

When you need to get this length information, you must first use INDEX to select the target endpoint 1~5

### 25.2.20 USB Endpoint FIFO Data Access Register (FIFOOn)

Symbol address	B7		B6	B5	B4 B3	B2		B1	B0
FIFO0	20H						FIFO0[7:0]		
FIFO1	21H						FIFO1[7:0]		
FIFO2	22H						FIFO2[7:0]		
FIFO3	23H						FIFO3[7:0]		
FIFO4	24H						FIFO4[7:0]		
FIFO5	25H						FIFO5[7:0]		

FIFOOn[7:0]: IN/OUT data indirect access register of each USB endpoint

### 25.2.21 USB Trace Control Register (UTRKCTL)

Symbol address	B7		B6	B5 B4	B3	B2 B1	B0	
UTRKCTL	30H	FTM1	FTM0	INTV[1:0]	ENST5		RES[2:0]	

FTM1: HFOSC Trim Control Bit

0: The hardware uses coarse and fine adjustments to adjust the frequency

1: Hardware adjusts HFOSC using only trim bits

FTM0: Valid when FTM1 is 0

0: UTRK uses all 128 trim levels

1: UTRK prohibits the use of maximum and minimum 12-level fine-tuning

INTV[1:0]: Select UTRK update cycle

INTV[1:0] period	
00	2ms
01	4ms
10	8ms
11	16ms

ENST5: Enable plus 5 and minus 5

0: The upper and lower calibration limits are 10%

1: The upper and lower calibration limits are 20%

RES[2:0]: UTRK auto adjust resolution setting

RES[2:0]	Resolution adjustment value
----------	-----------------------------

000	8	0.067%
001	12	0.100%
010	16	0.133%
<b>011</b>	<small>twenty four</small>	<b>0.200%</b>
100	28	0.233%
101	32	0.267%
110	48	0.4%
111	64	0.5%

## 25.2.22 USB Trace Status Register (UTRKSTS)

Symbol 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]	Status
00	INC 5
01	DEC 5
10	INC 1
11	DEC 1

TST\_UTRK: UTRK test mode control

0: Disable UTRK test mode

1: Enable UTRK test mode

UTRK\_RDY: UTRK calibration complete status bit

## 25.3 Notes on USB Product Development

Each USB product must have its own unique VID&PID combination in order to be correctly recognized by the computer. If two different USB products have the same VID&PID combination, the computer may not recognize the USB product abnormally, so that the USB product cannot be used normally. To avoid this situation, both VIDs and PIDs need to be uniformly planned and assigned through formal channels.

At present, STC has obtained the VID number 13503 (hexadecimal: 34BF) of STC's dedicated USB device through the USB-IF organization.

When customers use STC's USB chips to develop their own USB products, if you have obtained your own VID through other

The corresponding PID can be programmed by itself. If your USB product needs to use the official VID of STC, you must apply to STC for the PID of the product.

Please.

## 25.4 Example Program

### 25.4.1 Example of HID Human-Machine Interface Device

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

typedef unsigned char          BYTE;
typedef unsigned int typedef   WORD;
unsigned long                  DWORD;

#define FADDR              0
#define POWER               1
#define INTRIN1 #define     2
EP5INIF #define EP4INIF  0x20
#define EP3INIF #define    0x10
EP2INIF #define        0x08
EP1INIF #define EP0IF   0x04
#define INTROUT1           0x02
#define EP5OUTIF            0x01
#define EP4OUTIF #define    4
EP3OUTIF #define       0x20
EP2OUTIF #define       0x10
EP1OUTIF #define       0x08
INTRUSB #define SOFIF   0x04
#define RSTIF #define      0x02
RSUIF #define SUSIF    6
#define INTRIN1E #define   0x08
EP5INIE #define EP4INIE 0x04
#define EP3INIE #define   0x02
EP2INIE #define EP1INIE 0x01
#define EP0IE #define      7
INTROUT1E #define      0x20
EP5OUTIE #define      0x10
EP4OUTIE #define      0x08
EP3OUTIE #define      0x04
EP2OUTIE #define      0x02
EP1OUTIE #define      0x01
INTRUSBE #define SOFIE  9
#define RSTIE #define      0x20
RSUIE #define SUSIE    0x10
#define FRAME1 #define     0x08
FRAME2 #define INDEX    0x04
                           0x02
                           11
                           0x08
                           0x04
                           0x02
                           0x01
                           12
                           13
                           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 INCSEL1	17
#define INCLRDT	0x40
#define INSTSTL	0x20
#define INSDSTL	0x10
#define INFUSH	0x08
#define INUNDRUN	0x04
#define INFIFONE	0x02
#define INIPRDY	0x01
#define INCSEL2 #define	18
INAUTOSET #define	0x80
INISO #define INMODEIN	0x40
#define INMODEOUT	0x20
#define INENDDMA #define	0x00
INFCDT #define	0x10
OUTMAXP #define	0x08
OUTCSR1 #define	19
OUTCLRDT #define	20
OUTSTSTL #define	0x80
OUTSDSTL #define	0x40
OUTFLUSH #define	0x20
OUTDATERR #define	0x10
OUTOVERRUN #define	0x08
OUTFIFOFUL #define	0x04
OUTOPRDY #define	0x02
OUTCSR2 #define	0x01
OUTAUTOCLR #define	Twenty one
OUTISO #define	0x80
OUTENDDMA #define	0x40
OUTDMAMD #define	0x20
COUNT0 #define	0x10
OUTCOUNT1 #define	Twenty two
OUTCOUNT2 #define	Twenty three
FIFO0 #define FIFO1	Twenty four
#define FIFO2 #define	32
FIFO3 #define FIFO4	33
#define FIFO5 #define	34
UTRKCTL #define	35
UTRKSTS	36
	37
	48
	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 #define    0x03
SET_ADDRESS #define        0x05
GET_DESCRIPTOR #define     0x06
SET_DESCRIPTOR #define     0x07
GET_CONFIG #define         0x08
SET_CONFIG #define         0x09
GET_INTERFACE #define      0x0A
SET_INTERFACE #define      0x0B
SYNCH_FRAME               0x0C

#define GET_REPORT              0x01
#define GET_IDLE #define       0x02
GET_PROTOCOL #define       0x03
SET_REPORT #define         0x09
SET_IDLE #define           0x0A
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 #define   0x20
VENDOR_REQUEST #define      0x40
REQUEST_MASK               0x60

typedef struct
{
    BYTE bmRequestType;
    BYTE bRequest;
    BYTE wValueL;
    BYTE wValueH;
    BYTE wIndexL;
    BYTE wIndexH;
    BYTE wLengthL;
    BYTE wLengthH;
}SETUP;

typedef struct
{
    BYTE bStage;
    WORD wResidue;
    BYTE *pData;
}EP0STAGE;

void UsbInit();
BYTE ReadReg(BYTE addr);
void WriteReg(BYTE addr, BYTE dat);
BYTE ReadFifo(BYTE fifo, BYTE *pdat);
void WriteFifo(BYTE fifo, BYTE *pdat, BYTE cnt);

char code DEVICEDESC[18];
char code CONFIGDESC[41];
char code HIDREPORTDESC[27];
char code LANGIDDESC[4];
char code MANUFACTDESC[8];
char code PRODUCTDESC[30];

```

```

SETUP Setup;
EP0STAGE Ep0Stage;
BYTE xdata HidFeature[64];
BYTE xdata HidInput[64];
BYTE xdata HidOutput[64];

void main()
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                            //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                              //assign to      CPU The speed of executing the program is set to the fastest

    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 PLL
    DelayXns(100);
    CLKSEL = 0x02;
    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 introu;
    BYTE csr;
    BYTE cnt;
    WORD len;

    intrusb = ReadRegINTRUSB();
    intrin = ReadRegINTRIN1();
    introu = ReadRegINTROUT1();

    if (intrusb & RSTIF)
}

```

```
{  
    WriteReg(INDEX, 1);  
    WriteReg(INCSR1, INCLRDT);  
    WriteReg(INDEX, 1);  
    WriteReg(OUTCSR1, OUTCLRDT);  
    Ep0Stage.bStage = EPIDLE;  
}  
  
if (intr & EP0IF) {  
  
    WriteReg(INDEX, 0);  
    csr = ReadReg(CSR0);  
    if (csr & STSTL) {  
  
        WriteReg(CSR0, csr & ~STSTL);  
        Ep0Stage.bStage = EPIDLE;  
  
    } if (csr & SUEND)  
    {  
        WriteReg(CSR0, csr | SSUEND);  
    }  
  
    switch (Ep0Stage.bStage)  
    { case EPIDLE:  
  
        if (csr & OPRDY)  
        {  
            Ep0Stage.bStage = EPSTATUS;  
            ReadFifo(FIFO0, (BYTE *)&Setup);  
            ((BYTE *)&Ep0Stage.wResidue)[0] = Setup.wLengthH;  
            ((BYTE *)&Ep0Stage.wResidue)[1] = Setup.wLengthL;  
            switch (Setup.bmRequestType & REQUEST_MASK)  
            { case STANDARD_REQUEST: switch (Setup.bRequest)  
            { case SET_ADDRESS: WriteReg(FADDR,  
                Setup.wValueL); break; case SET_CONFIG:  
                WriteReg(INDEX, 1); WriteReg(INCSR2,  
                INMODEIN); WriteReg(INMAXP, 8);  
                WriteReg(INDEX, 1); WriteReg(INCSR2,  
                INMODEOUT); WriteReg(OUTMAXP, 8);  
                WriteReg(INDEX, 0); break; case  
                GET_DESCRIPTOR: Ep0Stage.bStage =  
                EPDATAIN; switch (Setup.wValueH) { case  
                DESC_DEVICE: Ep0Stage.pData =  
                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 = HidFeature;
    Ep0Stage.bStage = EPDATAIN;
    break; case SET_REPORT:
    Ep0Stage.pData = HidFeature;
    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);
```

```

        }
    }
}

if (intrin & EP1INIF)
{
    WriteReg(INDEX, 1);
    csr = ReadReg(INCSR1);
    if (csr & INSTSTL)
    {
        WriteReg(INCSR1, INCLRDT);
    }
    if (csr & INUNDRUN)
    {
        WriteReg(INCSR1, 0);
    }
}

if (intout & 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, 0x00,                                // bDeviceClass(0);
    0x00, 0x40,                                // bDeviceSubClass0);
    0xbff,0x34,                               // bDeviceProtocol(0);
    0x80,0x43,                               // bMaxPacketSize0(64);
    0x00,0x01,                                // idVendor(STCUSB:34bf);
    0x01, 0x02,                                // idProduct(4380);
    0x00, 0x01,                                // bcdDevice(1.00);
    0x00,                                     // iManufacturer(1);
    0x00,                                     // iProduct(2);
    0x00,                                     // iSerialNumber(0);
    0x00,                                     // bNumConfigurations(1);
};

char code CONFIGDESC[41] =

```

```

{
    0x09,                                // bLength(9);
    0x02,                                // bDescriptorType(Configuration);
    0x29, 0x00,                            // wTotalLength(41);
    0x01, 0x01,                            // bNumInterfaces(1);
    0x00, 0x80,                            // bConfigurationValue(1);
    0x32,                                // iConfiguration(0);
                                         // bmAttributes(BUSPower);
                                         // 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,                                // ilInterface(0);

    0x09,                                // bLength(9);
    0x21,                                // bDescriptorType(HID);
    0x01, 0x01,                            // bcdHID(1.01);
    0x00, 0x01,                            // bCountryCode(0);
    0x22,                                // bNumDescriptors(1);
    0x1b, 0x00,                            // bDescriptorType(HID Report);
                                         // wDescriptorLength(27);

    0x07,                                // bLength(7);
    0x05,                                // bDescriptorType(Endpoint);
    0x81,                                // bEndpointAddress(EndPoint1 as IN);
    0x03,                                // bmAttributes(Interrupt);
    0x40, 0x00,                            // wMaxPacketSize(64);
    0x01,                                // blInterval(10ms);

    0x07,                                // bLength(7);
    0x05,                                // bDescriptorType(Endpoint);
    0x01,                                // bEndpointAddress(EndPoint1 as OUT);
    0x03,                                // bmAttributes(Interrupt);
    0x40, 0x00,                            // wMaxPacketSize(64);
    0x01,                                // blInterval(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, 0x01,                           // INPUT(Data, Variable);
    0x9, 0x09,                            // USAGE(Consumer Control);
    0x02, 0xc0,                            // OUTPUT(Data, Variable);
                                         // 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,  
    't', 0,  
    'c', 0,  
    'e', 0,
```

```
};
```

## 25.4.2 HID (Human Interface Device) protocol example

After downloading the code to the experimental box, you can use the HID assistant in the latest STC-ISP download software to detect and test the detailed code, please refer to the "69-HID (Human Interface Device) protocol in the "STC32G Experiment Box Demonstration Program" package on the official website. example"

## 25.4.3 CDC (Communication Device Class) protocol example

The operating system below WIN10 needs to install the driver in the sys directory. After the WIN10 and WIN11 installation-free driver downloads the code to the experimental box, the device that can be recognized as a USB-to-serial port on the PC side can use the DB9 interface on the experimental box to connect with other devices. Serial port for communication The data bit of the serial port only supports 8 bits, and the stop bit only supports 1 check bit. It can support: no check, odd check, even check, 1 check and 0 check. , and supports custom baud rate, please refer to the "70-CDC (Communication Device Class) Protocol Example" in the "STC32G Experiment Box Demonstration Program" package on the official website.

## 25.4.4 Example of USB keyboard based on HID protocol

After downloading the code to the experiment box, the basic functions of the USB keyboard can be realized. LED17 in the marquee is the NumLock light, LED16 is the CapsLock light, and LED15 is the ScrollLock light. KEY0~KEY7 in the matrix keys are 1~8 in the keyboard respectively. Please refer to "71 - Example of USB Keyboard Based on HID Protocol" in the "STC32G Experiment Box Demonstration Program" package on the official website

## 25.4.5 Example of USB Mouse Based on HID Protocol

After downloading the code to the experimental box, the basic functions of the USB mouse can be realized. In the matrix keys, KEY0 is the left mouse button, KEY1 is the middle mouse button, and KEY2 is the right mouse button. In the matrix keys, KEY4 is left shift, KEY5 is right shift, and KEY6 is To move up, KEY7 to move down. For detailed code, please refer to "72 - USB Mouse Example Based on HID Protocol" in the "STC32G Experiment Box Demonstration Program" package on the official website

## 25.4.6 Example based on WINUSB protocol

Operating systems below WIN10 need to install the driver in the sys directory. For WIN10 and WIN11 installation-free drivers, you can use "STC\_WINUSB.exe" in the exe directory for testing. For the detailed code, please refer to the "STC32G Experiment Box Demonstration Program" package on the official website. "73 - Example based on WINUSB protocol"

## 25.4.7 MSC (Mass Storage Class) protocol example

After downloading the code to the experimental box, it can be recognized as a 512K U disk on the PC side

The U disk storage is the external 512K serial FLASH on the experimental box

On the experimental box without external FLASH, the EEPROM inside STC32G12K128 can also be used as memory

Just change the memory type to MEMTYPE\_INT in the config.h file

Then when the ISP downloads, set the EEPROM size to 64K, you can achieve a 64K capacity U disk

For the detailed code, please refer to the "74-MSC (Mass Storage Class) Protocol Example" in the "STC32G Experiment Box Demonstration Program" package on the official website

STCMCU

# 26 RTC real time clock

Some STC32G series microcontrollers integrate a real-time clock control circuit, which mainly has the following characteristics:

- ÿ Low power consumption: the working current of the RTC module is as low as 10uA
- ÿ Long time span: support from 2000 to 2099, and automatically determine leap years
- ÿ Alarm clock: support a set of alarm clock settings
- ÿ Supports multiple interruptions: alarm interruption, day interruption, hour interruption, minute interruption, second interruption, 1/2 second interruption, 1/8 second interruption, 1/32 second interruption
- ÿ Support power-down wake-up

## 26.1 RTC related registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
RTCCR	RTC Control Register	7EFE60H	-	-	-	-	-	-	-	-	RUNRTC xxxx,xxx0
RTCCFG	RTC Configuration Register	7EFE61H	-	-	-	-	-	-	-	-	RTCCKS SET RTC xxxx,xx00
RTCIN	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	SEC1F	SEC2IF	SEC8IF	SEC32IF	0000,0000
ALA HOUR	RTC alarm hour value	7EFE64H	-	-	-	-	-	-	-	-	xxxx0,0000
ALAMIN	RTC alarm minute value	7EFE65H	-	-	-	-	-	-	-	-	xx00,0000
ALASEC	RTC alarm seconds value	7EFE66H	-	-	-	-	-	-	-	-	xx00,0000
1/128	second value of ALASSEC RTC alarm 7EFE67H		-	-	-	-	-	-	-	-	x000,0000
INIYEAR	RTC year initialization	7EFE68H	-	-	-	-	-	-	-	-	x000,0000
INIMONTH	RTC monthly initialization	7EFE69H	-	-	-	-	-	-	-	-	xxxx,0000
INIDAY	RTC day initialization	7EFE6AH	-	-	-	-	-	-	-	-	xxx0,0000
INIHOUR	RTC hour initialization	7EFE6BH	-	-	-	-	-	-	-	-	xxx0,0000
INIMIN	RTC minute initialization	7EFE6CH	-	-	-	-	-	-	-	-	xx00,0000
INISEC	RTC seconds initialization	7EFE6DH	-	-	-	-	-	-	-	-	xx00,0000
INISSEC	1/128 seconds initialization	7EFE6EH	-	-	-	-	-	-	-	-	x000,0000
YEAR	Year count value for YEAR RTC	7EFE70H	-	-	-	-	-	-	-	-	x000,0000
MONTH	Month count value of MONTH RTC	7EFE71H	-	-	-	-	-	-	-	-	xxxx,0000
DAY	Daily count value of RTC	7EFE72H	-	-	-	-	-	-	-	-	xxx0,0000
HOUR	Hour Count Value	7EFE73H	-	-	-	-	-	-	-	-	xx00,0000
MIN	Minute count value for RTC	7EFE74H	-	-	-	-	-	-	-	-	xx00,0000
SEC	Second count value of RTC	7EFE75H	-	-	-	-	-	-	-	-	xx00,0000
SSEC	1/128 second count value of RTC	7EFE76H	-	-	-	-	-	-	-	-	x000,0000

### 26.1.1 RTC Control Register (RTCCR)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
RTCCR	7EFE60H	-	-	-	-	-	-	RUNRTC

RUNRTC: RTC module control bit

0: Turn off RTC, RTC stops counting

1: Enable RTC and start RTC counting

### 26.1.2 RTC CONFIGURATION REGISTER (RTCCFG)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
RTCCFG	7EFE61H	-	-	-	-	-	RTCCKS	SETRTC

RTCCKS: RTC clock source selection

0: Select the external 32.768KHz [clock source](#) (the software needs to start the external 32K crystal oscillator first)

1: Select the internal 32K [clock source](#) (need to start the internal 32K oscillator by software)

SETRTC: Set the initial value of RTC

write 0: meaningless

Write 1: trigger RTC register initialization. When SETRTC is set to 1, the hardware will automatically set the registers INIYEAR, INIMONTH,

The values in INIDAY, INIHOUR, INIMIN, INISEC, INISSEC are copied to registers YEAR, MONTH, DAY,

HOUR, MIN, SEC, SSEC. Hardware will automatically clear the SETRTC bit after initialization is complete.

Read 0: Set the RTC related time register completed

Read 1: Hardware is setting up the RTC, not yet done

[Note: Waiting for the initialization to complete, it needs to be judged after "RTC enable". Setting the RTC time requires 1 cycle time of 32768Hz,](#)

[About 30.5us. Due to synchronization, the actual waiting time is 0~30.5us. If you sleep without waiting for the setting to complete, the RTC will](#)

[It will stop counting because the setting is not completed, and continue to complete the setting and continue counting after waking up. If the setting is to use RTC at this time](#)

[If the interrupt is used to wake up, there will be a situation where the MCU cannot be woken up.](#)

### 26.1.3 RTC Interrupt Enable Register (RTCIEN)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0	
RTCIEN	7EFE62H	EALAI	EDAYI	EHOURI	EMINI	ESECI	ESEC2I	ESEC8I	ESEC32I

EALAI: Alarm Interrupt Bit

0: Disable the alarm interrupt

1: Enable alarm interrupt

EDAYI: One-day (24-hour) interrupt enable bit

0: Turn off the one-day interrupt

1: Enable one-day interrupt

EHOURI: One Hour (60 Minutes) Interrupt Enable Bit

0: Turn off the hour interrupt

1: Enable hour interrupt

EMINI: One minute (60 seconds) interrupt enable bit

0: Turn off the hour interrupt

1: Enable hour interrupt

ESECI: One-Second Interrupt Enable Bit

0: Disable seconds 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)

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0	
RTCIF	7EFE63H	ALAIF	DAYIF	HOURIF	MINIF		SECIF	SEC2IF	SEC8IF	SEC32IF

ALAIF: Alarm interrupt request bit. It needs to be cleared by software, and writing 1 by software is invalid.

DAYIF: One-day (24-hour) 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. It needs to be cleared by software, and writing 1 by software is invalid.

MINIF: One minute (60 seconds) interrupt request bit. It needs to be cleared by software, and writing 1 by software is invalid.

SECIF: One second interrupt request bit. It needs to be cleared by software, and writing 1 by software is invalid.

SEC2IF: 1/2 second interrupt request bit. It needs to be cleared by software, and writing 1 by software is invalid.

SEC8IF: 1/8 second interrupt request bit. It needs to be cleared by software, and writing 1 by software is invalid.

SEC32IF: 1/32 second interrupt request bit. It needs to be cleared by software, and writing 1 by software is invalid.

## 26.1.5 RTC Alarm Setting Register

Symbol address	B7		B6	B5	B4	B3	B2	B1	B0
ALA HOUR 7EFE64H	-	-	-						
ALAMIN 7EFE65H	-	-							
ALASEC 7EFE66H	-	-							
ALASSEC 7EFE67H	-								

ALA HOUR: Sets the hourly value of the daily alarm.

Note: The set value is not a **BCD code**, but a **HEX code**. For example, if you need to set the hour value from 20 to ALAHOUR, you need to use the following

Set the code below

```
MOV      DPTR, #ALAHOUR
MOV      A, #14H
MOVX @DPTR, A
```

ALAMIN: Set the minute value of the daily alarm. The numeric encoding is the same as ALAHOUR.

ALASEC: Set the seconds value of the daily alarm. The numeric encoding is the same as ALAHOUR.

ALASSEC: Sets the value of 1/128th of a second for the daily alarm. The numeric encoding is the same as ALAHOUR.

## 26.1.6 RTC real-time clock initial value setting register

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
INIYEAR	7EFE68H	-							
INIMONTH	7EFE69H								
INIDAY	7EFE6AH								
INI HOUR	7EFE6BH	-	-	-					
INIMIN	7EFE6CH	-	-						
INISEC	7EFE6DH	-	-						
INISSEC	7EFE6EH	-							

INIYEAR: Set the year value of the current real time. The valid value range is 00~99. Corresponding to 2000 to 2099

Note: The set value is not BCD code, but HEX code. For example, if you need to set 20 to INIYEAR, you need to use the following code to enter line settings

```
MOV      DPTR, #INIYEAR
MOV      A, #14H
MOVX @DPTR,A
```

INIMONTH: Set the monthly value of the current real-time time. The valid value range is 1~12. Numeric encoding is the same as INIYEAR.

INIDAY: Set the day value of the current real-time time. The valid value range is 1~31. Numeric encoding is the same as INIYEAR.

INI HOUR: Set the hour value of the current real time. The valid value range is 00~23. Numeric encoding is the same as INIYEAR.

INIMIN: Set the minute value of the current real time. The valid value range is 00~59. Numeric encoding is the same as INIYEAR.

INISEC: Sets the second value of the current real time. The valid value range is 00~59. Numeric encoding is the same as INIYEAR.

INISSEC: Sets the 1/128 second value of the current real time. The valid value range is 00~127. Numeric encoding is the same as INIYEAR.

After the user has set the initial value register above, the user also needs to write 1 to the SETRTC bit (RTCCFG).

Send the hardware to load the initial value into the RTC real-time counter

Also note: the hardware will not check the validity of the initialization data, and the user must ensure that the initial value is set.

The validity of the certification data cannot exceed its valid range.

## 26.1.7 RTC Real Time Clock Count Register

Symbol	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 a BCD code, but a HEX code

MONTH: The monthly value of the current real-time time. Numeric encoding is the same as YEAR.

DAY: The day value of the current real-time time. Numeric encoding is the same as YEAR.

HOUR: The hour value of the current real time. Numeric encoding is the same as YEAR.

MIN: The minute value of the current real-time time. Numeric encoding is the same as YEAR.

SEC: The second value of the current real time. Numeric encoding is the same as YEAR.

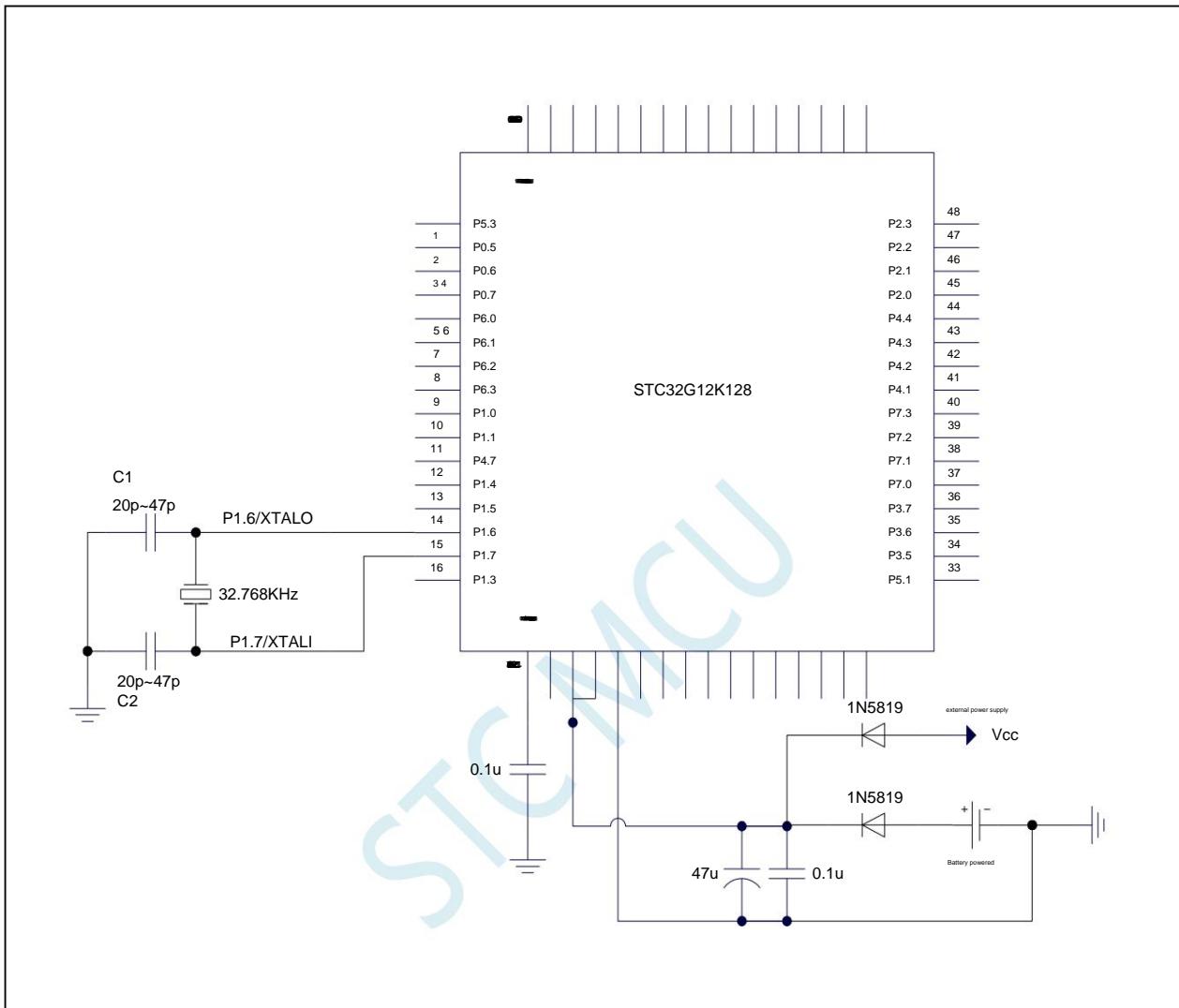
SSEC: 1/128 second value of the current real time. Numeric encoding is the same as YEAR.

**Note:** YEAR, MONTH, DAY, HOUR, MIN, SEC and SSEC are read-only registers.

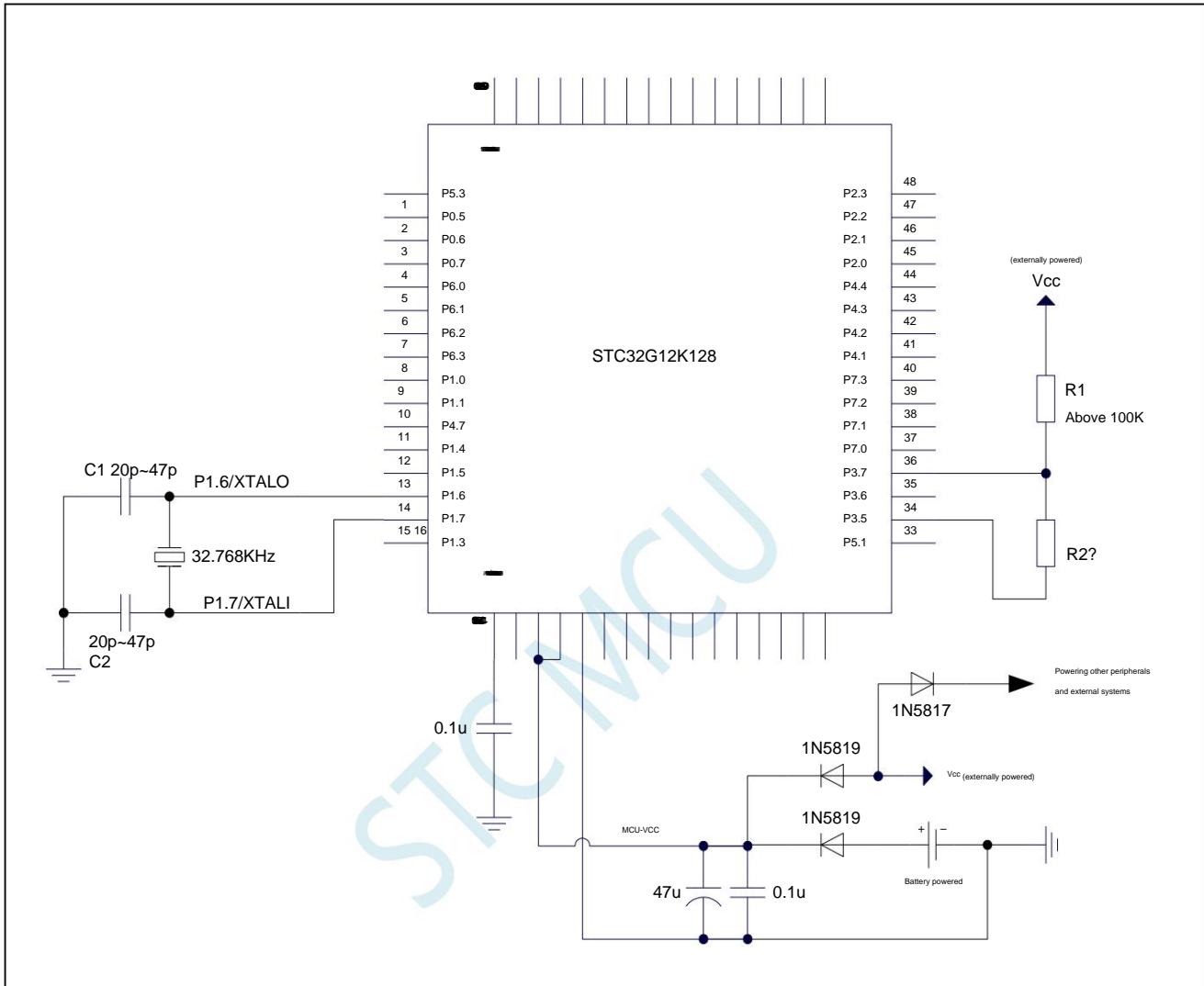
To perform a write operation, you must pass the registers **INIYEAR**, **INIMONTIH**, **INIDAT**, **INIHOU**, **INIMIN**, **INISEC**, **INISSEC** and **SETRTC** to achieve.

STCMCU

## 26.2 RTC reference circuit diagram (without VBAT pin)



## 26.3 RTC actual combat circuit diagram



## 26.4 Example Program

### 26.4.1 Serial port printing RTC clock example

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                                     //For header files, see Download Software
#include "intrins.h"
#include "stdio.h"

#define MAIN_Fosc      22118400L
#define Baudrate       115200L
#define TM             (65536 -(MAIN_Fosc/Baudrate/4))

bit      B1S_Flag;

void RTC_config(void);

void UartInit(void)
{
    SCON = (SCON & 0x3f) / 0x40;
    TL2 = TM;
    TH2 = TM>>8;
    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 whether the second is interrupted
    {
        RTCIF &= ~0x08;                            //clear interrupt flag
        B1S_Flag = 1;
    }
}

void main(void)
{
    EAXFR = 1;                                     Enable XFR
    CKCON = 0x00;                                  access to set external data bus speed to fastest
    WTST = 0x00;                                   //Set the program code to wait for parameters,
```

```

//assign as 0 can be CPU The speed of executing the program is set to the fastest

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 send a string

while (1)
{
    if(B1S_Flag)
    {
        B1S_Flag = 0;

        printf("Year=%d ", YEAR);
        printf("Month=%d ", MONTH);
        printf("Day=%d ", DAY);
        printf("Hour=%d ", HOUR);
        printf("Minute=%d ", MIN);
        printf("Second=%d ", SEC);
        printf("\r\n");
    }
}

void RTC_config(void)
{
    //select inside 32K
    IRC32KCR = 0x80;
    while (!(IRC32KCR & 0x01));
    RTCCFG |= 0x02;

    /// select external 32K
    X32KCR = 0xc0; //
    while (!(X32KCR & 0x01)); //
    RTCCFG &= ~0x02;

    INIYEAR = 21;
    INIMONTH = 12;
    INIDAY = 31;
    INIHOUR = 23;
    INIMIN = 59;
    INISEC = 50;
    INISSEC = 0;
    RTCCFG |= 0x01;
    while(RTCCFG & 0x01);

    //
    RTCIF = 0;
    RTCIEN = 0x08;
}

// Start the internal 32K
// Wait for the clock to stabilize
//select internal 32K as RTC clock source

// Start the external 32K
// Wait for the clock to stabilize
//select external 32K as RTC clock source

// Y:2021
// M:12
// D:31
// H:23
// M:59
// S:50
// S/128:0
// Trigger RTC register initialization
// Waiting for the completion of the initialization needs to determine the cycle time required to .
// set the time after enabling that possibly can not immediately start the counting
// 30.5us. , 0~30.5us.
// RTC
// Continue to complete the setting and continue counting after the completion of the stop counting wake-up .

// clear interrupt flag
// Enable second's interrupt

```

```
RTCCR = 0x01;  
}  


---


```

assembly code

*israsm*  
Save the following code as an ASM format file and load it into the project together, for example:

```
CSEG AT 0123H  
JMP      006BH  
END
```

---

## 27 LCM interface (8/16-bit color screen module I8080/M6800

### interface)

Some microcontrollers of the STC32G series integrate an LCM interface controller, which can be used to drive the current popular LCD display mode. piece. Can drive I8080 interface and M6800 interface color screen, support 8-bit and 16-bit data width

#### 27.1 LCM interface function pin switching

Symbol address	B7	B6	B5	B4	B3	B2	B1	B0
LCMIFCFG 7FFE50H LCMIFIE		-	LCMIFIP[1:0]	LCMIFDPS[1:0]		D16_D8 M68_I80		
LCMIFCFG2 7FFE51H		LCMIFCPS[1:0]		SETUPT[2:0]		HOLDT[1:0]		

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

LCMIFCPS[1:0] RS		Read signal RD of I8080	Write signal WR of I8080
		Enable signal E of M6800	Read and 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 bit LCM interface data pin selection bit

LCMIFDPS[1:0] D16_D8 00	Data bytes DAT[7:0]	
	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 bit LCM interface data pin selection bit

LCMIFDPS [1:0] D16_D8 Data high byte DAT[15:8]	Data low byte DAT[7:0]	
00	1	P2[7:0]
01	1	P6[7:0]
10	1	P2[7:0]
11	1	P6[7:0]

#### 27.2 LCM -related registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
LCMIFCFG LCM Interface Configuration Register	7FFE50H LCMIFIE		-	LCMIFIP[1:0]	LCMIFDPS[1:0] D16_D8 M68_I80	0x00,0000					
LCMIFCFG2 LCM Interface Configuration Register 2	7FFE51H	-	LCMIFCPS[1:0]	SETUPT[2:0]		HOLDT[1:0] x000,0000					
LCMIFCR LCM Interface Control Register	7FFE52H ENLCMIF		-	-	-	CMD[2:0]	0xxx,x000				
LCMIFSTA LCM Interface Status Register	7FFE53H		-	-	-	-	-	-	-	-	- LCMIFIF xxxx,xx00

LCMIDDATL	LCM interface low byte data 7EFES4H		LCMIFSAT[7:0]	0000,0000
LCMIDDATH	LCM interface high byte data 7EFEB55H		LCMIDAT [15:8]	0000,0000

### 27.2.1 LCM INTERFACE CONFIGURATION REGISTER (LCMIFCFG)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LCMIFCFG	7EFFE50H	LCMIFIE	-	LCMIFIP[1:0]	LCMIFDPS[1:0]	D16_D8	M68_I80		

LCMIFIE: LCM Interface Interrupt Enable Control Bit

0: Disable LCM interface interrupt

1: Enable LCM interface interrupt

LCMIFIP[1:0]: LCM interface interrupt priority control bits

LCMIFIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

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

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)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LCMIFCFG2	7EFFE51H	-	LCMIFCPS[1:0]	SETUPT[2:0]			HOLDT[1:0]		

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

LCMIFCPS[1:0] RS		Read signal RD of I8080 Enable signal E of M6800	Write signal WR of I8080 Read and 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 setup time control bit for LCM interface communication (see the timing diagram in the following chapters for details)

HOLDT[1:0]: Data hold time control bit for LCM interface communication (see the timing diagram in the following chapters for details)

### 27.2.3 LCM Interface Control Register (LCMIFCR)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
LCMIFCR 7EFFE52H	ENLCMIF	-	-	-	-	CMD[2:0]		

ELCMIF: LCM Interface Enable Control Bit

0: Disable LCM interface function

1: Enable LCM interface function

CMD[2:0]: LCM interface trigger command

CMD[2:0] trigger command
100 write commands
101 Write data
110 Read command/status
111 Read data

### 27.2.4 LCM Interface Status Register (LCMIFSTA)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
LCMIFSTA 7EFFE53H		-	-	-	-	-	-	LCMIFIF

LCMIFIF: LCM interface interrupt request flag, needs to be cleared by software

### 27.2.5 LCM Interface Data Registers (LCMIFDATL, LCMIFDATH)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
LCMIFDATL 7EFFE54H				LCMIFSAT[7:0]				
LCMIFDATH 7EFFE55H					LCMIDAT [15:8]			

LCMFDAT: 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 into 16-bit data

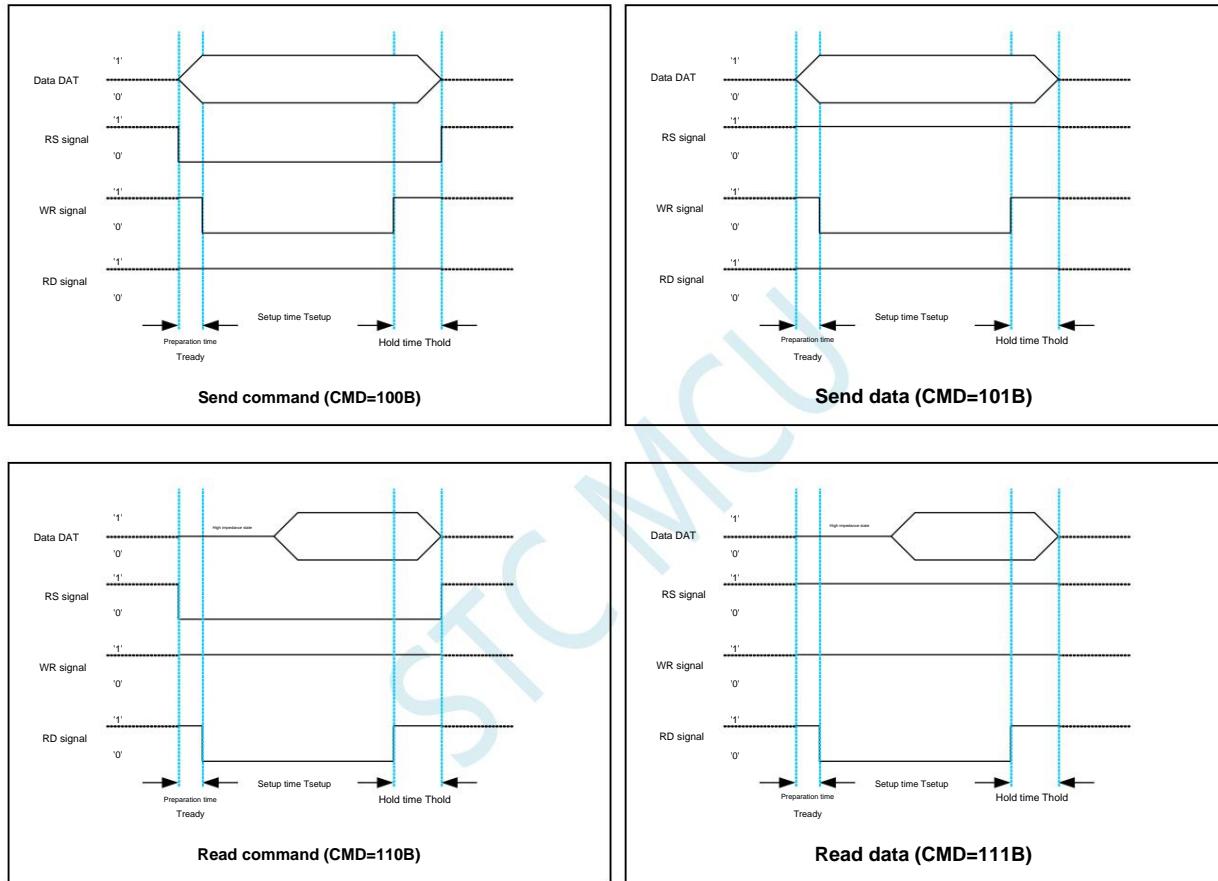
## 27.3 I8080/M6800 mode LCM interface timing diagram

Note: Tready = 1 system clock

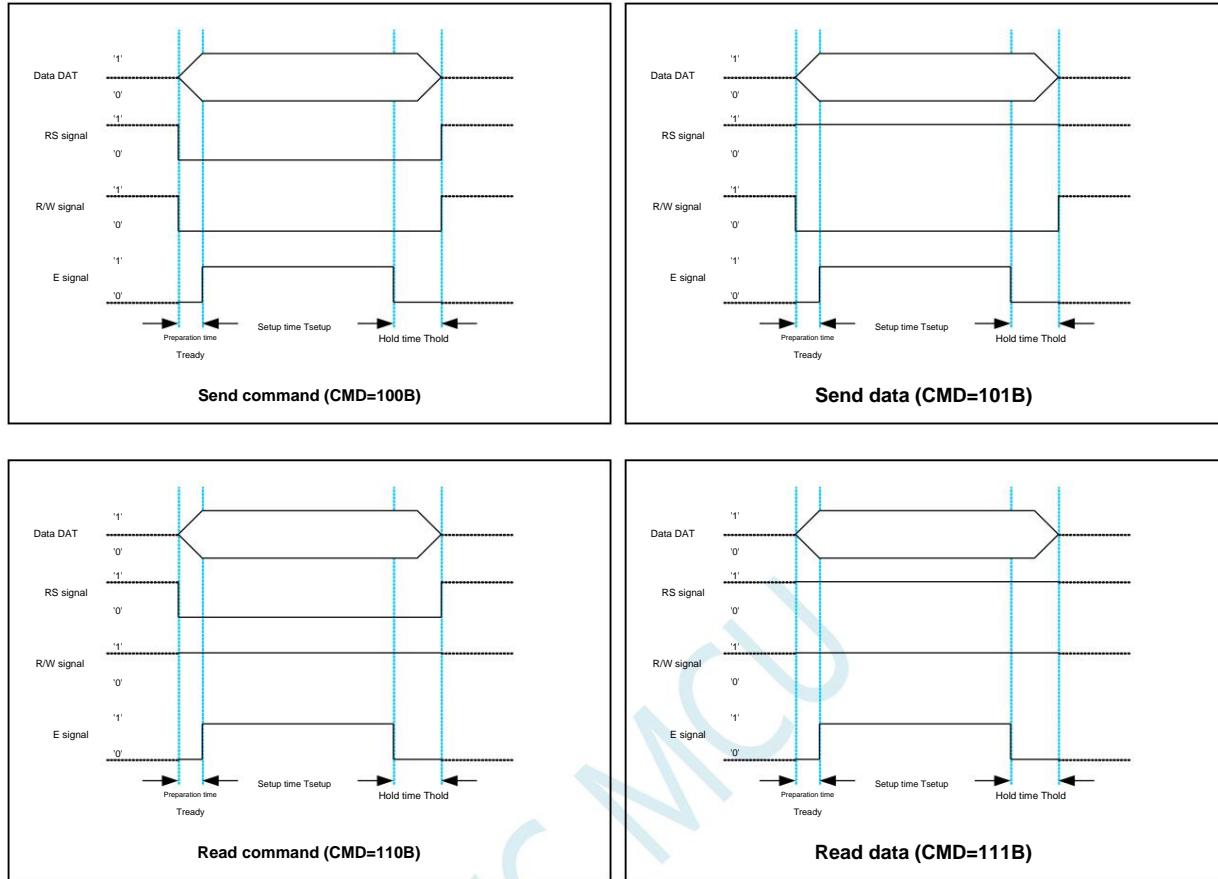
Tsetup = (SETUPT + 1) system clocks

Thold = (HOLDT+1) system clocks

### 27.3.1 I8080 Mode



## 27.3.2 M6800 Mode



# 28 DMA (Bulk Data Transfer)

Some microcontrollers of the STC32G series support the function of bulk data storage, that is, traditional DMA.

The following DMA operations are supported:

- ÿ M2M\_DMA: read and write data from XRAM memory to XRAM memory
- ÿ ADC\_DMA: Automatically scan the enabled ADC channels and automatically store the converted ADC data into XRAM
- ÿ SPI\_DMA: Automatically exchange data between XRAM data and SPI peripherals
- ÿ UR1T\_DMA: automatically send the data in XRAM through serial port 1
- ÿ UR1R\_DMA: automatically store the data received by serial port 1 into XRAM
- ÿ UR2T\_DMA: automatically send the data in XRAM through serial port 2
- ÿ UR2R\_DMA: automatically store the data received by serial port 2 into XRAM
- ÿ UR3T\_DMA: automatically send the data in XRAM through serial port 3
- ÿ UR3R\_DMA: automatically store the data received by serial port 3 into XRAM
- ÿ UR4T\_DMA: automatically send the data in XRAM through serial port 4
- ÿ UR4R\_DMA: automatically store the data received by serial port 4 into XRAM
- ÿ LCM\_DMA: Automatically exchange data between the data in XRAM and the LCM device
- ÿ I2CT\_DMA: automatically send the data in XRAM through the I2C interface
- ÿ I2CR\_DMA: automatically store the data received by I2C into XRAM
- ÿ I2ST\_DMA: automatically send the data in XRAM through I2S
- ÿ I2SR\_DMA: automatically store the data received by I2S into XRAM

The maximum data size for each DMA data transfer is 65536 bytes.

4-level access priority can be set for each DMA read and write operation to XRAM, and the hardware automatically arbitrates access to the XRAM bus.

Does not affect access to the CPU's XRAM. Under the same priority, the access order of different DMAs to XRAM is as follows: 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

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMA_M2M_CFG	M2M_DMA configuration register	7EFA00H	M2MIE	-	- TXACO RXACO	M2MIP[1:0]	-	-	-	-	M2MPTY[1:0] 0x00,0000
DMA_M2M_CR	M2M_DMA Control Register	7EFA01H	ENM2M TRIG	-	-	-	-	-	-	-	00xx,xxxx
DMA_M2M_STA	M2M_DMA Status Register	7EFA02H	-	-	-	-	-	-	-	-	M2MIF xx0x,xxx0
DMA_M2M_AMT	M2M_DMA transfer total bytes	7EFA03H	-	-	-	-	-	-	-	-	0000,0000
DMA_M2M_AMTH	M2M_DMA transfer total bytes	7EFA04H	-	-	-	-	-	-	-	-	0000,0000
DMA_M2M_DONE	M2M_DMA transfer completed bytes	7EFA04H	-	-	-	-	-	-	-	-	0000,0000
DMA_M2M_DONEH	M2M_DMA transfer completed bytes	7EFA05H	-	-	-	-	-	-	-	-	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,x000
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	C7	CH6	CH5 CH4 CH3		CH2	CH1	CH0	0000	0,0001
DMA_ADC_CHSW1	ADC_DMA channel enable	7EFA1BH	C15 CH14 C13 CH12 C11 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,x000
DMA_SPI_STA	SPI_DMA Status Register	7EFA22H	-	-	-	-	- TXO/VW RXLOSS	SPIIF	xxxx,x000		
DMA_SPI_AMT	SPI_DMA transfer total bytes	7EFA23H									0000,0000
DMA_SPI_AMTH	SPI_DMA transfer total bytes	7EFA84H									0000,0000
DMA_SPI_DONE	SPI_DMA transfer completed byte number	7EFA24H									0000,0000
DMA_SPI_DONEH	SPI_DMA transfer completed byte number	7EFA85H									0000,0000
DMA_SPI_TXAH	SPI_DMA transmit high address	7EFA25H									0000,0000
DMA_SPI_TXAL	SPI_DMA transmit 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 Register	7EFA30H	UR1TIE	-	-	-	UR1TIP[1:0]	UR1TPTY[1:0] 0xxx,0000			
DMA_UR1T_CR	UR1T_DMA Control Register	7EFA31H	ENUR1T TRIG		-	-	-	-	-	-	00xx,xxxx
DMA_UR1T_STA	UR1T_DMA Status Register	7EFA32H		-	-	-	- TXO/VW -		UR1TIF	xxxx,x0x0	
DMA_UR1T_AMT	UR1T_DMA transfer total bytes	7EFA33H									0000,0000
DMA_UR1T_AMTH	UR1T_DMA transfer total bytes	7EFA88H									0000,0000
DMA_UR1T_DONE	UR1T_DMA transfer completed bytes	7EFA34H									0000,0000
DMA_UR1T_DONEH	UR1T_DMA transfer completed bytes	7EFA89H									0000,0000
DMA_UR1T_TXAH	UR1T_DMA send high address	7EFA35H									0000,0000
DMA_UR1T_TXAL	UR1T_DMA transmit low address	7EFA36H									0000,0000
DMA_UR1R_CFG	UR1R_DMA Configuration Register	7EFA38H	UR1RIE	-	-	-	UR1RIP[1:0]	UR1RPTY[1:0] 0xxx,0000			
DMA_UR1R_CR	UR1R_DMA Control Register	7EFA39H	ENUR1R -		TRIG	-	-	-	-	-	CLRFIFO 000x,x000
DMA_UR1R_STA	UR1R_DMA Status Register	7EFA3AH		-	-	-	-	- RXLOSS UR1RI	xxxx,x000		
DMA_UR1R_AMT	UR1R_DMA transfer total bytes	7EFA3BH									0000,0000
DMA_UR1R_AMTH	UR1R_DMA transfer total bytes	7EFA8AH									0000,0000
DMA_UR1R_DONE	UR1R_DMA transfer completed byte number	7EFA3CH									0000,0000
DMA_UR1R_DONEH	UR1R_DMA transfer completed byte number	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 Register	7EFA40H	UR2TIE	-	-	-	UR2TIP[1:0]	UR2TPTY[1:0] 0xxx,0000			
DMA_UR2T_CR	UR2T_DMA Control Register	7EFA41H	ENUR2T TRIG		-	-	-	-	-	-	00xx,xxxx
DMA_UR2T_STA	UR2T_DMA Status Register	7EFA42H		-	-	-	- TXO/VW -		UR2TIF	xxxx,x0x0	
DMA_UR2T_AMT	UR2T_DMA transfer total bytes	7EFA43H									0000,0000
DMA_UR2T_AMTH	UR2T_DMA transfer total bytes	7EFA8CH									0000,0000

DMA_UR2T_DONE	UR2T_DMA transfer completed bytes 7EFA44H									0000,0000
DMA_UR2T_DONEH	UR2T_DMA transfer completed byte number 7EFA8DH									0000,0000
DMA_UR2T_TXAH	UR2T_DMA send high address 7EFA45H									0000,0000
DMA_UR2T_RXAL	UR2T_DMA transmit low address 7EFA46H									0000,0000
DMA_UR2R_CFG	UR2R_DMA Configuration Register 7EFA48H UR2RIE	-	-	-	UR2RIP[1:0]	UR2RPY[1:0] 0xxx,0000				
DMA_UR2R_CR	UR2R_DMA Control Register 7EFA49H ENUR2R -		TRIG	-	-	-	-	-	CLRFIFO 0x0,xxx0	
DMA_UR2R_STA	UR2R_DMA Status Register 7EFA4AH	-	-	-	-	-	-	RXLOSS UR2RIF	xxxx,xx00	
DMA_UR2R_AMT	UR2R_DMA transfer total bytes 7EFA4BH									0000,0000
DMA_UR2R_AMTH	UR2R_DMA transfer total bytes 7EFA8EH									0000,0000
DMA_UR2R_DONE	UR2R_DMA transfer completed bytes 7EFA4CH									0000,0000
DMA_UR2R_DONEH	UR2R_DMA transfer completed 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 Register 7EFA50H UR3TIE	-	-	-	UR3TIP[1:0]	UR3TPY[1:0] 0xxx,0000				
DMA_UR3T_CR	UR3T_DMA Control Register 7EFA51H ENUR3T TRIG		-	-	-	-	-	-	-	00xx,xxxx
DMA_UR3T_STA	UR3T_DMA Status Register 7EFA52H	-	-	-	-	-	TXO/VW -		UR3TIF	xxxx,x0x0
DMA_UR3T_AMT	UR3T_DMA transfer total bytes 7EFA53H									0000,0000
DMA_UR3T_AMTH	UR3T_DMA transfer total bytes 7EFA90H									0000,0000
DMA_UR3T_DONE	UR3T_DMA transfer completed bytes 7EFA54H									0000,0000
DMA_UR3T_DONEH	UR3T_DMA transfer completed bytes 7EFA91H									0000,0000
DMA_UR3T_TXAH	UR3T_DMA send high address 7EFA55H									0000,0000
DMA_UR3T_TXAL	UR3T_DMA transmit low address 7EFA56H									0000,0000
DMA_UR3R_CFG	UR3R_DMA Configuration Register 7EFA58H UR3RIE	-	-	-	UR3RIP[1:0]	UR3RPY[1:0] 0xxx,0000				
DMA_UR3R_CR	UR3R_DMA Control Register 7EFA59H ENUR3R -		TRIG	-	-	-	-	-	CLRFIFO 0x0,xxx0	
DMA_UR3R_STA	UR3R_DMA Status Register 7EFA5AH	-	-	-	-	-	RXLOSS UR3RIF	xxxx,xx00		
DMA_UR3R_AMT	UR3R_DMA transfer total bytes 7EFA5BH									0000,0000
DMA_UR3R_AMTH	UR3R_DMA transfer total bytes 7EFA92H									0000,0000
DMA_UR3R_DONE	UR3R_DMA transfer completed byte number 7EFA5CH									0000,0000
DMA_UR3R_DONEH	UR3R_DMA transfer completed bytes 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 Register 7EFA60H UR4TIE	-	-	-	UR4TIP[1:0]	UR4TPY[1:0] 0xxx,0000				
DMA_UR4T_CR	UR4T_DMA Control Register 7EFA61H ENUR4T TRIG		-	-	-	-	-	-	-	00xx,xxxx
DMA_UR4T_STA	UR4T_DMA Status Register 7EFA62H	-	-	-	-	-	TXO/VW -		UR4TIF	xxxx,x0x0
DMA_UR4T_AMT	UR4T_DMA transfer total bytes 7EFA63H									0000,0000
DMA_UR4T_AMTH	UR4T_DMA transfer total bytes 7EFA94H									0000,0000
DMA_UR4T_DONE	UR4T_DMA transfer completed bytes 7EFA64H									0000,0000
DMA_UR4T_DONEH	UR4T_DMA transfer completed bytes 7EFA95H									0000,0000
DMA_UR4T_TXAH	UR4T_DMA send high address 7EFA65H									0000,0000
DMA_UR4T_RXAL	UR4T_DMA transmit low address 7EFA66H									0000,0000
DMA_UR4R_CFG	UR4R_DMA Configuration Register 7EFA68H UR4RIE	-	-	-	UR4RIP[1:0]	UR4RPY[1:0] 0xxx,0000				
DMA_UR4R_CR	UR4R_DMA Control Register 7EFA69H ENUR4R -		TRIG	-	-	-	-	-	CLRFIFO 0x0,xxx0	
DMA_UR4R_STA	UR4R_DMA Status Register 7EFA6AH	-	-	-	-	-	RXLOSS UR4RIF	xxxx,xx00		
DMA_UR4R_AMT	UR4R_DMA transfer total bytes 7EFA6BH									0000,0000

DMA_UR4R_AMTH	JR4R_DMA transfer total bytes 7EFA96H										0000,0000	
DMA_UR4R_DONE	UR4R_DMA transfer completed byte number 7EFA6CH										0000,0000	
DMA_UR4R_DONEH	UR4R_DMA transfer completed bytes 7EA97H										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	E	NLCM	TRIGWC	TRIGWD	TRIGRC	TRIGRD	-	-	0000,0xxx	
DMA_LCM_STA	LCM_DMA Status Register	7EFA72H	-	-	-	-	-	-	-TXO	VW	LCMIF xxxx,xx00	
DMA_LCM_AMT	LCM_DMA transfer total bytes 7EFA73H										0000,0000	
DMA_LCM_AMTH	LCM_DMA transfer total bytes 7EFA86H										0000,0000	
DMA_LCM_DONE	LCM_DMA transfer completed bytes 7EFA74H										0000,0000	
DMA_LCM_DONEH	LCM_DMA transfer completed bytes 7EFA87H										0000,0000	
DMA_LCM_TXAH	LCM_DMA transmit high address	7EFA75H										0000,0000
DMA_LCM_TXAL	LCM_DMA transmit 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 register	7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]		I2CTPTY[1:0] 0xxx,0000			
DMA_I2CT_CR	I2CT_DMA Control Register	7EFA99H	E	NI2CT	TRIG	-	-	-	-	-	00xx,xxxx	
DMA_I2CT_STA	I2CT_DMA Status Register 7EFA9AH		-	-	-	-	-	-	-TXO	VW	-	I2CTIF xxxx,x0x0
DMA_I2CT_AMT	I2CT_DMA transfer total bytes 7EFA9BH										0000,0000	
DMA_I2CT_AMTH	I2CT_DMA transfer total bytes 7EFAA8H										0000,0000	
DMA_I2CT_DONE	I2CT_DMA transfer completed byte number 7EFA9CH										0000,0000	
DMA_I2CT_DONEH	I2CT_DMA transfer completed byte number 7EFAA9H										0000,0000	
DMA_I2CT_TXAH	I2CT_DMA send high address 7EFA9DH										0000,0000	
DMA_I2CT_TXAL	I2CT_DMA transmit low address	7EFA9EH										0000,0000
DMA_I2CR_CFG	I2CR_DMA Configuration Register 7EFAA0H	I2CRIE	-	-	-	-	I2CRIP[1:0]		I2CRPTY[1:0] 0xxx,0000			
DMA_I2CR_CR	I2CR_DMA Control Register 7EFAA1H	E	NI2CR	TRIG	-	-	-	-	-	-	CLRFIFO 00xx,xxxx	
DMA_I2CR_STA	I2CR_DMA Status Register 7EFAA2H		-	-	-	-	-	-	-RXLOSS	I2CRIF	xxxx,xx00	
DMA_I2CR_AMT	I2CR_DMA transfer total bytes 7EFAA3H										0000,0000	
DMA_I2CR_AMTH	I2CR_DMA transfer total bytes 7EFAAAH										0000,0000	
DMA_I2CR_DONE	I2CR_DMA transfer completed byte number 7EFAA4H										0000,0000	
DMA_I2CR_DONEH	I2CR_DMA transfer completed bytes 7EFABABH										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 register	7EFAB0H	I2STIE	-	-	-	I2STIP[1:0]		I2STPTY[1:0] 0xxx,0000			
DMA_I2ST_CR	I2ST_DMA Control Register	7EFAB1H	E	NI2ST	TRIG	-	-	-	-	-	00xx,xxxx	
DMA_I2ST_STA	I2ST_DMA Status Register	7EFAB2H	-	-	-	-	-	-TXO	VW	-	I2STIF xxxx, x0x0	
DMA_I2ST_AMT	I2ST_DMA transfer total bytes 7EFAB3H										0000,0000	
DMA_I2ST_AMTH	I2ST_DMA transfer total bytes 7EFAC0H										0000,0000	
DMA_I2ST_DONE	I2ST_DMA transfer completed byte number 7EFAB4H										0000,0000	
DMA_I2ST_DONEH	I2ST_DMA transfer completed byte number 7EFAC1H										0000,0000	

DMA_I2ST_TXAH I2ST_DMA transmit high address	7EFAB5H								0000,0000
DMA_I2ST_RXAL I2ST_DMA transmit low address	7EFAB6H								0000,0000
DMA_I2SR_CFG I2SR_DMA configuration register	7EFAB8H I2SRIE	-	-	-	I2SRIP[1:0]		I2SRPTY[1:0] 0xxx,0000		
DMA_I2SR_CR I2SR_DMA Control Register	7EFAB9H EI2SR	-	TRIG	-	-	-	-	CLRFIFO 0x00,xxx0	
DMA_I2SR_STA I2SR_DMA Status Register	7EFABAH	-	-	-	-	-	RXLOSS I2SRIF xxxx,xx00		
DMA_I2SR_AMT I2SR_DMA transfer total bytes	7EFABBH								0000,0000
DMA_I2SR_AMTH I2SR_DMA transfer total bytes	7EFAC2H								0000,0000
DMA_I2SR_DONE I2SR_DMA transfer completed byte number	7EFABCH								0000,0000
DMA_I2SR_DONEH I2SR_DMA transfer completed byte number	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 Data read and write between memory and memory (M2M\_DMA)

### 28.2.1 M2M\_DMA Configuration Register (DMA\_M2M\_CFG)

symbol	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: Disable M2M\_DMA interrupt

1: Enable M2M\_DMA interrupt

TXACO: M2M\_DMA source address (read address) changes direction

0: The address is automatically incremented after the data read is completed

1: The address is automatically decremented after the data read is completed

RXACO: M2M\_DMA target address (write address) changes direction

0: The address is automatically incremented after data writing is completed

1: The address is automatically decremented after data writing is completed

M2MIP[1:0]: M2M\_DMA interrupt priority control bits

M2MIP[1:0] Interrupt priority	
00	lowest level (0)
01	Lower (1)
10	more advanced (2)
11	Superlative (3)

M2MPTY[1:0]: M2M\_DMA data bus access priority control bits

M2MPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.2.2 M2M\_DMA Control Register (DMA\_M2M\_CR)

symbol	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: Enable M2M\_DMA function

TRIG: M2M\_DMA data read and write trigger control bit

0: Write 0 has no effect

1: Write 1 to start M2M\_DMA operation,

### 28.2.3 M2M\_DMA Status Register (DMA\_M2M\_STA)

symbol	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

M2M\_DMA interrupt will enter the interrupt service routine. The flag bit needs to be cleared by software

## 28.2.4 M2M\_DMA Transfer Total Byte Register (DMA\_M2M\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and [write 256 bytes](#)

## 28.2.5 M2M\_DMA Transfer Completion Byte Register (DMA\_M2M\_DONE)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_M2M_DONE	7EFA04H	DONE[7:0]						
DMA_M2M_DONEH	7EFA81H	DONE[15:0]:						

The number of bytes that have been read and written currently.

## 28.2.6 M2M\_DMA Transmit Address Register (DMA\_M2M\_TXAx)

symbol	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: Set the source address when reading and writing data. Data will be read from this address when performing an M2M\_DMA operation.

## 28.2.7 M2M\_DMA Receive Address Register (DMA\_M2M\_RXAx)

symbol	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: Set the target address when reading and writing data. When the M2M\_DMA operation is performed, the data will be written from this address.

according to:

## 28.3 ADC Data Automatic Storage (ADC\_DMA)

### 28.3.1 ADC\_DMA CONFIGURATION REGISTER (DMA\_ADC\_CFG)

symbol	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: Enable ADC\_DMA interrupt

ADCIP[1:0]: ADC\_DMA interrupt priority control bits

ADCIP[1:0] Interrupt priority	
00	lowest level (0)
01	Lower (1)
10	more advanced (2)
11	Superlative (3)

ADCPTY[1:0]: ADC\_DMA data bus access priority control bits

ADCPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.3.2 ADC\_DMA Control Register (DMA\_ADC\_CR)

symbol	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: Enable ADC\_DMA function

TRIG: ADC\_DMA operation trigger control bit

0: Write 0 has no effect

1: Write 1 to start ADC\_DMA operation,

### 28.3.3 ADC\_DMA Status Register (DMA\_ADC\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_STA	7EFA12H	-	-	-	-	-	-	ADCIF

ADCIF: ADC\_DMA interrupt request flag, when ADC\_DMA finishes scanning all enabled ADC channels, the hardware will automatically

ADCIF is set to 1, if the ADC\_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

### 28.3.4 ADC\_DMA Receive Address Register (DMA\_ADC\_RXAx)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_RXA	7EFA17H				ADDR[15:8]			

DMA_ADC_RXA	18H	ADDR[7:0]
-------------	-----	-----------

DMA\_ADC\_RXA: Set the storage address of ADC conversion data during ADC\_DMA operation.

### 28.3.5 ADC\_DMA CONFIGURATION REGISTER 2 (DMA\_ADC\_CFG2)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CFG2	7EF0	-	-	-	-	CVTIMESEL[3:0]			

CVTIMESEL[3:0]: Set the number of ADC conversions for each ADC channel during ADC\_DMA operation

CVTIMESEL[3:0] Conversion times	
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)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_ADC_CHSW0	7EF1	AH CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
DMA_ADC_CHSW1	7EF1	BH CH15 CHh	CH14	CH13	CH12	CH11	CH10	CH9	CH8

ADC channel to scan automatically during ADC\_DMA operation. Channel scanning always starts from the lower-numbered channel.

## 28.3.7 Data storage format of ADC\_DMA

Note: ADC conversion speed and conversion result alignment are set by ADC related registers

XRAM[DMA\_ADC\_RXA+0] = high byte of the 1st ADC conversion result of the 1st channel enabled;

XRAM[DMA\_ADC\_RXA+1] = the low byte of the 1st ADC conversion result of the enabled 1st channel;

XRAM[DMA\_ADC\_RXA+2] = high byte of the 2nd ADC conversion result of the enabled 1st channel;

XRAM[DMA\_ADC\_RXA+3] = the low byte of the 2nd ADC conversion result of the enabled 1st channel;

...

XRAM[DMA\_ADC\_RXA+2n-2] = the high byte of the nth ADC conversion result of the enabled channel 1;

XRAM[DMA\_ADC\_RXA+2n-1] = the 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 after the average value of the n ADC conversion results of the first channel;

XRAM[DMA\_ADC\_RXA+2n+2] = high byte of the average value of n ADC conversion results of the 1st channel;

XRAM[DMA\_ADC\_RXA+2n+3] = the low byte of the average value of n ADC conversion results of channel 1;

XRAM[DMA\_ADC\_RXA+(2n+3)+0] = high byte of the 1st ADC conversion result of the enabled 2nd channel;

XRAM[DMA\_ADC\_RXA+(2n+3)+1] = the low byte of the 1st ADC conversion result of the enabled 2nd channel;

XRAM[DMA\_ADC\_RXA+(2n+3)+2] = high byte of the 2nd ADC conversion result of the enabled 2nd channel;

XRAM[DMA\_ADC\_RXA+(2n+3)+3] = the low byte of the 2nd ADC conversion result of the enabled 2nd channel;

...

XRAM[DMA\_ADC\_RXA+(2n+3)+2n-2] = the high byte of the nth ADC conversion result of the enabled channel 2;

XRAM[DMA\_ADC\_RXA+(2n+3)+2n-1] = the low byte of the nth ADC conversion result of the enabled channel 2;

XRAM[DMA\_ADC\_RXA+(2n+3)+2n] = ADC channel number of channel 2;

XRAM[DMA\_ADC\_RXA+(2n+3)+2n+1] = the remainder after the average value of the n ADC conversion results of channel 2;

XRAM[DMA\_ADC\_RXA+(2n+3)+2n+2] = high byte of the average value of n ADC conversion results of channel 2;

XRAM[DMA\_ADC\_RXA+(2n+3)+2n+3] = the low byte of the average value of n ADC conversion results of channel 2;

...

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+0] = the high byte of the 1st ADC conversion result of the enabled mth channel;

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+1] = the low byte of the 1st ADC conversion result of the mth channel enabled;

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+2] = the high byte of the 2nd ADC conversion result of the enabled mth channel;

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+3] = the low byte of the 2nd ADC conversion result of the enabled mth channel;

...

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+2n-2] = the high byte of the nth ADC conversion result of the enabled mth channel;

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+2n-1] = the low byte of the nth ADC conversion result of the enabled mth channel;

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+2n] = ADC channel number of mth channel;

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+2n+1] = the remainder after the average value of the n ADC conversion results of the mth channel;

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+2n+2] = high byte of the average value of n ADC conversion results of channel m;

XRAM[DMA\_ADC\_RXA+(m-1)(2n+3)+2n+3] = the low byte of the average value of n ADC conversion results of the mth channel;

The form is as follows:

ADC channel	offset address	data
Channel 1	0	High byte of 1st ADC conversion result of enabled 1st channel
	1	Low byte of the 1st ADC conversion result of the enabled 1st channel
	2	High byte of 2nd ADC conversion result of enabled 1st channel
	3	Low byte of the 2nd ADC conversion result of the enabled 1st channel
	...	...
	2n-2	High byte of the nth ADC conversion result of the enabled channel 1
	2n-1	The low byte of the nth ADC conversion result of the enabled channel 1
	2n	ADC channel number of channel 1
	2n+1	The remainder after the average value of the n ADC conversion results of the 1st channel
	2n+2	High byte of the average value of n ADC conversion results of channel 1
	2n+3	The low byte of the average value of the nth ADC conversion result of the 1st channel
Channel 2	(2n+3) + 0	High byte of 1st ADC conversion result of enabled 2nd channel
	(2n+3) + 1	Low byte of the 1st ADC conversion result of the enabled 2nd channel
	(2n+3) + 2	High byte of 2nd ADC conversion result of enabled 2nd channel
	(2n+3) + 3	Low byte of 2nd ADC conversion result of enabled 2nd channel
	...	...
	(2n+3) + 2n-2	High byte of the nth ADC conversion result of channel 2 enabled
	(2n+3) + 2n-1	Low byte of the nth ADC conversion result of channel 2 enabled
	(2n+3) + 2n	ADC channel number of channel 2
	(2n+3) + 2n+1	The remainder after the average value of the n-th ADC conversion result of channel 2
	(2n+3) + 2n+2	High byte of the average value of n ADC conversion results of channel 2
	(2n+3) + 2n+3	The low byte of the average value of n ADC conversion results of channel 2
mth channel	...	...
	(m-1)(2n+3) + 0	High byte of the 1st ADC conversion result of the mth channel enabled
	(m-1)(2n+3) + 1	Low byte of the 1st ADC conversion result of the mth channel enabled
	(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 channel enabled
	...	...
	(m-1)(2n+3) + 2n-2	High byte of the nth ADC conversion result of the mth channel enabled
	(m-1)(2n+3) + 2n-1	Low byte of the nth ADC conversion result of the mth channel enabled
	(m-1)(2n+3) + 2n	ADC channel number of mth channel
	(m-1)(2n+3) + 2n+1	The remainder after averaging the nth ADC conversion result of channel m
	(m-1)(2n+3) + 2n+2	The high byte of the average value of the nth ADC conversion result of channel m
	(m-1)(2n+3) + 2n+3	The low byte of the average value of the nth ADC conversion result of the mth channel

## 28.4 Data exchange between SPI and memory (SPI\_DMA)

### 28.4.1 SPI\_DMA Configuration Register (DMA\_SPI\_CFG)

symbol	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: Enable SPI\_DMA interrupt

ACT\_TX: SPI\_DMA transmit data control bit

0: Disable SPI\_DMA to transmit data. In master mode, SPI only sends clock to SCLK port, but does not read data from XRAM

data, and do not send data to the MOSI port; in slave mode, the SPI does not read data from XRAM, nor to the MISO port

Send data over the mouth

1: Enable SPI\_DMA to transmit data. In master mode, SPI sends clock to SCLK port and reads data from XRAM at the same time,

And send the data to the MOSI port; in slave mode, the SPI reads the data from the XRAM and sends the data to the MISO port

mouth

ACT\_RX: SPI\_DMA receive data control bit

0: Disable SPI\_DMA to receive data. In master mode, the SPI only sends the clock to the SCLK port, but does not read from the MISO port data, also write data to XRAM; in slave mode, SPI does not read data from the MOSI port, nor write data to XRAM.

1: Enable SPI\_DMA to receive data. In master mode, the SPI sends the clock to the SCLK port and reads the data from the MISO port at the same time. data, and write data to XRAM; in slave mode, SPI reads data from MOSI port and writes it to XRAM.

SPIIP[1:0]: SPI\_DMA interrupt priority control bits

SPIIP[1:0] Interrupt priority	
00	lowest level (0)
01	Lower (1)
10	more advanced (2)
11	Superlative (3)

SPIPTY[1:0]: SPI\_DMA data bus access priority control bits

SPIPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.4.2 SPI\_DMA Control Register (DMA\_SPI\_CR)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CR 7EFA21H	ENSPIM TRIG_M TRIG_S			-	-	-	-	CLRFIFO

ENSPIM: SPI\_DMA function enable control bit

0: Disable SPI\_DMA function

1: Enable SPI\_DMA function

TRIG\_M: SPI\_DMA master mode trigger control bit

0: Write 0 has no effect

1: Write 1 to start SPI\_DMA master mode operation,

TRIG\_S: SPI\_DMA slave mode trigger control bit

0: Write 0 has no effect

1: Write 1 to start SPI\_DMA slave mode operation,

CLRFIFO: Clear SPI\_DMA receive FIFO control bit

0: Write 0 has no effect

1: Before starting the SPI\_DMA operation, first clear the built-in FIFO of SPI\_DMA

### 28.4.3 SPI\_DMA Status Register (DMA\_SPI\_STA)

symbol	address B7	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_STA	7EFA22H	SPIIF:	-	-	-	-	-	TXOVW RXLOSS SPIIF	

SPI\_DMA interrupt request flag, when the SPI\_DMA data exchange is completed, the hardware will automatically set SPIIF to 1, if SPI\_DMA is enabled

The interrupt enters the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: SPI\_DMA receive data discard flag. During the SPI\_DMA operation, when the XRAM bus is too busy, it is too late to clear

When the empty receive FIFO of SPI\_DMA causes the data received by SPI\_DMA to be discarded automatically, the hardware will automatically set RXLOSS to 1.

The flag bit needs to be cleared by software

TXOVW: SPI\_DMA data overwrite flag. SPI\_DMA is in the process of data transfer, the SPI in master mode writes the SPDAT register

When the controller triggers the SPI data transmission again, the data transmission will fail, and the hardware will automatically set TXOVW to 1. flag bit required  
software reset

### 28.4.4 SPI\_DMA Transfer Total Bytes Register (DMA\_SPI\_AMT)

symbol	address B7	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and write 256 bytes

### 28.4.5 SPI\_DMA Transfer Completion Byte Register (DMA\_SPI\_DONE)

symbol	address B7	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 number of bytes currently transferred.

### 28.4.6 SPI\_DMA Transmit Address Register (DMA\_SPI\_TXAx)

symbol	address B7	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: Set the source address for data transfer. Data will be read from this address when performing an SPI\_DMA operation.

## 28.4.7 SPI\_DMA Receive Address Register (DMA\_SPI\_RXAx)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_RXAH	7EFA	27H							ADDR[15:8]
DMA_SPI_RXAL	7EF	A28H							ADDR[7:0]

DMA\_SPI\_RXA: Set the target address for data transfer. Data will be written from this address when performing an SPI\_DMA operation.

## 28.4.8 SPI\_DMA Configuration Register 2 (DMA\_SPI\_CFG2)

symbol	address	B7	B6	B5	B4	B3	B2	B1	B0
DMA_SPI_CFG2	7EFA	29H	-	-	-	-	WRPSS		SSS[1:0]

WRPSS: Enable SS pin control bit during SPI\_DMA

0: During SPI\_DMA transfer, the SS pin is not automatically controlled

1: During the SPI\_DMA transfer process, the SS pin is automatically pulled down, and the original state is automatically restored after the transfer is completed.

SSS[1:0]: During the SPI\_DMA process, automatically control the SS selection bit

SSS[1:0]	SS feet
00	P1.2/P5.4[1]
01	P2.2
10	P7.4
11	P3.5

[1] : For models with P1.2 port, this function is on P1.2 port, for models without P1.2 port, this function is on P5.4 port

## 28.5 Data exchange between serial port 1 and memory (UR1T\_DMA, UR1R\_DMA)

### 28.5.1 UR1T\_DMA Configuration Register (DMA\_UR1T\_CFG)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CFG	7EF A30H	UR1TIE	-	-	-	UR1TIP[1:0]	UR1TPY[1:0]	

UR1TIE: UR1T\_DMA interrupt enable control bit

0: Disable UR1T\_DMA interrupt

1: Enable UR1T\_DMA interrupt

UR1TIP[1:0]: UR1T\_DMA interrupt priority control bits

UR1TIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

UR1TPY[1:0]: UR1T\_DMA data bus access priority control bits

UR1TPY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.5.2 UR1T\_DMA Control Register (DMA\_UR1T\_CR)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_CR	7EF A31H	ENUR1T	TRIG	-	-	-	-	-

ENUR1T: UR1T\_DMA function enable control bit

0: Disable UR1T\_DMA function

1: Enable UR1T\_DMA function

TRIG: UR1T\_DMA serial port 1 transmit trigger control bit

0: Write 0 has no effect

1: Write 1 to start UR1T\_DMA to send data automatically

### 28.5.3 UR1T\_DMA Status Register (DMA\_UR1T\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1T_STA	7EF A32H	-	-	-	-	TXOVW	-	UR1TIF

UR1TIF: UR1T\_DMA interrupt request flag, when the UR1T\_DMA data transmission is completed, the hardware will automatically set UR1TIF to 1, if

Enable the UR1T\_DMA interrupt to enter the interrupt service routine. The flag bit needs to be cleared by software

TXOVW: UR1T\_DMA data overwrite flag. UR1T\_DMA is in the process of data transfer, the serial port writes the SBUF register again

When the serial port is triggered to send data, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag needs to be cleared by software zero

## 28.5.4 UR1T\_DMA Transfer Total Bytes Register (DMA\_UR1T\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0, 1 byte is read and written, and when AMT is set to 255,

Read and write 256 bytes

## 28.5.5 UR1T\_DMA Transfer Completion Byte Register (DMA\_UR1T\_DONE)

symbol	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 number of bytes that have been sent currently.

## 28.5.6 UR1T\_DMA Transmit Address Register (DMA\_UR1T\_TXAx)

symbol	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 of automatic data transmission. Data is read from this address when performing a UR1T\_DMA operation.

## 28.5.7 UR1R\_DMA Configuration Register (DMA\_UR1R\_CFG)

symbol	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: Enable UR1R\_DMA interrupt

UR1RIP[1:0]: UR1R\_DMA interrupt priority control bits

UR1RIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

UR1RPTY[1:0]: UR1R\_DMA data bus access priority control bits

UR1RPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

## 28.5.8 UR1R\_DMA Control Register (DMA\_UR1R\_CR)

symbol	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: Enable UR1R\_DMA function

TRIG: UR1R\_DMA serial port 1 receive trigger control bit

0: Write 0 has no effect

1: Write 1 to start UR1R\_DMA to automatically receive data

CLRFIFO: Clear UR1R\_DMA receive FIFO control bit

0: Write 0 has no effect

1: Before starting the UR1R\_DMA operation, clear the built-in FIFO of the UR1R\_DMA

## 28.5.9 UR1R\_DMA Status Register (DMA\_UR1R\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR1R_STA	7EFA3AH	-	-	-	-	-	-	RXLOSS UR1RIF

UR1RIF: UR1R\_DMA interrupt request flag, when UR1R\_DMA receives data, the hardware will automatically set UR1RIF to 1,

If the UR1R\_DMA interrupt is enabled, the interrupt service routine will be entered. The flag bit needs to be cleared by software

RXLOSS: UR1R\_DMA receive data discard flag. During UR1R\_DMA operation, when the XRAM bus is too busy,

When the receive FIFO of UR1R\_DMA cannot be cleared, the data received by UR1R\_DMA is automatically discarded, the hardware will automatically

RXLOSS is set to 1. The flag bit needs to be cleared by software

## 28.5.10 UR1R\_DMA Transfer Total Bytes Register (DMA\_UR1R\_AMT)

symbol	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 that need to be read and written.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and write 256 bytes

## 28.5.11 UR1R\_DMA Transfer Complete Byte Register (DMA\_UR1R\_DONE)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0	
DMA_UR1R_DONE	7EFABCH	DONE[7:0]							
DMA_UR1R_DONEH	7EFA8BH	DONE[15:8]							

DONE[15:0]: The number of bytes that have been received currently.

## 28.5.12 UR1R\_DMA Receive Address Register (DMA\_UR1R\_RXAx)

symbol	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: Set the target address for automatically receiving data. Data is written from this address when performing a UR1R\_DMA operation.

STCMCU

## 28.6 Data exchange between serial port 2 and memory (UR2T\_DMA, UR2R\_DMA)

### 28.6.1 UR2T\_DMA Configuration Register (DMA\_UR2T\_CFG)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CFG	7EF A40H	UR2TIE	-	-	-	UR2TIP[1:0]	UR2TPY[1:0]	

UR2TIE: UR2T\_DMA interrupt enable control bit

0: Disable UR2T\_DMA interrupt

1: Enable UR2T\_DMA interrupt

UR2TIP[1:0]: UR2T\_DMA interrupt priority control bits

UR2TIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

UR2TPY[1:0]: UR2T\_DMA data bus access priority control bits

UR2TPY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.6.2 UR2T\_DMA Control Register (DMA\_UR2T\_CR)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_CR	7EF A41H	ENUR2T	TRIG	-	-	-	-	-

ENUR2T: UR2T\_DMA function enable control bit

0: Disable UR2T\_DMA function

1: Enable UR2T\_DMA function

TRIG: UR2T\_DMA serial port 1 transmit trigger control bit

0: Write 0 has no effect

1: Write 1 to start UR2T\_DMA to send data automatically

### 28.6.3 UR2T\_DMA Status Register (DMA\_UR2T\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2T_STA	7EF A42H	-	-	-	-	-	TXOVW	-

UR2TIF: UR2T\_DMA interrupt request flag, when the UR2T\_DMA data transmission is completed, the hardware will automatically set UR2TIF to 1, if

Enable the UR2T\_DMA interrupt to enter the interrupt service routine. The flag bit needs to be cleared by software

TXOVW: UR2T\_DMA data overwrite flag. UR2T\_DMA is in the process of data transmission, the serial port writes the S2BUF register again

When the serial port is triggered to send data, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag needs to be cleared by software

zero

## 28.6.4 UR2T\_DMA Transfer Total Bytes Register (DMA\_UR2T\_AMT)

symbol	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 that need to be read and written.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0, 1 byte is read and written, and when AMT is set to 255,

Read and write 256 bytes

## 28.6.5 UR2T\_DMA Transfer Completion Byte Register (DMA\_UR2T\_DONE)

symbol	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 number of bytes that have been sent currently.

## 28.6.6 UR2T\_DMA Transmit Address Register (DMA\_UR2T\_TXAx)

symbol	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\_TAXA: Sets the source address of automatic data transmission. Data will be read from this address when performing a UR2T\_DMA operation.

## 28.6.7 UR2R\_DMA Configuration Register (DMA\_UR2R\_CFG)

symbol	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: Enable UR2R\_DMA interrupt

UR2RIP[1:0]: UR2R\_DMA interrupt priority control bits

UR2RIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

UR2RPTY[1:0]: UR2R\_DMA data bus access priority control bits

UR2RPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

## 28.6.8 UR2R\_DMA Control Register (DMA\_UR2R\_CR)

symbol	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: Enable UR2R\_DMA function

TRIG: UR2R\_DMA serial port 1 receive trigger control bit

0: Write 0 has no effect

1: Write 1 to start UR2R\_DMA to automatically receive data

CLRFIFO: Clear UR2R\_DMA receive FIFO control bit

0: Write 0 has no effect

1: Before starting the UR2R\_DMA operation, clear the built-in FIFO of the UR2R\_DMA

## 28.6.9 UR2R\_DMA Status Register (DMA\_UR2R\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR2R_STA	7EFA4AH	-	-	-	-	-	RXLOSS	UR2RIF

UR2RIF: UR2R\_DMA interrupt request flag, when UR2R\_DMA receives data, the hardware will automatically set UR2RIF to 1,

If the UR2R\_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR2R\_DMA receive data discard flag. During UR2R\_DMA operation, when the XRAM bus is too busy,

When the receive FIFO of UR2R\_DMA cannot be cleared and the data received by UR2R\_DMA is automatically discarded, the hardware will automatically

RXLOSS is set to 1. The flag bit needs to be cleared by software

## 28.6.10 UR2R\_DMA Transfer Total Bytes Register (DMA\_UR2R\_AMT)

symbol	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 that need to be read and written.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and write 256 bytes

## 28.6.11 UR2R\_DMA Transfer Complete Byte Register (DMA\_UR2R\_DONE)

symbol	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 number of bytes that have been received currently.

## 28.6.12 UR2R\_DMA Receive Address Register (DMA\_UR2R\_RXAx)

symbol	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: Set the target address for automatically receiving data. Data is written from this address when performing a UR2R\_DMA operation.

STCMCU

## 28.7 Data exchange between serial port 3 and memory (UR3T\_DMA, UR3R\_DMA)

### 28.7.1 UR3T\_DMA Configuration Register (DMA\_UR3T\_CFG)

symbol	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: Enable UR3T\_DMA interrupt

UR3TIP[1:0]: UR3T\_DMA interrupt priority control bits

UR3TIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

UR3TPTY[1:0]: UR3T\_DMA data bus access priority control bits

UR3TPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.7.2 UR3T\_DMA Control Register (DMA\_UR3T\_CR)

symbol	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: Enable UR3T\_DMA function

TRIG: UR3T\_DMA serial port 1 transmit trigger control bit

0: Write 0 has no effect

1: Write 1 to start UR3T\_DMA to send data automatically

### 28.7.3 UR3T\_DMA Status Register (DMA\_UR3T\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3T_STA	7EFA52H	-	-	-	-	TXOVW	-	UR3TIF

UR3TIF: UR3T\_DMA interrupt request flag, when the UR3T\_DMA data transmission is completed, the hardware will automatically set UR3TIF to 1, if

Enable the UR3T\_DMA interrupt to enter the interrupt service routine. The flag bit needs to be cleared by software

TXOVW: UR3T\_DMA data overwrite flag. UR3T\_DMA is in the process of data transmission, the serial port writes the S3BUF register again

When the serial port is triggered to send data, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag needs to be cleared by software

zero

## 28.7.4 UR3T\_DMA Transfer Total Bytes Register (DMA\_UR3T\_AMT)

symbol	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 that need to be read and written.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0, 1 byte is read and written, and when AMT is set to 255,

Read and write 256 bytes

## 28.7.5 UR3T\_DMA Transfer Completion Byte Register (DMA\_UR3T\_DONE)

symbol	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 number of bytes that have been sent currently.

## 28.7.6 UR3T\_DMA Transmit Address Register (DMA\_UR3T\_TXAx)

symbol	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: Set the source address of automatic data transmission. Data will be read from this address when performing a UR3T\_DMA operation.

## 28.7.7 UR3R\_DMA Configuration Register (DMA\_UR3R\_CFG)

symbol	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: Enable UR3R\_DMA interrupt

UR3RIP[1:0]: UR3R\_DMA interrupt priority control bits

UR3RIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

UR3RPTY[1:0]: UR3R\_DMA data bus access priority control bits

UR3RPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

## 28.7.8 UR3R\_DMA Control Register (DMA\_UR3R\_CR)

symbol	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: Enable UR3R\_DMA function

TRIG: UR3R\_DMA serial port 1 receive trigger control bit

0: Write 0 has no effect

1: Write 1 to start UR3R\_DMA to automatically receive data

CLRFIFO: Clear UR3R\_DMA receive FIFO control bit

0: Write 0 has no effect

1: Before starting the UR3R\_DMA operation, first clear the built-in FIFO of the UR3R\_DMA

## 28.7.9 UR3R\_DMA Status Register (DMA\_UR3R\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR3R_STA	7EFA5AH	-	-	-	-	-	-	RXLOSS UR3RIF

UR3RIF: UR3R\_DMA interrupt request flag, when UR3R\_DMA receives data, the hardware will automatically set UR3RIF to 1,

If the UR3R\_DMA interrupt is enabled, the interrupt service routine will be entered. The flag bit needs to be cleared by software

RXLOSS: UR3R\_DMA receive data discard flag. During UR3R\_DMA operation, when the XRAM bus is too busy,

When the receive FIFO of UR3R\_DMA cannot be cleared, the data received by UR3R\_DMA is automatically discarded, the hardware will automatically

RXLOSS is set to 1. The flag bit needs to be cleared by software

## 28.7.10 UR3R\_DMA Transfer Total Bytes Register (DMA\_UR3R\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and write 256 bytes

## 28.7.11 UR3R\_DMA Transfer Completion Byte Register (DMA\_UR3R\_DONE)

symbol	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 number of bytes that have been received currently.

## 28.7.12 UR3R\_DMA Receive Address Register (DMA\_UR3R\_RXAx)

symbol	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: Set the target address for automatically receiving data. Data will be written from this address when performing a UR3R\_DMA operation.

STCMCU

## 28.8 Data exchange between serial port 4 and memory (UR4T\_DMA, UR4R\_DMA)

### 28.8.1 UR4T\_DMA Configuration Register (DMA\_UR4T\_CFG)

symbol	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: Enable UR4T\_DMA interrupt

UR4TIP[1:0]: UR4T\_DMA interrupt priority control bits

UR4TIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

UR4TPTY[1:0]: UR4T\_DMA data bus access priority control bits

UR4TPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.8.2 UR4T\_DMA Control Register (DMA\_UR4T\_CR)

symbol	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: Enable UR4T\_DMA function

TRIG: UR4T\_DMA serial port 1 transmit trigger control bit

0: Write 0 has no effect

1: Write 1 to start UR4T\_DMA to send data automatically

### 28.8.3 UR4T\_DMA Status Register (DMA\_UR4T\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4T_STA	7EFA62H	-	-	-	-	TXOVW	-	UR4TIF

UR4TIF: UR4T\_DMA interrupt request flag, when the UR4T\_DMA data transmission is completed, the hardware will automatically set UR4TIF to 1, if

Enable the UR4T\_DMA interrupt to enter the interrupt service routine. The flag bit needs to be cleared by software

TXOVW: UR4T\_DMA data overwrite flag. UR4T\_DMA is in the process of data transmission, the serial port writes the S4BUF register again

When the serial port is triggered to send data, the data transmission will fail. At this time, the hardware will automatically set TXOVW to 1. The flag needs to be cleared by software zero

## 28.8.4 UR4T\_DMA Transfer Total Bytes Register (DMA\_UR4T\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0, 1 byte is read and written, and when AMT is set to 255,

Read and write 256 bytes

## 28.8.5 UR4T\_DMA Transfer Completion Byte Register (DMA\_UR4T\_DONE)

symbol	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 number of bytes that have been sent currently.

## 28.8.6 UR4T\_DMA Transmit Address Register (DMA\_UR4T\_TXAx)

symbol	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: Set the source address of automatic data transmission. Data is read from this address when performing a UR4T\_DMA operation.

## 28.8.7 UR4R\_DMA Configuration Register (DMA\_UR4R\_CFG)

symbol	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: Enable UR4R\_DMA interrupt

UR4RIP[1:0]: UR4R\_DMA interrupt priority control bits

UR4RIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

UR4RPTY[1:0]: UR4R\_DMA data bus access priority control bits

UR4RPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

## 28.8.8 UR4R\_DMA Control Register (DMA\_UR4R\_CR)

symbol	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: Enable UR4R\_DMA function

TRIG: UR4R\_DMA serial port 1 receive trigger control bit

0: Write 0 has no effect

1: Write 1 to start UR4R\_DMA to automatically receive data

CLRFIFO: Clear UR4R\_DMA receive FIFO control bit

0: Write 0 has no effect

1: Before starting the UR4R\_DMA operation, clear the built-in FIFO of the UR4R\_DMA

## 28.8.9 UR4R\_DMA Status Register (DMA\_UR4R\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_UR4R_STA	7EFA5AH	-	-	-	-	-	-	RXLOSS UR4RIF

UR4RIF: UR4R\_DMA interrupt request flag, when UR4R\_DMA receives data, the hardware will automatically set UR4RIF to 1,

If the UR4R\_DMA interrupt is enabled, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: UR4R\_DMA receive data discard flag. During UR4R\_DMA operation, when the XRAM bus is too busy,

When the receive FIFO of UR4R\_DMA cannot be cleared and the data received by UR4R\_DMA is automatically discarded, the hardware will automatically

RXLOSS is set to 1. The flag bit needs to be cleared by software

## 28.8.10 UR4R\_DMA Transfer Total Bytes Register (DMA\_UR4R\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and write 256 bytes

## 28.8.11 UR4R\_DMA Transfer Complete Byte Register (DMA\_UR4R\_DONE)

symbol	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 number of bytes that have been received currently.

## 28.8.12 UR4R\_DMA Receive Address Register (DMA\_UR4R\_RXAx)

symbol	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: Set the target address for automatically receiving data. Data is written from this address when performing a UR4R\_DMA operation.

STCMCU

## 28.9 Data read and write between LCM and memory (LCM\_DMA)

### 28.9.1 LCM\_DMA CONFIGURATION REGISTER (DMA\_LCM\_CFG)

symbol	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: Enable LCM\_DMA interrupt

LCMIP[1:0]: LCM\_DMA interrupt priority control bits

LCMIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

LCMPTY[1:0]: LCM\_DMA data bus access priority control bits

LCMPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.9.2 LCM\_DMA Control Register (DMA\_LCM\_CR)

symbol	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: Enable LCM\_DMA function

TRIGWC: LCM\_DMA transmit command mode trigger control bit

0: Write 0 has no effect

1: Write 1 to start LCM\_DMA transmit command mode operation

TRIGWD: LCM\_DMA transmit data mode trigger control bit

0: Write 0 has no effect

1: Write 1 to start LCM\_DMA transmit data mode operation

TRIGRC: LCM\_DMA Read Command Mode Trigger Control Bit

0: Write 0 has no effect

1: Write 1 to start LCM\_DMA read command mode operation

TRIGRD: LCM\_DMA read data mode trigger control bit

0: Write 0 has no effect

1: Write 1 to start LCM\_DMA read data mode operation

### 28.9.3 LCM\_DMA Status Register (DMA\_LCM\_STA)

symbol	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 LCM\_DMA can be interrupted, it will enter the interrupt service routine. The flag bit needs to be cleared by software

TXOVW: LCM\_DMA data overwrite flag. LCM\_DMA is in the process of data transfer, LCMIF writes LCMIFDATL and

When the LCMIDDATH register is set, data transmission will fail, and the hardware will automatically set TXOVW to 1. Flags require software clear

### 28.9.4 LCM\_DMA Transfer Total Bytes Register (DMA\_LCM\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and write 256 bytes

### 28.9.5 LCM\_DMA Transfer Completion Byte Register (DMA\_LCM\_DONE)

symbol	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 number of bytes that have been transferred currently.

### 28.9.6 LCM\_DMA Transmit Address Register (DMA\_LCM\_TXAx)

symbol	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: Set the source address for data transfer. Data is read from this address when performing an LCM\_DMA operation.

### 28.9.7 LCM\_DMA Receive Address Register (DMA\_LCM\_RXAx)

symbol	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: Set the target address for data transfer. When LCM\_DMA operation is performed, data will be written from this address.

according to:

## 28.10 Data exchange between I2C and memory (I2CT\_DMA, I2CR\_DMA)

### 28.10.1 I2CT\_DMA CONFIGURATION REGISTER (DMA\_I2CT\_CFG)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CT_CFG 7EFA98H	I2CTIE	-	-	-	I2CTIP[1:0]	I2CTPPY[1:0]		

I2CTIE: I2CT\_DMA interrupt enable control bit

0: Disable I2CT\_DMA interrupt

1: Enable I2CT\_DMA interrupt

I2CTIP[1:0]: I2CT\_DMA interrupt priority control bits

I2CTIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

I2CTPPY[1:0]: I2CT\_DMA data bus access priority control bits

I2CTPPY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.10.2 I2CT\_DMA Control Register (DMA\_I2CT\_CR)

symbol	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: Enable I2CT\_DMA function

TRIG: I2CT\_DMA serial port 1 transmit trigger control bit

0: Write 0 has no effect

1: Write 1 to start I2CT\_DMA automatically sending data

### 28.10.3 I2CT\_DMA Status Register (DMA\_I2CT\_STA)

symbol	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 the I2CT\_DMA interrupt can be interrupted, it will enter the interrupt service routine. The flag bit needs to be cleared by software

TXOVW: I2CT\_DMA data overwrite flag. I2CT\_DMA is in the process of data transfer, write the I2C data register I2CTXD

When the data transmission fails, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software

## 28.10.4 I2CT\_DMA Transfer Total Bytes Register (DMA\_I2CT\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0, 1 byte is read and written, and when AMT is set to 255,

Read and write 256 bytes

## 28.10.5 I2CT\_DMA Transfer Completion Byte Register (DMA\_I2CT\_DONE)

symbol	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 number of bytes that have been sent currently.

## 28.10.6 I2CT\_DMA Transmit Address Register (DMA\_I2CT\_TXA)

symbol	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: Set the source address of automatic data transmission. Data will be read from this address when performing an I2CT\_DMA operation.

## 28.10.7 I2CR\_DMA CONFIGURATION REGISTER (DMA\_I2CR\_CFG)

symbol	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: Enable I2CR\_DMA interrupt

I2CRIP[1:0]: I2CR\_DMA interrupt priority control bits

I2CRIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

I2CRPTY[1:0]: I2CR\_DMA data bus access priority control bits

I2CRPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

## 28.10.8 I2CR\_DMA Control Register (DMA\_I2CR\_CR)

symbol	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: Enable I2CR\_DMA function

TRIG: I2CR\_DMA serial port 1 receive trigger control bit

0: Write 0 has no effect

1: Write 1 to start I2CR\_DMA to receive data automatically

CLRFIFO: Clear I2CR\_DMA receive FIFO control bit

0: Write 0 has no effect

1: Before starting the I2CR\_DMA operation, first clear the built-in FIFO of I2CR\_DMA

## 28.10.9 I2CR\_DMA Status Register (DMA\_I2CR\_STA)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_STA	7EFAA2H	-	-	-	-	-	-	RXLOSS I2CRIF

I2CRIF: I2CR\_DMA interrupt request flag bit, when I2CR\_DMA receives data, the hardware will automatically set I2CRIF to 1.

If I2CR\_DMA can be interrupted, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: I2CR\_DMA receive data discard flag. During the I2CR\_DMA operation, when the XRAM bus is too busy,

When the receive FIFO of I2CR\_DMA is cleared and the data received by I2CR\_DMA is automatically discarded, the hardware automatically converts the RXLOSS

Set to 1. The flag bit needs to be cleared by software

## 28.10.10 I2CR\_DMA Transfer Total Bytes Register (DMA\_I2CR\_AMT)

symbol	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 that need to be read and written.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0, 1 byte is read and written, and when AMT is set to 255,

Read and write 256 bytes

## 28.10.11 I2CR\_DMA Transfer Done Byte Register (DMA\_I2CR\_DONE)

symbol	address B7	B6	B5	B4	B3	B2	B1	B0
DMA_I2CR_DONE	7EFAA4H				DONE[7:0]			
DMA_I2CR_DONEH	7EFAAAH				DONE[15:8]			

DONE[15:0]: The number of bytes that have been received currently.

## 28.10.12 I2CR\_DMA Receive Address Register (DMA\_I2CR\_RXAx)

symbol	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: Set the target address for automatically receiving data. Data will be written from this address when performing an I2CR\_DMA operation.

### 28.10.13 I2C\_DMA Control Register (DMA\_I2C\_CR)

symbol	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 response received after sending the data is ACK

1: The response received after sending the data is NAK (requires software to clear)

ERRIE: ACKERR interrupt enable control bit

0: Disable ACKERR interrupt

1: Enable ACKERR [interrupt \(ACKERR interrupt entry address is I2C interrupt entry address FF:00C3H\)](#)

BMMEN: I2C DMA function enable bit

0: Disable I2C\_DMA function

1: Enable I2C\_DMA function

### 28.10.14 I2C\_DMA STATUS REGISTER (DMA\_I2C\_ST)

symbol	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 transferred by I2C\_DMA

Read register: when RDSEL=0, COUNT is the number of bytes to be transferred

When RDSEL=1, COUNT is the number of bytes that have been transmitted

## 28.11 Data exchange between I2S and memory (I2ST\_DMA, I2SR\_DMA)

### 28.11.1 I2ST\_DMA CONFIGURATION REGISTER (DMA\_I2ST\_CFG)

symbol	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: Enable I2ST\_DMA interrupt

I2STIP[1:0]: I2ST\_DMA interrupt priority control bits

I2STIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

I2STPTY[1:0]: I2ST\_DMA data bus access priority control bits

I2STPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

### 28.11.2 I2ST\_DMA Control Register (DMA\_I2ST\_CR)

symbol	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: Enable I2ST\_DMA function

TRIG: I2ST\_DMA serial port 1 transmit trigger control bit

0: Write 0 has no effect

1: Write 1 to start I2ST\_DMA automatically sending data

### 28.11.3 I2ST\_DMA Status Register (DMA\_I2ST\_STA)

symbol	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 enabled

The I2ST\_DMA interrupt enters the interrupt service routine. The flag bit needs to be cleared by software

TXOVW: I2ST\_DMA data overwrite flag. I2ST\_DMA is in the process of data transfer, write the I2S data register I2S\_DRH

and I2S\_DRL, it will cause data transmission failure, at this time, the hardware will automatically set TXOVW to 1. The flag bit needs to be cleared by software

## 28.11.4 I2ST\_DMA Transfer Total Bytes Register (DMA\_I2ST\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and write 256 bytes

## 28.11.5 I2ST\_DMA Transfer Completion Byte Register (DMA\_I2ST\_DONE)

symbol	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 number of bytes that have been sent currently.

## 28.11.6 I2ST\_DMA Transmit Address Register (DMA\_I2ST\_TXAx)

symbol	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: Set the source address of automatic data transmission. Data is read from this address when performing an I2ST\_DMA operation.

## 28.11.7 I2SR\_DMA CONFIGURATION REGISTER (DMA\_I2SR\_CFG)

symbol	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 interrupt

1: Enable I2SR\_DMA interrupt

I2SRIP[1:0]: I2SR\_DMA interrupt priority control bits

I2SRIP[1:0] Interrupt priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

I2SRPTY[1:0]: I2SR\_DMA data bus access priority control bits

I2SRPTY [1:0] Bus priority	
00	lowest level (0)
01	lower level (1)
10	Advanced (2)
11	Superlative (3)

## 28.11.8 I2SR\_DMA Control Register (DMA\_I2SR\_CR)

symbol	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: Enable I2SR\_DMA function

TRIG: I2SR\_DMA serial port 1 receive trigger control bit

0: Write 0 has no effect

1: Write 1 to start I2SR\_DMA to automatically receive data

CLRFIFO: Clear I2SR\_DMA receive FIFO control bit

0: Write 0 has no effect

1: Before starting the I2SR\_DMA operation, first clear the built-in FIFO of the I2SR\_DMA

## 28.11.9 I2SR\_DMA Status Register (DMA\_I2SR\_STA)

symbol	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 automatically set I2SRIF to 1.

If I2SR\_DMA can be interrupted, it will enter the interrupt service routine. The flag bit needs to be cleared by software

RXLOSS: I2SR\_DMA receive data discard flag. During the I2SR\_DMA operation, when the XRAM bus is too busy,

When the data received by I2SR\_DMA is automatically discarded and the receive FIFO of I2SR\_DMA is cleared, the hardware automatically converts the RXLOSS

Set to 1. The flag bit needs to be cleared by software

## 28.11.10 I2SR\_DMA Transfer Total Bytes Register (DMA\_I2SR\_AMT)

symbol	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 that need to read and write data.

Note: The actual number of bytes read and written is (AMT+1), that is, when AMT is set to 0 , 1 byte is read and written , and when AMT is set to 255 ,

Read and write 256 bytes

## 28.11.11 I2SR\_DMA Transfer Done Byte Register (DMA\_I2SR\_DONE)

symbol	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 number of bytes that have been received currently.

## 28.11.12 I2SR\_DMA Receive Address Register (DMA\_I2SR\_RXAx)

symbol	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: Set the target address for automatically receiving data. Data is written from this address when performing an I2SR\_DMA operation.

STCMCU

## 28.12 Example Program

### 28.12.1 Serial port 1 interrupt mode and computer transceiver test - DMA reception timeout interrupt

//The test frequency is 11.0592MHz

```
##include "stc8h.h"
##include "stc32g.h" // For header files, see Download Software
```

\*\*\*\*\* Function Description\*\*\*\*\*

Serial port 1 sends and receives communication programs in full-duplex interrupt mode. Send data to MCU through PC, MCU will automatically store the received data in DMA empty between. When the content received at one time is full of the set DMA space, the data of the storage space is sent by the DMA automatic sending function of serial port 1. according to the output. Use the serial port to receive interrupt to judge the timeout. If no new data is received after the timeout, it means that a series of data has been received, and the The received content is output, and the DMA space is cleared. Use timer as baud rate generator, it is recommended to use 1T mode (unless 12T is used for low baud rate), And choose a clock frequency that is divisible by the baud rate for better accuracy.

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

\*\*\*\*\*\*/

```
#include "stdio.h"

#define MAIN_Fosc      22118400L           // Define the master clock (accurate calculation 115200 baud rate)
#define Baudrate1     115200L
#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|logo
bit      DMATxFlag;
bit      DMARxFlag;
bit      BusyFlag;
u8      Rx_cnt;
u8      RX1_TimeOut;

u8      xdata DMABuffer[256];

void UART1_config(u8 brt);
void DMA_Config(void);

void UartPutc(unsigned char dat)
{
    BusyFlag = 1;
    SBUF = dat;
    while(BusyFlag);
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

void main(void)
```

```

{

    u16 i;

    CKCON = 0x00;
    WTST = 0x00;
    //Set the external data bus speed to the fastest
    //Set the program code to wait for parameters,
    //assign to CPU The speed of executing the program is set to the fastest

    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;
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;
    TR0 = 1;
    //Timer0 set as 1T, 16 bits timer auto-reload,
    //Timer0 interrupt enable
    //Timer0 run

    UART1_config(1);
    DMA_Config();
    EA = 1; //enable total interrupt

    printf("UART1 DMA Timeout Programme!\r\n");
    DMATxFlag = 0;
    DMARxFlag = 0;
    //use Timer1 do baud rate

    while (1)
    {
        if((DMATxFlag) && (DMARxFlag)) { //Judging the send completion flag and the reception completion flag

            Rx_cnt = 0;
            RX1_TimeOut = 0;
            printf("\r\nUART1 DMA FULL!\r\n");
            //UART1 send a string
            DMATxFlag = 0;
            DMA_UR1T_CR = 0xc0;
            //bit7 1: UART1_DMA, bit6 1:enable UART1_DMA auto delivery

            DMARxFlag = 0;
            DMA_UR1R_CR = 0xa1;
            //bit7 1: UART1_DMA, bit5 1: UART1_DMA automatic reception
            //bit0 1:enable FIFO clear

        }

        if(B_1ms) //1ms arrive
        {
            B_1ms = 0;
            if(RX1_TimeOut > 0) { //timeout count

                if(--RX1_TimeOut == 0)
                {
                    DMA_UR1R_CR = 0x00;
                    //closure UART1_DMA
                }
            }
        }
    }
}

```

```

printf("\r\nUART1 Timeout!\r\n"); //UART1      send a string

for(i=0;i<Rx_cnt;i++)
{
    UartPutc(DMABuffer[i]);
    printf("\r\n");
}

Rx_cnt = 0;
DMA_UR1R_CR = 0xa1;           //bit7 1:   UART1_DMA,
                             //bit5 1:   UART1_DMA   automatic reception
                             //bit0 1:enable flag clear
}
}

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:   UART1_DMA,
                             //bit6 1:enable UART1_DMA auto delivery

    DMA_UR1R_CFG = 0x80;
    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: Enable Interrupt
}
}

void SetTimer2Baudrate(u16 dat)
{
    AUXR &= ~(1<<4);           //Timer stop
    AUXR &= ~(1<<3);           //Timer2 set As Timer
    AUXR |= (1<<2);            //Timer2 set as 1T mode
    T2H = dat / 256;
    T2L = dat % 256;
    IE2 &= ~(1<<2);           //Disable interrupts
    AUXR |= (1<<4);            //Timer run enable
}
}

void UART1_config(u8 brt)
{
    /****** Baud rate usage timer2 *****/
    if(brt == 2)
    {
        AUXR |= 0x01;              //S1 BRT Use Timer2;
        SetTimer2Baudrate(65536UL - (MAIN_Fosc / 4) / Baudrate1);
    }

    /****** Baud rate usage timer1 *****/
    else
    {
}

```

```

    TR1 = 0;
    AUXR &= ~0x01;
    AUXR |= (1<<6);
    TMOD &= ~(1<<6);
    TMOD &= ~0x30;
    TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
    TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
    ET1 = 0;
    INTCLKO &= ~0x02;
    TR1 = 1;
}

// *****
SCON = (SCON & 0x3f) | 0x40;
//UART1 Mode :
//0x00: Sync Shift Output ,
//0x40: 8 Bit Data | BaudRate Bit ,
//0x80: 9 Data Fixed|BaudRate Bit ,
//0xc0: 9 Baud Rate
//PS = 1;
ES = 1;
REN = 1;
P_SW1 &= 0x3f;
P_SW1 |= 0x00;
//UART1 switch to:
//0x00: P3.0 P3.1,
//0x40: P3.6 P3.7,
//0x80: P1.6 P1.7,
//0xC0: P4.3 P4.4

RX1_TimeOut = 0;
}

void UART1_int (void) interrupt 4
{
    if(RI)
    {
        RI = 0;
        Rx_cnt++;
        if(Rx_cnt >= DMA_AMT_LEN) Rx_cnt = 0;
        RX1_TimeOut = 5; //if      5ms If no new data is received, it is determined that a series of data has been received.
    }

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

void timer0 (void) interrupt 1
{
    B_1ms = 1; //1ms|logo
}

void UART1_DMA_Interrupt(void) interrupt 13
{
    if (DMA_UR1T_STA & 0x01) //send completed
    {
        DMA_UR1T_STA &= ~0x01;
        DMATxFlag = 1;
}

```

```

}

if (DMA_UR1T_STA & 0x04)                                //data coverage
{
    DMA_UR1T_STA &= ~0x04;
}

if (DMA_UR1R_STA & 0x01)                                //Receive complete
{
    DMA_UR1R_STA &= ~0x01;
    DMARxFlag = 1;
}

if (DMA_UR1R_STA & 0x02)                                //data drop
{
    DMA_UR1R_STA &= ~0x02;
}

```

---

**//File: ISR.ASM**

//Interrupts with interrupt numbers greater than 31 need to perform interrupt entry address remapping processing

CSEG AT 012BH	;POINT_VECTOR
JMP POINT_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

POINT_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 port 1 interrupt mode and computer transceiver test - DMA data verification

---

//The test frequency is **11.0592MHz**

```

//#include "stc8h.h"
#include "stc32g.h"           // For header files, see Download Software

***** Function Description*****
Serial port 1 sends and receives communication programs in full-duplex interrupt mode. Send data to MCU through PC, MCU will automatically store the received data in DMA empty between. The last two bytes of the data packet are used as the check digit, and the routine performs the check with crc16_ccitt algorithm. Set the size when the DMA space is full After the content, the valid data is checked and calculated, and then compared with the last two check digits. Automatic transmission via DMA of serial port 1

The function outputs the data in the storage space. Use timer as baud rate generator, it is recommended to use 1T mode (unless 12T is used for low baud rate), and select Select a clock frequency that is divisible by the baud rate for improved accuracy.

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

```

```
#include "stdio.h"
```

```

#include "crc16.h"

#define MAIN_Fosc      22118400L           // Define the master clock (accurate calculation 115200 baud rate)
#define Baudrate1     115200L

#define DMA_AMT_LEN 255                  // Set the total number of bytes transferred (0~255) : DMA_AMT_LEN+1

bit      DMATxFlag;
bit      DMARxFlag;

u8      xdata DMABuffer[256];

void UART1_config(u8 brt);
void DMA_Config(void);

void UartPutc(unsigned char dat)
{
    SBUF = dat;
    while(TI == 0);
    TI = 0;
}

char putchar(char c)
{
    UartPutc(c);
    return c;
}

/*****CRC calculation function*******/
u16 crc16_ccitt(u8 *pbuf, u16 len)
{
    unsigned short code crc16_ccitt_table[256] =
    {
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
        0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
        0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
        0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
        0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
        0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
        0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
        0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
        0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
        0xDBFD, 0xCBDC, 0xFBBD, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
        0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
        0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
        0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
        0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
        0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
        0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
        0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
        0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
        0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
        0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
        0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
        0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
    };
}

```

```

0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
0x4A75, 0x5A54, 0x6A37, 0x7A16, 0xAF1, 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 the external data bus speed to the fastest
    WTST = 0x00; // Set the program code to wait for parameters,
                  // assign to CPU The speed of executing the program is set to the fastest

    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; // enable total interrupt

    DMATxFlag = 0;
    DMARxFlag = 0;

    while (1)
    {
}

```

```

if((DMA_TxFlag) && (DMA_RxFlag))
{
    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);
    }
    DMA_TxFlag = 0;
    DMA_UR1T_CR = 0xc0;                                //bit7 1:      UART1_DMA,
                                                        //bit6 1:enable UART1_DMA auto delivery

    DMA_RxFlag = 0;
    DMA_UR1R_CR = 0xa1;                                //bit7 1:      UART1_DMA,
                                                        //bit5 1:      UART1_DMA automatic reception
                                                        //bit0 1:enable FIFO clear
}
}

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:      UART1_DMA,
                                                        //bit6 1:enable UART1_DMA auto delivery

    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:      UART1_DMA,
                                                        //bit5 1:enable UART1_DMA automatic reception bit0 1:clear FIFO
}

void SetTimer2Baudrate(u16 dat)
{
    AUXR &= ~(1<<4);
    AUXR &= ~(1<<3);
    AUXR |= (1<<2);
    T2H = dat / 256;
    T2L = dat % 256;
    IE2 &= ~(1<<2);
    AUXR |= (1<<4);
}

void UART1_config(u8 brt)
{
    /****** Baud rate usage timer2 ******/

```

STCMU

```

if(brt == 2)
{
    AUXR |= 0x01;                                //S1 BRT Use Timer2;
    SetTimer2Baudrate(65536UL - (MAIN_Fosc / 4) / Baudrate1);
}

/*****      Baud rate usage timer1 *****/
else
{
    TR1 = 0;                                     //S1 BRT Use Timer1;
    AUXR &= ~0x01;                                //Timer1 set as 1T mode
    AUXR |= (1<<6);                            //Timer1 set As Timer
    TMOD &= ~(1<<6);                           //Timer1_16bitAutoReload;
    TMOD &= ~0x30;                               //Timer1_16bitAutoReload;
    TH1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) / 256);
    TL1 = (u8)((65536UL - (MAIN_Fosc / 4) / Baudrate1) % 256);
    ET1 = 0;                                     //Disable interrupts
    INTCLKO &= ~0x02;                            //Do not output clock
    TR1 = 1;
}
/***** *****/

```

**SCON = (SCON & 0x3f) | 0x40;** //UART1 Mode ,  
//0x00: Sync Shift Output ,  
//0x40: 8 Bit Data //BaudRate Bit ,  
//0x80: 9 Data Fixed //BaudRate Bit ,  
//0xc0: 9 Baud Rate  
//high priority interrupt  
//enable interrupt  
//Allow to receive

// PS = 1;  
// ES = 1;  
REN = 1; //REnabled  
P\_SW1 &= 0x3f;  
P\_SW1 |= 0x00;

}

```

void UART1_DMA_Interrupt(void) interrupt 13
{
    if(DMA_UR1T_STA & 0x01)                      //send completed
    {
        DMA_UR1T_STA &= ~0x01;
        DMATxFlag = 1;
    }
    if(DMA_UR1T_STA & 0x04)                      //data coverage
    {
        DMA_UR1T_STA &= ~0x04;
    }

    if(DMA_UR1R_STA & 0x01)                      //Receive complete
    {
        DMA_UR1R_STA &= ~0x01;
        DMARxFlag = 1;
    }
    if(DMA_UR1R_STA & 0x02)                      //data drop
    {
        DMA_UR1R_STA &= ~0x02;
    }
}

```

//File: ISR.ASM

//Interrupts with interrupt numbers greater than 31 need to perform interrupt entry address remapping processing

<b>CSEG AT 012BH</b>	<b>;POINT_VECTOR</b>
<b>JMP POINT_ISR</b>	
<b>CSEG AT 0133H</b>	<b>;P1INT_VECTOR</b>
<b>JMP P1INT_ISR</b>	
<b>CSEG AT 013BH</b>	<b>;P2INT_VECTOR</b>
<b>JMP P2INT_ISR</b>	
<b>CSEG AT 0143H</b>	<b>;P3INT_VECTOR</b>
<b>JMP P3INT_ISR</b>	
<b>CSEG AT 014BH</b>	<b>;P4INT_VECTOR</b>
<b>JMP P4INT_ISR</b>	
<b>CSEG AT 0153H</b>	<b>;P5INT_VECTOR</b>
<b>JMP P5INT_ISR</b>	
<b>CSEG AT 015BH</b>	<b>;P6INT_VECTOR</b>
<b>JMP P6INT_ISR</b>	
<b>CSEG AT 0163H</b>	<b>;P7INT_VECTOR</b>
<b>JMP P7INT_ISR</b>	
<b>CSEG AT 016BH</b>	<b>;P8INT_VECTOR</b>
<b>JMP P8INT_ISR</b>	
<b>CSEG AT 0173H</b>	<b>;P9INT_VECTOR</b>
<b>JMP P9INT_ISR</b>	
<b>CSEG AT 017BH</b>	<b>;M2MDMA_VECTOR</b>
<b>JMP M2MDMA_ISR</b>	
<b>CSEG AT 0183H</b>	<b>;ADCDMA_VECTOR</b>
<b>JMP ADCDMA_ISR</b>	
<b>CSEG AT 018BH</b>	<b>;SPIDMA_VECTOR</b>
<b>JMP SPIDMA_ISR</b>	
<b>CSEG AT 0193H</b>	<b>;U1TXDMA_VECTOR</b>
<b>JMP U1TXDMA_ISR</b>	
<b>CSEG AT 019BH</b>	<b>;U1RXDMA_VECTOR</b>
<b>JMP U1RXDMA_ISR</b>	
<b>CSEG AT 01A3H</b>	<b>;U2TXDMA_VECTOR</b>
<b>JMP U2TXDMA_ISR</b>	
<b>CSEG AT 01ABH</b>	<b>;U2RXDMA_VECTOR</b>
<b>JMP U2RXDMA_ISR</b>	
<b>CSEG AT 01B3H</b>	<b>;U3TXDMA_VECTOR</b>
<b>JMP U3TXDMA_ISR</b>	
<b>CSEG AT 01BBH</b>	<b>;U3RXDMA_VECTOR</b>
<b>JMP U3RXDMA_ISR</b>	
<b>CSEG AT 01C3H</b>	<b>;U4TXDMA_VECTOR</b>
<b>JMP U4TXDMA_ISR</b>	
<b>CSEG AT 01CBH</b>	<b>;U4RXDMA_VECTOR</b>
<b>JMP U4RXDMA_ISR</b>	
<b>CSEG AT 01D3H</b>	<b>;LCMDMA_VECTOR</b>
<b>JMP LCMDMA_ISR</b>	
<b>CSEG AT 01DBH</b>	<b>;LCMIF_VECTOR</b>
<b>JMP LCMIF_ISR</b>	

**POINT\_ISR:**

**P1INT\_ISR:**

**P2INT\_ISR:**

**P3INT\_ISR:**

**P4INT\_ISR:**

**P5INT\_ISR:**

**P6INT\_ISR:**

**P7INT\_ISR:**

**P8INT\_ISR:**

**P9INT\_ISR:**

**M2MDMA\_ISR:**

***ADCDMA\_ISR:******SPIDMA\_ISR:******U1TXDMA\_ISR:******U1RXDMA\_ISR:******U2TXDMA\_ISR:******U2RXDMA\_ISR:******U3TXDMA\_ISR:******U3RXDMA\_ISR:******U4TXDMA\_ISR:******U4RXDMA\_ISR:******LCMDMA\_ISR:******LCMIF\_ISR:*****JMP****006BH****END**

code testing method

According to the predefined DMA packet length (for example: 256 bytes), send a packet of data (254 bytes) through the serial port tool, and at the most

Then add 2 bytes of CCITT-CRC16 check code:



After the MCU receives the entire packet of data (256 bytes), it performs CRC16 check on the first 254 bytes of data, and the obtained check code is the same as the last two bytes.

Compare the bytes, if the values are equal, print "OK!" and the calculated check code, and then output the internal memory charged by the DMA space.

Allow.



If the checksum values are not equal, print "ERROR!" and the calculated checksum.

STCMCU

# 29 CAN bus

STC32G series MCU integrates two independent CAN bus functional units and supports CAN 2.0 protocol.

The main functions are as follows:

- ÿ Reception and transmission of standard frame and extended frame information
- ÿ 64 bytes of receive FIFO
- ÿ Single/dual acceptance filters in standard and extended formats
- ÿ Send and receive error counters
- ÿ Bus error analysis

## 29.1 CAN function pin switch

Symbol add	ress 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 selection bit

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

Symbol add	ress 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 selection bit

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

symbol	describe	address	Bit addresses and symbols								reset value			
			B7	B6	B5	B4	B3	B2	B1	B0				
CANICR	CANBUS Interrupt Control Register	F1H	PCAN1IF	PCAN2IF	PCAN1IE	PCAN2IE	PCAN1L	PCAN2L	PCAN1H	PCAN2H	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)

Symbol 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 group of CAN

1: Select the second group of CAN

## 29.2.2 CAN Bus Interrupt Control Register (CANICR)

Symbol 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, which needs 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 (highest)

CAN2IE: CAN2 bus interrupt enable control bit

0: Disable CAN2 interrupt

1: Enable CAN2 interrupt

CAN2IF: CAN2 bus interrupt request flag, which needs 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 (highest)

## 29.2.3 CAN Bus Address Register (CANAR)

Symbol address B7			B6	B5	B4	B3	B2	B1	B0
CANAR	7EFEBBH								

## 29.2.4 CAN Bus Data Register (CANDR)

Symbol address B7		B6	B5	B4	B3	B2	B1	B0
CANDR 7EFEBCH								

Reading and writing of CAN internal function registers requires indirect access through CANAR and CANDR

The method of reading the CAN internal function register:

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: need to read the value of CAN internal function register ISR

CANAR = 0x03; dat = //Write the address of the ISR to CANAR

CANDR; Method of //read CANDR to get the value of ISR

writing CAN internal function register:

1. Write the address of the CAN internal function register into the CAN bus address register CANAR
2. Write the value to be written into the CAN bus data register CANDR

For example: data 0x5a needs to be written into CAN internal function register TXBUFO

CANAR = 0x08; //Write the address of TXBUFO to CANAR

CANDR = 0x5a; //Write the value to be written 0x5a to CANDR

## 29.3 CAN internal function register

Note: The two groups of CANs are completely independent in hardware, and the internal registers of the two groups of CANs are also independent of each other, but the access addresses are the same. When you need to access the internal registers of different groups of CAN, you need to first select through AUXR2.CANSEL and then you can correctly accessed

	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	TXBUFO	TXBUF1	TXBUF2	TXBUF	RXBUFO	RXBUF1	RXBUF2	RXBUF3
00H	MR	CMR	SR	ISR	IMR	RMC	BTR0	BTR1

Symbol 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
ISR/IACK	0x03		ALI	EWI	EPI	RI	TI	BEI
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
BTR1	0x07	SAM TSG2.2		TSG2.1	TSG2.0	TSG1.3	TSG1.2	TSG1.1
TXBUFO	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)

Symbol address B7	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 for CAN modules

0: Disable LISTEN ONLY MODE

1: Enable LISTEN ONLY MODE

AFM: Receive filter selection for CAN module (see ACR register description)

0: The acceptance filter adopts the double filter setting

1: Accept filter with single filter setting

### 29.3.2 CAN Command Register (CMR)

Symbol address B7	B7	B6	B5	B4	B3	B2	B1	B0
CMR 0x01	-	-	-	-	-	TR	AT	-

TR: CAN module send request

0:

1: Initiate a frame transmission

AT: CAN module transmission terminated

0:

1: Terminate the current frame transmission

### 29.3.3 CAN Status Register (SR)

Symbol	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 without data frame

1: Receive BUFFER with data frame

DSO: Receive FIFO overflow loop flag

0: Receive FIFO has no overflow cycle generated

1: The receive FIFO has overflow loop generation

TBS: CAN module sends BUFFER status

0: The sending of BUFFER is prohibited, and the CPU cannot write to the BUFFER

1: Send BUFFER idle, CPU can write to BUFFER

RS: CAN module receiving status

0: CAN module receive idle

1: CAN module is receiving data frame

TS: CAN module sending status

0: CAN module send idle

1: CAN module is sending data frame

ES: CAN module error status

0: CAN module error register value does not reach 96

1: The CAN module has at least one error register value that reaches or exceeds 96

BS: CAN module BUS-OFF status

0: CAN module is not in BUS-OFF state

1: The CAN module is in the BUS-OFF state. When the CAN controller has more than 255 errors, it will trigger the BUS-OFF error.

error. Generally, the condition for BUS-OFF is that the CAN bus is disturbed by the surrounding environment, resulting in the data sent by the CAN transmitter to the bus.

It is judged as abnormal by the BUS bus, but the abnormal times exceed 255 times, the BUS bus is automatically set to the BUS-OFF state,

At this time, the bus is in a busy state, and data cannot be sent or received.

### 29.3.4 CAN Interrupt/Acknowledge Register (ISR/IACK)

symbol	address B7		B6	B5	B4	B3	B2	B1	B0
ISR/IACK	0x03	-	ALI	EWI	EPI	RI	TI	BEI	DOI

ALI: Arbitration Lost Interrupt

0:

1: Arbitration lost, write 1 to clear

EWI: Error warning interrupt

0:

1: This bit is set when the value of ES or BS in the SR register is 1. Write 1 to clear.

EPI: CAN Module Error Passive Interrupt

0:

1: This bit is set when the CAN error register operates on the passive error count value. Write 1 to clear.

RI: CAN module receive interrupt

0:

1: There is a data frame in the CAN module receiving BUFFER, the user needs to write 1 to RI to reduce the received message counter (RMC)

value.

TI: CAN module sends interrupt

0:

1: The CAN module data frame transmission is completed. The user needs to write 1 to TI to reset the write pointer of the transmit BUFFER.

BEI: CAN Module Bus Error Interrupt

0:

1: The CAN module has a bus error during reception or transmission

DOI: CAN module receive overflow interrupt

0:

1: CAN module receive FIFO overflow

### 29.3.5 CAN Interrupt Register (IMR)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
IMR	0x04	-	ALIM EWIM EPIM RIM TIM BEIM DOIIM				

ALIM: Arbitration Lost Interrupt

0: Arbitration lost interrupt mask

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 mask

1: CAN module passive error interrupt is enabled

RI: CAN module receive interrupt

0: CAN module receive interrupt mask

1: CAN module receive interrupt enabled

TI: CAN module sends interrupt

0: CAN module send interrupt mask

1: CAN module send interrupt on

BEI: CAN Module Bus Error Interrupt

0: CAN module bus error interrupt mask

1: CAN module bus error interrupt on

DOI: CAN module receive overflow interrupt

0: CAN module receive overflow interrupt mask

1: CAN module receive overflow interrupt on

### 29.3.6 CAN Frame Receive Counter (RMC)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
RMC	0x05	-	-	-	RMC[4;0]		

RMC: data frame received counter

### 29.3.7 CAN Bus Clock Register 0 (BTR0)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
BTR0 0x06	SJW[1:0]			BRP[5:0]			

BRP : CAN baud rate division factor

BRP= BTR0[5:0]+1; CAN module internal clock tq= tCLK\*BRP; tCLK = 1/ fXTAL (main frequency divided by 2)

SJW:: Resync Jump Width

SJW=SJW.1\*2+SJW.0

### 29.3.8 CAN Bus Clock Register 1 (BTR1)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
BTR1 0x07 SAM	TSG2[2:0]			TSG1[3:0]			

TSG1: Sync Sampling Segment 1

TSG2: Sync Sampling Segment 2

SAM: Number of bus level samples

0: The bus level is sampled once

1: The bus level is sampled 3 times

CAN baud rate=1/normal time bit

Normal time bits = (1+(TSG1+1)+(TSG2+1))\*tq

### 29.3.9 CAN bus data frame transmission buffer (TXBUFn)

Symbol 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, TXBUF0, TXBUF1, TXBUF2, TXBUF3, is written to BUFFER.

### 29.3.10 CAN bus data frame receive buffer (RXBUFn)

Symbol 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 the RXBUF3 register is written, the BUFFER pointer is automatically incremented by 1, RXBUF0, RXBUF1, RXBUF2, RXBUF3, is written to BUFFER. The data of a frame of CAN is up to 16 BYTE, so to receive a frame of data, you need to read RXBUF cyclically four times. The RXFIFO of the CAN module is a 64BYTE FIFO. When the amount of data is 1 BYTE, it can store up to

21 frames of data. When the data is 8 BYTE, it can store up to 5 frames of data. The number of received frames can be obtained by reading the RMC (RECEIVE MESSAGE COUNTER) register.

## CAN bus acceptance filter

With the help of the acceptance filter, the CAN controller can allow the RXFIFO to only receive data that match the identification code and the preset values in the acceptance filter.

The message acceptance filter is defined by the acceptance code register ACR and the acceptance mask register AMR.

### **29.3.11 CAN Bus Acceptance Code Register (ACRn)**

### **29.3.12 CAN Bus Acceptance Mask Register (AMRn)**

There are two filtering methods, which are selected by the AFM (MR.0) bit in the mode register: single filter mode (AFM bit is 1), double 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" (that is, set as irrelevant),

Whether the received corresponding frame ID bit is the same as the corresponding acceptance code bit will be indicated as reception; when the acceptance mask bit is "0" (ie, set to correlated), only if the corresponding frame ID bit and the corresponding acceptance code bit have the same value will be indicated as receiving. only in

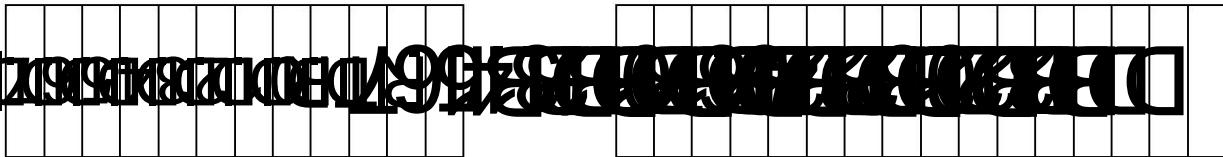
The CAN controller will only receive the message when all bits indicate reception.

### (1) Configuration of a single filter

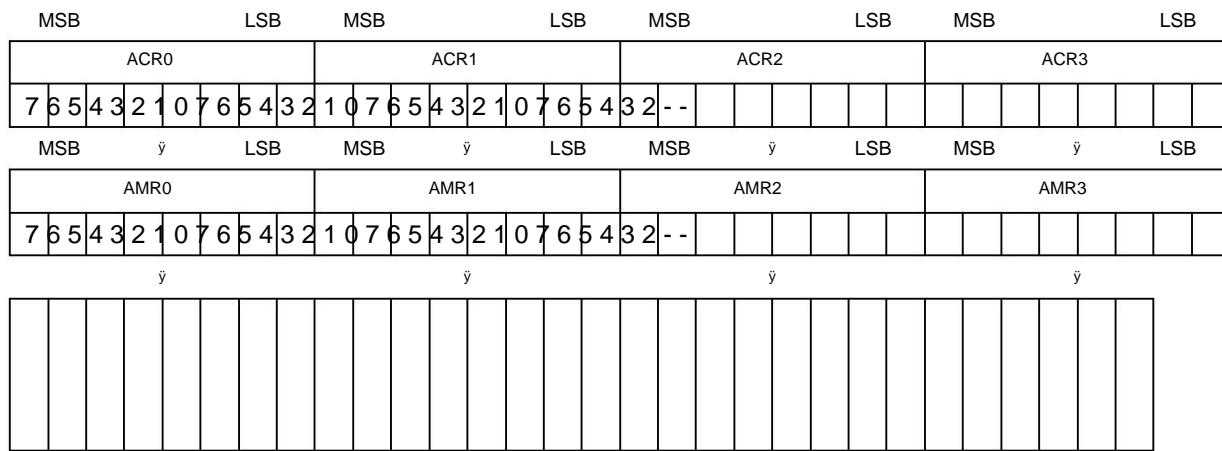
This filter configuration defines a long filter (4 bytes, 32 bits) consisting of 4 acceptance code registers and 4 acceptance mask register sets

The resulting acceptance filter, the bit correspondence between the filter byte and the information byte depends on the current received frame format. Receive CAN mark Quasi-frame single filter configuration: For standard frames, the 11-bit identifier, RTR bit, and the first two bytes of the data field participate in filtering; For wave data, the ACR bits corresponding to all bits whose AMR is 0 must be the same as the corresponding bits of the filtering data to be accepted; If there is no data byte due to setting the RTR=1 bit, or there is no or only one data byte due to the setting of the corresponding data length code Byte information, the message will also be received. For a successfully received message, all single bit comparisons in the filter must be It is "accept"; note that the lower four bits of AMR1 and ACR1 are not used. For compatibility with future products, these bits can be set by setting AMR1.3, AMR1.2, AMR1.1, and AMR1.0 are set to 1 as "does not affect".

Single filter configuration when receiving CAN standard frames



Single filter configuration when receiving CAN extended frames:



## (2) Configuration of double filters

This configuration can define two short filters consisting of 4 ACRs and 4 AMRs. The information on the bus only needs to pass through

Either filter is accepted. The bit correspondence between filter bytes and information bytes depends on the currently received frame format. catch

Configuration of dual filters for receiving CAN standard frames: If standard frame information is received, the two defined filters are different. the first

A filter consists of ACR0, ACR1, AMR0, AMR1 and the lower 4 bits of ACR3, AMR3, 11-bit identifier, RTR

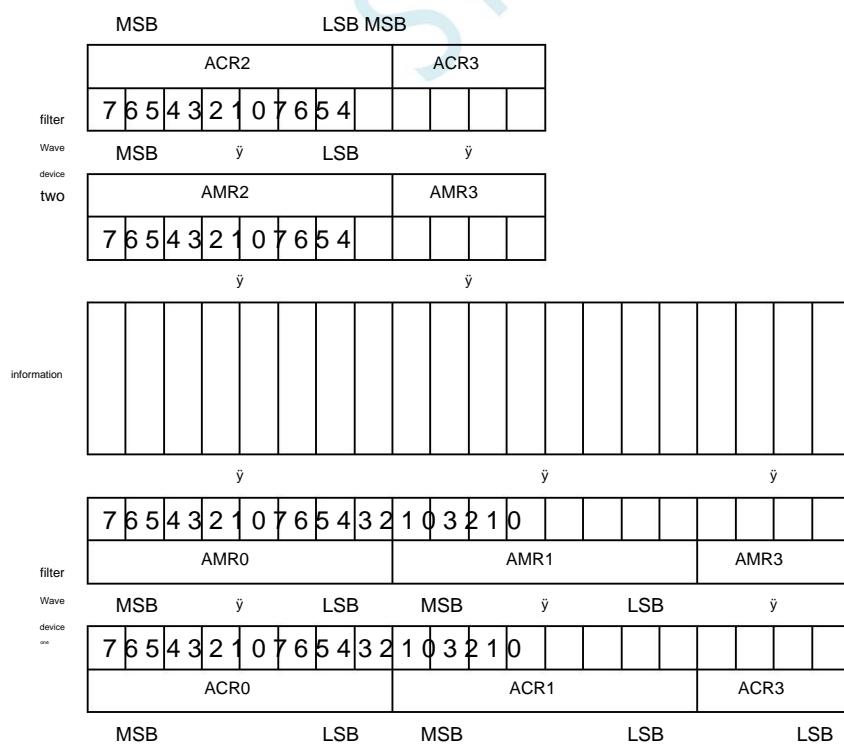
The first byte of the bit and data field participates in filtering; the second filter consists of ACR2, AMR2 and the high 4 bits of ACR3 and AMR3,

The 11-bit identifier and RTR bits are involved in filtering. In order to successfully receive information, at least one filter should be used when comparing all individual bits to indicate acceptance. The RTR bit is "1" or the data length code is "0", indicating that no data bytes exist;

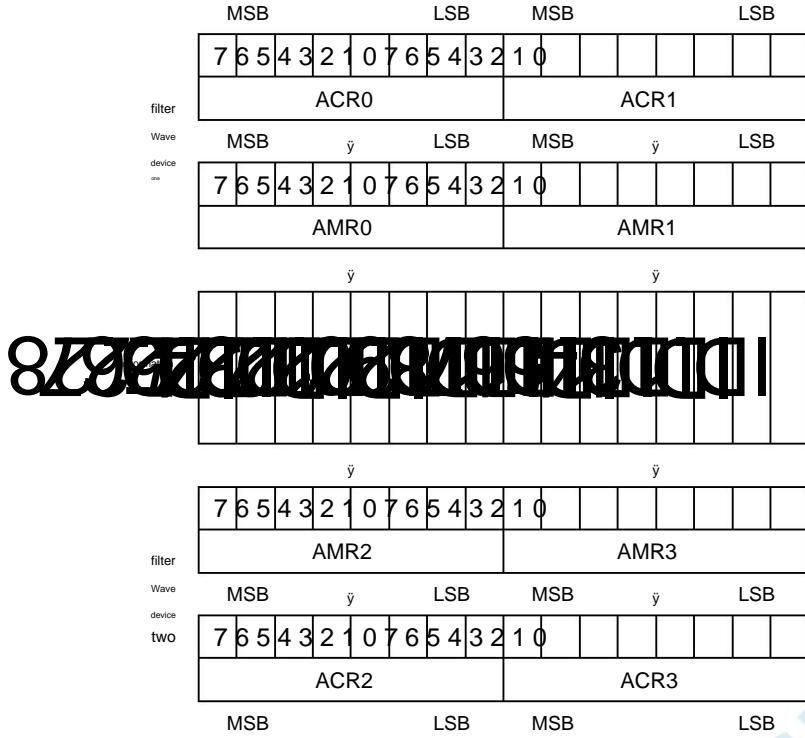
The parts are represented as received, and the information can pass through filter 1. If there are no data bytes to request filtering from the filter, AMR1 and

The lower 4 bits of AMR3 must be set to "1", ie "don't affect". At this point, the identification of both filters is to verify that the RTR bit is included the entire standard identification code included.

Dual filter configuration when receiving CAN standard frames:



Dual filter configuration when receiving CAN extended frames



### 29.3.13 CAN BUS ERROR INFORMATION REGISTER (ECC)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
ECC 0x18	RXWRN	TXWRN	EDIR	ACKER	FRMER	CRCER	SFTFER	BER	

RXWRN: This bit is set when RXERR is greater than or equal to 96.

**TXWRN:** This bit is set when TXERR is greater than or equal to 96.

EDIR: Transmission in wrong direction. 0: An error occurred while sending. 1: An error occurred while receiving.

ACKER: ACK error.

FRMER: Frame format error.

CRCER: CRC error.

#### STFFR: Bit stuffing error

BER: Bit error

### **29.3.14 CAN Bus Receive Error Counter (RXERR)**

Symbol	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. When the BUS-OFF event occurs, the counter value is changed by

## Hardware clear

### 29.3.15 CAN Bus Transmit Error Counter (TXERR)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
TXERR	0x1A					TXERR			

**TXERR**: The transmit error count register reflects the current value of the transmit error counter. In working mode, this register is read-only and reset by hardware.

The post register is initialized to 0. The error counter is initialized to 127 after the BUS-OFF time has occurred to count the maximum value defined by the bus. Small time (128 bus idle signals). The read transmit error counter during this time will reflect the status information of the bus shutdown recovery.

### 29.3.16 CAN Bus Arbitration Loss Register (ALC)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
ALC	0x1B	-	-	-		ALC[4:0]			

ALC: This register contains information on where arbitration was lost. After the current arbitration lost interrupt has been serviced (acknowledged), the value will be

Updated when a quorum is lost. The specific values correspond to the following table:

ALC[4:0]	Numerical value	describe
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	twenty one	Arbit lost in ID9*
10110	twenty two	Arbit lost in ID8*
10111	twenty three	Arbit lost in ID7*
11000	twenty four	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 Example Program

### 29.4.1 CAN BUS FRAME FORMAT

#### CAN2.0B standard frame

CAN standard frame information is 11 bytes, including two parts: information and data parts. The first 3 bytes are the information part.

Location	B7	B6	B5	B4	B3	B2	B1	B0
byte 01	FF	RTR	-	-	DLC (Data Length)			
byte 02					CAN ID[10:3]			
byte 03		CAN ID[2:0]		-	-	-	-	-
byte 04					data 1			
byte 05					data 2			
byte 06					data 3			
Byte 07					data 4			
Byte 08					Data 5			
Byte 09					data 6			
Byte 10					data 7			
Byte 11					data 8			

Byte 1 is frame information. The 7th bit (FF) represents the frame format, in the standard frame, FF=0; the 6th bit (RTR) represents the frame type,

RTR=0 means data frame, RTR=1 means remote frame; DLC means actual data length in data frame.

Bytes 2 and 3 are the message identification code, 11 bits are valid.

Bytes 4 to 11 are the actual data of the data frame, and are invalid in the remote frame.

#### CAN2.0B extended frame

CAN extended frame information is 13 bytes, including two parts, information and data parts. The first 5 bytes are the information part.

Location	B7	B6	B5	B4	B3	B2	B1	B0
byte 01	FF	RTR	-	-	DLC (Data Length)			
Byte 02					CAN ID[28:21]			
Byte 03					CAN ID[20:13]			
Byte 04					CAN ID[12:5]			
Byte 05		CAN ID[4:0]				-	-	-
byte 06					data 1			
byte 07					data 2			
byte 08					data 3			
byte 09					data 4			
byte 10					Data 5			
Byte 11					data 6			
Byte 12					data 7			
Byte 13					data 8			

Byte 1 is frame information. The 7th bit (FF) represents the frame format, in the extended frame, FF=1; the 6th bit (RTR) represents the frame type,

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 the upper 29 bits are valid.

Bytes 6 to 13 are the actual data of the data frame, and are invalid for remote frames.

### 29.4.2 CAN Bus Standard Frame Transceiver Example

---

```
//The test frequency is 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

typedef unsigned char u8;
typedef unsigned int u16;
typedef unsigned long u32;

#define MAIN_Fosc      24000000UL

/****** user-defined macro *****/
//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           //Resync jump width

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

u16 CAN_ID;
u8 RX_BUF[8];
u8 TX_BUF[8];

void CANInit();
void CanSendMsg(u16 canid, u8 *pdat);

void main(void)
{
    EAXF = 1; //Enable XFR
    CKCON = 0x00; //access to set external data bus speed to fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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

    CANInit();

    EA = 1; //Turn on total 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)                                // click it once P32 Press the button to send a frame of fixed data
    {
        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;
}

```

```

16 CanID;
8 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(16 canid, u8 *pdat)
{
    16 CanID;

    CanID = canid << 5;
    CanWriteReg(TX_BUF0,0x08);                                // Standard frage 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 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);                //take over CAN bus data

        CanSendMsg(CAN_ID+1,RX_BUF);                //CAN ID Add 1 and send as is CAN bus data
    }
}

```

---

### 29.4.3 CAN Bus Extended Frame Transceiver Example

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                //For header files, see Download Software
#include "intrins.h"

typedef unsigned char u8;
typedef unsigned int u16;
typedef unsigned long u32;

#define MAIN_Fosc      24000000UL

/********************************************* user-defined macro *****/
//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           //Resync jump width

/********************************************* *****/
u32 CAN_ID;
u8 RX_BUF[8];
u8 TX_BUF[8];

void CANInit();

```

```

void CanSendMsg(u32 canid, u8 *pdat);

void main(void)
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                            //assign to      CPU The speed of executing the program is set to the fastest

    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

    CANInit();

    EA = 1;                                    //Turn on total 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)                                //click it once P32 Press the button to send a frame of fixed data
        {
            CanSendMsg(CAN_ID, TX_BUF);
            while(!P32);                         //prevent retransmission
        }
    }
}

8 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_BUFO);
}

```

```

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, 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 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;
    AUXR2 |= 0x02;                                     //CAN interrupt enable
                                                       //CAN module is enabled
}

void CANBUS_Interrupt(void) interrupt 28
{
    u8 isr;

    isr = CanReadReg(ISR);
    if((isr & 0x04) == 0x04)
    {
        CANAR = 0x03;                                 //CLR FLAG
        CANDR = 0x04;

    }
    if((isr & 0x08) == 0x08)
    {
        CANAR = 0x03;                                 //CLR FLAG
        CANDR = 0x08;

        CAN_ID = CanReadMsg(RX_BUF);                  //take over CAN bus data

        CanSendMsg(CAN_ID+1,RX_BUF);                 //CANID Add 1 and send as is CAN bus data
    }
}

```

## 29.4.4 CAN Bus Standard Frame Transceiver Test (Assembly)

assembly code

;The test frequency is **11.0592MHz**

,\*\*\*\*\* Function Description \*\*\*\*\*

,This routine is based on **STC32G** Write and test the experimental box of the main control chip.

;use **Keil C251** translator, **Memory Model** Recommended settings **XSmall** mode, the default variable is defined in **edata**, single-clock access access speed is fast.

;**edata** It is recommended to keep **1K** It is used for the stack. When the space is not enough, large arrays and uncommon variables can be added.**xdata** keyword defined to **xdata** space.

;**CAN** Bus Transceiver Test Case

;**DCAN** is a support **CAN2.0B** The functional unit of the protocol.

,Send one frame per second standard **CAN** bus data .

,After receiving a standard frame, replace the original **CAN ID** with the data sent

;Default baud rate **500KHz**, Users can modify.

,select clock while downloading **24MHZ** ( Users can modify the frequency by themselves.)

,\*\*\*\*\*/

**\$include (STC32G.INC)**

,\*\*\*\*\* user-defined macro \*\*\*\*\*

**Fosc\_KHZ EQU 24000** ;24000KHZ

**STACK\_POINTER EQU 0D0H** ;stack start address

**Timer0\_Reload EQU (65536 - Fosc\_KHZ)** ; Timer 0 Interrupt frequency 1000 sub-second

;CAN bus baud rate=Fcik/((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** ;Resync jump width: 00H/040H/080H/0C0H

;bus baud rate **100KHz** The above is set to

**SAM EQU 00H**

0; 100KHz The following settings 1  
;are the number of bus level samples:  
**00H**: sampling 1 Second, ; **080H**: sampling 3 Second-rate

,\*\*\*\*\* IO mouth definition \*\*\*\*\*

,\*\*\*\*\* local variable declaration \*\*\*\*\*

**Flag0 DATA 20H**

**B\_1ms BIT Flag0.0**

; 1ms logo

<i>B_CanRead BIT</i>	<i>Flag0.1</i>	<i>; CAN</i> data received flag
<i>msecond</i>	<i>DATA</i>	<i>21H</i>
<i>CAN_ID</i>	<i>DATA</i>	<i>23H</i>
		<i>;</i>
<i>RX_BUF</i>	<i>DATA</i>	<i>30H</i>
<i>TX_BUF</i>	<i>DATA</i>	<i>38H</i>
		<i>;</i>
<i>TMP_BUF DATA</i>	<i>DATA</i>	<i>40H</i>

**ORG** *0000H* ;Program reset entry, the compiler automatically defines to **OFF0000H** address  
**LJMP** *F\_Main*

```
ORG      000BH           ;1 Timer0 interrupt  
LJMP    F_Timer0_Interrupt  
  
ORG      00E3H           ;28 CAN interrupt  
LJMP    F_CAN_Interrupt
```

main program		/* The compiler automatically defines to <b>OFF0200H</b> address
<b>ORG</b>	<b>0200H</b>	
<b>MOV</b>	<b>WTST, #00H</b>	;Set the program command delay parameter,
<b>ORL</b>	<b>P_SW2, #080H</b>	;Assign value <b>0</b> to <b>CPU</b> The speed of executing the command is set to the fastest
<b>MOV</b>	<b>CKCON, #00H</b>	;enable access to <b>XER</b>
		;set external data bus speed to fastest
<b>MOV</b>	<b>P0M1, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P0M0, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P1M1, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P1M0, #00H</b>	
<b>MOV</b>	<b>P2M1, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P2M0, #00H</b>	
<b>MOV</b>	<b>P3M1, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P3M0, #00H</b>	
<b>MOV</b>	<b>P4M1, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P4M0, #00H</b>	
<b>MOV</b>	<b>P5M1, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P5M0, #00H</b>	
<b>MOV</b>	<b>P6M1, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P6M0, #00H</b>	
<b>MOV</b>	<b>P7M1, #00H</b>	;set quasi-bidirectional port
<b>MOV</b>	<b>P7M0, #00H</b>	
<b>MOV</b>	<b>SP, #STACK_POINTER</b>	
<b>MOV</b>	<b>PSW, #0</b>	;choose <b>0</b> group <b>R0~R7</b>
<b>USING</b>	<b>0</b>	;the choice <b>0</b> group <b>R0~R7</b>

## **user initialization program**

<b>CLR</b>	<b>TR0</b>	
<b>ORL</b>	<b>AUXR, #(1 SHL 7)</b>	; Timer0_1T();
<b>ANL</b>	<b>TMOD, #NOT 04H</b>	; Timer0_AsTimer();
<b>ANL</b>	<b>TMOD, #NOT 03H</b>	; Timer0_16bitAutoReload();
<b>MOV</b>	<b>TH0, #Timer0_Reload / 256</b>	;Timer0_Load(Timer0_Reload);
<b>MOV</b>	<b>TL0, #Timer0_Reload MOD256</b>	
<b>SETB</b>	<b>ET0</b>	;Timer0_InterruptEnable();
<b>SETB</b>	<b>TR0</b>	;Timer0_Run();
<b>LCALL</b>	<b>F_CAN_Init</b>	
<b>SETB</b>	<b>EA</b>	;Turn on total interrupt

---

<b>MOV</b>	<b>WR4, #0345H</b>	
<b>MOV</b>	<b>CAN_ID, WR4</b>	;set default CAN ID
<b>MOV</b>	<b>TX_BUF+0, #11H</b>	;settings sent by defauCAN data
<b>MOV</b>	<b>TX_BUF+1, #22H</b>	
<b>MOV</b>	<b>TX_BUF+2, #33H</b>	
<b>MOV</b>	<b>TX_BUF+3, #44H</b>	
<b>MOV</b>	<b>TX_BUF+4, #55H</b>	
<b>MOV</b>	<b>TX_BUF+5, #66H</b>	
<b>MOV</b>	<b>TX_BUF+6, #77H</b>	
<b>MOV</b>	<b>TX_BUF+7, #88H</b>	

---

===== main loop =====

**L\_Main\_Loop:**

<b>JNB</b>	<b>B_1ms, L_Main_Loop</b>	;1ms did not arrive
<b>CLR</b>	<b>B_1ms</b>	

---

===== detect 1000ms whether to =====

<b>MOV</b>	<b>WR6, msecnd</b>	
<b>INC</b>	<b>WR6, #1</b>	;msecnd + 1
<b>MOV</b>	<b>msecnd, WR6</b>	
<b>CMP</b>	<b>WR6, #1000</b>	
<b>JC</b>	<b>L_Quit_Check_1000ms</b>	;if(msecnd < 1000), jmp
<b>MOV</b>	<b>WR6, #0</b>	
<b>MOV</b>	<b>msecnd, WR6</b>	;msecnd = 0

---

===== 1000ms arrive, deal with RTC =====

<b>MOV</b>	<b>WR6, #0</b>	
<b>MOV</b>	<b>msecnd, WR6</b>	;msecnd = 0
<b>MOV</b>	<b>A, #SR</b>	
<b>LCALL</b>	<b>F_CAN_ReadReg</b>	
<b>JB</b>	<b>ACC.0, L_CAN_BUSOFF</b>	
<b>LCALL</b>	<b>F_CAN_SendMsg</b>	
<b>LJMP</b>	<b>L_Quit_Check_1000ms</b>	

---

**L\_CAN\_BUSOFF:**

<b>MOV</b>	<b>A, #MR</b>	
<b>MOV</b>	<b>WR6, #WORD0 CANAR</b>	
<b>MOV</b>	<b>WR4, #WORD2 CANAR</b>	
<b>MOV</b>	<b>@DR4, R11</b>	
<b>MOV</b>	<b>WR6, #WORD0 CANDR</b>	
<b>MOV</b>	<b>WR4, #WORD2 CANDR</b>	
<b>MOV</b>	<b>R11, @DR4</b>	
<b>ANL</b>	<b>A, #NOT 04H</b>	;clear Reset Mode, fromBUS-OFF status exit
<b>MOV</b>	<b>@DR4, R11</b>	

---

**L\_Quit\_Check\_1000ms:**

```

JNB      B_CanRead, L_Main_Loop      ;not received CAN bus data
CLR      B_CanRead
LCALL   F_CAN_ReadMsg               ;take over CAN data, replace the original CAN_ID with the data sent
=====
LJMP   L_Main_Loop

```

,\*\*\*\*\*\*/

```

; function:  F_CAN_ReadReg
;          : CAN Function register read function
;          : A: Addr.
;          : A: Data.
; description parameter return value: V1.0, 2022-04-06
=====
```

```

F_CAN_ReadReg:
    MOV      WR6, #WORD0 CANAR
    MOV      WR4, #WORD2 CANAR
    MOV      @DR4, R11

    MOV      WR6, #WORD0 CANDR
    MOV      WR4, #WORD2 CANDR
    MOV      R11,@DR4
    RET

```

```

; function:  F_CAN_WriteReg
;          : CAN Function Register Configuration Function
;          : A: Addr, B: Data.
;          : none.
; description parameter return value: V1.0, 2022-04-06
=====
```

```

F_CAN_WriteReg:
    MOV      WR6, #WORD0 CANAR
    MOV      WR4, #WORD2 CANAR
    MOV      @DR4, R11

    MOV      WR6, #WORD0 CANDR
    MOV      WR4, #WORD2 CANDR
    MOV      @DR4, R10
    RET

```

```

; function:  F_CAN_ReadMsg
;          : CAN receive data function
;          : none.
;          : none.
; description parameter return value: V1.0, 2022-04-06
=====
```

```

F_CAN_ReadMsg:
    MOV      A, #RX_BUFO
    CALL   F_CAN_ReadReg
    MOV      TMP_BUF+0, A

    MOV      A, #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;
MOV      R6, TMP_BUF+1
MOV      R7, TMP_BUF+2
SRL     WR6
SRL     WR6
SRL     WR6

```

```

SRL      WR6
SRL      WR6
ANL      WR6, #07FFH
MOV      CAN_ID, WR6
;Parse CAN ID

RET

```

---

```

; function: F_CAN_SendMsg
;   : CAN send data function
;   : none.
;   : none.
; description parameter return value: 2022-04-06

```

---

**F\_CAN\_SendMsg:**

```

MOV      A, #TX_BUFO
MOV      B, #08H
;Standard Frame(0)/Extended Frame(1)
;Data Frame(0)/Multiframe Data(1)
;bit7: ;bit6: ;bit3~bit0: (DLC)

CALL    F_CAN_WriteReg

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_BUFO
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_BUFO
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
CALL    F_CAN_WriteReg
;Initiate a frame transfer
RET

```

---

```

; function: F_CAN_Init
;           CAN initializer
;           none
;           none.
;
```

;  
description parameter return type: V1.0 2022-04-06

---

#### F\_CAN\_Init:

```

ORL      AUXR2, #02H      ;CAN1 module enable
ANL      AUXR2, #NOT 08H   ;choose CAN1 module
MOV      P_SW1, #0

MOV      A, #MR
MOV      B, #04H
CALL   F_CAN_WriteReg
;Enable Reset Mode

```

```

;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
MOV      A, #ACR0
MOV      B, #00H
CALL   F_CAN_WriteReg
MOV      A, #ACR1
MOV      B, #00H
CALL   F_CAN_WriteReg
MOV      A, #ACR2
MOV      B, #00H
CALL   F_CAN_WriteReg
MOV      A, #ACR3
MOV      B, #00H
CALL   F_CAN_WriteReg

```

```

;Bus Acceptance Mask Register
MOV      A, #AMR0
MOV      B, #0FFH
CALL   F_CAN_WriteReg
MOV      A, #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           ; quit Reset Mode
CALL    F_CAN_WriteReg

MOV      CANICR, #02H      ; CAN interrupt enable
RET

```

\*\*\*\*\*  
; interrupt function

**F\_Timer0\_Interrupt:**

PUSH	PSW	;Timer0 1ms interrupt function
PUSH	ACC	;PSW push onto the stack
SETB	B_1ms	;ACC push onto the stack
POP	ACC	;1ms logo
POP	PSW	;ACC pop
RETI		;PSW pop

=====

**F\_CAN\_Interrupt:** ;CAN interrupt function

PUSH	PSW	;PSW push onto the stack
PUSH	ACC	;ACC push onto the stack
PUSH	R4	;R4 push onto the stack
PUSH	R5	;R5 push onto the stack
PUSH	R6	;R6 push onto the stack
PUSH	R7	;R7 push onto 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, write 1 clear

**ISRTTEST1:** JB ACC.0,CAN\_D0IFF

**ISRTTEST2:** JB ACC.1,CAN\_BEIFF

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 pop
POP	R6	;R6 pop
POP	R5	;R5 pop
POP	R4	;R4 pop
POP	ACC	;ACC pop
POP	PSW	;PSW pop
<b>RETI</b>		

***CAN\_DOIF:***

;	ORL	A,#01H	;clear flag, write 1 clear
;	MOV	@DR4, R11	
;	JMP	<b>ISRTEST1</b>	

***CAN\_BEIIF:***

;	ORL	A,#02H	;clear flag, write 1 clear
;	MOV	@DR4, R11	
;	JMP	<b>ISRTEST2</b>	

***CAN\_TIIF:***

;	ORL	A,#04H	;clear flag, write 1 clear
;	MOV	@DR4, R11	
;	JMP	<b>ISRTEST3</b>	

***CAN\_RIIF:***

;	ORL	A,#08H	;clear flag, write 1 clear
;	MOV	@DR4, R11	
;	SETB	B_CanRead	
;	JMP	<b>ISRTEST4</b>	

***CAN\_EPIIF:***

;	ORL	A,#010H ;	clear flag, write 1 clear
;	MOV	@DR4, R11	
;	JMP	<b>ISRTEST5</b>	

***CAN\_EWIIF:***

;	ORL	A,#020H	;clear flag, write 1 clear
;	MOV	@DR4, R11	
;	JMP	<b>ISRTEST6</b>	

***CAN\_ALIIF:***

;	ORL	A,#040H	;clear flag, write 1 clear
;	MOV	@DR4, R11	
;	JMP	<b>ISREXIT</b>	

---

**END**

# 30 LIN bus

The STC32G series microcontroller integrates the LIN bus functional unit and supports the LIN2.1 and LIN1.3 protocols.

The main functions are as follows:

- ÿ Frame header automatic processing
- ÿ Can switch between master and slave modes
- ÿ Timeout detection
- ÿ Error analysis

## 30.1 LIN function pin switch

Symbol 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 selection bit

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

symbol	describe	address	Bit addresses and symbols								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	xxx,0000

### 30.2.1 Auxiliary Register 2 (AUXR2)

Symbol 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)

Symbol 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, cleared by hardware (cleared by reading LSR).

PLINH, PLINL: LIN interrupt priority control bits

PLINH	PLINL	priority
0	0	0 (minimum)
0	1	1
1	0	2
1	1	3 (highest)

### 30.2.3 LIN Bus Address Register (LINAR)

Symbol address	B7	B6	B5	B4	B3	B2	B1	B0
LINAR	FAH							

### 30.2.4 LIN Bus Data Register (LINDR)

Symbol address	B7	B6	B5	B4	B3	B2	B1	B0
LINDR	FBH							

Reading and writing to LIN internal function registers requires indirect access through LINAR and LINDR

The method of [reading the LIN internal function register](#):

1. Write the address of the LIN internal function register into the LIN bus address register LINAR
2. Read the LIN bus data register LINDR

For example: need to read the value of LIN internal function register LSR

```
LINAR = 0x05; // //Write the address of LSR to LINAR
dat = LINDR; The read CANDR to get the value of LSR
```

method of [writing the LIN internal function register](#):

1. Write the address of the LIN internal function register into the LIN bus address register LINAR
2. Write the value to be written into the LIN bus data register LINDR

For example: data 0x5a needs to be written into the LIN internal function register LBUF

```
LINAR = 0x00; // //Write the address of LBUF to LINAR
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

Symbol 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	N13		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)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LBUF	0x00								

LBUF is the data interface of the internal data FIFO of the LIN module.

### 30.3.2 LIN Data Address Register (LSEL)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LSEL	0x01	AINC	-	-	-	INDEX[3:0]			

AINC address auto increment

0: The data address written to the LIN internal FIFO is determined by INDEX.

1: The data address is incremented automatically. Each time LBUF is operated, the data address of the LIN internal FIFO is automatically incremented by one.

### 30.3.3 LIN Frame ID Register (LID)

Symbol	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 number 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)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LER	0x03	-	OV	SYN	TOV	CHK	PER	BIT	FER

OV: over run error. When LIN is running a command (RDY is low), the user sends a new command to LIN.

SYN: Only in slave mode, an error occurs when LIN receives frame synchronization.

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, indicating that the data bit sent by LIN is inconsistent with the state monitored by the bus.

FER: Framing error, the received data does not contain a valid stop bit.

[Read clear error register.](#)

### 30.3.5 LIN Interrupt Enable Register (LIE)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LIE	0x04	-	-	-	-	ABORTE	ERRE	RDYE	LIDE

ABORTE: Enable abort interrupt.

ERRE: Enable error interrupt.

RDYE: Enable Ready interrupt.

LIDE: Enable head interrupt.

### 30.3.6 LIN Status Register (Read Only Register) (LSR)

Symbol	address B7		B6	B5	B4	B3	B2	B1	B0
LSR	0x05		LOG SIZE[3:0]			ABORT	ERR	RDY	LID

LOG SIZE: In LOG mode, the detected data length.

ABORT: Abort command is in progress.

ERR: Error status.

RDY: Ready state, can execute new commands.

LID: The correct header was received.

[Bit0-Bit3 are cleared after being read.](#)

### 30.3.7 LIN Control Register (Write-Only Register) (LCR)

Symbol	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: Main mode.

LIN13: Checksum selection.

0: Enhanced checksum, LIN2.1 protocol.

1: Normal checksum, LIN1.3 protocol.

SIZE[3:0]: data length

Set Length 0	Byte 1 Byte 2	set up	length
0000	Byte 3	1000	8 bytes
0001	Byte 4	1001	reserve
0010	Byte 5	1010	reserve
0011	Byte 6	1011	reserve
0100	Byte 7	1100	reserve
0101	Byte	1101	reserve
0110		1110	LOG MODE
0111		1111	The data length is determined by LID[5:4]

CMD[1:0]: LIN command.

- 00: Abort command.
- 01: Main mode Send header command.
- 10: TX response.
- 11: RX response.

### 30.3.8 LIN Baud Rate Register (DLL/DLH)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
DLL	0x06						DLL
DLH	0x07	SYNC					DLH

DLL and DLH determine the baud rate of the LIN module. baud rate = SYSclk/16/DL.

SYNC: Synchronous mode, only useful in slave mode.

### 30.3.9 LIN Header Delay Count Register (HDRL/HDRH)

Symbol 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 Divider Register (HDP)

Symbol address B7	B6	B5	B4	B3	B2	B1	B0
HDP	0x0A						HDP

HDR and HDP jointly define the delay time of a frame header, which is used to define the command to send the frame header and the LIN mode after the software is configured.

The time delay that the block actually sends the head frame header.

## 30.4 Example programs

### 30.4.1 LIN bus master transceiver example

---

```
//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#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: sleep; 1 : Work

#define LIN_MODE       1 // 0: LIN2.1; 1: LIN1.3
#define FRAME_LEN      8 // Data length 8 byte

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)
{
    EA邢R = 1; // enable access XFR
    CKCON = 0x00; // Set the external data bus speed to the fastest
    WTST = 0x00; // Set the program code to wait for parameters,
                  // assign to CPU The speed of executing the program is set to the fastest

    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; // LIN_TX, LIN_RX Foot open with internal pull-up
    P0PU = 0x0c;
    P_SW2 &= ~0x80;

    Lin_ID = 0x32;
}
```

```

LinInit();
EA = 1; // Turn on total 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-shake
                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 anti-shake
                P47 = ~P47;
                Key2_Flag = 1;
                LinSendHeader(0x13); // Host sends Header , the response data is sent by the slave with a specific identifier, and it is assembled 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);
    for(i=0;i<FRAME_LEN;i++)
    {
        pdat[i] = LinReadReg(LBUF);
    }
}

void LinSetMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL,0x80);
    for(i=0;i<FRAME_LEN;i++)
    {
        LinWriteReg(LBUF,pdat[i]);
    }
}

void LinSetID(u8 lid)
{
    LinWriteReg(LID, lid); //set bus ID
}

u8 GetLinError(void)
{
    u8sta;
    sta = LinReadReg(LER);
    return sta; //read clear error register
}

u8 WaitLinReady(void)
{
    u8 lsr;
    do{
        lsr = LinReadReg(LSR);
    }
}

```

```

}while(!(Isr & 0x02));
    return Isr;
}

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 bus ID
    LinSetMsg(pdat);

    SendHeadCmd();                                         //Master Send Seader
    WaitLinReady();                                         //Mode W ready
    GetLinError();                                         //read clear error register

    SendDatCmd();                                         //Send Data
    WaitLinReady();                                         //wait ready
    GetLinError();                                         //read clear error register
}

void LinSendHeader(u8 lid)
{
    LinSetID(lid);                                         //set send Response slave bus ID
    SendHeadCmd();                                         //Master send header
    WaitLinReady();                                         //Mode W ready
    GetLinError();                                         //read clear error register

    ResponseRxCmd();                                     //RX response
    WaitLinReady();                                         //ready state
}

```

```

GetLinError();                                //read clear error register

LinReadMsg(TX_BUF);                         //take over Lin Reply data sent by bus slave
}

void LinSetBaudrate(u16 brt)
{
    u16tmp;
    tmp = (MAIN_Fosc >> 4) / brt;
    LinWriteReg(DLH,(u8)(tmp>>8));           //set baud rate
    LinWriteReg(DLL,(u8)tmp);

}

void LinSetHeadDelay(u8 base_ms, u8 prescaler)
{
    u16tmp;
    tmp = (MAIN_Fosc * base_ms) / 1000;          //Notice base_ms The value range, the calculation result should not exceed 16 Bit cap
    LinWriteReg(HDRH,(u8)(tmp>>8));
    LinWriteReg(HDRL,(u8)tmp);                   //Set frame header delay count

    LinWriteReg(HDP,prescaler);                  // Set frame header delay frequency division
}

void LinInit()
{
    P_SW1 = 0x00;
    LINICR = 0x02;
    AUXR2 |= 0x01;

    GetLinError();
    LinWriteReg(LIE,0x00);
    LinSetBaudrate(9600);
    LinSetHeadDelay(0x01,0x00);

}

```



//Select P0.2, P0.3  
 //LIN module interrupt enable  
 //LIN module is enabled  
  
 //read clear error register  
 //LIE Interrupt Enable Register  
 //set baud rate  
 //Set frame header delay

### 30.4.2 LIN bus slave transceiver example

---

```

//The test frequency is 11.0592MHz

#ifndef include "stc8h.h"
#include "stc32g.h"                                     // For header files, see Download Software
#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 sleep; 1 work

#define LIN_MODE       1                               // 0: LIN2.1; 1: LIN1.3
#define FRAME_LEN      8                               // Data length byte

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 XFR
    CKCON = 0x00;
    WTST = 0x00;

    P0M1 = 0; P0M0 = 0;
    P1M1 = 0; P1M0 = 0;
    P2M1 = 0; P2M0 = 0;
    P3M1 = 0; P3M0 = 0;
    P4M1 = 0; P4M0 = 0;
    P5M1 = 0; P5M0 = 0;
    P6M1 = 0; P6M0 = 0;
    P7M1 = 0; P7M0 = 0;

    P_SW2 |= 0x80;
    P0PU = 0xc;
    P_SW2 &= ~0x80;

    Lin_ID = 0x32;
    LinInit();
    EA = 1;

    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();            //ready state
            GetLinError();             //read clear error register

            LinReadMsg(TX_BUF);       //take over 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);                                // The address is incremented, starting from 0

    for(i=0;i<FRAME_LEN;i++)
    {
        pdat[i] = LinReadReg(LBUF);
    }
}

void LinSetMsg(u8 *pdat)
{
    u8 i;

    LinWriteReg(LSEL,0x80);                                // The address is incremented, starting from 0
    for(i=0;i<FRAME_LEN;i++)
    {
        LinWriteReg(LBUF,pdat[i]);
    }
}

void LinSetID(u8 lid)
{
    LinWriteReg(LID, lid);                                //set bus ID
}

u8 GetLinError(void)
{
    u8sta;
    sta = LinReadReg(LER);                                //read clear error register
    return sta;
}

u8 WaitLinReady(void)
{
    u8 lsr;
    do{
        lsr = LinReadReg(LSR);
    }while(!(lsr & 0x02));                                //judge 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();                                      //readystate
    GetLinError();                                       //read clear error register
}

void LinSetBaudrate(u16 brt)
{
    u16tmp;
    tmp = (MAIN_Fosc >> 4) / brt;
    LinWriteReg(DLH,(u8)(tmp>>8));
    LinWriteReg(DLL,(u8)tmp);
}

void LinSetHeadDelay(u8 base_ms, u8 prescaler)
{
    u16tmp;
    tmp = (MAIN_Fosc * base_ms) / 1000;                  //Noticebase_msThe value range, the calculation result should not exceed 16 Bit cap
    LinWriteReg(HDRH,(u8)(tmp>>8));                   //Set frame header delay count
    LinWriteReg(HDRL,(u8)tmp);

    LinWriteReg(HDP,prescaler);                          //Set 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 clear error register
LinWriteReg(LIE,0x0F);                      //LIR Interrupt Enable Register
LinSetBaudrate(9600);                        //set baud rate
LinSetHeadDelay(0x01,0x00);                  //Set frame header delay
}

void LinBUS_Interrupt(void) interrupt LIN_VECTOR
{
    u8 isr;

    isr = LinReadReg(LSR);                      //receivedHeader, in Ready state
    if((isr & 0x03) == 0x03)
    {
        isr = LinReadReg(LER);                    //no errors were generated
        if(isr == 0x00)
        {
            P46 = ~P46;

            isr = LinReadReg(LID);                  //Determine whether the slave responds ID
            if(isr == 0x12)                         //return response data
            {
                LinTxResponse(TX_BUF);
            }
            else
            {
                RxFlag = 1;
            }
        }
    }
    else
    {
        isr = LinReadReg(LER);                  //read clear error register
    }
}

```

### 30.4.3 Example of Emulating a LIN Bus Using a Serial Port

LIN (Local Interconnect Network) bus is a low-cost serial communication based on UART/SCI (Universal Asynchronous Receiver/Serial Interface).

communication agreement. The byte field of LIN is in the same 8N1 format as UART, but the frame interval field of LIN is 13 consecutive dominant levels.

It is inconsistent with the 8 bits of the serial port. Here, 13 signals of dominant level width are output as the interval field by switching the baud rate.

Take 9600 baud as an example:

13 dominant signal time = 13/9600 = 1354us translates into sending a

byte 0x00, 1 start bit + 8 data bits + 1 stop bit, which contains 9 dominant levels.

Convert baud rate = 9/1354us = 6647 baud rate

The test method of this routine: UART1 is connected to the computer through the serial port tool; UART2 is connected to an external LIN transceiver and connected to the LIN bus. string computer

The data sent from the LIN bus is forwarded to the LIN bus; the data received from the LIN bus is forwarded to the computer serial port.

---

//The test frequency is **11.0592MHz**

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#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 transmission baud rate

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

#define UART1_BUF_LENGTH 32
#define UART2_BUF_LENGTH 32

#define LIN_ID 0x31

sbit SLP_N = P2^4; // LIN transceiver control pin: sleep; 1 work

u8 TX1_Cnt; //send count
u8 RX1_Cnt; //receive count
u8 TX2_Cnt; //send count
u8 RX2_Cnt; //receive count
bit B_TX1_Busy; //Send busy flag
bit B_RX2_Busy; //Send busy flag
u8 RX1_TimeOut; //receive buffer
u8 RX2_TimeOut; //receive buffer

u8 xdata RX1_Buffer[UART1_BUF_LENGTH]; 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 SetTimer2Baudrate(u16 dat);

void main(void)
{
    u8 i;

    CKCON = 0x00; //Set the external data bus speed to the fastest
    WTST = 0x00; //Set the program code to wait for parameters,
                  //assign to CPU The speed of executing the program is set to the fastest

    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

```

```

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

while(1)
{
    delay_ms(1);
    if(RX1_TimeOut > 0)
    {
        if(--RX1_TimeOut == 0) //If the timeout expires, the serial port reception ends.
        {
            if(RX1_Cnt > 0)
            {
                Lin_Send(RX1_Buffer); //Will UART1 Received data sent to LIN on the bus
            }
            RX1_Cnt = 0;
        }
    }

    if(RX2_TimeOut > 0)
    {
        if(--RX2_TimeOut == 0) //If the timeout expires, the serial port reception ends.
        {
            if(RX2_Cnt > 0)
            {
                for (i=0; i < RX2_Cnt; i++)
                {
                    UART1_TxByte(RX2_Buffer[i]); //end with stop^0
                }
            }
            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 change 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)                                // calculateLIN1.3 Classic check code
{
    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)
{
    SetTimer2Baudrate((u16)Baudrate_Break);
    UART2_TxByte(0);
    SetTimer2Baudrate((u16)Baudrate2);
}

void Lin_Send(u8 *puts)
{
    u8 i;

    Lin_SendBreak();
    UART2_TxByte(0x55);                                         // Transmit Interframe Field
    UART2_TxByte(Lin_CheckPID(LIN_ID));                         // send sync field
    for(i=0;i<8;i++)                                            // send identifier
    {
        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 SetTimer2Baudrate(u16 dat)
{
    AUXR &= ~(1<<4);                                         // Timer stop
    AUXR &= ~(1<<3);                                         // Timer2 set As Timer
    AUXR |= (1<<2);                                          // Timer2 set as 1T mode
    T2H = dat / 256;
}

```

```

T2L = dat % 256;
IE2 &= ~(1<<2);
AUXR |= (1<<4);
}

void UART1_config(void)
{
    TR1 = 0;
    AUXR &= ~0x01;
    AUXR |= (1<<6);
    TMOD &= ~(1<<6);
    TMOD |= ~0x30;
    TH1 = (u8)(Baudrate1 / 256);
    TL1 = (u8)(Baudrate1 % 256);
    ET1 = 0;
    INTCLKO &= ~0x02;
    TR1 = 1;

    SCON = (SCON & 0x3f) | 0x40;           //UART1 Mode bit data variable baud rate
    ES = 1;                                //enable interrupt
    REN = 1;                                //Allow to receive
    P_SW1 &= 0x3f;                          //UART1 switch to P3.0 P3.1

    B_TX1_Busy = 0;
    TX1_Cnt = 0;
    RX1_Cnt = 0;
}

void UART2_config(void)
{
    SetTimer2Baudrate((u16)Baudrate2);
    S2CON &= ~(1<<7);
    IE2 |= 1;
    S2CON |= (1<<4);
    P_SW2 &= ~0x01;

    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;
    }
}

```

```

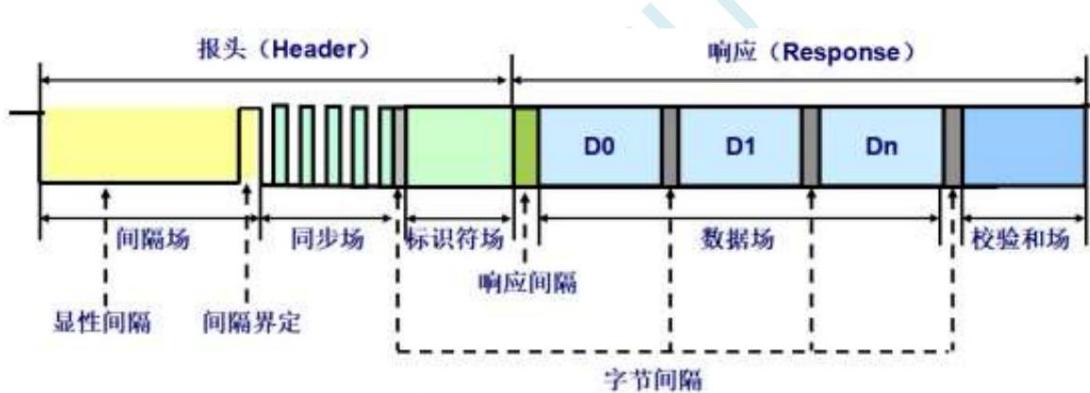
void UART2_int (void) interrupt 8
{
    if((S2CON & 1) != 0)
    {
        S2CON &= ~1;                                //clear flag
        if(RX2_Cnt >= UART2_BUF_LENGTH) RX2_Cnt = 0;
        RX2_Buffer[RX2_Cnt] = S2BUF;
        RX2_Cnt++;
        RX2_TimeOut = 5;
    }

    if((S2CON & 2) != 0)
    {
        S2CON &= ~2;                                //clear flag
        B_TX2_Busy = 0;
    }
}

```

---

### 30.4.4 LIN bus timing introduction



#### 1. Byte field 1)

Communication format based on UART/SCI;

2) It takes 10 bit times (TBIT) to send a byte

#### 2. Interval field 1)

indicates the start of a frame of message, which is sent by the master node;

2) The interval signal consists of at least 13 dominant bits;

3) The interval delimiter consists of at least one invisible bit;

4) The interval field is the only field that does not conform to the byte field format;

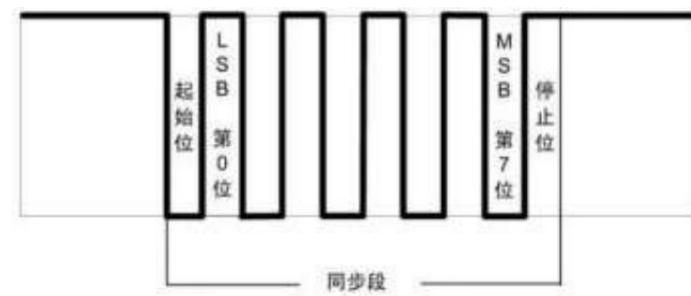
5) The slave node needs to detect at least 11 consecutive dominant bits to be considered as interval signals;



#### 3. Sync field 1)

Ensure that all slave nodes send and receive data using the same baud rate as the node;

2) One byte, fixed structure: 0x55;



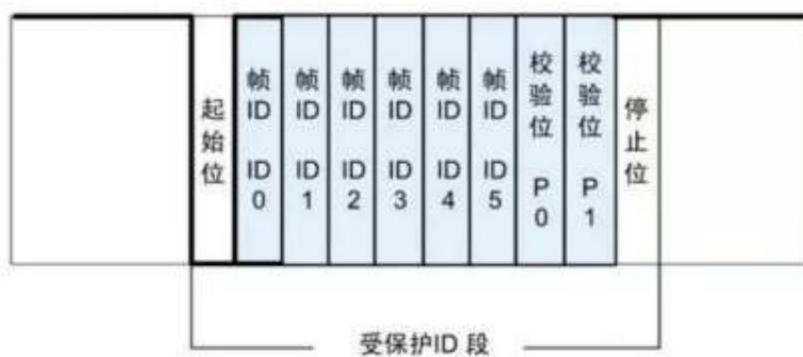
## 4. Identifier field

1) The range of ID is from 0 to 63 (0x3f);

2) Parity (Parity) P0, P1

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$$

$$P1 = \neg(ID1 \oplus ID3 \oplus ID4 \oplus ID5)$$



显性或隐性位

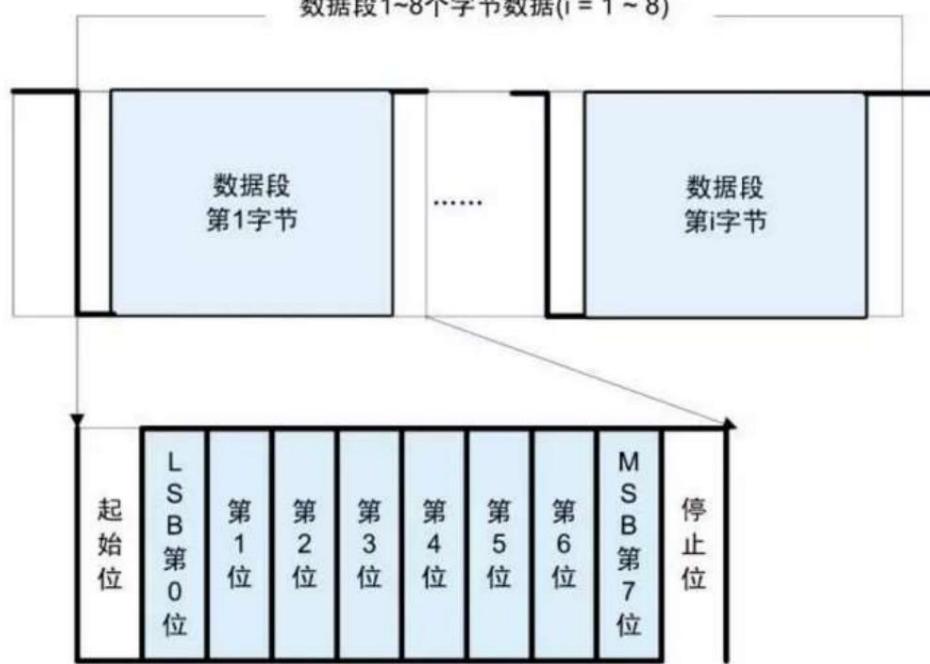
## 5. Data field 1)

The length of the data field is 1 to 8 bytes;

2) The low byte is sent first, and the high byte is sent first;

3) If the length of a signal exceeds 1 byte, it is sent in a low-order manner (little endian);

数据段1~8个字节数据(i = 1 ~ 8)



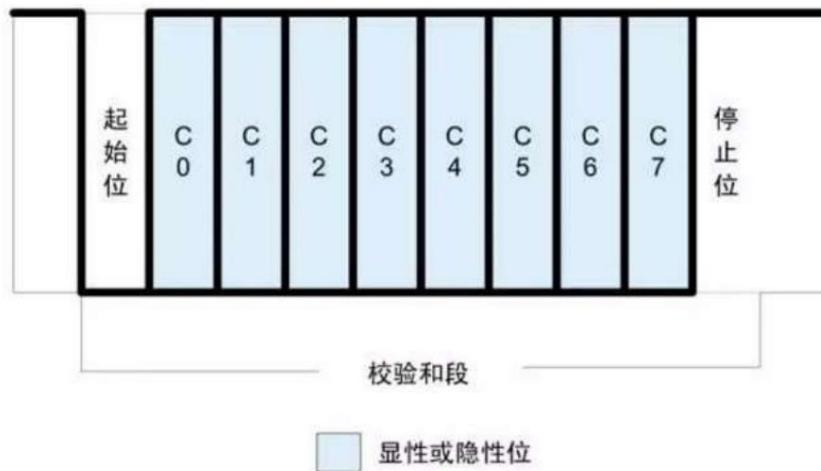
## 6. Checksum field

Used to verify that the received data is correct

- 1) Classic Checksum only checks the data field (LIN1.3)
- 2) Enhanced checksum (Enhance Checksum) check identifier field and data field content (LIN2.0, LIN2.1)

Frames with identifiers 0x3C and 0x3D can only use classic parity

Calculation method: Invert 8-bit summation



## 31 32-bit hardware multiply and divide unit (MDU32)

The multiply and divide unit (called MDU32) provides fast 32-bit arithmetic. MDU32 supports unsigned and two's complement signed integer Number operand. MDU32 is controlled by a dedicated direct memory access module (called DMA). All MDU32 arithmetic operations are performed by The DMA control writes the DMA instruction to start the register DMAIR. Operands and results of all arithmetic operations performed by the MDU32 module located in registers R0-R7.

Notice:

1. The execution time required for the DMA module to perform arithmetic operations, including:

- ÿ Operands are loaded from the DR0-DR4 registers to the MDU32 module
- ÿ MDU32 arithmetic operations
- ÿ Result storage from MDU32 module to R0-R7 registers

2. The execution time required for the processor to execute the C-compiled arithmetic function, including:

- ÿ DMA instruction writes to DMAIR register
- ÿ Operands are loaded from the DR0-DR4 registers to the MDU32 module
- ÿ MDU32 arithmetic operations
- ÿ Result storage from MDU32 module to R0-R7 registers
- ÿ Return from function (RET instruction)

3. When MDU32 performs multiplication and division operations, the microcontroller will automatically switch to IDLE mode, that is, the CPU stops the clock instruction, and other peripherals still continue to work. After the operation is completed, the microcontroller automatically switches to the normal working mode

### 31.1 Related Special Function Registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMAIR	DMA instruction register EDH		DMAIR[7:0]								0000,0000

### 31.2 Operation Execution Schedule

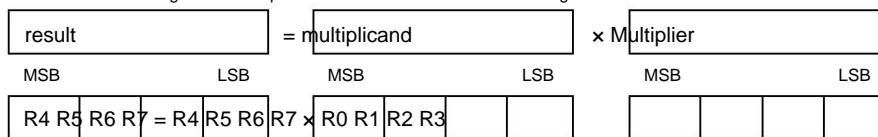
MDU operation	DMA instruction code		number of execution clocks
	HEX	DEC	
32-bit	0x02	2	3
multiplication 32-bit	0x04	4	19
unsigned division 32-bit signed division	0x06	6	twenty one

## 31.3 MDU32 Arithmetic Operations

### 31.3.1 32-bit multiplication

32-bit multiplication is performed on two unsigned or signed two's complement integer arguments. The first parameter is located in the R4-R7 registers,

The second parameter is located in the R0-R3 registers. The operation result is stored in the R4-R7 registers.



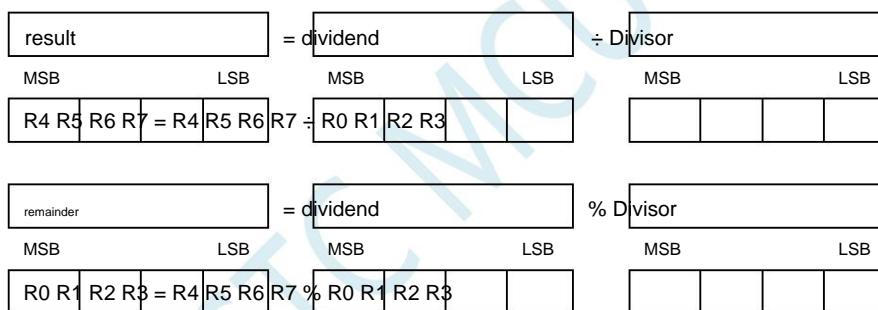
DMA instruction code: 0x02

execution time: 3clk

### 31.3.2 32-bit unsigned division

Performs a 32-bit unsigned division operation on two unsigned integer arguments. The first parameter "dividend" is located in the R4-R7 register,

The second parameter "Divisor" is located in the R0-R3 registers. The result is stored in the R4-R7 registers. The remainder is returned in R0-R3. Divide by zero returns 0xFFFFFFFF.



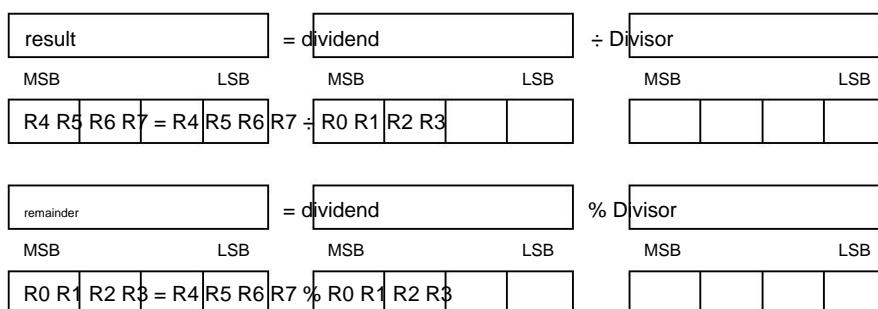
DMA instruction code: 0x04

execution time: 19clk

### 31.3.3 32-bit signed division

Performs a 32-bit signed division operation on two's complement signed arguments. The first parameter "dividend" is located in the R4-R7 registers,

The second parameter "Divisor" is located in the R0-R3 registers. The result is stored in the R4-R7 registers. The remainder is returned in R0-R3. Divide by zero returns 0xFFFFFFFF.



DMA instruction code: 0x06

execution time: 21clk

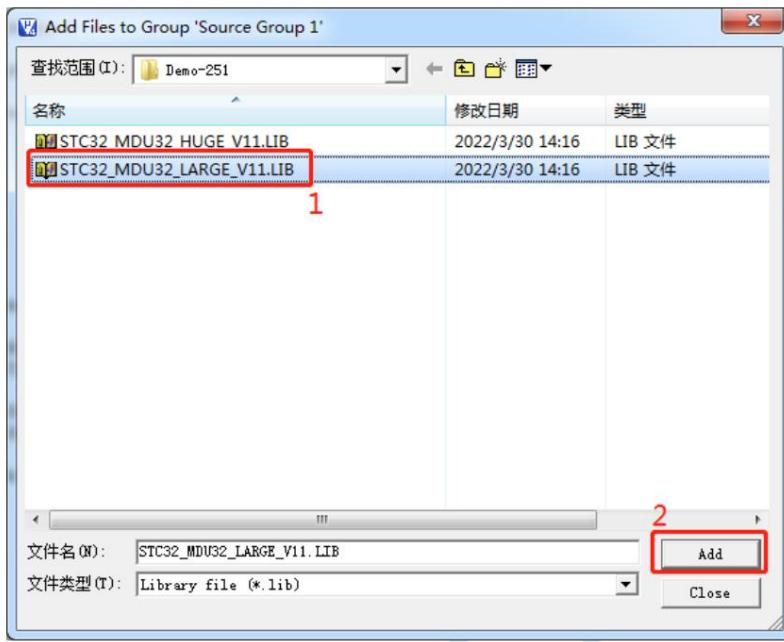
## 31.4 Example Program

When you want to use the 32-bit hardware multiplication and division unit, just add it to the keil project

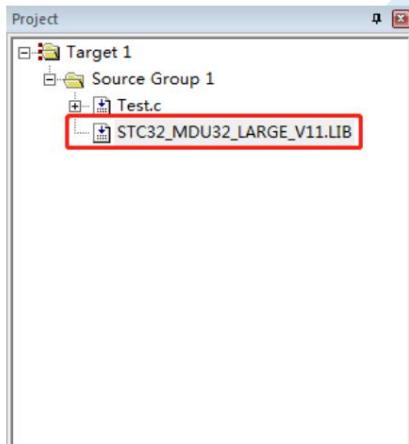
The library file "STC32\_MDU32\_LARGE\_Vxx.LIB" or "STC32\_MDU32\_HUGE\_Vxx.LIB" can be

(Note: The specific need to add the LAGRE version or the HUGE version needs to be selected according to the code size mode of the project. If the code

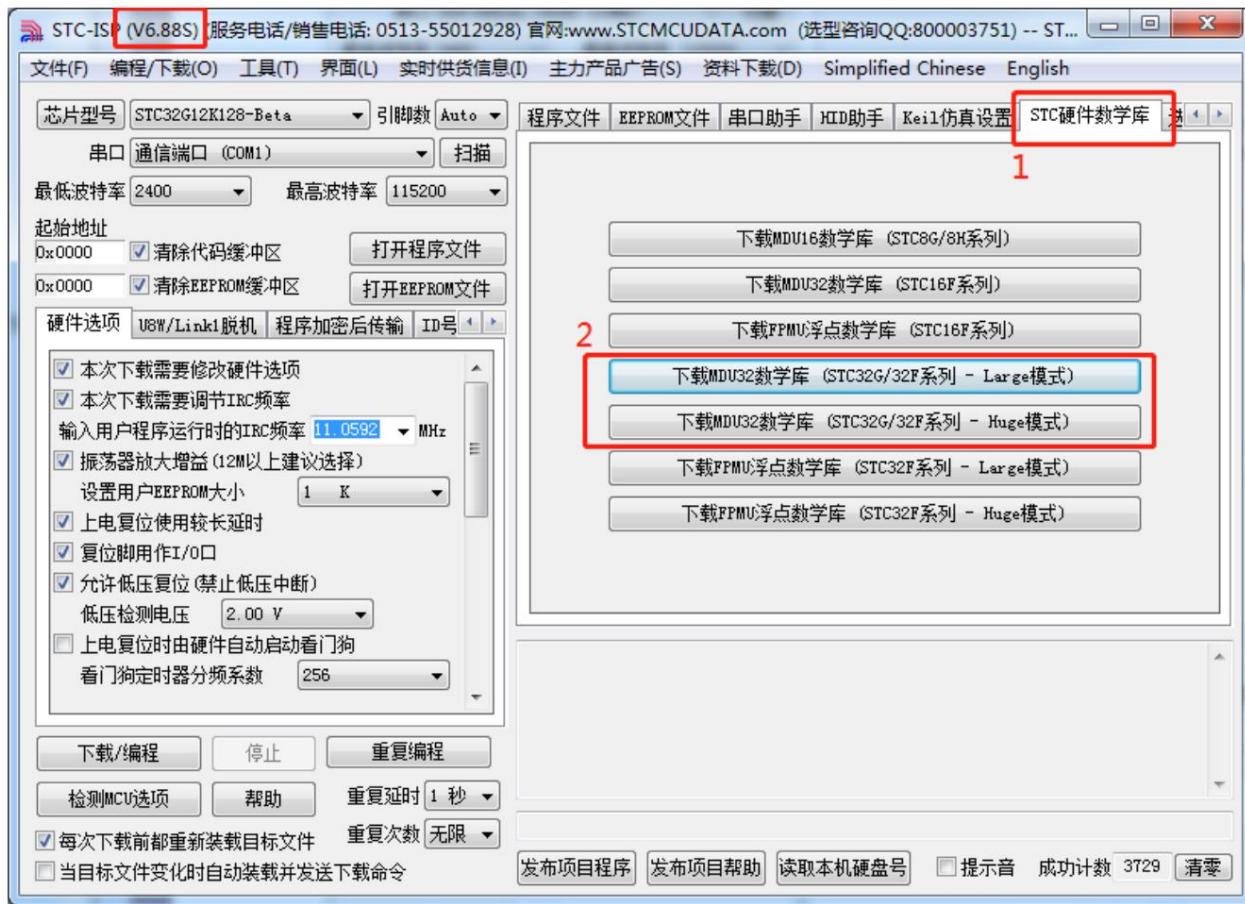
If the code size mode is Huge, you need to add the HUGE version of the library, and other modes need to add the LARGE version)



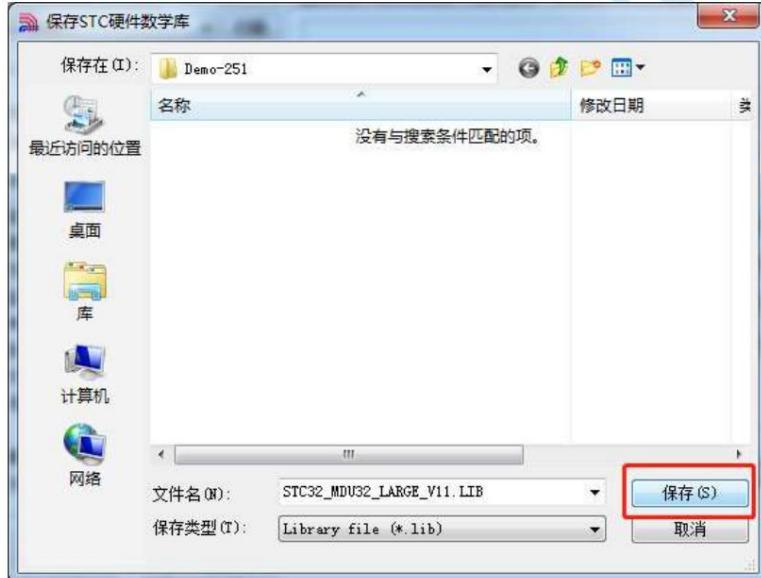
Add library files to the project:



How to obtain the library file: Through STC-ISP software V6.88S and later versions, click the "STC Hardware Math Library" tab to obtain it.



Click the button of the MDU32 math library to be downloaded, select the save location, and click the "Save" button:




---

// The test frequency is 11.0592MHz

```

//#include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
#include "intrins.h"

volatile unsigned long int near uint1, uint2, xuint;
volatile long int sint1, sint2, xsint;

```

```
void main(void)
{
    EAXFR = 1;                                //Enable      XFR
    CKCON = 0x00;                             //access to set external data bus speed to fastest
    WTST = 0x00;                            //Set the program code to wait for parameters,
                                            //assign to      CPU The speed of executing the program is set to the fastest

    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 arithmetic units (FPMU)

### 32.1 Introduction to FPMU Floating Point Operators

The single-precision floating-point unit (FPMU) provides fast single-precision floating-point arithmetic operations. The FPMU supports the addition, subtract, multiply, divide, square root, compare and trigonometric functions (sine, cosine, tangent and arctangent). Supports both integer types and single-precision floating point conversion between numbers. The input floating point number format conforms to the IEEE-754 standard. The FPMU is controlled by a dedicated direct memory access DMA. all Arithmetic operations are initiated by writing an arithmetic instruction to a control register called DMAIR. All arithmetic operations performed by the FPMU module. The operands (or pointers thereof) and the result (or pointers thereof) are located in registers R0-R7 of the current group.

### 32.2 Related Special Function Registers

symbol	describe	address	Bit addresses and symbols								reset value
			B7	B6	B5	B4	B3	B2	B1	B0	
DMAIR	DMA instruction register E0H										0000,0000

### 32.3 Operation Execution Schedule

FPMU operation	DMA instruction code		number of execution clocks
	HEX	DEC	
Floating Point	0x1C	28	31 to 40
Addition	0x1D	29	31 to 40
Floating Point	0x1E	30	26 to 34
Subtraction	0x1F	31	58 to 67
Floating Point	0x20	32	50 to 54
Multiplication	0x21	33	18
Floating Point	0x22	34	15
Division	0x2D	45	32 to 270
Floating Point	0x2E	46	32 to 270
Root Floating	0x2F	47	58 to 258
Point	0x30	48	62 to 175
Comparison Floating	0x23	35	19 to 30
Point Detection Sine	0x24	36	19 to 30
Function Cosine Function	0x25	37	23 to 39
Tangent Function Arc	0x27	39	23 to 33
Tangent Function Bit	0x28	40	23 to 33
Integer to Float 16-bit	0x29	41	24 to 33
Integer to Float 32-bit	0x31	49	2
Integer to Float	0x32	50	4
Initialize Coprocessor Clear Exception Read Status Register	0x33	51	4

Write Status Register	0x34	52	4
Read Control Register	0x35	53	4
Write Control Register	0x36	54	4

STCMCU

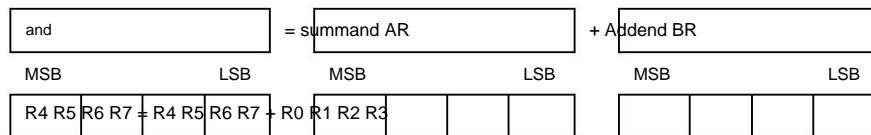
## 32.4 FPMU Basic Arithmetic Operations

This subsection describes all arithmetic operations that the FPMU module can perform using the DMA controller. All operands must be within the data in storage. The result of the operation is also stored in the data memory space of the current bank of R0-R7 selected by the PSW (0xD0) bits.

### 32.4.1 Floating-Point Number Addition (+)

Adds two floating-point numbers. The addend BR is located in the R0~R3 registers, and the summand AR is located in the R4~R7 registers.

The result sum is saved to the R4~R7 registers



Instruction code 0x1C(28)

Execution time (number of clocks) 31 to 40

### 32.4.2 Floating-point subtraction (-)

Subtract two floating point numbers. The subtrahend BR is located in the R0~R3 registers, and the minuend AR is located in the R4~R7 registers.

The result difference is saved to the R4~R7 registers



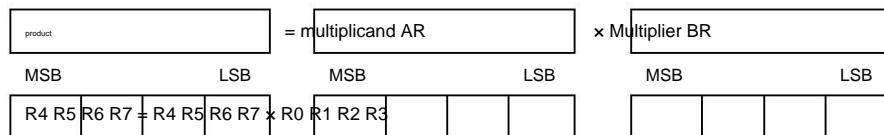
Instruction code 0x1D(29)

Execution time (number of clocks) 31 to 40

### 32.4.3 Multiplication of floating-point numbers (x)

Multiples two floating point numbers. The multiplier BR is located in the R0~R3 registers, and the multiplicand AR is located in the R4~R7 registers.

The fruit product is stored in the R4~R7 registers



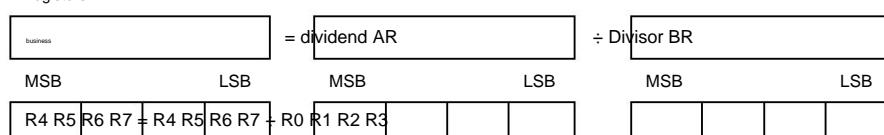
Command code 0x1E(30)

Execution time (number of clocks) 26 to 34,

### 32.4.4 Floating-point division (÷)

Divide two floating point numbers. The divisor BR is located in the R0~R3 registers, and the dividend AR is located in the R4~R7 registers.

The fruit quotient is saved to the R4~R7 registers



script

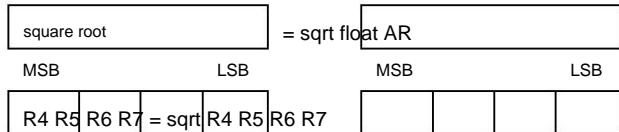
0x1F(31)

Execution time (number of clocks) 58 to 67

### 32.4.5 Floating-point square root/square root (sqrt)

Take the square root of a floating-point number. The square root AR is located in the R4~R7 registers, and the square root of the calculation result is stored in the R4~R7 registers.

register



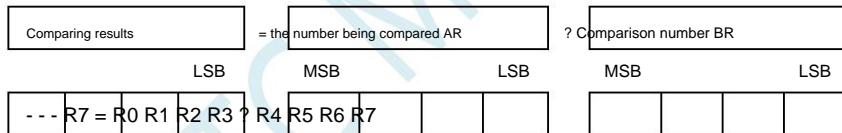
Script 0x20(32)

Execution time (number of clocks) 50 - 54

### 32.4.6 Floating-Point Comparison (comp)

Performs an arithmetic comparison operation on two floating-point numbers. The comparison number BR is located in the R0~R3 registers, and the compared number AR is located in the R4~R7 registers, the comparison result is saved to the R7 register

R7.3	R7.2	R7.1	R7.0	Comparison Results	Result Description
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



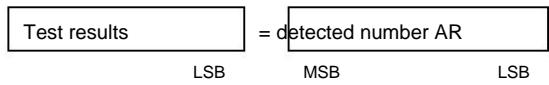
Script 0x21(33)

Execution time (number of clocks) 18

### 32.4.7 Floating-Point Number Check (check)

Check for 1 floating point number. The detected number AR is located in the R4~R7 registers, and the detection result is stored in the 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 canonical floating point number
0000	0	1	1	0	Negative canonical floating point number
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	negative zero
0000	1	1	0	0	positive denormal floating point number
0000	1	1	1	0	Negative denormalized floating point number





Script 0x22(34)

Execution time (number of clocks) 15

## 32.5 FPMU trigonometric functions

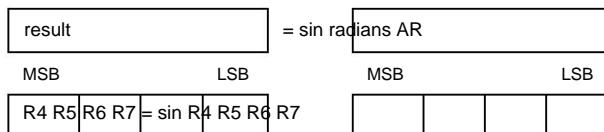
Note: The angle parameter type of all trigonometric functions is radians. The conversion formula of radian and angle:

$$\text{angle} = \frac{180^\circ}{\pi} \times \text{radian}$$

$$\text{radian} = \frac{\pi}{180^\circ} \times \text{angle}$$

### 32.5.1 The sine function (sin)

Finds the sine of a single-precision radian floating-point number. The number of radians AR is located in the registers R4~R7, and the calculation results are stored in the registers R4~R7

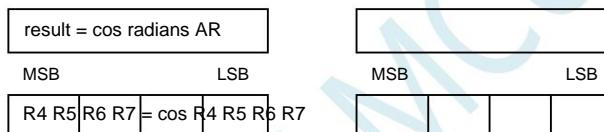


Script 0x2D(45)

Execution time (number of clocks) 32 - 270 clk

### 32.5.2 The cosine function (cos)

Finds the cosine of a single-precision radian floating-point number. The number of radians AR is located in the registers R4~R7, and the calculation results are stored in the registers R4~R7

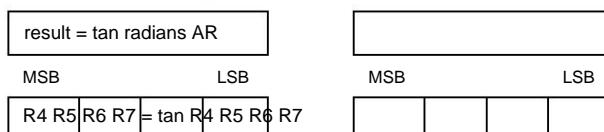


Instruction code 0x2E(46)

Execution time (number of clocks) 32 - 270

### 32.5.3 The tangent function (tan)

Finds the tangent of a single-precision radian floating-point number. The number of radians AR is located in the registers R4~R7, and the calculation results are stored in the registers R4~R7

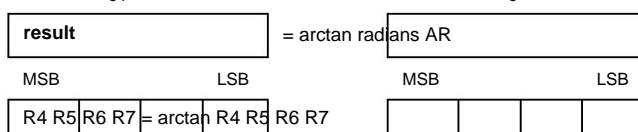


Instruction code 0x2F(47)

Execution time (number of clocks) 58 - 258

### 32.5.4 The arc tangent function (arctan)

Finds the arctangent of a single-precision radian floating-point number. The number of radians AR is located in the registers R4~R7, and the calculation results are stored in the registers R4~R7



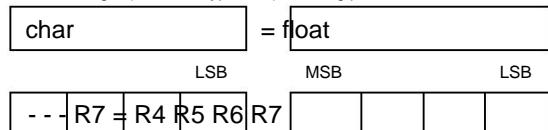
Script 0x30(48)

Execution time (number of clocks) 62 - 175

## 32.6 FPMU Data Conversion Operations

### 32.6.1 Floating point to 8-bit integer (float → char)

Converts a floating-point number to an 8-bit integer (character type char). Floating point numbers are located in registers R4~R7, and 8-bit integers are stored in R7



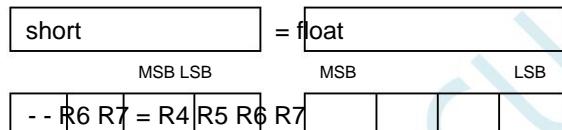
Script 0x23(35)

Execution time (number of clocks) 19 - 30

### 32.6.2 Floating point number to 16-bit integer (float → short)

Converts a floating point number to a 16-bit integer (short). Floating-point numbers are located in registers R4~R7, and 16-bit integers are stored in registers R6~R7

middle



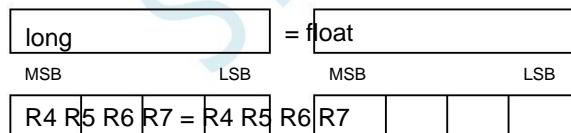
Script 0x24(36)

Execution time (number of clocks) 19 - 30

### 32.6.3 Floating point to 32-bit integer (float → long)

Converts a floating point number to a 32-bit integer (long). The floating-point number AR is located in the R4~R7 registers, and the 32-bit integer is stored in R4~R7.

in R7

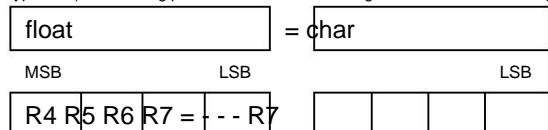


Script 0x25(37)

Execution time (number of clocks) 23 - 39

### 32.6.4 8-bit integer to floating point number (char → float)

Converts an 8-bit integer (character type char) to a floating point number. The 8-bit integer is located in the R7 register, and the floating-point number is stored in R4~R7



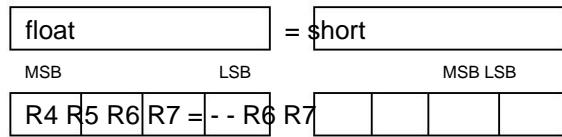
Script 0x27(39)

Execution time (number of clocks) 23 - 33

### 32.6.5 16-bit integer to floating point number (short → float)

Converts a 16-bit integer (short) to a floating-point number. The 16-bit integer is located in the R6~R7 registers, and the floating-point number is stored in R4~R7

middle



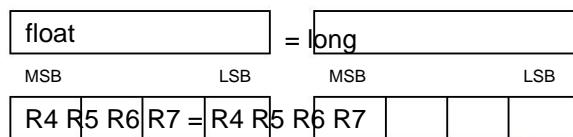
Script 0x28(40)

Execution time (number of clocks) 23 - 33

### 32.6.6 32-bit integer to floating point number (long → float)

Converts a 32-bit integer (long) to a floating-point number. 32-bit integers are located in registers R4~R7, and floating-point numbers are stored in registers R4~R7

middle



Script 0x29(41)

Execution time (number of clocks) 24 - 33

## 32.7 FPMU Coprocessor Control Operations

### 32.7.1 Initializing the Coprocessor

Initialize the FPMU coprocessor. After initialization, an abnormal state will be generated, which needs to be cleared by software.

Script 0x31(49)

Execution time (number of clocks) 2

### 32.7.2 Clear exceptions

Clear all abnormal states

Script 0x32(50)

Execution time (number of clocks) 4

### 32.7.3 Read Status Register

Read the coprocessor's status register. The result is saved to R7

Script 0x33(51)

Execution time (number of clocks) 4

### 32.7.4 Write Status Register

Write the status register of the coprocessor. The value to be written is located in the R0 register, and the new status register value is saved to R7 after completion

Script 0x34(52)

Execution time (number of clocks) 4

### 32.7.5 Read Control Register

Read the control register of the coprocessor. The result is saved to R0

Script 0x35(53)

Execution time (number of clocks) 4

### 32.7.6 Write Control Register

Write the control register of the coprocessor. The value to be written is located in the R0 register, and the new control register value is saved to R7 after completion

Script 0x36(54)

Execution time (number of clocks) 4

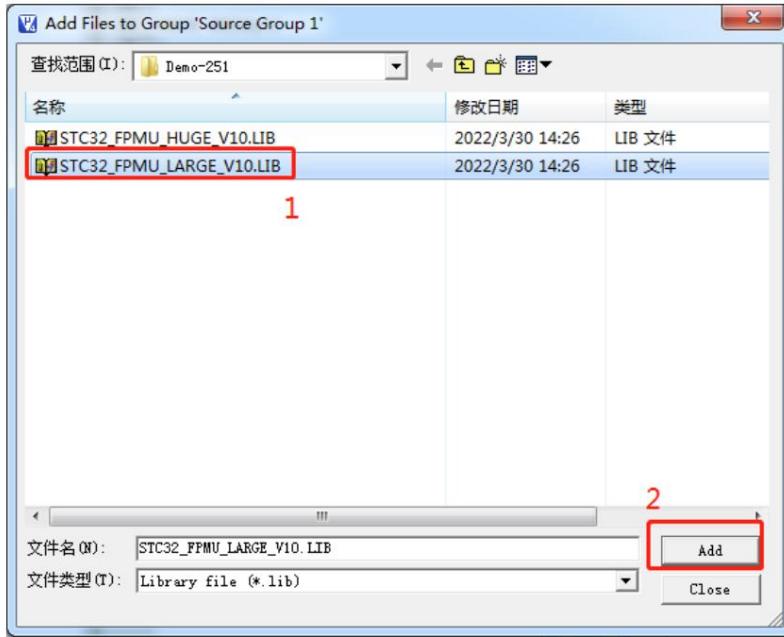
## 32.8 Sample Programs

When you want to use hardware floating point, just add in the keil project

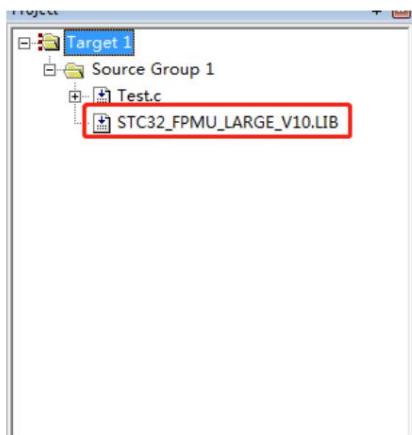
The library file "STC32\_FPMU\_LARGE\_Vxx.LIB" or "STC32\_FPMU\_HUGE\_Vxx.LIB" can be:

(Note: The specific need to add the LAGRE version or the HUGE version needs to be selected according to the code size mode of the project. If the code

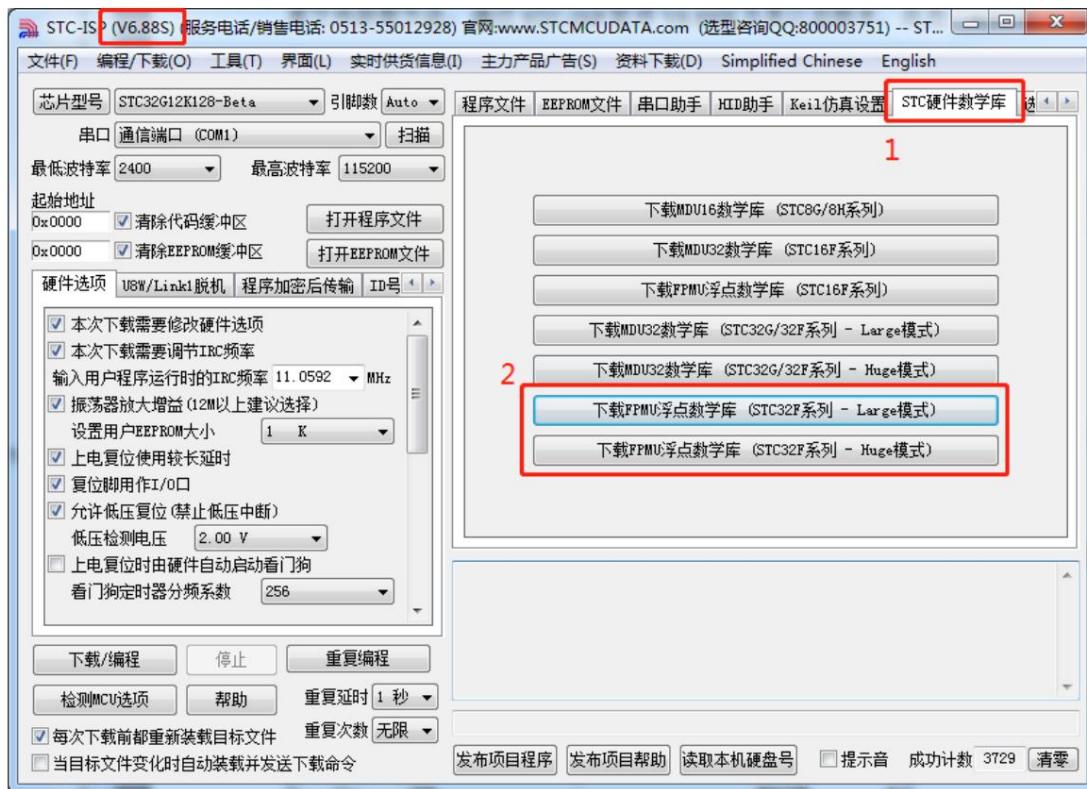
If the code size mode is Huge, you need to add the Huge version of the library, and other modes need to add the Large version)



Add library files to the project:



How to obtain the library file: Through STC-ISP software V6.88S and later versions, click the "STC Hardware Math Library" tab to obtain it.



Click the button of the FPMU math library you want to download, select the save location, and click the "Save" button:



//The test frequency is 11.0592MHz

```

##include "stc8h.h"
#include "stc32g.h"
#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 XFR
}

```

```
CKCON = 0x00; //Set the external data bus speed to the fastest
WTST = 0x00; //Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

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);

}
```

## Appendix A Instruction Set

### A.1 Introduction to the instruction set

#### A.1.1 BINARY mode and SOURCE mode

Binary mode and Source mode refer to two ways to provide opcodes for the STC32G architecture instruction set. Depending on the user program, Binary mode or Source mode may generate more efficient code. Binary mode refers to the standard opcodes of the MCU51. Source mode refers to the MCU251 specific opcode set, which extends the instruction set with additional operation and addressing modes. The special mnemonic 0xA5 is used to distinguish specific instructions in each mode. All unused opcodes are correctly decoded and executed as NOPs.

#### A.1.2 Instruction Set Marking

Instructions have five different addressing modes: immediate, direct, register, indirect, and relative. In immediate addressing mode, the operand is contained in the opcode. For direct addressing, the 8-bit address or 16-bit address is part of the opcode; for register addressing, a register is selected for the operation in the opcode. In indirect addressing mode, a register is selected in the opcode to point to the address used by the operation. Relative addressing mode is used for jump instructions. The table below provides the cycle count of the STC32G microcontroller core instruction set. One cycle is equal to one system clock. Tables 1 and 2 contain descriptions of the mnemonics used in the instruction set table. Tables 3 through 7 indicate the hexadecimal code, number of bytes, and number of system clocks required for each instruction to execute.

Rn	Current working byte registers R0, R1, ..., R7
N	Byte register index 0~7 Binary representation of
rrr	n byte registers R0, R1, ..., R15
Rm	
Rmd	target register
Rms	Source Number
m,md,ms	Register Byte Register Index: m, md, ms = 0, 1, ..., 15 binary
ssss	representation of m or md binary representation of ms Word
SSSS	registers WR0, WR2, ..., WR30
WR	
WRjd	target register
WRjs	Source register
@WRj	Indirect memory location addressed by word register
@WRj+dis	(0x0000~0xFFFF) Indirect memory location addressed by word register + 0 to 64KB offset
j, jd, js	(0x0000~0xFFFF) Word register index: j, jd, js = 0, 2, ..., 30 binary representation of j or jd binary representation of js Double word registers DR0, DR4, ..., DR28, DR56, DR60
TTTT	
DRk	

DRkd	target register
DRks	source register
@DRk	Indirect memory location addressed by double word register (0x000000~0xFFFF)
@DRk+dis	Indirect memory location addressed by double word register +0 to 64KB offset value (0x000000~0xFFFF)
k, kd, ks	Double word register index: k, kd, ks = 0, 4, ..., 28, 56, 60
uuuu	binary representation of k or kd
UUUU	binary representation of ks
dir8	128 internal memory locations, various special function registers
dir16	16-bit memory address (0x000000~0xFFFF)
@Ri	Indirect memory location (0x00~0xFF) addressed by register R0 or R1
#data	8-bit immediate data is included in the instruction
#data16	16-bit immediate data contained in the 2nd and 3rd bytes of the instruction
#0data16	32-bit immediate data; high word is filled with zeros, low word is included in the 2nd and 3rd bytes of the instruction
#1data16	32-bit immediate data; high word is filled with 1, low word is included in the 2nd and 3rd bytes of the instruction
#short	A constant equal to 1, 2, or 4, included in the instruction
vv	Binary representation of #short
bit	Memory location (0x20~0x7F) or direct addressing bits in any defined SFR
bit51	Direct addressing bits in memory or SFR (bit number = 0x00~0xFF). Bits 0x00~0x7F are bytes in internal memory The position is 128 bits of 0x20~0x2F. Bits 0x80~0xFF are 128 bits in 16 SFRs, and their addresses are 0 or 8. Tail: 0x80, 0x88, 0x90, ..., 0xF0, 0xF8
A	accumulator

Table 1. Notes on Data Addressing Modes

The addr24	24-bit target address can be anywhere in the 16MB address space. It is used for ECALL and EJMP instructions.
The target address of addr16	LCALL and LJMP can be anywhere within the 64KB program memory address space.
The target address of addr11	ACALL and AJMP will be in the same 2 KB program memory page as the first byte of the next instruction face.
rel	SJMP and all conditional jumps contain an 8-bit offset byte. The range is the first byte offset relative to the next instruction Shift +127 to -128 bytes.

Table 2. Program Addressing Mode Notes

-	This instruction does not modify the flag bits.
ÿ	This instruction sets or clears the flag bits as needed.
1	This instruction sets the flag bit.
0	This instruction clears the flag bit.

Table 3. Flag bit description notes

### A.1.3 Instruction List (Functional Sorting)

The total instruction execution time depends on the value of WTST (0xe9). Each table below indicates the number of clocks when WTST=0. Calculate the value of each instruction

The general formula for the total number of clocks is:

$$\text{Instruction Clocks} = \text{Clocks} + \text{nrPRGACS} * \text{WTST} \ (\text{nrPRGACS}=0 \text{ or } 1)$$

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.

$$\text{Instruction Clocks} = \text{Clocks} + \text{nrPRGACS} * \text{WTST} + \text{nrXDMACS} * \text{CKCON}$$

Instructions dedicated to binary mode are marked in red. These instructions need to be prefixed with 0xA5 before the opcode when executed in source mode (ESC). Instructions dedicated to source mode are marked in blue. When executed in binary mode, these instructions require before the opcode

Prefix with 0xA5 (ESC). All other instructions are always available regardless of CPU mode.

arithmetic operations

mnemonic	describe	Number of encoded byte	clocks
ADD A,Rn	Add register to accumulator Add	0x28-0x2F	1
ADD A,dir8	direct byte to accumulator Add	0x25	2
ADD A, @Ri	indirect memory to accumulator	0x26-0x27	1
ADD A,#data	Add immediate value to accumulator	0x24	2
ADD Rm,Rm	add byte register to byte register	0x2C	2
ADD WRj, WRj	add word register to word register	0x2D	2
ADD reg,op2(3) add operand to Rm, WRj or DRk		0x2E	Note 1 Note 2
ADD DRk,DRk	add double word register to double word register	0x2F	2
ADDC A,Rn	Add register to accumulator with carry flag Add direct byte to	0x38-0x3F	1
ADDC A, dir8	accumulator A with carry flag Add indirect memory to accumulator	0x35	2
ADDC A, @Ri	A with carry flag	0x36-0x37	1
ADDC A, #data add immediate	value to accumulator A with carry flag	0x34	2
SUBB A,Rn	Borrow from A and Subtract Register	0x98-0x9F	1
SUBB A,dir8	Borrow from A and Subtract Direct Byte	0x95	2
SUBB A, @Ri	Borrow from A and Subtract Indirect	0x96-0x97	1
SUBB A,#data	Memory Borrow from A and Subtract	0x94	2
SUB Rm,Rm	Literal Subtract Byte Register from Byte Register	0x9C	2
SUB WRj,WRj	Subtracts word register from word register	0x9D	2
SUB reg,op2(3)	Subtract operand from Rm, WRj or DRk	0x9E	Note 1 Note 2
SUB DRk,DRk	Subtracts double word register from double word register	0x9F	2
CMP Rm, Rm	Compare two byte registers	0xBC	2
CMP WRj, WRj	compare two word registers	0xBD	2
CMP reg,op2(3)	Compare Rm, WRj or DRk with operand	0xBE	Note 1 Note 2
CMP DRk,DRk	Compare two double word registers	0xBF	2
INC A	Increment accumulator	0x04	1
INC Rn	Increment register	0x08-0x0F	1
INC dir8	Increment direct byte	0x05	2
INC @Ri	Increment indirect memory	0x06-0x07	1
INC reg,#short(3)	Increment Rm, WRj or DRk	0x0B	2 Note 2

DEC A	decrement	0x14	1	1
DEC Rn	accumulator	0x18-0x1F	1	1
DEC dir8	decrement register	0x15	1	1
DEC @Ri	decrement direct byte decrement indirect memory	0x16-0x17	2	1
DEC reg, #short(3) decrements	Rm, WRj or DRk increments the data	0x1B	2	Note 2
INC DPTR	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 Divide Word	0x8C	1	6
DIV WRj,WRj	Register Divide Decimal	0x8D	1	10
DA A	Adjustment Accumulator Note 1:	0xD4	1	3

The number of bytes required by the instruction depends on the addressing mode determined by the byte that follows. Please refer to the instruction set for details.

Note 2: The cycles required for an instruction depend on the addressing mode determined by the immediately following byte. Please refer to the instruction set for details.

Note 3: Operands and addressing modes depend on the next byte. All options are described in Instruction Set Details.

#### logic operation

mnemonic	describe	Number of encoded bytes	clocks
ANL A,Rn	Register logic and accumulator	0x58-0x5F	1
ANL A,dir8	direct byte logic and accumulator	0x55	2
ANL A, @Ri	indirect memory logic and accumulator	0x56-0x57	1
ANL A,#data immediate logical AND accumulator		0x54	2
ANL dir8,A	Direct byte logic AND accumulator	0x52	2
ANL dir8, #data direct data logic and direct bytes		0x53	3
ANL Rm, Rm two-byte register logic AND		0x5C	2
ANL WRj, WRj two word register logic AND		0x5D	2
ANL reg,op2(3) operand logic AND Rm, WRj or DRk register logic OR accumulator		0x5E	Note 1 Note 2
ORL A,Rn	direct byte logic OR accumulator indirect memory logic	0x48-0x4F	1
ORL A,dir8	OR accumulator	0x45	2
ORL A, @Ri		0x46-0x47	1
ORL A, #data immediate logical OR accumulator		0x44	2
ORL dir8,A	accumulator direct byte	0x42	2
ORL dir8, #data immediate logical or immediate byte		0x43	3
ORL Rm, Rm two-byte register logical OR		0x4C	2
ORL WRj, WRj two word register logical OR		0x4D	2
ORL reg, op2(3) operand logical OR Rm, WRj or DRk		0x4E	Note 1 Note 2
XRL A,Rn	Register XOR Accumulator	0x68-0x6F	1
XRL A,dir8	Direct Byte XOR Accumulator	0x65	2
XRL A, @Ri	Indirect Memory XOR Accumulator	0x66-0x67	1

XRL A,#data immediate	XOR accumulator direct	0x64	2	1
XRL dir8,A	byte XOR accumulator	0x62	2	1
XRL dir8, #data immediate	to XOR direct bytes	0x63	3	1
XRL Rm, Rm two-byte register XOR		0x6C	2	1
XRL WRj, WRj XOR of two word registers		0x6D	2	1
XRL reg, op2(3) operand	XOR Rm, WRj or DRK	0x6E	Note 1	Note 2
CLR A	Accumulator clear	0xE4	1	1
CPL A	Invert the accumulator	0xF4	1	1
RL A	Rotate the accumulator to the left	0x23	1	1
RLC A	Rotate the accumulator to the left with a carry	0x33	1	1
RR A	Rotate Accumulator Right Rotate	0x03	1	1
RRC A	Accumulator with Carry Rotate Right Through	0x13	1	1
SRA reg(3)	MSB Shift Right Rm or WRj Shift Right Rm or	0x0E	2	1
SRL reg(3)	WRj Shift Left Rm or WRj Swap Nibble in	0x1E	2	1
SLL reg(3)	Accumulator	0x3E	2	1
SWAP A		0xC4	1	1

Note 1: The number of bytes required by the instruction depends on the addressing mode determined by the next byte. Please refer to the instruction set for details.

Note 2: The cycles required for an instruction depend on the addressing mode determined by the immediately following byte. Please refer to the instruction set for details.

Note 3: Operands and addressing modes depend on the next byte. All options are described in Instruction Set Details.

#### Boolean operation

mnemonic	describe	Number of encoded byte	clocks
CLR C	Carry flag bit clear Direct	0xC3	1
CLR bit	bit clear	0xC2	2
SETB C	carry flag bit set	0xD3	1
SETB bit is set directly		0xD2	2
CPL C	Invert the carry flag	0xB3	1
CPL bit	direct bit inversion	0xB2	2
ANL C,bit direct bit logic	AND carry flag	0x82	2
ANL C,/bit non-logical	AND carry flag of direct bit	0xB0	2
ORL C,bit direct bit logi	cal OR carry flag	0x72	2
ORL C, the non-logical	or carry flag of the /bit direct bit	0xA0	2
MOV C, bit is directly m	oved to the carry flag bit	0xA2	2
MOV bit, the C carry flag	is moved to the direct bit	0x92	2
Bit instr(1)	Bit instruction set (MCU251 specific)	0xA9	3

#### data transmission

mnemonic	describe	Number of encoded byte	clocks
MOV A,Rn	move register to accumulator move	0xE8-0xEF	1
MOV A, dir8	direct byte to accumulator	0xE5	2

MOV A, @Ri	Move indirect memory to	0xE6-0xE7	1	1
MOV A,#data	accumulator Move immediate data	0x74	2	1
MOV Rn,A	to accumulator Move accumulator	0xF8-0xFF	1	1
MOV Rn,dir8	to register Move direct byte to register	0xA8-0xAF	2	1
MOV Rn, #data moves the immediate data to the register		0x78-0x7F	2	1
MOV dir8,A	move accumulator to direct byte	0xF5	2	1
MOV dir8, Rn	move register to direct byte	0x88-0x8F	2	1
MOV dir8,dir8	move direct byte to direct byte	0x85	3	1
MOV dir8, @Ri	move indirect memory to direct bytes	0x86-0x87	2	1
MOV dir8,#data move immediate data to direct byte move		0x75	3	1
MOV @Ri,A	accumulator to indirect memory	0xF6-0xF7	1	1
MOV @Ri,dir8	move direct byte to indirect memory	0xA6-0xA7	2	1
MOV @Ri, #data moves immediate data to indirect memory		0x76-0x77	2	1
MOV Rm,Rm	move byte register to byte register	0x7C	2	1
MOV WRj, WRj move word	register to word register	0x7D	2	1
MOV reg, op2(3) move operand to Rm, WRj or DRk		0x7E	Note 1	Note 2
MOV DRk, DRk move double word register to double word register		0x7F	2	1
MOV WRj, @DRk moves indirect (24-bit) memory to WRj		0x0B	3	4
MOV @DRk,WRj moves WRj to indirect (24-bit) memory		0x1B	3	6
MOV Rm, @WRj+dis moves indirect (16-bit) memory at 16-bit offset to Rm		0x09	4	1
MOV @WRj+dis,Rm moves Rm to indirect (16-bit) memory at 16-bit offset		0x19	4	1
MOV Rm, @DRk+dis moves indirect (24-bit) memory at 16-bit offset to Rm		0x29	4	3
MOV @DRk+dis, Rm moves Rm to indirect (24-bit) memory at 16-bit offset		0x39	4	4
MOV WRj, @WRj+dis moves indirect (16-bit) memory at 16-bit offset to WRj		0x49	4	1
MOV @WRj+dis,WRj moves WRj to indirect (16-bit) memory at 16-bit offset		0x59	4	3
MOV WRj,@DRk+dis move indirect (24-bit) memory at 16-bit offset to WRj		0x69	4	4
MOV @DRk+dis,WRj moves WRj to indirect (24-bit) memory at 16-bit offset		0x79	4	6
MOV op1,reg(3)	Move Rm, WRj or DRk to operand	0x7A	Note 1	Note 2
MOVH DRk,#data16(4)	Move 16-bit immediate data to high word of double word register	0x7A	4	1
MOVZ WRj,Rm move byte	register to zero-extended word register	0x0A	2	1
MOVS WRj,Rm move byte	registers to sign-extended word registers	0x1A	2	1
MOV DPTR, #data16 loads	a 16-bit constant into the active DPTR	0x90	3	1
MOVC A,@A+DPTR moves	code bytes to accumulator at DPTR offset	0x93	1	4
MOVC A,@A+PC move	code byte to PC offset accumulator move external	0x83	1	3
MOVX A, @Ri	memory (8-bit address) to A	0xE2-0xE3	1	2*
MOVX A,@DPTR Move	external memory (16-bit address) to A Move A to external	0xE0	1	2*
MOVX @Ri,A	memory (8-bit address)	0xF2-0xF3	1	2*
MOVX @DPTR,A moves A	to external memory (16-bit address)	0xF0	1	2*
PUSH dir8	push direct bytes onto the IDM stack	0xC0	2	1

POP dir8	Pop Direct Byte from IDM Stack Push Operand on	0xD0	2	1
PUSH op1(3)	IDM Stack Pop Operand from IDM Stack Register	0xCA	Note 1	Note 2
POP op1(3)	Swap with Accumulator Direct Byte Swap with	0xDA	Note 1	Note 2
XCH A,Rn	Accumulator	0xC8-0xCF	1	1
XCH A,dir8		0xC5	2	1
XCH A, @Ri	Indirect memory swap with accumulator	0xC6-0xC7	1	1
XCHD A, @Ri	The lower nibble of indirect memory is swapped with A	0xD6-0xD7	1	3

Note 1: The number of bytes required by the instruction depends on the addressing mode determined by the next byte. Please refer to the instruction set for details.

Note 2: The cycles required for an instruction depend on the addressing mode determined by the immediately following byte. Please refer to the instruction set for details.

Note 3: Operands and addressing modes depend on the next byte. All options are described in Instruction Set Details.

Note 4: The first byte of the opcode of the MOvh instruction is the same as the MOV op1,reg group. Instructions are distinguished by the value of the second byte.

### program jump

mnemonic	describe	Number of encoded byte clocks		
ACALL addr11	Absolute subprogram calls	0x11-0xF1	2	3
LCALL addr16	long subprograms directly calls	0x12	3	3
ECALL addr24	extended subprograms directly calls	0x9A	4	3
ECALL @DRk	extended subprograms indirect calls	0x99	2	3
LCALL @WRj	Long subroutine indirect call	0x99	2	3
RET	subroutine return	0x22	1	3
ERET	Extended 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 extended jump	0x8A	4	3
LJMP @WRj	Indirect long jump	0x89	2	3
EJMP @DRk	Indirect extended jump	0x89	2	3
SJMP rel	Short jump (relative address)	0x80	2	3
JMP @A+DPTR	Indirect jump to DPTR offset	0x73	1	3
JZ rel	Jump if the accumulator is zero Jump if	0x60	2	1/3
JNZ rel	the accumulator is not zero Jump if the carry	0x70	2	1/3
JC rel	flag is set Jump if the carry flag is not set	0x40	2	1/3
JNC rel		0x50	2	1/3
JB bit,rel	Jump if direct bit is set Jump if direct bit	0x20	3	1/3
JNB bit,rel	is not set Jump if direct bit is set and clear	0x30	3	1/3
JBC bit, dir8 rel	bit Jump if less than or equal (signed) if greater than	0x10	3	1/3
JSLE rel	(signed) if less than or Jump if equals	0x08	2	1/3
JSG rel		0x18	2	1/3
JLE rel		0x28	2	1/3
JG rel	Jump if greater than	0x38	2	1/3

JSL rel	Jump if Less than (signed) Jump if	0x48	2	1/3
JSGE rel	Greater or Equal (signed) Jump if Equal Jump if Not	0x58	2	1/3
JE rel	Equal Compare direct byte to A, Jump if not equal	0x68	2	1/3
JNE rel		0x78	2	1/3
CJNE A,dir8 rel		0xB5	3	1/3
CJNE A,#data rel compare immediate with A, jump if not equal		0xB4	3	1/3
CJNE Rn,#data rel compares the immediate value to the register. Jump if not equal		0xB8-0xBF	3	1/3
CJNE @Ri, #data rel compares immediate to indirect memory. If not equal, jump 0xB6-0xB7 register decrement, if not zero, jump			3	1/3
DJNZ Rn,rel	directly byte decrement, if not zero, jump no operation	0xD8-0xDF	2	1/3
DJNZ dir8,rel		0xD5	3	1/3
NOP		0x00	1	1

Note: Jumps without conditions are executed in 1 clock cycle.

special order

mnemonic	describe	Number of encoded byte	clocks
TRAP	Trap Interrupt - Execute as NOP	0xB9	1
ESC		0xA5	1

### A.1.4 Instruction List (Machine Code Sorting)

Note: STC32G uses SOURCE mode

#### BINARY mode

opcode	mnemonic	opcode	mnemonic	opcode	mnemonic
00 H NOP		40 H JC rel		80H SJMP rel	
01 H AJMP addr11		41 H AJMP addr11		C0 H PUSH direct	
02 H LJMP addr16		42 H ORL direct,A		C1 H AJMP addr11	
03 H RR A		43 H ORL direct, #data		C2 H CLR bit	
04 H INC A		44 H ORL A, #data		C3H MOVC A,@A+PC	
05 H INC direct		45 H ORL A,direct		C4 H SWAP A	
06 H INC @R0		46 H ORL A,@R0		84 H DIV AB	
07 H INC @R1		47 H ORL A, @R1		85 H MOV direct, direct	
08 H INC R0		48 H ORL A, R0		C5 H XCH A, direct	
09 H INC R1		49 H ORL A, R1		86 H MOV direct, @R0	
0A H INC R2		4A H ORL A,R2		C6 H XCH A,@R0	
0B H INC R3		4B H ORL A, R3		87 H MOV direct, @R1	
0C H INC R4		4CH ORL A,R4		C7 H XCH A, @R1	
0D H INC R5		4D H ORL A, R5		88 H MOV direct, R0	
0E H INC R6		4E H ORL A, R6		C8 H XCH A,R0	
0F H INC R7		4F H ORL A, R7		89 H MOV direct, R1	
10 H JBC bit,rel		50 H JNC rel		CA H XCH A,R1	
11 H ACALL addr11		51 H ACALL addr11		CB H XCH A,R2	
12 H LCALL addr16		52 H ANL direct,A		CC H XCH A,R3	
13 H RRC A		53 H ANL direct, #data		CD H XCH A,R4	
14 H DEC A		54 H ANL A, #data		CE H XCH A,R5	
15 H DEC direct		55 H ANL A,direct		CF H XCH A,R6	
16 H DEC @R0		56 H ANL A,@R0		D0 H POP direct	
17 H DEC @R1		57 H ANL A,@R1		D1 H ACALL addr11	
18 H DEC R0		58 H ANL A,R0		D2 H SETB bit	
19 H DEC R1		59 H ANL A,R1		D3 H MOVC A,@A+DPTR	
1AH DEC R2		5A H ANL A,R2		D4 H DA A	
1BH DEC R3		5B H ANL A,R3		D5 H DJNZ direct, rel	
1CH DEC R4		5C H ANL A,R4		D6 H XCHD A,@R0	
1DH DEC R5		5D H ANL A,R5		D7 H XCHD A, @R1	
1EH DEC R6		5E H ANL A,R6		D8 H DJNZ R0,rel	
				D9 H DJNZ R1,rel	
				DA H DJNZ R2,rel	
				DB H DJNZ R3,rel	
				DC H DJNZ R4,rel	
				DD H DJNZ R5,rel	
				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
22H RET	62 H XRL direct,A	A2 H MOV C,bit	E2 H MOVX A,@R0
23 HR 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
2CH 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
30H JNB bit.rel	70H 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	B2H CPL bit	F2 H MOVX @R0,A
33 H RLC A	73 H JMP @A+DPTR	B3H 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	78H MOV R0,#data	B8 H CJNE R0,#data,rel	F8 H MOV R0,A
39 H ADDC A,R1	79H MOV R1,#data	B9 H CJNE R1,#data,rel	F9 H MOV R1,A
3AH ADDC A,R2	7AH MOV R2,#data	BA H CJNE R2,#data,rel	FA H MOV R2,A
3BH ADDC A, R3	7BH MOV R3, #data	BB H CJNE R3,#data,rel	FB H MOV R3,A
3CH ADDC A,R4	7CH MOV R4,#data	BC H CJNE R4, #data, rel	FC H MOV R4,A
3DH ADDC A,R5	7DH MOV R5, #data	BD H CJNE R5,#data,rel	FD H MOV R5,A
3EH ADDC A, R6	7EH MOV R6,#data	BE H CJNE R6,#data,rel	FE H MOV R6,A
3FH ADDC A, R7	7FH MOV R7,#data	BF H CJNE R7,#data,rel	FF H MOV R7,A

**SOURCE mode**

opcode	mnemonic	opcode	mnemonic	opcode	mnemonic	opcode	mnemonics
00 H NOP		40 H JC rel		80H 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 CRL direct,A		82 H ANL C,bit		C2 H CLR bit	
03 H RR A		43 H CRL direct, #data		83 H MOVC A,@A+PC		C3H CLR C	
04 H INC A		44 H CRL A, #data		84 H DIV AB		C4 H SWAP A	
05 H INC direct		45 H CRL A,direct		85 H MOV direct, direct		C5 H XCH A, direct	
06H -		46H -		86H -		C6H -	
07H -		47H -		87H -		C7H -	
08 H JSLE rel		48 H JSR rel		88H -		C8H -	
09 H MOV Rm, @WRj+dis		49 H MOV WRj, @WRj+dis		89 H LJMP @WRj EJMP @DRk		C9H -	
0A H MOVZ WRj,Rm		4A H -		8AH EJMP addr24		CA H PUSH op1	
0B H INC R,#short MOV WRj, @DRk		4B H -		8BH -		CB H -	
0CH -		4C H CRL Rm,Rm		8CH DIV Rm,Rm		CC H -	
0D H -		4D H CRL WRj,WRj		8DH DIV WRj,WRj		CD H -	
0E H SRA reg		4E H CRL reg, op2		8EH -		CE H -	
0F H -		4FH -		8FH -		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	
16H -		56H -		96H -		D6H -	
17H -		57H -		97H -		D7H -	
18 H JSR rel		58 H JSRGE rel		98H -		D8H -	
19 H MOV @WRj+dis,Rm		59 H MOV @WRj+dis,WRj		99 H LCALL @WRj ECALL @DRk		D9H -	
1AH MOVS WRj,Rm		5AH -		9AH ECALL addr24		DA H POP op1	
1BH DEC R,#short MOV @DRk, WRj		5BH -		9BH -		DB H -	
1CH -		5CH ANL Rm,Rm		9CH SUB Rm,Rm		DC H -	
1DH -		5DH ANL WRj,WRj		9DH SUB WRj, WRj		DD H -	
1EH SRL reg		5EH ANL reg, op2		9EH SUB reg, op2		DE H -	

1F H -		5F H -		9F H SUB DRk,DRk		DF H -	
20 H J <sub>B</sub> bit.rel		60 H J <sub>Z</sub> 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	
22H RET		62 H XRL direct,A		A2 H MOV C,bit		E2 H MOVX A,@R0	
23 H R <sub>A</sub>		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	
26H -		66H -		A6 H -		E6H -	
27H -		67H -		A7 H -		E7H -	
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		E9H -	
2AH -		6A H -		AA HERET		EA H -	
2BH -		6B H -		AB H -		EB H -	
2CH 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 -	
2EH ADD reg,op2		6E H XRL reg,op2		AE H -		EE H -	
2FH ADD DRk,DRk		6F H -		AF H -		EF H -	
30 H JNB bit.rel		70H 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		B2H CPL bit		F2 H MOVX @R0,A	
33 H RLC A		73 H JMP @A+DPTR		B3H 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	
36H -		76H -		B6H -		F6H -	
37H -		77H -		B7H -		F7H -	
38 H JG rel		78 H JNE rel		B8H -		F8 H -	
39 H MOV @DRk+dis,Rm		79 H MOV @DRk+dis,WRj		B9 H TRAP		F9H -	
3AH -		7AH MOVH DRk,#data16 MOV op1,reg		BA H -		FA H -	
3BH -		7BH -		BB H -		FB H -	
3CH -		7CH MOV Rm,Rm		BC H CMP Rm,Rm		FC H -	
3DH -		7DH MOV WRj,WRj		BD H CMP WRj,WRj		FD H -	
3EH SLL reg		7EH MOV reg,op2		BE H CMP reg,op2		FE H -	
3FH -		7FH MOV DRk,DRk		BF H CMP DRk,DRk		FF H -	

## A.2 Instruction details

### ACALL <addr11>

Function: absolute call

Explanation: ACALL unconditionally calls the subroutine at the specified address. This instruction increments the PC twice to get the address of the next instruction, then push the 16-bit result onto the stack (low byte first) and increment the stack pointer twice. The destination address is 5 incremented by the PC. The high order bits, opcode bits 7 to 5, and the second byte of the instruction are concatenated in sequence. Therefore, the called subroutine must begin in the same 2K program memory block as the first byte of the instruction after ACALL. Does not affect the flag bit.

Affected flags:

CY	AC	OV	N	Z
—	—	—	—	—

Binary mode Source mode

Instruction length: 2

Number of Clocks: 3

Hex: script      script

[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

(PC)	$\ddot{y}(\text{PC}) + 2$
(SP)	$\ddot{y}(\text{SP}) + 1$
((SP))	$\ddot{y}(\text{PC.7:0})$
(SP)	$\ddot{y}(\text{SP}) + 1$
((SP))	$\ddot{y}(\text{PC.15:8})$
(PC.10:0)	$\ddot{y}$ Page address

### ADD <dest>,<src>

Function: The source number is added to the destination operand.

Description: Add the source operand to the destination operand, which can be a register or accumulator, and leave the calculation result in the register or accumulator.

The CY flag is set if bit 7 (CY) has a carry.

If a byte variable is added, and if bit 3 has a carry (AC), the AC flag is set. For addition of unsigned integers,

The CY flag indicates that an overflow has occurred. If the 6th bit carries but the 7th bit does not carry, or the 7th bit has a carry but the 6th bit

If there is no carry, the OV flag is set. When adding signed integers, the OV flag indicates that the sum of the two positive operands has occurred

A negative number, or a positive number appears in the sum of two negative operands. Bits 6 and 7 in this description refer to the highest value of the operand

valid bytes (8, 16, or 32 bits). The result affects the N and Z flags. The allowed addressing modes for the source operand are register,

Direct, register indirect and immediate addressing.

### ADD A,Rn

Command operation: (PC)       $\ddot{y}(\text{PC}) + 1$

(A)       $\ddot{y}(A) + (Rn)$

Affected flags:

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Command code] 28H – 2FH

0 0 1 0 1 rrr	
---------------	--

Binary mode Source mode

Command length: 1                          2  
 Number of Clocks: 1                          1  
**Hex:** script                                  [A5] Script

---

**ADD A,Direct**

Command operation: (PC)                           $\ddot{y}(PC) + 2$   
 (A)     $\ddot{y}(A) + (\text{direct})$   
 Affected flags:                                      

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

  
 [Command code] 25H                                    

001001	101	Direct Address
--------	-----	----------------

  
 Binary mode Source mode  
 Instruction length: 2                                  2  
 Number of Clocks: 1                                  1  
**Hex:** script    script

---

**ADD A, @Ri**

Command operation: (PC)                           $\ddot{y}(PC) + 1$   
 (A)     $\ddot{y}(A) + ((Ri))$   
 Affected flags:                                      

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

  
 [Command code] 26H, 27H                                

001001	1 i
--------	-----

  
 Binary mode Source mode  
 Command length: 1    2  
 Number of Clocks: 1                                  1  
**Hex:** script    [A5] Script

---

**ADD A,#DATA**

Command operation: (PC)                           $\ddot{y}(PC) + 2$   
 (A)     $\ddot{y}(A) + \#data$   
 Affected flags:                                      

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

  
 [command code] 24H                                    

00100100	immediate	
----------	-----------	--

  
 Binary mode Source mode  
 Instruction length: 2                                  2  
 Number of Clocks: 1                                  1  
**Hex:** script    script

---

**ADD Rmd,Rms**

Command operation: (PC)                           $\ddot{y}(PC) + 2$   
 (Rmd)     $\ddot{y}(Rms) + (Rmd)$   
 Affected flags:                                      

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

  
 [Command code] 2CH

0 0 1 0 1	1 0 0 ssss	SSSS	
-----------	------------	------	--

**Binary mode Source mode**

Instruction length: 3      2  
 Number of Clocks: 1      1

**Hex:** [A5] Script Code Script Code**ADD WRjd, WRjs**

Command operation: (PC)	$\ddot{y}(PC) + 2$				
(WRjd)	$\ddot{y}(WRjs) + (WRjd)$				
Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[command code] 2DH

0 0 1 0 1	1 0 1 tttt	TTTT	
-----------	------------	------	--

**Binary mode Source mode**

Instruction length: 3      2  
 Number of Clocks: 1      1

**Hex:** [A5] Script Code Script Code**ADD DRkd, DRks**

Command operation: (PC)	$\ddot{y}(PC) + 2$				
(DRkd)	$\ddot{y}(DRks) + (DRkd)$				
Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[command code] 2FH

0 0 1 0 1	1 1 1	uuuu	UUUU	
-----------	-------	------	------	--

**Binary mode Source mode**

Instruction length: 3      2  
 Number of Clocks: 1      1

**Hex:** [A5] Script Code Script Code**ADD Rm,#DATA**

Command operation: (PC)	$\ddot{y}(PC) + 3$				
(Rm)	$\ddot{y}(Rm) + \#data$				
Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Instruction code] 2E(s)0H

0 0 1 0 1	1 1 0	ssss	0 0 0 0	immediate	
-----------	-------	------	---------	-----------	--

**Binary mode Source mode**

Instruction length: 4      3  
 Number of Clocks: 1      1

**Hex:** [A5] Script Code Script Code**ADD WRj, #DATA16**

Command operation: (PC)	$\ddot{y}(PC) + 4$				
(WRj)	$\ddot{y}(WRj) + \#data16$				
Affected flags:	CY	AC	OV	N	Z

ÿ	-	ÿ	ÿ	ÿ
---	---	---	---	---

[Command code] 2E(t)4H

0 0 1 0 1	1 1 0 ttt 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] Script Code Script Code**ADD DRK,#0DATA16**Command operation: (PC)  $\ddot{y}(PC) + 4$ (DRk)  $\ddot{y}(DRk) + \#data16$ 

Affected flags:	CY	AC	OV	N	Z
	ÿ	-	ÿ	ÿ	ÿ

[Instruction code] 2E(u)8H

0 0 1 0 1	1 1 0 uuuu 0 0 0 immediate high byte immediate low byte		
-----------	---	--	--

**Binary mode Source mode**

Instruction length: 5 4

Number of Clocks: 1 1

**Hex:** [A5] Script Code Script Code**ADD Rm, DIR8**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm) + (\text{dir8})$ 

Affected flags:	CY	AC	OV	N	Z
	ÿ	ÿ	ÿ	ÿ	ÿ

[Command code] 2E(s)1H

0 0 1 0 1	1 1 0 ssss 0 0 0 1 Direct address	
-----------	-----------------------------------	--

**Binary mode Source mode**

Instruction length: 4 3

Number of Clocks: 1 1

**Hex:** [A5] Script Code Script Code**ADD WRj, DIR8**Command operation: (PC)  $\ddot{y}(PC) + 3$ (WRj)  $\ddot{y}(WRj) + (\text{dir8})$ 

Affected flags:	CY	AC	OV	N	Z
	ÿ	-	ÿ	ÿ	ÿ

[Command code] 2E(t)5H

0 0 1 0 1	1 1 0 ttt 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] Script Code Script Code**ADD Rm, DIR16**Command operation: (PC)  $\ddot{y}(PC) + 4$

(Rm) $\ddot{y}(Rm) + (\text{dir16})$				
Affected flags:	CY	AC	OV	N
	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Command code] 2E(s)3H

0 0 1 0 1	1 1 0 sssss 0 0 1	1 address high byte address low byte
-----------	-------------------	--------------------------------------

**Binary mode Source mode**

Instruction length: 5 4

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**ADD WRj, DIR16**Command operation: (PC)  $\ddot{y}(PC) + 4$ 

(WRj) $\ddot{y}(WRj) + (\text{dir16})$				
Affected flags:	CY	AC	OV	N
	$\ddot{y}$	-	$\ddot{y}$	$\ddot{y}$

[Command code] 2E(t)7H

0 0 1 0 1	1 1 0 ttt 0 1	1 1 address high byte address low byte
-----------	---------------	--

**Binary mode Source mode**

Instruction length: 5 4

Number of clocks: 2

**Hex:** [A5] Script Code Script Code**ADD Rm,@DRk**Command operation: (PC)  $\ddot{y}(PC) + 3$ 

(Rm) $\ddot{y}(Rm) + ((DRk))$				
Affected flags:	CY	AC	OV	N
	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Instruction code] 2E(u)B(s)0H

0 0 1 0 1	1 1 0 uuuu 1 0 1	1 ssss 0 0 0 0
-----------	------------------	----------------

**Binary mode Source mode**

Instruction length: 4 3

Number of Clocks: 3

**Hex:** [A5] Script script**ADDC A,<src-byte>**

Function: Adds A to the source operand, adds one (1) if CY is set, and places the result in A.

Description: ADCD simultaneously adds the specified byte variable, the carry flag and the accumulator contents, leaving the result in the accumulator. If the 7th

If the bit or the 3rd bit has a carry, then the carry and auxiliary carry flags are set respectively, otherwise it is cleared. When adding unsigned integers,

The carry flag indicates that an overflow has occurred. If the 6th bit has a carry but the 7th bit does not, or the 7th bit has a carry but the 6th bit

If the bit is not carried, OV is set; otherwise OV is cleared. When adding signed integers, OV indicates the occurrence of the sum of two positive operands

A negative number appears, or a positive number appears in the sum of two negative operands. The N and Z flags are also affected depending on the result. source operand

Four addressing modes are allowed: Register, Direct, Register Indirect, or Literal.

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

**ADDC A,Rn**

Command operation: (PC)  $\ddot{y}(PC) + 1$   
 (A)  $\ddot{y}(A) + (C) + (Rn)$

[Command code] 38H – 3FH

0	0	1	1	rrr
---	---	---	---	-----

Binary mode Source mode

Command length: 1 2  
 Number of Clocks: 1  
 Hex: script [A5] Script

**ADDC A, DIRECT**

Command operation: (PC)  $\ddot{y}(PC) + 2$   
 (A)  $\ddot{y}(A) + (C) + (\text{direct})$

[Command code] 35H

0	0	1	1	0	1	0	1	Direct address
---	---	---	---	---	---	---	---	----------------

Binary mode Source mode

Instruction length: 2 2  
 Number of Clocks: 1  
 Hex: script script

**ADDC A, @Ri**

Command operation: (PC)  $\ddot{y}(PC) + 1$   
 (A)  $\ddot{y}(A) + (C) + ((Ri))$

[Command code] 36H, 37H

0	0	1	1	0	1	i
---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1 2  
 Number of Clocks: 1  
 Hex: script [A5] Script

**ADDC A,#DATA**

Command operation: (PC)  $\ddot{y}(PC) + 2$   
 (A)  $\ddot{y}(A) + (C) + \#data$

[command code] 34H

0	0	1	1	0	0	immediate
---	---	---	---	---	---	-----------

Binary mode Source mode

Instruction length: 2 2  
 Number of Clocks: 1  
 Hex: script script

**AJMP addr11**

Function: absolute jump

Description: AJMP transfers program execution to the specified address at run time by concatenating the upper 5 bits of the PC (incrementing the PC by two times), bits 7 to 5 of the opcode, and the second byte of the instruction. Therefore, the target must be referred to after the AJMP

The first byte of the command is located in the same 2K program memory block.

Affected flags:

CY	AC	OV	N	Z
----	----	----	---	---

—	—	—	—	—
---	---	---	---	---

Command operation: (PC)  $\ddot{y}(PC) + 2$   
 (PC10-0)  $\ddot{y}$ page address  
 [Instruction code] 01H, 21H, 41H, 61H, 81H, A1H, C1H, E1H

a10 a9 a8 0 0 0 1 a7 a6 a5 a4 a3 a2 a1 a0	
---	--

Binary mode Source mode

Instruction length: 2  
 Number of Clocks: 3  
**Hex:** script      script

#### ANL <dest>,<src>

Function: Logical AND of byte operands

Description: Performs a bitwise logical AND operation between the specified variables and stores the result in the destination variable. These two operands allow 10 addressing pattern combination. When the destination is a register or accumulator, the source can use register, direct, register indirect, or immediate Number addressing; when the destination is a direct address, the source number can be an accumulator or an immediate number. The N and Z flags are affected by the result influences.

Note: When this instruction is used to modify an output port, the value used as raw port data is read from the output data latch, while not an input pin.

#### ANL A,Rn

Command operation: (PC)  $\ddot{y}(PC) + 1$   
 (A)  $\ddot{y}(A)$  and (Rn)  
 Affected flags:  

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 58H – 5FH  

0 1 0 1	1 rrr
---------	-------

Binary mode Source mode

Command length: 1  
 Number of Clocks: 1  
**Hex:** script      [A5] Script

#### ANL A,Direct

Command operation: (PC)  $\ddot{y}(PC) + 2$   
 (A)  $\ddot{y}(A)$  and (direct)  
 Affected flags:  

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 55H  

0 1 0 1 0 1 0 1 Direct address
--------------------------------

Binary mode Source mode

Instruction length: 2  
 Number of Clocks: 1  
**Hex:** script      script

#### ANL A, @Ri

Command operation: (PC)  $\ddot{y}(PC) + 1$

(A)	$\ddot{y}(A)$ and $((R_i))$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 56H, 57H

0 1 0 1 0 1	1 i
-------------	-----

Binary mode Source mode

Command length: 1 2

Number of Clocks: 1 1

Hex: script [A5] Script

**ANL A,#DATA**Command operation: (PC)  $\ddot{y}(PC) + 2$ 

(A)	$\ddot{y}(A)$ and #data				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 54H

0 1 0 1 0 1 0	immediate	
---------------	-----------	--

Binary mode Source mode

Instruction length: 2 2

Number of Clocks: 1 1

Hex: script script

**ANL Direct,A**Command operation: (PC)  $\ddot{y}(PC) + 2$ 

(direct)	$\ddot{y}(\text{direct})$ and (A)				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 52H

0 1 0 1 0 0 1 0	Direct Address	
-----------------	----------------	--

Binary mode Source mode

Instruction length: 2 2

Number of Clocks: 1 1

Hex: script script

**ANL Direct, #DATA**Command operation: (PC)  $\ddot{y}(PC) + 3$ 

(direct)	$\ddot{y}(\text{direct})$ and #data				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 53H

0 1 0 1 0 0 1	1	direct address	immediate
---------------	---	----------------	-----------

Binary mode Source mode

Instruction length: 3 3

Number of Clocks: 1 1

Hex: script script

**ANL Rmd, Rms**

Command operation: (PC)

ŷ(PC) + 2

(Rmd)

ŷ(Rms) and (Rmd)

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[Command code] 5CH

0 1 0 1	1 1 0 0	ssss	SSSS	
---------	---------	------	------	--

Binary mode Source mode

Instruction length: 3

2

Number of Clocks: 1

1

Hex: [A5] Script Code Script Code

**ANL WRjd, WRjs**

Command operation: (PC)

ŷ(PC) + 2

(WRjd)

ŷ(WRjs) and (WRjd)

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[command code] 2DH

0 0 1 0 1	1 0 1	tttt T	TTT	
-----------	-------	--------	-----	--

Binary mode Source mode

Instruction length: 3

2

Number of Clocks: 1

1

Hex: [A5] Script Code Script Code

**ANL Rm, #DATA**

Command operation: (PC)

ŷ(PC) + 3

(Rm)

ŷ(Rm) and #data

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[Instruction code] 5E(s)0H

0 1 0 1	1 1 1 0	ssss	0 0 0 0 immediate	
---------	---------	------	-------------------	--

Binary mode Source mode

Instruction length: 4

3

Number of Clocks: 1

1

Hex: [A5] Script Code Script Code

**ANL WRj, #DATA16**

Command operation: (PC)

ŷ(PC) + 4

(WRj)

ŷ(WRj) and #data16

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[Command code] 5E(t)4H

0 1 0 1	1 1 1 0	ttt 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] Script Code Script Code**ANL Rm, DIR8**

Command operation: (PC)

 $\ddot{y}(PC) + 3$ 

(Rm)

 $\ddot{y}(Rm)$  and (dir8)

Affected flags:

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 5E(s)1H

0 1 0 1	1 1 1 0	ssss	0 0 0 1	Direct address	
---------	---------	------	---------	----------------	--

**Binary mode Source mode**

Instruction length: 4

3

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**ANL WRj, DIR8**

Command operation: (PC)

 $\ddot{y}(PC) + 3$ 

(WRj)

 $\ddot{y}(WRj)$  and (dir8)

Affected flags:

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 5E(t)5H

0 1 0 1	1 1 1 0	ttt	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] Script Code Script Code**ANL Rm, DIR16**

Command operation: (PC)

 $\ddot{y}(PC) + 4$ 

(Rm)

 $\ddot{y}(Rm)$  and (dir16)

Affected flags:

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 5E(s)3H

0 1 0 1	1 1 1 0	ssss	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] Script Code Script Code**ANL WRj, DIR16**

Command operation: (PC)

 $\ddot{y}(PC) + 4$ 

(WRj)

 $\ddot{y}(WRj)$  and (dir16)

Affected flags:

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 5E(t)7H

0 1 0 1	1 1 1 0	ttt	0 1	1 address high byte	address low byte
---------	---------	-----	-----	---------------------	------------------

**Binary mode Source mode**

Instruction length: 5 4

Number of clocks: 2

**Hex:** [A5] Script Code Script Code**ANL Rm, @WRj**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm)$  and  $((WRj))$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction code] 5E(t)9(s)0H

0 1 0 1	1 1 1 0 ttt	1 0 0 1		ssss 0 0 0 0	
---------	-------------	---------	--	--------------	--

Binary mode Source mode

Instruction length: 4 3

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**ANL Rm,@DRk**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm)$  and  $((DRk))$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction code] 5E(u)B(s)0H

0 1 0 1	1 1 1 0 uuuu	1 0 1	1	ssss 0 0 0 0	
---------	--------------	-------	---	--------------	--

Binary mode Source mode

Instruction length: 4 3

Number of Clocks: 3

**Hex:** [A5] Script script**ANL C,<src-bit>**

Function: Logical AND of bit operands

Description: If the Boolean value of the source bit is logic 0, the carry flag is cleared; otherwise, the carry flag remains in its current state. assembly language

A slash ("") before the number indicates that the logical complement of the addressed bit is used as the value of the source number, but the source bits themselves are not affected. does not affect

other flags. Only direct bit addressing is allowed as a source operand.

**ANL C,bit51**Command operation: (PC)  $\ddot{y}(PC) + 2$ (A)  $\ddot{y}(C)$  and (bit51)

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[Command code] 82H

1 0 0 0 0 0 1	0 bit address	
---------------	---------------	--

Binary mode Source mode

Instruction length: 2 2

Number of Clocks: 1

**Hex:** script

**ANL C,/bit51**Command operation: (PC)  $\ddot{y}(PC) + 2$ (B)  $\ddot{y}(C)$  and  $/\text{(bit}51)$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[command code] B0H

1	0	1	1	0	0	0	bit address
---	---	---	---	---	---	---	-------------

Binary mode Source mode

Instruction length: 2

Number of Clocks: 1

Hex: script

script

**ANL C,bit**Command operation: (PC)  $\ddot{y}(PC) + 3$ (A)  $\ddot{y}(C)$  and (bit)

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[Command code] A98(y)H

1	0	1	0	1	0	0	1	1	0	0	0	yyy	Direct address
---	---	---	---	---	---	---	---	---	---	---	---	-----	----------------

Binary mode Source mode

Instruction length: 4

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**ANL C,/bit**Command operation: (PC)  $\ddot{y}(PC) + 3$ (A)  $\ddot{y}(C)$  and  $/\text{(bit)}$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[Command code] A9F(y)H

1	0	1	0	1	0	0	1	1	1	1	0	yyy	direct address
---	---	---	---	---	---	---	---	---	---	---	---	-----	----------------

Binary mode Source mode

Instruction length: 4

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**CJNE <dest-byte>,<src-byte>,rel**

Function: Compare, jump if not equal.

Remarks: CJNE compares the size of the first two operands and jumps if their values are not equal. Add PC to last byte of instruction

The signed relative offset of the jump target is calculated, and then the PC is incremented to the beginning of the next instruction. If unsigned integer

If the value of &lt;dest-byte&gt; is less than the value of the unsigned integer &lt;src-byte&gt;, the carry flag is set; otherwise, the carry is cleared. Two fuck

Numbers are not affected. The first two operands allow four addressing mode combinations: the accumulator can be combined with any directly addressed byte or

Or an immediate comparison, any indirect memory location or working register can be compared with an immediate. Affects C, N and Z flags bit.

**CJNE A,Direct,rel**

Command operation:	(PC)	$\ddot{y}(PC) + 3$
if	(A)	$\ddot{y}$ (direct)
then	(PC)	$\ddot{y}(PC) + \text{relative offset}$
if	(A)	< (direct)
then	(C)	$\ddot{y}1$
else	(A)	$\ddot{y}0$

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$

[command code] B5H

1 0 1	1	0 1 0 1	direct address	relative address
-------	---	---------	----------------	------------------

Binary mode Source mode

Instruction length: 3

3

Number of clocks: 1/3

1/3

Hex: script

script

**CJNE A,#DATA,rel**

Command operation:	(PC)	$\ddot{y}(PC) + 3$
if	(A)	$\ddot{y}$ data
then	(PC)	$\ddot{y}(PC) + \text{relative offset}$
if	(A)	< data
then	(C)	$\ddot{y}1$
else	(C)	$\ddot{y}0$

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$

[command code] B4H

1 0 1	1	0 1 0 0	Literal	relative address
-------	---	---------	---------	------------------

Binary mode Source mode

Instruction length: 3

3

Number of clocks: 1/3

1/3

Hex: script

script

**CJNE Rn,#DATA,rel**

Command operation:	(PC)	$\ddot{y}(PC) + 3$
if	(Rn)	$\ddot{y}$ data
then	(PC)	$\ddot{y}(PC) + \text{relative offset}$
if	(Rn)	< data
then	(C)	$\ddot{y}1$
else	(C)	$\ddot{y}0$

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] B8H - BFH

1 0 1	1	1	r r r	Immed	ate relative address
-------	---	---	-------	-------	----------------------

Binary mode Source mode

Instruction length: 3

4

Number of clocks: 1/3

1/3

Hex: script

[A5] Script

**CJNE @Ri,#DATA,rel**

Command operation:	(PC)	$\ddot{y}(PC) + 3$
if	((Ri))	$\ddot{y}$ data

then	(PC)	$\hat{y}(PC) + \text{relative offset}$			
if	((Ri))	< data			
then	(C)	$\hat{y}1$			
else	(C)	$\hat{y}0$			
Affected flags:	CY	AC	OV	N	Z
	$\hat{y}$	-	-	$\hat{y}$	$\hat{y}$

[Command code] B6H, B7H

1	0	1	1	i	immediate relative address
---	---	---	---	---	----------------------------

**Binary mode Source mode**

Instruction length: 3 4

Number of clocks: 1/3 1/3

Hex: script [A5] Script

**CLR A**

Function: clear the accumulator

Description: The accumulator is cleared (all bits set to zero). Affects the N and Z flags.

Command operation: (PC)	$\hat{y}(PC) + 1$				
(A)	$\hat{y}0$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\hat{y}$	$\hat{y}$

[command code] E4H

1	1	0	0	1	0	0
---	---	---	---	---	---	---

**Binary mode Source mode**

Command length: 1 1

Number of Clocks: 1 1

Hex: script script

**CLR bit51**

Function: bit clear

Description: The specified bit is cleared (reset to zero). Other flags are not affected.

Command operation: (PC)	$\hat{y}(PC) + 2$				
bit	$\hat{y}0$				
Affected 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

Number of Clocks: 1 1

Hex: script script

**CLR C**

Function: carry clear

Description: The carry flag is cleared (reset to zero). Other flags are not affected.

Command operation: (PC)	$\hat{y}(PC) + 1$				
(C)	$\hat{y}0$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

ÿ	-	-	-	-
---	---	---	---	---

[Command code] C3H

1	1	0	0	0	1	1
---	---	---	---	---	---	---

**Binary mode Source mode**

Command length: 1

1

Number of Clocks: 1

1

**Hex:** script

script

**CLR bit**

Function: bit clear

Description: The specified bit is cleared (reset to zero). Other flags are not affected.

Command operation: (PC)

ÿ(PC) + 3

(bit)

ÿ0

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] A9C(y)H

1	1	0	1	0	1	1	1	0	0	ÿ	ÿ	Direct address\$	
---	---	---	---	---	---	---	---	---	---	---	---	------------------	--

**Binary mode Source mode**

Instruction length: 4

3

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**CMP <dest>,<src>**

Function: Compare

Description: Subtracts the source operand from the destination operand. The result is not stored in the destination operand. If the 7th bit requires a borrow, then CY (borrow

bit) flag bit is set; otherwise cleared. When subtracting signed integers, the OV flag indicates a negative result when subtracting a negative number from a positive number,

Or to indicate that a positive result occurs when a negative number is subtracted from a positive number. Bit 7 in this description refers to the most significant byte of the operand (8, 16 or

32 bits). AC is only affected by byte operands. The N and Z flags are affected depending on the result. The source operand allows four

Addressing modes: Register, Direct, Literal, and Indirect addressing.

**CMP Rmd, Rms**

Command operation: (PC)

ÿ(PC) + 2

(Rmd)

-(Rms)

Affected flags:

CY	AC	OV	N	Z
ÿ	ÿ	ÿ	ÿ	ÿ

[command code] BCH

1	0	1	1	1	0	0	ssss	SSSS	
---	---	---	---	---	---	---	------	------	--

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**CMP WRjd, WRjs**

Command operation: (PC)

ÿ(PC) + 2

(WRjd)

-(WRjs)

Affected flags:

CY	AC	OV	N	Z

ÿ	-	ÿ	ÿ	ÿ
[script] BDH				
1 0 1      1   1  1 0 1 tttt TTTT				
<b>Binary mode Source mode</b>				
Instruction length: 3      2				
Number of Clocks: 1      1				
<b>Hex: [A5] Script Code Script Code</b>				

**CMP DRkd, DRks**

Command operation: (PC)	ÿ(PC) + 2										
(DRkd)	-(DRks)										
Affected flags:	<table border="1"> <tr> <td style="text-align: center;">CY</td><td style="text-align: center;">AC</td><td style="text-align: center;">OV</td><td style="text-align: center;">N</td><td style="text-align: center;">Z</td></tr> <tr> <td style="text-align: center;">ÿ</td><td style="text-align: center;">-</td><td style="text-align: center;">ÿ</td><td style="text-align: center;">ÿ</td><td style="text-align: center;">ÿ</td></tr> </table>	CY	AC	OV	N	Z	ÿ	-	ÿ	ÿ	ÿ
CY	AC	OV	N	Z							
ÿ	-	ÿ	ÿ	ÿ							
[command code] BFH											
1 0 1      1   1  1 1 1 uuuu UUUU											
<b>Binary mode Source mode</b>											
Instruction length: 3	2										
Number of Clocks: 1	1										
<b>Hex: [A5] Script Code Script Code</b>											

**CMP Rm,#DATA**

Command operation: (PC)	ÿ(PC) + 3										
(Rm)	-#data										
Affected flags:	<table border="1"> <tr> <td style="text-align: center;">CY</td><td style="text-align: center;">AC</td><td style="text-align: center;">OV</td><td style="text-align: center;">N</td><td style="text-align: center;">Z</td></tr> <tr> <td style="text-align: center;">ÿ</td><td style="text-align: center;">ÿ</td><td style="text-align: center;">ÿ</td><td style="text-align: center;">ÿ</td><td style="text-align: center;">ÿ</td></tr> </table>	CY	AC	OV	N	Z	ÿ	ÿ	ÿ	ÿ	ÿ
CY	AC	OV	N	Z							
ÿ	ÿ	ÿ	ÿ	ÿ							
[Instruction code] BE(s)0H											
1 0 1      1   1  1 1 0 ssss 0 0 0 0 immediate											
<b>Binary mode Source mode</b>											
Instruction length: 4	3										
Number of Clocks: 1	1										
<b>Hex: [A5] Script Code Script Code</b>											

**CMP WRj, #DATA16**

Command operation: (PC)	ÿ(PC) + 4										
(WRj)	-#data16										
Affected flags:	<table border="1"> <tr> <td style="text-align: center;">CY</td><td style="text-align: center;">AC</td><td style="text-align: center;">OV</td><td style="text-align: center;">N</td><td style="text-align: center;">Z</td></tr> <tr> <td style="text-align: center;">ÿ</td><td style="text-align: center;">-</td><td style="text-align: center;">ÿ</td><td style="text-align: center;">ÿ</td><td style="text-align: center;">ÿ</td></tr> </table>	CY	AC	OV	N	Z	ÿ	-	ÿ	ÿ	ÿ
CY	AC	OV	N	Z							
ÿ	-	ÿ	ÿ	ÿ							
[Command code] BE(t)4H											
1 0 1      1   1  1 1 0 tttt 0   0 0 immediate high byte immediate low byte											
<b>Binary mode Source mode</b>											
Instruction length: 5	4										
Number of Clocks: 1	1										
<b>Hex: [A5] Script Code Script Code</b>											

**CMPDRk,#0DATA16**

Command operation: (PC)      ÿ(PC) + 4

(DRk)	-#0data16				
Affected flags:	CY	AC	OV	N	Z
	ÿ	-	ÿ	ÿ	ÿ

[Command code] BE(u)8H

1	0	1	1	1	0	uuuu	1	0	0	immediate high byte	immediate low byte	
---	---	---	---	---	---	------	---	---	---	---------------------	--------------------	--

**Binary mode Source mode**

Instruction length: 5      4

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**CMP DRk, #1DATA16**

Command operation: (PC)      ÿ(PC) + 4

(DRk)      -#1data16

(DRk)	-#1data16				
Affected flags:	CY	AC	OV	N	Z
	ÿ	-	ÿ	ÿ	ÿ

[Command code] BE(u)CH

1	0	1	1	1	0	uuuu	1	1	0	0	immediate high byte	immediate low byte
---	---	---	---	---	---	------	---	---	---	---	---------------------	--------------------

**Binary mode Source mode**

Instruction length: 5      4

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**CMP Rm, Dir8**

Command operation: (PC)      ÿ(PC) + 3

(Rm)      -(dir8)

(Rm)	-(dir8)				
Affected flags:	CY	AC	OV	N	Z
	ÿ	ÿ	ÿ	ÿ	ÿ

[Command code] BE(s)1H

1	0	1	1	1	0	ssss	0	0	1	Direct address	
---	---	---	---	---	---	------	---	---	---	----------------	--

**Binary mode Source mode**

Instruction length: 4      3

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**CMP WRj, Dir8**

Command operation: (PC)      ÿ(PC) + 3

(WRj)      -(dir8)

(WRj)	-(dir8)				
Affected flags:	CY	AC	OV	N	Z
	ÿ	-	ÿ	ÿ	ÿ

[Command code] BE(t)5H

1	0	1	1	1	0	ttt	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] Script Code Script Code

**CMP Rm, Dir16**

Command operation: (PC)

ÿ(PC) + 4

(Rm)

-(dir16)

Affected flags:

CY	AC	OV	N	Z
ÿ	ÿ	ÿ	ÿ	ÿ

[Command code] BE(s)3H

1	0	1	1	1	0	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] Script Code Script Code**CMP WRj, Dir16**

Command operation: (PC)

ÿ(PC) + 4

(WRj)

-(dir16)

Affected flags:

CY	AC	OV	N	Z
ÿ	-	ÿ	ÿ	ÿ

[Command code] BE(t)7H

1	0	1	1	1	0	t	t	0	1	1	1	address high byte	address low byte
---	---	---	---	---	---	---	---	---	---	---	---	-------------------	------------------

**Binary mode Source mode**

Instruction length: 4

3

Number of clocks: 1(2 for SFR) 1(2 for SFR)

**Hex:** [A5] Script Code Script Code**CMP Rm, @WRj**

Command operation: (PC)

ÿ(PC) + 3

(Rm)

-((WRj))

Affected flags:

CY	AC	OV	N	Z
ÿ	ÿ	ÿ	ÿ	ÿ

[Instruction code] BE(t)9(s)0H

1	0	1	1	1	0	t	t	1	0	0	1	ssss	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	------	---	---	---

BE(WRj)9(Rm)0

**Binary mode Source mode**

Instruction length: 4

3

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**CMP Rm, @DRk**

Command operation: (PC)

ÿ(PC) + 3

(Rm)

-((DRk))

Affected flags:

CY	AC	OV	N	Z
ÿ	ÿ	ÿ	ÿ	ÿ

[Instruction code] BE(u)B(s)0H

1	0	1	1	1	0	u	u	u	1	0	1	1	ssss	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	------	---	---	---

BE(DRk)B(Rm)0

**Binary mode****Source mode**

---

Instruction length: 4	3
Number of Clocks: 3	3
Hex: [A5] Script	script

---

**CPL A**

Function: Invert the accumulator

Description: Each bit of the accumulator is logically inverted (one by one). Bits that were previously 1 will be changed to 0 and vice versa. Only affects N and Z flag bit.

Command operation: (PC)  $\ddot{y}(PC) + 1$ (A)  $\ddot{y}/(A)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] F4H

1	1	1	1	0	1	0
---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1 1

Number of Clocks: 1 1

Hex: script script

**CPL Bit51**

Function: bit inversion

Description: Invert the specified bit variable. Bits that were previously 1 are changed to zero and vice versa. Other flags are not 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 input pin.

Command operation: (PC)  $\ddot{y}(PC) + 2$ (bit51)  $\ddot{y}/(bit51)$ 

Affected 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

Number of Clocks: 1 1

Hex: script script

**CPL C**

Function: carry inversion

Description: Invert the carry flag bit. Bits that were previously 1 are changed to zero and vice versa.

Command operation: (PC)  $\ddot{y}(PC) + 1$ (C)  $\ddot{y}/(C)$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[Command code] B3H

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1 1

Number of Clocks: 1                            1  
**Hex:** script                                 script

---

**CPL Bit**

Function: bit inversion  
 Description: Invert the specified bit.

Command operation: (PC)                         $\ddot{y}(PC) + 3$   
 (bit)     $\ddot{y}/(bit)$   
 Affected flags:                                    CY    AC    OV    N    Z  

-	-	-	-	-
---	---	---	---	---

  
 [Command code] A9B(y)H  

1	0	1	0	1	0	1	0	y	yyy direct address
---	---	---	---	---	---	---	---	---	--------------------

  
 Binary mode Source mode  
 Instruction length: 4                            3  
 Number of Clocks: 1                               1  
**Hex:** [A5] Script Code Script Code

---

**DA A**

Function: Adjust accumulator after decimal addition  
 Description: DA A adjusts the 8-bit value in the accumulator, which is generated by adding the previous two variables (each variable is in compressed BCD format), generates two 4-digit numbers. Any ADD or ADDC instruction may be used to perform the addition. If accumulator bits 3 to 0 greater than 9 (xxxx1010xxxx1111), or if the AC flag is 1, increment the accumulator by 6 so that the lower nibble is generated Correct BCD number. This internal addition will set the carry flag if the carry from the lower nibble field is passed through all the upper bits, otherwise, the carry flag will not be cleared. If the carry flag is now set, or if the four high-order bits are now more than nine (1010xxxx-1111xxxx), these high bits will be incremented by six, producing the correct BCD number in the high nibble. Likewise, if high A bit carry will set the carry flag, but will not clear the carry bit. Therefore, the carry flag indicates the original two BCD variables whether the sum is greater than 100, allowing for multiple exact decimal additions. OV is not affected. All this happens in one finger within the order period. Essentially, this instruction performs a decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, specifically Depends on initial accumulator and PSW conditions. Affects the C, N and Z flags.

Note: DA A cannot simply convert the hexadecimal number in the accumulator to BCD notation, nor does DA A work in decimal subtraction.

Command operation:                                 $(PC)\ddot{y}(PC) + 3$   
 if     $[(A3-0) > 9] \wedge [(AC = 1)]$   
 then     $(A3-0) \ddot{y} (A3-0) + 6$   
 next if     $[(A7-4) > 9] \wedge [(C = 1)]$   
 then     $(A7-4) \ddot{y} (A7-4) + 6$   
 Affected flags:                                    CY    AC    OV    N    Z  

$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$
------------	---	---	------------	------------

  
 [command code] B4H  

1	0	1	1	0	0
---	---	---	---	---	---

  
 Binary mode Source mode

Command length: 1                                1  
 Number of Clocks: 3                               3  
**Hex:** script                                       script

---

**DEC byte**



-	-	-	ÿ	ÿ
---	---	---	---	---

[Command code] 16H, 17H

0 0 0 1 0 1	1 i
-------------	-----

**Binary mode Source mode**

Command length: 1 2

Number of Clocks: 1

**Hex:** script [A5] Script**DEC <dest>,<src>**

Function: Decreases the target value

Description: Decrements the variable specified by the destination operand by 1, 2, or 4. The raw value 00H will underflow to 0FFH. Affects the N and Z flags.

The decremented value is encoded as follows:

vv	value
00	1
01	2
10	3

**DEC Rm,#short**

Command operation: (PC) ÿ(PC) + 2

(Rm) ÿ(Rm) - #short

Affected flags:	CY	AC	OV	N	Z
	-	-	-	ÿ	ÿ

[Command code] 1B(s)(00v)H

0 0 0 1	1 0 1	1	ssss 0 0 vv	
---------	-------	---	-------------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**DEC WRj,#short**

Command operation: (PC) ÿ(PC) + 2

(WRj) ÿ(WRj) - #short

Affected flags:	CY	AC	OV	N	Z
	-	-	-	ÿ	ÿ

[Instruction code] 1B(t)(01v)H

0 0 0 1	1 0 1	1	tttt 0 1 vv	
---------	-------	---	-------------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**DEC DRk,#short**

Command operation: (PC) ÿ(PC) + 2

(DRk) ÿ(DRk) - #short

Affected flags:	CY	AC	OV	N	Z
	-	-	-	ÿ	ÿ

[Instruction code] 1B(u)(1v)H

0 0 0 1	1 0 1	1 $\mu$ uuu 1	1 vv
---------	-------	---------------	------

Binary mode Source mode

Instruction length: 3 2

Number of Clocks: 1 1

Hex: [A5] Script Code Script Code

**DIV AB**

Remarks: DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator holds the integer of the quotient

Number part, register B holds the integer remainder. The carry and OV flags will be cleared. The N and Z flags are also affected by the result to influence.

Exception: If B was originally placed in 00H, the value returned to the accumulator and the B register is undefined. In addition, the overflow flag Position. The carry flag is cleared in any case. Other flags are undefined.

Command operation: (PC)  $\ddot{y}(PC) + 1$ (A15-8)  $\ddot{y} (A) / (B)$  – Bits 15..8 of the result  $\ddot{y}$ (A7-0)  $(A) / (B)$  – Bits 7..0 of the result

Affected flags:	CY	AC	OV	N	Z
	0	-	$\ddot{y}^*$	$\ddot{y}^*$	$\ddot{y}^*$

\*) – if B was originally put in 00H, then OV=1, N and Z are undefined

[Command code] 84H

1 0 0 0 0 1 0 b
-----------------

Binary mode Source mode

Command length: 1 1

Number of Clocks: 6 6

Hex: script script

**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 flags.

For byte operands (&lt;dest&gt;, &lt;src&gt; = Rmd, Rms), the result is 16 bits. The 8-bit quotient is stored in the high-order word of Rmd In the byte, the 8-bit remainder is stored in the low byte of the word where Rmd is located. For example: put register 1 into 251 (0FBH), register 5 into 18 (12H). After executing the DIV R1 instruction, the R5 register 1 is put into 13 (0DH or 00001101B);

Register 0 is loaded with 17 (11H or 00010001B), because  $251 = (13 \times 18) + 17$ ; the CY and OV bits are also cleared.

The CY flag is cleared. The N flag is set if the most significant bit of the quotient is set. If the quotient is zero, the Z flag position bit.

Exception: If &lt;src&gt; is placed in 00H, the value returned in both operands is undefined; the CY flag is cleared, OV

The flag bit is set, and the remaining flag bits are undefined.

**DIV Rmd,Rms**Command operation: (PC)  $\ddot{y}(PC) + 2$ 

(Rmd) Remainder (Rmd) / (Rms) if &lt;dest&gt; md = 0,2,4,...,14

(Rmd+1) Quotient (Rmd) / (Rms)

(Rmd-1) Remainder (Rmd) / (Rms) if &lt;dest&gt; md = 1,3,5,...,15

(Rmd) Quotient (Rmd) / (Rms)

Affected flags:	CY	AC	OV	N	Z
	0	-	$\ddot{y}^*$	$\ddot{y}^*$	$\ddot{y}^*$

\*) – if the source number was originally put in 0, then OV=1, N and Z are undefined

[Command code] 8CH

1 0 0 0 1	1 0 0 ssss	SSSS	
-----------	------------	------	--

**Binary mode Source mode**

Instruction length: 3      2

Number of Clocks: 6      6

Hex: [A5] Script Code Script Code

**DIV WRjd, WRjs**

Command operation: (PC)	$\ddot{y}(PC) + 2$	
(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 middle. For example, for destination register WR4, suppose the quotient is 1122H and the remainder is 3344H. Then, store the results in these

Register file location: (4)-&gt;0x33, (5)-&gt;0x44, (6)-&gt;0x11, (7)-&gt;0x22.

Affected flags:	CY	AC	OV	N	Z
	0	-	$\ddot{y}^*$	$\ddot{y}^*$	$\ddot{y}^*$

\*) – if the source number was originally put in 0, then OV=1, N and Z are undefined

[command code] 8DH

1 0 0 0 1	1 0 1	tttt TTTT	
-----------	-------	-----------	--

**Binary mode Source mode**

Instruction length: 3      2

Number of clocks: 10      10

Hex: [A5] Script Code Script Code

**DJNZ <byte>,<rel-addr>**

Function: decrement, jump if not zero

Remarks: DJNZ decrements the value at the specified location by 1 and jumps to the address specified by the second operand if the resulting value is not zero. 00H's

The original value will underflow to 0FFH. Only the N and Z flags are affected. Add the PC to the signed relative offset in the last byte of the instruction move to calculate the jump target, then increment the PC to the first byte of the next instruction. The decremented position can be a register or Directly addressed bytes.

Note: When this instruction is used to modify an output port, the value used as raw port data is read from the output data latch, while not an input pin.

**DJNZ Rn,rel**

Command operation:  $(PC)\ddot{y}(PC) + 2$   
                          $(Rn)\ddot{y}(Rn) - 1$   
                         if                           $(Rn) \neq 0$   
                         then                          $(PC)\ddot{y}(PC) + rel$

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Script] D8H - DFH

1 1 0 1	1 rrr relative address
---------	------------------------

**Binary mode Source mode**

Instruction length: 2      3

Number of clocks: 1/3

1/3

**Hex:** script**[A5] Script****DJNZ Direct,rel**

Command operation:

 $\text{y}(PC) + 3$  $(\text{direct})\text{y}(\text{direct}) - 1$ 

if

 $(\text{direct}) \neq 0$ 

then

 $(PC)\text{y}(PC) + \text{rel}$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	y	y

[Command code] D5H

1	1	0	1	0	1	direct address relative address
---	---	---	---	---	---	---------------------------------

**Binary mode Source mode**

Instruction length: 3

4

Number of Clocks: 3

3

**Hex:** script**[A5] Script****ECALL <dest>**

Function: extension call

Description: Call the subroutine at the specified address. This instruction increments the program counter by 4 to generate the address of the next instruction, then increments the 24-bit

The result is pushed onto the stack (high byte first) and the stack pointer is incremented by 3. Then load the 8-bit high word and 16-bit low word of the PC into the

The second, third and fourth bytes of the ECALL instruction. The program continues to execute the instruction at that address. Therefore, subroutines can be

Start anywhere in the 16MB storage space. Does not affect the flag bit.

**ECALL addr24**

Command operation: (PC)

 $\text{y}(PC) + 4$ 

(SP)

 $\text{y}(SP) + 1$ 

((SP))

 $\text{y}(PC.23:16)$ 

(SP)

 $\text{y}(SP) + 1$ 

((SP))

 $\text{y}(PC.15:8)$ 

(SP)

 $\text{y}(SP) + 1$ 

((SP))

 $\text{y}(PC.7:0)$ 

(PC)

 $\text{y}(addr.23:0)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 9AH

1	0	0	1	1	0	addr 23-16	addr 15-8	addr 7-0
---	---	---	---	---	---	------------	-----------	----------

**Binary mode Source mode**

Instruction length: 5

4

Number of Clocks: 3

3

**Hex:** [A5] Script Code Script Code**ECALL @DRk**

Command operation: (PC)

 $\text{y}(PC) + 4$ 

(SP)

 $\text{y}(SP) + 1$ 

((SP))

 $\text{y}(PC.23:16)$

(SP)	$\ddot{y}(SP) + 1$										
((SP))	$\ddot{y}(PC.15:8)$										
(SP)	$\ddot{y}(SP) + 1$										
((SP))	$\ddot{y}(PC.7:0)$										
(PC)	$\ddot{y}((DRk))$										
Affected flags:	<table border="1"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							

[Command code] 99(u)8H

1 0 0 1	1 0 0 1	uuuu	1 0 0 0	
---------	---------	------	---------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 3 3

Hex: [A5] Script Code Script Code

**EJMP <dest>**

Function: extended jump

Description: Unconditional jump by loading the second, third and fourth bytes of the instruction into the 8-bit upper word and 16-bit lower word of the PC

Go to the specified address. Therefore, the target can be anywhere in the entire 16MB memory space. Does not affect the flag bit.

**EJMP addr24**

Command operation: (PC)	$\ddot{y}(addr.23:0)$										
Affected flags:	<table border="1"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							

[command code] 8AH

1 0 0 0	1 0 1 0	addr 23-16	addr 15-8	addr 7-0	
---------	---------	------------	-----------	----------	--

**Binary mode Source mode**

Instruction length: 5 4

Number of Clocks: 3 3

Hex: [A5] Script Code Script Code

**EJMP @DRk**

Command operation: (PC)	$\ddot{y}((DRk))$										
Affected flags:	<table border="1"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							

[Command code] 89(u)8H

1 0 0 0	1 0 0 0	1 uuuu	1 0 0 0	
---------	---------	--------	---------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 3 3

Hex: [A5] Script Code Script Code

**ESC**

Function: switch to the opposite mode

Description: Continue execution of the immediately following instruction in the opposite mode. Apart from the PC, no registers or flags are affected.

Command operation: (PC)	$\ddot{y}(PC) + 1$										
Affected flags:	<table border="1"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							

-	-	-	-	-
---	---	---	---	---

[command code] A5H

1	0	1	0	1	0
---	---	---	---	---	---

Binary mode Source mode

Command length: 1

1

Number of Clocks: 1

1

Hex: script

script

**ERET**

Function: extended return

Description: Pop the 3rd byte, 2nd byte, 1st byte and 0th byte of the PC from the stack in sequence, and decrement the stack pointer by 3. program

Execution continues at the result address, usually the instruction immediately following ECALL. Does not affect the flag bit.

Command operation: (PC.7:0)	~((SP))
(SP)	~(SP) - 1
(PC.15:8)	~((SP))
(SP)	~(SP) - 1
(PC.23:16)	~((SP))
(SP)	~(SP) - 1

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[script] AAH

1	0	1	0	1	0
---	---	---	---	---	---

Binary mode Source mode

Instruction length: 2

1

Number of Clocks: 3

3

Hex: [A5] Script Code Script Code

**INC byte**

Function: Increment

Explanation: INC increments the specified variable by 1. The original value of OFFh will overflow to 00h. Only the N and Z flags are affected. Three addressing modes are allowed:

Register, direct or register indirect addressing.

Note: When this instruction is used to modify an output port, the value used as raw port data is read from the output data latch, while not an input pin.

**INC A**

Command operation: (PC)	~(PC) + 1
(A)	~(A) + 1

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 04H

0	0	0	0	1	0
---	---	---	---	---	---

Binary mode Source mode

Command length: 1

1

Number of Clocks: 1

1

Hex: script

script

**INC Rn**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

(Rn)

 $\ddot{y}(Rn) + 1$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Instruction code] 08H - 0FH

0 0 0 0 1 rrr	
---------------	--

**Binary mode Source mode**

Command length: 1

2

Number of Clocks: 1

1

**Hex:** script**[A5]** Script**INC Direct**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

(direct)

 $\ddot{y}(direct) + 1$ 

Affected 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:** script**script****INC @Ri**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

((Ri))

 $\ddot{y}((Ri)) + 1$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 06H, 07H

0 0 0 0 0 1	1 i
-------------	-----

**Binary mode Source mode**

Command length: 1

2

Number of Clocks: 1

1

**Hex:** script**[A5]** Script**INC <dest>,<src>**

Function: Increment

Description: Increments the specified variable by 1, 2, or 4. The original value 0FFH overflows to 00H. Only the N and Z flags are affected. Encoding of incrementing values

as follows:

vv value	
00	1
01	2
10	3

**INC Rm,#short**

---

Command operation: (PC)	$\ddot{y}(PC) + 2$				
(Rm)	$\ddot{y}(Rm) + \#short$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$
[Instruction code] 0B(s)(00v)H					
	0 0 0 0 1 0 1	1	ssss 0 0 vv		
Binary mode Source mode					
Instruction length: 3	2				
Number of Clocks: 1	1				
Hex: [A5] Script Code Script Code					

---

**INC WRj,#short**

Command operation: (PC)	$\ddot{y}(PC) + 2$				
(WRj)	$\ddot{y}(WRj) + \#short$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$
[Instruction code] 0B(t)(01v)H					
	0 0 0 0 1 0 1	1	tttt 0 1 vv		
Binary mode Source mode					
Instruction length: 3	2				
Number of Clocks: 1	1				
Hex: [A5] Script Code Script Code					

---

**INC DRK,#short**

Command operation: (PC)	$\ddot{y}(PC) + 2$				
(DRk)	$\ddot{y}(DRk) + \#short$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$
[Instruction code] 0B(u)(11v)H					
	0 0 0 0 1 0 1	1	uuuu 1		1 vv
Binary mode Source mode					
Instruction length: 3	2				
Number of Clocks: 1	1				
Hex: [A5] Script Code Script Code					

---

**INC DPTR**

Function: Increment data pointer

Description: Add 1 to the 16-bit data pointer. Perform a 16-bit increment (modulo 2^16); the low byte of the data pointer (DPL) overflows from 0FFH

When it reaches 00H , it will increment to the high-order byte (DPH). High byte (DPH) overflow does not increment the extended data pointer

High word (DPX = DR56). Only the N and Z flags are affected.

Command operation: (PC)	$\ddot{y}(PC) + 1$				
(DPTR)	$\ddot{y}(DPTR) + 1$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$
[command code] A3H					
	1 0 1 0 0 0 1	1			

**Binary mode Source mode**

Command length:	1
Number of Clocks:	1
Hex: script	script

**JB**

Function: Jump if set

Description: If the specified bit is 1, jump to the specified address; otherwise, continue to the next instruction. Add PC to the third byte of the instruction

The signed relative offset to calculate the jump target, then increment the PC to the first byte of the next instruction. No modification is detected bit. Does not affect the flag bit.

**JB Bit51,rel**

Command operation:  
if  $(PC) \ddot{\>} (PC) + 3$   
then  $(bit51) \ddot{\>} \ddot{\>}$   
 $(PC) \ddot{\>} (PC) + rel$

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 20H

0	0	1	0	0	0	0	bit address relative address
---	---	---	---	---	---	---	------------------------------

**Binary mode Source mode**

Instruction length:	3
Number of clocks: 1/3	1/3
Hex: script	script

**JB Bit,rel**

Command operation:  
if  $(PC) \ddot{\>} (PC) + 3$   
then  $(bit) \ddot{\>} \ddot{\>}$   
 $(PC) \ddot{\>} (PC) + rel$

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] A92(y)H

1	0	1	0	1	0	0	1	0	0	y	yy	Direct address relative address	
---	---	---	---	---	---	---	---	---	---	---	----	---------------------------------	--

**Binary mode Source mode**

Instruction length:	4
Number of clocks: 1/3	1/3

Hex: [A5] Script Code Script Code

**JBC**

Function: Jump if set and clear the bit

Description: If the specified bit is 1, jump to the specified address; otherwise, continue to the next instruction. In either case, the specified bit is cleared.

Calculate the jump target by adding the PC to the signed relative offset in the third byte of the instruction, then increment the PC to the next instruction the first byte of . Does not affect the flag bit.

NOTE: When this instruction is used to test the output pins, the value used as raw data will be read from the output data latch, not input pin.

**JBC Bit51,rel**

Command operation:	$(PC) \ddot{y} (PC) + 3$				
if	$(bit51) \ddot{y} \ddot{y}$				
then	$(PC) \ddot{y} (PC) + rel$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 10H

0	0	0	1	0	0	0	0-bit address-relative address
---	---	---	---	---	---	---	--------------------------------

Binary mode Source mode

Instruction length: 3

3

Number of clocks: 1/3

1/3

Hex: script

script

**JBC Bit,rel**

Command operation:	$(PC) \ddot{y} (PC) + 3$				
if	$(bit) \ddot{y} \ddot{y}$				
then	$(PC) \ddot{y} (PC) + rel$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] A91(y)H

1	0	1	0	1	0	0	1	0	yyy Direct address relative address
---	---	---	---	---	---	---	---	---	-------------------------------------

Binary mode Source mode

Instruction length: 5

4

Number of clocks: 1/3

1/3

Hex: [A5] Script Code Script Code

**JC**

Function: Jump if carry is set

Description: If the carry flag is set, jump to the specified address; otherwise, continue to the next instruction. Add PC to the second word of the instruction

The jump destination is calculated from a signed relative offset in the section, and then the PC is incremented twice. The detected bits are not modified. Does not affect the sign bit.

Command operation:	$(PC) \ddot{y} (PC) + 2$				
if	$(C) \ddot{y} \ddot{y}$				
then	$(PC) \ddot{y} (PC) + rel$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 40H

0	1	0	0	0	0	0	Relative address
---	---	---	---	---	---	---	------------------

Binary mode Source mode

Instruction length: 2

2

Number of clocks: 1/3

1/3

Hex: script

script

**JE**

Function: Jump if equal

Description: If the Z flag is set, jump to the specified address; otherwise, continue to the next instruction. Add the PC to the second byte of the instruction

A signed relative offset is calculated to calculate the jump target, then increment the PC twice.

Command operation:  $(PC)\ddot{\wedge}(PC) + 2$

if  $(Z) \ddot{\wedge} \ddot{\wedge}$   
then  $(PC)\ddot{\wedge}(PC) + rel$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 68H

0	1	0	1	0	0	0	Relative address
---	---	---	---	---	---	---	------------------

Binary mode Source mode

Instruction length: 3 2

Number of clocks: 1/3 1/3

Hex: [A5] Script Code Script Code

---

**JG**

Function: jump if greater than

Description: If both the Z flag and the CY flag are cleared, jump to the specified address; otherwise, continue to the next instruction. Add the PC to the finger

Let the signed relative offset in the second byte calculate the jump destination, then increment the PC twice.

Command operation:  $(PC)\ddot{\wedge}(PC) + 2$

if  $(Z=0 \text{ and } C=0) \ddot{\wedge} \ddot{\wedge}$   
then  $(PC)\ddot{\wedge}(PC) + rel$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 38H

0	0	1	1	0	0	0	Relative address
---	---	---	---	---	---	---	------------------

Binary mode Source mode

Instruction length: 3 2

Number of clocks: 1/3 1/3

Hex: [A5] Script Code Script Code

---

**JLE**

Function: Jump if less than or equal to

Description: If both the Z flag and the CY flag are set, jump to the specified address; otherwise, continue to the next instruction. add the PC

A signed relative offset in the second byte of the instruction is used to calculate the jump target, after which the PC is incremented twice.

Command operation:  $(PC)\ddot{\wedge}(PC) + 2$

if  $(Z=1 \text{ and } C=1) \ddot{\wedge} \ddot{\wedge}$   
then  $(PC)\ddot{\wedge}(PC) + rel$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 28H

0	0	1	0	0	0	Relative address
---	---	---	---	---	---	------------------

Binary mode Source mode

Instruction length: 3 2

Number of clocks: 1/3 1/3

Hex: [A5] Script Code Script Code

---

**JMP**

Function: indirect jump

Description: Adds the 8-bit unsigned contents of the accumulator to the 16-bit data pointer and loads the resulting sum into the program counter. This will be the follow-up

The address of the instruction fetch. Perform a 16-bit addition (modulo  $2^{16}$ ): the carry from the lower 8 bits is transferred to the upper bit. accumulator and data pointer

Neither has changed. Does not affect the flag bit.

Command operation: (PC)

$\ddot{y}(A) + (DPTR)$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 73H

0	1	1	0	0	1
---	---	---	---	---	---

Binary mode Source mode

Command length: 1

1

Number of clocks: 1/3

1/3

Hex: script

script

## JNB

Function: Jump if bit is not set

Description: If the specified bit is zero, jump to the specified address; otherwise, continue to the next instruction. Add PC to the third byte of the instruction

The signed relative offset to calculate the jump target, then increment the PC to the first byte of the next instruction. No modification is detected  
bit. Does not affect the flag bit.

## JNB Bit51,rel

Command operation:

$(PC)\ddot{y}(PC) + 3$

if

$(bit51) \ddot{y} \ddot{y}$

then

$(PC)\ddot{y}(PC) + rel$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 30H

0	0	1	1	0	0	0	0-bit address-relative address
---	---	---	---	---	---	---	--------------------------------

Binary mode Source mode

Instruction length: 3

3

Number of clocks: 1/3

1/3

Hex: script

script

## JNB Bit,rel

Command operation:

$(PC)\ddot{y}(PC) + 3$

if

$(bit) \ddot{y} \ddot{y}$

then

$(PC)\ddot{y}(PC) + rel$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] A93(y)H

1	0	1	0	1	0	0	1	1	0	y	yy Direct address relative address
---	---	---	---	---	---	---	---	---	---	---	------------------------------------

Binary mode Source mode

Instruction length: 5

4

Number of clocks: 1/3

1/3

Hex: [A5] Script Code Script Code

## JNC

Function: Jump if carry is not set

Description: If the carry flag is zero, jump to the specified address; otherwise, continue to the next instruction. Add the PC to the second byte of the instruction

A signed relative offset in to calculate the jump target, then increment the PC to the next instruction.

Command operation:

$$(PC)\ddot{y}(PC) + 2$$

if

$$(C) \ddot{y} \ddot{y}$$

then

$$(PC)\ddot{y}(PC) + rel$$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 50H

0	1	0	1	0	0	0	Relative address
---	---	---	---	---	---	---	------------------

Binary mode Source mode

Instruction length: 2

2

Number of clocks: 1/3

1/3

Hex: script

script

## JNE

Function: Jump if not equal

Description: If the Z flag is cleared, jump to the specified address; otherwise, continue to the next instruction. Add PC to the second byte of the instruction

The signed relative offset of , to calculate the jump target, then increment the PC twice.

Command operation:

$$(PC)\ddot{y}(PC) + 2$$

if

$$(Z) \ddot{y} \ddot{y}$$

then

$$(PC)\ddot{y}(PC) + rel$$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 78H

0	1	1	1	0	0	0	Relative address
---	---	---	---	---	---	---	------------------

Binary mode Source mode

Instruction length: 3

2

Number of clocks: 1/3

1/3

Hex: [A5] Script Code Script 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. Add the PC to the instruction second

A signed relative offset in bytes to calculate the jump target, after which the PC is incremented twice.

Command operation:

$$(PC)\ddot{y}(PC) + 2$$

if

$$(A) \ddot{y} 0$$

then

$$(PC)\ddot{y}(PC) + rel$$

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 70H

0	1	1	0	0	0	0	Relative address
---	---	---	---	---	---	---	------------------

Binary mode Source mode

Instruction length: 2

2

Number of clocks: 1/3

1/3

Hex: script

script

**JSG**

Function: jump if greater than (signed)

Description: If the Z flag is cleared and the N flag and the OV flag have the same value, jump to the specified address; otherwise continue next instruction. Add the PC to the signed relative offset in the second byte of the instruction to calculate the jump target, then increment the PC twice.

Command operation:

(PC)  $\ddot{y}$  (PC) + 2  
if                   (Z=0 and N=OV)  
then                (PC)  $\ddot{y}$  (PC) + rel

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 18H

0 0 0 1	1 0 0 0 Relative address
---------	--------------------------

Binary mode Source mode

Instruction length: 3

2

Number of clocks: 1/3

1/3

Hex: [A5] Script Code Script Code

**JSGE**

Function: greater than or equal to jump (signed)

Description: If the N flag and the OV flag have the same value, jump to the specified address; otherwise, continue to the next instruction. put the PC Add the signed relative offset in the second byte of the instruction to calculate the jump target, then increment the PC twice.

Command operation:

(PC)  $\ddot{y}$  (PC) + 2  
if                   (N=OV)  
then                (PC)  $\ddot{y}$  (PC) + rel

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 58H

0 1 0 1	1 0 0 0 Relative address
---------	--------------------------

Binary mode Source mode

Instruction length: 3

2

Number of clocks: 1/3

1/3

Hex: [A5] Script Code Script Code

**JSL**

Function: jump if less than (signed)

Description: If the N flag and the OV flag have different values, jump to the specified address; otherwise, continue to the next instruction. put the PC Add the signed relative offset in the second byte of the instruction to calculate the jump target, then increment the PC twice.

Command operation:

(PC)  $\ddot{y}$  (PC) + 2  
if                   (N  $\ddot{y}$  OV)  
then                (PC)  $\ddot{y}$  (PC) + rel

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 48H

0 1 0 0 1 0 0 0 Relative address
----------------------------------

Binary mode Source mode

Instruction length: 3                          2  
 Number of clocks: 1/3                          1/3  
**Hex:** [A5] Script Code Script Code

**JSLE**

**Function:** Jump if less than or equal to (signed)

**Description:** Jump to the specified address if the Z flag is set or if the N flag and the OV flag have different values; no

Continue to the next instruction. The jump target is calculated by adding the PC to the signed relative offset in the second byte of the instruction, then adding

The PC is incremented twice.

Command operation:	(PC)jy(PC) + 2				
if	(Z=1 or (NyOV))				
then	(PC)jy(PC) + rel				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 08H

0	0	0	0	1	0	0	0	Relative address
---	---	---	---	---	---	---	---	------------------

**Binary mode Source mode**

Instruction length: 3                          2  
 Number of clocks: 1/3                          1/3  
**Hex:** [A5] Script Code Script 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

A signed relative offset in bytes to calculate the jump target, after which the PC is incremented twice. The accumulator is not modified. Does not affect the mark  
 Chi bit.

Command operation:	(PC)jy(PC) + 2				
if	(A) j y				
then	(PC)jy(PC) + rel				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 60H

0	1	1	0	0	0	0	0	Relative address
---	---	---	---	---	---	---	---	------------------

**Binary mode Source mode**

Instruction length: 2                          2  
 Number of clocks: 1/3                          1/3  
**Hex:** script                                  script

**LCALL**

**Function:** long call

**Description:** Call the subroutine at the specified address. This instruction increments the program counter by 3 to generate the address of the next instruction, then increments the 16-bit

The result is pushed onto the stack (low byte first), and the stack pointer is incremented by 2. Then load the upper and lower bytes of the PC into LCALL respectively

The second and third bytes of the instruction. The program continues to execute the instruction at that address. Therefore, subroutines can be stored in the entire 64KB program memory  
 starts anywhere in the memory address space. Does not affect the flag bit.

**LCALL addr16**

Command operation: (PC)	$\ddot{y}(PC) + 3$
(SP)	$\ddot{y}(SP) + 1$
((SP))	$\ddot{y}(PC.7:0)$
(SP)	$\ddot{y}(SP) + 1$
((SP))	$\ddot{y}(PC.15:8)$
(PC)	$\ddot{y}(addr.15:0)$

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 12H

0	0	0	1	0	0	1	0 address high byte address low byte
---	---	---	---	---	---	---	--------------------------------------

Binary mode Source mode

Instruction length: 3	3
Number of Clocks: 3	3
Hex: script	script

**LCALL @WRj**

Command operation: (PC)	$\ddot{y}(PC) + 3$
(SP)	$\ddot{y}(SP) + 1$
((SP))	$\ddot{y}(PC.7:0)$
(SP)	$\ddot{y}(SP) + 1$
((SP))	$\ddot{y}(PC.15:8)$
(PC)	$\ddot{y}((WRj))$

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 99(t)4H

1	0	0	1	1	0	0	1	tttt	0	1	0	0
---	---	---	---	---	---	---	---	------	---	---	---	---

Binary mode Source mode

Instruction length: 3	2
Number of Clocks: 3	3
Hex: [A5] Script Code Script Code	

**LJMP**

Function: long jump

Description: LJMP performs an unconditional jump to the specified by loading the second and third bytes of the instruction into the upper and lower bytes of the PC, respectively. address. Therefore, the target can be anywhere in the entire 64KB program memory address space. Does not affect the flag bit.

**LJMP addr16**

Command operation: (PC)	$\ddot{y}(addr.15:0)$
Affected 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
Number of Clocks: 3	3
Hex: script	script

**LJMP @WRj**Command operation: (PC)  $\ddot{y}((WRj))$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 89(t)4H

1	0	0	1	0	0	1	tttt	0	1	0	0
---	---	---	---	---	---	---	------	---	---	---	---

Binary mode Source mode

Instruction length: 3 2

Number of Clocks: 3

Hex: [A5] Script Code Script Code

**MOV**

Function: handling variables

Description: The variable specified by the second operand is copied to the location specified by the first operand. Source bytes are not affected. no other registers or flags are affected. This is by far the most flexible operation. 24 combinations of source and destination addressing modes are allowed.

**MOV A,Rn**Command operation: (PC)  $\ddot{y}(PC) + 1$ (A)  $\ddot{y}(Rn)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] E8H - EFH

1	1	1	0	1	rr
---	---	---	---	---	----

Binary mode Source mode

Command length: 1 2

Number of Clocks: 1

Hex: script [A5] Script

**MOV A, Direct**Command operation: (PC)  $\ddot{y}(PC) + 2$ (A)  $\ddot{y}(\text{direct})$ 

Note: MOV A, ACC are valid instructions.

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] E5H

1	1	1	0	0	1	0	1	Direct address
---	---	---	---	---	---	---	---	----------------

Binary mode Source mode

Instruction length: 2 2

Number of Clocks: 1

Hex: script script

**MOV A, @Ri**Command operation: (PC)  $\ddot{y}(PC) + 1$ (A)  $\ddot{y}((Ri))$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

-	-	-	-	-
---	---	---	---	---

[Command code] E6H, E7H

1	1	1	0	1	i
---	---	---	---	---	---

Binary mode Source mode

Command length: 1 2

Number of Clocks: 1

Hex: script [A5] Script

**MOV A,#DATA**Command operation: (PC)  $\ddot{y}(PC) + 2$ (A)  $\ddot{y}\#data$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 74H

0	1	1	1	0	1	0	0	immediate
---	---	---	---	---	---	---	---	-----------

Binary mode Source mode

Instruction length: 2 2

Number of Clocks: 1

Hex: script script

**MOV Rn,A**Command operation: (PC)  $\ddot{y}(PC) + 1$ (Rn)  $\ddot{y}(A)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] F8H - FFH

1	1	1	1	1	rrr
---	---	---	---	---	-----

Binary mode Source mode

Command length: 1 2

Number of Clocks: 1

Hex: script [A5] Script

**MOV Rn,Direct**Command operation: (PC)  $\ddot{y}(PC) + 2$ (Rn)  $\ddot{y}(\text{direct})$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] A8H - AFH

1	0	1	0	1	rrr	direct address	
---	---	---	---	---	-----	----------------	--

Binary mode Source mode

Instruction length: 2 3

Number of Clocks: 1

Hex: script [A5] Script

**MOV Rn,#DATA**Command operation: (PC)  $\ddot{y}(PC) + 2$

(Rn)	$\ddot{y}$ #data				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 78H - 7FH

0	1	1	1	rrr direct address
---	---	---	---	--------------------

Binary mode Source mode

Instruction length: 2

3

Number of Clocks: 1

1

Hex: script

[A5] Script

**MOV Direct,A**Command operation: (PC)  $\ddot{y}(PC) + 2$ 

(direct)	$\ddot{y}(A)$				
Affected 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: script

[A5] Script

**MOV Direct, Rn**Command operation: (PC)  $\ddot{y}(PC) + 2$ 

(direct)	$\ddot{y}(Rn)$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 88H - 8FH

1	0	0	0	1	rrr direct address	
---	---	---	---	---	--------------------	--

Binary mode Source mode

Instruction length: 2

3

Number of Clocks: 1

1

Hex: script

[A5] Script

**MOV Direct, Direct**Command operation: (PC)  $\ddot{y}(PC) + 3$ 

(direct)	$\ddot{y}(\text{direct})$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 85H

1	0	0	0	1	0	1	Source Direct Address	Destination Direct Address
---	---	---	---	---	---	---	-----------------------	----------------------------

Binary mode Source mode

Instruction length: 3

3

Number of Clocks: 1

1

Hex: script

script

**MOV Direct, @Ri**

Command operation: (PC)	$\ddot{y}(PC) + 2$										
(direct)	$\ddot{y}((Ri))$										
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							
[Command code]	86H, 87H										
Binary mode	Source mode										
Instruction length:	3										
Number of Clocks:	1										
Hex: script	[A5] Script										

**MOV Direct, #DATA**

Command operation: (PC)	$\ddot{y}(PC) + 2$										
(direct)	$\ddot{y}\#data$										
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							
[Command code]	75H										
Binary mode	Source mode										
Instruction length:	3										
Number of Clocks:	1										
Hex: script	script										

**MOV @Ri,A**

Command operation: (PC)	$\ddot{y}(PC) + 1$										
((Ri))	$\ddot{y}(A)$										
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							
[Command code]	F6H, F7H										
Binary mode	Source mode										
Command length:	2										
Number of Clocks:	1										
Hex: script	[A5] Script										

**MOV @Ri,Direct**

Command operation: (PC)	$\ddot{y}(PC) + 2$										
((Ri))	$\ddot{y}(\text{direct})$										
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							
[Command code]	A6H, A7H										
Binary mode	Source mode										
Instruction length:	3										
Number of Clocks:	1										

**Hex:** script**[A5] Script****MOV @Ri,#DATA**

Command operation: (PC)

 $\ddot{y}(\text{PC}) + 2$  $((\text{Ri}))$  $\ddot{y}(\text{#data})$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 76H, 77H

0	1	1	0	1	1 i immediate value
---	---	---	---	---	---------------------

**Binary mode Source mode**

Instruction length: 2

3

Number of Clocks: 1

1

**Hex:** script**[A5] Script****MOV Rmd, Rms**

Command operation: (PC)

 $\ddot{y}(\text{PC}) + 2$  $((\text{Rmd}))$  $\ddot{y}(\text{Rms})$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 7CH

0	1	1	1	1 0 0 ssss	SSSS	
---	---	---	---	------------	------	--

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**MOV WRjd, WRjs**

Command operation: (PC)

 $\ddot{y}(\text{PC}) + 2$  $((\text{WRjd}))$  $\ddot{y}(\text{WRjs})$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 7DH

0	1	1	1	1 0 1 tttt TTTT	
---	---	---	---	-----------------	--

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**MOV DRkd, DRks**

Command operation: (PC)

 $\ddot{y}(\text{PC}) + 2$  $((\text{DRkd}))$  $\ddot{y}(\text{DRks})$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 7FH

0	1	1	1	1 1 1 uuuu UUUU	
---	---	---	---	-----------------	--

**Binary mode Source mode**

Instruction length: 3 2  
 Number of Clocks: 1  
**Hex:** [A5] Script Code Script Code

---

**MOV Rm,#DATA**

Command operation: (PC)  $\ddot{y}(\text{PC}) + 3$   
 (Rm)  $\ddot{y}\#\text{data}$   
 Affected flags: 

CY	AC	OV	N	Z
-	-	-	-	-

  
 [Instruction code] 7E(s)0H 

0	1	1	1	1	0	s	s	s	0	0	0	immediate		
---	---	---	---	---	---	---	---	---	---	---	---	-----------	--	--

**Binary mode Source mode**

Instruction length: 4 3  
 Number of Clocks: 1  
**Hex:** [A5] Script Code Script Code

---

**MOV WRj, #DATA16**

Command operation: (PC)  $\ddot{y}(\text{PC}) + 4$   
 (WRj)  $\ddot{y}\#\text{data16}$   
 Affected flags: 

CY	AC	OV	N	Z
-	-	-	-	-

  
 [Command code] 7E(t)4H 

0	1	1	1	1	1	0	t	t	0	1	0	0	immediate	high byte immediate	low byte	
---	---	---	---	---	---	---	---	---	---	---	---	---	-----------	---------------------	----------	--

**Binary mode Source mode**

Instruction length: 5 4  
 Number of Clocks: 1  
**Hex:** [A5] Script Code Script Code

---

**MOV DRk,#0DATA16**

Command operation: (PC)  $\ddot{y}(\text{PC}) + 4$   
 (DRk)  $\ddot{y}\#\text{data16}$   
 Affected flags: 

CY	AC	OV	N	Z
-	-	-	-	-

  
 [Command code] 7E(u)8H 

0	1	1	1	1	1	0	u	u	u	1	0	0	immediate	high byte immediate	low byte	
---	---	---	---	---	---	---	---	---	---	---	---	---	-----------	---------------------	----------	--

**Binary mode Source mode**

Instruction length: 5 4  
 Number of Clocks: 1  
**Hex:** [A5] Script Code Script Code

---

**MOV DRk, #1DATA16**

Command operation: (PC)  $\ddot{y}(\text{PC}) + 4$   
 (DRk)  $\ddot{y}\#\text{1data16}$   
 Affected flags: 

CY	AC	OV	N	Z
-	-	-	-	-

  
 [Command code] 7E(u)CH

0	1	1	1	1	0	uu	uu	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] Script Code Script Code**MOV Rm, Dir8**Command operation: (PC)                   $\ddot{y}(PC) + 3$ (Rm)                   $\ddot{y}(dir8)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 7E(s)1H

0	1	1	1	1	0	ss	ss	0	0	0	1	Direct address	
---	---	---	---	---	---	----	----	---	---	---	---	----------------	--

**Binary mode Source mode**

Instruction length: 4                  3  
 Number of Clocks: 1                  1

**Hex:** [A5] Script Code Script Code**MOV WRj, Dir8**Command operation: (PC)                   $\ddot{y}(PC) + 3$ (WRj)                   $\ddot{y}(dir8)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 7E(t)5H

0	1	1	1	1	0	ttt	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] Script Code Script Code**MOV DRk, Dir8**Command operation: (PC)                   $\ddot{y}(PC) + 3$ (DRk)                   $\ddot{y}(dir8)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 7E(u)DH

0	1	1	1	1	0	uu	uu	1		1	0	1	Direct address
---	---	---	---	---	---	----	----	---	--	---	---	---	----------------

**Binary mode Source mode**

Instruction length: 4                  3  
 Number of clocks: 2(4 for SFR)                  2(4 for SFR)

**Hex:** [A5] Script Code Script Code**MOV Rm, Dir16**Command operation: (PC)                   $\ddot{y}(PC) + 4$ (Rm)                   $\ddot{y}(dir16)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

-	-	-	-	-
---	---	---	---	---

[Command code] 7E(s)3H

0	1	1	1	1	0	ssss	0	0	1	1	address high byte	address low byte
---	---	---	---	---	---	------	---	---	---	---	-------------------	------------------

Binary mode Source mode

Instruction length: 5 4

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**MOV WRj, Dir16**Command operation: (PC)  $\ddot{y}(PC) + 4$ (WRj)  $\ddot{y}(dir16)$ 

Affected flags:	CY	AC	OV	N	Z
-	-	-	-	-	-

[Command code] 7E(t)7H

0	1	1	1	1	0	ttt	0	1	1	1	address high byte	address low byte
---	---	---	---	---	---	-----	---	---	---	---	-------------------	------------------

Binary mode Source mode

Instruction length: 5 4

Number of clocks: 2

Hex: [A5] Script Code Script Code

**MOV DRk, Dir16**Command operation: (PC)  $\ddot{y}(PC) + 4$ (DRk)  $\ddot{y}(dir16)$ 

Affected flags:	CY	AC	OV	N	Z
-	-	-	-	-	-

[Command code] 7E(u)FH

0	1	1	1	1	0	uuu	1	1	1	1	address high byte	address low byte
---	---	---	---	---	---	-----	---	---	---	---	-------------------	------------------

Binary mode Source mode

Instruction length: 5 4

Number of clocks: 2

Hex: [A5] Script Code Script Code

**MOV Rm, @WRj**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}((WRj))$ 

Affected flags:	CY	AC	OV	N	Z
-	-	-	-	-	-

[Instruction code] 7E(t)9(s)0H

0	1	1	1	1	0	ttt	1	0	0	1	ssss	0	0	0
---	---	---	---	---	---	-----	---	---	---	---	------	---	---	---

Binary mode Source mode

Instruction length: 4 3

Number of clocks: 2

Hex: [A5] Script Code Script Code

**MOV Rm,@DRk**Command operation: (PC)  $\ddot{y}(PC) + 3$

(Rm)	$\ddot{y}((DRk))$							
Affected flags:	CY	AC	OV	N	Z			
	-	-	-	-	-			
[Instruction code] 7E(u)B(s)0H								
Binary mode			Source mode					
Instruction length: 4	3							
Number of Clocks: 3	3							
Hex: [A5] Script	script							

**MOV WRjd, @WRjs**

Command operation: (PC)	$\ddot{y}(PC) + 3$							
(WRjd)	$\ddot{y}((WRjs))$							
Affected flags:	CY	AC	OV	N	Z			
	-	-	-	-	-			
[Instruction code] 0B(u)8(s)0H								
Binary mode			Source mode					
Instruction length: 4	3							
Number of Clocks: 1	1							
Hex: [A5] Script	script							

**MOV WRj, @DRk**

Command operation: (PC)	$\ddot{y}(PC) + 3$							
(WRj)	$\ddot{y}((DRk))$							
Affected flags:	CY	AC	OV	N	Z			
	-	-	-	-	-			
[Instruction code] 0B(u)A(t)0H								
Binary mode			Source mode					
Instruction length: 4	3							
Number of Clocks: 4	4							
Hex: [A5] Script	Script Code							

**MOV Dir8,Rm**

Command operation: (PC)	$\ddot{y}(PC) + 3$							
(dir8)	$\ddot{y}(Rm)$							
Affected flags:	CY	AC	OV	N	Z			
	-	-	-	-	-			
[Command code] 7A(s)1H								
Binary mode			Source mode					
Instruction length: 4	3							
Number of Clocks: 1	1							
Hex: [A5] Script	Script Code							

**MOV Dir8, WRj**

Command operation: (PC)

 $\ddot{y}(PC) + 3$ 

(dir8)

 $\ddot{y}(WRj)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Instruction code] 7A(t)5H

0	1	1	1	0	1	0	ttt	0	1	0	1	Direct address	
---	---	---	---	---	---	---	-----	---	---	---	---	----------------	--

Binary mode Source mode

Instruction length: 4

3

Number of clocks: 2

2

Hex: [A5] Script Code Script Code

**MOV Dir8,DRk**

Command operation: (PC)

 $\ddot{y}(PC) + 3$ 

(dir8)

 $\ddot{y}(DRk)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 7A(u)DH

0	1	1	1	0	uuuu	1		1	0	1	Direct address	
---	---	---	---	---	------	---	--	---	---	---	----------------	--

Binary mode Source mode

Instruction length: 4

3

Number of clocks: 2

2

Hex: [A5] Script Code Script Code

**MOV Dir16,Rm**

Command operation: (PC)

 $\ddot{y}(PC) + 4$ 

(dir16)

 $\ddot{y}(Rm)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 7A(s)3H

0	1	1	1	0	ssss	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] Script Code Script Code

**MOV Dir16, WRj**

Command operation: (PC)

 $\ddot{y}(PC) + 4$ 

(dir16)

 $\ddot{y}(WRj)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 7A(t)7H

0	1	1	1	0	tttt	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] Script Code Script Code**MOV Dir16,DRk**

Command operation: (PC)

 $\ddot{y}(PC) + 4$ 

((dir16))

 $\ddot{y}(DRk)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 7A(u)FH

0	1	1	1	0	1	0	uuuu	1	1	1	address high byte	address low byte
---	---	---	---	---	---	---	------	---	---	---	-------------------	------------------

**Binary mode Source mode**

Instruction length: 5

4

Number of clocks: 2

2

**Hex:** [A5] Script Code Script Code**MOV @WRj,Rm**

Command operation: (PC)

 $\ddot{y}(PC) + 3$ 

((WRj))

 $\ddot{y}(Rm)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Instruction code] 7A(t)9(s)0H

0	1	1	1	0	1	0	tttt	1	0	0	1	ssss	0	0	0
---	---	---	---	---	---	---	------	---	---	---	---	------	---	---	---

**Binary mode Source mode**

Instruction length: 4

3

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**MOV @DRk,Rm**

Command operation: (PC)

 $\ddot{y}(PC) + 3$ 

((DRk))

 $\ddot{y}(Rm)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Instruction code] 7A(u)B(s)0H

0	1	1	1	0	1	0	uuuu	1	0	1	1	ssss	0	0	0
---	---	---	---	---	---	---	------	---	---	---	---	------	---	---	---

**Binary mode Source mode**

Instruction length: 4

3

Number of Clocks: 4

4

**Hex:** [A5] Script

script

**MOV @WRjd,WRjs**

Command operation: (PC)

 $\ddot{y}(PC) + 3$ 

((WRjd))

 $\ddot{y}(WRjs)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Instruction code] 1B(t)8(T)0H

0	0	0	1	1	0	1	1	tttt	1	0	0	0	T	T	T	T
---	---	---	---	---	---	---	---	------	---	---	---	---	---	---	---	---

**Binary mode Source mode**



0 0 1 0 1 0 0 1		ssssuuuu offset	high byte offset	low byte	
-----------------	--	-----------------	------------------	----------	--

Binary mode Source mode

Instruction length: 5 4

Number of Clocks: 3

Hex: [A5] Script Code Script Code

**MOV WRj, @DRk+dis**Command operation: (PC)  $\ddot{y}(PC) + 4$ ((WRj))  $\ddot{y}((DRk)+dis)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 69H

0 1	1 0 1 0	0 1 ttttuuuu offset	high byte offset	low byte		
-----	---------	---------------------	------------------	----------	--	--

Binary mode Source mode

Instruction length: 5 4

Number of Clocks: 4

Hex: [A5] Script Code Script Code

**MOV @WRj+dis,Rm**Command operation: (PC)  $\ddot{y}(PC) + 4$ ((WRj)+dis)  $\ddot{y}(Rm)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 19H

0 0 0 1	1 0 0 1 sssstttt	offset high byte	offset low byte		
---------	------------------	------------------	-----------------	--	--

Binary mode Source mode

Instruction length: 5 4

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**MOV @WRjd+dis,WRjs**Command operation: (PC)  $\ddot{y}(PC) + 4$ ((WRjd)+dis)  $\ddot{y}(WRjs)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] 59H

0 1 0 1	1 0 0 1 TTTT t		t	t	t offset high byte	offset low byte
---------	----------------	--	---	---	--------------------	-----------------

Binary mode Source mode

Instruction length: 5 4

Number of Clocks: 3

Hex: [A5] Script Code Script Code

**MOV @DRk+dis,Rm**Command operation: (PC)  $\ddot{y}(PC) + 4$ ((DRk)+dis)  $\ddot{y}(Rm)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

-	-	-	-	-
---	---	---	---	---

[command code] 39H

0	0	1	1	1 0 0 1 sssuuuu offset high byte offset low byte		
---	---	---	---	--	--	--

**Binary mode Source mode**

Instruction length: 5 4

Number of Clocks: 4

**Hex:** [A5] Script Code Script Code**MOV @DRk+dis,WRj**Command operation: (PC)  $\ddot{y}(PC) + 4$ ((DRk)+dis)  $\ddot{y}(WRj)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 79H

0	1	1	1	1 0 0 1 tttuuuu offset high byte offset low byte		
---	---	---	---	--	--	--

**Binary mode Source mode**

Instruction length: 5 4

Number of Clocks: 6

**Hex:** [A5] Script Code Script Code**MOV <dest-bit>,<src-bit>**

Function: carry bit data

Description: The Boolean variable specified by the second operand (directly addressable bit) is copied into the carry flag. no other registers or The operator flag is affected.

**MOV C, Bit51**Command operation: (PC)  $\ddot{y}(PC) + 2$ (C)  $\ddot{y}(bit51)$ 

Affected flags:

CY	AC	OV	N	Z
$\ddot{y}$	-	-	-	-

[command code] A2H

1	0	1	0	0	0	1	0 -bit address	
---	---	---	---	---	---	---	----------------	--

**Binary mode Source mode**

Instruction length: 2 2

Number of Clocks: 1 1

**Hex:** script script**MOV C, Bit**Command operation: (PC)  $\ddot{y}(PC) + 3$ (C)  $\ddot{y}(bit)$ 

Affected flags:

CY	AC	OV	N	Z
$\ddot{y}$	-	-	-	-

[Command code] A9A(y)H

1	0	1	0	1	0	0	1	1	1 0 1 0 yyyy direct address	
---	---	---	---	---	---	---	---	---	-----------------------------	--

**Binary mode Source mode**

Instruction length: 4 3

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**MOV Bit51,C**

Command operation: (PC)

ŷ(PC) + 2

(bit51)

ŷ(C)

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] 92H

1	0	0	1	0	0	1	0-bit address	
---	---	---	---	---	---	---	---------------	--

**Binary mode Source mode**

Instruction length: 2

2

Number of Clocks: 1

1

**Hex:** script

script

**MOV Bit, C**

Command operation: (PC)

ŷ(PC) + 3

(bit)

ŷ(C)

Affected flags:

CY	AC	OV	N	Z
ŷ	-	-	-	-

[Command code] A99(y)H

1	0	1	0	0	1	1	0 0 1 yyyy direct address	
---	---	---	---	---	---	---	---------------------------	--

**Binary mode Source mode**

Instruction length: 4

3

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**MOV DPTR, #DATA16**

Function: data pointer to load 16-bit constant

Description: The data pointer loads the specified 16-bit constant. 16-bit constants are loaded into the second and third bytes of the instruction. second byte

(DPH) is the high-order byte, while the third byte (DPL) is the low-order byte. Does not affect the flag bit.

Command operation: (PC)

ŷ(PC) + 3

DPH

ŷ immediate 15..8

DPL

ŷ immediate 7..0

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 90H

1	0	0	1	0	0	0	immediate high byte immediate low byte	
---	---	---	---	---	---	---	--	--

**Binary mode Source mode**

Instruction length: 3

3

Number of Clocks: 1

1

**Hex:** script

script

**MOVC**

Function: carry code bytes

Description: The MOVC instruction loads code bytes or constants from program memory into the accumulator. The fetched byte address is the original None

The sum of the contents of the signed 8-bit accumulator and the contents of the 16-bit base register, which can be either a data pointer or a PC. In the latter case, the PC is incremented to the address of the next instruction plus the accumulator; otherwise, the base register is not changed. Change. A 16-bit addition is performed, so the carry from the lower 8 bits can be passed to the upper bit. Does not affect the flag bit.

**MOVC A,@A+DPTR**

Command operation: (PC)	$\ddot{y}(PC) + 1$										
(A)	$\ddot{y}((A) + (DPTR))$										
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							

[Command code] 93H

1 0 0 1 0 0 1	1
---------------	---

**Binary mode Source mode**

Command length: 1

1

Number of Clocks: 4

4

**Hex:** script

script

**MOVC A,@A+PC**

Command operation: (PC)	$\ddot{y}(PC) + 1$										
(A)	$\ddot{y}((A) + (PC))$										
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							

[Command code] 83H

1 0 0 0 0 0 1	1
---------------	---

**Binary mode Source mode**

Command length: 1

1

Number of Clocks: 3

3

**Hex:** script

script

**MOVH DRk, #DATA16**

Function: Move the 16-bit immediate data to the high word of the dword (double word) register.

Description: Move a 16-bit immediate value to the high word of a double word (32-bit) register. The low-order word of the double-word register is unchanged.

Command operation: (PC)	$\ddot{y}(PC) + 4$										
(DRk).31-16	$\ddot{y}\#data16$										
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							

[Command code] 7A(u)CH

0 1	1	1	1 0 1 0	uuuu	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] Script Code Script Code**MOVS WRj,Rm**

Function: Move 8-bit register to 16-bit register with sign extension

Description: Move 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 filled with sign extension, the sign is obtained from the most significant bit of the 8-bit source register.

Command operation: (PC)	$\ddot{y}(PC) + 2$
(WRj).7-0	$\ddot{y}(Rm)$
(WRj).15-8	$\ddot{y}$ Sign extension
Affected flags:	
CY	AC
	OV
-	N
-	Z
[command code] 1AH	
0 0 0 1	1 0 1 0 ttttssss
<b>Binary mode Source mode</b>	
Instruction length: 3	2
Number of Clocks: 1	1
<b>Hex: [A5] Script Code Script Code</b>	

**MOVX**

Function: carry external

Explanation: The MOVX instruction transfers data between the accumulator and a byte of external data memory, so MOV is appended with an X. there are two types of instructions differ in whether they provide 8-bit or 16-bit indirect addresses to external data memory. in the first type , the contents of the current register bank 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)	$\ddot{y}(PC) + 1$
(A)	$\ddot{y}((Ri))$
Affected flags:	
CY	AC
	OV
-	N
-	Z
[Command code] E2H, E3H	
1 1 1 0 0 0 1 i	
<b>Binary mode Source mode</b>	
Command length: 1	1
Number of clocks: 3*	3*
<b>Hex: script</b>	<b>script</b>

**MOVX A,@DPTR**

Command operation: (PC)	$\ddot{y}(PC) + 1$
(A)	$\ddot{y}((DPTR))$
Affected flags:	
CY	AC
	OV
-	N
-	Z
[command code] E0H	
1 1 1 0 0 0 0 0	
<b>Binary mode Source mode</b>	
Command length: 1	1
Number of clocks: 2*	2*
<b>Hex: script</b>	<b>script</b>

**MOVX @Ri,A**

Command operation: (PC)	$\ddot{y}(PC) + 1$
((Ri))	$\ddot{y}(A)$

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] F2H, F3H

1	1	1	1	0	0	1	i
---	---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1

1

Number of clocks: 3\*

3\*

Hex: script

script

**MOVX @DPTR,A**Command operation: (PC)  $\ddot{y}(PC) + 1$ ((DPTR))  $\ddot{y}(A)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[instruction code] F0H

1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1

1

Number of clocks: 3\*

3\*

Hex: script

script

**MOVZ WRj,Rm**

Function: move 8-bit register to 16-bit register and extend zero

Description: Move 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)  $\ddot{y}(PC) + 2$ (WRj).7-0  $\ddot{y}(Rm)$ (WRj).15-8  $\ddot{y}0$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 0AH

0	0	0	0	1	0	ttttssss		
---	---	---	---	---	---	----------	--	--

Binary mode Source mode

Instruction length: 3

2

Number of Clocks: 1

1

Hex: [A5] Script Code Script Code

**MUL**

Function: Multiply

Description: Multiplies the unsigned integer in the source register with the unsigned integer in the destination register. Only register addressing is allowed. for 8

Bit operand, the result is 16 bits. The most significant byte of the result is stored in the low byte of the word in which the destination register is located. least effective

Bytes are stored in the immediately following byte register. The OV flag is set if the product is greater than 255 (0FFH); otherwise it is cleared zero. For 16-bit operands, the result is 32 bits. The most significant word is stored in the low-order word of the double word where the destination register is located.

The least significant word is stored in the immediately following word register. In this operation, if the product is greater than 0FFFFH, the OV flag position bit, otherwise cleared. The CY flag is always cleared. The N flag bit is set when the most significant byte of the result is set. when the result is The Z flag bit is set at zero hour.

**MUL AB**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

(A)

 $\ddot{y}(A) \times (B)$ 

- result bits 7..0

(B)

 $\ddot{y}(A) \times (B)$ 

- result bits 15..8

Affected flags:

CY	AC	OV	N	Z
0	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[command code] A4H

1	0	1	0	0
---	---	---	---	---

**Binary mode Source mode**

Command length: 1

1

Number of Clocks: 1

1

**Hex:** script

script

**MUL Rmd, Rms**

Command operation:

 $(PC)\ddot{y}(PC) + 2$ 

if

&lt;dest&gt;md = 0,2,4..,14

Rmd $\ddot{y}$ Rmd  $\times$  Rms high byteRmd+1 $\ddot{y}$ Rmd  $\times$  Rms low byte

if

&lt;dest&gt;md = 1,3,5..,15

Rmd-1 $\ddot{y}$ Rmd  $\times$  Rms high byteRmd $\ddot{y}$ Rmd  $\times$  Rms low byte

Affected flags:

CY	AC	OV	N	Z
0	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[command code] ACH

1	0	1	0	0	ssss	SSSS	
---	---	---	---	---	------	------	--

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**MUL WRjd, WRjs**

Command operation:

 $(PC)\ddot{y}(PC) + 2$ 

if

&lt;dest&gt;jd = 0,4,8..,28

WRjd $\ddot{y}$ WRjd  $\times$  High byte of WRjsWRjd+2 $\ddot{y}$ WRjd  $\times$  Lower byte of WRjs

if

&lt;dest&gt;jd = 2,6,10..,30

WRjd-2 $\ddot{y}$ WRjd  $\times$  High byte of WRjsWRjd $\ddot{y}$ WRjd  $\times$  Lower byte of WRjs

Affected flags:

CY	AC	OV	N	Z
0	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[script] ADH

1	0	1	0	1	ttt	T	TTT
---	---	---	---	---	-----	---	-----

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code

**NOP**

Function: no operation

Description: Continue to execute the next command. Except for the PC, no registers or flags are affected.

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[command code] 00H

0	0	0	0	0	0	0
---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1

1

Number of Clocks: 1

1

Hex: script

script

**ORL**

Function: Logical OR of variables

Description: Performs a bitwise logical OR operation between the specified variables, storing the result in the destination operand. The destination operand can be a register,

accumulator or direct address. These two operands allow 12 combinations of addressing modes. When the destination is an accumulator, the source number can be Register, direct, register indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate number. When the destination is a register, the source can be a register, immediate, direct and indirect addressing. Only the N and Z flags are affected.

Note: When this instruction is used to modify an output port, the value used as raw port data is read from the output data latch, while not an input pin.

**ORL A,Rn**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

(A)

 $\ddot{y}(A)$  or  $(R_n)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 48H - 4FH

0	1	0	0	1	rrr
---	---	---	---	---	-----

Binary mode Source mode

Command length: 1

2

Number of Clocks: 1

1

Hex: script

[A5] Script

**ORL A, Dir**

Command operation: (PC)

 $\ddot{y}(PC) + 2$ 

(A)

 $\ddot{y}(A)$  or (direct)

Affected flags:

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 45H

0	1	0	0	0	0	1	0	1	Direct	address
---	---	---	---	---	---	---	---	---	--------	---------

Binary mode Source mode

Instruction length: 2

2

Number of Clocks: 1

1

Hex: script

script

**ORL A, @Ri**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ (A)  $\ddot{y}(A)$  or  $((Ri))$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 46H, 47H

0 1 0 0 0 1	1 i
-------------	-----

**Binary mode Source mode**

Command length: 1

2

Number of Clocks: 1

1

**Hex:** script

[A5] Script

**ORL A,#DATA**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ (A)  $\ddot{y}(A)$  or #data

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 44H

0 1 0 0 0 1 0	0 immediate	
---------------	-------------	--

**Binary mode Source mode**

Instruction length: 2

2

Number of Clocks: 1

1

**Hex:** script

script

**ORL Dir,A**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

(direct)

 $\ddot{y}(\text{direct})$  or (A)

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 42H

0 1 0 0 0 0 1 0	Direct Address	
-----------------	----------------	--

**Binary mode Source mode**

Instruction length: 2

2

Number of Clocks: 1

1

**Hex:** script

script

**ORL Dir, #DATA**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

(direct)

 $\ddot{y}(\text{direct})$  or #data

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 43H

0 1 0 0 0 0 1	1	direct address	immediate
---------------	---	----------------	-----------

**Binary mode Source mode**

Instruction length: 3

3

Number of Clocks: 1

1

**Hex:** script

script

**ORL Rmd,Rms**

Command operation: (PC)

ÿ(PC) + 2

(Rmd)

ÿ(Rms) or (Rmd)

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[Command code] 4CH

0 1 0 0 1	1 0 0 ssss	SSSS	
-----------	------------	------	--

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**ORL WRjd, WRjs**

Command operation: (PC)

ÿ(PC) + 2

(WRjd)

ÿ(WRjs) or (WRjd)

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[command code] 4DH

0 1 0 0 1	1 0 1 tttt		SSS	
-----------	------------	--	-----	--

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**ORL Rm,#DATA**

Command operation: (PC)

ÿ(PC) + 3

(Rm)

ÿ(Rm) or #data

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[Instruction code] 4E(s)0H

0 1 0 0 1	1 1 0 ssss	0 0 0 0 immediate		
-----------	------------	-------------------	--	--

**Binary mode Source mode**

Instruction length: 4

3

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**ORL WRj, #DATA16**

Command operation: (PC)

ÿ(PC) + 4

(WRj)

ÿ(WRj) or #data16

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[Command code] 4E(t)4H

0 1 0 0 1	1 1 0 tttt	0 0 0 immediate	high byte immediate	low byte	
-----------	------------	-----------------	---------------------	----------	--

**Binary mode Source mode**

Instruction length: 4

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**ORL Rm, Dir8**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm)$  or (dir8)

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 4E(s)1H

0 1 0 0 1	1 1 0	ssss	0 0 0 1	Direct address	
-----------	-------	------	---------	----------------	--

**Binary mode Source mode**

Instruction length: 4

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**ORL WRj, Dir8**Command operation: (PC)  $\ddot{y}(PC) + 3$ (WRj)  $\ddot{y}(WRj)$  or (dir8)

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 4E(t)5H

0 1 0 0 1	1 1 0	ttt	0 1 0 1	direct address	
-----------	-------	-----	---------	----------------	--

**Binary mode Source mode**

Instruction length: 4

Number of clocks: 1(2 for SFR) 1(2 for SFR)

**Hex:** [A5] Script Code Script Code**ORL Rm, Dir16**Command operation: (PC)  $\ddot{y}(PC) + 4$ (Rm)  $\ddot{y}(Rm)$  or (dir16)

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 4E(s)3H

0 1 0 0 1	1 1 0	ssss	0 0 1	1 address high byte	address low byte
-----------	-------	------	-------	---------------------	------------------

**Binary mode Source mode**

Instruction length: 5

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**ORL WRj, Dir16**Command operation: (PC)  $\ddot{y}(PC) + 4$ (WRj)  $\ddot{y}(WRj)$  or (dir16)

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 4E(t)7H

0 1 0 0 1	1 1 0 ttt	0 1	1 1 address high byte address low byte
-----------	-----------	-----	--

**Binary mode Source mode**

Instruction length: 5 4

Number of clocks: 2

**Hex:** [A5] Script Code Script Code**ORL Rm, @WRj**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm)$  or  $((WRj))$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction code] 4E(t)9(s)0H

0 1 0 0 1	1 1 0 ttt		1 0 0 1	ssss 0 0 0 0	
-----------	-----------	--	---------	--------------	--

**Binary mode Source mode**

Instruction length: 4 3

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**ORL Rm,@DRk**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm)$  or  $((DRk))$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction code] 4E(u)B(s)0H

0 1 0 0 1	1 1 0 uu	u U 1 0 1	1	ssss 0 0 0 0	
-----------	----------	-----------	---	--------------	--

**Binary mode Source mode**

Instruction length: 4 3

Number of Clocks: 3

**Hex:** [A5] Script script**ORL CY,<src-bit>**

Function: logical OR of bit variables

Description: If the boolean value is logic 1, the carry flag is set; otherwise, the carry remains in its current state. Slash before operand in assembly language

The dash ("") indicates that the logical complement of the addressed bit is used as the value of the source number, but the source bits themselves are not affected. Other flags are not affected.

**ORL C, Bit51**Command operation: (PC)  $\ddot{y}(PC) + 2$ (C)  $\ddot{y}(C)$  or (bit51)

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[command code] 72H

0 1	1 1	0 0 1 0-bit address
-----	-----	---------------------

**Binary mode Source mode**

Instruction length: 2 2

Number of Clocks: 1

**Hex:** script      **script**

---

**ORL C,/Bit51**

Command operation: (PC)       $\ddot{y}(PC) + 2$

(C)       $\ddot{y}(C)$  or  $/bit51$

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[Command code] A0H

1	0	1	0	0	0	0	bit address	
---	---	---	---	---	---	---	-------------	--

Binary mode **Source** mode

Instruction length: 2      2

Number of Clocks: 1      1

**Hex:** script      **script**

---

**ORL C,bit**

Command operation: (PC)       $\ddot{y}(PC) + 2$

(C)       $\ddot{y}(C)$  or (bit)

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[Command code] A97(y)H

1	0	1	0	0	1	0	1	0	y	yyy	direct address
---	---	---	---	---	---	---	---	---	---	-----	----------------

Binary mode **Source** mode

Instruction length: 4      3

Number of Clocks: 1      1

**Hex:** [A5] Script Code Script Code

---

**ORL C,/Bit**

Command operation: (PC)       $\ddot{y}(PC) + 2$

(C)       $\ddot{y}(C)$  or  $/bit$

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[Command code] A9E(y)H

1	0	1	0	0	1	1	1	0	0	y	yyy	Direct address
---	---	---	---	---	---	---	---	---	---	---	-----	----------------

Binary mode **Source** mode

Instruction length: 4      3

Number of Clocks: 1      1

**Hex:** [A5] Script Code Script Code

---

**POP**

Function: pop the stack

Description: Read the contents of the on-chip memory location addressed by the stack pointer, then decrement the stack pointer by 1. The value read at the previous memory location is

Transfer to the newly addressed location, the value can be 8-bit or 16-bit. Does not affect the flag bit.

**POP Dir8**

Command operation: (PC)       $\ddot{y}(PC) + 2$

(dir8)       $\ddot{y}((SP))$

(SP)	$\ddot{y}(SP) - 1$			
Affected flags:	CY	AC	OV	N
	-	-	-	-

[Command code] D0H

1	1	0	1	0	0	0	Direct Address
---	---	---	---	---	---	---	----------------

**Binary mode Source mode**

Instruction length: 2

2

Number of Clocks: 1

1

**Hex:** script

script

**POP Rm**Command operation: (PC)  $\ddot{y}(PC) + 2$ (Rm)  $\ddot{y}((SP))$ 

(SP)	$\ddot{y}(SP) - 1$			
Affected flags:	CY	AC	OV	N
	-	-	-	-

[Instruction code] DA(s)8H

1	1	0	1	1	0	1	0	SSSS	1	0	0
---	---	---	---	---	---	---	---	------	---	---	---

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**POP WRj**Command operation: (PC)  $\ddot{y}(PC) + 2$ (WRj)  $\ddot{y}((SP))$ 

(SP)	$\ddot{y}(SP) - 2$			
Affected flags:	CY	AC	OV	N
	-	-	-	-

[Instruction code] DA(t)9H

1	1	0	1	1	0	1	tttt	1	0	0	1
---	---	---	---	---	---	---	------	---	---	---	---

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**PODRk**Command operation: (PC)  $\ddot{y}(PC) + 2$ (DRk)  $\ddot{y}((SP))$ 

(SP)	$\ddot{y}(SP) - 3$			
Affected flags:	CY	AC	OV	N
	-	-	-	-

[Command code] DA(u)BH

1	1	0	1	1	0	uuuu	1	0	1	1
---	---	---	---	---	---	------	---	---	---	---

**Binary mode Source mode**

Instruction length: 3

2

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**PUSH**

Function: push onto the stack

Description: Increment the stack pointer by 1. The contents of the specified variable are then copied to the on-chip memory location addressed by the stack pointer. Does not affect the flag bit.

**PUSH Dir8**Command operation: (PC)  $\ddot{y}(PC) + 2$ (SP)  $\ddot{y}(SP) + 1$ ((SP))  $\ddot{y}(dir8)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[instruction code] C0H

1	1	0	0	0	0	0	Direct Address
---	---	---	---	---	---	---	----------------

**Binary mode Source mode**

Instruction length: 2

Number of Clocks: 1

**Hex:** script

script

**PUSH #DATA**Command operation: (PC)  $\ddot{y}(PC) + 2$ (SP)  $\ddot{y}(SP) + 1$ ((SP))  $\ddot{y}\#data$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] CA02H

1	1	0	0	1	0	1	0	0	0	0	0	1	0	immediate		
---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----------	--	--

**Binary mode Source mode**

Instruction length: 3

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**PUSH #DATA16**Command operation: (PC)  $\ddot{y}(PC) + 2$ (SP)  $\ddot{y}(SP) + 1$ ((SP))  $\ddot{y}$  MSB #data(SP)  $\ddot{y}(SP) + 1$ ((SP))  $\ddot{y}$  LSB #data

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] CA06H

1	1	0	0	1	0	1	0	0	0	0	0	1			1	0	immediate high byte	immediate low byte
---	---	---	---	---	---	---	---	---	---	---	---	---	--	--	---	---	---------------------	--------------------

**Binary mode Source mode**

Instruction length: 4

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**PUSH Rm**Command operation: (PC)  $\ddot{y}(PC) + 2$ (SP)  $\ddot{y}(SP) + 1$ ((SP))  $\ddot{y}(Rm)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] CA(s)8H

1	1	0	0	1	0	ssss		1	0	0	0
---	---	---	---	---	---	------	--	---	---	---	---

Binary mode **Source** mode

Instruction length: 3 2

Number of Clocks: 1 1

**Hex:** [A5] Script Code Script Code**PUSH WRj**Command operation: (PC)  $\ddot{y}(PC) + 2$ (SP)  $\ddot{y}(SP) + 1$ ((SP))  $\ddot{y}(WRj)$ (SP)  $\ddot{y}(SP) + 1$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] CA(t)9H

1	1	0	0	1	0	tttt	1	0	0	1	
---	---	---	---	---	---	------	---	---	---	---	--

Binary mode **Source** mode

Instruction length: 3 2

Number of Clocks: 1 1

**Hex:** [A5] Script Code Script Code**PUSH DRk**Command operation: (PC)  $\ddot{y}(PC) + 2$ (SP)  $\ddot{y}(SP) + 1$ ((SP))  $\ddot{y}(DRk)$ (SP)  $\ddot{y}(SP) + 3$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] CA(u)BH

1	1	0	0	1	0	uuuu	1	0	1		1
---	---	---	---	---	---	------	---	---	---	--	---

Binary mode **Source** mode

Instruction length: 3 2

Number of Clocks: 1 1

**Hex:** [A5] Script Code Script Code**RET**

Function: return from subroutine

Explanation: RET pops the upper and lower bytes of the PC from the stack in sequence, and decrements the stack pointer by 2. The program continues execution at the result address by

Usually the instruction immediately following ACALL or LCALL. Does not affect the flag bit.

Command operation: (PC15-8)	$\ddot{y}((SP))$										
(SP)	$\ddot{y}(SP) - 1$										
(PC7-0)	$\ddot{y}((SP))$										
(SP)	$\ddot{y}(SP) - 1$										
Affected flags:	<table border="1"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table>	CY	AC	OV	N	Z	-	-	-	-	-
CY	AC	OV	N	Z							
-	-	-	-	-							

[command code] 22H

0	0	1	0	0	1	0
---	---	---	---	---	---	---

**Binary mode Source mode**

Command length: 1	1
Number of Clocks: 3	3
Hex: script	script

## RETI

Function: return from interrupt

Description: This instruction pops two or four bytes from the stack, depending on the INTR bit in the CONFIG1 register. if INTR

= 0, RETI pops the upper and lower bytes of the PC from the stack and uses it as the 16-bit return address for the FF: area. stack

The pointer is decremented by 2. Without affecting other registers, neither PSW nor PSW1 will automatically return to their pre-interrupt state. If INTR = 1,

RETI pops four bytes from the stack: PSW1 and three bytes for PC. The three bytes of the PC are the return address, which can be

Anywhere in the 16MB memory space. The stack pointer is decremented by four. PSW1 returns to pre-interrupt state, but PSW does not

state before interruption. Other registers are not affected. For any value of INTR, the hardware will restore the interrupt logic to receive and just process

The interrupt has the same priority as other interrupts. Program execution continues at the return address, which is usually where an interrupt request was detected

subsequent instructions. If a RETI instruction is executed while an interrupt of the same or lower priority is waiting, the waiting interrupt is processed

Execute this instruction first.

Command operation: INTR=1:		INTR=0:											
(PC15-8)	$\ddot{y}((SP))$	(PC15-8)	$\ddot{y}((SP))$										
(SP)	$\ddot{y}(SP) - 1$	(SP)	$\ddot{y}(SP) - 1$										
(PC7-0)	$\ddot{y}((SP))$	(PC7-0)	$\ddot{y}((SP))$										
(SP)	$\ddot{y}(SP) - 1$	(SP)	$\ddot{y}(SP) - 1$										
(PC23-16)	$\ddot{y}((SP))$												
(SP)	$\ddot{y}(SP) - 1$												
PSW1	$\ddot{y}((SP))$												
(SP)	$\ddot{y}(SP) - 1$												
Affected flags:	<table border="1"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr> </tbody> </table>	CY	AC	OV	N	Z	-	-	-	-	-		
CY	AC	OV	N	Z									
-	-	-	-	-									

[command code] 32H

0	0	1	0	0	1	0
---	---	---	---	---	---	---

**Binary mode Source mode**

Command length: 1	1
Number of Clocks: 3	3
Hex: script	script

## RL

Function: Rotate the accumulator to the left

Description: Eight bits in the accumulator are rotated one bit to the left. The 7th bit loops to the 0th bit position. Only the N and Z flags are affected.

Command operation: (PC)	$\ddot{y}(PC) + 1$				
(An + 1)	$\ddot{y}(An) n = 0-6$				
(A0)	$\ddot{y}(A7)$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 23H

0	0	1	0	0	1	1
---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1  
Number of Clocks: 1  
Hex: script

## RLC

Function: Rotate left accumulator with carry flag

Description: The eight bits in the accumulator and the carry flag are rotated one bit to the left together. The 7th bit is entered into the carry flag;

The previous state is shifted into the 0th bit position. The N and Z flags are also affected.

Command operation: (PC)	$\ddot{y}(PC) + 1$				
(An + 1)	$\ddot{y}(An) n = 0-6$				
(A0)	$\ddot{y}(C)$				
(C)	$\ddot{y}(A7)$				
Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 33H

0	0	1	0	0	1	1
---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1  
Number of Clocks: 1  
Hex: script

## RR

Function: accumulator circularly shifted right

Description: The eight bits in the accumulator are rotated one bit to the right. The 0th bit is cycled to the 7th bit position. Only the N and Z flags are affected.

Command operation: (PC)	$\ddot{y}(PC) + 1$				
(An)	$\ddot{y}(An + 1) n = 0-6$				
(A7)	$\ddot{y}(A0)$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 03H

0	0	0	0	0	1	1
---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1  
Number of Clocks: 1  
Hex: script

## RRC

Function: rotate right accumulator with carry flag

Description: The eight bits in the accumulator and the carry flag are rotated one bit to the right. The 0th bit is entered into the carry flag;

The previous state is shifted into the 7th bit position. The N and Z flags are also affected.

Command operation: (PC)	$\ddot{y}(PC) + 1$
(An)	$\ddot{y}(An + 1) n = 0-6$
(A7)	$\ddot{y}(C)$
(C)	$\ddot{y}(A0)$

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 13H

0 0 0 1 0 0 1	1
---------------	---

**Binary mode Source mode**

Command length: 1	1
Number of Clocks: 1	1
<b>Hex:</b> script	script

## SETB

Function: set

Remarks: SETB sets the specified bit to 1. SETB can operate on the carry flag or any directly addressable bit. does not affect other flags.

## SETB C

Command operation: (PC)	$\ddot{y}(PC) + 1$
(C)	$\ddot{y}1$

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	-	-

[Command code] D3H

1 1 0 1 0 0 1	1
---------------	---

**Binary mode Source mode**

Command length: 1	1
Number of Clocks: 1	1
<b>Hex:</b> script	script

## SETB Bit51

Command operation: (PC)	$\ddot{y}(PC) + 2$
(bit51)	$\ddot{y}1$

Affected 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	3
<b>Hex:</b> script	script

## SETB Bit

Command operation: (PC)	$\ddot{y}(PC) + 3$
-------------------------	--------------------

(bit)	$\ddot{y}1$				
Affected 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	Direct address
---------------	---	---------------	----------------

**Binary mode Source mode**

Instruction length: 4 3

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**SJMP**

Function: short jump

Description: Program control jumps unconditionally to the specified address. Calculate the jump by adding the PC to the signed relative offset in the second byte of the instruction

Turn the target, then the PC is incremented twice. Therefore, the range of allowed targets is from 128 bytes before the instruction to after it between 127 bytes. Does not affect the flag bit.

Command operation: (PC)  $\ddot{y}(PC) + 2$ (PC)  $\ddot{y}(PC) + rel$ 

(PC)	CY	AC	OV	N	Z
	-	-	-	-	-

[command code] 80H

1 0 0 0 0 0 0	0	Relative address
---------------	---	------------------

**Binary mode Source mode**

Instruction length: 2 2

Number of Clocks: 3

Hex: script script

**SLL**

Function: logical left shift 1 bit

Description: Shifts the specified variable to the left by 1 bit, and replaces the least significant bit with zero. The most significant bit (MSB) of the shifted out bit is stored in the CY bit.

The N and Z flags are also affected.

**SLL Rm**Command operation: (PC)  $\ddot{y}(PC) + 2$ (Rm).a + 1  $\ddot{y}(Rm).a$ (Rm).0  $\ddot{y}0$ CY  $\ddot{y}(Rm).7$ 

(Rm).0	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction code] 3E(s)0H

0 0 1	1	1	1	1 0 sss	sss 0 0 0 0
-------	---	---	---	---------	-------------

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 1

Hex: [A5] Script Code Script Code

**SLL WRj**

Command operation: (PC)	$\ddot{y}(PC) + 2$										
(WRj).b + 1	$\ddot{y}(WRj).b$										
(WRj).0	$\ddot{y}0$										
CY	$\ddot{y}(WRj).15$										
Affected flags:	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td><math>\ddot{y}</math></td><td>-</td><td>-</td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td></tr> </tbody> </table>	CY	AC	OV	N	Z	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$
CY	AC	OV	N	Z							
$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$							
[Command code] 3E(t)4H											
	0 0 1      1   1  1  1 0 ttt   0 1 0 0										

**Binary mode Source mode**

Instruction length: 3                          2  
 Number of Clocks: 1                          1

**Hex:** [A5] Script Code Script Code

**SRA**

Function: Arithmetic right shift 1 bit (signed)

Description: Shift the specified variable to the right by 1 bit arithmetically. The most significant bit is unchanged. The shifted out bit (LSB) is stored in the CY bit. N and Z

Flag bits are also affected.

**SRA Rm**

Command operation: (PC)	$\ddot{y}(PC) + 2$										
(Rm).7	$\ddot{y}(Rm).7$										
(Rm).a	$\ddot{y}(Rm).a+1$										
CY	$\ddot{y}(Rm).0$										
Affected flags:	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td><math>\ddot{y}</math></td><td>-</td><td>-</td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td></tr> </tbody> </table>	CY	AC	OV	N	Z	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$
CY	AC	OV	N	Z							
$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$							
[Instruction code] 0E(s)0H											
	0 0 0 1   1  1 0 sssss   0 0 0 0										

**Binary mode Source mode**

Instruction length: 3                          2  
 Number of Clocks: 1                          1

**Hex:** [A5] Script Code Script Code

**SRA WRj**

Command operation: (PC)	$\ddot{y}(PC) + 2$										
(WRj).15	$\ddot{y}(WRj).15$										
(WRj).b	$\ddot{y}(WRj).b + 1$										
CY	$\ddot{y}(WRj).0$										
Affected flags:	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>CY</th><th>AC</th><th>OV</th><th>N</th><th>Z</th></tr> </thead> <tbody> <tr> <td><math>\ddot{y}</math></td><td>-</td><td>-</td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td></tr> </tbody> </table>	CY	AC	OV	N	Z	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$
CY	AC	OV	N	Z							
$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$							
[Instruction code] 0E(t)4H											
	0 0 0 1   1  1 0 ttt   0 1 0 0										

**Binary mode Source mode**

Instruction length: 3                          2  
 Number of Clocks: 1                          1

**Hex:** [A5] Script Code Script Code

**SRL**

Function: Logical right shift 1 bit

Explanation: SRL shifts the specified variable to the right by 1 bit, replacing the most significant bit with zero. The shifted out bit (LSB) is stored in the CY bit. N and Z

Flag bits are also affected.

**SRL Rm**Command operation: (PC)  $\ddot{y}(PC) + 2$ (Rm).7  $\ddot{y}(Rm).0$ (Rm).a  $\ddot{y}(Rm).a+1$ CY  $\ddot{y}(Rm).0$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction 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 2

Number of Clocks: 1 1

**Hex:** [A5] Script Code Script Code**SRL WRj**Command operation: (PC)  $\ddot{y}(PC) + 2$ (WRj).15  $\ddot{y}0$ (WRj).b  $\ddot{y}(WRj).b + 1$ CY  $\ddot{y}(WRj).0$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction 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 2

Number of Clocks: 1 1

**Hex:** [A5] Script Code Script Code**SUB**

Function: Subtract

Description: Subtracts the specified variable from the destination operand, leaving the result in the destination operand. SUB is set if the 7th bit requires a borrow

bit CY flag bit (borrow), otherwise clear CY bit. When subtracting signed integers, the OV flag indicates when a positive number is subtracted from a negative number

A negative number occurs, or a positive result occurs when a negative number is subtracted from a positive number. Bit 7 in this description refers to the most significant byte of the operand

(8, 16 or 32 bits). The source operand allows four addressing modes: immediate, indirect, register, and direct addressing. Except for AC,

All flags are affected, it does not affect word and double word subtraction.

**SUB Rmd, Rms**Command operation: (PC)  $\ddot{y}(PC) + 2$ (Rmd)  $\ddot{y}(Rms) - (Rmd)$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Command code] 9CH

1 0 0 1	1 1 0 0 ssss	SSSS	
---------	--------------	------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**SUB WRjd, WRjs**Command operation: (PC)  $\ddot{y}(PC) + 2$ (WRjd)  $\ddot{y}(WRjs) - (WRjd)$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[command code] 9DH

1 0 0 1	1 1 0 1	tttt TTTT	
---------	---------	-----------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**SUB DRkd, DRks**Command operation: (PC)  $\ddot{y}(PC) + 2$ (DRkd)  $\ddot{y}(DRks) - (DRkd)$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Command code] 9FH

1 0 0 1	1 1 1 1	uuuu UUUU	
---------	---------	-----------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 4

**Hex:** [A5] Script Code Script Code**SUB Rm, #DATA**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm) - \#data$ 

Affected flags:	CY	AC	OV	N	Z
	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Command code] 9E(s)0H

1 0 0 1	1 1 1 0 ssss	0 0 0 0 immediate	
---------	--------------	-------------------	--

**Binary mode Source mode**

Instruction length: 4 3

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**SUB WRj, #DATA16**Command operation: (PC)  $\ddot{y}(PC) + 4$ (WRj)  $\ddot{y}(WRj) - \#data16$

Affected flags:

CY	AC	OV	N	Z
ÿ	-	ÿ	ÿ	ÿ

[Command code] 9E(t)4H

1 0 0 1	1 1 1 0 ttt 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] Script Code Script Code

**SUB DRk,#0DATA16**

Command operation: (PC)

ÿ(PC) + 4

(DRk)

ÿ(DRk) - #data16

Affected flags:

CY	AC	OV	N	Z
ÿ	-	ÿ	ÿ	ÿ

[Command code] 9E(u)8H

1 0 0 1	1 1 1 0 uuuu	1 0 0 immediate high byte	immediate low byte	
---------	--------------	---------------------------	--------------------	--

Binary mode Source mode

Instruction length: 5

4

Number of Clocks: 1

1

Hex: [A5] Script Code Script Code

**SUB Rm, Dir8**

Command operation: (PC)

ÿ(PC) + 3

(Rm)

ÿ(Rm) - (dir8)

Affected flags:

CY	AC	OV	N	Z
ÿ	ÿ	ÿ	ÿ	ÿ

[Command code] 9E(s)1H

1 0 0 1	1 1 1 0 ssss	0 0 0 1 Direct address	
---------	--------------	------------------------	--

Binary mode Source mode

Instruction length: 4

3

Number of Clocks: 1

1

Hex: [A5] Script Code Script Code

**SUB WRj, Dir8**

Command operation: (PC)

ÿ(PC) + 3

(WRj)

ÿ(WRj) - (dir8)

Affected flags:

CY	AC	OV	N	Z
ÿ	-	ÿ	ÿ	ÿ

[Command code] 9E(t)5H

1 0 0 1	1 1 1 0 ttt	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] Script Code Script Code

**SUB Rm, Dir16**

Command operation: (PC)	$\ddot{y}(PC) + 4$														
(Rm)	$\ddot{y}(Rm) - (\text{dir16})$														
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td></tr> </table>					CY	AC	OV	N	Z	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$
CY	AC	OV	N	Z											
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$											
[Command code] 9E(s)3H															
	1 0 0 1	1 1 1 0	ssss	0 0 1	1 address high byte address low byte										
<b>Binary mode Source mode</b>															
Instruction length: 5	4														
Number of Clocks: 1	1														
<b>Hex: [A5] Script Code Script Code</b>															

**SUB WRj, Dir16**

Command operation: (PC)	$\ddot{y}(PC) + 4$														
(WRj)	$\ddot{y}(WRj) - (\text{dir16})$														
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td><math>\ddot{y}</math></td><td>-</td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td></tr> </table>					CY	AC	OV	N	Z	$\ddot{y}$	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$
CY	AC	OV	N	Z											
$\ddot{y}$	-	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$											
[Command code] 9E(t)7H															
	1 0 0 1	1 1 1 0	ttt	0 1	1 1 address high byte address low byte										
<b>Binary mode Source mode</b>															
Instruction length: 5	4														
Number of clocks: 2	2														
<b>Hex: [A5] Script Code Script Code</b>															

**SUB Rm, @WRj**

Command operation: (PC)	$\ddot{y}(PC) + 3$														
(Rm)	$\ddot{y}(Rm) - ((WRj))$														
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td></tr> </table>					CY	AC	OV	N	Z	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$
CY	AC	OV	N	Z											
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$											
[Instruction code] 9E(t)9(s)0H															
	1 0 0 1	1 1 1 0	ttt	1 0 0 1	ssss 0 0 0 0										
<b>Binary mode Source mode</b>															
Instruction length: 4	3														
Number of Clocks: 1	1														
<b>Hex: [A5] Script Code Script Code</b>															

**SUB Rm,@DRk**

Command operation: (PC)	$\ddot{y}(PC) + 3$														
(Rm)	$\ddot{y}(Rm) - ((DRk))$														
Affected flags:	<table border="1"> <tr> <td>CY</td><td>AC</td><td>OV</td><td>N</td><td>Z</td></tr> <tr> <td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td><td><math>\ddot{y}</math></td></tr> </table>					CY	AC	OV	N	Z	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$
CY	AC	OV	N	Z											
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$											
[Instruction code] 9E(u)B(s)0H															
	1 0 0 1	1 1 1 0	uuu	U 1 0 1	1 ssss 0 0 0 0										
<b>Binary mode Source mode</b>															
Instruction length: 4	3														
Number of Clocks: 3	3														
<b>Hex: [A5] Script Code</b>															

**SUBB A,<src-byte>**

Function: Borrow and Subtract

Remarks: SUBB subtracts the specified variable and the carry flag from the accumulator together, leaving the result in the accumulator. If the 7th bit is required Borrow, SUBB sets the carry (borrow) flag, otherwise clears C. (If C was set before executing the SUBB instruction bits, which means a borrow is required for the previous step in many exact subtractions, so the carry is subtracted from the accumulator along with the source operand). AC is set if a borrow is required for bit 3, otherwise cleared. If you need to borrow to bit 6 instead of bit 7, or you need to OV is set when borrowing to bit 7 instead of bit 6. The OV flag bit indicates that a negative number occurs when a positive number is subtracted from a negative number, or a negative number A positive number appeared when subtracting a positive number. The source operand allows four addressing modes: register, direct, register indirect, or immediate site. All flag bits are affected.

**SUBB A,Rn**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ (A)  $\ddot{y}(A) - (C) - (Rn)$ 

Affected flags:

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Command code] 98H - 9FH

1 0 0 1	1 rrr
---------	-------

Binary mode Source mode

Command length: 1

2

Number of Clocks: 1

1

Hex: script

[A5] Script

**SUBB A,Direct**

Command operation: (PC)

 $\ddot{y}(PC) + 2$ (A)  $\ddot{y}(A) - (C) - (\text{direct})$ 

Affected flags:

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[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: script

script

**SUBB A, @Ri**

Command operation: (PC)

 $\ddot{y}(PC) + 1$ (A)  $\ddot{y}(A) - (C) - ((Ri))$ 

Affected flags:

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Command code] 96H, 97H

1 0 0 1 0 1	1 i
-------------	-----

Binary mode Source mode

Command length: 1

2

Number of Clocks: 1

1

Hex: script

[A5] Script

**SUBB A,#DATA**

Command operation: (PC)

 $\ddot{y}(PC) + 2$ 

(A)

 $\ddot{y}(A) - (C) - \#data$ 

Affected flags:

CY	AC	OV	N	Z
$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$	$\ddot{y}$

[Command code] 94H

1	0	0	1	0	1	0	immediate	
---	---	---	---	---	---	---	-----------	--

**Binary mode Source mode**

Instruction length: 2

2

Number of Clocks: 1

1

**Hex:** script

script

**SWAP A**

Function: accumulator swap nibbles

Description: SWAP A swaps the low and high nibble (four-bit field) of the accumulator (bits 3 to 0 and 7 to 4). the fuck

The operation can also be thought of as a four-bit loop instruction. Only the N and Z flags are affected.

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

(A3-0)

 $\ddot{y}(A7-4)$ 

(A7-4)

 $\ddot{y}(A3-0)$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] C4H

1	1	0	0	0	1
---	---	---	---	---	---

**Binary mode Source mode**

Command length: 1

1

Number of Clocks: 1

1

**Hex:** script

script

**TRAP**

Function: Execute as NOP

Command operation: (PC)

 $\ddot{y}(PC) + 1$ 

Affected flags:

CY	AC	OV	N	Z
-	-	-	-	-

[Command code] B9H

1	0	1	1	1	0	0
---	---	---	---	---	---	---

**Binary mode Source mode**

Instruction length: 2

1

Number of Clocks: 1

1

**Hex:** [A5] Script Code Script Code**XCH A,<byte>**

Function: accumulator swap byte variable

Explanation: XCH loads the contents of the specified variable into the accumulator and writes the previous contents of the accumulator to the specified variable. Source number, target operation

Numbers can be addressed using registers, direct, or register indirect. Does not affect the flag bit.

**XCH A,Rn**Command operation: (PC)  $\ddot{y}(PC) + 1$ (A)  $\ddot{y}(Rn)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] C8H - CFH

1	1	0	0	1	rri
---	---	---	---	---	-----

Binary mode Source mode

Command length: 1 2

Number of Clocks: 1

Hex: script [A5] Script

**XCH A,Direct**Command operation: (PC)  $\ddot{y}(PC) + 2$ (A)  $\ddot{y}(\text{direct})$ 

Affected 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

Hex: script script

**XCH A, @Ri**Command operation: (PC)  $\ddot{y}(PC) + 1$ (A)  $\ddot{y}((Ri))$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Instruction code] C6H, C7H

1	1	0	0	0	1	i
---	---	---	---	---	---	---

Binary mode Source mode

Command length: 1 2

Number of Clocks: 1

Hex: script [A5] Script

**XCHD A, @Ri**

Function: nibble swap numbers

Description: XCHD compares the low nibble of the accumulator (bits 3 to 0, usually representing hexadecimal or BCD numbers) with the specified register

Nibble swap of indirectly addressed internal memory locations. The upper nibble (bits 7 to 4) of each register is not affected.

Does not affect the flag bit.

Command operation: (PC)  $\ddot{y}(PC) + 1$ (A3-0)  $\ddot{y}((Ri)3-0)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	-	-

[Command code] D6H, D7H

1	1	0	1	0	1
---	---	---	---	---	---

**Binary mode Source mode**

Command length: 1

2

Number of Clocks: 3

3

**Hex:** script

[A5] Script

**XRL**

Function: Logical XOR of variables

Description: Performs a bitwise logical XOR operation between the specified variables, storing the result in the destination. The destination operand can be an accumulator, register or direct address. These two operands allow 12 combinations of addressing modes. When the destination is an accumulator or a register, the source number can be register, direct, register indirect, or immediate addressing; when the destination is a direct address, the source number can be accumulated. Adder or immediate value. Only the N and Z flags are affected.

Note: When this instruction is used to modify an output port, the value used as raw port data is read from the output data latch, while not an input pin.

**XRL A,Rn**Command operation: (PC)  $\ddot{y}(PC) + 1$ (A)  $\ddot{y}(A)$  xor  $(R_n)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 68H - 6FH

0	1	1	0	1	rr
---	---	---	---	---	----

**Binary mode Source mode**

Command length: 1

2

Number of Clocks: 1

1

**Hex:** script

[A5] Script

**XRL A,Direct**Command operation: (PC)  $\ddot{y}(PC) + 2$ (A)  $\ddot{y}(A)$  xor (direct)

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[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:** script

[A5] Script

**XRL A, @Ri**Command operation: (PC)  $\ddot{y}(PC) + 1$ (A)  $\ddot{y}(A)$  xor  $((R_i))$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 66H, 67H

0	1	1	0	0	1	1	i
---	---	---	---	---	---	---	---

**Binary mode Source mode**

Command length: 2

2

Number of Clocks: 1

1

**Hex:** script

[A5] Script

**XRL A,#DATA**Command operation: (PC)  $\ddot{y}(PC) + 2$ (A)  $\ddot{y}(A) \text{ xor } \#data$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 64H

0 1	1 0 0	1 0 1	immediate	
-----	-------	-------	-----------	--

**Binary mode Source mode**

Instruction length: 2

2

Number of Clocks: 1

1

**Hex:** script

script

**XRL Direct,A**Command operation: (PC)  $\ddot{y}(PC) + 2$ (direct)  $\ddot{y}(\text{direct}) \text{ xor } (A)$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 62H

0 1	1 0 0	0 1 0	Direct Address	
-----	-------	-------	----------------	--

**Binary mode Source mode**

Instruction length: 2

2

Number of Clocks: 1

1

**Hex:** script

script

**XRL Direct, #DATA**Command operation: (PC)  $\ddot{y}(PC) + 3$ (direct)  $\ddot{y}(\text{direct}) \text{ xor } \#data$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 63H

0 1	1 0 0	0 1	1	direct address	immediate
-----	-------	-----	---	----------------	-----------

**Binary mode Source mode**

Instruction length: 3

3

Number of Clocks: 1

1

**Hex:** script

script

**XRL Rmd,Rms**Command operation: (PC)  $\ddot{y}(PC) + 2$ (Rmd)  $\ddot{y}(\text{Rms}) \text{ xor } (\text{Rmd})$ 

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 6CH

0 1	1 0 1	1 0 0 ssss	SSSS	
-----	-------	------------	------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**XRL WRjd, WRjs**Command operation: (PC)  $\ddot{y}(PC) + 2$ (WRjd)  $\ddot{y}(WRjs)$  xor (WRjd)

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[command code] 6DH

0 1	1 0 1	1 0 1	tttt TTTT	
-----	-------	-------	-----------	--

**Binary mode Source mode**

Instruction length: 3 2

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**XRL Rm,#DATA**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm)$  xor #data

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction code] 6E(s)0H

0 1	1 0 1	1	1 0 ssss	0 0 0 immediate	
-----	-------	---	----------	-----------------	--

**Binary mode Source mode**

Instruction length: 4 3

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**XRL WRj, #DATA16**Command operation: (PC)  $\ddot{y}(PC) + 4$ (WRj)  $\ddot{y}(WRj)$  xor #data16

Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Command code] 6E(t)4H

0 1	1 0 1	1 0 1 tttt 0 1 0 0 immediate high byte	immediate low byte	
-----	-------	--	--------------------	--

**Binary mode Source mode**

Instruction length: 5 4

Number of Clocks: 1

**Hex:** [A5] Script Code Script Code**XRL Rm, Dir8**Command operation: (PC)  $\ddot{y}(PC) + 3$ (Rm)  $\ddot{y}(Rm)$  xor (dir8)

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[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] Script Code Script Code

**XRL WRj, Dir8**

Command operation: (PC)

ÿ(PC) + 3

(WRj)

ÿ(WRj) xor (dir8)

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[Instruction 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] Script Code Script Code

**XRL Rm, Dir16**

Command operation: (PC)

ÿ(PC) + 4

(Rm)

ÿ(Rm) xor (dir16)

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[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] Script Code Script Code

**XRL WRj, Dir16**

Command operation: (PC)

ÿ(PC) + 4

(WRj)

ÿ(WRj) xor (dir16)

Affected flags:

CY	AC	OV	N	Z
-	-	-	ÿ	ÿ

[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] Script Code Script Code

**XRL Rm, @WRj**

Command operation: (PC)	$\ddot{y}(PC) + 3$				
(Rm)	$\ddot{y}(Rm) \text{ xor } ((WRj))$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction code] 6E(t)9(s)0H

0 1	1 0 1	1 1 0	ttt	1 0 0 1		ssss 0 0 0 0
-----	-------	-------	-----	---------	--	--------------

Binary mode Source mode

Instruction length: 4 3

Number of Clocks: 1

Hex: [A5] Script Code Script Code

## XRL Rm, @DRk

Command operation: (PC)	$\ddot{y}(PC) + 3$				
(Rm)	$\ddot{y}(Rm) \text{ xor } ((DRk))$				
Affected flags:	CY	AC	OV	N	Z
	-	-	-	$\ddot{y}$	$\ddot{y}$

[Instruction code] 6E(u)B(s)0H

0 1	1 0 1	1 1 0	uuuu	1 0 1	1	ssss 0 0 0 0
-----	-------	-------	------	-------	---	--------------

Binary mode Source mode

Instruction length: 4 3

Number of Clocks: 3

Hex: [A5] Script script

## Appendix B Basics of Logical Algebra

—Users who do not have the principle of microcomputer, please start learning from this chapter

The main contents of this chapter are: yBasic knowledge of arithmetic operations in digital equipment - number system and coding; yDigital circuits

Some common logical operations and their graphic symbols in They are the foundation of this course on Microcontrollers. For users who do not have the basis of microcomputer principle

And classmates, please start with this chapter.

### B.1 Number System and Coding

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

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

The carry counting system is to divide the number into different digits, accumulate them digit by digit, add to a certain number, then start from zero, and at the same time carry up to the high digit.

bit. The carry counting system has three elements: digital sign, carry rule and counting base. The following table is a general introduction to each common number system.

Commonly used number system representation	digital sign	base rule	counting base
symbol binary decimal B	0, 1	Every two	2
D	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	into one every ten into a 10	
hex	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	Every sixteen to one	16

In our daily life, we generally use decimal system for counting. The binary is used in the computer, because the binary is simple in operation and easy to implement.

It is practical and reliable, and provides a favorable approach for logic design, saving equipment and other advantages. In order to distinguish it from other base numbers, the writing of binary numbers is usually

Always note the base 2 at the bottom right of the number, or add B after it. Each digit in a binary number has only two possible digits, 0 and 1, so

The count base is 2. Addition and multiplication of binary numbers are as follows:

$$\begin{array}{lll} 0 + 0 = 0 & 0 + 1 = 1 + 0 = 1 & 1 + 1 = 10 \\ 0 \times 0 = 0 & 0 \times 1 = 1 \times 0 = 0 & 1 \times 1 = 1 \end{array}$$

Because the median of binary numbers is too long and not easy to remember, for the convenience of description, hexadecimal is often used as the abbreviation of binary.

Hexadecimal is usually represented by a trailing mark H or subscript 16 to distinguish it.

#### B.1.1 Number system conversion

Now let's introduce the conversion between these common number systems.

One: binary-decimal conversion

Method: Expand the binary number according to the weight (as shown in the following formula), and then add the values of each item according to the decimal number to get the corresponding equivalent decimal system number.

For example: N=(1101.101)B, what is the decimal number corresponding to

N? Expand by weight  $N=1\times2^3 +1\times2^2 +0\times2^1 +1\times2^{-1} +0\times2^{-2} +1\times2^{-3} =8+4+0+1+0.5+0 +0.125 =(13.625)D$

Two: Decimal to Binary Conversion

Method: Divide into two parts, the integer part and the decimal part.

ÿInteger part conversion (base division): ÿ Divide the number we want

to convert by the binary base (the binary base is 2), and use the remainder as the lowest digit of the binary; ÿ Divide the last quotient by the binary base ( That is, 2), take the remainder as the second-lowest bit of the binary; ÿ Continue the previous step until the final quotient is zero,

and the remainder is the highest bit of the binary. ÿConversion of the fractional part (radix multiplication): ÿ Convert the decimal of the number

to be converted Multiply the part by the binary base (the binary base is 2), and use the obtained integer part as the highest digit of the binary

fraction; ÿ Multiply the decimal part obtained in the previous step by the binary base (ie 2), and use the integer part as The second highest bit of the binary

fractional part; ÿ Continue the previous step until the fractional part becomes zero. Or meet the predetermined requirements.

For example, to convert (213.8125)<sub>10</sub> to binary, proceed as follows:

Prepend the integer part:

$$\begin{array}{r}
 2 \ 2 \ | 3 \\
 2 \ 1 0 6 \ | 2 \\
 5 3 \ 2 \ | 2 6 \\
 1 3 \ | \\
 2 \ | \\
 2 \ | 6 \\
 2 \ | 3 \\
 2 \ | 1 \\
 0
 \end{array}
 \begin{array}{l}
 \text{remainder}=1=k_0 \\
 \text{remainder}=0=k_1 \\
 \text{remainder}=1=k_2 \\
 \text{remainder}=0=k_3 \\
 \text{remainder}=1=k_4 \\
 \text{remainder}=0=k_5 \\
 \text{remainder}=1=k_6 \\
 \text{remainder}=1=k_7
 \end{array}$$

So the integer part (213)<sub>10</sub>=(11010101)<sub>2</sub>

Rescale the fractional part:

$$\begin{array}{r}
 0.8125 \\
 \times \quad 2 \\
 \hline
 1.6250 \quad \text{----- Integer part}=1=k_{-1} \\
 0.6250 \\
 \times 2 \\
 \hline
 1.2500 \quad \text{----- Integer part}=1=k_{-2} \\
 0.2500 \\
 \times 2 \\
 \hline
 0.5000 \quad \text{----- Integer part}=0=k_{-3} \\
 0.5000 \\
 \times 2 \\
 \hline
 1.0000 \quad \text{----- Integer part}=1=k_{-4}
 \end{array}$$

So the fractional part (0.8125)<sub>10</sub>=(0.1101)<sub>2</sub>

To sum up, the decimal number 213.8125=(11010101.1101)<sub>2</sub>=(11010101.1101)<sub>B</sub>

### Three: binary-hexadecimal conversion

**Method:** The relationship between binary and hexadecimal is 24, so the binary to be converted is grouped by 4 bits from low to high, high to low.

When there are insufficient bits, add "0" in front of the significant bits, and then convert each group of binary numbers into hexadecimal.

For example, to convert (010111011110.11010010)B to hexadecimal:

$$= ( \begin{smallmatrix} \ddot{y} & \ddot{y} & \ddot{y} & \ddot{y} \\ 5 & DEB & 2 & H \end{smallmatrix} \text{ So, } )$$

(010111011110.11010010)B=(5DE.B2)H

#### Four: Hexadecimal - binary conversion

**Method:** When converting from hexadecimal to binary, reverse the above process of converting from binary to hexadecimal, that is, you only need to convert the hexadecimal to hexadecimal.

Each digit in the base is replaced by the equivalent 4-bit binary.

Example: Convert (C1B.C6)H to binary:

$$(C1B.C6)H$$

$\ddot{y} \ddot{y} \ddot{y} \ddot{y}$

$$= (1100\ 0001\ 1011\ 1100\ 0110)B$$

$$\text{So, } (C1B.C6)H = (110000011011.11000110)_B$$

## Five: Hexadecimal-Decimal Conversion

Method: Expand the hexadecimal number according to the weight (as shown in the following formula), and then add the values of each item according to the decimal number to get the corresponding equivalent ten base number.

For example:  $N=(2A.7F)H$ , what is the decimal number corresponding to N?

Expand by weight  $N=2 \times 161 + 10 \times 160 + 7 \times 16 - 1 + 15 \times 16 - 2 = 32 + 10 + 0.4375 + 0.05859375 = (42.49609375)D$

$$\text{So, } (2A.7F)H = (42.49609375)D$$

## Six: Decimal-Hexadecimal Conversion

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

Convert system numbers to equivalent hexadecimal numbers.

### B.1.2 Original code, inverse code and complement code

In life, numbers are divided into positive and negative. How to represent the positive and negative signs of numbers in the computer?

When expressing numbers in life, we usually add a "+" in front of positive numbers and a "-" in front of negative numbers, but computers do not

To recognize these, usually add a sign bit in front of the binary number. A sign bit of "0" means "+", and a sign bit of "1" means "-".

Binary numbers in this form are called primitives. If the original code is a positive number, the inverse and complement of the original code are the same as the original code. If the original code is negative,

Then invert the original code (except the sign bit) bit by bit, and the new binary number obtained is called the inverse code of the original code, and the inverse code plus 1 is its complement.

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

	True value	original code	one's complement	complement
Positive	+N	0N	0N	0N
-N		1N	$(2^n - 1) + N$	$2^n + N$

Example 1: Find +18 and -18 8-bit original code, inverse code, complement code form.

True value	original code+18	Inverse	complement
00010010	-18	10010010	00010010
		11101101	11101110

### B.1.3 Commonly used codes

Specifying a set of binary numbers to represent a specified information is called encoding.

One: Decimal encoding

Decimal numbers represented by binary codes are called decimal codes. It has the form of binary and also has the characteristics of decimal. It can be used as

An inter-representation of people's connection to digital systems. There are many kinds of decimal codes, the most commonly used one is BCD code, also known as 8421 code.

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

Type of encoding Decimal	8421 yards (BCD code)	3 yards left	2421 yards	5211 yards	7321 yards
0	0000	0011	0000	0000	0000
1	0001	0100	0001	0001	0001
2	0010	0101	0010	0100	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0111	0101
5	0101	1000	1011	1000	0110
6	0110	1001	1100	1001	0111
7	0111	1010	1101	1100	1000
8	1000	1011	1110	1101	1001
9	1001	1100	1111	1111	1010
rights	8421		2421	5211	7321

Decimal encoding is divided into authorized and unauthorized encoding. The authorized code means that each decimal digit is represented by a set of four-digit binary codes.

And each bit of the binary code has a fixed weight. Weightless coding means that each bit in the binary code does not have a fixed weight. in the table above

Code 8421 (ie BCD code), code 2421, code 5211 and code 7321 are all authorized codes, while the remaining 3 codes are unauthorized codes.

Two: parity check code

In the process of data access, operation and transmission, it is inevitable that errors will occur, and "1" is mistaken for "0" or "0" is mistaken for "1". strange

Even parity code is a code that can check for such errors. It is divided into two parts; information bits and parity bits. An odd number of "1"s is called an odd school

If there is an even number of "1"s, it is called even parity.

## B.2 Several commonly used logical operations and their graphic symbols

Commonly used operations in logical algebra are: AND (AND), OR (OR), NOT (NOT), AND NOT (NAND), OR NOT (NOR), AND OR NOT

(AND-NOR), XOR (EXCLUSIVE OR), XOR (EXCLUSIVE NOR), etc. Among them and (AND), or (OR), not (NOT)

There are three basic types of operations.

One: AND operation and AND gate

AND Operation: An event occurs when all conditions that determine the outcome of an event are met at the same time.

When logical variables A and B are ANDed, they can be written as:  $Y=A \cdot B$

truth table		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

AND gate: A unit circuit that performs AND logic operations.

AND gate graphic symbol:



Two: OR operation and OR gate

OR Operation: As long as any one of the conditions that determine the outcome of an event is satisfied, the event will occur.

When logical variables A and B are ORed, they can be written as:  $Y=A+B$

truth table		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

OR gate: A unit circuit that performs an OR logical operation.

OR gate graphic symbol:



Three: NOT operation and NOT gate

NOT operation: When the conditions are met, the event will not occur; when the conditions are not met, the event will occur.

When the logical variable A is not operated, it can be written as:  $Y=\bar{A}$

truth table	
A	Y
0	1
1	0

NOT gate: A unit circuit that performs non-logical operations.

NOT gate graphic symbol:



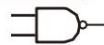
Four: NAND operation and NAND graphic symbols

AND NOT operation: first perform AND operation, then invert the result, and finally get the result of AND NOT operation.

Logical variables A and B can be written as:  $Y = (A \cdot B) \bar{}$

truth table		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

with non-graphical symbols:



Five: NOR operation and NOR graphic symbols

NOR operation: first perform the OR operation, then invert the result, and finally get the result of the NOR operation.

Logical variables A and B can be written as:  $Y = (A + B) \bar{}$

truth table		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

or non-graphical symbols:



Six: AND or NOT operation and AND or non-graphical symbols

AND or NOT operation: There are 4 logical variables A, B, C, D in the AND or NOT logical operation. Suppose A and B are a group, C and D are a group,

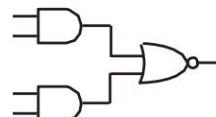
The relationship between A and B and between C and D is AND relationship. As long as any group of A, B or C, D is 1 at the same time, the output Y is 0. Only

There is when each set of inputs is not all 1, the output Y is 1.

Logical variables A and B can be written as:  $Y = (A \cdot B + C \cdot D) \bar{}$

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

AND or non-graphical symbols:



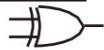
## Seven: XOR operation and XOR graphic symbols

XOR operation: when A and B are different, the output Y is 1; and when A and B are the same, the output Y is 0. The logical variables A and B are XORed

Or can be written as:  $Y = A \oplus B = (A \cdot \bar{B}) + (\bar{A} \cdot B)$

truth table		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XOR graphic notation:



## Eight: XOR operation and XOR graphic symbols

XOR operation: when A and B are different, the output Y is 1; and when A and B are the same, the output Y is 0. Logical variables A and B are identical

Or can be written as:  $Y = A \oplus B = (A \cdot B) + (\bar{A} \cdot \bar{B})$

truth table		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Same or graphic symbols:

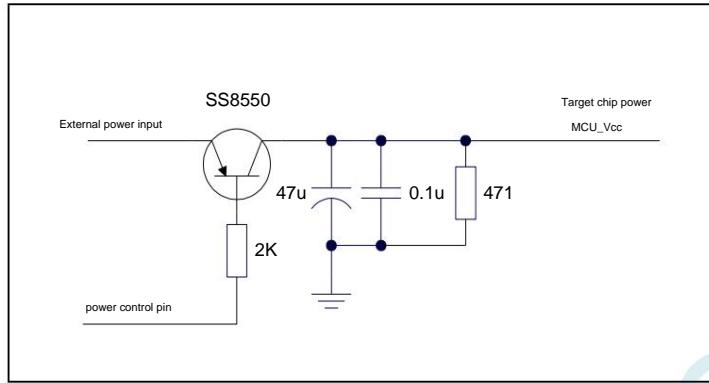


## Appendix C Using third-party MCU to STC32G series MCU

### Perform ISP download sample program

STC chip ISP download requires hardware reset of the target to enter ISP download mode. When using a third-party MCU to process the STC chip

When performing ISP download, it is recommended to use the following power control circuit for implementation.



#### C language code

// Note that this code pair can be **STC8H**. It must be executed when the series of microcontrollers are downloaded. **Download** after the code  
 // used to power on 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 definition
#define FALSE          0
#define TRUE           1
#define LOBYTE(w)       ((BYTE)(WORD)(w))
#define HIBYTE(w)       ((BYTE)((WORD)(w) >> 8))

#define MINBAUD        2400L
#define MAXBAUD        115200L

#define FOSC            11059200L
#define BR(n)           (65536 - FOSC/4/(n))
#define T1MS             (65536 - FOSC/1000)
// Main control chip operating frequency
// Calculation formula of serial port baud rate of main control chip
// Main control chip timing initial value

#define FUSER           24000000L
#define RL(n)           (65536 - FUSER/4/(n))
// STC32G Serial target chip operating frequency
// STC32G Serial target chip serial port baud rate calculation formula

sfr    AUXR = 0x8e;
sfr    P3M1 = 0xB1;
sfr    P3M0 = 0xB2;
```

```

// Variable definitions
BOOL f1ms; // 1ms flag bit
BOOL UartBusy; // Serial port sends busy flag
BOOL UartReceived; // Serial port data reception completion flag bit
BYTE UartRecvStep; // Serial data receiving control
BYTE TimeOut; // Serial communication timeout counter
BYTE xdata TxBuffer[256]; // Serial data transmission buffer
BYTE xdata RxBuffer[256]; // Serial data receive buffer
char code DEMO[256]; // demo code data

// function declaration
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
BOOL Download(BYTE *pdat, long size);

// main function entry
void main(void)
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    Initial();
    if (Download(DEMO, 256))
    {
        // download 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
    {
        // download failed
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
    }
}

```

```
while (1);

}

//1ms Timer Interrupt Service Routine
void tm0(void) interrupt
1 {
    static BYTE Counter100;

    f1ms = TRUE;
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}

// Serial Interrupt Service Routine
void uart(void) interrupt
4 {
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat = SBUF;
        switch (UartRecvStep)
        { case 1:

            if (dat != 0xb9) goto L_CheckFirst;
            UartRecvStep++; break; case 2:

            if (dat != 0x68) goto L_CheckFirst;
            UartRecvStep++; break; case 3:

            if (dat != 0x00) goto L_CheckFirst;
            UartRecvStep++; break; case 4:

            RecvSum = 0x68 + dat;
            RecvCount = dat - 6;
            RecvIndex = 0;
            UartRecvStep+
            +; break; case 5:

            RecvSum += dat;
            RxBuffer[RecvIndex++] = dat;
            if (RecvIndex == RecvCount) UartRecvStep++;

        }
    }
}
```

```

        break;
    case 6:
        if (dat != HIBYTE(RecvSum)) goto L_CheckFirst;
        UartRecvStep++; break; case 7:

            if (dat != LOBYTE(RecvSum)) goto L_CheckFirst;
            UartRecvStep++; break; case 8:

                if (dat != 0x16) goto L_CheckFirst;
                UartReceived = TRUE;
                UartRecvStep++; break;

L_CheckFirst:
    case 0:
    default:
        CommInit();
        UartRecvStep = (dat == 0x46 ? 1 : 0);
        break;
    }
}
}

// system initialization
void Initial(void)
{
    UartBusy = FALSE;

    SCON = 0xd0; // Serial data mode must be 8 bit data +1 bit even test
    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 program
BYTE UartSend(BYTE dat)
{
    while (UartBusy);

    UartBusy = TRUE;
}

```

```

acc = dat;
TB8 = P;
SBUF = ACC;

return dat;
}

// Serial communication initialization
void CommInit(void)
{
    UartRecvStep = 0;
    TimeOut = 20;
    UartReceived = FALSE;
}

// Send serial communication packets
void CommSend(BYTE size)
{
    WORD sum;
    BYTE i;

    UartSend(0x46);
    UartSend(0xb9);
    UartSend(0x6a);
    UartSend(0x00);
    sum = size + 6 + 0x6a;
    UartSend(size + 6);
    for (i=0; i<size; i++)
    {
        sum += UartSend(TxBuffer[i]);
    }
    UartSend(HIBYTE(sum));
    UartSend(LOBYTE(sum));
    UartSend(0x16);
    while (UartBusy);

    CommInit();
}

//right STC32G series of chips ISP Downloader
BOOL Download(BYTE *pdat, long size)
{
    BYTE offset;
    BYTE cnt;
    DWORD addr; // overtake 64K code space, the address needs to be defined as 4 byte

    // shake hands
    CommInit();
    while (1)
    {
        if (UartRecvStep == 0)
        {
            UartSend(0x7f);
            DelayXms(10);
        }
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x50) break;
            return FALSE;
        }
    }
}

```

```

        }

}

// Set the parameter to set the slave chip to use the highest baud rate
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;
    }
}

// Prepare
TH1 = HIBYTE(BR(MAXBAUD));
TL1 = LOBYTE(BR(MAXBAUD));
DelayXms(10);
TxBUFFER[0] = 0x05;
TxBUFFER[1] = 0x00;
TxBUFFER[2] = 0x00;
TxBUFFER[3] = 0x5a;
TxBUFFER[4] = 0xa5;
CommSend(5);
while (1) {

    if (TimeOut == 0) return FALSE;
    if (UartReceived) {

        if (RxBuffer[0] == 0x05) break;
        return FALSE;
    }
}

// erase
DelayXms(10);
TxBUFFER[0] = 0x03;
TxBUFFER[1] = 0x00;
TxBUFFER[2] = 0x00;
TxBUFFER[3] = 0x5a;
TxBUFFER[4] = 0xa5;
CommSend(5);
TimeOut = 100;
while (1) {

    if (TimeOut == 0) return FALSE;
    if (UartReceived) {

        if (RxBuffer[0] == 0x03) break;
        return FALSE;
    }
}

```

```

        }

}

//write user code
DelayXms(10);
addr = 0;
TxBuffer[0] = 0x22;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
offset = 5;
while (addr < size)
{
    if (addr < 0x10000)           // The target address information of the program code needs to be changed from the original HEX obtained from the file
    {
        TxBuffer[0] |= 0x10;      // The target programming address is FE:0000~FE:FFFF
    }
    else
    {
        TxBuffer[0] &= ~0x10; // The 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;
}

// Download completed
return TRUE;
}

char code DEMO[256]
= {
    0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
    0xD9,0xFC,0x22,
};

```

## Appendix D Serial Interrupt Transceiver-MODBUS Protocol

**C** language code

---

//The test frequency is **11.0592MHz**

```
##include "stc8h.h"
#include "stc32g.h" // For header files, see Download Software
```

```
#define MAIN_Fosc 11059200L // define the master clock
/***** Function Description *****/
```

Please don't modify the program first, download it directly "**08**-serial port **1** interrupt sending and receiving **C** language **MODBUS** in the agreement "UART1.hex" Test frequency selection **11.0592MHz**. and test it normally, then modify and transplant .

serial port **1** according to **MODBUS-RTU** Protocol communication This example is a slave program. The host is generally a computer.

This routine only supports multi-register read and multi-register write. The register length is **64** individual command users can press **MODBUS-RTU** Agreement added by itself plus.

This example data uses big endian mode (with **C51** consistent), **CRC16** Use little endian mode (with **PC** consistent).

The default :  
parameter **1** port settings are all **1** bit start bit **8** bit data bit **1** Bit stop bit without parity .  
serial ports **1(P3.0 P3.1): 9600bps**.

When the **0** Used for timeout timing, the serial port will reset the timeout count every time it receives a byte. When the serial port is idle for more than timer is .  
received and the user modifies the baud rate, pay attention to modifying the timeout period

This routine is just an application example. This **MODBUS-RTU** The agreement is not within the scope of responsibility of this example. Users can search the Internet for related agreement text references. .  
example defines a register address range write. **0x1000~0x103f.**

```
4 (8 bytes ):  
10 10 1000 0004 08 1234 5678 90AB CDEF 4930  
return:  
10 10 10 00 00 04 4B C6  
read #register :  
10 03 1000 0004 4388  
return:  
10 03 08 12 34 56 78 90 AB CD EF 3D D5
```

Command error return information :  
**0x90:** Custom function code unsupported function code received  
**0x91:** Command length error Write or read number of  
**0x92:** registers or bytes error Register address error  
**0x93:**

Pay attention to the broadcast address received **0x00** to process information but not return a response .

---

**typedef unsigned char u8;**

```

typedef unsigned int u16;
typedef unsigned long u32;

/****** Local constants *****/
#define RX1_Length 128 /* #define declare receive buffer */
#define TX1_Length 128 /* length send buffer length */

/****** local variable declaration *****/
u8xdataRX1_Buffer[RX1_Length]; // receive buffer
u8xdataTX1_Buffer[TX1_Length]; // send buffer

u8RX1_cnt; //Received byte count
u8TX1_cnt; //send byte count
u8TX1_number; //number of bytes to send
u8RX1_TimeOut; // receive timeout timer

bitB_RX1_OK; // receive data flag
bitB_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 usage timer2, Use other values Timer1 Do
baud rate io=0: serial port 1 switch to P3.0 P3.1, =1: switch to P3.6 P3.7, =2: switch to P1.6 P1.7, =3: switch to P4.3 P4.4.
u8 Timer0_Config(u8 t, u32 reload); // t=0: reload
                                         // The value is the number of master clock cycles, t=1: reload value is a time unit us), return 0 correct, return
                                         // back 1 Load value too large error
u16 MODBUS_CRC16(u8 *p, u8 n);
u8 MODBUS_RTU(void);

#define SL_ADDR 0x10 /* This slave station number address */
#define REG_ADDRESS 0x1000 /* Register first address */
#define REG_LENGTH 64 /* register length */
u16 xdata modbus_reg[REG_LENGTH]; /* register address */

//=====
// function void main(void)
// describe the main function
// parameter none.
// return none.
// Version VER1.0
// date : 2018-4-2
// Remark
//=====

void main(void)
{

```

```




EAXFR = 1; //Enable XFR
CKCON = 0x00; //access to set external data bus speed to fastest
WTST = 0x00; //Set the program code to wait for parameters,
//assign to CPU The speed of executing the program is set to the fastest

Timer0_Config(0, MAIN_Fosc / 10000); // t=0: reload The value is the number of master clock cycles ( Interrupt frequency 20000 sub-second)
UART1_config(9600UL, 1, 0); // brt: Communication baud rate timer2: Baud rate usage timer2, Use other values Timer1 do baud rate io=0:
serial port 1 switch to P3.0 P3.1, =1: switch to P3.6 P3.7, =2: switch to P1.6 P1.7, =3: switch to P4.3 P4.4.

EA = 1;

while (1)
{
    if(B_RX1_OK && !B_TX1_Busy)// received data conduct MODBUS-RTU Protocol analysis
    {
        if(MODBUS_CRC16(RX1_Buffer, RX1_cnt) == 0)// Judge first CRC16 If it is correct, or not, ignore it, do not process it or return it
        information
        {
            if((RX1_Buffer[0] == 0x00) || (RX1_Buffer[0] == SL_ADDR))// Then judge whether the station number address is correct or whether broadcast
            address (does not return information)
            {
                if(RX1_cnt > 2) RX1_cnt -= 2; // remove CRC16 check byte
                i = MODBUS_RTU(); // MODBUS-RTU Protocol analysis
                if(i != 0) // error handling
                {
                    TX1_Buffer[0] = SL_ADDR; // Station address
                    TX1_Buffer[1] = i; //error code
                    crc = MODBUS_CRC16(TX1_Buffer, 2);
                    TX1_Buffer[2] = (u8)(crc>>8); // CRC is little endian
                    TX1_Buffer[3] = (u8)crc;
                    B_TX1_Busy = 1; //flag send busy
                    TX1_cnt = 0; // send byte count
                    TX1_number = 4; //number of bytes to send
                    TI = 1; //start sending
                }
            }
        }
        RX1_cnt = 0;
        B_RX1_OK = 0;
    }
}
}

```

```

***** MODBUS_CRC (shift) ***** past test 06-11-27 *****
calculate CRC, calling method MODBUS_CRC16(&CRC,8); &CRC headed address, 8 is the number of bytes
CRC-16 for MODBUS
CRC16=X16+X15+X2+1
TEST: ---> ABCDEFGHIJ CRC16=0xBEE 1627T
*/
//=====
// function u16 MODBUS_CRC16(u8 *p, u8 n)
// describe computation CRC16 function
// parameter *p: data pointer to compute
//           n: number of bytes to count
// return CRC16 value
// Version V1.0, 2022-3-18 Liang Gong
//=====

u16 MODBUS_CRC16(u8 *p, u8 n)
{
    u8i;
    u16 crc16;

    crc16 = 0xffff; // Preset 16 bit CRC register is 0xffff (that is, all1)
    do
    {
        crc16 ^= (u16)*p; //put 8 bit data with 16 bit CRC XOR the low-order bits of the register, placing the result in CRC register
        for(i=0; i<8; i++) //8 bit data
        {
            if(crc16 & 1) crc16 = (crc16 >> 1) ^ 0xA001; // If the least significant bit is 1, put CRC The contents of the register are shifted to the right by one bit toward the lower bit, using 0 fill
            fill the highest position,
            //XOR polynomial 0xA001
            else crc16 >>= 1; //If the least significant bit is 0, put CRC The contents of the register are shifted to the right by one bit toward the lower bit, using 0 fill the highest
        }
        p++;
    }  

    }while(--n != 0);
    return (crc16);
}

/*
***** modbus protocol *****/
***** *****

```

write multi-registers Address function code register address register number write byte number write data **CRC16**  
offset: 0 1 2 3 4 5 6 7~ at last 2 byte  
byte : 1 byte 1 byte 2 byte 2 byte 1 byte 2\*nbyte 2byte  
addr 0x10 xxxx xxxx xx xx....xx xxxx

return data Address function code register number of address registers **CRC16**  
offset: 0 1 2 3 4 5 6 7

byte : 1 byte 1 byte 2 byte 2 byte                          2 bytes  
addr      0x10 xxxx xxxx                          xxxx

Read multiple registers (address)    Number of function code register address registers    **CRC16**

offset:      0    1 2 3                  4 5                  6 7

byte :      1 byte 1 byte 2 byte 2 byte                          2 bytes  
addr 0x03 xxxx xxxx xxxx

return data (address)    Function Code Read Bytes Read Data    **CRC16**

offset:      0    1 2                  3~    last byte

byte : 1 byte 1 byte                  1 byte                  2\*n bytes 2 bytes  
addr      0x03      xx                  xx....XX XXXX

return error code (data)    error code    **CRC16**

offset:      0    1    at last 2 byte

byte : 1 byte 1 byte 2 byte  
addr      0x03 xxxx

---

**u8MODBUS\_RTU(void)**

{

8i,j,k;

16 reg\_addr;// register address

8reg\_len;// Number of write registers

16crc;

  \*\*\*\*\*  
if(RX1\_Buffer[1] == 0x10)//    write multiple registers

  {

    if(RX1\_cnt < 9)    return 0x91; //    wrong command length

    if((RX1\_Buffer[4] != 0) || ((RX1\_Buffer[5] \* 2) != RX1\_Buffer[6]))return 0x92; //    Wrong number of registers written and number of bytes

    if((RX1\_Buffer[5]==0) || (RX1\_Buffer[5] > REG\_LENGTH))return 0x92; //    Wrong number of registers written

    reg\_addr = ((u16)RX1\_Buffer[2] << 8) + RX1\_Buffer[3]; //    register address

    reg\_len = RX1\_Buffer[5]; //    Number of write registers

    if((reg\_addr+(u16)RX1\_Buffer[5]) > (REG\_ADDRESS+REG\_LENGTH)) return 0x93; //    wrong register address

    if(reg\_addr< REG\_ADDRESS) return 0x93; //    wrong register address

    if((reg\_len\*2+7) != RX1\_cnt) return 0x91; //    wrong command length

    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]; //    data input,    big endian

      k += 2;

```

}

if(RX1_Buffer[0] != 0) // Answers to non-broadcast addresses
{
    for(i=0; i<6; i++) TX1_Buffer[i] = RX1_Buffer[i]; // reply to be returned
    crc = MODBUS_CRC16(TX1_Buffer, 6);
    TX1_Buffer[6]=(u8)(crc>>8); //CRC is little endian
    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; //start sending
}
}

else if(RX1_Buffer[1] == 0x03) // read multiple registers
{
    if(RX1_Buffer[0] != 0) // Answers to non-broadcast addresses
    {
        if(RX1_cnt != 6) return 0x91; // wrong command length
        if(RX1_Buffer[4] != 0) return 0x92; // The number of read registers is wrong
        if((RX1_Buffer[5]==0) || (RX1_Buffer[5] > REG_LENGTH))return 0x92; // The number of read registers is wrong

        reg_addr = ((u16)RX1_Buffer[2] << 8) + RX1_Buffer[3]; // register address
        reg_len = RX1_Buffer[5]; // Read the number of registers
        if((reg_addr+(u16)RX1_Buffer[5]) > (REG_ADDRESS+REG_LENGTH)) return 0x93; // wrong register address
        if(reg_addr< REG_ADDRESS) return 0x93; // wrong register address

        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; // returns the number of bytes

        for(k=3, i=0; i<reg_len; i++,j++)
        {
            TX1_Buffer[k++] = (u8)(modbus_reg[j] >> 8); // Data is in big endian mode
            TX1_Buffer[k++] = (u8)modbus_reg[j];
        }
        crc = MODBUS_CRC16(TX1_Buffer, k);
        TX1_Buffer[k++] = (u8)(crc>>8); //CRC is little endian
        TX1_Buffer[k++] = (u8)crc;
        B_TX1_Busy = 1; //flag send busy
        TX1_cnt = 0; // send byte count
        TX1_number = k; //number of bytes to send
        TI = 1; //start sending
    }
}

```

```

else return 0x90; // Function code error

return 0; // Parse correctly
}

//=====
// function u8 Timer0_Config(u8 t, u32 reload)
// describe timer0 initialization function
// parameter t: Reload value type 0 time, and the reload is the number of system clock. The rest of the values indicate that the reload is the (us).
//      reload: reload value
// return 0: Initialized correctly 1: Reload value too large initialization error +
// Version V1.0, 2018-3-5
//=====

u8Timer0_Config(u8 t, u32 reload) //t=0: reload
{
    TR0 = 0; // stop counting

    if(t != 0) reload = (u32)((float)MAIN_Fosc * (float)reload)/1000000UL); // The value is the number of master clock cycles, t=1: reload unit value is timer0 us
    number

    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; // enable interrupt
    TMOD &=0xf0;
    TMOD |= 0; // Operating mode 0:16 Bit auto-reload 1:16 Bit timing count, 2: 8 Bit auto-reload 3:16 Bit auto-reload non-maskable interrupt
    TR0 = 1; // start operation
    return 0;
}

//=====
// function void timer0_ISR (void) interrupt TIMER0_VECTOR
// describe timer0 interrupt function
// parameter none.
// return none.
// Version V1.0, 2016-5-12
//=====

void timer0_ISR (void) interrupt 1

```

```

{

if(RX1_TimeOut != 0)

{

    if(--RX1_TimeOut == 0) // time out

    {

        if(RX1_cnt != 0)// receive data

        {

            B_RX1_OK = 1; // Indicates that the data block has been received

        }

    }

}

}

//=====

// function SetTimer2Baudrate(u16 dat)

// Description settings Timer2 Do the baud rate generator.

// parameter dat: Timer2 reload value of

// return none.

// Version VER1.0

// date : 2018-4-2

// Remark

//=====

void SetTimer2Baudrate(u16 dat) // select baud rate 2: use Timer2 Do other values of baud rate use Timer1 do baud rate

{

    AUXR &= ~(1<<4); // Timer stop

    AUXR &= ~(1<<3); // Timer2 set As Timer

    AUXR |= (1<<2); // 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)

// describe UART1 initialization function.

// parameter brt: Communication baud rate

// timer: Timer used for baud rate timer=2: Baud rate usage timer2, Use other values Timer1 do baud rate

// io: serial port 1 switched to IO, io=0: serial port 1 switch to P3.0P3.1, =1: switch to P3.6 P3.7, =2: switch to P1.6 P1.7, =3: cut

change to P4.3 P4.4.

// return none.

// Version VER1.0

// date : 2018-4-2

// Remark

```

```

//=====

void UART1_config(u32 brt, u8 timer, u8 io) // brt: Communication baud rate, timer=2: Baud rate usage timer2, Use other values Timer1 Do
baud rate io=0: serial port 1 switch to P3.0 P3.1, =1: switch to P3.6 P3.7, =2: switch to P1.6 P1.7, =3: switch to P4.3 P4.4.

{
    brt = 65536UL - (MAIN_Fosc / 4) / brt;
    if(timer == 2) // Baud rate usage timer2
    {
        AUXR |= 0x01; // S1 BRT Use Timer2;
        SetTimer2Baudrate((u16)brt);
    }

    else // Baud rate usage timer1
    {
        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; // prohibit Timer1 interrupt
        TR1 = 1; // run Timer1
    }

    P_SW1 &= ~0xc0; // switch by default 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 bit start bit 1 Bit stop bit without parity
 $// PS = 1; //$  high priority interrupt
 $ES = 1; //$  enable interrupt
 $REN = 1; //$  Allow to receive
}

```

---



---

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

```
// function void UART1_ISR (void) interrupt UART1_VECTOR
```

```
// describe serial port 1 interrupt function
```

```
// parameter none.
```

```
// return none.
```

```
// Version VER1.0
```

```
// date : 2018-4-2
```

```
// Remark
```

---



---

```
void UART1_ISR (void) interrupt 4
```

```
{
```

```
if(RI)
```

```
{
```

```
RI = 0;
```

```
if(!B_RX1_OK) // Receive buffer is free
```

```
{
```

```
if(RX1_cnt >= RX1_Length) RX1_cnt = 0;
```

```
RX1_Bufier[RX1_cnt++] = SBUF;
```

```
RX1_TimeOut = 36; // receive timeout timer 35 single digit time
```

```
}
```

```
}
```

```
if(TI)
```

```
{
```

```
TI = 0;
```

```
if(TX1_number != 0) // have data to send
```

```
{
```

```
SBUF = TX1_Buffer[TX1_cnt++];
```

```
TX1_number--;
```

```
}
```

```
else B_TX1_Busy = 0;
```

```
}
```

```
}
```

## Appendix E on whether to bake before reflow

According to the requirements of the International Moisture Sensitivity Level 3 (MSL3) specification, the SMD components will be released within 168 hours after unpacking the vacuum package.

Within 7 days, the reflow soldering patch must be completed, if not, it must be baked again at high temperature.

SOP/TSSOP plastic tubes cannot withstand high temperatures above 100 degrees, and must be reflow soldered within 7 days after unpacking.

Otherwise, before reflow soldering, remove the plastic tube that cannot withstand high temperature above 100 degrees, put it in a metal tray, and re-bake: 110°C~125°C, 4~8 hours is fine

LQFP/QFN/DFN trays can withstand high temperatures above 100 degrees, and must be reflow soldered within 7 days after unpacking.

Otherwise, it must be re-baked before reflow soldering: 110°C~125°C, 4~8 hours.

## Appendix F How to use a multimeter to check the quality of the chip I/O port

According to the requirements of the International Moisture Sensitivity Level 3 (MSL3) specification, the SMD components must be reflowed within 168 hours and within 7 days after the vacuum packaging is unpacked. If not, they must be baked again at high temperature. If there is no high-temperature baking process and reflow soldering is performed directly, the metal wires inside the chip may be pulled off due to uneven heating inside and outside the chip, and the final phenomenon is that the chip I/O port is damaged.

In the chip design of STC's single-chip microcomputer, each I/O port has two protection diodes respectively connected to VCC and GND, which can be measured with the diode monitoring file of the multimeter. You can use this method to simply judge the quality of the I/O pins. The measurement method using a multimeter is as follows  
(Note: a digital multimeter is used here)

First, adjust the multimeter to the diode detection position, do not supply power to the chip under test, connect the red test lead of the multimeter to the **GND** pin of the tested chip, and **measure each I/O port in turn with the black test lead**. If the parameter displayed by the multimeter is 0.7V Left and right, it means that the protection diode from the internal I/O of the chip to GND is normal, that is, the wiring is also intact. If the displayed parameter is 0V, it means that the wiring inside the chip has been pulled off.

The above method is a method to detect the wire bonding inside the chip.

In addition, if there is no protection circuit for the pins of the microcontroller on the user board, once overcurrent or overvoltage occurs, the I/O may be burned out. In order to detect whether the pin is 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 detect the protection diode from I/O port to VCC as follows:

First, adjust the multimeter to the diode detection gear, do not supply power to the chip under test, connect the black test lead of the multimeter to the **VCC** pin of the tested chip, and **measure each I/O port in turn with the red test lead**. If the parameter displayed by the multimeter is 0.7V Left and right, it means that the protection diode from the internal I/O of the chip to VCC is normal. If the displayed parameter is 0V, it means that the port of the chip has been damaged.

## Appendix G mass production, how to save the special programmer member, how to not burn the link

For mass production, before you assemble the control board with STC MCU as the main control chip into the device, after you patch the STC MCU to your control board, you must test the quality of your control board. Don't say 100%, there is no problem with pass-through, then

It is to lift the bar, not to engage in production. As long as it is produced, it will be soldered and short-circuited, some originals will be wrongly pasted, and some originals will be purchased incorrectly.

So after the patch comes back, before assembling it into the housing, you must test whether your control board with STC MCU is good or not,

The good ones are assembled, the bad ones are repaired and rescued.

For testing, mass production, there must be a test stand/connect our offline programming tool U8W/U8W-Mini/STC-USB below

Link1, but also connected to other control parts

Through USER-VCC, P3.0, P3.1, GND connection, workers are required to turn on the power every time

Connect through S-VCC, P3.0, P3.1, GND, don't turn on the power, STC's offline tool will automatically supply power to you

The cost of making a test stand for you outside is less than 500 yuan, that is, plexiglass, fixtures, and thimbles.

1 worker management to test if your control board is working 2 - 3 test racks

Operating procedures:

1. Insert your STC MCU control board to the test rack 1

2. Insert your STC MCU control board to the test rack 2, the program on the test rack 1 has been burned/can't feel burning recording time

3. Test whether the STC main control board on the test rack 1 is functioning normally. It is normally placed in the normal area, and if it is not normal, it is not placed normally.

Area

4. Card the new untested unprogrammed control board to test rack 1

5. The untested control board/program on the test rack 2 has been burnt without knowing when, and the untested untested is replaced with a new one.

Burned control panel

6. Cycle steps 3 to 5

===== No need to arrange programming personnel

## Appendix H Notes on 0xFD Issues in Keil Software

It is well known that all versions of Keil software's 8051 and 80251 compilers have a problem called 0xFD, mainly in the word

The character string cannot contain Chinese characters encoded with 0xFD, otherwise Keil software will skip 0xFD and cause garbled characters when compiling.

Regarding this issue, Keil's official response is: 0xfd, 0xfe, 0xff These three character codes are used internally by the Keil compiler, so

If the code contains a string of 0xfd, 0xfd will be automatically skipped by the compiler.

The solution officially provided by Keil: add a 0xfd after the Chinese character with 0xfd encoding. E.g:

```
printf("Math"); //Keil will display garbled characters after compiling
```

```
printf("Number\xfd study"); //Display is normal
```

Here "\xfd" is an escape character in standard C code, and "\x" means that the following 1~2 characters are hexadecimal numbers. "\xfd"

Indicates that the hexadecimal number 0xfd is inserted into the string.

Since the Chinese character encoding of "Number" is 0xCAFD, Keil will skip FD when compiling, and only compile CA into the target file, and then

By manually adding another 0xfd to the target file through the escape character, a complete 0xCAFD is formed, which can be displayed normally.

There are many patches on the Internet about 0xFD, which are basically only valid for the old version of Keil software. The method of patching is in the executable file

Find the key code [80 FB FD] in [80 FB FD] and modify it to [80 FB FF]. The key code found by this modification method is too simple and easy to modify.

To other unrelated places, resulting in inexplicable problems when the compiled object file is running. So, the strings in the code have packages

When it contains the following Chinese characters, it is recommended to use the official solution provided by Keil to solve the problem

In GB2312, the Chinese characters with 0xfd encoding are as follows:

Complimentary cakes are removed from waiting for spies, Er, prisoners, and Geng to accumulate arrows and embers

The case is slow, condensed, and condensed.

Spinning Demon Introduces Zhazheng Casting

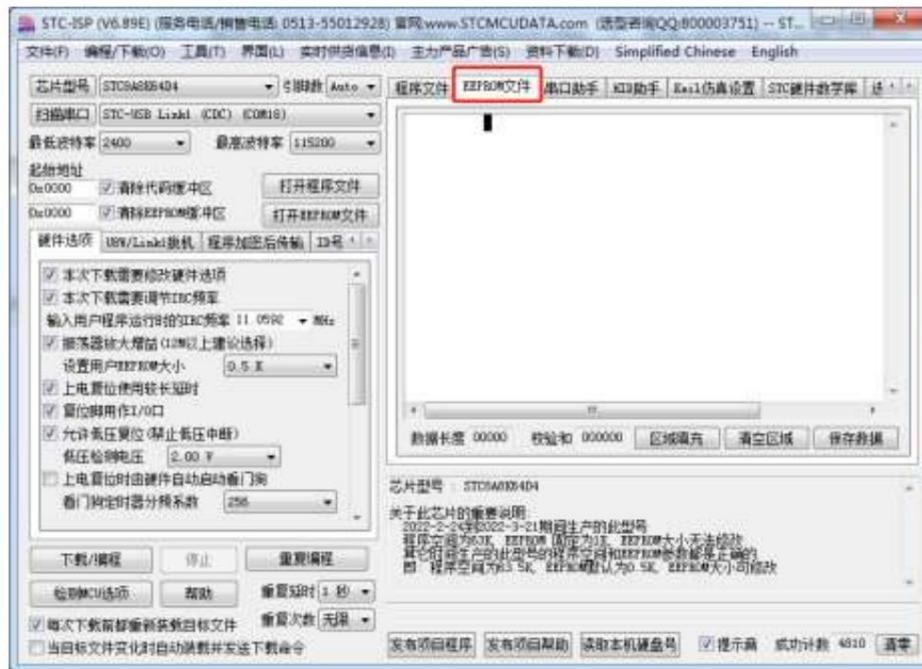
ÿkai ÿÿÿÿÿSao

snoring

In addition, the characters of the Keil project path name cannot contain Chinese characters with 0xFD encoding, otherwise the Keil software will not be able to compile this program correctly. project.

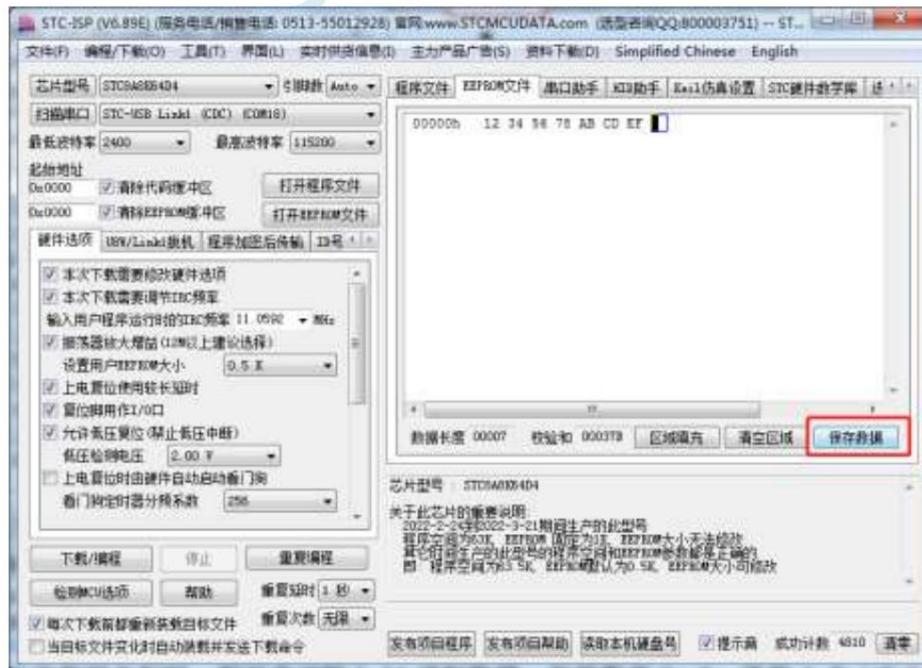
# Appendix I How to use STC-ISP download software to make and edit EEPROM file

Open any version of STC-ISP download software, select the "EEPROM" page, click the data window, as shown below



When the black rectangle cursor appears, you can manually input hexadecimal data, including numbers 0~9, letters A~F (common case)

After the data input is completed, click the "Save Data" button to save the EEPROM data



## Appendix J STC32 series header file definition

```
#ifndef __STC32G_H_
#define __STC32G_H_

///////////////////////////////
#include <intrins.h>

//After including this header file, there is no need to include "REG51.H" in addition

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 sbit = ADC_CONTR^7;
ADC_START = sbit ADC_CONTR^6;
ADC_FLAG = sbit ADC_CONTR^5;
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 = sbit WDT_CONTR^7;
EN_WDT = sbit CLR_WDT WDT_CONTR^5;
= 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	DPL1	=	0xe4;
sfr	DPH1	=	0xe5;
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 and write 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 #define MS_START_SENDDAT_RECVACK #define MS_SENDDAT_RECVACK #define MS_RECVDAT_SENDACK #define MS_RECVDAT_SENDAK (*(unsigned char volatile far *)0x7efe82) 0x09
#define MSBUSY           0x0a
#define MSB0              0x0b
#define MS_RECVDAT_SENDAK 0x0c
#define I2CMSST          0x0b
#define MSIF              0x0d
#define MSACKI            0x0e
#define MSACKO            0x0f
#define I2CSLCR          (*(unsigned char volatile far *)0x7efe83)
#define ESTAI             0x10
#define ERXI              0x11
#define ETXI              0x12

```

#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 *)0x7feea0)
#define TM1PS	(*(unsigned char volatile far *)0x7feea1)
#define TM2PS	(*(unsigned char volatile far *)0x7feea2)
#define TM3PS	(*(unsigned char volatile far *)0x7feea3)
#define TM4PS	(*(unsigned char volatile far *)0x7feea4)
#define ADCTIM	(*(unsigned char volatile far *)0x7feea8)
#define T3T4PS	(*(unsigned char volatile far *)0x7feac)
#define ADCEXCFG	(*(unsigned char volatile far *)0x7feead)
#define CMPEXCFG	(*(unsigned char volatile far *)0x7faeee)
#define PWMA_ETRPS	(*(unsigned char volatile far *)0x7feb0)
#define PWMA_ENO	(*(unsigned char volatile far *)0x7feb1)
#define PWMA_PS	(*(unsigned char volatile far *)0x7feb2)
#define PWMA_IOAUX	(*(unsigned char volatile far *)0x7feb3)
#define PWMB_ETRPS	(*(unsigned char volatile far *)0x7feb4)
#define PWMB_ENO	(*(unsigned char volatile far *)0x7feb5)
#define PWMB_PS #define	(*(unsigned char volatile far *)0x7feb6)
PWMB_IOAUX #define	(*(unsigned char volatile far *)0x7feb7)
CANAR	(*(unsigned char volatile far *)0x7febb)
#define CANDR	(*(unsigned char volatile far *)0x7febbc)

```

#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_PSRH          (*(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 #define PWMB_CR2 #define PWMB_SMCR #define PWMB_ETR #define PWMB_IER #define PWMB_SR1 #define PWMB_SR2 #define PWMB_EGR #define PWMB_CCMR1 #define PWMB_CCMR2 #define PWMB_CCMR3 #define PWMB_CCMR4 #define PWMB_CCER1 #define PWMB_CCER2

```

```

#define PWMB_CNTRH      (*(unsigned char volatile far *)0x7efeee)
#define PWMB_CNTRL      (*(unsigned char volatile far *)0x7fefef)
#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 *)0x7effff)

///////////////////////////////
//7E:FD00H-7E:FDFFH
/////////////////////////////

```

```

#define PWM2_OISR        (*(unsigned char volatile far *)0x7effff)

#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 P0IMO           (*(unsigned char volatile far *)0x7efd20)
#define P1IMO           (*(unsigned char volatile far *)0x7efd21)
#define P2IMO           (*(unsigned char volatile far *)0x7efd22)
#define P3IMO           (*(unsigned char volatile far *)0x7efd23)
#define P4IMO           (*(unsigned char volatile far *)0x7efd24)
#define P5IMO           (*(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 *)0x7efdcf)
#define USART2BRH        (*(unsigned char volatile far *)0x7efdce)
#define USART2BRL        (*(unsigned char volatile far *)0x7efdcb)

#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 *)0x7efbf0)
#define HSPWMA_ADR        (*(unsigned char volatile far *)0x7efbf1)
#define HSPWMA_DAT        (*(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, first set EAXFR to 1
//Instructions:
//    char val;
//
//    EAXFR = 1;                                //Enable access to XFR
//    READ_HSPWMA(PWMA_CR1, val); val |=        //Asynchronously read the PWMA group register
//    0x01;
//    WRITE_HSPWMA(PWMA_CR1, val);               //Asynchronously write the PWMA group register

#define READ_HSPWMA(reg, dat) {
    \
    \
    while (HSPWMA_ADR & 0x80);
    HSPWMA_ADR = ((char)&(reg)) | 0x80; while
    (HSPWMA_ADR & 0x80); (dat) = HSPWMA_DAT;
    \
}

#define WRITE_HSPWMA(reg, dat) {
    \
    \
    while (HSPWMA_ADR & 0x80);
    HSPWMA_DAT = (dat);
    HSPWMA_ADR = ((char)&(reg)) & 0x7f;
}

#define READ_HSPWMB(reg, dat) {
    \
    while (HSPWMB_ADR & 0x80);
    HSPWMB_ADR = ((char)&(reg)) | 0x80; while
    (HSPWMB_ADR & 0x80); (dat) = HSPWMB_DAT;
    \
}

#define WRITE_HSPWMB(reg, dat) {
    \
    \
    while (HSPWMB_ADR & 0x80);
    HSPWMB_DAT = (dat);
    HSPWMB_ADR = ((char)&(reg)) & 0x7f;
}

```

```
//////////  
//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_ONE	(*(unsigned char volatile far *)0x7efa3b)
#define DMA_UR1R_ONE	(*(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_ASTA       (*(unsigned char volatile far *)0x7efaa2)
#define DMA_I2CR_A          (*(unsigned char volatile far *)0x7efaa3)

```

```

#define DMA_I2CR_DONE #define          (*(unsigned char volatile far *)0x7efaa4)
DMA_I2CR_RXAH #define          (*(unsigned char volatile far *)0x7efaa5)
DMA_I2CR_RXAL          (*(unsigned char volatile far *)0x7efaa6)

#define DMA_I2CT_AMTH #define          (*(unsigned char volatile far *)0x7efaa8)
DMA_I2CT_DONEH #define          (*(unsigned char volatile far *)0x7efaa9)
DMA_I2CR_AMTH #define          (*(unsigned char volatile far *)0x7efaaa)
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 *)0x7efaaaf)

```

//////////

```

//sfr CANICR =          0xf1;
##define CANAR          (*(unsigned char volatile far *)0x7efebb)
##define CANDR          (*(unsigned char volatile far *)0x7efebc)

```

//To use the following macro, first set EAXFR to 1

//Instructions:

```

//      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 Register

//////////

```
//sfr    LINICR      =      0xf9;
//sfr    LINAR       =      0xfa;
//sfr    LINDR       =      0xfb;
```

//Instructions:

```
//      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 LIDs                 0x02
#define LER                  0x03
#define LIE                  0x04
#define LSR                  0x05
#define LCR                  0x05
#define DLLs                 0x06
#define DLH                  0x07
#define HDRL                 0x08
#define HDRH                 0x09
#define HDP                  0x0A
```

```

///////////
//USB Control Register
///////////

//sfr USBCLK =           0xdc;
//sfr USBDAT =          0xec;
//sfr USBCON=           0xf4;
//sfr USBADR =          0xfc;

//Instructions:
//      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 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 INCSCR2          (USBBASE+18)
#define INAUTOSET         0x80
#define INISO             0x40
#define INMODEIN          0x20
#define INMODEOUT         0x00
#define INENDDMA          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 OUTENDDMA         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 #define	5	//002BH
LVD_VECTOR //#define	6	//0033H
PCA_VECTOR #define	7	//003BH
UART2_VECTOR #define	8	//0043H
SPI_VECTOR #define	9	//004BH
INT2_VECTOR #define	10	//0053H
INT3_VECTOR #define	11	//005BH
TMR2_VECTOR #define	12	//0063H
USER_VECTOR #define	13	//006BH
BRK_VECTOR #define	14	//0073H
ICEP_VECTOR #define	15	//007BH
INT4_VECTOR #define	16	//0083H
UART3_VECTOR #define	17	//008BH
UART4_VECTOR #define	18	//0093H
TMR3_VECTOR #define	19	//009BH
TMR4_VECTOR #define	20	//00A3H
CMP_VECTOR //#define	twenty one	//00ABH
PWM_VECTOR //#define	twenty two	//00B3H
PWMFD_VECTOR #define	twenty three	//00BBH
I2C_VECTOR #define	twenty four	//00C3H
USB_VECTOR #define	25	//00CBH
PWMA_VECTOR #define	26	//00D3H
PWMB_VECTOR #define	27	//00DBH
CAN1_VECTOR #define	28	//00E3H
CAN2_VECTOR #define	29	//00EBH
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 #define		//017BH
DMA_ADC_VECTOR 48 #define		//0183H
DMA_SPI_VECTOR 49 #define		//018BH
DMA_UR1T_VECTOR 50 #define		//0193H
DMA_UR1R_VECTOR 51 #define		//019BH
DMA_UR2T_VECTOR 52 #define		//01A3H
DMA_UR2R_VECTOR 53 #define		//01ABH
DMA_UR3T_VECTOR 54 #define		//01B3H
DMA_UR3R_VECTOR 55		//01BBH

```

#define DMA_UR4T_VECTOR 56 #define //01C3H
DMA_UR4R_VECTOR 57 #define //01CBH
DMA_LCM_VECTOR 58 #define //01D3H
LCM_VECTOR #define DMA_I2CT_VECTOR 59 //01DBH
#define DMA_I2CR_VECTOR #define 60 //01E3H
I2S_VECTOR #define DMA_I2ST_VECTOR 61 //01EBH
#define DMA_I2SR_VECTOR 62 //01F3H
63 //01FBH
64 //0203H

///////////////////////
#define EAXSFR() EAXFR = 1 /* MOVX A,@DPTR/MOVX @DPTR,A instruction
                           The operation object is extended SFR(XSFR) */
#define EAXRAM() EAXFR = 0 /* MOVX A,@DPTR/MOVX @DPTR,A instruction
                           The operation object 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
```

STCMCU

## Appendix K Electrical Characteristics

Absolute Maximum Ratings

parameter	Min	Max	Unit		illustrate
storage temperature	-55	+155	°C		
Operating temperature	-40	+85	°C	Use internal high-speed IRC (36M) and external crystal oscillator	
	-40	+125	°C	When the temperature is higher than 85°C, please use an external high temperature crystal oscillator, And the working frequency is recommended to be controlled below 24M	
Working voltage VDD	1.9	5.5	V	When the working temperature is lower than -40°C, the working voltage shall not be lower than 3.0V	
voltage VDD voltage to ground	+5.5		V		
-0.3 I/O port voltage to ground	-0.3	VDD+0.3V			

DC characteristics (VSS=0V, VDD=5.0V, test temperature=25°C)

label	parameter	Range Min Typ Max Units				test environment
		Range	Min	Typ	Max	
IPD	Power-down mode	-	0.6	-	-	uA
IWKT	current Power-down	-	4.4	-	-	uA
ILVD	wake-up timer Low	-	430	-	-	uA
IIDL	voltage detection module Idle	-	0.99	-	-	mA
	mode current (6MHz) Idle mode current	-	1.14	-	-	mA
	(11.0592MHz) Idle mode current	-	1.37	-	-	mA
	(24MHz) Idle mode current (internal)	-	0.57	-	-	mA
INOR	32KHz) Normal mode current	-	1.65	-	-	mA
	(6MHz) Normal mode current	-	2.29	-	-	mA
	(11.0592MHz) ) Normal mode current	-	3.49	-	-	mA
	(24MHz) Normal mode current (internal)	-	0.57	-	-	mA
ICC	32KHz) Normal working mode current	-	4	20 mA		
VIL1	input low level	-	-	1.32 V	on Schmitt trigger	
		-	-	1.48 V	to turn off Schmitt trigger	
VIH1	input high level (normal I/O)	1.60	-	-	V turns on Schmitt trigger	
		1.54	-	-	V turns off Schmitt trigger	
VIH2	input high level (reset pin) output	1.60	-	1.32V		
IOL1	low level sink 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) - logic 0	20	-	-	mA	port voltage 2.4V
IIL	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	4.1	4.2	4.4 kΩ		

I/O speed	I/O high current drive, I/O fast conversion	36		MHz	PxDR=0, PxSR=0
	I/O small current drive, I/O fast conversion	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 conversion	20		MHz	PxDR=1, PxSR=1
Compare device	fastest speed analog filter	10		MHz off	all analog and digital filtering
	time	0.1		us	
	digital filter time	0		system	LCDTY=0
		n+2		clock	LCDTY=n (n=1~63)
Power-down mode power consumption with comparator enabled on IPD2-		460	-	uA	
Power-down mode power consumption with LVD enabled by IPD3-		520	-	uA	

## DC characteristics (VSS=0V, VDD=3.3V, test temperature=25°C)

label	parameter					test environment
		Range	Min	Typ	Max	
IPD	Power-down mode	-	0.4	-	-	uA
IWKT	current Power-down	-	1.5	-	-	uA
ILVD	wake-up timer Low	-	364	-	-	uA
IIDL	voltage detection module Idle	-	0.89	-	-	mA
	mode current (6MHz) Idle mode current	-	1.05	-	-	mA
	(11.0592MHz) Idle mode current	-	1.28	-	-	mA
	(24MHz) Idle mode current (internal)	-	0.47	-	-	mA
INOR	32KHz) Normal mode current	-	1.55	-	-	mA
	(6MHz) Normal mode current	-	2.19	-	-	mA
	(11.0592MHz) Normal mode current	-	3.38	-	-	mA
	(24MHz) Normal mode current (internal)	-	0.47	-	-	mA
ICC	32KHz) Normal working mode current	-	4	20	mA	
VIL1	input low level	-	-	0.99	V to turn on Schmitt trigger	
		-	-	1.07	V to turn off Schmitt trigger	
VIH1	input high level (normal I/O)	1.18	-	-	V turns on Schmitt trigger	
		1.09	-	-	V turns off Schmitt trigger	
VIH2	input high level (reset pin) output	1.18	-	0.99	V	
IOL1	low level sink 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) - logic 0	20	-	-	mA port voltage 2.4V	
IIL	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 speed	I/O high current drive, I/O fast conversion		25		MHz	PxDR=0, PxSR=0
	I/O small current drive, I/O fast conversion		twenty two		MHz	PxDR=1, PxSR=0
	I/O high current drive, I/O slow conversion		16		MHz	PxDR=0, PxSR=1
	I/O small current drive, I/O slow conversion		12		MHz	PxDR=1, PxSR=1
Compare device	fastest speed analog filter		10		MHz off	all analog and digital filtering
	time		0.1		us	
	digital filter time		0		system clock	LCDTY=0
			n+2			LCDTY=n (n=1~63)
	Power-down mode power consumption with comparator enabled on IPD2-	400	-	uA		
	Power-down mode power consumption with LVD enabled by IPD3-	470	-	uA		

Internal IRC temperature drift characteristics (reference temperature 25°C)

temperature	scope		
	minimum	Typical value	maximum value
-40°C~85°C		-1.38%~+1.42%	
-20°C~65°C		-0.88%~+1.05%	

Low voltage reset threshold voltage (test temperature 25°C)

level	Voltage		
	minimum	Typical value	maximum value
POR		1.9V	
LVR0		2.0V	
LVR1		2.4V	
LVR2		2.7V	
LVR3		3.0V	

## Appendix L Update Record

ÿ 2022/8/4

1. Corrected some typos in the DMA chapter 2. Corrected the description of some SFRs affected by the LOCK bit in the Advanced PWM chapter

ÿ 2022/7/11

1. Added STC-USB Link1D tool is the instruction manual

ÿ 2022/7/6

1. Correct the description error of ADC\_DMA channel enable register 2.  
Add the precautions for using the SWRSTF register bits in the reset register RSTFLAG

ÿ 2022/6/13

1. Add the description of automatic mass production process  
in the appendix 2. Add the method of using STC-ISP software to make and edit EEPROM files 3.  
Add the product authorization letter 4. Add the header file definition of STC32G in the appendix 5.  
Update the timer for external counting Example program 6. Add the description of PSW1 register  
7. Add the connection diagram of Type-C interface in the USB download reference circuit diagram

ÿ 2022/6/6

1. Added the description of ISP download related hardware options (Chapter 5)
  2. Added the description of the '0xFD' problem in the Keil software in the appendix
  3. Added the principle, example description and precautions to the capture sample program in the Advanced PWM chapter 4.
- The Advanced PWM chapter adds a sample program to capture the period and duty cycle of 4 signals at the same time

ÿ 2022/5/31

1. Corrected the error of the network name in the capture block diagram in the Advanced PWM chapter 2. Added the reference circuit diagram of the USB direct download of the PDIP40 package chip 3. Added the EAXFR register usage instruction subsection in the memory chapter. Circuit diagram, and adjust the order of chapters 5. Add the example of enabling the internal pull-up resistor of the I/O port

## ÿ 2022/5/19

1. Corrected the typo in the comparator chapter 2.

Added the description of the precautions for using the I/O port in the I/O port chapter 3.

Corrected the error that the S1BRT register is not set in the example of serial port 1 using timer 1 as the baud rate generator 4. Updated the description of the endpoint packet size register in the USB chapter 5. Added the description of the CKCON register in the memory chapter 6. Added the CKCON initialization statement in the sample program 7. Added the STC32G and STC8G/8H interrupt entry address comparison table in the interrupt chapter

## ÿ 2022/5/5

8. Correct the error of the interrupt number in the example program in the Advanced

PWM chapter. 9. Add the method of identifying the first pin and the chip version by the chip silk screen under the pin diagram.

10. Add the description of the P3.2 port when using USB to download directly. 11. Added the description of how to install Keil's C51, C251 and MDK at the same time

## ÿ 2022/4/22

1. Updated the serial port baud rate calculation

instructions 2. Added the chapter "How to use a multimeter to detect the quality of the chip I/O port" in the appendix 3. Added a chapter that introduces the use of third-party extended interrupt number tools

## ÿ 2022/4/18

1. Add the description of CANAR and CANDR register 2. Add the description of

LINAR and LINDR register 3. Add the description of bit addressable area and the definition and usage example of bit variable in Memory chapter (9.2.8)

## ÿ 2022/4/7

1. Corrected the wrong description of the register in the USB chapter (INCSR1) 2.

Corrected the address of the AUXR2 register 3. Added the example program of advanced PWM cycle triggering ADC 4. Added the assembly example program of CAN transceiver

## ÿ 2022/3/30

1. Added the MODBUS protocol sample program in the serial port chapter

2. Added the encoder sample program in the Advanced PWM chapter 3.

Corrected the incorrect translation in the instruction set 4. Updated the usage instructions of the MDU32 library and the FPMU library

## ÿ 2022/3/28

1. Translate the instruction set into Chinese
2. Add the basic chapter of logic algebra in the appendix
3. Correct the error of DMA-related register names

### ÿ 2022/3/17

1. Add the description chapter of assembler programming
2. Add the description of mutual conversion between STC8H series projects and STC32G series projects

### ÿ 2022/3/15

1. Added the description about the working voltage, working frequency and working temperature in the electrical characteristics  
 2. Added the physical diagram and hardware connection diagram of the STC-USB Link1 tool 3.  
 Updated the EEPROM example program 4. Added the use of the internal 1.19V reference voltage in the ADC  
 chapter Example program for signal source inverse working voltage

### ÿ 2022/3/11

1. Added automatic frequency calibration chapter and sample  
 program 2. Added sample program for accessing off-chip extended  
 RAM 3. Added project setting description of "How to reserve EEPROM space when setting projects" 4. Added "Using  
 STC-USB Link1 to STC32G12K128 series 5. Added chapters "Method of creating multi-file project" and "Processing method of interrupt number greater than  
 31" 6. Added the example code of "Using third-party MCU to download STC32G to ISP" in the appendix

### ÿ 2022/3/9

1. Modify the feature description of the  
 document cover 2. Modify the error in the interrupt list in the  
 Interrupt chapter 3. Update the register description in the RTC  
 chapter 4. Add the clock description in the clock tree block diagram  
 5. Correct the problem of wrong register naming in the DMA chapter 64K code  
 project setting 7. Add EEPROM address description, add EEPROM sample  
 program 8. Update BLDC brushless DC motor drive circuit diagram

### ÿ 2022/3/2

1. Moved the content of "Classic Circuit Diagram of DAC Using I/O and R-2R Resistor Divider" to ADC chapter and appendix 2. Added sample program to  
 clock chapter 3. Added high-speed PWM chapter and sample program 4. Added high-speed SPI Chapters and Example Programs

### ÿ 2022/3/1

1. Added "Classic circuit diagram of DAC using I/O and R-2R resistor divider" in I/O chapter

- 
- 2. Update the clock tree in the clock chapter

## ÿ 2022/2/28

- 1. Update the description of the memory chapter
- 2. In all sample programs, the statement "WTST" is added to set the waiting time for CPU to read program code from FLASH  
= 0;", that is, no additional waiting is required to achieve the fastest execution of the code by the CPU.

## ÿ 2022/1/19

- 1. Retype all headings in the document
- 2. Translate the FPMU single-precision floating-point arithmetic chapter
- 3. Update the description of the memory chapter

## ÿ 2022/1/18

- 1. Added RTC real-time clock, LCM color screen, DMA description chapters

## ÿ 2022/1/17

- 1. Complete the first version of the STC32G series MCU technical reference manual document

## Standard sales contract for this series of products

one. Product quality standard: The goods are brand new and authentic. Complies with ROHS quality standards. two.

Responsibility of the supplier: If there is a quality problem of the supplier, after confirmation by both parties, the buyer will return the chip, one replacement, and one year warranty.

three. Responsibilities of the buyer: A. Acceptance: When the express delivery arrives, the buyer confirms that the quantity is correct, no chips are scattered, no pins are deformed,

and there is no other abnormal quality before signing. If there is an abnormality, the buyer cannot sign for the receipt, and the courier company shall be responsible. Once the purchaser signs for receipt, the purchaser recognizes that the supplier has completed the order as required and has no other joint and several liabilities.

B. Storage and SMD processing: According to the requirements of the International Moisture Sensitivity 3 (MSL3) specification, the SMD components must be reflow soldered within 168 hours and 7 days after the vacuum packaging is opened. LQFP/QFN/DFN trays can withstand high temperatures above 100 degrees, and must be reflow soldered within 7 days after unpacking. SOP/TSSOP plastic tubes cannot withstand high temperatures above 100 degrees, and must be reflow soldered within 7 days after unpacking. Otherwise, the plastic tubes that cannot withstand high temperatures above 100 degrees must be removed before reflow soldering, and placed in metal trays. Re-bake: 110~125, 4~8 hours is fine

Because the goods returned by customers often contain products of unknown origin, and after the original SMD components are unpacked in vacuum, the reflow soldering process needs to be completed within 168 hours/7 days. Our company has no capacity to re-test and re-bake the returned devices in detail, and is unable to evaluate the so-called unopened chips returned by customers. In order to ensure the interests of all customers, once the products are out of the warehouse, they will not be returned. Ensure quality and ensure the safety of all customers.

Four. Dispute resolution method: The two parties shall negotiate and resolve any disputes that are not detailed in this contract. If the negotiation fails, apply for arbitration at the supplier's location. five.

Other terms: The contract is in duplicate. It will take effect upon signing by both parties. If the supplier fails to deliver due to external factors, the supplier shall promptly notify the buyer and renegotiate the relevant matters of this contract, and the buyer shall exempt the supplier from its obligations. The clauses not included in this contract can be included in detail in the annex to the contract.

six. This contract will take effect only after it is signed by representatives of both parties and the payment is received.

Remarks: In special circumstances, the model purchased by the buyer should be replaced with other models, and the supplier also agrees:

1. After 13 hours of high-temperature baking, 1,000 yuan per time 2. One time start-up test of RMB500, +0.2 yuan/piece

# Product Power of Attorney

**To: Jiangsu Guoxin Technology Co., Ltd.**

The intellectual property rights of STC32G series products belong to Shenzhen Guoxin Artificial Intelligence Co., Ltd.

all. Jiangsu Guoxin Technology Co., Ltd. is now authorized to engage in STC32G series products in

Promotion and sales work in China.

Authorized unit: 深圳国芯人工智能有限公司

Authorization time limit: October 24, 2019 - December 31, 2024



## Independent property rights, controllable production

Shenzhen Guoxin Artificial Intelligence Co., Ltd. is a wholly-owned enterprise in the mainland of the People's Republic of China, according to Chinese law

An enterprise operating independently by laws and regulations, with its registered address at No. 1A, Qianwan 1st Road, Qianhai Shenzhen-Hong Kong Cooperation Zone, Shenzhen

Building 201.

The devices described in this manual are independently developed in China and have independent intellectual property rights.

The core research and development of the product is in China, with chip design, packaging design, structural design, reliable

All design capabilities such as performance design, device simulation, process simulation, etc.; product core R&D team members and leaders

The leaders are all composed of domestic personnel, and the leader of the R&D team has more than ten years of experience in R&D.

Possess long-term and stable follow-up support capabilities, and have patent certificates and software authorships applied for in my country

rights, etc.

Wafer fabrication: wafer fabrication and processing after the device design is completed, in the mainland of the People's Republic of China

The domestic fab has been processed and manufactured, and is subject to the management, supervision and control of the laws and regulations of the People's Republic of China.

Totally controllable.

Packaging and manufacturing: The packaging and manufacturing after the design of this device is completed, in the mainland of the People's Republic of China

The packaging factory is completed, and it is supervised and controlled by the laws and regulations of the People's Republic of China, and is completely controllable.

Test: After the design of this device is completed, the test is completed in the mainland of the People's Republic of China.

Under the management, supervision and control of the laws and regulations of the People's Republic of China, it is completely controllable.

All key processes of this device are completed on my country's own production lines, which can be supplied for a long time without being compromised.

Suspension troubles.

It is hereby stated.

